Synthesis of Maximally Permissive Strategies for LTL_f Specifications

Shufang Zhu[*] and **Giuseppe De Giacomo**
Sapienza University of Rome, Rome, Italy
{zhu,degiacomo}@diag.uniroma1.it

Abstract

In this paper, we study synthesis of *maximally permissive strategies* for Linear Temporal Logic on finite traces (LTL_f) specifications. That is, instead of computing a single strategy (aka plan, or policy), we aim at computing the entire set of strategies at once and then choosing among them while in execution, without committing to a single one beforehand. Maximally permissive strategies have been introduced and investigated for safety properties, especially in the context of Discrete Event Control Theory. However, the available results for safety properties do not apply to reachability properties (eventually reach a given state of affair) nor to LTL_f properties in general. In this paper, we show that maximally permissive strategies do exist also for reachability and general LTL_f properties, and can in fact be computed with minimal overhead wrt the computation of a single strategy using state-of-the-art tools.

1 Introduction

One of the key problems of Artificial Intelligence is to equip intelligent agents with autonomous capability of deliberating the execution of complex courses of action to accomplish desired tasks [Reiter, 2001; Ghallab *et al.*, 2016]. This problem is related to reactive synthesis in Formal Methods: we have an agent acting in an adversarial environment such that the agent controls certain variables (the agent actions) and the environment controls the others (the environment reactions); given a specification of the task, the agent has to find a strategy (plan/policy/controller/program) to choose its actions to fulfill the task in spite of all possible environment reactions [Pnueli and Rosner, 1989]. Specifically, reactive synthesis shares deep similarities with planning in fully observable nondeterministic domains (FOND, strong plans) [Cimatti *et al.*, 2003; Geffner and Bonet, 2013].

In Formal Methods, the most common formalism for specifying tasks is Linear Temporal Logic (LTL) [Pnueli, 1977]. In AI, a finite trace variant of LTL (LTL_f) is popular [Baier and McIlraith, 2006; De Giacomo and Vardi, 2013; De Giacomo and Vardi, 2015; Zhu *et al.*, 2017; De Giacomo and Rubin, 2018; Camacho *et al.*, 2019]. The interest on finite traces is due to the observation that typically intelligent agents are not dedicated to a single task all their life, but are supposed to accomplish one task after another.

Deliberating courses of action (i.e., strategies) is also related to controller synthesis in Discrete Event Systems (DES) [Ehlers *et al.*, 2017]. In DES we are primarily interested in a variant of the problem: suitably control an agent's behavior to conform to a set of specifications while preserving its autonomy as much as possible. In other words, we want to synthesize a controller that embeds all possible allowed courses of action so that whatever actual course of action the agent chooses, the specifications are satisfied. Such a controller is said to be *maximally permissive* [Wonham and Ramadge, 1987; Cassandras and Lafortune, 2008; Wonham and Cai, 2019]. Maximally permissive controllers are indeed also of interest in AI, as discussed, e.g., in [De Giacomo *et al.*, 2012; De Giacomo *et al.*, 2013; Banihashemi *et al.*, 2016; Banihashemi *et al.*, 2018; De Giacomo *et al.*, 2020].

In this paper, following this idea, instead of computing a classical (aka, deterministic) strategy for fulfilling a task, we aim at computing the *maximally permissive strategy*, i.e., the entire set of strategies that fulfil the task. This allows the agent to be able to choose among them while in execution, without committing to any specific one beforehand.

A classical strategy for a task, while already considers all possible environment reactions, prescribes in any circumstances exactly what action to do until the task is fulfilled. This leaves no freedom to the agent in choosing actions that would not harm fulfilling the original task, but would allow the agent to take additional opportunities, such as additional tasks. The maximally permissive strategy, i.e., the entire set of strategies fulfilling a task, instead, gives the maximal freedom of the agent in taking more opportunities, while guaranteeing that the original task is being accomplished.

To capture the notion of maximally permissive strategy, we rely on nondeterministic strategies. A classical (aka, deterministic) strategy is a function from the history so far to the next action the agent should perform. A *nondeterministic strategy* is a function from the history to *a set of next actions* such that by arbitrarily choosing among them, at each step, we get a deterministic strategy that accomplishes the task. In other words, a nondeterministic strategy represents a set of

[*]Corresponding Author

deterministic strategies.

The foundational result of DES is that for the kind of specification of interest in DES, which are *safety* specifications, there exists a unique *nondeterministic strategy* that is able to represent the maximally permissive strategy, i.e., the entire set of deterministic strategies. Unfortunately, this powerful result applies only to safety properties [Bernet *et al.*, 2002]. In particular, a *nondeterministic strategy* that is able to represent the entire set of deterministic strategies does not exist for reachability properties, or for LTL_f specifications. Yet, the need for computing the maximally permissive strategies exists, and techniques to approximate it have been studied, e.g., in [Sakakibara and Ushio, 2020].

In this paper, we show that maximally permissive strategies for reachability and general LTL_f properties are still representable via nondeterministic strategies. Specifically, the entire set of deterministic strategies to fulfil an LTL_f task can be characterized by *two* (vs. one) nondeterministic strategies and a constraint that requires to eventually switch from the first one to the second one. Interestingly, these two nondeterministic strategies can be computed with minimal overhead wrt the computation of a single deterministic strategy using state-of-the-art tools.

2 Preliminaries

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

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

where $a \in Prop$ is an *atom*, $\circ$ for *Next*, and $\mathcal{U}$ for *Until* are temporal operators. We make use of standard Boolean abbreviations such as $\vee$ (or) and $\rightarrow$ (implies), *true* and *false*. In addition, we define the following abbreviations *Weak Next* $\bullet\varphi \equiv \neg\circ\neg\varphi$, *Eventually* $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$ and *Always* $\Box\varphi \equiv false\,\mathcal{R}\,\varphi$, where $\mathcal{R}$ is for *Release*.

