Formal Methods in System Design (2022) 61:139–163
https://doi.org/10.1007/s10703-023-00413-2

Finite-trace and generalized-reactivity specifications in temporal synthesis

Giuseppe De Giacomo[1] · Antonio Di Stasio[1] · Lucas M. Tabajara[2] · Moshe Y. Vardi[2] · Shufang Zhu[1]

Received: 2 April 2022 / Accepted: 25 January 2023 / Published online: 15 March 2023

Abstract

Linear Temporal Logic (LTL) synthesis aims at automatically synthesizing a program that complies with desired properties expressed in LTL. Unfortunately it has been proved to be too difficult computationally to perform full LTL synthesis. There have been two success stories with LTL synthesis, both having to do with the form of the specification. The first is the GR(1) approach: use safety conditions to determine the possible transitions in a game between the environment and the agent, plus one powerful notion of fairness, Generalized Reactivity(1), or GR(1). The second, inspired by AI planning, is focusing on finite-trace temporal synthesis, with LTL $_f$ (LTL on finite traces) as the specification language. In this paper we take these two lines of work and bring them together. We first study the case in which we have an LTL $_f$ agent goal and a GR(1) environment specification. We then add to the framework safety conditions for both the environment and the agent, obtaining a highly expressive yet still scalable form of LTL synthesis.

Keywords Reactive synthesis · LTL $_f$ · GR(1) · Games

✉ Shufang Zhu
shufang.zhu@cs.ox.ac.uk

Giuseppe De Giacomo
giuseppe.degiacomo@cs.ox.ac.uk

Antonio Di Stasio
antonio.distasio@cs.ox.ac.uk

Lucas M. Tabajara
l.martinelli.tabajara@gmail.com

Moshe Y. Vardi
vardi@cs.rice.edu

[1] Department of Computer Science, University of Oxford, Oxford, UK

[2] Computer Science, Rice University, Houston, USA

1 Introduction

Program synthesis is considered the culmination of the ideal of declarative programming [1, 2]. By describing a system in terms of what it should do, instead of how it should do it, we are able, on the one hand, to simplify the program design process while avoiding human mistakes and, on the other hand, to allow an autonomous agent to self-program itself just from high-level specifications. Linear Temporal Logic (LTL) synthesis [3] is possibly one of the most popular variants of program synthesis, being the problem of automatically designing a reactive system with the guarantee that all its behaviors comply with desired dynamic properties expressed in LTL, the most used system/process specification language in Formal Methods. Unfortunately this dream of LTL synthesis has proven to be too difficult, and, in spite of a full-fledged theory, we still do not have good scalable algorithms after more than 30 years [4].

There have been two successful responses to these difficulties, both having to do with limiting the expressive power of the formalism used for the specification. The first approach, developed in Formal Methods, has been what we may call the GR(1), response [5]: essentially you focus on safety conditions, determining the possible transitions in a game between the environment and the agent, plus one powerful notion of fairness called Generalized Reactivity(1), or GR(1). This approach has found numerous applications, for example, in robotic motion-and-mission planning [6]. The second approach, developed in AI and inspired by classical AI planning, is of finite-horizon temporal synthesis, with LTL $_f$ (LTL on finite traces) [7] as the specification language. In this approach [8], we specify the agent's goal in LTL $_f$, together possibly with environment specifications, such as safety conditions, possibly specified as a nondeterministic planning domain [9–13], or simple fairness and stability conditions (both special cases of GR(1) fairness) [14]. There are also studies in which general LTL environment specifications are used for LTL $_f$ goals, but in this case the difficulties of handling LTL can indeed manifest [12, 15, 16]. Since LTL $_f$ is a fragment of LTL, as shown in [7], the problem of LTL $_f$ synthesis under LTL environment specifications can be reduced to LTL synthesis, as, e.g., explicitly pointed out by [12]. However, LTL synthesis algorithms do not scale well due to the difficulty of Büchi automata determinization, see e.g., [1].

In this work we propose to take these two lines of work, which are really the only successful stories in LTL synthesis, and bring them together. We first study the case in which we have an LTL $_f$ agent goal and a GR(1) environment specification. We propose an approach based on using the automaton corresponding to the LTL $_f$ goal as the game arena on which the environment has to satisfy its GR(1) environment specification. This means that we are able to reduce the problem to that of GR(1) synthesis over the new arena. We prove the correctness of the approach.

We then add to the framework safety conditions for both the environment and the agent, obtaining a highly expressive yet still scalable form of LTL synthesis. These two kinds of safety conditions differ, since the environment needs to maintain its safety indefinitely (as usual for safety), while the agent has to maintain its safety conditions only until s/he fulfills its LTL $_f$ goal, i.e., within a finite horizon, something that makes them similar to "maintenance goals" in Planning [17]. We show that we can specify these safety conditions in a very general way by using LTL $_f$. In particular, our safety conditions require that *all prefixes* of a trace satisfy an LTL $_f$ formula. For the environment safety conditions, we consider all finite prefixes of infinite traces, while for the agent safety conditions, we consider all prefixes of the finite trace satisfying the agent's LTL $_f$ goal. Again, we prove the correctness of our approach and demonstrate its scalability through an experimental analysis.

Differences to conference paper. This paper is an extension of the previous conference version [18], we extend the content in the following way:

- Provide the full proofs for all the theorems and lemmas;
- Extend Sect. 2 to include more background knowledge to make the paper more self-contained;
- Extend Sects. 3 and 5 by (*i*) new subsections consisting of detailed descriptions of how to reduce our synthesis problem settings to LTL synthesis; (*ii*) complexity analysis of our proposed synthesis approaches;
- Extend Sect. 6 by (*i*) a new subsection of detailed description and complete formulation of the benchmarks; (*ii*) a new subsection to present a detailed reduction to LTL synthesis of the studied synthesis problems with the proposed benchmarks;
- Add Sect. 7 for a thorough discussion with related work;
- Add Sect. 8 of conclusion and future work.

2 Preliminaries

2.1 LTL and LTL $_f$

LTL is one of the most popular logics for temporal properties [19]. Given a set of propositions *Prop*, the formulas of LTL are generated as follows:

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

where $a \in Prop$, $\circ$ (*next*) and $\mathcal{U}$ (*until*) are temporal operators. We use common abbreviations, so we have *eventually* as $\Diamond\varphi \equiv true \mathcal{U} \varphi$ and *always* as $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

LTL formulas are interpreted over infinite traces $\pi \in (2^{Prop})^{\omega}$. 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. Given π, we define when an LTL formula φ *holds* at position i, 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 $\pi, i + 1 \models \varphi$;
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists $j \geq i$ such that $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

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

LTL $_f$ is a variant of LTL interpreted over *finite traces* instead of infinite traces [7]. The syntax of LTL $_f$ is exactly the same as the syntax of LTL. We define $\pi, i \models \varphi$, stating that φ holds at position i, as for LTL, except that for the temporal operators we have:

- $\pi, i \models \circ\varphi$ iff $i < \mathsf{lst}(\pi)$ and $\pi, i + 1 \models \varphi$;

- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists j such that $i \leq j \leq \mathsf{lst}(\pi)$ and $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

where we denote the last position (i.e., index) in the finite trace π by $\mathsf{lst}(\pi)$. In addition, we define the *weak next* operator $\bullet$ as abbreviation of $\bullet\varphi \equiv \neg\circ\neg\varphi$. Note that, over finite traces, $\neg\circ\varphi \not\equiv \circ\neg\varphi$, instead $\neg\circ\varphi \equiv \bullet\neg\varphi$. We say that a trace *satisfies* an LTL $_f$ formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

2.1.1 Generalized reactivity(1) formulas

Generalized Reactivity(1) [5], or GR(1), is a fragment of LTL that generalizes fairness ($\Box\Diamond\varphi$) and stability ($\Diamond\Box\varphi$) formulas (cf. [14]). Given a set of propositions *Prop*, a GR(1) formula φ is required to be of the form

$$\varphi = \bigwedge_{i=1}^{m} \Box\Diamond\mathcal{J}_i \rightarrow \bigwedge_{j=1}^{n} \Box\Diamond\mathcal{K}_j$$

where $\mathcal{J}_i$ and $\mathcal{K}_j$ are Boolean formulas over *Prop*.

2.2 Deterministic automata

A *deterministic automaton* (DA, for short) is a tuple $\mathcal{A} = (\Sigma, S, s_0, \delta, \alpha)$, where Σ is a finite alphabet, S is a finite set of states, $s_0 \in S$ is the initial state, $\delta : S \times \Sigma \rightarrow S$ is the transition function, $\alpha \subseteq S^\omega$ is an acceptance condition. Given an infinite word $\pi = a_0 a_1 a_2 \ldots \in \Sigma^\omega$, the *run* of $\mathcal{A}$ on π, denoted by $\mathcal{A}(\pi)$ is the sequence $r = s_0 s_1 s_2 \ldots \in S^\omega$ starting at the initial state s_0 where $s_{i+1} = \delta(s_i, a_i)$. The automaton $\mathcal{A}$ *accepts* the word π if $\mathcal{A}(\pi) \in \alpha$. The *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. In this work we specifically consider reachability, safety, and reachability-safety acceptance conditions:

- *Reachability conditions.* Given a set $T \subseteq S$ of target states, $\mathrm{Reach}(T) = \{s_0 s_1 s_2 \ldots \in S^\omega | \exists k \geq 0 : s_k \in T\}$ requires that a state in T is visited at least once.
- *Safety conditions.* Given a set $T \subseteq S$ of target states, $\mathrm{Safe}(T) = \{s_0 s_1 s_2 \ldots \in S^\omega | \forall k \geq 0 : s_k \in T\}$ requires that only states in T are visited. This is the dual of reachability conditions.
- *Reachability-Safety conditions.* Given two sets $T_1, T_2 \subseteq S$ of target states corresponding to reachability and safety conditions, respectively, $\mathrm{Reach\text{-}Safe}(T_1, T_2) = \{s_0 s_1 s_2 \ldots \in S^\omega | \exists i \geq 0 : s_i \in T_1 \text{ and } \forall j, i \geq j \geq 0 : s_j \in T_2\}$ requires that a state in T_1 is visited at least once, and until then only states in T_2 are visited.

