LTL_f Synthesis with Fairness and Stability Assumptions

Shufang Zhu,[1] **Giuseppe De Giacomo,**[2] **Geguang Pu,**[1] **Moshe Y. Vardi**[3]
[1]Each China Normal University, [2]Sapienza Università di Roma, [3]Rice University
shufangzhu.szhu@gmail.com, degiacomo@diag.uniroma1.it, ggpu@sei.ecnu.edu.cn, vardi@cs.rice.edu

Abstract

In synthesis, assumptions are constraints on the environment that rule out certain environment behaviors. A key observation here is that even if we consider systems with LTL_f goals on finite traces, environment assumptions need to be expressed over infinite traces, since accomplishing the agent goals may require an unbounded number of environment action. To solve synthesis with respect to finite-trace LTL_f goals under infinite-trace 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 2EXPTIME-complete), the algorithms available for LTL synthesis are much more difficult in practice than those for LTL_f synthesis. In this work we show that in interesting cases we can avoid such a detour to LTL synthesis and keep the simplicity of LTL_f synthesis. Specifically, we develop a BDD-based fixpoint-based technique for handling basic forms of fairness and of stability assumptions. We show, empirically, that this technique performs much better than standard LTL synthesis.

Introduction

In many situations we are interested in expressing properties over an unbounded but finite sequence of successive states. Linear-time Temporal Logic over finite traces (LTL_f) and its variants have been thoroughly investigated for doing so. There has been broad research for logical reasoning (De Giacomo and Vardi 2013; Li et al. 2019), synthesis (De Giacomo and Vardi 2015; Camacho et al. 2018), and planning (Camacho et al. 2017; De Giacomo and Rubin 2018).

Recently synthesis under assumptions in LTL_f has attracted specific interest (De Giacomo and Rubin 2018; Camacho, Bienvenu, and McIlraith 2018). First, planning for LTL_f goals can be considered as a form of LTL_f synthesis under assumptions, where the assumptions are the dynamics of the environment encoded in the planning domain (Green 1969; Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2018; 2019). However, more generally, assumptions can be arbitrary constraints on the environment that can be exploited by the agent in devising a strategy to fulfill its goal.

Synthesis under assumptions has been extensively studied in LTL, where environment assumptions are expressed as LTL formulas (Chatterjee and Henzinger 2007; Chatterjee, Henzinger, and Jobstmann 2008; D'Ippolito et al. 2013; Bloem, Ehlers, and Könighofer 2015; Brenguier, Raskin, and Sankur 2017). In fact, LTL formulas can be used as assumptions as long as it is guaranteed that the environment is able to behave so as to keep the assumptions true, i.e., assumptions are environment realizable. Under these circumstances, it is possible to reduce synthesis for LTL goal ψ_G under assumptions ψ_A to standard synthesis for $\psi_A \rightarrow \psi_G$. Note that because of the guarantee of ψ_A being environment realizable, no agent strategy can win $\psi_A \rightarrow \psi_G$ by falsifying ψ_A. See (Aminof et al. 2019) for a discussion.

When we turn to LTL_f, a key observation is that even if we consider (finite-trace) LTL_f goals for the agent, assumptions need to be expressed considering infinite traces, since accomplishing the agent goals may require an unbounded number of environment action. So we have an assumption ψ_A expressed in LTL and a goal ϕ_G expressed in LTL_f. To solve synthesis under assumptions in LTL_f, we could translate ϕ_G into LTL getting ψ_G, by applying the translation of LTL_f into LTL in (De Giacomo and Vardi 2013), and then do LTL synthesis for $\psi_A \rightarrow \psi_G$, see e.g. (Camacho, Bienvenu, and McIlraith 2018).

Unfortunately, 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 prohibitive for finding scalable implementations (Fogarty et al. 2013; Finkbeiner 2016). In spite of recent advancement 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. 2018), LTL synthesis remains very challenging. In contrast, LTL_f synthesis is based on a translation to Deterministic Finite Automaton (DFA) (Rabin and Scott 1959), which can be seen as a game arena where environment and agent make their own moves. On this arena, the agent wins if a simple fixpoint condition (reachability of the DFA accepting states) is satisfied.

Hence, when we introduce assumptions, an important question arises: can we retain the simplicity of LTL_f synthesis? In particular, we are thinking about algorithms based on devising some sort of arena and then extracting winning strategies by relying on computing a small number of nested fixpoints (note that the reduction of LTL synthesis to parity games may generate exponentially many nested fixpoints (Grädel, Thomas, and Wilke 2002)).

We consider here two different basic, but quite significant, forms of assumptions: a basic form of *fairness* $GF\alpha$ (always eventually α), and a basic form of *stability* $FG\alpha$ (eventually always α), where in both cases the truth value of α is under the control of the environment, and hence the assumptions are trivially realizable by the environment. Note that due to the existence of LTL_f goals, synthesis under both kinds of assumptions does not fall under known easy forms of synthesis, such as GR(1) formulas (Bloem et al. 2012). For these kinds of assumptions, we devise a specific algorithm based on using the DFA for the LTL_f goal as the arena and then computing 2-nested fixpoint properties over such arena. It should be noted that the kind of nested fixpoint that we compute for *fairness* $GF\alpha$ is similar to the one in (De Giacomo and Rubin 2018), but it is clear that the "fairness" stated there is different from what we claim in this paper. The "fairness" in (De Giacomo and Rubin 2018) is interpreted as all effects happening fairly, therefore the assumption is hardcoded in the arena itself. Here, instead, we only require that a selected condition α happens fairly, and our technique extends to deal with *stability* assumptions as well. We compare the new algorithm with standard LTL synthesis (Meyer, Sickert, and Luttenberger 2018) and show empirically that this algorithm performs significantly better, in the sense that solving more cases with less time cost. Some proofs have been removed due to the lack of space.[1]

Preliminaries

Linear-time Temporal Logic over finite traces (LTL_f) has the same syntax as LTL over infinite traces introduced in (Pnueli 1977). Given a set of propositions $\mathcal{P}$, the syntax of LTL_f formulas is defined as $\phi ::= a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 U \phi_2$. Every $a \in \mathcal{P}$ is an *atom*. A literal l is an atom or the negation of an atom. X for "Next", and U for "Until", are temporal operators. We make use of the standard Boolean abbreviations, such as $\vee$ (or) and $\rightarrow$ (implies), $true$ and $false$. Additionally, we define the following abbreviations "Weak Next" $X_w\phi \equiv \neg X \neg\phi$, "Eventually" $F\phi \equiv true U \phi$ and "Always" $G\phi \equiv false R \phi$, where R is for "Release".

