Two-Stage Technique for LTL_f Synthesis Under LTL Assumptions

*Giuseppe De Giacomo[1], Antonio Di Stasio[1], Moshe Y. Vardi[2], Shufang Zhu[1,3]
[1] Sapienza Università di Roma, Roma, Italy
[2] Rice University, Houston, TX, USA
[3] Shanghai Industrial Control Safety Innovation Technology Co., Ltd, Shanghai, China
{degiacomo,distasio}@diag.uniroma1.it, vardi@cs.rice.edu, shufangzhu.szhu@gmail.com

Abstract

In synthesis, assumption are constraints on the environments that rule out certain environment behaviors. A key observation is that even if we consider a system with LTL_f goals on finite traces, assumptions need to be expressed considering infinite traces, using LTL on infinite traces, since the decision to stop the trace is controlled by the agent. To solve synthesis of LTL_f goals under LTL assumptions, we could reduce the problem to LTL synthesis. Unfortunately, while synthesis in LTL_f and in LTL have the same worst-case complexity (both are 2EXPTIME-complete), the algorithms available for LTL synthesis are much harder in practice than those for LTL_f synthesis. Recently, it has been shown that in basic forms of fairness and stability assumptions we can avoid such a detour to LTL and keep the simplicity of LTL_f synthesis. In this paper, we generalize these results and show how to effectively handle any kind of LTL assumptions. Specifically, we devise a two-stage technique for solving LTL_f synthesis under general LTL assumptions and show empirically that this technique performs much better than standard LTL synthesis.

1 Introduction

Automated program synthesis is one of the most ambitious problem of CS and AI: devise a "mechanical translation of human-understandable task specifications to a program that is known to meet the specifications" (Kreitz 1998; Vardi 2018). One of the most interesting forms of synthesis in AI is reactive synthesis, where one synthesizes a program for interactive/reactive ongoing computations (Church 1963; Pnueli and Rosner 1989), and which is tightly related to planning in nondeterministic domains (Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013). We have a set of boolean variables $\mathcal{X}$ controlled by the environment (the fluents) and a set of boolean variables $\mathcal{Y}$ controlled by the agent (the actions) and a specification Φ of the task of interest in terms of linear-time temporal logic (LTL) (Pnueli 1977), which is one of the most used logics in formal verification. The synthesis has to produce a program, aka a strategy, for the agent such that for every strategy adopted by the environment the simultaneous execution of the two strategies produce a trace that satisfies Φ (Pnueli and Rosner 1989; Finkbeiner 2016; Ehlers et al. 2017).

*Authors names are ordered alphabetically by last name

Recently the problem of (reactive) synthesis has been studied in the case the task specification involves properties over an unbounded but finite sequence of successive states. In this case we do synthesis for LTL over finite traces (LTL_f) and its variants (De Giacomo and Vardi 2015). The algorithms for LTL_f synthesis are much simpler that those for LTL synthesis and as a results much more scalable as shown experimentally (Zhu et al. 2017; Camacho et al. 2018a; Bansal et al. 2020).

In standard synthesis the environment is free to choose an arbitrary move at each step, but in AI typically we have a model of the world, i.e., of the environment's behavior, e.g., encoded in a planning domain (Green 1969; Geffner and Bonet 2013; De Giacomo and Rubin 2018), or more generally directly in temporal logic (Chatterjee, Henzinger, and Jobstmann 2008; Bloem et al. 2014; Bonet et al. 2017; D'Ippolito, Rodríguez, and Sardiña 2018). In other words, we are interested in synthesis under assumptions (Aminof et al. 2018; Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2019; Zhu et al. 2020), which can be reduced to standard synthesis of the implication:

$$Env \rightarrow Goal$$

where Env is the specification of the environment (the assumption) and $Goal$ is the specification of the task of the agent (Chatterjee, Henzinger, and Jobstmann 2008; Aminof et al. 2019). The agent has to realize its task $Goal$ only on those traces that satisfy the assumption Env on the environment. It is of interest to study synthesis under assumptions for LTL_f goals. But, while it is natural to consider the task specification $Goal$ as an LTL_f formula, requiring that also Env is an LTL_f formula is often too strong. Intuitively, the environment has to react to the agent's moves anyway, independently of whether the agent accomplishes its task (in a finite number of steps or not).

As a result Env typically needs to be expressed in LTL not LTL_f (Camacho, Bienvenu, and McIlraith 2018; Zhu et al. 2020). So, even when focusing on LTL_f, what we need to study is the case where we have the task $Goal$ expressed in LTL_f and the assumption Env expressed in LTL.

One way to handle this case is to translate $Goal$ into LTL (De Giacomo and Vardi 2013) and then do LTL synthesis for $Env \rightarrow Goal$, c.f. (Zhu et al. 2017). But, as mentioned above, while synthesis in LTL_f and in LTL have the same worst-case complexity, being both 2EXPTIME-complete (Pnueli and

Rosner 1989; De Giacomo and Vardi 2015), the algorithms available for LTL synthesis are much harder in practice than those for LTL_f synthesis. In particular, the lack of efficient algorithms for the crucial step of automata determinization is a major obstacle for scalable implementations (Fogarty et al. 2013; Finkbeiner 2016). In spite of several advancements in synthesis, such as reducing to parity games (Meyer, Sickert, and Luttenberger 2018), bounded synthesis based on solving iterated safety games (Kupferman and Vardi 2005; Finkbeiner and Schewe 2013; Gerstacker, Klein, and Finkbeiner 2018), or recent techniques based on iterated FOND planning (Camacho et al. 2018b), LTL synthesis remains challenging.

In contrast, in LTL_f synthesis the determination step can be done through the much simpler subset construction (Rabin and Scott 1959) and moreover the resulting DFA can be seen as a game arena where environment and agent make their own moves. On this arena, the agent wins if adversarial reachability of the DFA accepting states is fulfilled (De Giacomo and Vardi 2015).

In (Zhu et al. 2020) it is shown that one can maintain the simplicity of LTL_f synthesis in presence of LTL assumptions expressing a basic form of *fairness* $\Box\Diamond\alpha$, i.e., always eventually α, and a basic form of *stability* $\Diamond\Box\alpha$, i.e., eventually always α (where in both cases the truth value of α is under the control of the environment).

In this paper, we generalize this result to arbitrary LTL formulas, and provide a two-stage technique to effectively handle any kind of LTL assumptions. These techniques take advantage of the simpler way to handle LTL_f goals in stage 1 and confines the difficulty of handling LTL assumption to the bare minimum in stage 2. As a result, as long as the part of assumptions that really require LTL and not LTL_f is small we do obtain scalability. Indeed, we show empirically that this technique performs much better than standard LTL synthesis.

2 Preliminaries

LTL and LTL_f. LTL is one of the most popular logics to express dynamic properties in Formal Verification (Pnueli 1977). Given a set of propositions $\mathcal{P}$ the formulas of LTL are generated by the following grammar:

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

where $a \in \mathcal{P}$. We use common abbreviations such as *eventually* as $\Diamond\varphi \doteq true\,\mathcal{U}\,\varphi$; *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$.

Formulas of LTL are interpreted over infinite traces $\pi \in \mathcal{P}^\omega$. A *trace* $\pi = \pi_1, \pi_2, \ldots$ is a sequence of propositional interpretations (sets), where for all $i \geq 0$, $\pi_i \in 2^{\mathcal{P}}$ is the i-th interpretation of π. Intuitively, π_i is interpreted as the set of propositions which are $true$ at instant i. Given π, we define when an LTL formula φ *holds* at position i, written $\pi, i \models \varphi$, inductively on the structure of φ, as follows:

- $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in \mathcal{P}$);
- $\pi, i \models \neg\varphi$ 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 $\pi \models \varphi$, 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).[1] We denote the last position (i.e., index) in the finite trace π by $last(\pi)$. The syntax of LTL_f is exactly the same to the syntax of LTL. We define the satisfaction relation $\pi, i \models \varphi$, stating that φ holds at position i, as for LTL, except that for the temporal operators we have:

- $\pi, i \models \circ\varphi$ iff $i < last(\pi)$ 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 $\pi, k \models \varphi_1$.

We say that a trace *satisfies* an LTL_f formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$. In addition to the abbreviations used for LTL we define the *weak next* operator $\bullet\varphi \triangleq \neg\circ\neg\varphi$. Over finite traces, $\neg\circ\varphi \not\equiv \circ\neg\varphi$, but we have that $\neg\circ\varphi \equiv \bullet\neg\varphi$.

Two-Player Games. We consider two-player games played on finite arenas. Informally, two players, agent and environment, play on a game arena. $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of Boolean variables: $\mathcal{X}$ controlled by the environment and $\mathcal{Y}$ controlled by the agent. Playing the game proceeds in rounds. In each round, first, the environment sets the truth value X of $\mathcal{X}$ and then the agent replies by setting the truth value Y of $\mathcal{Y}$.[2] A play describes how the agent and environment set their variables at each round till the game stops.