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

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

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

Reactive Synthesis. The problem of reactive synthesis can be viewed as a game of the *environment* and the *agent*. The goal of reactive synthesis is to synthesize an agent strategy such that no matter how the environment behaves, the combined behavior trace of both players satisfy desired properties [Pnueli and Rosner, 1989]. Formally, a reactive synthesis problem is described as a tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, where $\mathcal{X}$ and $\mathcal{Y}$ are two disjoint sets of variables controlled by the *environment* and the *agent*, respectively, and φ is a logical specification formula over $\mathcal{X} \cup \mathcal{Y}$ expressing desired properties. A *deterministic* agent strategy is a function $\sigma_{ag}\colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$. A *play* is a sequence $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1)\cdots \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ over the alphabet $2^{\mathcal{X}\cup\mathcal{Y}}$. A play π is *compatible* with an agent strategy σ_{ag} if $\sigma_{ag}(\pi^{i-1}, X_i) = Y_i$ for every $i \geq 0$. Analogously, finite prefix π^k is *compatible* with σ_{ag} if $\sigma_{ag}(\pi^{i-1}, X_i) = Y_i$ for every $0 \leq i \leq k$. Given a synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, an agent strategy σ_{ag} *realizes* φ, with respect to the corresponding $\mathcal{X}$ and $\mathcal{Y}$, if every play π that is compatible with σ_{ag} satisfies φ. Sometimes, for simplicity, we do not explicitly mention $\mathcal{X}$ and $\mathcal{Y}$ if they are clear from the context. In this paper, we focus on LTL_f synthesis.

Definition 1 (LTL_f synthesis)**.** *The problem of* LTL_f *synthesis is described as a tuple* $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$*, where* φ *is an* LTL_f *formula over* $\mathcal{X} \cup \mathcal{Y}$*. Computing an agent strategy that realizes* φ *if one exists, is called the* LTL_f *synthesis problem.*

Solving LTL_f synthesis is known to be 2EXPTIME-complete [De Giacomo and Vardi, 2015].

Two-player Games. Classical solutions to LTL_f synthesis relies on a reduction to two-player games. A *two-player game* is described as a tuple $\mathbf{G} = (\mathbf{A}, W)$, where $\mathbf{A}$ is the game arena and W is the *winning objective*. For synthesis, we often take the corresponding deterministic automaton of the formula as the game arena, which is a tuple $\mathbf{A} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \delta)$, where $\delta\colon S \times 2^{\mathcal{X}\cup\mathcal{Y}} \rightarrow S$ is the *transition function*. Given a play $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1)\ldots \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$, running π on $\mathbf{A}$ gives us an infinite sequence $\rho = s_0 s_1 \ldots \in S^\omega$ such that s_0 is the initial state and $s_{i+1} = \delta(s_i, X_i \cup Y_i)$ for all $i \geq 0$. Since the transitions in $\mathbf{A}$ are all deterministic, we thus denote by $\rho = \mathsf{Run}(\pi, \mathbf{A})$ the unique sequence of running π on $\mathbf{A}$. Analogously, we denote by $\rho^k = \mathsf{Run}(\pi^k, \mathbf{A})$ the unique finite sequence of running π^k on $\mathbf{A}$, and $\rho^k = s_0 s_1 \ldots s_{k+1}$. The *winning objective* W indicates a set of desired plays for the agent. A play π is a *winning play* with respect to winning objective W if $\pi \in W$. In this paper, we only consider two-player games with *Reachability objectives* (Reachability games). More specifically, the winning objective $W = \mathrm{Reach}(F)$, and $\mathrm{Reach}(F) = \{\pi \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega \mid \rho = \mathsf{Run}(\pi, \mathbf{A}) \text{ and } \exists \ell \geq 0\colon \rho_\ell \in F\}$, where $F \subseteq S$ is a set of goal states. Intuitively, reachability games require that a goal state in F is visited at least once.

An agent strategy σ_{ag} is *winning* in $\mathbf{G} = (\mathbf{A}, \mathrm{Reach}(F))$ if every play π that is compatible with σ_{ag} is winning, i.e., $\pi \in W$. In *Reachability games*, $s \in S$ is a *winning* state for the agent (resp. environment) if the agent (resp. the environment) has a winning strategy in the game $\mathbf{G} = (\mathbf{A}', \mathrm{Reach}(F))$, where $\mathbf{A}' = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s, \delta)$, i.e., the same arena $\mathbf{A}$ but with the new initial state s. By $\mathsf{W}_{ag}(\mathbf{G})$ (resp. $\mathsf{W}_{env}(\mathbf{G})$) we denote the set of all agent (resp. environment) winning states.

Intuitively, W_{ag} represents the "agent winning region", from which the agent is able to win the game, no matter how the environment behaves.

Symbolic LTL$_f$ Synthesis. State-of-the-art LTL$_f$ synthesis tools [Bansal *et al.*, 2020; De Giacomo and Favorito, 2021] leverage the symbolic synthesis framework presented in [Zhu *et al.*, 2017; Tabajara and Vardi, 2019], which solves the synthesis problem via a symbolic reachability game computation. The game arena $\mathbf{A} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \delta)$ is compactly represented *symbolically*, by encoding the state space using a logarithmic number of propositions. The *symbolic* representation of $\mathbf{A}$ is a tuple $\mathbf{A}_s = (\mathcal{X}, \mathcal{Y}, \mathcal{Z}, I, \eta)$, where $\mathcal{Z}$ is a set of variables such that $|\mathcal{Z}| = \lceil \log |S| \rceil$, and every state $s \in S$ corresponds to an interpretation $Z \in 2^{\mathcal{Z}}$ over $\mathcal{Z}$. I is a Boolean formula satisfied only by the interpretation of the initial state s_0. $\eta\colon 2^{\mathcal{X}} \times 2^{\mathcal{Y}} \times 2^{\mathcal{Z}} \to 2^{\mathcal{Z}}$ is a Boolean function mapping interpretations X, Y and Z of the propositions of $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ to a new interpretation Z' of the propositions of $\mathcal{Z}$, such that if Z corresponds to a state $s \in S$ then Z' corresponds to the state $\delta(s, X \cup Y)$. The set of goal states is represented by a Boolean formula g over $\mathcal{Z}$ that is only satisfied by the interpretations of states in F. Accordingly, we denote the symbolic reachability game as $\mathbf{G}_s = (\mathbf{A}_s, \mathrm{Reach}(g))$.

