Finite-Trace and Generalized-Reactivity Specifications in Temporal Synthesis

Giuseppe De Giacomo[1], **Antonio Di Stasio**[1], **Lucas M. Tabajara**[2], **Moshe Vardi**[2] and **Shufang Zhu**[1*]

[1] Sapienza University of Rome, Rome, Italy
[2] Rice University, Houston, US

{degiacomo, distasio, zhu}@diag.uniroma1.it, vardi@cs.rice.edu, l.martinelli.tabajara@gmail.com

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) assumption. 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.

1 Introduction

Program synthesis is considered the culmination of the ideal of declarative programming [Finkbeiner, 2016; Ehlers *et al.*, 2017]. 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 [Pnueli and Rosner, 1989] 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 [Kupferman, 2012].

*Corresponding Author

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 [Bloem *et al.*, 2012]: 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 [Kress-Gazit *et al.*, 2009]. 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) [De Giacomo and Vardi, 2013] as the specification language. In this approach [De Giacomo and Vardi, 2015], we specify the agent's goal in LTL_f, together possibly with some assumptions on the environment, such as safety conditions, possibly specified as a nondeterministic planning domains [Camacho *et al.*, 2017; De Giacomo and Rubin, 2018; Aminof *et al.*, 2018; Camacho *et al.*, 2018; He *et al.*, 2019], or simple fairness and stability conditions (both special cases of GR(1) fairness) [Zhu *et al.*, 2020]. There are also studies in which general LTL assumptions are used for LTL_f goals, but in this case the difficulties of handling LTL can indeed manifest [Camacho *et al.*, 2018; Aminof *et al.*, 2019; De Giacomo *et al.*, 2020b]. Since LTL_f is a fragment of LTL, as shown in [De Giacomo and Vardi, 2013], the problem of LTL_f synthesis under LTL assumptions can be reduced to LTL synthesis, as, e.g., explicitly pointed out by [Camacho *et al.*, 2018]. However, LTL synthesis algorithms do not scale well due to the difficulty of Büchi automata determinization, see e.g., [Finkbeiner, 2016].

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) assumption. 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) assumption. 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 fulfils its LTL_f goal, i.e., within a finite horizon, something that makes them similar to "maintenance goals" in Planning [Ghallab *et al.*, 2004]. 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.

2 Preliminaries

LTL and LTL_f. LTL is one of the most popular logics for temporal properties [Pnueli, 1977]. Given a set of propositions $Prop$, the formulas of LTL are generated as follows:

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

where $a \in Prop$. 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 $\pi, i \models \varphi_2$;
- $\pi, i \models \bigcirc\varphi$ 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 [De Giacomo and Vardi, 2013]. The syntax of LTL_f is exactly the same to 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:

- $\pi, i \models \bigcirc\varphi$ iff $i < last(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff there exists j such that $i \leq j \leq last(\pi)$ 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 $last(\pi)$. In addition we define the *weak next* operator $\bullet$ as abbreviation of $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$. Note that, over finite traces, $\neg\bigcirc\varphi \not\equiv \bigcirc\neg\varphi$, instead $\neg\bigcirc\varphi \equiv \bullet\neg\varphi$. We say that a trace *satisfies* an LTL_f formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

Generalized Reactivity(1) formulas. Generalized Reactivity(1) [Piterman *et al.*, 2006], or GR(1), is a fragment of LTL that generalizes fairness ($\Box\Diamond\varphi$) and stability ($\Diamond\Box\varphi$) formulas (cf. [Zhu *et al.*, 2020]). 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 $Prop$.

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 $w = a_0a_1a_2\ldots \in \Sigma^\omega$, the *run* of $\mathcal{A}$ on w, denoted by $\mathcal{A}(w)$ is the sequence $r = s_0s_1s_2\ldots \in S^\omega$ starting at the initial state s_0 where $s_{i+1} = \delta(s_i, a_i)$. The automaton $\mathcal{A}$ *accepts* the word w 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, $Reach(T) = \{s_0s_1s_2\ldots \in S^\omega \mid \exists k \geq 0 : s_k \in T\}$ requires that a state in T is visited at least once.