Formally, an *arena* is a tuple $\mathbf{A} = \langle \Sigma, S, s_0, \delta \rangle$, where $\Sigma = 2^{\mathcal{X} \cup \mathcal{Y}}$ is the alphabet, S is a set of *states*, s_0 is an *initial state*, and $\delta : S \times \Sigma \rightharpoonup S$ is the *partial transition function*.

A *play* in $\mathbf{A}$ is an infinite sequence $\rho = s_0, (X_0 \cup Y_0), s_1, (X_1 \cup Y_1), \ldots$ such that s_0 is the initial state and $s_{i+1} = \delta(s_i, X_i \cup Y_i)$ for all $i \geq 0$. A *history* $\rho^n = s_0, (X_0 \cup Y_0), s_1, (X_1 \cup Y_1), \ldots, s_{n-1}, (X_{n-1} \cup Y_{n-1}), s_n$ is a finite prefix of a play ending in a state, and we denote by $\mathsf{lst}(\rho^n) = s_n$ its last state. The set of plays is denoted as $\mathsf{Play}(\mathbf{A})$, and the set of histories is denoted as $\mathsf{Hist}(\mathbf{A})$. Moreover, given $\rho \in \mathsf{Play}(\mathbf{A})$ (resp., $h \in \mathsf{Hist}(\mathbf{A})$) we denote by $\rho_{|\Sigma}$ (resp., $h_{|\Sigma}$) the projection of ρ (resp., h) on Σ. Intuitively, $h_{|\Sigma}$ keeps only values of variables from Σ in ρ.

A *strategy* in $\mathbf{A}$ for the agent is a function $\sigma_{ag} : \mathsf{Hist}(\mathbf{A}) \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$, and for the environment is a function $\sigma_{env} : \mathsf{Hist}(\mathbf{A}) \to 2^{\mathcal{X}}$. The strategies σ_{ag} and σ_{env} can be equivalently defined as functions $\sigma_{ag} : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$ and $\sigma_{env} : (2^{\mathcal{Y}})^* \to 2^{\mathcal{X}}$, respectively, as δ is deterministic. A strategy σ_{ag} is *memoryless* if $\sigma_{ag}(\mathsf{lst}(h), X) = \sigma_{ag}(\mathsf{lst}(h'), X)$ for all $h, h' \in \mathsf{Hist}(\mathbf{A})$ and $X \in \mathcal{X}$, that is, the strategy only depends on the last location of the history. We define memoryless strategies for the environment analogously. Note that a memoryless strategy can be defined on the set of states, instead of the set of histories. Thus we have that the strategies are of the form $\sigma_{ag} : S \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ and $\sigma_{env} : S \to 2^{\mathcal{X}}$.

[1] In this paper we focus on LTL_f for simplicity, however one can adopt its extension LDL_f instead, without any changes in the results and techniques reported.

[2] Here, we consider the environment as the first-player (as typical in planning), but a version where the agent moves first can be obtained by a small modification.

The *outcome* of two strategies σ_{ag} and σ_{env} in $\mathbf{A}$, denoted $outcome(\mathbf{A}, \sigma_{ag}, \sigma_{env})$, is the play $\rho = s_0(X_0 \cup Y_0)s_1(X_1 \cup Y_1) \ldots \in \mathsf{Play}(\mathbf{A})$ such that for all $i \geq 0$, we have $\sigma_{env}(\rho^i) = X_i$ and $s_{i+1} = \delta(s_i, X_i \cup \sigma_{ag}(\rho^i, X_i))$. A play π is *consistent* with agent strategy σ_{ag} (resp., environment σ_{env}) if $\pi = outcome(\mathbf{A}, \sigma_{ag}, \sigma_{env})$ for some environment strategy σ_{env} (resp., agent σ_{ag}).

A *game* is a tuple $\mathbf{G} = \langle \mathbf{A}, W \rangle$, where $\mathbf{A}$ is the arena of the game and W is the *winning objective*, which is set of desirable plays $W \subseteq \mathsf{Play}(\mathbf{A})$ for the agent (resp. for the environment). A play $\rho \in \mathsf{Play}(\mathbf{A})$ *satisfies* the winning objective W if $\rho \in W$. An agent (resp., environment) strategy σ_{ag} (resp., σ_{env}) is *winning* in $\mathbf{G} = \langle \mathbf{A}, W \rangle$ for a winning objective W if for every play $\rho \in \mathsf{Play}(\mathbf{A})$ consistent with σ_{ag} (resp., σ_{env}) we have that $\rho \in W$.

Given a state s', we also say that the agent (resp., the environment) has a winning strategy from s' in $\mathbf{G} = \langle \mathbf{A}, W \rangle$ for a winning objective W if the agent (resp., the environment) has a winning strategy in the game $\mathbf{G} = \langle \mathbf{A}', W \rangle$, where $\mathbf{A}' = \langle S, s', \Sigma, \delta \rangle$, i.e., the same arena $\mathbf{A}$ but with the new initial state s'. By $\mathrm{Win}_{ag}(\mathbf{G})$ (resp., $\mathrm{Win}_{env}(\mathbf{G})$) we denote the set of states, from which the agent (resp., the environment) has a winning strategy.

Here, we specifically consider reachability, safety, parity, and LTL objectives.

- *Reachability objectives.* Given a set $T \subseteq S$ of target states, the reachability objective

$$Reach(T) = \{\rho \in \mathsf{Play}(\mathbf{A}) \mid \exists k \geq 0 : \mathsf{lst}(\rho^k) \in T\}$$

  requires that a state in T is visited at least once.

- *Safety objectives.* Given a set $T \subseteq S$ of safe states, the safety objective

$$Safe(T) = \{\rho \in \mathsf{Play}(\mathbf{A}) \mid \forall k \geq 0 : \mathsf{lst}(\rho^k) \in T\}$$

  requires that only states in T are visited. This is the dual of reachability objectives.

- LTL *objectives.* Given an LTL formula φ over the variables $\mathcal{X} \cup \mathcal{Y}$, the LTL objective

$$\mathrm{LTL}(\varphi) = \{\rho \mid \rho_{|\Sigma} \models \varphi\}$$

  requires that plays make the LTL formula φ true. Note that from the play we are projecting out the states of the arena (which are anyway determined by the sequences of valuations of Σ).

- *Parity objective.* For some $d \in \mathbb{N}_{\mathrm{o}}$, let $\mathsf{p} : S \times \Sigma \to \{0, \ldots, d-1\}$ be a *priority function*.[3] Given a play $\rho = s_i(X_i \cup Y_i)_{i \in \mathbb{N}_{\mathrm{o}}} \in \mathsf{Play}(\mathbf{A})$, by $\mathsf{p}(\rho) = (\mathsf{p}(s_i, X_i \cup Y_i))_{i \in \mathbb{N}_{\mathrm{o}}}$ we denote the associated priority sequence. The parity objective $Parity(\mathsf{p})$ is defined as the set of ρ such that the minimum priority that appears infinitely often along $\mathsf{p}(\rho)$ is even.

[3]In this paper we define the priority function on transitions, which is nowadays the preferred definition for implementation and in-line with other recent papers and tools (Giannakopoulou and Lerda 2002; Duret-Lutz et al. 2016; Babiak et al. 2015; Kretínský, Meggendorfer, and Sickert 2018).

Depending on the actual winning objective we get reachability, safety, LTL, or parity games. Reachability, safety, and parity games admit memoryless strategies, while LTL games require finite-state strategies. All these games are *determined*, i.e., from each state of the game if the player has no winning strategy then the opponent has one (Martin 1975).

Finite-State Automata on Finite and Infinite Words. A *finite-state automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$, where Σ is a finite input alphabet, Q is the finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to 2^Q$ is the *nondeterministic* transition function. An automaton $\mathcal{A}$ is *deterministic* if $|\delta(q, a)| = 1$, for all $(q, a) \in Q \times \Sigma$, i.e., $\delta : Q \times \Sigma \to Q$.

A *run* on a *finite* word $a_0 \ldots a_n$ is a finite sequence $q_0 \ldots q_n$ such that $q_{i+1} \in \delta(q_i, a_{i+1})$ for all $0 \leq i < n$. A run on an *infinite* word $a_0 a_1 \cdots$ is an infinite sequence $q_0 q_1 \ldots$ such that $q_{i+1} \in \delta(q_i, a_{i+1})$ for all $i \geq 0$.

Nondeterministic and deterministic finite-state automata (NFA and DFA, respectively) on finite words are a pair $(\mathcal{A}, F)$, where $F \subseteq Q$ is the set of accepting states. A run $q_0 \ldots q_n$ of $\mathcal{A}$ is *accepting* if $q_n \in F$. By $\mathcal{L}(\mathcal{A})$ we denote the set of all words over Σ accepted by $\mathcal{A}$.

A *nondeterministic Büchi automaton* (NBA) on infinite words is a pair $(\mathcal{A}, F)$, where $F \subseteq Q$, as for NFA, and a run $q_0 q_1 \ldots$ of $\mathcal{A}$ is *accepting* if for infinitely many i, $q_i \in F$.