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

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

The computation reaches a fixpoint when $w_{i+1} \equiv w_i$. At this point, no more states will be added, and so all agent winning states have been collected. By evaluating w_{i+1} on I we can know if there exists a winning strategy. If that is the case, t_{i+1} can be used to compute a winning strategy through the mechanism of Boolean synthesis [Fried *et al.*, 2016].

Nondeterministic Strategies. An agent strategy can also be *nondeterministic*, which prescribes for each history $\pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$, and an environment action $X \in 2^{\mathcal{X}}$, a set of agent actions to choose from nondeterministically. Formally, a *nondeterministic* agent strategy is denoted as a function $\Pi_{ag}\colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{2^{\mathcal{Y}}}$. We denote the set of deterministic strategies induced by a nondeterministic strategy Π_{ag} as the set $[[\Pi_{ag}]]$. Every deterministic strategy $\sigma_{ag} \in [[\Pi_{ag}]]$ is obtained by a corresponding choice function $\mathsf{ch}\colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ such that for every $\pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$ that is compatible with ch and $X \in 2^{\mathcal{X}}$, $\mathsf{ch}(\pi^k, X) \in \Pi_{ag}(\pi^k, X)$ and $\sigma_{ag}(\pi^k, X) = \mathsf{ch}(\pi^k, X)$. Nondeterministic strategy Π_{ag} realizes φ if every induced strategy $\sigma_{ag} \in [[\Pi_{ag}]]$ realizes φ. Sometimes, for simplicity, we say σ_{ag} (resp. Π_{ag}) is a winning strategy of φ, instead of σ_{ag} (resp. Π_{ag}) realizes φ. A trace π (finite or infinite) is *compatible* with a nondeterministic strategy Π_{ag} if $Y_i \in \Pi_{ag}(\pi^{i-1}, X_i)$ for every $i \geq 0$.
Comparing Strategies. Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ be a synthesis problem, and Π_{ag} a winning strategy of φ. We consider the permissiveness of Π_{ag} with respect to the deterministic strategies that it induces.

Definition 2. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ be a synthesis problem, Π_{ag} and Π'_{ag} two winning strategies of φ. Π_{ag} is more permissive than Π'_{ag} if $[[\Pi'_{ag}]] \subseteq [[\Pi_{ag}]]$.*

3 Maximally Permissive Strategy

We first introduce the notion of maximally permissive strategy for a task, denoted by MaxSet, which consists of the entire set of deterministic winning strategies for the task, from which to choose one during the execution. Then we show that for special cases, we can compute a single nondeterministic strategy to capture MaxSet. However, in general, this does not happen for LTL$_f$ specifications.

Definition 3 (Maximally Permissive Strategy)**.** *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ be a synthesis problem, the maximally permissive strategy of φ is $\mathsf{MaxSet}(\varphi) = \{\sigma_{ag} \mid \sigma_{ag}$ is a deterministic winning strategy of $\varphi\}$.*

Theorem 1. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ be a synthesis problem. If there exists a nondeterministic winning strategy Π_{ag} that is more permissive than every other winning strategy, then $[[\Pi_{ag}]] = \mathsf{MaxSet}(\varphi)$.*[1]

Proof. Suppose there exists a nondeterministic winning strategy Π_{ag} that is more permissive than every other winning strategy Π'_{ag}, such that $[[\Pi'_{ag}]] \subseteq [[\Pi_{ag}]]$. Note that every deterministic strategy σ_{ag} can be considered as a special case of a nondeterministic strategy, in which case, $[[\sigma_{ag}]] = \{\sigma_{ag}\}$. Therefore, it holds that $\sigma_{ag} \in [[\Pi_{ag}]]$ for every deterministic winning strategy of φ, and so $[[\Pi_{ag}]] = \mathsf{MaxSet}(\varphi)$. ☐

Although it is practical to utilize a single nondeterministic strategy Π_{ag} to express the maximally permissive strategy $\mathsf{MaxSet}(\varphi)$, this only applies to the cases where φ specifies a *safety property*, meaning that every violating trace has a finite bad prefix that falsifies the specification φ. This is because the problem of synthesizing safety properties can be reduced to a safety game, which always has a nondeterministic winning strategy Π_{ag} that is more permissive than every other winning strategy [Bernet *et al.*, 2002]. However, this result does not apply to properties that are not safety [Bernet *et al.*, 2002]. In particular, we give here a direct proof for LTL$_f$ specifications.

Theorem 2. *In general, LTL$_f$ specifications do not admit a nondeterministic winning strategy Π_{ag} that is more permissive than every other strategy.*

Proof. We prove by showing that such a strategy does not even exist for a simple LTL$_f$ specification $\varphi = \Diamond b$. In particular, we have $b \in \mathcal{Y}$ s.t. b is under the control of the agent. It is trivial to see that no matter how the environment behaves on $\mathcal{X}$, the satisfaction of $\Diamond b$ only relies on the agent. Suppose Π_{ag} is a nondeterministic winning strategy of $\Diamond b$ that is

[1]It should be noted that the definition of maximally permissive strategy and Theorem 1 also applies to LTL synthesis.

more permissive than every other one. For every deterministic strategy $\sigma_{ag} \in [[\Pi_{ag}]]$, its corresponding choice function $\mathsf{ch}\colon (2^{\{b\}})^* \to 2^{\{b\}}$ satisfies that $\exists k \geq 0$ such that ch emits b at time step k (note that $\mathcal{X}$ can be ignored here), and ch keeps emitting $\neg b$ for $0 \leq i < k$. Otherwise, σ_{ag} is indeed not a winning strategy. Therefore, we can label every choice function with its corresponding k-value. We now take the choice function ch_m that has the greatest k-value, comparing to other choice functions of Π_{ag}. Then we can construct another choice function ch' that keeps copying ch_m but postponing the moment of outputting b for one timestep.