We define the *complement* of a DA $\mathcal{A} = (\Sigma, S, s_0, \delta, \alpha)$ as $\overline{\mathcal{A}} = (\Sigma, S, s_0, \delta, S^\omega \backslash \alpha)$. Note that $\mathcal{L}(\overline{\mathcal{A}}) = \Sigma^\omega \backslash \mathcal{L}(\mathcal{A})$. Note also that $S^\omega \backslash \mathrm{Reach}(T) = \mathrm{Safe}(S \backslash T)$ and $S^\omega \backslash \mathrm{Safe}(T) = \mathrm{Reach}(S \backslash T)$. Therefore, the complement of a DA with a reachability acceptance condition is a DA with a safety acceptance condition, and vice-versa. We also define the *intersection* of two DAs $\mathcal{A}_1 = (\Sigma, S_1, s_1^0, \delta_1, \alpha_1)$ and $\mathcal{A}_2 = (\Sigma, S_2, s_2^0, \delta_2, \alpha_2)$ as $\mathcal{A}_1 \cap \mathcal{A}_2 = (\Sigma, S_1 \times S_2, (s_1^0, s_2^0), \delta', \alpha')$, where $\delta'((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$ and $\alpha' = \{(s_1^0, s_2^0)(s_1^1, s_2^1)(s_1^2, s_2^2) \ldots \in (S_1 \times S_2)^\omega | s_1^0 s_1^1 s_1^2 \ldots \in \alpha_1 \text{ and } s_2^0 s_2^1 s_2^2 \ldots \in \alpha_2\}$. Note that if $\alpha_1 = \text{Safe}(T_1)$ and $\alpha_2 = \text{Safe}(T_2)$, then $\alpha' = \text{Safe}(T_1 \times T_2)$. If $\alpha_1 = \text{Reach}(T_1)$ and $\alpha_2 = \text{Safe}(T_2)$ we define a *bounded intersection*, where $\alpha' = \text{Reach-Safe}(T_1, T_2)$.

2.3 GR(1) games

Following [5], we define a GR(1) *game structure* as a tuple $\mathcal{G} = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \theta_a, \theta_p, \rho_a, \rho_p, \varphi \rangle$ where:

- $\mathcal{V} = \{v_1, \dots, v_k\}$ is a set of Boolean state variables. A state of the game is given by an assignment $s \in 2^{\mathcal{V}}$ of these variables. $\mathcal{I} \subseteq \mathcal{V}$ is the set of input variables controlled by the antagonist. $\mathcal{O} = \mathcal{V} \setminus \mathcal{I}$ is the set of output variables controlled by the protagonist.
- θ_a is a Boolean formula over $\mathcal{I}$ representing the initial states of the antagonist. θ_p is a Boolean formula over $\mathcal{V}$ representing the initial states of the protagonist.
- ρ_a is a Boolean formula over $\mathcal{V} \cup \mathcal{I}'$, where $\mathcal{I}'$ is the set of primed copies of $\mathcal{I}$. This formula represents the transition relation of the antagonist, between a state $s \in 2^{\mathcal{V}}$ and a possible input $s_{\mathcal{I}} \in 2^{\mathcal{I}}$ for the next state.
- ρ_p is a Boolean formula over $\mathcal{V} \cup \mathcal{I}' \cup \mathcal{O}'$, where $\mathcal{O}'$ is the set of primed copies of $\mathcal{O}$. This formula represents the transition relation of the protagonist, relating a pair $(s, s_{\mathcal{I}}) \in 2^{\mathcal{V}} \times 2^{\mathcal{I}}$ of state s and input $s_{\mathcal{I}}$ to an output $s_{\mathcal{O}}$.
- φ is the winning condition for the protagonist given by a GR(1) formula.

We use the terms *antagonist* and *protagonist* instead of *environment* and *agent* to avoid confusion when we switch roles.

2.4 LTL$_f$ synthesis under environment specifications

Let $\mathcal{X}$ and $\mathcal{Y}$ be Boolean variables, with $\mathcal{X}$ controlled by the environment and $\mathcal{Y}$ controlled by the agent. An *agent strategy* is a function $\sigma_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$, and an environment strategy is a function $\sigma_{env} : (2^{\mathcal{Y}})^+ \to 2^{\mathcal{X}}$. A *trace* is a sequence $(X_0 \cup Y_0)(X_1 \cup Y_1) \cdots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$. An agent strategy *induces* a trace $(X_i \cup Y_i)_i$ if $\sigma_{ag}(\epsilon) = Y_0$ and $\sigma_{ag}(X_0 X_1 \cdots X_j) = Y_{j+1}$ for every $j \geq 0$. An environment strategy *induces* a trace $(X_i \cup Y_i)_i$ if $\sigma_{env}(Y_0 Y_1 \cdots Y_j) = X_j$ for every $j \geq 0$. For an agent strategy σ_{ag} and an environment strategy σ_{env} let $\mathsf{play}(\sigma_{ag}, \sigma_{env})$ denote the unique trace induced by both σ_{ag} and σ_{env}, and $\mathsf{play}^k(\sigma_{ag}, \sigma_{env})$ be the finite trace that is a prefix up to k.

Let φ^a_{task} be an LTL$_f$ formula over $\mathcal{X} \cup \mathcal{Y}$. An agent strategy σ_{ag} *realizes* φ^a_{task} if for every environment strategy σ_{env} there exists $k \geq 0$, chosen by the agent, such that the finite trace $\mathsf{play}^k(\sigma_{ag}, \sigma_{env})$ satisfies φ^a_{task}, that is, φ^a_{task} is *agent realizable*.

In standard synthesis the environment is free to choose an arbitrary move at each step, but in AI typically the agent has some knowledge of how the environment works, which it can exploit in order to enforce the goal, specified as an LTL$_f$ formula φ^a_{task}. Here, we specify the environment behaviour by an LTL formula *Env* and call it *environment specification*. In particular, *Env* specifies the set of environment strategies that enforce *Env* [20]. Moreover, we require that *Env* must be *environment realizable*, i.e., the set of environment strategies that enforce *Env* is not empty. Formally, given an LTL formula φ, we say that an environment strategy *enforces* φ, written $\sigma_{env} \rhd \varphi$, if for every agent strategy σ_{ag} we have $\mathsf{play}(\sigma_{ag}, \sigma_{env}) \models \varphi$.

The problem of LTL$_f$ *synthesis under environment specifications* is to find an agent strategy σ_{ag} such that

$$\exists\sigma_{ag}\forall\sigma_{env} \rhd Env : \exists k.\mathsf{play}^k(\sigma_{ag},\sigma_{env}) \models \varphi^a_{task}.$$

As shown in [20], this can be reduced to solving the synthesis problem for the implication $Env \rightarrow LTL(\varphi^a_{task})$, with $LTL(\varphi^a_{task})$ being a suitable LTL $_f$-to-LTL transformation [7], which is 2EXPTIME-complete [3].

3 LTL $_f$ synthesis under GR(1) environment specifications

In this section, we first study LTL $_f$ synthesis under GR(1) environment specifications. Formally, we are interested in solving the following synthesis problem.

Definition 1 (LTL $_f$ synthesis under GR(1) Environment Specifications) The problem is described as a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{Y}, \varphi^e_{GR(1)}, \varphi^a_{task})$, where $\varphi^e_{GR(1)}$ is a GR(1) formula and φ^a_{task} is an LTL $_f$ formula. Realizability of $\mathcal{P}$ checks whether

$$\exists\sigma_{ag}\forall\sigma_{env} : \mathsf{play}(\sigma_{ag},\sigma_{env}) \models \varphi^e_{GR(1)} \rightarrow \exists k.\mathsf{play}^k(\sigma_{ag},\sigma_{env}) \models \varphi^a_{task}.$$

Synthesis of $\mathcal{P}$ computes a strategy σ_{ag} if exists.

Reduction to LTL *synthesis*. A naive approach to solve $\mathcal{P}$ is to reduce it to standard LTL synthesis. Since $\varphi^e_{GR(1)}$ can be naturally considered as an LTL formula, we are able to reduce the problem of $\varphi^e_{GR(1)} \rightarrow \varphi^a_{task}$ by directly reducing to LTL synthesis, applying the reduction in [14, 16]. However, reducing to LTL synthesis has not shown promising results. Hence specific techniques have been proposed that try to avoid, if possible, the Büchi determinization and the solution of parity games, see e.g., [12, 16]. In the next, we will show how to avoid the detour to LTL synthesis for our case.

3.1 Reduction to GR(1) game

To solve the problem $\mathcal{P}$, we first observe that the agent's goal is to satisfy $\neg\varphi^e_{GR(1)} \vee \varphi^a_{task}$, while the environment's goal is to satisfy $\varphi^e_{GR(1)} \wedge \neg\varphi^a_{task}$. Moreover, we know that φ^a_{task} can be represented by a DA with a reachability condition [8]. Then, focusing on the environment point of view, we show that $\mathcal{P}$ can be reduced into a GR(1) game in which the game arena is the complement of the DA for φ^a_{task}, i.e., a DA with safety condition, and $\varphi^e_{GR(1)}$ is the GR(1) winning condition. Since we want a winning strategy for the agent, we need to deal with the complement of the GR(1) game to obtain a winning strategy for the antagonist. More specifically, we can solve the problem by taking the following steps:

1. Translate φ^a_{task} into $\mathcal{A}_{ag} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s^0, \delta, \mathrm{Reach}(T))$ that accepts a trace π iff $\pi \models \varphi^a_{task}$ [8].
2. Complement $\mathcal{A}_{ag}$ into $\overline{\mathcal{A}}_{ag} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s^0, \delta, \mathrm{Safe}(T'))$ with $T' = S\backslash T$. Note that $\overline{\mathcal{A}}_{ag}$ accepts a trace π iff π has no prefix satisfying φ^a_{task}.
3. Define a GR(1) game $\mathcal{G}_{\mathcal{P}}$ with the environment as the protagonist, where the arena is given by $\overline{\mathcal{A}}_{ag}$ and the winning condition is given by $\varphi^e_{GR(1)}$.
4. Solve this game for the antagonist, i.e. the agent.

3.1.1 Building the GR(1) game

We now detail how to build the GR(1) game $\mathcal{G}_{\mathcal{P}}$ (c.f., step 3 above). Given $\overline{\mathcal{A}_{ag}} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s^0, \delta, \mathrm{Safe}(T'))$, we start by encoding the state space S into a logarithmic set of variables $\mathcal{Z}$ (similarly to [21]). In what follows we identify assignments to $\mathcal{Z}$ with states in S, respectively. Given a subset $\mathcal{Y} \subseteq \mathcal{V}$ and a state $s \in 2^{\mathcal{V}}$, we denote by $s\mid_{\mathcal{Y}}$ the projection of s to $\mathcal{Y}$. We then construct the GR(1) game structure $\mathcal{G}_{\mathcal{P}} = \langle \mathcal{V}, \mathcal{I}, \mathcal{O}, \theta_a, \theta_p, \eta_a, \eta_p, \varphi \rangle$ as follows:

- $\mathcal{V} = \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}, \mathcal{I} = \mathcal{Y}, \mathcal{O} = \mathcal{X} \cup \mathcal{Z}$;
- $\theta_a = \top$; θ_p is a formula satisfied by an assignment $s \in 2^{\mathcal{V}}$ iff $s\mid_{\mathcal{Z}} = s^0$;
- $\eta_a = \top$; η_p is a formula satisfied by assignments $s \in 2^{\mathcal{V}}$ and $s' \in 2^{\mathcal{V}'}$ iff $\delta(s\mid_{\mathcal{Z}}, s'\mid_{\mathcal{X}'\cup\mathcal{Y}'}) = s'\mid_{\mathcal{Z}'}, s'\mid_{\mathcal{Z}'} \in T'$;
- $\varphi = \varphi^e_{GR(1)}$.