A *deterministic parity automaton* (DPA) on infinite words is a pair $(\mathcal{A}, \mathsf{p})$, where $\mathsf{p} : Q \times \Sigma \to \{0, \ldots, d-1\}$ for some $d \in \mathbb{N}_{\mathrm{o}}$, is a priority function as defined for parity objectives above. A run ρ of $\mathcal{A}$ is *accepting* if the minimum priority that appears infinitely often along $\mathsf{p}(\rho)$ is even.

Here, we consider games played on arenas based on automata. Specifically, we consider games $\mathbf{G} = \langle \mathcal{A}, W \rangle$, for a variety of objectives W, where the arena is an automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$.

Reactive Synthesis. *Reactive Synthesis* is the problem of producing a strategy for the agent such that it satisfies a given property no matter how the environment behaves (Church 1963; Pnueli and Rosner 1989). Let $\mathcal{X}$ and $\mathcal{Y}$ disjoint boolean variables, with $\mathcal{X}$ controlled by environment and $\mathcal{Y}$ controlled by the agent. As for two-player games, the idea is that the environment sets the variables in $\mathcal{X}$, and the agent then responds by setting the variables in $\mathcal{Y}$. An *agent strategy* is a function $\sigma_{ag} : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$, and an environment strategy is a function $\sigma_{env} : (2^Y)^* \to 2^X$. A *trace* is a sequence $(X_0 \cup Y_0)(X_1 \cup Y_1) \ldots$ over the alphabet $2^{\mathcal{X} \cup \mathcal{Y}}$. An agent strategy *induces* a trace $(X_i \cup Y_i)_i$ if $\sigma_{ag}(X_0 X_1 \ldots X_j) = Y_j$ for every $j \geq 0$. An environment strategy *induces* a trace $(X_i \cup Y_i)_i$ if $\sigma_{env}(\epsilon) = X_0$ and $\sigma_{env}(Y_0 Y_1 \ldots Y_j) = X_{j+1}$ for every $j \geq 0$. For an agent strategy σ_{ag} and an environment strategy σ_{env} let $\tau(\sigma_{ag}, \sigma_{env})$ denote the unique trace induced by both σ_{ag} and σ_{env}.

LTL Synthesis. Let φ be an LTL formula over $\mathcal{X} \cup \mathcal{Y}$. An agent strategy σ_{ag} (resp., environment strategy σ_{env}) *realizes* φ if for every environment strategy σ_{env} (resp., agent strategy σ_{ag}), the trace $\tau(\sigma_{ag}, \sigma_{env})$ satisfies φ. In this case we say that φ is *agent realizable (resp., environment realizable)*.

The problem of LTL *synthesis* is to decide whether φ is agent realizable and if so to compute a finite-state strategy (Pnueli and Rosner 1989). Algorithm 1 shows a classical approach to solve LTL synthesis.

Algorithm 1 LTL synthesis

Input: LTL formula φ;
Output: agent strategy σ_{ag} that realizes φ;
1: Compute the corresponding NBA $\mathcal{A}_\varphi$;
2: Determinize $\mathcal{A}_\varphi$ into a DPA $\mathcal{B}_\varphi$;
3: Solve the parity game over the arena $\mathcal{B}_\varphi$.

LTL synthesis is 2EXPTIME-complete (Pnueli and Rosner 1989), but more importantly computing the resulting DPA (Safra 1988; Piterman 2007) remains difficult to scale, despite extensive research (Esparza and Kretínský 2014; Sickert et al. 2016; Kretínský et al. 2017; Esparza et al. 2017). Moreover, solving *parity games* requires essentially to compute nested fixpoints corresponding to priorities (exponentially many in general in the case of LTL synthesis). So even this problem, although in UPTIME $\cap$ COUPTIME (Jurdzinski 1998) and in fact quasi-polynomial (Calude et al. 2017), remains hard in practice for large numbers of priorities.

LTL$_f$ Synthesis. Let φ be an LTL$_f$ formula over $\mathcal{X} \cup \mathcal{Y}$. Let $\tau^m(\sigma_{ag}, \sigma_{env})$ be the finite trace that is a prefix up to m of the trace $\tau(\sigma_{ag}, \sigma_{env})$. An agent strategy σ_{ag} (resp., environment strategy σ_{env}) *realizes* φ if for every environment strategy σ_{env} (resp., agent strategy σ_{ag}) there exists $m \geq 0$, chosen by the agent, such that the finite trace $\tau^m(\sigma_{ag}, \sigma_{env})$ satisfies φ, that is, φ is *agent (resp., environment) realizable*.

The problem of LTL$_f$ *synthesis* is to decide whether φ is agent realizable and computing a finite-state strategy if one exists (De Giacomo and Vardi 2015). The algorithm for solving LTL$_f$ synthesis is reported in Algorithm 2.

Algorithm 2 LTL$_f$ synthesis

Input: LTL$_f$ formula φ;
Output: agent strategy σ_{ag} that realizes φ;
1: Compute the corresponding NFA $\mathcal{A}_\varphi$;
2: Determinize $\mathcal{A}_\varphi$ into a DFA $\mathcal{B}_\varphi$;
3: Solve the reachability game on the arena $\mathcal{B}_\varphi$.

LTL$_f$ synthesis is 2EXPTIME-complete (De Giacomo and Vardi 2013), i.e., the same as for infinite traces, but good algorithms exist for obtaining DFA: compute NFA (De Giacomo and Vardi 2015) and determinize using well-known subset construction (Rabin and Scott 1959). Moreover, solving DFA *game* just requires solving adversarial reachability. Several recent papers are showing experimentally that LTL$_f$ synthesis is indeed scalable (Zhu et al. 2017; Camacho et al. 2018a; Camacho, Bienvenu, and McIlraith 2019; Zhu, Pu, and Vardi 2019; Zhu et al. 2020; Bansal et al. 2020).

3 LTL$_f$ Synthesis Under LTL Assumptions

In this paper, we are interested in solving synthesis under environment assumptions (Aminof et al. 2018; Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2019), i.e., assuming that the behavior of the environment is forced to satisfy certain restrictions. Examples of these are (nondeterminisitc) domains specifications in planning, which specify effect of actions (controlled by agent) in terms of fluents (controlled by the environment) (Geffner and Bonet 2013), fairness assumptions in strong cyclic planning (Cimatti et al. 2003), and trajectories constrains in generalized planning (D'Ippolito, Rodríguez, and Sardiña 2018; Bonet et al. 2017). Moreover environment assumptions have been investigated also in formal methods (Chatterjee, Henzinger, and Jobstmann 2008; Bloem et al. 2014).

Here, we focus on LTL$_f$ synthesis under environment assumptions, but it is important to clarify that even in this setting we may need to consider environment assumptions expressed over infinite traces, since in LTL$_f$ synthesis it is the agent that chooses when to terminate a trace not the environment (De Giacomo and Vardi 2015; Camacho, Bienvenu, and McIlraith 2018; Zhu et al. 2020). Formally, we are interested in solving synthesis for:[4]

$$Env \to Goal$$

where $Goal$ is an arbitrary LTL$_f$ formula, which is the specification of a task for the agent, and Env is an arbitrary LTL formula, which expresses restrictions on the environment behavior.

We observe that not every LTL formula Env can be considered a meaningful environment assumption. To be so it is required that the environment must have a strategy to win Env in spite of whatever the agent does. That is the environment must be able to react to every agent action, resolving its *nondeterminism* without getting stuck. Formally, Env must be realizable by the environment, i.e., there must be an environment's strategy that solves Env. If not, the agent can defeat the assumption on the environment instead of realizing its goal, trivializing the synthesis for the implication $Env \to Goal$ (Chatterjee, Henzinger, and Jobstmann 2008; Aminof et al. 2019). Nevertheless, for the results in this paper, no restriction on the LTL formula Env need to be imposed.

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

1. *The problem of* LTL$_f$ *synthesis under* LTL *assumptions is 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 agent, Env is an* LTL *formula over* $\mathcal{X} \cup \mathcal{Y}$*, and Goal is an* LTL$_f$ *formula over* $\mathcal{X} \cup \mathcal{Y}$*.*
2. *An agent strategy* $\sigma_{ag} : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$ *realizes Goal under assumption Env if for every* $\lambda = X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$*, such that* $\pi \models Env$*, there exists* $k \geq 0$ *such that* $\pi^k \models Goal$*, where* $\pi = (X_0 \cup \sigma_{ag}(X_0)), (X_1 \cup \sigma_{ag}(X_0, X_1)), \ldots$ *and* π^k *is the prefix of* π *ending at* k *(i.e.,* $\pi^k = (X_0 \cup \sigma_{ag}(X_0)), (X_1 \cup \sigma_{ag}(X_0, X_1)), \ldots, (X_k \cup \sigma_{ag}(X_0, X_1, \ldots, X_k))$*).*
3. *Solving* $\mathcal{P}$ *consists in finding an agent strategy that realizes Goal under assumption Env.*

[4] In fact, the use of an implication in this context requires some care. We refer to (Aminof et al. 2019) for a thorough discussion.