Indeed, ch' leads to a deterministic winning strategy σ'_{ag}, since σ'_{ag} outputs b at timestep $m+1$ and thus the compatible play π (a unique sequence, since $\mathcal{X}$ can be ignored in this example) is such that $\pi^{m+1} \models \Diamond b$ holds. Therefore, $\sigma'_{ag} \in \mathsf{MaxSet}(\Diamond b)$, but $\sigma'_{ag} \notin [[\Pi_{ag}]]$ leading to a contradiction. Thus, in general, LTL$_f$ specifications do not admit a nondeterministic winning strategy Π_{ag} that is more permissive than every other strategy. □

4 MaxSet of LTL$_f$ Synthesis

We now explain how to compute $\mathsf{MaxSet}(\varphi)$ of LTL$_f$ synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$, by reasoning on its corresponding reachability game. Recall that a winning strategy for safety games can be interpreted as a "staying in the winning region" strategy, since as long as the game stays in the winning region, the agent is able to force the game to move to another state that still belongs to the winning region [Bernet *et al.*, 2002]. Nevertheless, for reachability games, such strategies do not necessarily lead to winning. This is because such a strategy can possibly generate a play that only stays in the winning region but never visits the goal states, which is indeed not a winning play. Therefore, apart from "staying in the winning region" such that the agent has the ability to win, no matter how the strategy keeps "deferring" the winning moment, a winning strategy also needs to start "non-deferring" such that eventually reaching goal states.

Here, we show how to capture the entire set of winning strategies of reachability games with two nondeterministic strategies. Intuitively, the first nondeterministic strategy keeps the agent in the winning region, where the agent is always able to win the game, which is called *deferring* strategy. The second nondeterministic strategy aims to taking the agent to goal states without any hesitation such that aggressively winning the game, which is called *non-deferring* strategy. Both strategies are defined with respect to the agent winning region, therefore, we assume that the agent winning region $\mathsf{W}_{ag} = \bigcup_{0 \leq i \leq n} \mathsf{W}_i$ is computed apriori via a least fixpoint computation and $s_0 \in \mathsf{W}_{ag}$ such that there is at least one agent winning strategy of the reachability game [De Giacomo and Vardi, 2015].

4.1 Deferring Strategy

For reachability games $\mathbf{G} = (\mathbf{A}, \mathrm{Reach}(F))$, a *deferring* strategy should keep the play in the agent winning region W_{ag}, if the reachability goal F is not visited yet, and so the agent is able to keep deferring the winning moment.

Definition 4 (Deferring strategy of reachability games)**.** *Let $\boldsymbol{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game and W_{ag} the set of agent winning states. $\sigma^{\mathrm{df}}_{ag}$ is a deferring strategy, if for every play that is compatible with $\sigma^{\mathrm{df}}_{ag}$, $\rho = \mathsf{Run}(\pi, \boldsymbol{A})$, such that $\rho = s_0 s_1 \ldots$ satisfies $s_0 \in \mathsf{W}_{ag}$ and $\forall \ell \geq 0$, if $s_\ell \notin F$ then $s_{\ell+1} \in \mathsf{W}_{ag}$.*

Lemma 1. *Let $\boldsymbol{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game, and W_{ag} be the agent winning region. Then $s_0 \in \mathsf{W}_{ag}$ iff there exists a deferring strategy.*

In order to obtain a deferring strategy, we define a strategy generator based on the winning region W_{ag}, represented as a nondeterministic transducer, where nondeterminism is of the kind "don't-care": all nondeterministic choices are equally good. The strategy generator $\mathcal{T}^{\mathrm{df}} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s_0, \varrho, \tau)$ is constructed as follows:

- $2^{\mathcal{X} \cup \mathcal{Y}}$, S and s_0 are the same as in $\mathbf{A}$;
- $\varrho^{\mathrm{df}}\colon S \times 2^{\mathcal{X}} \to 2^S$ is the transition function such that $\varrho^{\mathrm{df}}(s, X) = \{s' | s' = \delta(s, X \cup Y) \text{ and } Y \in \tau^{\mathrm{df}}(s, X)\}$;
- $\tau^{\mathrm{df}}\colon S \times 2^{\mathcal{X}} \to 2^{2^{\mathcal{Y}}}$ is the output function such that $\forall X \in 2^{\mathcal{X}}$, $\tau^{\mathrm{df}}(s, X) = \{Y \mid \delta(s, X \cup Y) \in \mathsf{W}_{ag}\}$ if $s \in \mathsf{W}_{ag} \backslash F$; otherwise, $\tau^{\mathrm{df}}(s, X) = 2^{\mathcal{Y}}$.