Safety conditions. Given a set $T \subseteq S$ of target states, $Safe(T) = \{s_0s_1s_2\ldots \in S^\omega \mid \forall k \geq 0 : s_k \in T\}$ 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, $Reach - Safe(T_1, T_2) = \{s_0s_1s_2\ldots \in S^\omega \mid \exists i \geq 0 : s_i \in T_1$ 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 $S^\omega \backslash Reach(T) = Safe(S \backslash T)$ and $S^\omega \backslash Safe(T) = Reach(S \setminus T)$. 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 $\alpha' = \{(s_1^0, s_2^0)(s_1^1, s_2^1)(s_1^2, s_2^2)\ldots \in (S_1 \times S_2)^\omega \mid s_1^0s_1^1s_1^2\ldots \in \alpha_1$ and $s_2^0s_2^1s_2^2\ldots \in \alpha_2\}$. Note that if $\alpha_1 = Safe(T_1)$ and $\alpha_2 = Safe(T_2)$, then $\alpha' = Safe(T_1 \times T_2)$. If $\alpha_1 = Reach(T_1)$ and $\alpha_2 = Safe(T_2)$ we define a *bounded intersection*, where $\alpha' = Reach - Safe(T_1, T_2)$.

GR(1) Games. Following [Piterman *et al.*, 2006], 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:

$\mathcal{V} = \{v_1, \ldots, v_k\}$ 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. $\mathcal{O} = \mathcal{V} \backslash \mathcal{I}$ 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.

3 LTL$_f$ Synthesis under GR(1) Assumptions

In this section, we first study LTL$_f$ synthesis under assumptions, i.e., assuming that the behaviour of the environment is forced to satisfy certain restrictions, specified as GR(1) formulas. Formally, we are interested in solving synthesis for $\varphi^e_{GR(1)} \rightarrow \varphi^a_{task}$ [1] ,where φ^a_{task} is an LTL$_f$ formula specifying the agent task, and $\varphi^e_{GR(1)}$ is a GR(1) formula that expresses restrictions on the environment behaviour.

Definition 1 (LTL$_f$ Synthesis under GR(1) Assumptions)**.**

1. *The problem is described as a tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e_{GR(1)}, \varphi^a_{task} \rangle$, where $\mathcal{X}$ and $\mathcal{Y}$ are two disjoint sets of Boolean variables, controlled respectively by the environment and the agent, $\varphi^e_{GR(1)}$ is a* GR(1) *formula, and φ^a_{task} in an* LTL$_f$ *formula.*
2. *An agent strategy $\sigma_{ag} : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ realizes φ^a_{task} under assumption $\varphi^e_{GR(1)}$ if for every $\pi = \pi_0, \pi_1, \ldots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ consistent with σ_{ag} such that $\pi \models \varphi^e_{GR(1)}$, there exists $k \geq 0$ such that $\pi^k = \pi_0, \ldots, \pi_k \models \varphi^a_{task}$.*
3. *Solving $\mathcal{P}$ consists in finding an agent strategy that realizes φ^a_{task} under assumption $\varphi^e_{GR(1)}$.*

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 acceptance condition [De Giacomo and Vardi, 2015]. 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 $\mathcal{A}_{ag} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s^0, \delta, Reach(T))$ that accepts a trace π iff $\pi \models \varphi^a_{task}$.
2. Complement $\mathcal{A}_{ag}$ into $\overline{\mathcal{A}_{ag}} = (\Sigma, S, s^0, \delta, Safe(T'))$ with $T' = S \setminus T$, and $\Sigma = 2^{\mathcal{X} \cup \mathcal{Y}}$. 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.

[1]We refer to [Aminof *et al.*, 2019] for a deep discussion on this.

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 $\overline{\mathcal{A}_{ag}} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s^0, \delta, Safe(T'))$, we start by encoding the state space S into a logarithmic set of variables $\mathcal{Z}$ (similarly to [Zhu *et al.*, 2017b]). 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 $s|_{\mathcal{Y}}$ 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 $s|_{\mathcal{Z}} = s^0$;
- $\eta_a = \top$; η_p is a formula satisfied by assignments $s \in 2^{\mathcal{V}}$ and $s' \in 2^{\mathcal{V}'}$ iff $\delta(s|_{\mathcal{Z}}, s'|_{\mathcal{X}' \cup \mathcal{Y}'}) = s'|_{\mathcal{Z}'}, s'|_{\mathcal{Z}'} \in T'$;
- $\varphi = \varphi^e_{GR(1)}$.