A *trace* $\rho = \rho[0], \rho[1], \ldots$ is a sequence of propositional interpretations (sets), where $\rho[m] \in 2^{\mathcal{P}}$ ($m \geq 0$) is the m-th interpretation of ρ, and $|\rho|$ represents the length of ρ. Intuitively, $\rho[m]$ is interpreted as the set of propositions which are $true$ at instant m. Trace ρ is an *infinite* trace if $|\rho| = \infty$, which is formally denoted as $\rho \in (2^{\mathcal{P}})^{\omega}$; otherwise ρ is a *finite* trace, denoted as $\rho \in (2^{\mathcal{P}})^{*}$. LTL_f formulas are interpreted over finite, nonempty traces. Given a finite trace ρ and an LTL_f formula ϕ, we inductively define when ϕ is $true$ on ρ at step i ($0 \leq i < |\rho|$), written $\rho, i \models \phi$, as follows:

- $\rho, i \models a$ iff $a \in \rho[i]$;
- $\rho, i \models \neg\phi$ iff $\rho, i \not\models \phi$;
- $\rho, i \models \phi_1 \wedge \phi_2$ iff $\rho, i \models \phi_1$ and $\rho, i \models \phi_2$;
- $\rho, i \models X\phi$ iff $i + 1 < |\rho|$ and $\rho, i+1 \models \phi$;
- $\rho, i \models \phi_1 U \phi_2$ iff there exists j such that $i \leq j < |\rho|$ and $\rho, j \models \phi_2$, and for all k, $i \leq k < j$, we have $\rho, k \models \phi_1$.

An LTL_f formula ϕ is $true$ on ρ, denoted by $\rho \models \phi$, if and only if $\rho, 0 \models \phi$.

LTL_f *synthesis* can be viewed as a game of two players, the *environment* and the *agent*, contrasting each other. The aim is to synthesize a strategy for the agent such that no matter how the environment behaves, the combined behavior trace of both players satisfy the logical specification expressed in LTL_f (De Giacomo and Vardi 2015).

Fair and Stable LTL_f Synthesis

In this paper, we focus on LTL_f synthesis under assumptions in two different basic forms: fairness and stability, which we call in the following *fair* LTL_f *synthesis* and *stable* LTL_f *synthesis*, respectively. In such synthesis problems, both players (environment and agent) have Boolean variables under their respective control. Here, we use $\mathcal{X}$ to denote the set of environment variables that are uncontrollable for the agent, and $\mathcal{Y}$ the set of agent variables that are controllable for the agent. Therefore, $\mathcal{X}$ and $\mathcal{Y}$ are disjoint.

In general, assumptions are specific forms of constraints.

Definition 1 (Environment Constraint)**.** *An environment constraint α is a Boolean formula over $\mathcal{X}$.*

In particular, we define here two different basic, but common forms of assumptions.

Definition 2 (Fairness and Stability Assumptions)**.** *An* LTL *formula ψ is considered as a fairness assumption if it is of the form $GF\alpha$, and a stability assumption if of the form $FG\alpha$, where in both cases α is an environment constraint.*

A fair or stable trace can then be defined in terms of the corresponding assumption (fairness or stability).

Definition 3 (Fair and Stable Traces)**.** *A trace $\rho \in (2^{\mathcal{X} \cup \mathcal{Y}})^{\omega}$ is α-fair if $\rho \models GF\alpha$ and it is α-stable if $\rho \models FG\alpha$.*

Intuitively, α holds infinitely often on an α-fair trace, while eventually holds forever on an α-stable trace. Note that, if trace ρ is not α-fair, i.e., $\rho \not\models GF\alpha$, then $\rho \models FG(\neg\alpha)$ such that ρ is $\neg\alpha$-stable. Similarly, if trace ρ is not α-stable, i.e., $\rho \not\models FG\alpha$, then $\rho \models GF(\neg\alpha)$ such that ρ is $\neg\alpha$-fair. Although there is a duality between fairness and stability, such duality breaks when applying these environment assumptions to the problem of LTL_f synthesis. This is because in addition to the assumptions, the synthesis problems also require the LTL_f specification to be satisfied.

We now define fair and stable LTL_f synthesis by making use of fair and stable traces.

[1]A full version is available on arXiv. Geguang Pu is the corresponding author.

Definition 4 (Fair (Stable) LTL$_f$ Synthesis)**.** *LTL$_f$ formula ϕ, defined over $\mathcal{X} \cup \mathcal{Y}$, is α-fair (resp., α-stable) realizable if there exists a strategy $g : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$, such that for an arbitrary environment trace $\lambda = X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, if λ is α-fair (resp., α-stable), then we can find $k \geq 0$ such that ϕ is $true$ in the finite trace $\rho^k = (X_0 \cup g(X_0)), (X_1 \cup g(X_0, X_1)), \ldots, (X_k \cup g(X_0, X_1, \ldots, X_k))$.*

A fair (resp., stable) LTL$_f$ synthesis problem, described as a tuple $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$, consist in checking whether ϕ, defined over $\mathcal{X} \cup \mathcal{Y}$, is α-fair (resp., α-stable) realizable. The synthesis procedure aims to computing a strategy if realizable.

Intuitively speaking, ϕ describes the desired goal when the environment behaviors satisfy the assumption. An agent strategy $g : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$ for fair (resp., stable) synthesis problem $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ is *winning* if it guarantees the satisfaction of the objective ϕ under the condition that the environment behaves in a way that α holds infinitely often (resp., α eventually holds forever). The *realizability procedure* of $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ aims to answer the existence of a winning strategy g and the *synthesis procedure* amounts for computing g if it exists. In fact one can consider two variants of the synthesis problem, depending on the player who moves first, in the sense of assigning values to variables under its control first. Here we consider the environment as the first-player (as in planning), but a version where the agent moves first can be obtained by a small modification.