In the game $\mathcal{G}_{\mathcal{P}}$, the environment takes the role of protagonist, and the agent of antagonist. States in the game are given by assignments of $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$, where the $\mathcal{X}$ and $\mathcal{Y}$ components represent respectively the last assignment of the environment and agent variables chosen by the players, and the $\mathcal{Z}$ component represent the current state of $\overline{\mathcal{A}_{ag}}$. The agent first chooses the $\mathcal{Y}$ component of the next state. There is no restriction on what it can be, so $\theta_a = \eta_a = \top$. Then, the environment chooses the $\mathcal{X}$ component, and based on the chosen assignments assigns the $\mathcal{Z}$ variables as well. θ_p and η_p enforce that the assignment to the $\mathcal{Z}$ variables is consistent with $\overline{\mathcal{A}_{ag}}$, and η_p also enforces that the safety condition Safe(T') is not violated. Note that a play of $\mathcal{G}_{\mathcal{P}}$, given by $\mathsf{play} = \rho_0\rho_1 \ldots \in (2^{\mathcal{V}})^{\omega}$, corresponds to the run $r = (\rho_0\mid_{\mathcal{Z}})(\rho_1\mid_{\mathcal{Z}})(\rho_2\mid_{\mathcal{Z}})\ldots$ of $\overline{\mathcal{A}_{ag}}$ on trace $(\rho_1\mid_{\mathcal{X}\cup\mathcal{Y}})(\rho_2\mid_{\mathcal{X}\cup\mathcal{Y}})\ldots$. Since ρ_0 satisfies θ_p, $\rho_0\mid_{\mathcal{Z}} = s^0$, and since every (ρ_i, ρ_{i+1}) satisfy η_p, $\delta(\rho_i\mid_{\mathcal{Z}}, \rho_{i+1}\mid_{\mathcal{X}\cup\mathcal{Y}}) = \rho_{i+1}\mid_{\mathcal{Z}}$. Therefore, r is a valid run for $\overline{\mathcal{A}_{ag}}$.

Given a play play of $\mathcal{G}_{\mathcal{P}}$, there are two ways the environment can lose in play. The first is by being unable to pick an assignment that satisfies η_p. Since the transition relation δ of $\overline{\mathcal{A}_{ag}}$ is total, this can only happen if $s\mid_{\mathcal{Z}} \notin T'$, meaning that $\overline{\mathcal{A}_{ag}}$ rejects a run visiting $s\mid_{\mathcal{Z}}$. The other is by failing to satisfy $\varphi^e_{GR(1)}$. These correspond to the two ways that the specification can be satisfied: by satisfying φ^a_{task} or by violating the GR(1) environment specification. Therefore, a play satisfies the specification iff it is losing for the protagonist of $\mathcal{G}_{\mathcal{P}}$ (i.e, the environment) and thus wining for the antagonist (i.e., the agent).

3.1.2 Correctness

The correctness of the reduction described above is illustrated by the following theorem.

Theorem 1 *$\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e_{GR(1)}, \varphi^a_{task} \rangle$ is realizable iff the antagonist has a winning strategy in the GR(1) game $\mathcal{G}_{\mathcal{P}}$.*

Proof Let $\overline{\mathcal{A}_{ag}} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s^0, \delta, \mathrm{Safe}(T'))$ be the constructed DA of $\mathcal{P}$. We now prove the theorem in two directions.

$\leftarrow$: If the antagonist has a winning strategy κ_a in $\mathcal{G}_{\mathcal{P}}$, then every play $\rho_0, \rho_1, \ldots \in (2^{\mathcal{V}})^{\omega}$ that is consistent with κ_a is an antagonist winning play. That is to say, either of the following holds:

- protagonist's goal is violated, such that $\varphi^e_{GR(1)}$ does not hold on this play. Therefore, ignoring the initial state of ρ_0, the corresponding sequence $(\rho_1 |_{\mathcal{X}\cup\mathcal{Y}})$, $(\rho_2 |_{\mathcal{X}\cup\mathcal{Y}}), \ldots \in (2^{\{\mathcal{X}\cup\mathcal{Y}\}})^\omega$ violates $\varphi^e_{GR(1)}$. Thus $\varphi^e_{GR(1)} \rightarrow \varphi^a_{task}$ holds.
- There exists k, such that the protagonist's transition constraint η_p is violated, thus $\delta(\rho_k |_{\mathcal{Z}}, \rho_{k+1} |_{\mathcal{X}'\cup\mathcal{Y}'}) \notin T'$. That is to say, the finite trace $(\rho_1 |_{\mathcal{X}\cup\mathcal{Y}}), (\rho_2 |_{\mathcal{X}\cup\mathcal{Y}}), \ldots (\rho_{k+1} |_{\mathcal{X}\cup\mathcal{Y}})$ satisfies φ^a_{task}, then $\varphi^e_{GR(1)} \rightarrow \varphi^a_{task}$ holds.

$\rightarrow$: If $\mathcal{P}$ is realizable, then the agent has a winning strategy σ_{ag}. So for every trace $\pi_0, \pi_1, \ldots \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ that is consistent with σ_{ag}, either of the following holds:

- $\varphi^e_{GR(1)}$ is violated, therefore, starting from the initial state ρ_0, following σ_{ag}, the antagonist is able to force the corresponding play $\rho_0, \rho_1, \rho_2 \ldots \in (2^{\mathcal{V}})^\omega$ to violate the protagonist's goal. Thus this play is winning for the antagonist.
- There exists k such that $\pi = \pi_0, \pi_1, \ldots \pi_k \models \varphi^a_{task}$. Therefore, the corresponding run of π on $\overline{\mathcal{A}}_{ag}$ ends in a state $\notin T'$. That is to say, in the corresponding play ρ, the antagonist is able to force the protagonist to violate its transition constraint η_p, by having the move from ρ_k to ρ_{k+1}. Thus this play is winning for the antagonist.

□

3.1.3 Complexity

We now discuss the computational properties of the synthesis algorithm described above.

Theorem 2 *The algorithm described above algorithm solves the synthesis problem* $\mathcal{P} = (\mathcal{X}, \mathcal{Y}, \varphi^e_{GR(1)}, \varphi^a_{task})$ *in* 2Exptime (*the problem is indeed* 2Exptime*-complete*).

Proof We first prove that the problem is 2Exptime. Specifically, building the corresponding DA with reachability condition for φ^a_{task} takes doubly exponential time in the size of φ^a_{task} [8]. This allows us to exploit subset construction and minimization rather than Büchi determinization, making it more scalable in practice, as for LTL $_f$ synthesis. The complementation takes linear time in the size of the automaton. Finally, building the GR(1) game takes linear time in the size of the automaton and solving the corresponding GR(1) game is quadratic in the size of the game [5].

The hardness is immediate from 2Exptime-completeness of LTL $_f$ synthesis itself [8]. Notice that as a special case of our problem, we have standard LTL $_f$ synthesis by considering trivially *Env* to be *true*. □

4 Introducing safety conditions

Next we introduce safety conditions into the framework. Safety conditions are properties that assert that the behaviors of the environment or the agent always remain within some allowed boundaries. A notable example of safety conditions for the environment are effect specifications in planning domains that describe how the environment can react to agent actions in a given situation. A notable example of safety conditions for the agent are action preconditions, i.e. the agent cannot violate the precondition of actions.

Another notable example of safety conditions for the agent comes from planning with maintenance goals (c.f. [17]). Observe though that there is a difference between the safety conditions on the environment and those on the agent: the first must hold forever, while the second must hold until the agent task is terminated, i.e., the goal is fulfilled.

4.1 Expressing safety as LTL $_f$ formulas

Typically, we capture general safety conditions as LTL formulas that, if invalid, are always violated within a finite number of steps. Alternatively, we can think of them as properties that need to hold for all prefixes of an infinite trace. Under this second view, we can also describe the finite variant of safety by simply requiring that the safety condition holds for all prefixes of a finite trace determined by the LTL $_f$ agent task requirement. This view of safety conditions as properties that must hold for all prefixes also allows us to specify them in LTL $_f$. Indeed, all prefixes are, in fact, finite traces. Formally, in order to use LTL $_f$ formulas to specify safety conditions, we need to define an alternative notion of satisfaction that interprets a formula over *all* prefixes of a trace:

Definition 2 A (finite or infinite) trace π satisfies an LTL $_f$ formula φ on *all prefixes*, denoted $\pi \models_\forall \varphi$, if every non-empty finite prefix of π satisfies φ. That is, $\pi^k = \pi_0\pi_1, \dots, \pi_k \models \varphi$, for every $0 \leq k <| \pi |$.

Next we show that we can specify all possible safety conditions expressible in LTL, i.e., all first-order (logic) safety properties [22], using LTL $_f$ on prefixes.

It is known that every safety property expressible in LTL can be expressed by a formula of the form $\Box\varphi$, where φ is a pure-past formula [22]. Let us denote by PLTL $_f$ these pure-past formulas. For every PLTL $_f$ formula φ, there exists an LTL $_f$ formula φ' such that every finite trace π that satisfies φ (i.e., $\pi, \mathsf{lst}(\pi) \models \varphi$) also satisfies φ' (i.e., $\pi, 0 \models \varphi'$) [23]. As an example of the equivalence between PLTL $_f$ and LTL $_f$, consider the PLTL $_f$ formula $\varphi = ((\neg p)\,\mathcal{S}\,q)$, where $\mathcal{S}$ stands for pure-past connective *Since*. We refer to [23] for more details about PLTL $_f$. An equivalent LTL $_f$ formula is $\varphi' = \Diamond(q \wedge \bullet\Box(\neg p))$. Considering this discussion, we can prove the following result.