This transducer represents a nondeterministic deferring strategy $\Pi^{\mathrm{df}}_{ag}\colon (2^{\mathcal{X} \cup \mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{2^{\mathcal{Y}}}$ in the following way: $\forall \pi^k \in (2^{\mathcal{X} \cup \mathcal{Y}})^*$ that is compatible with Π^{df}_{ag}, and $\forall X \in 2^{\mathcal{X}}$,

$$\Pi^{\mathrm{df}}_{ag}(\pi^k, X) = \begin{cases} Y \in 2^{\mathcal{Y}} & \text{if } \rho^k \text{ visits } F, \\ \tau^{\mathrm{df}}(s_{k+1}, X) & \text{otherwise.} \end{cases}$$

where $\rho^k = \mathsf{Run}(\pi^k, \mathbf{A})$ such that $\rho^k = s_0 s_1 \ldots s_{k+1}$. Every deterministic deferring strategy $\sigma^{\mathrm{df}}_{ag}$ is generated from Π^{df}_{ag} by a corresponding choice function $\mathsf{ch}^{\mathrm{df}}\colon (2^{\mathcal{X} \cup \mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ such that $\forall \pi^k \in (2^{\mathcal{X} \cup \mathcal{Y}})^*$ that is compatible with $\mathsf{ch}^{\mathrm{df}}$, and $\forall X \in 2^{\mathcal{X}}$, $\mathsf{ch}^{\mathrm{df}}(\pi^k, X) \in \Pi^{\mathrm{df}}_{ag}(\pi^k, X)$ and $\sigma_{ag}(\pi^k, X) = \mathsf{ch}^{\mathrm{df}}(\pi^k, X)$.

Theorem 3. *Let $\boldsymbol{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game, and Π^{df}_{ag} be computed as above. Then deterministic strategy σ_{ag} is a deferring strategy iff $\sigma_{ag} \in [[\Pi^{\mathrm{df}}_{ag}]]$.*

4.2 Non-Deferring Strategy

Different from a deferring strategy that keeps deferring the winning moment, a non-deferring strategy brings the play closer to goal states F at every timestep until reaching it. In order to represent the distance towards F, for every state $s \in \mathsf{W}_{ag}$, where $\mathsf{W}_{ag} = \bigcup_{0 \leq i \leq n} \mathsf{W}_i$, computed by a least fixpoint computation, we introduce a distance value $\mathsf{d}(s)$ that indicates the shortest distance from s to F. Therefore, $\forall s \in \mathsf{W}_{ag}$, if $s \in F$ (i,e., $s \in \mathsf{W}_0$) then $\mathsf{d}(s) = 0$; otherwise, since $s \in \mathsf{W}_{i+1} \backslash \mathsf{W}_i$ for some $0 \leq i \leq n$, we have $\mathsf{d}(s) = i + 1$. We now introduce non-deferring strategy.

Definition 5 (Non-Deferring strategy of reachability games)**.** *Let $\boldsymbol{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game and W_{ag} be the agent winning region. $\sigma^{\mathrm{ndf}}_{ag}$ is a non-deferring strategy, if for every play π that is compatible with $\sigma^{\mathrm{ndf}}_{ag}$, $\rho = \mathsf{Run}(\pi, \boldsymbol{A})$ such that $\rho = s_0 s_1 \ldots$ satisfies $s_0 \in \mathsf{W}_{ag}$ and $\exists \ell \geq 0. \mathsf{d}(s_\ell) = 0$ and $\forall 0 \leq j \leq \ell. \mathsf{d}(s_{j+1}) = \mathsf{d}(s_j) - 1$.*

Lemma 2. *Let $\mathbf{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game, and W_{ag} be the agent winning region. Then $s_0 \in \mathsf{W}_{ag}$ iff there exists a non-deferring strategy.*

In order to obtain a non-deferring strategy, we also define a strategy generator based on the winning region $\mathsf{W}_{ag} = \bigcup_{0 \le i \le n} \mathsf{W}_i$, represented as a nondeterministic transducer $\mathcal{T}^{\mathrm{ndf}} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \varrho^{\mathrm{ndf}}, \tau^{\mathrm{ndf}})$, where

- $2^{\mathcal{X}\cup\mathcal{Y}}$, S and s_0 are the same as in $\mathbf{A}$;
- $\varrho^{\mathrm{ndf}} \colon S \times 2^{\mathcal{X}} \to 2^S$ is the transition function such that $\varrho^{\mathrm{ndf}}(s, X) = \{s' | s' = \delta(s, X \cup Y) \text{ and } Y \in \tau^{\mathrm{ndf}}(s)\}$;
- $\tau^{\mathrm{ndf}} \colon S \times 2^{\mathcal{X}} \to 2^{2^{\mathcal{Y}}}$ is the output function such that $\forall X \in 2^{\mathcal{X}}$, $\tau^{\mathrm{ndf}}(s, X) = \{Y | \delta(s, X \cup Y) \in \mathsf{W}_i\}$ if $s \in \mathsf{W}_{i+1} \backslash \mathsf{W}_i$, otherwise $\tau^{\mathrm{ndf}}(s, X) = 2^{\mathcal{Y}}$.

This transducer represents a nondeterministic non-deferring strategy $\Pi^{\mathrm{ndf}}_{ag} \colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{2^{\mathcal{Y}}}$ in the following way: $\forall \pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$ that is compatible with Π^{ndf}_{ag}, and $\forall X \in 2^{\mathcal{X}}$

$$\Pi^{\mathrm{ndf}}_{ag}(\pi^k, X) = \begin{cases} Y \in 2^{\mathcal{Y}} & \text{if } \rho^k \text{ visits } F, \\ \tau^{\mathrm{ndf}}(s_{k+1}, X) & \text{otherwise.} \end{cases}$$

where $\rho^k = \mathsf{Run}(\pi^k, \mathbf{A})$ such that $\rho^k = s_0 s_1 \ldots s_{k+1}$. Every deterministic strategy $\sigma^{\mathrm{ndf}}_{ag}$ is generated from Π^{ndf}_{ag} by a corresponding choice function $\mathsf{ch}^{\mathrm{ndf}} \colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ such that $\forall \pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$ that is compatible with $\mathsf{ch}^{\mathrm{ndf}}$, and $\forall X \in 2^{\mathcal{X}}, \mathsf{ch}^{\mathrm{ndf}}(\pi^k, X) \in \Pi^{\mathrm{ndf}}_{ag}(\pi^k, X)$ and $\sigma^{\mathrm{ndf}}_{ag}(\pi^k, X) = \mathsf{ch}^{\mathrm{ndf}}(\pi^k, X)$.