Since every LTL$_f$ formula ϕ can be translated to a Deterministic Finite Automaton (DFA) $\mathcal{G}_\phi$ that accepts exactly the same language as ϕ (De Giacomo and Vardi 2013), we are able to reduce the problems of fair LTL$_f$ synthesis and stable LTL$_f$ synthesis to specific two-player DFA games, in particular, fair DFA game and stable DFA game, respectively. We start with introducing DFA games.

Games over DFA

Two-player games on DFA are games consisting of two players, the *environment* and the *agent*. $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of environment Boolean variables and agent Boolean variables, respectively. The *specification* of the game arena is given by a DFA $\mathcal{G} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s_0, \delta, Acc)$, where $2^{\mathcal{X} \cup \mathcal{Y}}$ is the alphabet, S is a set of states, $s_0 \in S$ is an initial state, $\delta : S \times 2^{\mathcal{X} \cup \mathcal{Y}} \to S$ is a transition function and $Acc \subseteq S$ is a set of accepting states.

A *round* of the game consists of both players setting the values of variables under their respective control. A *play* ρ over $\mathcal{G}$ records how two players set the values at each round and how the DFA proceed according to the values. Formally, a play ρ from state s_{i_0} is an infinite trace $(s_{i_0}, X_0 \cup Y_0), (s_{i_1}, X_1 \cup Y_1) \ldots \in (S \times 2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ such that $s_{i_{j+1}} = \delta(s_{i_j}, X_j \cup Y_j)$. Moreover, we also assign the *environment* as the first-player, which sets values first.

A play ρ is considered as a winning play if it follows a certain winning condition. Different winning conditions lead to different games. In this paper, we consider two specific two-player games, fair DFA game and stable DFA game, both of which are described as $\langle \mathcal{G}, \alpha \rangle$, where $\mathcal{G}$ is the game arena and α is the environment constraint.

Fair DFA Game. Although the ultimate goal for solving a fair DFA game is to perform winning plays for the agent, since it is more straightforward to formulate the game considering the environment as the protagonist, we first define the winning condition of the environment over a play. A play $\rho = (s_{i_0}, X_0 \cup Y_0), (s_{i_1}, X_1 \cup Y_1) \ldots$ over $\mathcal{G}$ is *winning* for the *environment* with respect to a fair DFA game $\langle \mathcal{G}, \alpha \rangle$ if the following two conditions hold:

• *Recurrence*: ρ is α-fair (that is, $\rho \models GF\alpha$),
• *Safety*: $s_{i_j} \notin Acc$ for all $j \geq 0$ (Acc is avoided).

Consequently, a play ρ is *winning* for the *agent* if one of the following conditions holds:

• *Stability*: ρ is not α-fair (that is, $\rho \models FG(\neg\alpha)$),
• *Reachability*: $s_{i_j} \in Acc$ for some $j \geq 0$ (Acc is reached).

Stable DFA Game. As for a stable DFA game $\langle \mathcal{G}, \alpha \rangle$, a play $\rho = (s_{i_0}, X_0 \cup Y_0), (s_{i_1}, X_1 \cup Y_1) \ldots$ over $\mathcal{G}$ is *winning* for the *environment* if the following two conditions hold:

• *Stability*: ρ is α-stable (that is, $\rho \models FG\alpha$),
• *Safety*: $s_{i_j} \notin Acc$ for all $j \geq 0$ (Acc is avoided).

Consequently, a play ρ is *winning* for the *agent* if one of the following conditions holds:

• *Recurrence*: ρ is not α-stable (that is, $\rho \models GF(\neg\alpha)$),
• *Reachability*: $s_{i_j} \in Acc$ for some $j \geq 0$ (Acc is reached).

Since we consider here the environment as the first-player, a strategy for the agent is a function $g : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$, deciding the values of the controllable variables for every possible history of the uncontrollable variables. Respectively, an environment strategy is a function $h : (2^{\mathcal{Y}})^* \to 2^{\mathcal{X}}$. A play $\rho = (s_{i_0}, X_0 \cup Y_0), (s_{i_1}, X_1 \cup Y_1) \ldots \in (S \times 2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ *follows* a strategy g (resp., a strategy h), if $Y_j = g(X_0, \ldots, X_j)$ for all $j \geq 0$ (resp., $X_j = h(Y_0, \ldots, Y_{j-1})$ for all $j > 0$).

We can now define winning states and winning strategies.

Definition 5 (Winning State and Winning Strategy)**.** *In the game $\langle \mathcal{G}, \alpha \rangle$ described above, $s \in S$ is a winning state for the agent (resp., environment) if there exists strategy g (resp., h) s.t. every play ρ from s that follows g (resp., h) is an agent (resp., environment) winning play. Then g (resp., h) is a winning strategy for the agent (resp., environment) from s.*

As shown in (Martin 1975), both of the fair DFA game and stable DFA game described above are *determined*, that is, a state $s \in S$ is a winning state for the agent if and only if s is not a winning state for the environment. The *realizability* procedure of the game consists of checking whether there exists a winning strategy for the agent from initial state s_0. The *synthesis* procedure aims to computing such a strategy.

We then show how to reduce the problems of fair LTL$_f$ synthesis and stable LTL$_f$ synthesis to fair DFA game and stable DFA game, respectively. Hence we can solve the DFA game, thus settling the corresponding synthesis problem.

Solution to Fair LTL$_f$ Synthesis

In order to perform fair synthesis on LTL$_f$, given problem $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$, we first translate the LTL$_f$ specification ϕ into a DFA $\mathcal{G}_\phi$. We then view $\langle \mathcal{G}_\phi, \alpha \rangle$ as a fair DFA game, and consider exactly the separation between environment and agent variables as in the original synthesis problem. Specifically, we assign $\mathcal{X}$ as the environment variables and $\mathcal{Y}$ as the agent variables. Finally, we solve the fair DFA game, thus settling the fair LTL$_f$ synthesis problem. The following theorem assesses the correctness of this technique.