An agent strategy $\sigma_{ag} : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$ for the synthesis problem $P = \langle \mathcal{X}, \mathcal{Y}, Goal, Env \rangle$ is *winning* if it guarantees the satisfaction of $Goal$ under the condition that the environment behaves as specified by Env. A *realizability procedure* for $\mathcal{P}$ aims verifying the existence of a winning strategy σ_{ag} and the *synthesis procedure* amounts to actually computing σ_{ag}, if it exists.

We observe that one way to solve this form of synthesis is to translate $Goal$ into LTL (De Giacomo and Vardi 2013) and then do standard LTL synthesis for $Env \to Goal$, see e.g. (Camacho, Bienvenu, and McIlraith 2018). A much more efficient technique to solve the problem $\mathcal{P}$ in the special case of LTL assumptions of the form $\Box\Diamond a$ (fairness) and $\Diamond\Box a$ (stability) with a propositional is proposed in (Zhu et al. 2020). Here we generalize some of the ideas in (Zhu et al. 2020) to handle any kind of LTL assumptions.

4 Solving Synthesis

To solve the synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$ we proceed as follows:

- Translate the LTL$_f$ formula $Goal$ into the corresponding DFA $(\mathcal{A}_{Goal}, Acc)$ with $\mathcal{A}_{Goal} = \langle 2^{\mathcal{X}\cup\mathcal{Y}}, Q^G, q_0^G, \delta^G \rangle$;
- Focus on the environment as the main player; observe that all the games we consider in this paper are *determined* (see Section 2) so if the environment (resp., agent) does not win its objective, then the agent (resp. environment) wins the complement objective;
- Set as winning objective for the environment the LTL objective LTL$(Env \wedge \Box(\neg Acc))$, where Acc is a proposition true iff the current state of the DFA is accepting;
- Solve the LTL game $\mathbf{G} = \langle \mathcal{A}_{Goal}, \text{LTL}(Env \wedge \Box(\neg Acc)) \rangle$, where LTL$(Env \wedge \Box(\neg Acc))$ is the winning objective for the environment.
- Return the agent winning strategy, if one exists.

We show that this procedure is indeed sound and complete.

Theorem 1. *$\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, Env, Goal \rangle$ is realizable iff the environment does not have a winning strategy in the environment* LTL *game: $\mathbf{G} = \langle \mathcal{A}_{Goal}, \text{LTL}(Env \wedge \Box(\neg Acc)) \rangle$ i.e., iff the agent has a winning strategy in environment game $\mathbf{G}$ (that is the agent wins the complement* LTL *objective $(Env \to \Diamond Acc)$ in the arena $\mathcal{A}_{Goal}$). Moreover, every agent winning strategy for $\mathbf{G}$ is a strategy for $\mathcal{P}$, and vice-versa.*

Proof. We prove the theorem in both directions.
$\leftarrow$: If σ_{ag} is an agent winning strategy in $\mathcal{A}_{Goal}$ for the complement winning objective $(Env \to \Diamond Acc)$ then define the strategy $h_{ag}(\sigma_{ag})$ of $\mathcal{P}$ as a mapping $\lambda \in (2^{\mathcal{X}})^+$ to $\sigma_{ag}(h)$. Thus, a play $\rho \in \mathsf{Play}(\mathbf{A})$ consistent with the strategy σ_{ag} is winning for the agent in the environment game $\mathbf{G}$ such that either of the following condition holds:
• $\rho_{|\Sigma} \not\models Env$, then the trace $\pi = (X_0 \cup h_{ag}(X_0)), (X_1 \cup h_{ag}(X_0, X_1)), \ldots$ consistent with h_{ag} does not satisfy Env.
• $\rho_{|\Sigma} \models Env$ and there exists $j \geq 0$ such that $\mathsf{lst}(\rho^j) \in Acc$. This implies that $\rho^j_{|\Sigma} \models Goal$. Therefore, the trace $\pi = (X_0 \cup h_{ag}(X_0)), (X_1 \cup h_{ag}(X_0, X_1))$ consistent with h_{ag} satisfies Env, and $\pi^j = (X_0 \cup h_{ag}(X_0)), (X_1 \cup h_{ag}(X_0, X_1)), \ldots, (X_j \cup h_{ag}(X_0, X_1, \ldots, X_j))$ satisfies $Goal$. Conclude that h_{ag} is an agent winning strategy that realizes $Goal$ assuming Env.
$\rightarrow$: For this direction we assume that $\mathcal{P}$ is realizable, i.e., there exists a strategy h_{ag} that realizes $Goal$ assuming Env. Then, define the agent strategy $\sigma_{ag}(h_{ag})$ of $\mathbf{G}$ as a mapping from $\lambda \in (2^{\mathcal{X}})^+$ to $h_{ag}(\lambda)$. Thus, consider a trace π induced by h_{ag} such that either of the following condition holds:
• $\pi \not\models Env$, then the play $\rho = s_0, (X_0 \cup \sigma_{ag}(X_0)), s_1, (X_1 \cup \sigma_{ag}(X_0, X_1)), \ldots$ consistent with σ_{ag} does not satisfy Env.
• $\pi \models Env$ and there exists $j \geq 0$ such that $\pi^j \models Goal$. Therefore, the play $\rho = s_0, (X_0 \cup \sigma_{ag}(X_0)), s_1, (X_1 \cup \sigma_{ag}(X_0, X_1)), \ldots$ consistent with σ_{ag} satisfies Env. Moreover, since $\pi^j \models Goal$ the play $\rho^j_{|\Sigma} \models Goal$ and then $\mathsf{lst}(\rho^j) \in Acc$. Conclude that σ_{ag} is an agent winning strategy in the environment game $\mathbf{G} = \langle \mathcal{A}_{Goal}, \text{LTL}(Env \wedge \Box(\neg Acc)) \rangle$, i.e., winning for the complement objective $(Env \to \Diamond Acc)$ in the arena $\mathcal{A}_{Goal}$. $\Box$

We remind the reader that LTL games over an arena $\mathbf{A}$ with an LTL winning objective LTL(ψ) can be solved by translating the formula ψ into a DPA and making the product of such a DPA with the game arena $\mathbf{G}$ (Harding, Ryan, and Schobbens 2005; Sohail and Somenzi 2009) obtaining a party game, which can be solved with standard techniques, e.g., (Zielonka 1998; Di Stasio et al. 2016). We show next that we can do better by considering the specific form of $Env \wedge \Box(\neg Acc)$, which is a conjunction of a *safety formula* $\Box\neg Acc$ (Manna and Pnueli 1990) and a general LTL formula Env.

5 Two-stage Technique for Synthesis

Following (Sohail and Somenzi 2009), we can treat separately the safety component, and use such separation to reduce the size of the game arena before making the product with the DPA for Env. Based on this idea, we devise a two-stage algorithm.

Stage 1. In stage 1, given the DFA $(\mathcal{A}_{Goal}, Acc)$ for $Goal$, we solve the reachability game $\mathbf{G} = \langle \mathcal{A}_{Goal}, Reach(Acc) \rangle$ by computing the agent winning set Win_{ag} and the memoryless winning strategy σ_{ag} for the winning objective $Reach(Acc)$. Therefore, for each state in Win_{ag}, σ_{ag} returns the assignment for the agent controlled variables Y, which eventually leads to a final state. Therefore, if the initial state is in Win_{ag}, that is, σ_{ag} realizes $Goal$ independently from Env being satisfied or not, nothing else is needed and then the algorithm can stop returning Win_{ag} and σ_{ag}.

Stage 2. In stage 2, we prune the game arena by removing all states in Win_{ag}, which are winning for the agent, and hence loosing for the environment, and all transitions in and out of them, and by disabling all environment moves that can lead to Win_{ag}. Formally, given $\mathbf{G} = \langle \mathcal{A}_{Goal}, Reach(Acc) \rangle$, we define the game $\mathbf{G}_{|\neg \mathrm{Win}_{ag}} = \langle \mathbf{A}_{|\neg \mathrm{Win}_{ag}}, W \rangle$, in which arena $\mathbf{A}_{|\neg \mathrm{Win}_{ag}}$ is given as $(2^{\mathcal{X}\cup\mathcal{Y}}, S', s_0', \delta')$, where $S' = Q^G \setminus \mathrm{Win}_{ag}$, $s_0' = q_0^G$, and the transition function is defined as follows: $\delta'(s', X \cup Y)$ is undefined if either $s' \in \mathrm{Win}_{ag}$ or there exists $Y \in 2^{\mathcal{Y}}$ such that $\delta(s', X \cup Y) \in \mathrm{Win}_{ag}$ for any $X \in 2^{\mathcal{X}}$; $\delta'(s', X \cup Y) = \delta^G(s', X \cup Y)$ otherwise.