Theorem 4. *Let $\mathbf{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game, and Π^{ndf}_{ag} be computed as above. Then deterministic strategy σ_{ag} is a non-deferring strategy iff $\sigma_{ag} \in [[\Pi^{\mathrm{ndf}}_{ag}]]$.*

Theorem 5. *Let $\mathbf{G} = (\boldsymbol{A}, \mathrm{Reach}(F))$ be a reachability game, Π^{df}_{ag} and Π^{ndf}_{ag} be computed as above, respectively. Then $[[\Pi^{\mathrm{ndf}}_{ag}]] \subseteq [[\Pi^{\mathrm{df}}_{ag}]]$.*

4.3 Maximally Permissive Strategy

We now explain how to obtain the maximally permissive strategy MaxSet through nondeterministic strategies Π^{df}_{ag} and Π^{ndf}_{ag}. To do so, we start with showing how to induce deterministic strategies out of Π^{df}_{ag} and Π^{ndf}_{ag}. Every deterministic strategy σ_{ag} is generated from nondeterministic strategies Π^{df}_{ag} and Π^{ndf}_{ag} by a corresponding choice function $\mathsf{ch} \colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$. Intuitively speaking, apart from restricting σ_{ag} to be a deferring strategy such that being able to postpone the winning moment, ch also restricts σ_{ag} to change to a non-deferring strategy such that eventually reaching the goal states. Formally, we require ch satisfying both of the following conditions:

1. for every $\pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$ compatible with ch, and $\forall X \in 2^{\mathcal{X}}$. $\mathsf{ch}(\pi^k, X) \in \Pi^{\mathrm{df}}_{ag}(\pi^k, X)$;
2. for every $\pi \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ compatible with ch, $\exists j \ge 0$ such that $\forall h \ge j, \mathsf{ch}(\pi^{h-1}, X_h) \in \Pi^{\mathrm{ndf}}_{ag}(\pi^{h-1}, X_h)$.

Every choice function $\mathsf{ch} \colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ that satisfies the above characteristic is able to induce a deterministic strategy out of nondeterministic strategies Π^{df}_{ag} and Π^{ndf}_{ag}. Note that choice functions are a general mathematical means to extract deterministic strategies from nondeterministic ones. Their role here is to prove soundness (every choice function with the above characteristic will create a deterministic winning strategy) and completeness (every deterministic winning strategy can be captured by one such choice function). We denote the set of all induced deterministic strategies by $[[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]]$. The following theorem shows that $[[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]]$ can capture the maximally permissive strategy.

Theorem 6. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi \rangle$ be an LTL_f synthesis problem, and $\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}$ be computed as above. Then $[[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]] = \mathsf{MaxSet}(\varphi)$.*

Proof. We prove $[[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]] = \mathsf{MaxSet}(\varphi)$ by showing that $\sigma_{ag} \in [[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]]$ iff $\sigma_{ag} \in \mathsf{MaxSet}(\varphi)$.

($\Leftarrow$) We need to prove that every deterministic winning strategy $\sigma_{ag} \in \mathsf{MaxSet}(\varphi)$ is such that $\sigma_{ag} \in [[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]]$. We now show that we can construct a choice function $\mathsf{ch} \colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ that is able to generate the same agent strategy from Π^{df}_{ag} and Π^{ndf}_{ag}. First, we require ch being such that $\forall \pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$ that is compatible with σ_{ag}, $\forall X \in 2^{\mathcal{X}}, \mathsf{ch}(\pi^k, X) = \sigma_{ag}(\pi^k, X)$. Indeed, every play π that is compatible with ch is also compatible with σ_{ag}. Therefore, $\forall \pi \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ that is compatible with ch, it is trivial to see that $\forall k \ge 0, \mathsf{ch}(\pi^{k-1}, X_k) = \sigma_{ag}(\pi^{k-1}, X_k) \in \Pi^{\mathrm{df}}_{ag}(\pi^{k-1}, X_k)$, since π has to stay in W_{ag} before reaching F, otherwise, σ_{ag} cannot be a winning strategy. Moreover, $\exists \ell \ge 0, \pi^\ell \models \varphi$. Note that we take the smallest ℓ for $\pi^\ell \models \varphi$. Indeed, $\rho^\ell = \mathsf{Run}(\pi^\ell, \mathbf{A})$ such that $\rho^\ell = s_0 s_1 \ldots s_{\ell+1}$ satisfies that $s_{\ell+1} \in F$. Therefore, we have that $\mathsf{ch}(\pi^{\ell-1}, X_\ell) = \sigma_{ag}(\pi^{\ell-1}, X_\ell) \in \Pi^{\mathrm{ndf}}_{ag}(\pi^{\ell-1}, X_\ell)$, which, by construction, consists of all the Ys that lead s^ℓ to F in one step with X_ℓ. Furthermore, $\forall h > \ell, \mathsf{ch}(\pi^{h-1}, X_h) = \sigma_{ag}(\pi^{h-1}, X_h) \in \Pi^{\mathrm{ndf}}_{ag}(\pi^{h-1}, X_h)$, since ρ^h visits F already, and so $\Pi^{\mathrm{ndf}}_{ag}(\pi^{h-1}, X_h) = 2^{\mathcal{Y}}$. We thus conclude that $\sigma_{ag} \in [[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]]$ holds.

