On-the-fly Synthesis for LTL over Finite Traces

Shengping Xiao[1], Jianwen Li[1*], Shufang Zhu[2,3], Yingying Shi[1], Geguang Pu[1*] and Moshe Vardi[4]

[1]East China Normal University, [2]Sapienza Università di Roma,
[3]Shanghai Trusted Industrial Control Platform Co., Ltd, [4]Rice University

Abstract

We present a new synthesis framework based on the on-the-fly DFA construction for LTL over finite traces (LTL_f). Extant approaches rely heavily on the construction of the complete DFA w.r.t. the input LTL_f formula, whose size can be doubly exponential to the size of the formula in the worst case. Under those approaches, the synthesis cannot be conducted unless the whole DFA is completely constructed, which is not only inefficient but also not scalable in practice. Indeed, the DFA construction *is* the main bottleneck of LTL_f synthesis in prior work. To mitigate this challenge, we follow two steps in this paper: Firstly, we present several light-weight preprocessing techniques such that the synthesis result can be obtained even without DFA construction; Secondly, we propose to achieve the synthesis together with the on-the-fly DFA construction such that the synthesis result can be obtained before constructing the whole DFA. The on-the-fly DFA construction is implemented using the SAT-based techniques for automata generation. We compared our new approach with the traditional ones on extensive LTL_f synthesis benchmarks. Experimental results showed that the pre-processing techniques have a significant advantage on the synthesis performance in terms of scalability, and the on-the-fly synthesis is able to complement extant approaches on both realizable and unrealizable cases.

Introduction

Synthesis for Linear Temporal Logic over finite traces, i.e., LTL_f (De Giacomo and Vardi 2013), has emerged as a popular research topic in the AI community due to applications such as motion planning (Zhu et al. 2020; Rintanen 2004). LTL_f is a formal logic that has received considerable concerns from the AI community, due to its simplicity and ability to formalize and validate behaviors of AI systems (De Giacomo and Vardi 2013; Giacomo and Vardi 2015). While standard Linear Temporal Logic (LTL) is interpreted on infinite traces (Pnueli 1977), LTL_f is interpreted over finite traces (De Giacomo and Vardi 2013). Since first introduced in 2013, the fundamental problems of LTL_f, e.g., satisfiability (Li et al. 2014, 2019) and synthesis (Giacomo and Vardi 2015; Zhu et al. 2017), have been extensively studied in prior work. Towards applications, researchers successfully reduced the planning problem with LTL_f goals to the synthesis problem for LTL_f (Camacho et al. 2017; Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2018, 2019; Zhu et al. 2020), which makes the logic very attractive in this domain. We focus on LTL_f synthesis in this paper.

Given an LTL_f formula ϕ with input/output atomic proposition sets $\mathcal{X}, \mathcal{Y}$ such that $\mathcal{X} \cap \mathcal{Y} = \emptyset$ and $\mathcal{P} = \mathcal{X} \cup \mathcal{Y}$ is the set of atomic propositions of ϕ, the synthesis problem asks whether every finite trace generated as the result of a game between the environment controlling the input propositions and the system controlling the output propositions can satisfy the formula ϕ (see Definition 1 for details). Extant solutions rely on a reduction from LTL_f synthesis to DFA (Deterministic Finite Automata) games (Giacomo and Vardi 2015). Explicitly, the DFA that recognizes the same language as the LTL_f formula has to be constructed at first. Then a backward fixpoint calculation is performed on the generated DFA from the set of accepting states. The calculation initially marks the accepting states as the *winning* states, and recursively extends this winning set based on the current information. If finally the initial state of the DFA is included in the winning set, we conclude that the formula ϕ is realizable w.r.t. the input/output sets $\mathcal{X}$ and $\mathcal{Y}$. Otherwise, the formula is unrealizable. It is well-known that the LTL_f-to-DFA translation is the most demanding part of the computation, as the size of the generated DFA can be doubly exponential to the size of the formula (Kupferman and Vardi 2001). Indeed, the DFA construction is the main bottleneck in the current approach (Zhu et al. 2017).