Stage 2 then proceeds as follows: (*i*) computes the corresponding DPA $(\mathcal{A}_{Env}, \mathsf{p})$ with $\mathcal{A}_{Env} = \langle 2^{\mathcal{X}\cup\mathcal{Y}}, Q^E, q_0^E, \delta^E \rangle$ for Env; (*ii*) builds the game product $\mathbf{G}_{|\neg \mathrm{Win}_{ag}} \times \mathcal{A}_{Env} = \langle \mathbf{A}^\iota, Parity(\mathsf{p}^\iota) \rangle$ of $\mathbf{G}_{|\neg \mathrm{Win}_{ag}}$ and $\mathcal{A}_{Env}$ as follows: $\mathbf{A}^\iota = \langle \Sigma, S^\iota, s_0^\iota, \delta^\iota \rangle$, where $S^\iota = S' \times Q^E$, $s_0^\iota = (s_0, q_0^E)$, and for $(s,q) \in S^\iota$ and $a \in \Sigma$ we have that $\delta^\iota((s,q),a) = (\delta'(s,a), \delta^E(q,a))$ if $\delta'(s,a)$ and $\delta^E(s,a)$ are defined, undefined otherwise. $Parity(\mathsf{p}^\iota)$ is the parity objective for the environment, where p^ι is the priority function defined as $\mathsf{p}^\iota((s,q),a) = \mathsf{p}(s,a)$ for each $(s,q) \in S^\iota$ and $a \in \Sigma$; (*iii*) solves the resulting parity game $\mathbf{G}_{|\neg \mathrm{Win}_{ag}} \times \mathcal{A}_{Env}$ for the environment by returning the winning states Win'_{env} and Win'_{ag} for the environment and the agent respectively, and the corresponding memoryless strategies σ'_{env} and σ'_{ag} (note that these return the $\mathcal{X}$ and the $\mathcal{Y}$ from each state in Win'_{env} and each state in Win'_{ag} respectively) , see e.g., (Zielonka 1998). Finally, if the initial state of $\mathbf{G}_{|\neg \mathrm{Win}_{ag}} \times \mathcal{A}_{Env}$ is in Win'_{ag}, i.e., the agent has a winning strategy for falsifying Env in $\mathbf{G}_{|\neg \mathrm{Win}_{ag}}$, then the algorithm returns the strategies σ_{ag} over $\mathbf{G}$ and σ'_{ag} over $\mathbf{G}_{|\neg \mathrm{Win}_{ag}} \times \mathcal{A}_{Env}$, along with Win_{ag} and Win'_{ag} respectively, that have to be combined to construct the winning strategy for the agent in the LTL game $\mathbf{G} = \langle \mathcal{A}_{Goal}, \mathrm{LTL}(Env \wedge \Box(\neg Acc)) \rangle$ for the complement winning objective $\mathrm{LTL}(Env \rightarrow \Diamond Acc)$. Then, we combine the strategies obtained in the two stages in a strategy for the original problem. The two-stage algorithm is detailed in Algorithm 3.

Algorithm 3 LTL_f synthesis under LTL assumptions

procedure SYNTHESIZE($\mathcal{P}$)
  / Stage 1 */*
  $(\mathcal{A}_{Goal}, Acc) = \mathrm{LTL}_f\text{\_TO\_DFA}\ (Goal)$
  let $\mathbf{G} = \langle \mathcal{A}_{Goal}, Reach(Acc) \rangle$;
  $(\mathrm{Win}_{ag}, \sigma_{ag}) = \mathrm{RGSOLVE}_{ag}(\mathbf{G})$
  if $s_0 \in \mathrm{Win}_{ag}$ **then**
    return GETSTRAT$(\mathrm{Win}_{ag}, \sigma_{ag})$
  / Stage 2 */*
  $(\mathcal{A}_{Env}, \mathsf{p}) = \mathrm{LTL\_TO\_DPA}\ (Env)$
  $(\mathrm{Win}'_{ag}, \sigma'_{ag}) = \mathrm{PGSOLVE}_{env}(\mathbf{G}_{|\neg \mathrm{Win}_{ag}} \times \mathcal{A}_{Env})$
  if $s'_0 \in \mathrm{Win}'_{ag}$ **then**
    return GETSTRAT$(\mathrm{Win}_{ag}, \sigma_{ag}, \mathrm{Win}'_{ag}, \sigma'_{ag})$
  else
    return *"Unrealizable"*

The algorithm works by calling the following procedures: (*i*) $\mathrm{LTL}_f\text{\_TO\_DFA}\ (\phi)$ translates an LTL_f formula ϕ into the corresponding DFA; (*ii*) $\mathrm{RGSOLVE}_{ag}(\mathbf{G})$ solves the reachability game $\mathbf{G}$ for the agent by returning the winning set Win_{ag} and the memoryless strategy σ_{ag}; (*iii*) LTL_TO_DPA (φ) translates an LTL formula φ into the corresponding DPA; (*iv*) $\mathrm{PGSOLVE}_{env}(\mathbf{G})$ solves the parity game $\mathbf{G}$ for the environment by returning the winning set Win'_{ag} and the memoryless strategy σ'_{ag} for the agent. Note that, we do not report Win_{env} and σ_{env} for the environment as output of the solving game procedure since we are only interested in the winning sets and strategies of the agent; (*iv*) GETSTRAT combines the strategies returned by stage 1 and 2 to generate the strategy σ_{ag} for the original problem as we detail next.

Strategy Extraction. We now detail the procedure GETSTRAT that given $(\mathrm{Win}_{ag}, \sigma_{ag})$ and $(\mathrm{Win}'_{ag}, \sigma'_{ag})$ builds a finite-state *transducer* representing the strategy σ_{ag} of the original problem. We denote by $\eta((s,q)) = s$ for $(s,q) \in S^\iota$ the projection of $S' \times Q^E$ on S'. Formally, the winning strategy for the agent $\sigma_{ag} : (2^{\mathcal{X}})^+ \rightarrow 2^{\mathcal{Y}}$ can be represented as a deterministic finite transducer $\mathcal{T} = \langle 2^{\mathcal{X}\cup\mathcal{Y}}, Q^T, q_0^T, \varrho, \omega_f \rangle$ based on the output of Algorithm 3:

- $q_0^T = s_0$ if $s_0 \in \mathrm{Win}_{ag}$, $q_0^T = s_0^\iota$ otherwise.
- $Q^T = \mathrm{Win}_{ag} \cup \mathrm{Win}'_{ag}$;
- $\varrho : Q^T \times 2^{\mathcal{X}} \rightarrow Q^T$ is the transition function such that given $Y = \omega_f(q, X)$ we have that:

$$\varrho(q,X) = \begin{cases} \delta^G(q, X \cup Y) & \text{if } q \in Win_{ag} \\ \delta^E(q, X \cup Y) & \text{if } \delta^E(q, X \cup Y) \\ & \text{is defined} \\ \delta^G(\eta(q), X \cup Y) & \text{otherwise} \end{cases}$$

- $\omega_f : Q^T \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ is the output function defined as:

$$\omega_f(q,X) = \begin{cases} \sigma_{ag}(q,X) & \text{if } q \in \mathrm{Win}_{ag} \\ \sigma'_{ag}(q,X) & \text{if } q \in \mathrm{Win}'_{ag} \\ & \text{and } \sigma'_{ag}(q,X) \neq \bot \\ \text{choose } Y \text{ such that} & \\ \delta^G(\eta(q), X \cup Y) \in \mathrm{Win}_{ag} & \text{if } q \in \mathrm{Win}'_{ag} \\ & \text{and } \sigma'_{ag}(q,X) = \bot \end{cases}$$

The transducer $\mathcal{T}$ generates σ_{ag} in the sense that for every $\lambda \in (2^{\mathcal{X}})^\omega$, we have $\sigma_{ag}(\lambda) = \omega_f(\varrho(\lambda))$, with the usual extension of ω_f to words over $2^{\mathcal{X}}$ from q_0^T. Intutively the strategy generated by $\mathcal{T}$ says that, if the environment decides to stay in $\mathbf{G}_{|\neg \mathrm{Win}_{ag}}$ then the agent follows the strategy σ'_{ag} falsifying Env. Otherwise, the environment could escape and visit some state $s \in \mathrm{Win}_{ag}$ in $\mathbf{G}$, as showed by Lemma 1, but in this case the agent follows the strategy σ_{ag} reaching an accepting state and then satisfying $Goal$.

Lemma 1. *Let $q \in \mathrm{Win}'_{ag}$ and $\sigma'_{ag}(q,X) = \bot$ for some $X \in 2^{\mathcal{X}}$. Then there exists $Y \in 2^{\mathcal{Y}}$ such that $\delta^G(\eta(q), X \cup Y) = s'$ and $s' \in Win_{ag}$.*

Proof. Let $F = \{s \mid \exists X \exists Y. \delta^P(s, X \cup Y) \in Win_{ag}\} \setminus \mathrm{Win}_{ag}$, i.e., the set of predecessors of the states in Win_{ag}. Thus, $F \subseteq S'$ and $F \times Q^E$ is a subset of S^ι. Then for every $(s,q) \in F \times Q^E$ we have that $\sigma'((s,q),X) = \bot$ for some $X \in 2^{\mathcal{X}}$ and there exists $Y \in 2^{\mathcal{X}}$ such that $\delta^P(\eta(s,q), X \cup Y) \in Win_{ag}$. $\square$

Correctness. Now we prove soundness and completeness. By Theorem 1 it suffices to show that $\mathcal{T}$ generates a winning strategy σ_{ag} for the agent in the LTL game for the environment $\mathbf{G} = \langle \mathcal{A}_{Goal}, \mathrm{LTL}(Env \wedge \Box(\neg Acc)) \rangle$.