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 $\rho = \rho_0\rho_1 \ldots \in (2^{\mathcal{V}})^\omega$, corresponds to the run $r = (\rho_0|_{\mathcal{Z}})(\rho_1|_{\mathcal{Z}})(\rho_2|_{\mathcal{Z}}) \ldots$ of $\overline{\mathcal{A}_{ag}}$ on trace $(\rho_1|_{\mathcal{X} \cup \mathcal{Y}})(\rho_2|_{\mathcal{X} \cup \mathcal{Y}}) \ldots$. Since ρ_0 satisfies θ_p, $\rho_0|_{\mathcal{Z}} = s^0$, and since every (ρ_i, ρ_{i+1}) satisfy η_p, $\delta(\rho_i|_{\mathcal{Z}}, \rho_{i+1}|_{\mathcal{X} \cup \mathcal{Y}}) = \rho_{i+1}|_{\mathcal{Z}}$. Therefore, r is a valid run for $\overline{\mathcal{A}_{ag}}$.

Given a play ρ of $\mathcal{G}_{\mathcal{P}}$, there are two ways the environment can lose in ρ. 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 $s|_{\mathcal{Z}} \notin T'$, meaning that $\overline{\mathcal{A}_{ag}}$ rejects a run visiting $s|_{\mathcal{Z}}$. 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) assumption. 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).

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 $\mathcal{G}_{\mathcal{P}}$.*

We observe that an alternative approach to the problem of LTL$_f$ synthesis under GR(1) assumptions can be obtained by a reduction to standard LTL synthesis, as $\varphi^e_{GR(1)}$ is a GR(1) formula (and therefore already in LTL), and φ^a_{task} is an LTL$_f$ formula, which can be linearly translated into LTL [De Giacomo and Vardi, 2013; Zhu *et al.*, 2020].

4 Introducing Safety Conditions

Next we introduce safety conditions into the framework. Safety conditions are properties that assert that the behavior of the environment or the agent always remains 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 coming from planning are maintenance goals (c.f. [Ghallab *et al.*, 2004]). 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.

Typically we capture general safety conditions as LTL formulas that, if invalid, are always violated within a finite number of steps.[2] 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 indeed 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, $\pi^k = \pi_1, \ldots, \pi_k \models \varphi$, for every $1 \leq k \leq |\pi|$.*

Next we show that we can specify all possible safety conditions expressible in LTL, i.e., all first-order (logic) safety properties [Lichtenstein *et al.*, 1985], using LTL_f on prefixes.

Theorem 2. *Every first-order safety property can be expressed as an* LTL_f *formula on all prefixes.*

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 [Lichtenstein *et al.*, 1985]. 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., $\pi, last(\pi) \models \varphi$) also satisfies φ' (i.e., $\pi, 0 \models \varphi'$) [De Giacomo *et al.*, 2020a]. That is to say, $\pi \models \Box\varphi$ happens iff every non-empty prefix π' satisfies φ', which by definition happens iff $\pi \models_\forall \varphi'$. □

Turning to safety conditions for the agent, we observe that the fact that an LTL_f formula holds for every prefix of an 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 [De Giacomo and Vardi, 2013]. Nevertheless, translating an LTL_f formula on all prefixes to an LTL_f formula may require exponential blowups in general.

[2]*Safety* LTL is a syntactic fragment of LTL often used to specify safety conditions. It is, nevertheless, open whether it is able to capture all safety conditions expressible in LTL [Zhu *et al.*, 2017a].

5 Adding Safety into LTL_f Synthesis under GR(1) Assumptions

We now enrich our synthesis framework by adding safety assumptions, expressed in LTL_f, both on the environment and on the agent, following the considerations made previously. In this setting, we are interested in solving the synthesis for:

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

where $\varphi^e_{GR(1)}$ and φ^e_{safe} are, respectively, a GR(1) formula and an LTL_f formula expressing safety conditions, and φ^a_{task} and φ^a_{safe} are LTL_f formulas that express the agent task and the safety conditions, respectively.

The problem that we aim to solve is defined as follows.