Several optimizations for DFA construction have been proposed. (Zhu et al. 2017) has shown using MONA (Henriksen et al. 1995; Elgaard, Klarlund, and Möller 1998) to symbolically construct the minimal DFA can be much faster than using Spot (Duret-Lutz and Poitrenaud 2004), which employs an explicit construction. Later, (Zhu, Pu, and Vardi 2019) performed an extensive comparison over different encodings from LTL_f to the input format of MONA to look for the best-performing encoding, and showed the outperformance of First Order Logic (FOL) encoding. (Tabajara and Vardi 2019) introduced a partitioning technique to decompose the generation of a large DFA into small ones (Meyer, Sickert, and Luttenberger 2018). Recently, (Bansal et al. 2020) combined both of the explicit and symbolic DFA state-space representations and successfully achieved a bet-

*Both are corresponding authors (lijwen2748@gmail.com).
Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ter DFA construction than those using only one single representation. Nonetheless, none of these techniques above can avoid the double exponential-up cost, as the DFA construction is the indispensable part for LTL_f synthesis. Therefore, the question raises up whether it is possible to solve LTL_f synthesis without generating the whole DFA.

We present here a novel approach to achieve this goal. First of all, we present three light-weight pre-processing techniques for LTL_f synthesis, which can be easily implemented and with low running-cost. Notably, such pre-processing techniques can be performed directly on the input formula even without constructing the DFA. As a result, they can be integrated into all other available LTL_f synthesis approaches. Secondly, we propose a new synthesis framework that is based on DFA construction *on the fly*, i.e., the DFA states are created only as necessary. We start from the initial state s_0 (that is the input formula ϕ under our framework) and then proceed forward to continuously generate successor states when necessary. As soon as the new created state is determined as *winning/failure*, a backtrack procedure is invoked to determine whether the predecessors are winning/failure based on stored information. As soon as the initial state can be determined as winning (resp. failure), the realizable (resp. unrealizable) result can be concluded. In the worst case, the algorithm returns unrealizable when the whole DFA is constructed.

We conducted extensive experimental evaluations on both the pre-processing techniques and the new synthesis framework, by comparing to extant LTL_f synthesis tools SYFT (Zhu et al. 2017) and LISASYNT (Bansal et al. 2020). Results show that: (1) the pre-processing techniques can speed-up the synthesis with up to an exponentially better performance; (2) the new synthesis framework via on-the-fly DFA construction is able to complement the extant ones by uniquely solving a significant number of instances.

Preliminaries

Given a set $\mathcal{P}$ of atomic propositions, an LTL_f formula ϕ has the form: $\phi ::= tt \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2$, where tt is true, $p \in \mathcal{P}$ is an atomic proposition, $\neg$ is the *negation* operator, $\wedge$ is the *and* operator, $\bigcirc$ is the strong *Next* operator and $\mathcal{U}$ is the *Until* operator. We also have the corresponding dual operators ff (false) for tt, $\vee$ (or) for $\wedge$, $\mathcal{N}$ (weak Next) for $\bigcirc$ and $\mathcal{R}$ (Release) for $\mathcal{U}$. A *literal* is an atom $p \in \mathcal{P}$ or its negation ($\neg p$). Moreover, we use the notation $\Box\phi$ (Globally) and $\Diamond\phi$ (Eventually) to represent $ff\mathcal{R}\phi$ and $tt\mathcal{U}\phi$, respectively. Notably, $\bigcirc$ is the standard *Next* operator, while $\mathcal{N}$ is *weak Next*; $\bigcirc$ requires the existence of a successor instance, while $\mathcal{N}$ does not. Thus $\mathcal{N}\phi$ is always true in the last instance of a finite trace, since no successor exists there. This distinction is specific to LTL_f.