($\Rightarrow$) We need to prove that every deterministic strategy $\sigma_{ag} \in [[\Pi^{\mathrm{df}}_{ag}, \Pi^{\mathrm{ndf}}_{ag}]]$ is a winning strategy, such that $\sigma_{ag} \in \mathsf{MaxSet}(\varphi)$. Indeed, σ_{ag} is generated by a corresponding choice function $\mathsf{ch} \colon (2^{\mathcal{X}\cup\mathcal{Y}})^* \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ that follows our defined conditions. That is to say, $\forall \pi^k \in (2^{\mathcal{X}\cup\mathcal{Y}})^*$ that is compatible with σ_{ag}, and $\forall X \in 2^{\mathcal{X}}, \sigma_{ag}(\pi^k, X) \in \Pi^{\mathrm{df}}_{ag}(\pi^k, X)$. Moreover, $\forall \pi \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ that is compatible with σ_{ag}, $\exists j \ge 0$ such that $\forall h \ge j, \sigma_{ag}(\pi^{h-1}, X_h) \in \Pi^{\mathrm{ndf}}_{ag}(\pi^{h-1}, X_h)$. Thus, for every play π that is compatible with σ_{ag}, $\rho = \mathsf{Run}(\pi, \mathbf{A})$ such that $\rho = s_0 s_1 \ldots$ satisfies that $s_0 \in \mathsf{W}_{ag}$ and $\forall \ell \ge 0$, if $s_{\ell+1} \notin \mathsf{W}_{ag}$, then $\rho^\ell = s_0 s_1 \ldots s_{\ell+1}$ definitely has visited F already such that φ is satisfied already. This is because Π^{df}_{ag} restricts π to stay in W_{ag} before reaching F. Otherwise, from $\ell = j$, σ_{ag} switches to Π^{ndf}_{ag}, which restricts ρ to eventually reach F such that φ is satisfied. Hence, σ_{ag} is a winning strategy and so $\sigma_{ag} \in \mathsf{MaxSet}(\varphi)$. □

5 Implementation and Empirical Evaluation

In this section, we first describe how to leverage synthesis framework being integrated in state-of-the-art LTL_f synthesis tools to compute the maximally permissive strategy. Then, by empirical evaluation, we show computing the maximally permissive strategy only brings a minimal overhead with respect to the computation of a single winning strategy.

5.1 Symbolic MaxSet Computation

Recall that the symbolic LTL_f synthesis framework presented in [Zhu *et al.*, 2017; Tabajara and Vardi, 2019] consists of computing two Boolean formulas w and t, which represent the agent winning region W_{ag} and tuples of winning states, environment actions with suitable agent actions, respectively. We observe that Boolean formula t over $\mathcal{Z} \cup \mathcal{X} \cup \mathcal{Y}$, in fact, is compatible with the non-deferring strategy generator $\mathcal{T}^{\text{ndf}} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \varrho^{\text{ndf}}, \tau^{\text{ndf}})$ introduced in Section 4.2. More specifically, t is satisfied by interpretation $(Z \cup X \cup Y) \in 2^{\mathcal{Z}} \times 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ iff $Y \in \tau^{\text{ndf}}(s, X)$, where s is the corresponding state of Z. This is because, during the fixpoint computation, an interpretation $(Z \cup X \cup Y) \in 2^{\mathcal{Z}} \times 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ satisfies t_{i+1} if: either $(Z \cup X \cup Y)$ satisfies t_i; or Z was identified as a new winning state ($Z \models w_{i+1} \wedge \neg w_i$), and so for every environment action $X \in 2^{\mathcal{X}}$ we can move from Z to w_i by setting agent action to some $Y \in 2^{\mathcal{Y}}$ as long as following the transition function, expressed by $w_i(\eta(\mathcal{Z}, \mathcal{X}, \mathcal{Y}))$. Note that the second case restricts the agent to be *non-deferring* such that the next transition will move closer to goal states. For nondeterministic deferring strategy Π^{df}_{ag}, we compute a boolean formula t^{df} over $\mathcal{Z} \cup \mathcal{X} \cup \mathcal{Y}$ as follows:

$$t^{\text{df}}(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) = t(\mathcal{Z}, \mathcal{X}, \mathcal{Y}) \vee (w_{i+1}(\mathcal{Z}) \wedge w_{i+1}(\eta(\mathcal{Z}, \mathcal{X}, \mathcal{Y}))).$$

Note that $(Z \cup X \cup Y) \in 2^{\mathcal{Z}\cup\mathcal{X}\cup\mathcal{Y}}$ satisfies t^{df} if: either $(Z \cup X \cup Y)$ satisfies t, since every non-deferring strategy is a deferring strategy; or the agent can move back to the winning region $w_{i+1}(\mathcal{Z})$ with another $Y \in 2^{Y}$ as long as following the transition function, expressed by $w_{i+1}(\eta(\mathcal{Z}, \mathcal{X}, \mathcal{Y}))$. In this case, t^{df} is satisfied by interpretation $(Z \cup X \cup Y) \in 2^{\mathcal{Z}\cup\mathcal{X}\cup\mathcal{Y}}$ iff they are compatible with the deferring strategy generator $\mathcal{T}^{\text{df}} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \varrho^{\text{df}}, \tau^{\text{df}})$ such that $Y \in \tau^{\text{df}}(s, X)$, where s is the corresponding state of Z.

5.2 Empirical Evaluation

Implementation. We implemented the symbolic maximally permissive strategy computation technique for LTL_f synthesis by extending the symbolic synthesis framework [Zhu *et al.*, 2017; Tabajara and Vardi, 2019] integrated in state-of-the-art synthesis tools [Bansal *et al.*, 2020; De Giacomo and Favorito, 2021]. In particular, we based on LTL_f synthesis tool LYDIA [2], the overall best performing tool, and implemented our fixpoint-based computation presented in Section 5.1. More specifically, the computation consists of two steps. In the first step, we based on the code of LYDIA, to perform realizability checking, and the nondeterministic non-deferring strategy is also obtained as a side-effect, represented in Binary Decision Diagrams (BDDs). The second step makes use of Boolean operations provided by BDD library CUDD-3.0.0 [Somenzi, 2016] to compute the nondeterministic deferring strategy. [3]

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

[3]Source code at https://github.com/Shufang-Zhu/SyftMax

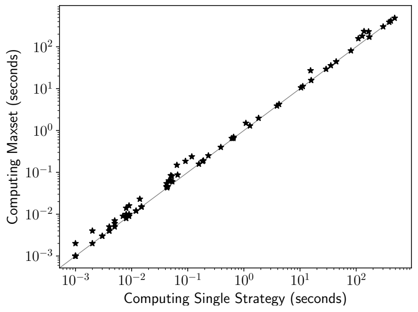

Figure 1: Comparison results on realizable benchmarks.