Theorem 3 *Every first-order safety property can be expressed as an LTL $_f$ formula on all prefixes, and viceversa.*

Proof It has been shown that every first-order safety property can be expressed by a formula of the form $\Box\varphi$, where φ is PLTL $_f$ (pure-past) formula [22]. From the semantics of $\Box\varphi$ when φ is a pure-past formula, $\pi \models \Box\varphi$ iff every non-empty prefix π' of π satisfies φ. Moreover, for every PLTL $_f$ formula φ, there exists an LTL $_f$ formula φ' such that every finite trace π that satisfies φ (i.e., $\pi, \mathit{last}(\pi) \models \varphi$) also satisfies φ' (i.e., $\pi, 0 \models \varphi'$) [23]. That is to say, $\pi \models \Box\varphi$ happens iff every non-empty prefix π' satisfies φ', which by definition happens iff $\pi \models_\forall \varphi'$.

The other direction is proved by the definition of LTL $_f$ on all prefixes. Indeed, by definition, a trace $\pi \models_\forall \varphi$ if every non-empty prefix of π satisfies φ. Therefore, if $\pi \not\models_\forall \varphi$, then there exists a finite prefix that does not satisfy φ. Then, the LTL $_f$ formula φ on all prefixes expresses a safety property. □

Turning to safety conditions for the agent, we observe that the fact that an LTL $_f$ formula holds for every prefix of a finite trace (in our case the trace satisfying the task of the agent), is expressible in first-order logic on finite traces, and hence directly as an LTL $_f$ formula [7]. Nevertheless, translating an LTL $_f$ formula on all prefixes to an LTL $_f$ formula may require exponential blowups in general.

5 Adding safety into LTL $_f$ synthesis under GR(1) environment specifications

We now enrich our synthesis framework by adding safety conditions, expressed in LTL $_f$, on both the environment and the agent, following the considerations made previously. In this setting, we are interested in solving the synthesis problem defined as follows.

Definition 3 (LTL $_f$ under GR(1) Environment Specifications, adding safety conditions) The problem is described as a tuple $\mathcal{P}' = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$ where $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe} \rangle$, $Goal = \langle \varphi^a_{task}, \varphi^a_{safe} \rangle$, and $\varphi^e_{GR(1)}$ is a GR(1) formula and φ^a_{task}, φ^e_{safe} and φ^a_{safe} are LTL $_f$ formulas. Realizability of $\mathcal{P}$ checks whether

$$\exists \sigma_{ag} \forall \sigma_{env} : \mathsf{play}(\sigma_{ag}, \sigma_{env}) \models \varphi^e_{GR(1)} \text{ and } \mathsf{play}(\sigma_{ag}, \sigma_{env}) \models_\forall \varphi^e_{safe} \rightarrow$$
$$\exists k.\mathsf{play}^k(\sigma_{ag}, \sigma_{env}) \models \varphi^a_{task} \text{ and } \mathsf{play}^k(\sigma_{ag}, \sigma_{env}) \models_\forall \varphi^a_{safe}.$$

Synthesis of $\mathcal{P}$ computes a strategy σ_{ag} if exists.

This class of synthesis problem is able to naturally reflect the structure of many reactive systems in practice. We illustrate this with a relatively simple example representing a three-way handshake used to establish a TCP connection.

Example 1 In this example, the server and client involved in TCP connection are considered as environment and agent, respectively. Let $\mathcal{X} = \{SynAck\}$ and $\mathcal{Y} = \{Syn, Ack\}$.

- The server can only send a SYN-ACK message after the client has sent a SYN message. $\varphi^e_{safe} = \Box\neg Syn \rightarrow \Box\neg SynAck$ Note that, φ^e_{safe} is an LTL $_f$ formula and represents a safety property with respect to all prefixes semantic (see Definition 2).
- If the client keeps sending a SYN message, the server eventually responds with a SYN-ACK message. $\varphi^e_{GR(1)} = \Box\Diamond Syn \rightarrow \Box\Diamond SynAck$
- The client should eventually send an ACK message, establishing the TCP connection. $\varphi^a_{task} = \Diamond Ack$
- The client can only send an ACK message after the server has sent a SYN-ACK message. $\varphi^a_{safe} = \Box\neg SynAck \rightarrow \Box\neg Ack$

5.1 Reduction to LTL synthesis

Notice that there is a critical difference between the safety conditions of the environment and the ones of the agent, where we require the environment safety conditions to hold forever (leading to an infinite trace), and the agent safety conditions to hold until fulfilling the agent task (expressed as an LTL $_f$ formula such that leading to a finite trace). We start with considering the case of adding environment safety conditions only, and then discuss the case of also adding agent safety conditions.

5.1.1 Adding environment safety conditions

Suppose $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe} \rangle$ and *Goal* is an LTL $_f$ formula φ^a_{task}, we are still able to reduce the problem of $\varphi^e_{GR(1)} \wedge \varphi^e_{safe} \rightarrow \varphi^a_{task}$ to LTL synthesis. The reduction here, however, is not as intuitive as the ones studied for LTL $_f$ synthesis [21] or LTL $_f$ synthesis under LTL environment specifications [14, 16]. This is because the *safety* condition φ^e_{safe} is interpreted over all prefixes, and there is no known linear translation from this type of formula to LTL. Nevertheless, it is still possible to reduce $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$ to LTL synthesis. Note that the problem can be rewritten as

$$\varphi^e_{GR(1)} \rightarrow (\neg\varphi^e_{safe} \vee \varphi^a_{task})$$

where $\neg\varphi^e_{safe}$ is interpreted using standard LTL $_f$ semantics. That is to say, an agent strategy $\sigma_{ag} : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ realizes *Goal* under environment specification *Env*, if for every $\pi = \pi_0, \pi_1, \ldots \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ consistent with σ_{ag} it is the case that

- **if** $\pi \models \varphi^e_{GR(1)}$, **then**

  - $\pi \models_\forall \varphi^e_{safe}$ **is violated, or**
  - there exists $k \geq 0$ such that $\pi^k \models \varphi^a_{task}$.

This interpretation allows us to shift environment specification φ^e_{safe} to be part of the agent goal, such that *Goal* is satisfied by π if $\pi \models_\forall \varphi^e_{safe}$ is violated or there exists $k \geq 0$ such that $\pi^k \models \varphi^a_{task}$.

Regarding the first case, note that $\pi \models_\forall \varphi^e_{safe}$ requires that every non-empty finite prefix of π satisfies φ^e_{safe}. That is, $\pi_0, \ldots, \pi_k \models \varphi^e_{safe}$, for every $0 \leq k <| \pi |$. Therefore, $\pi \models_\forall \varphi^e_{safe}$ **is violated** iff there exists a non-empty finite prefix of π that falsifies φ^e_{safe} in the standard LTL $_f$ semantics. That is, $\pi_0, \ldots, \pi_k \models \neg\varphi^e_{safe}$, for some $0 \leq k <| \pi |$, which is equivalent to $\pi \models \neg\varphi^e_{safe}$ in standard LTL $_f$ semantics. Therefore, *Goal* is satisfied by π if there exists $k \geq 0$ such that $\pi^k \models (\neg\varphi^e_{safe}) \vee \varphi^a_{task}$.

Consequently, the original problem of *Goal* under environment specification *Env* is realized by agent strategy $\sigma_{ag} : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$, if for every $\pi = \pi_0, \pi_1, \ldots \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega$ consistent with σ_{ag} it is the case that

- **if** $\pi \models \varphi^e_{GR(1)}$, **then**
- there exists $k \geq 0$ such that $\pi^k \models (\neg\varphi^e_{safe}) \vee \varphi^a_{task}$

Since $\varphi^e_{GR(1)}$ is an LTL formula, this is a standard problem of LTL $_f$ synthesis under LTL environment specifications. Therefore, following the reduction in [14, 16], consider synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$, where $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe} \rangle$ and *Goal* is φ^a_{task}, we are able to reduce it to solving the LTL synthesis problem $\langle \mathcal{X}, \mathcal{Y} \cup \{alive\}, \varphi^e_{GR(1)} \rightarrow LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task}) \rangle$. The operation of $LTL(\varphi)$ takes an LTL $_f$ formula φ as input, applies the LTL $_f$-to-LTL translation introduced in [7], and returns the corresponding LTL formula ψ, that is equa-satisfiable to φ. Moreover, proposition *alive* is introduced by LTL $_f$-to-LTL translation and assigned as agent variable.

The following theorem guarantees the correctness of this reduction.

Theorem 4 *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal\rangle$, where $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe}\rangle$ and Goal is φ^a_{task}, be the defined synthesis problem, $\mathcal{P}' = \langle \mathcal{X}, \mathcal{Y} \cup \{alive\}, \varphi^e_{GR(1)} \to LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})\rangle$ be the reduced LTL synthesis problem. We have $\mathcal{P}$ is realizable if and only if $\mathcal{P}'$ is realizable.*

Proof We prove the two directions separately.

- $\leftarrow$: Since $\mathcal{P}'$ is realizable with respect to $\langle \mathcal{X}, \mathcal{Y} \cup \{alive\}\rangle$, there exists a winning strategy $\sigma'_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}\cup\{alive\}}$ such that every trace π' that is consistent with σ'_{ag} satisfies $\neg\varphi^e_{GR(1)} \vee LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$. Therefore, one of the following conditions holds for π':
  - $\neg\varphi^e_{GR(1)}$ is true, and therefore the environment specification $\varphi^e_{GR(1)}$ is violated. π' thus realizes $\mathcal{P}$ by violating environment specification $\varphi^e_{GR(1)}$.
  - $LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$ is true, and therefore there exists a position k such that either $\pi'_0, \dots, \pi'_k \models \neg\varphi^e_{safe}$ or $\pi'_0, \dots, \pi'_k \models \varphi^a_{task}$. The former happens, then environment specification φ^e_{safe} is violated, and π' thus realizes $\mathcal{P}$ by violating environment specification φ^e_{safe}. The latter happens when agent task φ^a_{task} is accomplished, and π' thus realizes $\mathcal{P}$.