LTL_f formulas are interpreted over finite traces (De Giacomo and Vardi 2013). Given an atomic set $\mathcal{P}$, we define $\Sigma = 2^{\mathcal{P}}$ be the family of sets of atoms. Let $\eta \in \Sigma^+$ be a finite nonempty trace, with $\eta = \sigma_0\sigma_1 \ldots \sigma_n$. $|\eta| = n + 1$ denotes the length of η. Moreover, for $0 \leq i \leq n$, we denote $\eta[i]$ as the i-th position of η, and η_i to represent $\sigma_i\sigma_{i+1} \ldots \sigma_n$, which is the suffix of η from position i. We define the satisfaction relation $\eta \models \phi$ as follows:

- $\eta \models tt$; and $\eta \models p$, if $p \in \mathcal{P}$ and $p \in \eta[0]$;
- $\eta \models \neg\phi$, if $\eta \not\models \phi$;
- $\eta \models \phi_1 \wedge \phi_2$, if $\eta \models \phi_1$ and $\eta \models \phi_2$;
- $\eta \models \bigcirc\phi$ if $|\eta| > 1$ and $\eta_1 \models \phi$;
- $\eta \models \phi_1\mathcal{U}\phi_2$, if there exists $0 \leq i < |\eta|$ such that $\eta_i \models \phi_2$ and for every $0 \leq j < i$ it holds that $\eta_j \models \phi_1$;

Definition 1 (LTL_f Synthesis)**.** *Let ϕ be an LTL_f formula with the atomic set $\mathcal{P}$ and $\mathcal{X}, \mathcal{Y}$ be two atomic sets such that $\mathcal{X} \cap \mathcal{Y} = \emptyset$ and $\mathcal{X} \cup \mathcal{Y} = \mathcal{P}$. $\mathcal{X}$ is the set of input variables controlled by the environment and $\mathcal{Y}$ is the set of output variables controlled by the system. ϕ is realizable with respect to $\langle \mathcal{X}, \mathcal{Y} \rangle$ if*

- *for the **Environment-first** synthesis, there exists a strategy $g : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$, such that for an arbitrary infinite sequence $\lambda = X_0, X_1, \cdots \in (2^{\mathcal{X}})^{\omega}$ of propositional interpretations over $\mathcal{X}$, we can find $k \geq 0$ such that $\rho \models \phi$ is true, where $\rho = (X_0 \cup g(X_0)), (X_1 \cup g(X_0, X_1)), \cdots, (X_k \cup g(X_0, \cdots, X_k))$.*
- *for the **System-first** synthesis, there exists a strategy $g : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$, such that for an arbitrary infinite sequence $\lambda = X_0, X_1, \cdots \in (2^{\mathcal{X}})^{\omega}$ of propositional interpretations over $\mathcal{X}$, we can find $k > 0$ such that $\rho \models \phi$ is true, where $\rho = (X_0 \cup g(\epsilon)), (X_1 \cup g(X_0)), \cdots, (X_k \cup g(X_0, \cdots, X_{k-1}))$. ($\epsilon$ means the empty trace.)*

In this paper, we focus on the **System-first** LTL_f synthesis. Extant solutions originate from the approach given in (Giacomo and Vardi 2015), which has to construct the DFA w.r.t. the formula at first, and then computes the winning state set back from the accepting states to check whether the initial state is included in such set. The result is realizable if this is the case; otherwise, it is unrealizable. Readers are referred to the literature for more details.

Notation. We say an LTL_f formula is in *Negation Normal Form* (NNF), if the negation operator appears only in front of an atom. It should be noted that every LTL_f formula can be converted into its NNF in linear time. We use $cl(\phi)$ to denote the set of subformulas of ϕ. The two LTL_f formulas ϕ_1, ϕ_2 are semantically equivalent, denoted as $\phi_1 \equiv \phi_2$, iff for every finite trace η, $\eta \models \phi_1$ iff $\eta \models \phi_2$. Obviously, we have $(\phi_1 \vee \phi_2) \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\mathcal{N}\phi \equiv \neg \bigcirc \neg\phi$ and $(\phi_1\mathcal{R}\phi_2) \equiv \neg(\neg\phi_1\mathcal{U}\neg\phi_2)$.