Definition 3 (LTL_f under assumptions GR(1) assumptions, adding safety conditions)**.**

1. *The problem is described as a tuple $\mathcal{P}' = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$, where $\mathcal{X}$ and $\mathcal{Y}$ are two disjoint sets of Boolean variables, controlled respectively by the environment and the agent, $Env = \langle \varphi^e_{GR(1)}, \varphi^e_{safe} \rangle$, and $Goal = \langle \varphi^a_{task}, \varphi^a_{safe} \rangle$, where $\varphi^e_{GR(1)}$ is a* GR(1) *formula and φ^a_{task}, φ^e_{safe} and φ^a_{safe} are* LTL_f *formulas.*
2. *An agent strategy $\sigma_{ag} : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ realizes Goal under assumption Env if for every $\pi = \pi_0, \pi_1, \ldots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ consistent with σ_{ag} s.t. $\pi \models \varphi^e_{GR(1)}$ and $\pi \models_\forall \varphi^e_{safe}$, then there exists $k \geq 0$ s.t. $\pi^k \models \varphi^a_{task}$ and $\pi^k \models_\forall \varphi^a_{safe}$.*
3. *Solving $\mathcal{P}'$ consists in finding an agent strategy that realizes Goal under assumption Env.*

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.*
  $\varphi^e_{safe} = \Box\neg Syn \rightarrow \Box\neg SynAck$
- *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.*
  $\varphi^a_{safe} = \Box\neg SynAck \rightarrow \Box\neg Ack$

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 Section 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 $\mathcal{A}^a_t = (\Sigma, S_1, s^0_1, \delta_1, Reach(T_1))$ of φ^a_{task}.
2. Build the DA $\mathcal{A}^a_s = (\Sigma, S_2, s^0_2, \delta_2, Safe(T_2 \cup \{s^0_2\}))$ that accepts a trace π iff $\pi \models_\forall \varphi^a_{safe}$.
3. Take the bounded intersection of $\mathcal{A}^a_t$ and $\mathcal{A}^a_s$ into $\mathcal{A}^a_{t\wedge s} = (\Sigma, S_1 \times S_2, (s^0_1, s^0_2), \delta', Reach{-}Safe(T_1, T_2 \cup \{s^0_2\})$. 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 $\mathcal{A}_{ag} = (\Sigma, S_1 \times S_2, (s^0_1, s^0_2), \delta', Reach(T))$, 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 $\overline{\mathcal{A}_{ag}} = (\Sigma, S_1 \times S_2, (s^0_1, s^0_2), \delta', Safe(T'))$ with $T' = (S_1 \times S_2) \setminus T$.
6. Build the DA $\mathcal{A}_{env} = (\Sigma, Q, q_0, \delta^e, Safe(R))$ that accepts a trace π iff $\pi \models_\forall \varphi^e_{safe}$.
7. Intersect $\overline{\mathcal{A}_{ag}}$ and $\mathcal{A}_{env}$ into a DA $\mathcal{B} = (\Sigma, S_1 \times S_2 \times Q, (s^0_1, s^0_2, q_0), \alpha, Safe(T' \times R))$. 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 Section 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 $\alpha = Reach - Safe(T_1, T_2)$. We describe a reduction to a $\mathcal{A}' = (\Sigma, S, s_0, \delta', \alpha')$ with a reachability condition $\alpha' = Reach(T)$ 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 $\alpha' = Reach(T_1 \cap T_2)$. 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 1. *Let $\mathcal{A}$ and $\mathcal{A}'$ be as above, then $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof. (Sketch) The idea is that since all unsafe states (states not in T_2) are converted to sinks, a run that reaches an unsafe state always gets stuck there, and therefore never reaches the accepting states $T_1 \cup T_2$. □

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

Theorem 3. *$\mathcal{P}' = \langle \mathcal{X}, \mathcal{Y}, Env, Goal\rangle$, 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. (Sketch) Follows from Theorem 1 and Lemma 1. □

6 Experimental Analysis

We implemented the approach described in Section 5, which subsumes the method described in Section 3, in a tool called GFSYNTH. In this section, we first describe the implementation of GFSYNTH, and then introduce two representative benchmarks that are able to capture 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 [Zhu *et al.*, 2017b] to read and parse the input and construct corresponding DAs. We then perform the reduction to a GR(1) game following the steps in Section 5. Since all DAs are symbolically represented by Binary Decision Diagrams (BDDs), as in [Zhu *et al.*, 2017b], we make use of the BDD library CUDD-3.0.0 [Somenzi, 2016] to implement operations such as bounded intersection, the reduction from reachability-safety to reachability and the final reduction to a GR(1) game. Finally we save the GR(1) game in the input format of the GR(1)-synthesis tool SLUGS [Ehlers and Raman, 2016]. 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.