  Finally, in order to obtain the winning strategy σ_{ag}, we have $\sigma_{ag}(\lambda) = \sigma'_{ag}(\lambda)\mid_{\mathcal{Y}}$, where $\lambda \in (2^{\mathcal{X}})^*$.
- $\to$: Since $\mathcal{P}$ is realizable, there exists a winning strategy $\sigma_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ such that every trace π that is consistent with σ_{ag} realizes φ^a_{task} under environment specification Env. Therefore, one of the following conditions holds:
  - The environment behaves in a way that violates $\varphi^e_{GR(1)}$, in which case environment specification $\varphi^e_{GR(1)}$ does not hold. Therefore, π realizes $\mathcal{P}'$ by satisfying $\varphi^e_{GR(1)} \to LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$.
  - The environment behaves such as violating φ^e_{safe}, in which case environment specification φ^e_{safe} does not hold. That is to say, there exists a position k such that $\pi_0, \dots, \pi_k \models \neg\varphi^e_{safe}$. Since *alive* is assigned as an agent variable, we can construct π' such that $\pi'_i = \pi_i \cup \{alive\}$ for $0 \le i \le k$ and $\pi'_i = \pi_i$ for $i > k$. Thus, we have π' such that $\pi' \models LTL(\neg\varphi^e_{safe})$. Therefore, $\pi' \models \varphi^e_{GR(1)} \to LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$ by satisfying the right side of the implication $LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$.
  - *Goal* is accomplished, and therefore φ^a_{task} holds. That is to say, there exists k such that $\pi^k \models \varphi^a_{task}$. We again construct π' similarly as above, and $\pi' \models LTL(\varphi^a_{task})$ holds. Therefore, $\pi' \models \varphi^e_{GR(1)} \to LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$ by satisfying the right side of the implication $LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$.

  Finally, we obtain the agent winning strategy as follows. Given k the position in which $LTL((\neg\varphi^e_{safe}) \vee \varphi^a_{task})$ is satisfied, we have that $\sigma'_{ag}(X_0, \dots, X_i) = \sigma_{ag}(X_0, \dots, X_i) \cup \{alive\}$ for $0 \le i \le k$, and $\sigma'_{ag}(X_0, \dots, X_i) = \sigma_{ag}(X_0, \dots, X_i)$, for $i > k$.

□

5.1.2 Adding environment and agent safety conditions

We now have $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe}\rangle$ and $Goal = \langle \varphi^a_{task}, \varphi^a_{safe}\rangle$, where φ^a_{safe} is expressed in LTL $_f$ on all prefixes. As mentioned above, the agent safety condition requires that the agent remains in desired boundaries until fulfilling the task. In this case, there is no known linear translation from an arbitrary φ^a_{safe} directly to LTL $_f$ or LTL. However, later in the experiments, we show that for specific φ^a_{safe} formulas, one can still obtain an equivalent LTL$_f$ formula ψ by analyzing the properties specified by φ^a_{safe}. Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal\rangle$, where $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe}\rangle$ and $Goal = \langle \varphi^a_{task}, \varphi^a_{safe}\rangle$, be the defined synthesis problem. Suppose there is an LTL$_f$ formula ψ that is equivalent to φ^a_{safe}, we are able to reduce the original problem to $\widehat{\mathcal{P}} = \langle \mathcal{X}, \mathcal{Y}, Env, \widehat{Goal}\rangle$, where $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe}\rangle$, $\widehat{Goal} = \langle \widehat{\varphi^a_{task}}\rangle$, and $\widehat{\varphi^a_{task}} = \varphi^a_{task} \wedge \psi$. This reduction allows us to solve $\mathcal{P}$ by applying the reduction from $\widehat{\mathcal{P}}$ to LTL synthesis described in Sect. 5.1.1.

5.2 Reduction to GR(1) game

We now show that the synthesis problem $\mathcal{P}'$ can be reduced into a GR(1) game $\mathcal{G}_{\mathcal{P}'}$, analogously to the construction of $\mathcal{G}_{\mathcal{P}}$ in Sect. 3. To solve this problem, the first thing to note is that $\varphi^a_{task} \wedge \varphi^a_{safe}$ can be represented by a DA with reachability-safety condition. As we will show later in this section, this DA can then be reduced into one with a pure reachability condition. Now, since the environment's goal is to satisfy $\varphi^e_{GR(1)} \wedge \varphi^e_{safe} \wedge \neg(\varphi^a_{task} \wedge \varphi^a_{safe})$, then we can reduce $\mathcal{P}'$ to solving a GR(1) game whose game arena is the product of the DA for φ^e_{safe} with safety condition and the complement of the DA for $\varphi^a_{task} \wedge \varphi^a_{safe}$ with reachability condition, i.e. a DA with safety condition. Note that in what follows, we consider $\Sigma = 2^{\mathcal{X} \cup \mathcal{Y}}$.

To solve the synthesis problem $\mathcal{P}'$ we proceed as follows:

1. Build the DA $\mathcal{A}^a_t = (\Sigma, S_1, s^0_1, \delta_1, \text{Reach}(T_1))$ of φ^a_{task} [8].
2. Build the DA $\mathcal{A}^a_s = (\Sigma, S_2, s^0_2, \delta_2, \text{Safe}(T_2 \cup \{s^0_2\}))$ that accepts a trace π iff $\pi \models_\forall \varphi^a_{safe}$ (see Sect. 6.3.1).
3. Take the bounded intersection of $\mathcal{A}^a_t$ and $\mathcal{A}^a_s$ into $\mathcal{A}^a_{t\wedge s} = (\Sigma, S_1 \times S_2, (s^0_1, s^0_2), \delta', \text{Reach-Safe}(T_1, T_2 \cup \{s^0_2\})$. Note that $\mathcal{A}^a_{t\wedge s}$ accepts a trace π iff there exists $k \geq 0$ such that $\pi^k \models \varphi^a_{task}$ and $\pi^k \models_\forall \varphi^a_{safe}$.
4. Reduce $\mathcal{A}^a_{t\wedge s}$ to $\mathcal{A}_{ag} = (\Sigma, S_1 \times S_2, (s^0_1, s^0_2), \delta', \text{Reach}(T))$, as described later in this section. We have that $\mathcal{L}(\mathcal{A}^a_{t\wedge s}) = \mathcal{L}(\mathcal{A}_{ag})$.
5. Complement $\mathcal{A}_{ag}$ into $\overline{\mathcal{A}_{ag}} = (\Sigma, S_1 \times S_2, (s^0_1, s^0_2), \delta', \text{Safe}(T'))$ with $T' = (S_1 \times S_2)\backslash T$.
6. Build the DA $\mathcal{A}_{Env} = (\Sigma, Q, q_0, \delta^e, \text{Safe}(R))$ that accepts a trace π iff $\pi \models_\forall \varphi^a_{safe}$.
7. Intersect $\overline{\mathcal{A}_{ag}}$ and $\mathcal{A}_{Env}$ into a DA $\mathcal{B} = (\Sigma, S_1 \times S_2 \times Q, (s^0_1, s^0_2, q_0), \alpha, \text{Safe}(T' \times R))$. Note that $\mathcal{B}$ accepts exactly the safe prefixes for the environment.
8. Define a GR(1) game $\mathcal{G}_{\mathcal{P}'}$ with the environment as the protagonist, where the arena is given by $\mathcal{B}$ and the winning condition is given by $\varphi^e_{GR(1)}$ (see Sect. 3).
9. Solve this game for the antagonist, i.e. the agent.

We now detail the construction at Step 4 above. Let $\mathcal{A} = (\Sigma, S, s_0, \delta, \alpha)$ be a DA with a reachability-safety condition $\alpha = \text{Reach-Safe}(T_1, T_2)$. We describe a reduction to a $\mathcal{A}' = (\Sigma, S, s_0, \delta', \alpha')$ with a reachability condition $\alpha' = \text{Reach}(T)$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. We define the transition relation of $\mathcal{A}'$ as follows:

$$\delta'(s, \sigma) = \begin{cases} \delta(s, \sigma) & \text{if } s \in T_2 \\ s & \text{if } s \notin T_2 \end{cases}$$

Intuitively, the only change we make is to turn all non-safe states (states not in T_2) into sink states. We then define the reachability condition as $\alpha' = \text{Reach}(T_1 \cap T_2)$. Intuitively, we want to reach a goal state (a state in T_1) that is also safe (i.e., it is in T_2). The two automata are indeed equivalent:

Lemma 5 *Let $\mathcal{A}$ and $\mathcal{A}'$ be as above, then $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof ($\supseteq$) Assume $\pi = \pi_0\pi_1 \ldots \in \mathcal{L}(\mathcal{A})$. Then, the run $r = s_0s_1s_2 \ldots = \mathcal{A}(\pi)$ has a prefix r^k that ends in T_1 and for which every state visited (including the last one) is in T_2. Let $r' = \mathcal{A}'(\pi)$ be the corresponding run in $\mathcal{A}'$. We prove that $(r')^k = r^k$ and $(r')^k$ satisfies the reachability condition α'.

To prove that $(r')^k = r^k$, note that both start at s_0 and that every state s_i in r^k is in T_2. Therefore, $\delta'(s_i, \pi_i) = \delta(s_i, \pi_i)$ for every state s_i in r^k, by the definition of δ'. By induction, $(r')^k = r^k$.

To prove that $(r')^k$ satisfies the reachability objective α', we need to prove that $\mathsf{lst}((r')^k) \in T_1 \cap T_2$. But we already know that $\mathsf{lst}(r^k) \in T_1$ and $\mathsf{lst}(r^k) \in T_2$. Since $(r')^k = r^k$, the conclusion follows.

($\subseteq$) Assume $\pi = \pi_0\pi_1 \ldots \in \mathcal{L}(\mathcal{A}')$. Then, the run $r' = s_0s_1s_2 \ldots = \mathcal{A}'(\pi)$ has a prefix $(r')^k$ that ends in $T_1 \cap T_2$. Let $r = \mathcal{A}(\pi)$ be the corresponding run in $\mathcal{A}$. We prove that $r^k = (r')^k$ and r^k satisfies the reachability-safety condition α.

To prove both of these, we first prove that all states in $(r')^k$ are in T_2. By the definition of δ', if there was a state s in $(r')^k$ that was not in T_2, then every following state would also be s, and therefore not in T_2. But we already know that $\mathsf{lst}((r')^k) \in T_1 \cap T_2$, and therefore the last state is in T_2. Thus, all states in $(r')^k$ must be in T_2.

To prove that $r^k = (r')^k$, note that both r^k and $(r')^k$ start at s_0, and that $\delta(s_i, \pi_i) = \delta'(s_i, \pi_i)$ for every state in T_2. Since we have already proved that all states in $(r')^k$ are in T_2, it follows by induction that $r^k = (r')^k$.

To prove that r^k satisfies the reachability-safety condition α, note that $\mathsf{lst}((r')^k) \in T_1 \cap T_2$, and therefore $\mathsf{lst}((r')^k) \in T_1$. We have also proved that all states in $(r')^k$ are in T_2. Since $r^k = (r')^k$, this is enough to prove that r^k satisfies α. □

Hence, we are able to reduce the synthesis problem $\mathcal{P}' = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$ to a GR(1) game as well.

5.3 Correctness

The correctness of the reduction described above is illustrated by the following theorem.