Approach Overview

This section presents the high-level ideas behind both the pre-processing and on-the-fly synthesis techniques. The pre-processing aims to determine the synthesis result directly based on the input formula and the input/output pair. First of all, it is trivial to know that the formula cannot (resp. can) be realizable if it is unsatisfiable (resp. valid). Therefore, one can run the satisfiability checkers, such as aaltaf (Li et al. 2019), to check the satisfiability of the input formula before synthesis, ruling out the meaningless instances (Theorem 1). Next, for an LTL_f formula with the atomic set $\mathcal{P} = \mathcal{X} \cup \mathcal{Y}$, if there is $Y \in 2^{\mathcal{Y}}$ such that $Y \models \phi$ is true (Y being considered as a length-one finite trace), we can know that ϕ is realizable (Theorem 2). Consider $\phi = \Box(a \vee b)$ with $\mathcal{X} = \{a\}$

and $\mathcal{Y} = \{b\}$ as an example. Since $\{b\} \models \phi$ is true, ϕ is realizable. For an unrealizable formula ϕ, we first define the *projection* (Definition 3), i.e., $\phi|_{\mathcal{P}_\phi}$, which is a Boolean formula including all first-position elements of traces accepted by ϕ. Then, if there is no $Y \in 2^{\mathcal{Y}}$ such that $Y \models \phi|_{\mathcal{P}_\phi}$ is true, we can conclude that ϕ is unrealizable (Theorem 3). For instance, consider $\phi = \Box(a \wedge b)$ with $\mathcal{X} = \{a\}$ and $\mathcal{Y} = \{b\}$. ϕ is determined as unrealizable even without constructing the corresponding DFA.

We apply the SAT-based technique presented in (Li et al. 2019) that constructs automata (NFA) states on the fly, to solve the synthesis problem. Given initial state ϕ (essentially a formula), the technique can generate a *propositional assignment*, which includes one transition information starting from the state ϕ. We adapt this technique accordingly such as to generate one transition for the transition-based DFA (TDFA, a variant of DFA that is better for on-the-fly construction). Then, the new methodology creates TDFA states in four different ways as shown in Figure 1. From current state s_1, our approach creates state s_2 by fixing $Y_0 \in 2^{\mathcal{Y}}$ and enumerating $X \in 2^{\mathcal{X}}$ (currently is X_0). After that,

(a) if s_2 cannot be recognized as winning or failure, the DFS (Depth-first Search) strategy is applied to create a new state s_3;

(b) if s_2 is recognized as winning, another $X \in 2^{\mathcal{X}}$ (here is X_1) is selected by fixing Y_0, obtaining a new state s_3. If all successor states are winning, then s_1 is winning;

(c) if s_2 is recognized as failure, another $Y \in 2^{\mathcal{X}}$ (here is Y_1) is selected and Y_0 is ruled out from the winning strategy in current search. If no more Y can be selected, s_1 is a failure state;

(d) if s_2 has been visited already, i.e., a loop is found during the state computation, we can prove that choosing Y_0 cannot belong to the winning strategy in current search, and another $Y \in 2^{\mathcal{X}}$ (here is Y_1) has to be selected.

One can see that our on-the-fly approach is able to return either realizable or unrealizable result before the whole DFA is constructed.