Theorem 1. *Fair* LTL$_f$ *synthesis problem* $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ *is realizable iff fair* DFA *game* $\langle \mathcal{G}_\phi, \alpha \rangle$ *is realizable.*

Proof. We prove the theorem in both directions.
$\leftarrow$: Since $\langle \mathcal{G}_\phi, \alpha \rangle$ is realizable for the agent, the initial state s_0 is an agent winning state with winning strategy $g : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$. Therefore, a play $\rho = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots$ over $\mathcal{G}_\phi$ from s_0 following g is a winning play for the agent. Moreover, for every such play ρ from s_0, either of the following conditions holds:
- $\rho \not\models GF\alpha$ such that ρ is not α-fair.
- $\rho \models GF\alpha$ such that ρ is α-fair. Since ρ is winning for the agent, there exists $j \geq 0$ such that $s_j \in Acc$. This implies that $\rho^j \models \phi$ holds, where $\rho^j = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots, (s_j, X_j \cup g(X_0, X_1, \ldots, X_j))$.

Consequently, the strategy g assures that for an arbitrary environment trace $\lambda = X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, if λ is α-fair, then there is $j \geq 0$ such that ϕ is $true$ in the finite trace ρ^j. We conclude that $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ is realizable.
$\rightarrow$: For this direction, we assume that $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ is realizable, then there exists a strategy $g : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$ that realizes ϕ. Thus consider an arbitrary environment trace $\lambda \in (2^{\mathcal{X}})^\omega$, either of the following conditions holds:
- λ is not α-fair, then the induced play $\rho = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots$ over $\mathcal{G}_\phi$ from s_0 that follows g is winning for the agent by default.
- λ is α-fair, then on the induced play $\rho = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots$ over $\mathcal{G}_\phi$ from s_0, there exists $j \geq 0$ such that ϕ is $true$ in the finite trace $\rho^j = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots, (s_j, X_j \cup g(X_0, X_1, \ldots, X_j))$, in which case $s_j \in Acc$. Therefore, ρ is winning for the agent.

Consequently, we conclude that fair DFA game $\langle \mathcal{G}_\phi, \alpha \rangle$ is realizable for the agent. $\square$

Fair DFA Game Solving

Winning fair DFA games means that the agent can eventually reach an "agent wins" region from which if the constraint α holds, then it is possible to reach an accepting state. Given a fair DFA game $\langle \mathcal{G}, \alpha \rangle$, we proceed as follows: (1) Compute "agent wins" region in fair DFA game $\langle \mathcal{G}, \alpha \rangle$; (2) Check realizability; (3) Return an agent winning strategy if realizable.

Since the environment winning condition is more intuitive, in order to show the solution to fair DFA game, we start by solving the *Recurrence-Safety* game, which considers the environment as the protagonist. The idea for winning such game is that the environment should remain in an "environment wins" region from which the constraint α holds infinitely often referring to *Recurrence* game, meanwhile the accepting states are forever avoidable referring to *Safety* game. Therefore, in order to have both of *Recurrence* such that having $GF\alpha$ holds and *Safety* such that avoiding accepting states $s \in Acc$, the "environment wins" region computation is defined as:

$Env_f = \nu Z.\mu \hat{Z}.(\exists X.\forall Y.((X \models \alpha \wedge \delta(s, X \cup Y) \in Z \backslash Acc) \vee \delta(s, X \cup Y) \in \hat{Z} \backslash Acc))$,

where X ranges over $2^{\mathcal{X}}$ and Y over $2^{\mathcal{Y}}$.

The fixpoint stages for Z (note $Z_{i+1} \subseteq Z_i$, for $i \geq 0$, by monotonicity) are:
- $Z_0 = S$,
- $Z_{i+1} = \mu \hat{Z}.(\exists X.\forall Y.((X \models \alpha \wedge \delta(s, X \cup Y) \in Z_i \backslash Acc) \vee \delta(s, X \cup Y) \in \hat{Z} \backslash Acc))$.

Eventually, $Env_f = Z_k$ for some k such that $Z_{k+1} = Z_k$.

The fixpoint stages for $\hat{Z}$ with respect to Z_i (note $\hat{Z}_j \subseteq \hat{Z}_{j+1}$, for $j \geq 0$, by monotonicity) are:
- $\hat{Z}_{i,0} = \emptyset$,
- $\hat{Z}_{i,j+1} = \exists X.\forall Y.((X \models \alpha \wedge \delta(s, X \cup Y) \in Z_i \backslash Acc) \vee \delta(s, X \cup Y) \in \hat{Z}_{i,j} \backslash Acc)$.

Finally, $\hat{Z}_i = \hat{Z}_{i,k}$ for some k such that $\hat{Z}_{i,k+1} = \hat{Z}_{i,k}$.

The following theorem assures that the nested fixpoint computation of Env_f collects exactly all environment winning states in fair DFA game.

Theorem 2. *For a fair* DFA *game* $\langle \mathcal{G}, \alpha \rangle$ *and a state* $s \in S$, *we have* $s \in Env_f$ *iff* s *is an environment winning state.*

Proof. We prove the two directions separately.
$\leftarrow$: We prove by showing the contrapositive. If a state $s \notin Env_f$, then s must be removed from Env_f at stage $i+1$, therefore, $s \in Z_i \backslash Z_{i+1}$. Then $s \notin \mu \hat{Z}.(\exists X.\forall Y.((X \models \alpha \wedge \delta(s, X \cup Y) \in Z \backslash Acc) \vee \delta(s, X \cup Y) \in \hat{Z} \backslash Acc))$. That is, no matter what the (environment) strategy h is, traces from s satisfy **neither** of the following conditions:
- α holds and the trace gets trapped in Z without visiting accepting states such that $X \models \alpha \wedge \delta(s, X \cup Y) \in Z \backslash Acc$ holds, in which case s is a new environment winning state;
- α eventually gets hold and from there we can have α as true infinitely often without visiting accepting states such that $\delta(s, X \cup Y) \in \hat{Z} \backslash Acc$ holds, in which case s is a new environment winning state.

Therefore, s is not an environment winning state. So if s is an environment winning state then $s \in Env_f$ holds.
$\rightarrow$: If a state $s \in Env_f$, then $s \in \mu \hat{Z}.(\exists X.\forall Y.((X \models \alpha \wedge \delta(s, X \cup Y) \in Z \backslash Acc) \vee \delta(s, X \cup Y) \in \hat{Z} \backslash Acc))$. That is, no matter what the (agent) strategy g is, traces from s satisfy either of the following conditions:
- α holds and the trace gets trapped in Z without visiting accepting states such that $X \models \alpha \wedge \delta(s, X \cup Y) \in Z \backslash Acc$ holds, in which case s is a new environment winning state;
- α eventually gets hold and from there we can have α as true infinitely often without visiting accepting states such that $\delta(s, X \cup Y) \in \hat{Z} \backslash Acc$ holds, in which case s is a new environment winning state.