Experiment Setup. All tests were ran on a computer cluster. Each test took exclusive access to a node with Intel(R) Xeon(R) CPU E5-2650 v2 processors running at 2.60GHz. Time out was set to 300 seconds.

Benchmarks. We took the LTL_f synthesis benchmarks from literature [Tabajara and Vardi, 2019; Bansal *et al.*, 2020; De Giacomo and Favorito, 2021]. The goal of this experiment is to evaluate the overhead caused by computing the maximally permissive strategy, we thus filtered the benchmarks by taking only the realizable ones that can be solved within the time limit, which are 391 in total.

Evaluation Results. We evaluated the overhead caused by maximally permissive strategy computation by comparing with the time cost of computing a single strategy, which is represented by the time taken from receiving a synthesis instance to finishing realizability checking, by then Boolean formula t, i.e., the nondeterministic non-deferring strategy, is also obtained. The Boolean synthesis time for "choosing" a deterministic strategy out of t is omitted, for a fair comparison, since the same procedure would also be required by "choosing" a deterministic strategy out of the maximally permissive strategy. Figure 1 displays a scatter plot (in log scale) comparing the time cost of two computations for each realizable instance. The gray line represents the points where the x-axis value is equal to the y-axis value. As shown in Figure 1, the points are closely above the gray line, this demonstrates that computing the maximally permissive strategy only brings minor overhead comparing to computing a single strategy.

6 Conclusions

Maximally permissive strategies are of interest when we want to allow for maximal freedom to the agent while still achieving the desired task. We have shown that we can naturally characterize maximally permissive strategies for LTL_f specifications with two nondeterministic strategies and a constraint requiring to eventually switch from one to the other. Moreover, such a pair can be computed with minimal overhead. Given this nice computational result, it would be interesting to understand if maximally permissive strategies can be used to benefit LTL_f synthesis itself. Indeed, state-of-the-art LTL_f synthesis tools integrate techniques to compose specifications. Instead, one could compose the maximally permissive strategy of each specification piece, leading to an even smaller state space, and thus bring promising computational benefits. We leave this for future work.

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), the JPMorgan AI Faculty Research Award "Resilience-based Generalized Planning and Strategic Reasoning"

References

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila A. McIlraith. Planning with temporally extended goals using heuristic search. In *ICAPS*, pages 342–345. AAAI, 2006.

[Banihashemi *et al.*, 2016] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Online agent supervision in the situation calculus. In *IJCAI*, pages 922–928. IJCAI/AAAI Press, 2016.

[Banihashemi *et al.*, 2018] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Hierarchical agent supervision. In *AAMAS*, pages 1432–1440, 2018.

[Bansal *et al.*, 2020] Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pages 9766–9774, 2020.

[Bernet *et al.*, 2002] Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO Theor. Informatics Appl.*, 36(3):261–275, 2002.

[Camacho *et al.*, 2019] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, pages 58–67. AAAI Press, 2019.

[Cassandras and Lafortune, 2008] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, second edition, 2008.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. 1–2(147), 2003.

[De Giacomo and Favorito, 2021] Giuseppe De Giacomo and Marco Favorito. Compositional approach to translate LTL_f/LDL_f into deterministic finite automata. In *ICAPS*, pages 122–130, 2021.

[De Giacomo and Rubin, 2018] Giuseppe De Giacomo and Sasha Rubin. Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals. In *IJCAI*, pages 4729–4735, 2018.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 2015.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Yves Lespérance, and Christian J. Muise. On supervising agents in situation-determined congolog. In *AAMAS*, pages 1031–1038. IFAAMAS, 2012.

[De Giacomo *et al.*, 2013] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardiña. Automatic behavior composition synthesis. *Artif. Intell.*, 196:106–142, 2013.

[De Giacomo *et al.*, 2020] Giuseppe De Giacomo, Bastien Maubert, and Aniello Murano. Nondeterministic strategies and their refinement in strategy logic. In *KR*, pages 294–303, 2020.

[Ehlers *et al.*, 2017] R. Ehlers, S. Lafortune, S. Tripakis, and M. Vardi. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems*, 27(2), 2017.

[Fried *et al.*, 2016] Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. Bdd-based boolean functional synthesis. In *CAV*, pages 402–421, 2016.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Ghallab *et al.*, 2016] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge, 2016.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, 1989.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT, 2001.

[Sakakibara and Ushio, 2020] Ami Sakakibara and Toshimitsu Ushio. On-line permissive supervisory control of discrete event systems for scltl specifications. *IEEE Control. Syst. Lett.*, 4(3):530–535, 2020.

[Somenzi, 2016] Fabio Somenzi. CUDD: CU decision diagram package 3.0.0. universiy of colorado at boulder. 2016.

[Tabajara and Vardi, 2019] Lucas Martinelli Tabajara and Moshe Y. Vardi. Partitioning techniques in LTL_f synthesis. In Sarit Kraus, editor, *IJCAI*, pages 5599–5606, 2019.

[Wonham and Cai, 2019] W. Murray Wonham and Kai Cai. *Supervisory Control of Discrete-Event Systems*. Springer, 2019.

[Wonham and Ramadge, 1987] WM Wonham and PJ Ramadge. On the supremal controllable sub-language of a given language. *SIAM J Contr. Optim.*, 25(3):637–659, 1987.

[Zhu *et al.*, 2017] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTL_f synthesis. In *IJCAI*, pages 1362–1369, 2017.