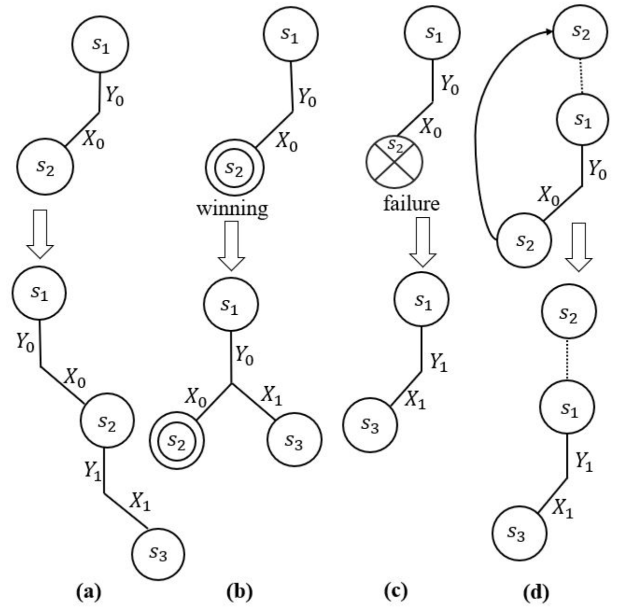

Figure 1: A demonstration of key operations for on-the-fly LTL_f synthesis.

Pre-processing Techniques for LTL_f Synthesis

In this section, we introduce three pre-processing techniques for LTL_f synthesis, which can be evaluated immediately on the given LTL_f formula w.r.t. the atomic set $\mathcal{P} = \mathcal{X} \cup \mathcal{Y}$. First, the following Theorem is straightforward according to the LTL_f semantics and Definition 1.

Theorem 1. *If the LTL_f formula ϕ is unsatisfiable, then ϕ is unrealizable; If ϕ is valid, then ϕ is realizable.*

Theorem 1 suggests that an unsatisfiable/valid LTL_f formula is not quite useful as a synthesis specification in practice. To that end, an extant LTL_f-satisfiability solver, e.g., aaltaf (Li et al. 2019), can be used to check the satisfiability/validity of the formula before synthesis.

We now consider realizability in traces of length 1. We first define the satOnce predicate.

Definition 2. *Given an LTL_f formula ϕ and $\omega \in 2^{\mathcal{P}}$, we define the predicate* $\mathsf{satOnce}(\omega, \phi)$ *as true iff*

- *ϕ is tt; or*
- *ϕ is an atom and $\phi \in \omega$; or*
- *$\phi = \neg\psi$ and $\neg\mathsf{satOnce}(\omega, \psi)$ is true; or*
- *$\phi = \phi_1 \wedge \phi_2$ and $\mathsf{satOnce}(\omega, \phi_1)$ and $\mathsf{satOnce}(\omega, \phi_2)$ are true; or*
- *$\phi = \phi_1 \vee \phi_2$ and $\mathsf{satOnce}(\omega, \phi_1)$ or $\mathsf{satOnce}(\omega, \phi_2)$ is true; or*
- *$\phi = \phi_1 \mathcal{U} \phi_2$ or $\phi = \phi_1 \mathcal{R} \phi_2$ and $\mathsf{satOnce}(\omega, \phi_2)$ is true.*

Notably, $\mathsf{satOnce}(\omega, \bigcirc\psi)$ can never be true, while $\mathsf{satOnce}(\omega, \mathcal{N}\psi)$ is always true. Based on Definition 2 and the semantics of LTL_f formulas, the lemma below is straightforward.

Lemma 1. *Given an LTL_f formula ϕ and $\omega \in 2^{\mathcal{P}}$,* $\mathsf{satOnce}(\omega, \phi)$ *is true iff $\omega \models \phi$ holds.*

In fact, Definition 2 can be considered as the simplified version of LTL_f semantics in which the length of the finite trace is restricted to be one. Definition 2 is provided as a better option for the implementation purpose.