Theorem 6 *$\mathcal{P}' = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$, with $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe} \rangle$ and $Goal = \langle \varphi^a_{task}, \varphi^a_{safe} \rangle$, is realizable iff the antagonist has a winning strategy in the GR(1) game $\mathcal{G}_{\mathcal{P}'}$.*

Proof The proof here naturally follows the proof for Theorem 1. This is because the accomplishing $Goal = \langle \varphi^a_{safe}, \varphi^a_{task} \rangle$ can be reduced to a reachability condition ${\varphi^a_{task}}'$, as stated by Lemma 5. Moreover, violating environment safety condition φ^e_{safe} can be considered as a reachability condition as well, thus we have $\neg\varphi^e_{safe}$. Consequently, following the detailed constructions described in Sect. 5, we can consider the original problem of $(\varphi^e_{GR(1)} \wedge \varphi^e_{safe}) \rightarrow (\varphi^a_{task} \wedge \varphi^a_{safe})$ as $\varphi^e_{GR(1)} \rightarrow (\neg\varphi^e_{safe} \vee {\varphi^a_{task}}')$, where $(\neg\varphi^e_{safe} \vee {\varphi^a_{task}}')$ can be considered as a reachability condition, as in the problem defined in Sect. 3. Therefore, the solution of reducing to a GR(1) game remains correct. □

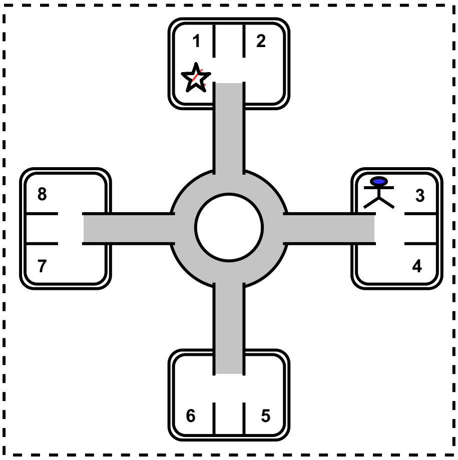


Fig. 1 Illustration of *Finding Nemo with $n = 4$*

5.3.1 Complexity

We study the computational properties of the synthesis technique presented in this section, and analyze the operations required in our synthesis technique in detail.

Theorem 7 *The synthesis problem $\mathcal{P} = (\mathcal{X}, \mathcal{Y}, Env, Goal)$ can be solved with the algorithm described above in* 2EXPTIME (*the problem is indeed* 2EXPTIME*-complete*).

Proof We first prove the membership. Specifically, building the DAs for φ^a_{task} and φ^a_{safe} takes doubly exponential time in the size of φ^a_{task} and φ^a_{safe}, respectively. Second, the computation of the bounded intersection between the two DAs, and the reduction to a DA with reachability condition and complementation is done in linear time in the size of the automaton. Then, building the corresponding DA with safety condition for φ^e_{safe} takes double exponential time in the size of φ^e_{safe}, complementing it and performing the intersection with the DA built previously takes linear time in the size of the automaton. Finally, building the GR(1) game takes linear time in the size of the automaton and solving the corresponding GR(1) game is quadratic in the size of the game.

The hardness is immediate from 2EXPTIME-completeness of LTL $_f$ synthesis itself [8]. Notice that as a special case of our problem, we have standard LTL $_f$ synthesis by considering trivially $\varphi^e_{GR(1)}$ and φ^e_{safe} to be *true*. □

6 Experimental analysis

We implemented the approach described in Sect. 5, which subsumes the method described in Sect. 3, in a tool called GFSYNTH .[1] In this section, we first describe the implementation of GFSYNTH, and then introduce two representative benchmarks that are able to capture

[1] Tool (also benchmarks) available at https://github.com/Shufang-Zhu/GFSynth.

commonly used sensor-based robotic tasks. An empirical evaluation is shown at the end to show the performance of our approach.

6.1 Implementation

GFSynth runs in three steps: automaton construction, reduction to GR(1) game, and GR(1) game solving. In the first step, we use code from the LTL $_f$-synthesis tool Syft [21] to read and parse the input and construct corresponding DAs. All DAs in GFSynth are symbolically represented by Binary Decision Diagrams (BDDs), as in [21], with each explicit state represented as an assignment over a set of state variables. Therefore, each state in the automaton for the bounded intersection of φ^a_{safe} and φ^a_{task} is a concatenation of the state assignments from both automata. In symbolic representation, the transition function is given by a sequence of BDDs, each computing the next assignment for one of the state variables. Analogously, the transition function in the bounded-intersection automaton can be obtained by concatenating the transition functions from both automata. Regarding the reachability-safety to reachability reduction, where the key part is turning all non-safe states into sink states, consider the set of non-safe states represented by BDD B_{unsafe}. For each BDD B_z in the transition function corresponding to state variable z, if the current state is a non-safe state, we restrict B_z to return the same value as in the current state assignment. Otherwise, we don't change the value that B_z is supposed to return. This can be computed as $(\neg B_{unsafe} \wedge B_z) \vee (B_{unsafe} \wedge z)$. The necessary BDD operations are available in the CUDD−3.0.0 [24] BDD library. Finally, we solve the GR(1) game in the input format of the GR(1)-synthesis tool Slugs [25]. To solve and compute a strategy for the antagonist, we call Slugs using the `-CounterStrategy` option.

6.2 Benchmarks

For the experimental evaluation, we use two sets of benchmarks based on examples of reactive synthesis from the literature, slightly modified to adapt them to our framework. Both examples involve an agent navigating around an environment in order to perform a task. In both cases, we can use a parameter n to scale the number of regions, and thus measure how our tool performs as the size of the problem grows.

6.2.1 Finding Nemo

This example is based on the running example from [6]. The agent is a robot that moves in a workspace consisting of a circular hallway with n sections, each of which leads to two rooms with no other exits, adding up to $3n$ regions in total. Figure 1 shows an illustration with $n = 4$.

The agent is searching for "Nemo", who can move around the odd-numbered rooms. The robot has a sensor that detects if Nemo is in the current region the robot is in, and has a camera that it can use to record Nemo if it finds him. The input variable *SenseNemo* indicates whether the sensor has detected Nemo in the current room. The sensor can also detect if Nemo is leaving the region, which is represented by the input variable *NemoLeaving*. The output variables $R = \{Hallway_1, \dots, Hallway_n, Room_1, \dots, Room_{2n}\}$ indicate which region the robot is currently at. The output variable *CameraOn* indicates whether the robot's camera is turned on. We use $T \subseteq R^2$ to denote the transition relation between regions. Specifically, $(r, r') \in T$ iff the robot can move directly from region r to region r'. Following the room layout described above, we define T as follows:

$$\begin{aligned} T = \bigcup_{1 \le i \le n} \{ & (Hallway_i, Hallway_{(i-1) \bmod n}), \\ & (Hallway_i, Hallway_i), \\ & (Hallway_i, Hallway_{(i+1) \bmod n}), \\ & (Hallway_i, Room_{2i-1}), \\ & (Hallway_i, Room_{2i}), \\ & (Room_{2i-1}, Room_{2i-1}), \\ & (Room_{2i-1}, Hallway_i), \\ & (Room_{2i}, Room_{2i}), \\ & (Room_{2i}, Hallway_i)\} \end{aligned}$$

The following are the components of the specification, and note that all the safety conditions are expressed in LTL $_f$ on all prefixes:

- Nemo can only be found in the odd-numbered rooms. (environment safety condition)

$$safe^1_{env} = \Box\left(\left(\bigwedge_{i=1}^{n} \neg Room_{2i-1}\right) \to \neg SenseNemo\right)$$

- Nemo leaves a room if and only if the sensor has detected so in the previous timestep. (environment safety condition)

$$\begin{aligned} safe^2_{env} = \Box((SenseNemo \wedge \bigvee_{i=1}^{2n} \left(Room_i \wedge \circ Room_i\right)) \\ \to (\circ SenseNemo \leftrightarrow \neg NemoLeaving)) \end{aligned}$$

- If Nemo is found in a room, then Nemo will stay in that room for at least one time step after it is found. (environment safety condition)

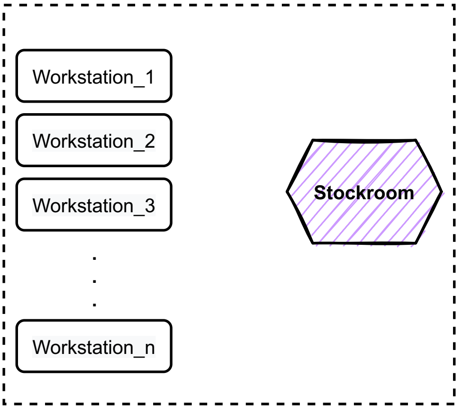
$$\begin{aligned} safe^3_{env} = \Box((\neg SenseNemo \wedge \circ SenseNemo \wedge \\ \bigvee_{i=1}^{2n} \left(\circ Room_i \wedge \circ\circ Room_i\right)) \to \\ \circ\circ SenseNemo) \end{aligned}$$

- If the rooms where Nemo appears are visited infinitely often, then Nemo is found infinitely often. (environment GR(1) condition)

$$GR(1)_{env} = \left(\bigwedge_{i=1}^{n} \Box\Diamond Room_{2i-1}\right) \to \Box\Diamond SenseNemo$$

- Only one region can be visited at a time. (agent safety condition)

Fig. 2 Illustration of *Workstation Resupply*



$$safe^1_{agn} = \Box \bigwedge_{r \in R} \left(r \rightarrow \bigwedge_{r' \neq r} \neg r' \right)$$

- The robot can only move to a region that has a connection to the region it is currently in. (agent safety condition)

$$safe^2_{agn} = \Box \bigwedge_{r \in R} \left(r \rightarrow \bigvee_{(r,r') \in T} \bullet r' \right)$$

- The camera should not be on if Nemo is not present to be recorded. (agent safety condition)

$$safe^3_{agn} = \Box(\neg SenseNemo \rightarrow \neg CameraOn)$$

- The robot should get 3 timesteps worth of footage of Nemo. (agent task)

$$\begin{aligned} task_{agn} = \Diamond(&SenseNemo \wedge CameraOn \wedge \\ &\circ\Diamond(SenseNemo \wedge CameraOn \wedge \\ &\circ\Diamond(SenseNemo \wedge CameraOn))) \end{aligned}$$

6.2.2 Workstation resupply

This example is based on the scenario presented in [26] of a robot responsible for resupplying workstations in a factory with parts from a stockroom, as illustrated in Fig. 2

There are n stations that the robot needs to resupply. In order to resupply a station, the robot must first pick up a part from the stockroom, then bring it to the workstation. A workstation may be occupied, in which case the robot has to wait until it is vacated before going inside. The environment must guarantee that the workstations will be vacated infinitely often and that they won't become occupied when the robot is already inside.