Theorem 2. *Let $\mathbf{G} = \langle \mathcal{A}_{Goal}, \text{LTL}(Env \wedge \Box(\neg Acc))\rangle$ be an* LTL *game and σ_{ag} a strategy for the agent. If $\sigma_{ag}(\lambda) = \omega_f(\varrho(\lambda))$ then σ_{ag} is a winning strategy for the agent in $\mathbf{G}$ for the complement winning objective* $\text{LTL}(Env \rightarrow \Diamond Acc)$.

Proof. Let $\lambda \in (2^{\mathcal{X}})^\omega$ be an arbitrary infinite sequence and $\rho = s_0, (X_0 \cup \sigma_{ag}(X_0)), s_1, (X_1 \cup \sigma_{ag}(X_0, X_1)), \ldots \in \mathsf{Play}(\mathbf{A})$ the corresponding play consistent with the strategy σ_{ag} for the complement winning objective $\text{LTL}(Env \rightarrow \Diamond Acc)$. We show that $\rho \in \text{LTL}(Env \rightarrow \Diamond Acc)$.

- $s_0 \in Win_{ag}$, then Algorithm 3 stops at the first stage returning a memoryless winning strategy σ_{ag} for the agent in $\mathbf{G} = \langle \mathcal{A}_{Goal}, Reach(Acc)\rangle$ for the objective $Reach(Acc)$. Thus, the construction of ω_f follows σ_{ag} and then there exists $j \geq 0$ such that $\mathsf{lst}(\rho^j) \in Acc$, that is, $\rho^j_{|\Sigma}$ satisfies $Goal$.
- $s_0 \notin \text{Win}'_{ag}$ and every state s along the play ρ belongs to $\mathbf{G}_{|\neg \text{Win}_{ag}}$. Thus the strategy generated by ω_f follows the strategy σ'_{ag} over $\mathbf{G}_{|\neg \text{Win}_{ag}} \times \mathcal{A}_{Env}$ and then $\rho_{|\Sigma} \not\models Env$.
- $s_0 \notin \text{Win}_{ag}$ and there exists $j \geq 0$ such that $\rho^j \in \mathsf{Hist}(\mathbf{A})$ and $\mathsf{lst}(\rho^j) = s_j \in \text{Win}_{ag}$. Then, from s_j the construction of ω_f generates the strategy based on σ_{ag} that leads to some state in Acc. Therefore, there exists $k \geq j$ such that $\mathsf{lst}(\rho^k) \in Acc$, i.e., $\rho^k_{|\Sigma} \models Goal$. □

Finally, we show the optimality of the algorithm:

Theorem 3. *Algorithm 3 solves* LTL_f *synthesis under assumptions in* 2EXPTIME.

Proof. Stage 1 needs to build the corresponding DFA of an LTL_f formula which worst-case is 2EXPTIME (De Giacomo and Vardi 2015), and solve the reachability game over the DFA that is linear in the size of the game. Stage 2 builds the corresponding DPA of an LTL formula which worst-case, again, is 2EXPTIME (Safra 1988), and solves the parity game over the DPA which cost is polynomial in the number of the states and exponential in the number of the priorities. □

6 Separating LTL_f Assumptions

It is interesting to consider the case where part of the assumptions are expressed in LTL_f, i.e., the environment assumptions have the form $Env = Env_\infty \wedge Env_f$, where Env_∞ can be expressed as an LTL formula and Env_f as an LTL_f formula. In this case the synthesis problem $Env \rightarrow Goal$ becomes

$$(Env_\infty \wedge Env_f) \rightarrow Goal$$

which is equivalent to

$$Env_\infty \rightarrow (Env_f \rightarrow Goal)$$

where $(Env_f \rightarrow Goal)$ is expressible in LTL_f. Therefore, we can synthesize for

$$Env_\infty \rightarrow Goal'$$

where $Goal' = (Env_f \rightarrow Goal)$ is an LTL_f formula and Env_∞ is an LTL one. In this way, Env_f does not contribute the resulting DPA and it can be handled during Stage 1 instead of Stage 2 of our technique. Specifically we build a DFA as the union of the DFA $\overline{\mathcal{A}_{Env_f}}$, i.e., the complement of the DFA for Env_f, and the DFA $\mathcal{A}_{Goal}$ for the goal. Thus, we get an important advantage:

We handle a possibly large part of the environment assumption at stage 1, thus avoiding its use during the construction of the DPA, *which is the most costly part of the technique.*

A notable case of Env_f is when it expresses in LTL_f a propositional planning domain (De Giacomo and Vardi 2013; Aminof et al. 2019). Translating a possibly nondeterministic domain in LTL_f can be done in linear time in the size of a compact representation like PDDL (McDermott et al. 1998). In fact, we can even do better by avoiding the explicit translation into LTL_f, as shown next.

Planning Domains. In (De Giacomo and Rubin 2018) it is observed that each nondeterministic domain Dom, typical of FOND planning (Geffner and Bonet 2013), can translated in linear time into an equivalent DFA $\mathcal{A}_{Dom}$.[5] So we can use Dom directly to build the arena of the game at stage 1. In fact, more can be done. If the nondeterministic domain is expressed in compact language as PDDL, we can transform directly this representation in linear time into a symbolic representation of a DFA $\mathcal{A}_{Dom}$, and hence its complement $\overline{\mathcal{A}_{Dom}}$, and take advantage of this to speed up the stage 1 of our technique. In this case we get a second advantage:

We avoid the cost of the translation of the environment assumption corresponding to the domain in generating the DFA *at stage 1.*

We illustrate the whole construction of our two-stage technique with a simple example. Consider the following simplified version of the classical Yale shooting domain in (De Giacomo and Rubin 2018), where we have that a turkey is either alive or not and the actions are shoot, which may either kill the turkey or not, and wait, which does nothing (both actions do not have preconditions). Consider as $Goal$ the LTL_f formula $\Diamond \neg a$ and as assumption Env the LTL formula $\Box\Diamond shoot \rightarrow \Diamond(\neg\, a)$. Figures 1 and 2 represent the DFAs $\mathcal{A}_{Dom}$ and $\mathcal{A}_{Goal}$ for the domain and the goal, respectively. Then, we build the arena of the game at stage 1 doing the product between $\overline{\mathcal{A}_{Dom}}$ and $\mathcal{A}_{Goal}$, obtaining the DFA $\mathcal{A}_u$ depicted in Figure 3.

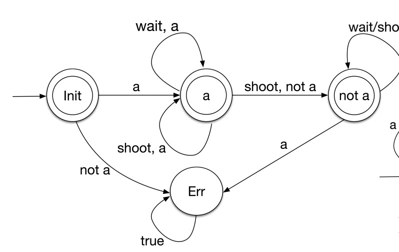

Figure 1: DFA $\mathcal{A}_{Dom}$ for the domain. Note that, any agent action in $init$ leads to a.

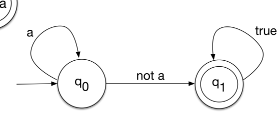

Figure 2: DFA for $\Diamond\neg a$

We now solve the reachability game over the game arena $\mathcal{A}_u$ by computing the set of states Win_{ag}, from which the

[5]For simplicity we do not consider preconditions, but instead assume that domains have a special fluent $PrecViolated$ that is the effect of doing an action violating the preconditions. Once $PrecViolated$ is true no action can make it false.

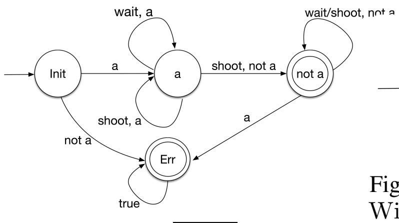

Figure 3: $\mathcal{A}_u = \overline{\mathcal{A}_{Dom}} \cup \mathcal{A}_{Goal}$

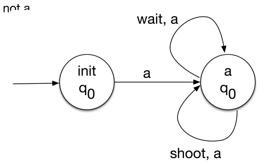

Figure 4: $\mathcal{A}_u$ after removing Win_{ag}, say $\mathcal{A}'_u$.

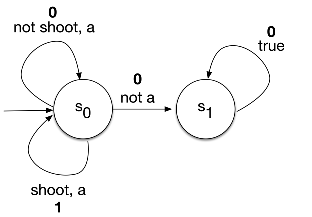

Figure 5: DPA for $\Box\Diamond shoot \rightarrow \Diamond(not\ a)$.

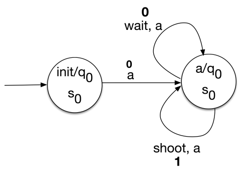

Figure 6: Parity Game obtained by cross product of the DPA and $\mathcal{A}'_u$

agent can force to reach the final states, in our case the final states themself. The agent winning strategy on Win_{ag} is defined as follows: $(not\ a, q_1) \rightarrow \{shoot, wait\}$ and $(Err, q_1) \rightarrow \{shoot, wait\}$. Since the initial state is not in Win_{ag}, we move to the stage 2.