Finding Nemo. Based on the running example from [Kress-Gazit *et al.*, 2009]. The agent is a robot that moves in a workspace consisting of a circular hallway with n sections, each leading into two different rooms. The agent is looking for "Nemo", who can appear in any of the odd-numbered rooms. The agent has a camera and its task is to record three timesteps worth of footage of Nemo.

Workstation Resupply. Based on the scenario presented in [DeCastro *et al.*, 2014] of a robot responsible for resupplying workstations in a factory with parts from a stockroom. The robot's task is to resupply n separate stations, something it can only do after picking up a part from the stockroom. A workstation may be occupied, in which case the robot has to wait until it is vacated before going inside.

The agent safety conditions ensure movement constraints and other domain requirements, for example that the agent has picked up a part in the stockroom before resupplying a station. The environment safety conditions guarantee for example that the sensors are well-behaved and that stations don't become occupied while the agent is inside. The GR(1) condition in *Finding Nemo* guarantees that if the robot visits the odd-numbered rooms infinitely often, it will find Nemo infinitely often, while in *Workstation Resupply* it guarantees that each workstation will be vacated infinitely often.

6.3 Empirical Evaluation

Comparing to LTL Synthesis. We want to compare GFSYNTH to a state-of-the-art LTL synthesis tool. In Section 3 we pointed out how $\varphi^e_{GR(1)}$ and φ^a_{task} can be translated to LTL. We can handle the environment safety condition φ^e_{safe} by observing that $(\varphi^e_{GR(1)} \land \varphi^e_{safe}) \to \varphi^a_{task}$ (with φ^e_{safe} interpreted on all prefixes) is equivalent to $\varphi^e_{GR(1)} \to {\varphi^a_{task}}'$ with ${\varphi^a_{task}}' = (\neg\varphi^e_{safe} \lor \varphi^a_{task})$ interpreted using standard LTL$_f$ semantics. Indeed, the agent can violate the environment safety condition by producing a single prefix that violates φ^e_{safe}. Then, $(\neg\varphi^e_{safe} \lor \varphi^a_{task})$ can be interpreted as a single LTL$_f$ formula and reduced to LTL. Handling the agent safety condition φ^a_{safe}, however, cannot be done easily, since, as discussed in Section 4, there is no known way to translate an LTL$_f$ formula on all prefixes to LTL$_f$ and LTL without exponential blowups. Hence, in order to compare GFSYNTH with tools for LTL synthesis, we manually translated the specific φ^a_{safe} of our benchmarks above to an equivalent LTL$_f$ formula to be included directly as a new conjunct in φ^a_{task}.

We then converted the entire specification to LTL and synthesized it with STRIX [Meyer *et al.*, 2018], the winner of the LTL-synthesis track of the synthesis competition SYNTCOMP 2020 [Jacobs and Perez, 2020], 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 $\bigcirc$ before all variables controlled by the environment, a transformation that essentially corresponds to ignoring the first move by the environment.

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 seconds).

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

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 Section 6.2. Comparison on both classes of benchmarks show that GFSYNTH outperforms STRIX.

Figure 1 and Figure 2 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

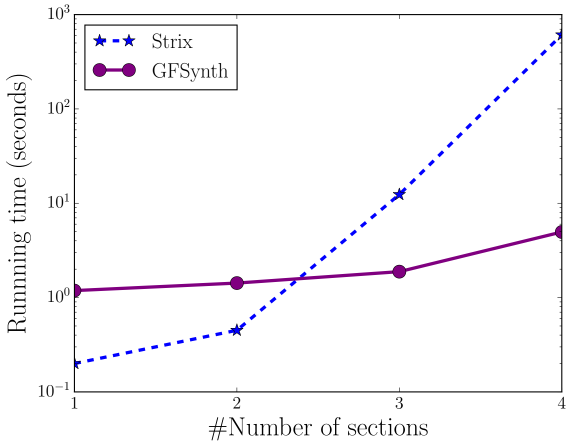

Figure 1: Benchmark *Finding Nemo*

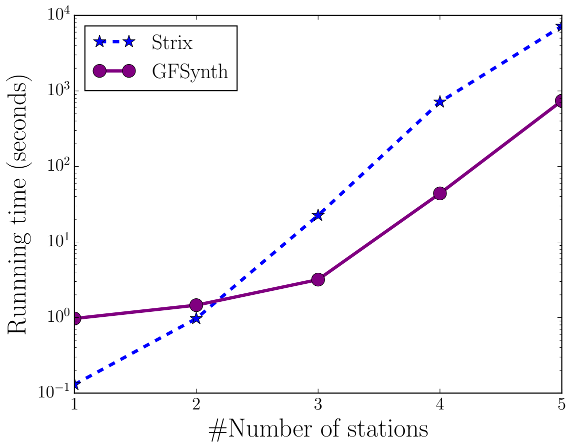

Figure 2: Benchmark *Workstation Resupply*

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 $n = 5$.