We use a map consisting of $2n+2$ regions. The set of regions R is represented by the output variables *Stockroom*, $Station_1, \dots, Station_n$, *OutsideStockroom*, $OutsideStation_1, \dots, OutsideStation_n$. We define the transition relation T as follows:

$$\begin{aligned}
T = &(Stockroom, OutsideStockroom) \cup \\
&(OutsideStockroom, Stockroom) \cup \\
&\bigcup_{1 \le i \le n} \{(Station_i, OutsideStation_i), \\
&\qquad (OutsideStation_i, Station_i), \\
&\qquad (OutsideStation_i, OutsideStockroom), \\
&\qquad (OutsideStockroom, OutsideStation_i)\} \cup \\
&\bigcup_{1 \le i \le n} \bigcup_{1 \le j \le n} \{(OutsideStation_i, OutsideStation_j)\} \cup \\
&\{(r, r) | r \in R\}
\end{aligned}$$

The output variables also include *PickUpPart* and *Resupply*, representing the actions that the robot can take in the stockroom and workstations, respectively. The input variables are $Occupied_1, \dots, Occupied_n$, which indicate if each workstation is occupied at a given point in time.

The following are the components of the specification, and note that all the safety conditions are expressed in LTL $_f$ on all prefixes:

- A station cannot become occupied after the robot is already inside. (environment safety condition)

$$\begin{aligned}
safe_{env} = \bigwedge_{i=1}^{n} \Box((\neg Occupied_i \wedge \circ Station_i) \rightarrow \\
\neg \circ Occupied_i)
\end{aligned}$$

- Every workstation must be vacated infinitely often. (environment GR(1) condition)

$$GR(1)_{env} = \Box\Diamond\top \rightarrow \left(\bigwedge_{i=1}^{n} \Box\Diamond\neg Occupied_i\right)$$

- Only one region can be visited at a time. (agent safety condition)

$$safe^1_{ag} = \Box \bigwedge_{r \in R} \left(r \rightarrow \bigwedge_{r' \neq r} \neg r'\right)$$

- The robot can only move to a region that has a connection to the region it is currently in. (agent safety condition)

$$safe^2_{ag} = \Box \bigwedge_{r \in R} \left(r \rightarrow \bigvee_{(r, r') \in T} \bullet r'\right)$$

- The robot cannot be in a station if it is occupied. (agent safety condition)

$$safe^3_{ag} = \Box \bigwedge_{i=1}^{n} \left(Occupied_i \rightarrow \neg Station_i\right)$$

- The robot needs to be inside the stockroom to pick up a part. (agent safety condition)

$$safe^4_{ag} = \Box(PickUpPart \rightarrow Stockroom)$$

- The robot needs to be inside a station to resupply. (agent safety condition)

$$safe^5_{ag} = \Box\left(Resupply \rightarrow \bigvee_{i=1}^{n} Station_i\right)$$

- The robot can only resupply a station if it has picked up a part since last resupplying. Intuitively, this also requires that whenever *Resupply* holds, it only holds for one time step. (agent safety condition)[2]

$$\begin{aligned} safe^6_{ag} = {} & \Diamond(last \wedge Resupply) \rightarrow \\ & (\Diamond(PickUpPart \wedge \\ & \quad \neg Resupply\, \mathcal{U}\, (last \wedge Resupply))) \end{aligned}$$

- The robot should resupply all workstations. (agent task)

$$task_{ag} = \bigwedge_{i=1}^{n} \Diamond(Resupply \wedge Station_i)$$

6.3 Empirical evaluation

In this section, we describe in detail the techniques that allow us to compare GFSynth against a state-of-the-art LTL synthesis tool Strix [27], the winner of the LTL-synthesis track of the synthesis competition SYNTCOMP 2020 [28], using it as the baseline of comparison to our tool. Note that since our benchmarks assume that the agent moves first, while Strix assumes the environment moves first, we had to slightly modify the specifications by adding a $\circ$ before all variables controlled by the environment, a transformation that essentially corresponds to ignoring the first move by the environment.

Consider synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal\rangle$, the alternative approach of reducing to LTL synthesis exists regarding different configurations of *Env* and *Goal*, as shown in Sects. 3 and 5. In particular, when considering both GR(1) and environment safety conditions (no agent safety conditions), there exists a linear time reduction to LTL synthesis. However, for the case of adding also agent safety conditions, although the reduction to LTL synthesis works, there is no naive reduction to LTL synthesis. This is because there is no known linear translation from an arbitrary φ^a_{safe} directly to LTL $_f$ or LTL. Nevertheless, in both of benchmarks *Finding Nemo* and *Workstation Resupply*, we are able to obtain an

[2] We use *last* as a shorthand for $\bullet\bot$, denoting the last point in the prefix.

equivalent LTL $_f$ formula by analyzing the properties specified by φ^a_{safe}. We now elaborate their details.

6.3.1 Reduction to LTL synthesis with proposed benchmarks

Suppose $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe} \rangle$ and $Goal = \langle \varphi^a_{task}, \varphi^a_{safe} \rangle$, where φ^a_{safe} is expressed in LTL $_f$ on all prefixes. Note that in both of benchmarks *Finding Nemo* and *Workstation Resupply*, φ^a_{safe} is a conjunction over smaller safety conditions $safe^*_{ag}$. Basically, $safe^*_{ag}$ is specified in one of the following three patterns, where β denotes a propositional formula:

1. $safe^*_{ag} = \Box(\beta)$
2. $safe^*_{ag} = \Box(\beta \rightarrow \bigvee_i \bullet\beta_i)$
3. $safe^*_{ag} = \Diamond(last \wedge Resupply) \rightarrow (\Diamond(PickUpPart\ \wedge \neg Resupply\ \mathcal{U}(last \wedge Resupply)))$

We now describe the corresponding equivalent LTL $_f$ formula $safe^*_{ag}{}'$ that accepts the same language as each pattern of $safe^*_{ag}$:

1. $safe^*_{ag}{}' = \Box(\beta)$ (no changes)
2. $safe^*_{ag}{}' = \Box(\beta \rightarrow \bigvee_i \bullet\beta_i)$ (no changes)
3. $safe^*_{ag}{}' = ((\neg Resupply)\ \mathcal{W}\ PickUpPart) \wedge \Box(Resupply \rightarrow (\bullet((\neg Resupply)\ \mathcal{W}\ PickUpPart)))$

The equivalence between each pattern of $safe^*_{ag}$ of the corresponding LTL $_f$ formula $safe^*_{ag}{}'$ can be shown by comparing the corresponding DAs respectively. It is worth noting that, we only consider finite safety of agent safety conditions. Therefore, we reload the definition of DA in Sect. 2 with finite-trace interpretation. Basically, the *reachability condition* is reloaded as $\text{Reach}(T) = \{s_0 s_1 s_2 \dots s_k \in S^* | s_k \in T\}$ requires that the run of finite trace π on $\mathcal{D}$ ends in T. The *safety condition* is reloaded as $\text{Safe}(T) = \{s_0 s_1 s_2 \dots s_k | \forall 0 \le i \le k : s_i \in T\}$ requires that the run of finite trace π on $\mathcal{D}$ only visits states in T.

We now show how to obtain a DA of an LTL $_f$ formula on all prefixes. Consider formula $safe^*_{ag}$ on all prefixes. In order to obtain a DA $\mathcal{D}$ such that π is accepted by $\mathcal{D}$ iff $\pi \models_\forall safe^*_{ag}$, we proceed as follows:

1. Construct the DA $\mathcal{A} = (\Sigma, S, s_0, \delta, \text{Reach}(T))$ such that π is accepted by $\mathcal{A}$ iff $\pi \models safe^*_{ag}$.
2. We construct $\mathcal{D} = (\Sigma, S, s_0, \delta', \text{Safe}(T))$ based on $\mathcal{A} = (\Sigma, S', s_0, \delta, \text{Reach}(T))$ as follows. Note that a sink state is a non-accepting state that has only one outgoing edge but going back to itself.
   - $S' = \begin{cases} S & \text{if there is a sink state in } \mathcal{A} \\ S \cup \{sink\} & \text{if there is no sink state in } \mathcal{A} \end{cases}$
   - $\delta'(s, \sigma) = \begin{cases} sink & \text{if } \delta(s, \sigma) \notin T \\ \delta(s, \sigma) & \text{if } \delta(s, \sigma) \in T \end{cases}$

Consider an arbitrary trace $\pi \in \Sigma^*$. By the construction of $\mathcal{D}$, π is accepted by $\mathcal{D}$ iff the corresponding run of π only visits T. That is, for every $0 \le k <| \pi |$, the corresponding run of $\pi_0, \dots, \pi_k$ ends in T, and therefore $\pi_0, \dots, \pi_k \models \varphi$. By definition, $\pi \models_\forall safe^*_{ag}$.

The DA $\mathcal{D}'$ of the corresponding LTL $_f$ formula $safe^*_{ag}{}'$ can be achieved directly using existing LTL $_f$-to-DA tools [21]. By checking the equivalence between the DA $\mathcal{D}$ and the DA $\mathcal{D}'$, we can address the equivalence check between LTL $_f$ formula $safe^*_{ag}$ over all prefixes and LTL $_f$ formula $safe^*_{ag}{}'$.

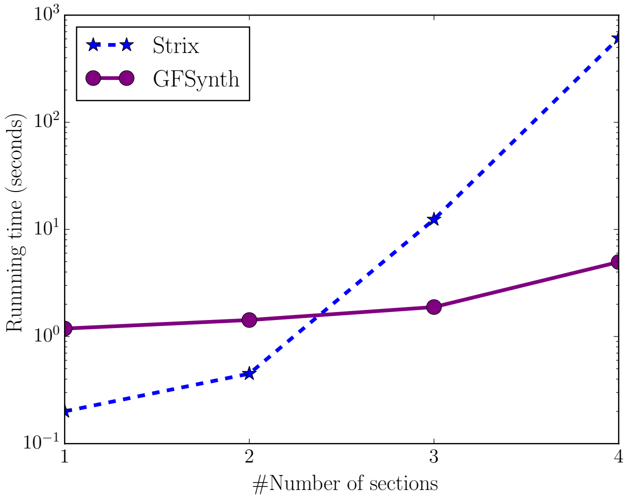


Fig. 3 Benchmark *Finding Nemo*

We now can reduce to LTL synthesis following the subsequent steps described in Sect. 5.1.2.

Baseline and Experiment Setup. All tests were run on a computer cluster. Each test had exclusive access to a node with Intel(R) Xeon(R) CPU E5-2650 v2 processors running at 2.60GHz. Time out was set to two hours (7200 s).