Thus s is a winning state for the environment. $\square$

Due to the determinacy of fair DFA game, the set of agent winning states Sys_f can be computed by negating Env_f:
$Sys_f = \mu Z.\nu \hat{Z}.(\forall X.\exists Y.((X \models \neg\alpha \vee \delta(s, X \cup Y) \in Z \cup Acc) \wedge \delta(s, X \cup Y) \in \hat{Z} \cup Acc))$.

Theorem 3. *A fair* DFA *game* $\langle \mathcal{G}, \alpha \rangle$ *has an agent winning strategy if and only if* $s_0 \in Sys_f$.

Strategy Extraction

Having completed the realizability checking procedure, this section deals with the agent winning strategy generation if $\langle \mathcal{G}, \alpha \rangle$ is realizable. It is known that if some strategy that realizes ϕ exists, then there also exists a *finite-state strategy* generated by a finite-state *transducer* that realizes ϕ (Buchi and Landweber 1990). Formally, the agent *winning strategy* $g : (2^{\mathcal{X}})^+ \rightarrow 2^{\mathcal{Y}}$ can be represented as a deterministic finite transducer based on the set Sys_f, described as below.

Definition 6 (Deterministic Finite Transducer)**.** *Given a fair* DFA *game $\langle \mathcal{G}, \alpha \rangle$, where $\mathcal{G} = (2^{\mathcal{X} \cup \mathcal{Y}}, S, s_0, \delta, Acc)$, a deterministic finite transducer $\mathcal{T} = (2^{\mathcal{X}}, 2^{\mathcal{Y}}, Q, s_0, \varrho, \omega_f)$ of such game is defined as follows:*
- *$Q \subseteq S$ is the set of agent winning states s.t. $Q = Sys_f$;*
- *$\varrho : Q \times 2^{\mathcal{X}} \rightarrow Q$ is the transition function such that $\varrho(q, X) = \delta(q, X \cup Y)$ and $Y = \omega_f(q, X)$;*
- *$\omega_f : Q \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ is the output function such that at an agent winning state q with assignment X, $\omega_f(q, X)$ returns an assignment Y leading to an agent winning play.*

The transducer $\mathcal{T}$ generates g in the sense that for every $\lambda \in (2^{\mathcal{X}})^\omega$, we have $g(\lambda) = \omega_f(\varrho(\lambda))$, with the usual extension of δ to words over $2^{\mathcal{X}}$ from s_0. Note that there are many possible choices for the output function ω_f. The transducer $\mathcal{T}$ defines a winning strategy by restricting ω_f to return only one possible setting of $\mathcal{Y}$.

We extract the output function $\omega_f : Q \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ for the game from the approximates for Z assuming $\hat{Z}$ to be Sys_f, from where no matter what the environment strategy is, traces have to always get $\neg\alpha$ hold. Thus, we consider: $\mu Z.(\forall X.\exists Y.((X \models \neg\alpha \vee \delta(s, X \cup Y) \in Z \cup Acc) \wedge \delta(s, X \cup Y) \in Sys_f \cup Acc))$ with approximates defined as:
- $Z_0 = \emptyset$,
- $Z_{i+1} = \forall X.\exists Y.((X \models \neg\alpha \vee \delta(s, X \cup Y) \in Z_i \cup Acc) \wedge \delta(s, X \cup Y) \in Sys_f \cup Acc)$.

Define an output function $\omega_f : Sys_f \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ as follows: for $s \in Z_{i+1} \backslash Z_i$, for all possible values $X \in 2^{\mathcal{X}}$, set Y to be such that $(X \models \neg\alpha \vee \delta(s, X \cup Y) \in Z_i \cup Acc) \wedge \delta(s, X \cup Y) \in Sys_f \cup Acc$ holds for $s \notin Acc$. Consider a deterministic finite transducer $\mathcal{T}$ defined in the sense that constructing ω_f as described above, the following theorem guarantees that $\mathcal{T}$ generates an agent winning strategy g.

Theorem 4. *Strategy g with $g(\lambda) = \omega_f(\varrho(\lambda))$ is a winning strategy for the agent.*

Solution to Stable LTL$_f$ Synthesis

Solving stable LTL$_f$ synthesis problem $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ relies on solving the stable DFA game $\langle \mathcal{G}_\phi, \alpha \rangle$, where $\mathcal{G}_\phi$ is the corresponding DFA of ϕ. The following theorem guarantees the correctness of such reduction.

Theorem 5. *Stable* LTL$_f$ *synthesis problem $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ is realizable iff stable* DFA *game $\langle \mathcal{G}_\phi, \alpha \rangle$ is realizable.*

Proof. We prove the theorem in both directions.
$\leftarrow$: Since $\langle \mathcal{G}_\phi, \alpha \rangle$ is realizable for the agent, the initial state s_0 is an agent winning state with winning strategy $g : (2^{\mathcal{X}})^+ \rightarrow 2^{\mathcal{Y}}$. Therefore, a play $\rho = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots$ over $\mathcal{G}_\phi$ from s_0 following g is a winning play for the agent. Moreover, for every such play ρ from s, either of the following conditions holds:
- $\rho \not\models FG\alpha$ such that ρ is not α-stable.
- $\rho \models FG\alpha$ such that ρ is α-stable. Since ρ is winning for the agent, there exists $j \geq 0$ such that $s_j \in Acc$. Therefore, $\rho^j \models \phi$ holds, where $\rho^j = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots, (s_j, X_j \cup g(X_0, X_1, \ldots, X_j))$.