At stage 2, we solve the parity game obtained by the product between the DPA for Env (Figure 5) and $\mathcal{A}_u$ after removing Win_{ag} (Figure 4), and we compute the winning states and the winning stratetegy for the agent. By looking at Figure 6, we can observe that all states are loosing for the environment. Indeed, in $a/q_0, s_0$ the agent can continuously choose shoot so that the smallest priority visited infinitely often is 1, that is odd. Thus, an agent winning strategy is defined as follows: $(init/q_0, s_0) \rightarrow \{wait, shoot\}$; $(a/q_0, s_0) \rightarrow shoot$. Finally, the winning strategy for the agent for the original problem is the combination of the strategies of the two stages as described in Section 5.

7 Experimental Analysis

We now examine the performance of our two-stage technique experimentally. We have implemented our two-stage algorithm (Algorithm 3) in a new tool called 2SLS, written in C++. 2SLS exploits the CUDD package as library for the manipulation of *Binary Decisions Diagrams* (BDDs) (Bryant 1986) used for the symbolic representations of DFAs and DPAs. Specifically, 2SLS takes in input an LTL$_f$ formula $Goal$, an LTL formula Env, the sets $\mathcal{X}$ and $\mathcal{Y}$ of input and output variables, respectively, and (*i*) it builds the symbolic DFA for $Goal$ borrowing the construction from Syft (Zhu et al. 2017), a tool for solving LTL$_f$ synthesis based on a symbolic approach. Syft exploits MONA (Henriksen et al. 1995) to build the DFA and then converts it into a symbolic representation. Then, (*ii*) it constructs the DPA for Env by using OWL (Kretínský, Meggendorfer, and Sickert 2018), a tool for translating LTL into different types of automata, constructs its symbolic representation. Finally, (*ii*) it executes Algorithm 3, which exploits APT as parity games solver (Di Stasio et al. 2016), and returns an agent winning strategy.

In order to evaluate the performance of 2SLS, we compare it to a direct reduction to LTL synthesis, which allows us to utilize state-of-the-art tools for standard LTL synthesis. Specifically, we have employed the LTL$_f$-to-LTL translator implemented in SPOT (Duret-Lutz et al. 2016) and chosen Strix (Meyer, Sickert, and Luttenberger 2018), the winner of the synthesis competition SYNTCOMP 2019 [6] over LTL synthesis track, as the LTL synthesis solver.

In addition, in special cases where assumptions are LTL formulas of the form $\Box\Diamond a$ (fairness) and $\Diamond\Box a$ (stability), with a propositional, we have also compared the performance of 2SLS with FSyft and StSyft (Zhu et al. 2020), tools for solving LTL synthesis with fairness and stability assumptions, respectively, which apply a BDD-based fixpoint-evaluation on the corresponding DFA of $Goal$ for checking the assumptions $\Box\Diamond a$ and $\Diamond\Box a$. FSyft and StSyft perform much better then Strix based on a direct reduction to LTL synthesis.

Experiment Setup. All tests were ran on a computer cluster. Each test took an exclusive access to a node with Intel(R) Xeon(R) CPU E5-2650 v2 processors running at 2.60GHz. Time out was set to 1000 seconds.

Experiments on Fairness and Stability. Specifically, we start by evaluating the performance of 2SLS on simple assumptions of the form $\Box\Diamond a$ and $\Diamond\Box a$, so as to compare it also with FSyft and StSyft. Actually (Zhu et al. 2020) adopt an ad-hoc technique to solve these types of assumptions and hence their technique is expected to perform significantly better. Instead, we show that our general technique performs comparably. This is a quite interesting outcome.

As a first benchmark, following (Zhu et al. 2020), we used problems generated from a scalable counter game, described as follows: (*i*) there is an n-bit binary. At each round, the environment chooses whether to increment the counter or not. The agent can choose to grant the request or ignore it; (*ii*) the goal is to get the counter having all bits set to 1, so the counter reaches the maximal value; (*iii*) the fairness assumption is to have the environment infinitely request the counter to be incremented; (*iv*) the stability assumption is to have the environment eventually keep requesting the counter to be incremented. We can easily reduce solving the counter game above to solving LTL$_f$ synthesis with LTL assumptions.

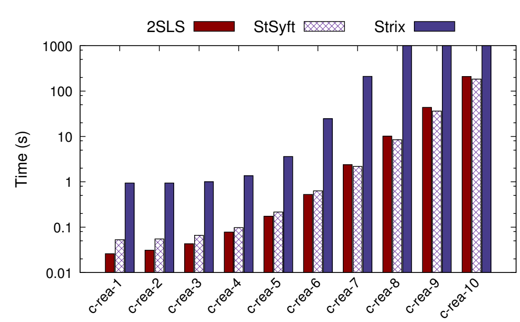

Figure 7: Stable LTL$_f$ synthesis. Comparison of running time among 2SLS, StSyft and Strix, in log scale.

We evaluated the efficiency of 2SLS in terms of the number of solved cases and total time cost expressed in seconds. Figure 8 and Figure 7 show the running time of the various

[6] http://www.syntcomp.org/syntcomp-2019-results/

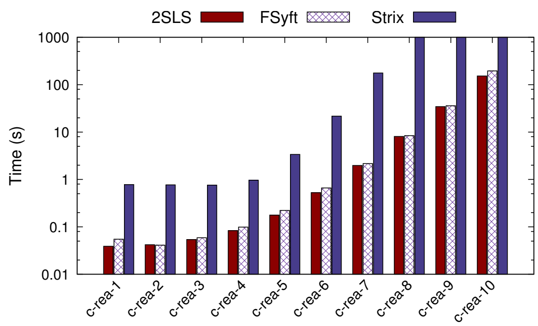

Figure 8: Fair LTL_f synthesis. Comparison of running time among 2SLS, FSyft and Strix, in log scale.

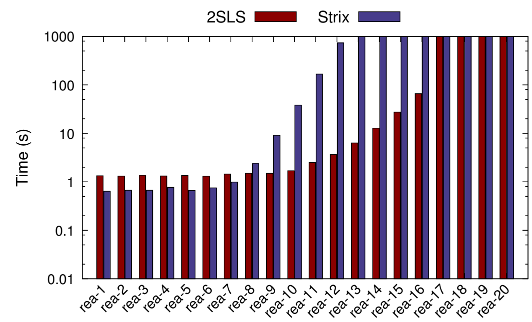

Figure 9: General LTL assumptions. Comparison of running time among 2SLS and Strix, in log scale.

tools. Here we only show realizable cases with counter bits $n \leq 10$. The x-labels *c-rea-n* indicate the realizability and the number of counter bits of each case. As expected, both of FSyft and StSyft perform much better than Strix, solving more cases in less running time. Interestingly, 2SLS is able to obtain comparable performance wrt FSyft when dealing with LTL_f synthesis under fairness assumptions, and even a slight advantage over StSyft for stability assumptions.

Experiments of General LTL Assumptions. Next we evaluate our approach with general LTL assumptions. We drop the comparison with FSyft and StSyft that do not support general assumptions, and focus instead on Strix, which is one of the best tools for LTL synthesis currently available. In particular, we translate LTL_f goals to LTL through SPOT, and use Strix on the resulting pure LTL formula.

To test the advantage of our two-stage technique it is important that both the LTL assumption and the LTL_f goal contribute to the synthesis. Indeed, in the limit case where we have only an LTL assumption with an empty LTL_f goal our technique would do nothing in the first stage and would simply solve classical LTL synthesis in the second stage.

Moreover, the part of assumption that specifies the possible transitions of the environment, which can be quite large as it happens for planning domains, can be expressed in LTL_f instead of LTL. Hence we are in the situation where the assumption Env is of the form $Env_\infty \wedge Env_f$ where Env_∞ is expressible in LTL and Env_f in LTL_f. In this case we can reduce $Env_\infty \wedge Env_f \rightarrow Goal$ to $Env_\infty \rightarrow Goal'$ where $Goal' = (Env_f \rightarrow Goal)$ as discussed in Section 6.

Based on these considerations we built benchmarks where the $Goal$ is a conjunction of increasing size of random LTL_f formulas of the form $\Box(p_j \rightarrow \Diamond q_j)$ with p_j and q_j propositions under the control of the environment and the agent, respectively; and the LTL assumption is a conjunction of formulas of the form $(\Box\Diamond p_i \vee \Diamond\Box q_i)$, where we start with one conjunct and introduce a new conjunct every 10 conjuncts in the $Goal$. Note that, formulas of the form $(\Box\Diamond p_i \vee \Diamond\Box q_i)$ are the most general kind of LTL formulas according to (Maler and Pnueli 1990).

As we see from Figure 9, when the entire formula is small the optimization of Strix makes it a little bit faster than 2SLS, but as the formula becomes larger than 8 conjuncts, 2SLS starts showing an exponential improvement with respect to Strix. This gives us an indication of the advantage of adapting our two-stage technique.