Discussions. 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 has been various studies on LTL$_f$-to-automata translation. Possibly the most successful attempt is the decompositional approach presented in [Bansal *et al.*, 2020]. For future work, we will take this approach into account to improve GFSYNTH.

Acknowledgments

Research 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.

References

[Aminof *et al.*, 2018] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning and synthesis under assumptions. *CoRR*, 2018.

[Aminof *et al.*, 2019] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Planning under LTL environment specifications. In *ICAPS*, 2019.

[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.

[Bloem *et al.*, 2012] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[Camacho *et al.*, 2017] Alberto Camacho, Eleni Triantafillou, Christian Muise, Jorge A. Baier, and Sheila McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*, pages 3716–3724, 2017.

[Camacho *et al.*, 2018] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, pages 454–463, 2018.

[De Giacomo and Rubin, 2018] Giuseppe De Giacomo and Sasha Rubin. Automata-theoretic foundations of fond planning for LTL$_f$/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*, pages 854–860, 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.*, 2020a] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. Pure-past linear temporal and dynamic logic on finite traces. In *IJCAI 2020*, pages 4959–4965, 2020.

[De Giacomo *et al.*, 2020b] Giuseppe De Giacomo, Antonio Di Stasio, Moshe Y. Vardi, and Shufang Zhu. Two-stage technique for LTL$_f$ synthesis under LTL assumptions. In *KR*, 2020.

[DeCastro *et al.*, 2014] Jonathan A. DeCastro, Rüdiger Ehlers, Matthias Rungger, Ayca Balkan, Paulo Tabuada, and Hadas Kress-Gazit. Dynamics-based reactive synthesis and automated revisions for high-level robot control. *CoRR*, abs/1410.6375, 2014.

[Ehlers and Raman, 2016] Rüdiger Ehlers and Vasumathi Raman. Slugs: Extensible GR(1) synthesis. In *CAV*, volume 9780 of *Lecture Notes in Computer Science*, pages 333–339. Springer, 2016.

[Ehlers *et al.*, 2017] Rüdiger Ehlers, Stéphane Lafortune, Stavros Tripakis, and Moshe Y. Vardi. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems*, 27(2):209–260, 2017.

[Finkbeiner, 2016] Bernd Finkbeiner. Synthesis of Reactive Systems. *Dependable Software Sys. Eng.*, 45:72–98, 2016.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning – Theory and Practice*. Elsevier, 2004.

[He *et al.*, 2019] Keliang He, Andrew M. Wells, Lydia E. Kavraki, and Moshe Y. Vardi. Efficient symbolic reactive synthesis for finite-horizon tasks. In *ICRA*, pages 8993–8999, 2019.

[Jacobs and Perez, 2020] Swen Jacobs and Guillermo A. Perez. 7th Reactive Synthesis Competition: SYNTCOMP 2020. http://www.syntcomp.org/syntcomp-2020-results/, 2020.

[Kress-Gazit *et al.*, 2009] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.

[Kupferman, 2012] Orna Kupferman. Recent Challenges and Ideas in Temporal Synthesis. In *SOFSEM 2012*, pages 88–98, 2012.

[Lichtenstein *et al.*, 1985] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Logic of Programs*, pages 196–218, 1985.

[Meyer *et al.*, 2018] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In *CAV*, pages 578–586, 2018.

[Piterman *et al.*, 2006] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, LNCS, pages 364–380. Springer, 2006.

[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. pages 46–57, 1977.

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

[Zhu *et al.*, 2017a] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. A symbolic approach to Safety LTL synthesis. In *HVC*, pages 147–162, 2017.

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

[Zhu *et al.*, 2020] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, and Moshe Y. Vardi. LTL$_f$ synthesis with fairness and stability assumptions. In *AAAI*, pages 3088–3095, 2020.