Consequently, the strategy g assures that for an arbitrary environment trace $\lambda = X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, if λ is α-stable, then there exists $j \geq 0$ such that ϕ is $true$ in finite trace ρ^j. Thus $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ is realizable.
$\rightarrow$: For this direction, we assume that $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ is realizable, then there exists a strategy $g : (2^{\mathcal{X}})^+ \rightarrow 2^{\mathcal{Y}}$ that realizes ϕ. Thus consider an arbitrary environment trace $\lambda \in (2^{\mathcal{X}})^\omega$, either of the following conditions holds:
- λ is not α-stable, then the induced play $\rho = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots$ over $\mathcal{G}_\phi$ from s_0 that follows g is winning for the agent by default.
- λ is α-stable, then on the induced play $\rho = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots$ over $\mathcal{G}_\phi$ from s_0, there exists $j \geq 0$ such that ϕ is $true$ in the finite trace $\rho^j = (s_0, X_0 \cup g(X_0)), (s_1, X_1 \cup g(X_0, X_1)), \ldots, (s_j, X_j \cup g(X_0, X_1, \ldots, X_j))$, in which case $s_j \in Acc$. Therefore, ρ is winning for the agent.

Consequently, we conclude that stable DFA game $\langle \mathcal{G}_\phi, \alpha \rangle$ is realizable for the agent. $\square$

Stable DFA Game Solving

Despite the duality between fairness and stability, solving the stable DFA game here cannot directly dualize the solution to fair DFA game. This is because the computation here involves a *Stability-Safety* game, which is not dual to the *Recurrence-Safety* game in fair DFA game solving. In order to deal with stable DFA game, we again first consider the environment as the protagonist. We compute the set of environment winning states as follows:
$Env_{st} = \mu Z.\nu \hat{Z}.(\exists X.\forall Y.((X \models \alpha \wedge \delta(s, X \cup Y) \in \hat{Z} \backslash Acc) \vee \delta(s, X \cup Y) \in Z \backslash Acc))$,
where X ranges over $2^{\mathcal{X}}$ and Y over $2^{\mathcal{Y}}$.

The following theorem assures that the nested fixpoint computation of Env_{st} collects exactly all environment winning states in stable DFA game.

Theorem 6. *For a stable* DFA *game $\langle \mathcal{G}, \alpha \rangle$ and a state $s \in S$, we have $s \in Env_{st}$ iff s is an environment winning state.*

Correspondingly, since stable DFA game is determined, the set of agent winning states can be computed as follows:
$Sys_{st} = \nu Z.\mu \hat{Z}.(\forall X.\exists Y.((X \models \neg\alpha \vee \delta(s, X \cup Y) \in \hat{Z} \cup Acc) \wedge \delta(s, X \cup Y) \in Z \cup Acc))$.

Theorem 7. *A stable* DFA *game $\langle \mathcal{G}, \alpha \rangle$ has an agent winning strategy if and only if $s_0 \in Sys_{st}$.*

Strategy Extraction

Here, the agent *winning strategy* $g : (2^{\mathcal{X}})^+ \rightarrow 2^{\mathcal{Y}}$ can also be represented as a deterministic finite transducer $\mathcal{T} = (2^{\mathcal{X}}, 2^{\mathcal{Y}}, Q, s_0, \varrho, \omega_{st})$ in terms of the set of agent winning states such that $Q = Sys_{st}$.

We extract the output function $\omega_{st} : Q \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ for the game from the approximates for Z assuming $\hat{Z}$ to be Sys_{st}, from where no matter what the environment strategy is, traces cannot always get α hold. Thus, we consider the fixpoint computation as follows:

$\nu Z.(\forall X.\exists Y.((X \models \neg\alpha \vee \delta(s, X \cup Y) \in Sys_{st} \cup Acc) \wedge \delta(s, X \cup Y) \in Z \cup Acc))$.

Define an output function $\omega_{st} : Sys_{st} \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ s.t. for $s \in Z_{i+1} \cap Z_i$, for all possible values $X \in 2^{\mathcal{X}}$, set Y to be s.t. $(X \models \neg\alpha \vee \delta(s, X \cup Y) \in Sys_{st} \cup Acc) \wedge \delta(s, X \cup Y) \in Z_i \cup Acc$ holds for $s \notin Acc$. The following theorem guarantees that $\mathcal{T}$ generates an agent winning strategy g.

Theorem 8. *Strategy g with $g(\lambda) = \omega_{st}(\varrho(\lambda))$ is a winning strategy for the agent.*

Evaluation

We observe that a straightforward approach to LTL$_f$ synthesis under assumptions can be obtained by a reduction to standard LTL synthesis, which allows us to utilize tools for LTL synthesis to solve the fair (or stable) LTL$_f$ synthesis problem. In this section, we first revisit the reduction to standard LTL synthesis, and then show an experimental comparison with the approach proposed earlier in this paper.

Reduction to LTL Synthesis. The insight of reducing LTL$_f$ synthesis under assumptions to LTL synthesis comes from the reduction in (Zhu et al. 2017b) for general LTL$_f$ synthesis, and in (Camacho, Bienvenu, and McIlraith 2018) for constraint LTL$_f$ synthesis, where the constraint describes the desired environment behaviors, under which the goal is to satisfy the given LTL$_f$ specification. Both reductions adopt the translation rules in (De Giacomo and Vardi 2013) to polynomially transform an LTL$_f$ formula ϕ over $\mathcal{X} \cup \mathcal{Y}$ into an LTL formula ψ over $\mathcal{X} \cup \mathcal{Y} \cup \{alive\}$, retaining the satisfiability equivalence, where proposition $alive$ indicates the last instance of the finite trace. Such translation bridges the gap between LTL$_f$ over finite traces and LTL over infinite traces. Based on the translation from LTL$_f$ to LTL, we then reduce fair (resp., stable) LTL$_f$ synthesis problem $\langle \mathcal{X}, \mathcal{Y}, \alpha, \phi \rangle$ to LTL synthesis problem $\langle \mathcal{X}, \mathcal{Y} \cup \{alive\}, GF\alpha \to \psi \rangle$ (resp., $\langle \mathcal{X}, \mathcal{Y} \cup \{alive\}, FG\alpha \to \psi \rangle$).