In spite the advantage over Strix, 2SLS also did not shine in this experiment. Indeed, Strix reaches this timeout with 13 conjuncts in the $Goal$ and 2 conjuncts in the LTL assumption, while 2SLS reaches the timeout with 17 conjuncts in the $Goal$ (and still 2 conjuncts in the LTL assumption). In fact, 2SLS reaches the timeout in building the DFA, a step performed by MONA. Specifically, the LTL_f formula is translated into first-order logic on finite sequences (Zhu et al. 2017) and then MONA is used as a black box to generate the (minimal) DFA. In such DFA states are represented explicit and transitions symbolically. Then, the DFA if further manipulated to have a full symbolic representation (this step is linear in the DFA). In this process the bottleneck is the construction of the DFA by MONA, which reaches the timeout because the intermediate steps of the construction of the DFA. This is the reason of the timeout we get for 17 conjuncts. There are recent techniques that aim at improving the construction of the DFA, see, e.g., (Bansal et al. 2020) and these could give a much higher scalability.

Planning domains. When dealing with nondeterministic planning domains as part of the assumption, an essential step is translating the domain expressed as PDDL into a symbolic DFA with maximum efficiency. While how to translate PDDL into DFA is well-known (De Giacomo and Rubin 2018), how to perform it in a practically efficient and optimized way is open to further investigation. A promising direction for the translation might be borrowing insights from reactive planning (He et al. 2019). Instead of having only agent actions with nondeterministic effects, the reactive planning domain has agent actions and environment actions. Agent and environment actions contrast each other in a turn-based fashion. We intend to look into this approach in the future.

8 Conclusion

In this paper, we have proposed a two-stage technique for synthesis in LTL_f under general LTL assumptions. The interesting aspect of out technique is that it confines the use of DPA, which is per se problematic, only when it is really necessary (stage 2). Experiments, although preliminary, show the effectiveness of our two-stage technique.

Acknowledgments

Work supported in part by European Research Council under the European Union's Horizon 2020 Programme through the ERC Advanced Grant WhiteMech (No. 834228), NSF grants IIS-1527668, CCF-1704883, IIS-1830549.

References

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2018. Synthesis under assumptions. In *KR*, 615–616. AAAI Press.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL environment specifications. In *ICAPS*, 31–39.

Babiak, T.; Blahoudek, F.; Duret-Lutz, A.; Klein, J.; Kretínský, J.; Müller, D.; Parker, D.; and Strejcek, J. 2015. The hanoi omega-automata format. In *CAV*, 479–486.

Bansal, S.; Li, Y.; Tabajara, L. M.; and Vardi, M. Y. 2020. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*.

Bloem, R.; Ehlers, R.; Jacobs, S.; and Könighofer, R. 2014. How to handle assumptions in synthesis. In *SYNT*.

Bonet, B.; De Giacomo, G.; Geffner, H.; and Rubin, S. 2017. Generalized planning: Non-deterministic abstractions and trajectory constraints. In *IJCAI*, 873–879.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691.

Calude, C. S.; Jain, S.; Khoussainov, B.; Li, W.; and Stephan, F. 2017. Deciding parity games in quasipolynomial time. In *STOC*, 252–263.

Camacho, A.; Baier, J. A.; Muise, C. J.; and McIlraith, S. A. 2018a. Finite LTL synthesis as planning. In *ICAPS*, 29–38.

Camacho, A.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2018b. LTL realizability via safety and reachability games. In *IJCAI*, 4683–4691.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2018. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, 454–463.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2019. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, 58–67. AAAI Press.

Chatterjee, K.; Henzinger, T. A.; and Jobstmann, B. 2008. Environment assumptions for synthesis. In *CONCUR*, 147–161.

Church, A. 1963. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.* 147(1-2):35–84.

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of FOND planning for LTLf and LDL_f goals. In *IJCAI*, 4729–4735.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 854–860.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*.

Di Stasio, A.; Murano, A.; Perelli, G.; and Vardi, M. Y. 2016. Solving parity games using an automata-based algorithm. In *CIAA 2016*, 64–76.

D'Ippolito, N.; Rodríguez, N.; and Sardiña, S. 2018. Fully observable non-deterministic planning as assumption-based reactive synthesis. *J. Artif. Intell. Res.* 61:593–621.

Duret-Lutz, A.; Lewkowicz, A.; Fauchille, A.; Michaud, T.; Renault, E.; and Xu, L. 2016. Spot 2.0 - A framework for LTL and ω -automata manipulation. In *ATVA*, 122–129.

Ehlers, R.; Lafortune, S.; Tripakis, S.; and Vardi, M. Y. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems* 27(2):209–260.

Esparza, J., and Kretínský, J. 2014. From LTL to deterministic automata: A safraless compositional approach. In *CAV*, 192–208.

Esparza, J.; Kretínský, J.; Raskin, J.; and Sickert, S. 2017. From LTL and limit-deterministic büchi automata to deterministic parity automata. In *TACAS*, 426–442.

Finkbeiner, B., and Schewe, S. 2013. Bounded Synthesis. *STTT* 15(5-6):519–539.

Finkbeiner, B. 2016. Synthesis of Reactive Systems. *Dependable Software Systems Eng.* 45:72–98.

Fogarty, S.; Kupferman, O.; Vardi, M. Y.; and Wilke, T. 2013. Profile Trees for Büchi Word Automata, with Application to Determinization. In *GandALF*, 107–121.

Geffner, H., and Bonet, B. 2013. *A Coincise Introduction to Models and Methods for Automated Planning*. Morgan&Claypool.

Gerstacker, C.; Klein, F.; and Finkbeiner, B. 2018. Bounded Synthesis of Reactive Programs. In *ATVA*, 441–457.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning – Theory and Practice*.

Giannakopoulou, D., and Lerda, F. 2002. From states to transitions: Improving translation of LTL formulae to büchi automata. In *FORTE*, 308–326.

Green, C. 1969. Theorem proving by resolution as basis for question-answering systems. In *Machine Intelligence*, volume 4. American Elsevier. 183–205.

Harding, A.; Ryan, M.; and Schobbens, P. 2005. A new algorithm for strategy synthesis in LTL games. In *TACAS*, 477–492.

He, K.; Wells, A. M.; Kavraki, L. E.; and Vardi, M. Y. 2019. Efficient symbolic reactive synthesis for finite-horizon tasks. In *ICRA*, 8993–8999.

Henriksen, J. G.; Jensen, J. L.; Jørgensen, M. E.; Klarlund, N.; Paige, R.; Rauhe, T.; and Sandholm, A. 1995. Mona: Monadic Second-order Logic in Practice. In *TACAS*, 89–110.

Jurdzinski, M. 1998. Deciding the winner in parity games is in UP \cap co-up. *Inf. Process. Lett.* 68(3):119–124.

Kreitz, C. 1998. Program synthesis. In Springer., ed., *Automated Deduction—A Basis for Applications*. 105–134.

Kretínský, J.; Meggendorfer, T.; Waldmann, C.; and Weininger, M. 2017. Index appearance record for transforming rabin automata into parity automata. In *TACAS*, 443–460.

Kretínský, J.; Meggendorfer, T.; and Sickert, S. 2018. Owl: A library for ω-words, automata, and LTL. In *ATVA*, 543–550.

Kupferman, O., and Vardi, M. Y. 2005. Safraless Decision Procedures. In *FOCS*, 531–542.

Maler, O., and Pnueli, A. 1990. Tight bounds on the complexity of cascaded decomposition of automata. In *FOCS 1990*, 672–682.

Manna, Z., and Pnueli, A. 1990. A hierarchy of temporal properties. In *PODC*, 377–410.

Martin, D. 1975. Borel Determinacy. *Annals of Mathematics* 65:363–371.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl – the planning domain definition language – version 1.2. Technical report, TR-98-003, Yale Center for Computational Vision and Contro.

Meyer, P. J.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit Reactive Synthesis Strikes Back! In *CAV*, 578–586.

Piterman, N. 2007. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science* 3(3).

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *POPL*.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*, 46–57.

Rabin, M. O., and Scott, D. S. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development* 3(2):114–125.

Safra, S. 1988. On the complexity of omega-automata. In *FOCS*, 319–327.

Sickert, S.; Esparza, J.; Jaax, S.; and Kretínský, J. 2016. Limit-deterministic büchi automata for linear temporal logic. In Chaudhuri, S., and Farzan, A., eds., *CAV*, 312–332.

Sohail, S., and Somenzi, F. 2009. Safety first: A two-stage algorithm for LTL games. In *FMCAD*, 77–84.

Vardi, M. Y. 2018. The Siren Song of Temporal Synthesis (Invited Talk). In *CONCUR*.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic ltlf synthesis. In *IJCAI*, 1362–1369.

Zhu, S.; De Giacomo, G.; Pu, G.; and Vardi, M. 2020. LTL_f synthesis with fairness and stability assumptions. In *AAAI*.

Zhu, S.; Pu, G.; and Vardi, M. Y. 2019. First-order vs. second-order encodings for LTLf -to-automata translation. In *TAMC*, 684–705.

Zielonka, W. 1998. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theor. Comput. Sci.* 200(1-2):135–183.