Correctness. Our implementation was verified by comparing the results returned by GFSynth with those from Strix. No inconsistency encountered for the solved cases.

6.3.2 Results

We compared GFSynth against Strix by performing an end-to-end (from specification to winning strategy if realizable) comparison experiment over the benchmarks described in Sect. 6.2. Comparison on both classes of benchmarks show that GFSynth outperforms Strix.

Figures 3 and 4 show the running time of GFSynth and Strix on both benchmarks, respectively. The x-axis indicates the value of the scalable parameter n for each benchmark. The y-axis is in log scale. Results of cases on which both tools failed are not shown. For benchmark *Finding Nemo*, in small cases where $n \leq 2$, there is no large gap in the time cost. However, as n grows, the time cost of GFSynth increases linearly, while the time cost of Strix increases exponentially. Regarding benchmark *Workstation Resupply*, the exponential gap is not so obvious. Nevertheless, as the benchmark grows, Strix almost always takes around 10 times longer than GFSynth. Strix also failed for $n = 5$.

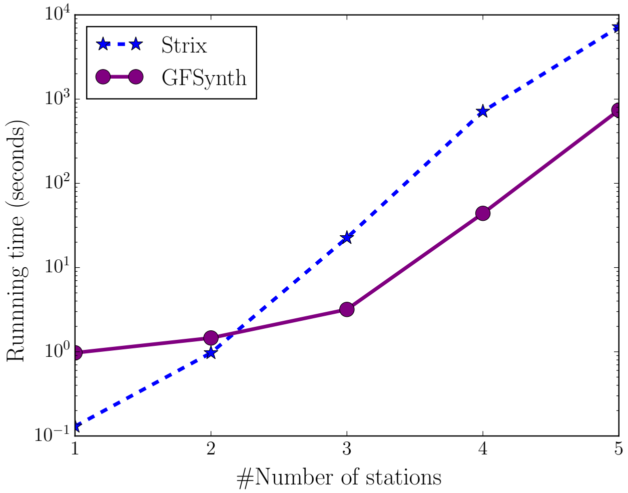


Fig. 4 Benchmark *Workstation Resupply*

7 Related work

There has been extensive studies on the problem of LTL $_f$ synthesis under environment specifications. The paper [29] introduces environment specifications in LTL making the observation that even if the agent goal is expressed in LTL $_f$ the environment specifications should be expressed in LTL on infinite traces, since it is the agent and not the environment that decides when to stop. This observation was already implicit in earlier works [9, 10] on planning for LTL $_f$ goals (the planning domain can be considered as environment safety specifications), indeed in FOND planning the environment fairness specification of FOND is already on infinite traces, but [12] spells this point out in details. [12] also gives a reduction of the LTL $_f$ synthesis under environment specifications problem to LTL synthesis and a specific technique to work with environment safety and co-safety specifications. Interestingly, environment specifications were also studied in [11, 30] and then in [15] under the perspective of seeing environment specifications as sets of environment strategies.

In [31] one of the bases of our technique was developed: create the game arena out of the LTL $_f$ goals, and impose on that arena simple fairness and stability conditions, both subsumed by GR(1). Note, however, that in general the arena built from the LTL $_f$ formula must be combined with the arena coming from the environment specifications, and if expressed in LTL this will lead to solving a parity game, but above all the construction of the arena for the LTL part will require determinization of Büchi automata for which we have no scalable algorithms. This is spelled out in [16].

In this paper, we consider as environment specifications only safety conditions and GR(1) formulas, again avoiding Büchi determinization. GR(1) has been used before in synthesis/planning to specify temporally extended goal on infinite traces, e.g., in [32], or in fact much earlier in [29, 33]. Here instead we are using GR(1) for the environment specifications, while keeping the goal on finite traces in LTL $_f$. In this way we take advantage of the two most successful cases in synthesis, namely GR(1) and LTL $_f$.

8 Conclusion and future work

In this paper, we brought together the two most successful stories in LTL synthesis, GR(1) and LTL $_f$, obtaining a form of reactive synthesis which is highly expressive yet still scalable. More specifically, we studied the problem in the form of LTL $_f$ synthesis under environment specifications, where the environment specification is expressed as a GR(1) formula, and the agent task is specified as an LTL $_f$ formula. Our approach bases on a reduction to GR(1) game, where the game arena only derivates from the LTL $_f$ part, and the game winning condition comes from by the GR(1) part. In particular, the crux of the reduction is that we aim to obtaining a winning strategy of the antagonist of the GR(1) game, which, in fact, leads to solving a dual GR(1) game. Moreover, we enriched the problem setting by allowing safety conditions on both players. To do so, we first presented a different way of specifying safety properties via LTL $_f$ formulas, which is able to cover all first-order expressible safety properties, and provide a natural way of expressing safety properties. We showed that, with additional safety conditions on both players, our synthesis approach of reducing to GR(1) game stills works. Finally, we integrated our approach in a tool GFSynth, and proved its efficiency through empirical evaluations.

Looking deeper into GFSynth, we observed that on those cases where GFSynth fails, the automata can not be constructed by the MONA library employed by Syft for automata construction from LTL $_f$. There have been various studies on LTL $_f$-to-automata translation. Possibly the most successful attempt is the compositional approach presented in [34, 35]. For future work, we will take this approach into account to improve GFSynth.

Acknowledgements This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), NSF grants IIS-1527668, CCF-1704883, IIS-1830549, and an award from the Maryland Procurement Office. We thank the anonymous reviewer for valuable feedback on this paper.

References

1. Finkbeiner B (2016) Synthesis of reactive systems. Depend Softw Syst Eng 45:72–98
2. Ehlers R, Lafortune S, Tripakis S, Vardi MY (2017) Supervisory control and reactive synthesis: a comparative introduction. Discrete Event Dyn Syst 27(2):209–260
3. Pnueli A, Rosner R (1989) On the synthesis of a reactive module. In: POPL, pp 179–190
4. Kupferman O (2012) Recent challenges and ideas in temporal synthesis. In: SOFSEM 2012, pp 88–98
5. Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'ar Y (2012) Synthesis of Reactive(1) designs. J Comput Syst Sci 78:911–938
6. Kress-Gazit H, Fainekos GE, Pappas GJ (2009) Temporal-logic-based reactive mission and motion planning. IEEE Trans Robot 25(6):1370–1381
7. De Giacomo G, Vardi MY (2013) Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp 854–860
8. De Giacomo G, Vardi MY (2015) Synthesis for LTL and LDL on finite traces. In: IJCAI

9. Camacho A, Triantafillou E, Muise C, Baier JA, McIlraith S (2017) Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In: AAAI, pp 3716–3724
10. De Giacomo G, Rubin S (2018) Automata-theoretic foundations of fond planning for LTL$_f$/LDL$_f$ goals. In: IJCAI, pp 4729–4735
11. Aminof B, Giacomo GD, Murano A, Rubin S (2018) Planning and synthesis under assumptions. CoRR
12. Camacho A, Bienvenu M, McIlraith SA (2018) Finite LTL synthesis with environment assumptions and quality measures. In: KR, pp 454–463
13. He K, Wells AM, Kavraki LE, Vardi MY (2019) Efficient symbolic reactive synthesis for finite-horizon tasks. In: ICRA, pp 8993–8999
14. Zhu S, Giacomo GD, Pu G, Vardi MY (2020) LTL$_f$ synthesis with fairness and stability assumptions. In: AAAI, pp 3088–3095
15. Aminof B, De Giacomo G, Murano A, Rubin S (2019) Planning under LTL environment specifications. In: ICAPS, pp 31–39
16. De Giacomo G, Di Stasio A, Vardi MY, Zhu S (2020) Two-stage technique for LTL$_f$ synthesis under LTL assumptions. In: KR, pp 304–314
17. Ghallab M, Nau DS, Traverso P (2004) Automated planning - theory and practice
18. De Giacomo G, Di Stasio A, Tabajara LM, Vardi MY, Zhu S (2021) Finite-trace and generalized-reactivity specifications in temporal synthesis. In: IJCAI, pp 1852–1858
19. Pnueli A (1977) The temporal logic of programs. In: FOCS, pp 46–57
20. Aminof B, De Giacomo G, Murano A, Rubin S (2019) Planning under LTL environment specifications. In: ICAPS, pp 31–39
21. Zhu S, Tabajara LM, Li J, Pu G, Vardi MY (2017) Symbolic LTL$_f$ synthesis. In: IJCAI, pp 1362–1369
22. Lichtenstein O, Pnueli A, Zuck LD (1985) The glory of the past. In: Logic of programs, pp 196–218
23. De Giacomo G, Di Stasio A, Fuggitti F, Rubin S (2020) Pure-past linear temporal and dynamic logic on finite traces. In: IJCAI 2020, pp 4959–4965
24. Somenzi F (2016) CUDD: CU decision diagram package 3.0.0. University of Colorado at Boulder
25. Ehlers R, Raman V (2016) Slugs: extensible GR(1) synthesis, CAV. Lect Notes Comput Sci 9780:333–339
26. DeCastro JA, Ehlers R, Rungger M, Balkan A, Tabuada P, Kress-Gazit H (2014) Dynamics-based reactive synthesis and automated revisions for high-level robot control. CoRR arXiv:1410.6375
27. Meyer PJ, Sickert S, Luttenberger M (2018) Strix: explicit reactive synthesis strikes back! In: CAV, pp 578–586
28. Jacobs S, Perez GA (2020) 7th reactive synthesis competition: SYNTCOMP 2020. http://www.syntcomp.org/syntcomp-2020-results/
29. Sardiña S, Giacomo GD (2008) Realizing multiple autonomous agents through scheduling of shared devices. In: ICAPS 2008, pp 304–312
30. Aminof B, Giacomo GD, Murano A, Rubin S (2018) Synthesis under assumptions. In: KR, pp 615–616
31. Zhu S, De Giacomo G, Pu G, Vardi M (2020) LTL$_f$ synthesis with fairness and stability assumptions. In: AAAI, pp 3088–3095
32. Camacho A, Bienvenu M, McIlraith SA (2019) Towards a unified view of AI planning and reactive synthesis. In: ICAPS, pp 58–67
33. De Giacomo G, Felli P, Patrizi F, Sardiña S (2010) Two-player game structures for generalized planning and agent composition. In: AAAI, pp 297–302
34. Bansal S, Li Y, Tabajara LM, Vardi MY (2020) Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In: AAAI, pp 9766–9774
35. De Giacomo G, Favorito M (2021) Compositional approach to translate LTL$_f$/LDL$_f$ into deterministic finite automata. In: ICAPS 2021, pp 122–130