Implementation. Based on the LTL$_f$ synthesis tool *Syft* [2], we implemented our fixpoint-based techniques for solving fair LTL$_f$ synthesis and stable LTL$_f$ synthesis in two tools called *FSyft* and *StSyft*, respectively (name after *Syft*). Both frameworks consist of two steps: the symbolic DFA construction and the respective DFA game solving. In the first step, we based on the code of *Syft*, to construct the symbolic DFA represented in Binary Decision Diagrams (BDDs). The implementation of the nested fixpoint computation for solving DFA games over such symbolic DFA, borrows techniques from (Zhu et al. 2017a) for greatest fixpoint computation and from *Syft* for least fixpoint computation. The construction of the transducer for generating the winning strategy utilizes the boolean-synthesis procedure introduced in (Fried, Tabajara, and Vardi 2016) for realizable formulas. The implementation makes use of the BDD library CUDD-3.0.0 (Somenzi 2016). In order to evaluate the performance of *FSyft* and *StSyft*, we compared it against the solution of reducing to standard LTL synthesis shown above. For such comparison, we employed the LTL$_f$-to-LTL translator implemented in SPOT (Duret-Lutz et al. 2016) and chose Strix (Meyer, Sickert, and Luttenberger 2018), the winner of the synthesis competition SYNTCOMP 2019 [3] over LTL synthesis track, as the baseline.

[2]https://github.com/saffiepig/Syft

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

Experimental Methodology

Benchmarks. We collected 1200 formulas consisting of two classes of benchmarks: 1000 randomly conjuncted LTL$_f$ formulas over 100 basic cases, generated in the style described in (Zhu et al. 2017b), the length of which, indicating the number of conjuncts, ranges form 1 to 5. The assumption (either fairness or stability) is assigned by randomly selecting one variable from all environment variables; 200 LTL$_f$ synthesis benchmarks with assumptions generated from a scalable counter game, described as follows:

• There is an n-bit binary counter. At each round, the environment chooses whether to increment the counter or not. The agent can choose to grant the request or ignore it.
• The goal is to get the counter having all bits set to 1, so the counter reaches the maximal value.
• The fairness assumption is to have the environment infinitely request the counter to be incremented.
• The stability assumption is to have the environment eventually keep requesting the counter to be incremented.

We reduce solving the counter game above to solving LTL$_f$ synthesis with assumptions. First, we have n agent variables $\{b_{n-1}, b_{n-2}, \ldots, b_0\}$ denoting the value of n counter bits. We also introduce another $n+1$ agent variables $\{c_n, c_{n-1}, \ldots, c_0\}$ representing the carry bits. In addition, we have an environment variable add representing the environment making an increment request or not, and c_0 as $true$ is considered as the agent granting the request. We then formulate the counter game into LTL$_f$ formula as follows:
$Init = ((\neg c_0) \wedge \ldots \wedge (\neg c_{n-1}) \wedge (\neg b_0) \wedge \ldots (\neg b_{n-1})))$,
$Goal = F(b_0 \wedge \ldots \wedge b_{n-1})$,
$B = G((\neg add) \to X_w(\neg c_0))$,

$$B_i = \begin{cases} (((\neg c_i) \wedge (\neg b_i)) \to X_w((\neg b_i) \wedge (\neg c_{i+1}))) \\ (((\neg c_i) \wedge b_i) \to X_w(b_i \wedge (\neg c_{i+1}))) \\ (((c_i \wedge \neg b_i) \to X_w(b_i \wedge (\neg c_{i+1}))) \\ (((c_i \wedge b_i) \to X_w((\neg b_i) \wedge c_{i+1}))). \end{cases}$$

The LTL$_f$ formula ϕ is then $(Init \wedge B \wedge \bigwedge_{0 \le i \le n} G(B_i)) \wedge Goal$, and the constraint α is add. Obviously, such counter game only returns realizable cases, since a winning strategy for the agent is to grant all increment requests.

In order to get unrealizable cases, we can make some modifications on the counter game above. One possibility is to have the counter increment by 2 if the agent chooses to grant the request sent by the environment. Such modification leads to no winning strategy for the agent, since the maximal counter value of having each bit as 1 is odd. However, incrementing by 2 at each time will never reach an odd value. Therefore, for bit B_i such that $i > 0$, we keep the same formulation. While for bit B_0, we change as follows:

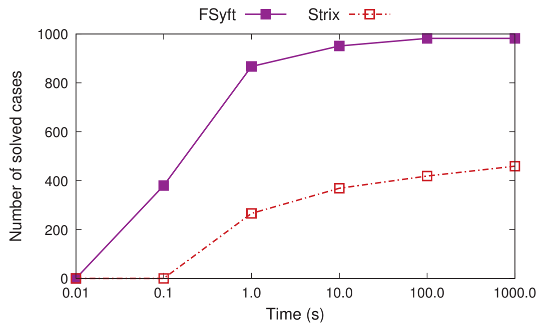

Figure 1: Fair LTL_f synthesis. Comparison of the number of solved cases with limited time between *FSyft* and Strix over random conjunction benchmarks.

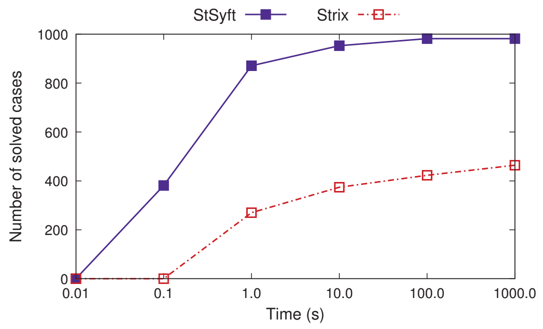

Figure 2: Stable LTL_f synthesis. Comparison of the number of solved cases with limited time between *StSyft* and Strix over random conjunction benchmarks.

$$B_0 = \begin{cases} (((\neg c_0) \wedge (\neg b_0)) \to X_w((\neg b_0) \wedge (\neg c_1))) \\ (((\neg c_0) \wedge b_0) \to X_w(b_0 \wedge (\neg c_1))) \\ (((c_0 \wedge \neg b_0) \to X_w(\neg b_0 \wedge (c_1))) \\ (((c_0 \wedge b_0) \to X_w((b_0) \wedge c_1))). \end{cases}$$

Therefore, we have 200 counter game benchmarks in total, with the number of counter bits n ranging from 1 to 100, and both realizable and unrealizable cases for each n.

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.

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

Experimental Results.

We evaluated the efficiency of *FSyft* and *StSyft* in terms of the number of solved cases and total time cost. We compared these two tools against Strix by performing an end-to-end comparison experiment. Therefore, both of the DFA construction time and the fixpoint computation time were counted for *FSyft* and *StSyft*. For Strix, we counted the running time from feeding the corresponding LTL formula to Strix to receiving the result. Both comparison on two classes of benchmarks show the advantage of the fixpoint-based

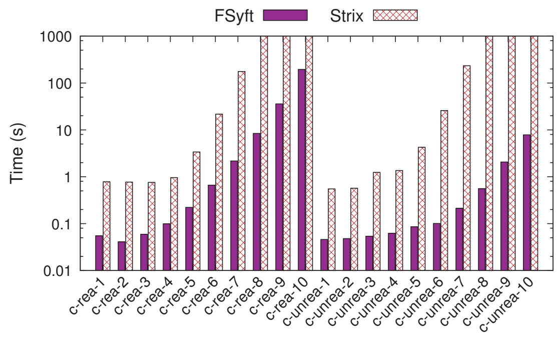

Figure 3: Fair LTL_f synthesis. Comparison of running time between *FSyft* and Strix, in log scale. Bars of the maximum height indicate cases timed out.

technique proposed in this paper as an effective method for both of fair LTL_f synthesis and stable LTL_f synthesis [4].

Randomly Conjuncted Benchmarks. Figure 1 and Figure 2 show the number of solved cases as the given time increases on fair LTL_f synthesis and stable LTL_f synthesis, respectively. As shown in the figures, both of *FSyft* and *StSyft* are able to handle almost all cases (1000 in total for each), while Strix only solves a small fraction of the cases that *FSyft* and *StSyft* can solve. Moreover, as presented there, half of the cases that can be solved by *FSyft* and *StSyft*, around 400, are finished in less than 0.1 second, while Strix is unable to solve any cases given such time limit.

Counter Game. Figure 3 and Figure 4 show the running time of all tools on the counter game benchmarks. Since all of them got failed on cases with counter bits $n > 10$, here we only show realizable/unrealizale cases with counter bits $n \leq 10$, so we have 20 cases for each synthesis problem. The x-labels *c-rea/unrea-n* indicate the realizability and the number of counter bits of each case. Both of *FSyft* and *StSyft* are able to deal with cases with $n \leq 10$, while Strix only solves cases with n up to 7, either stable LTL_f synthesis or fair LTL_f synthesis. For those common solved cases, both of *FSyft* and *StSyft* take much less time than Strix.

Conclusions

In this paper we presented a fixpoint-based technique for LTL_f synthesis with assumptions for basic forms of fairness and stability, which is quite effective, as our experiment shows. Our technique can be summarized as follows: use the DFA for the LTL_f formula as the arena to play a game for the environment whose winning condition is to avoid reaching the accepting states while making the assumption true. Note that for a general LTL assumption (see (Aminof et al. 2019)), we can transform such an assumption into a parity automaton, take the Cartesian product with the DFA and play the parity/reachability game over the resulting arena. Comparing this possible approach to the reduction to LTL synthesis is a subject for future work.

Acknowledgments. Work supported in part by European

[4]We recommend viewing the figures online for a better vision.

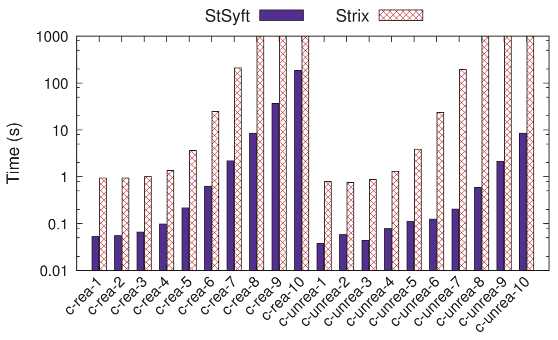

Figure 4: Stable LTL$_f$ synthesis. Comparison of running time between *StSyft* and Strix, in log scale. Bars of the maximum height indicate cases timed out.

Research Council under the European Union's Horizon 2020 Programme through the ERC Advanced Grant WhiteMec (No. 834228), NSF grants IIS-1527668, CCF-1704883, and IIS-1830549, NSFC Projects No. 61572197, No. 61632005 and No. 61532019.

References

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

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL Environment Specifications. In *ICAPS*.

Bloem, R.; Jobstmann, B.; Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2012. Synthesis of Reactive(1) Designs. *J. Comput. Syst. Sci.* 78(3):911–938.

Bloem, R.; Ehlers, R.; and Könighofer, R. 2015. Cooperative Reactive Synthesis. In *ATVA*, volume 9364 of *Lecture Notes in Computer Science*, 394–410. Springer.

Brenguier, R.; Raskin, J.; and Sankur, O. 2017. Assume-admissible synthesis. *Acta Inf.* 54(1):41–83.

Buchi, J. R., and Landweber, L. H. 1990. *Solving Sequential Conditions by Finite-State Strategies*.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In *AAAI*.

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

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2018. Finite LTL Synthesis with Environment Assumptions and Quality Measures. In *KR*, 454–463.

Chatterjee, K., and Henzinger, T. A. 2007. Assume-guarantee synthesis. In *TACAS*, 261–275.

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

De Giacomo, G., and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTL$_f$/LDL$_f$ Goals. In *IJCAI*, 4729–4735.

De Giacomo, G., and Vardi, M. 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*, 1558–1564.

D'Ippolito, N.; Braberman, V. A.; Piterman, N.; and Uchitel, S. 2013. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.* 22(1):9:1–9:36.

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.

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*.

Fried, D.; Tabajara, L. M.; and Vardi, M. Y. 2016. BDD-Based Boolean Functional Synthesis. In *CAV*.

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

Grädel, E.; Thomas, W.; and Wilke, T., eds. 2002. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer.

Green, C. 1969. Theorem Proving by Resolution as Basis for Question-Answering Systems. In *Machine Intelligence*, volume 4. American Elsevier. 183–205.

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

Li, J.; Rozier, K. Y.; Pu, G.; Zhang, Y.; and Vardi, M. Y. 2019. Sat-based explicit ltlf satisfiability checking. In *AAAI*, 2946–2953.

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

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

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. 1959. Finite Automata and Their Decision Problems. *IBM J. Res. Dev.* 3:114–125.

Somenzi, F. 2016. CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017a. A Symbolic Approach to Safety LTL Synthesis. In *HVC*, 147–162.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017b. Symbolic LTL$_f$ Synthesis. In *IJCAI*, 1362–1369.