Theorem 2. *Given an LTL_f formula ϕ with alphabet $\mathcal{X} \cup \mathcal{Y}$, if there exists $Y \in 2^{\mathcal{Y}}$ such that* $\mathsf{satOnce}(Y, \phi)$ *is true, then ϕ is realizable.*

Proof. From Lemma 1, $\mathsf{satOnce}(Y, \phi)$ is true implies that $Y \models \phi$ is true. As a result, for every $X \in 2^{\mathcal{X}}$, $X \cup Y \models \phi$ is true, providing that $\mathcal{X} \cap \mathcal{Y} = \emptyset$. Therefore, there exists a strategy g with $g(\epsilon) = Y$ such that for every $X \in 2^{\mathcal{X}}$ it holds that $X \cup g(\epsilon) \models \phi$. According to Definition 1, g is a winning strategy for the system and ϕ is realizable. $\Box$

Consider the formula $a\mathcal{U}b$ with $\mathcal{X} = \{a\}$ and $\mathcal{Y} = \{b\}$ as an example. Let $Y = \{b\}$ and since $\mathsf{satOnce}(Y, a\mathcal{U}b)$ is true, ϕ is realizable according to Theorem 2. It is easy to see

that Theorem 2 can be extremely helpful for the synthesis instances like $\psi\mathcal{U}b$ with $\mathcal{Y} = \{b\}$, under which the realizable result can be achieved by Theorem 2 directly without further automata construction. This advantage may potentially lead to an exponential better performance, which will be discussed in the experimental section later.

Next, we introduce the concept of *formula projection* for the pre-processing of unrealizable formulas.

Definition 3 (Formula Projection). *Given an LTL_f formula ϕ in NNF with the atomic set $\mathcal{P}$, we define its projection on $\mathcal{P}$, denoted as $\phi|_{\mathcal{P}}$, as a Boolean formula as follows:*

- $\phi|_{\mathcal{P}} = \phi$ *if* ϕ *is* tt, ff *or a literal;*
- $\phi|_{\mathcal{P}} = tt$ *if* $\phi = \bigcirc\psi$ *or* $\phi = \mathcal{N}\psi$*;*
- $\phi|_{\mathcal{P}} = \phi_1|_{\mathcal{P}} \wedge \phi_2|_{\mathcal{P}}$ *if* $\phi = \phi_1 \wedge \phi_2$*;*
- $\phi|_{\mathcal{P}} = \phi_1|_{\mathcal{P}} \vee \phi_2|_{\mathcal{P}}$ *if* $\phi = \phi_1 \vee \phi_2$*;*
- $\phi|_{\mathcal{P}} = \phi_1|_{\mathcal{P}} \vee \phi_2|_{\mathcal{P}}$ *if* $\phi = \phi_1\mathcal{U}\phi_2$*;*
- $\phi|_{\mathcal{P}} = \phi_2|_{\mathcal{P}}$ *if* $\phi = \phi_1\mathcal{R}\phi_2$.

Lemma 2. *Given an LTL_f formula ϕ and a finite trace η, $\eta \models \phi$ implies that $\eta[0] \models \phi|_{\mathcal{P}}$.*

The proof can be done by induction over the structure of ϕ, which is omitted here. Lemma 2 indicates that $\phi|_{\mathcal{P}}$ is sufficient to capture all the first-position elements of ϕ's accepting traces. Inspired by that, we have the following theorem that can help with identifying a formula being unrealizable.

Theorem 3. *Given an LTL_f formula ϕ with the atomic set $\mathcal{X} \cup \mathcal{Y}$, if there does not exist $Y \in 2^{\mathcal{Y}}$ such that $Y \models \phi|_{\mathcal{X}\cup\mathcal{Y}}$, then ϕ is unrealizable.*

Proof. Assume ϕ is realizable. According to Definition 1, there exists a winning strategy g such that for an arbitrary infinite sequence $X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, there exists $k \geq 0$ such that $\rho = (X_0 \cup g(\epsilon)), (X_1 \cup g(X_0)), \ldots, (X_k \cup g(X_{k-1}))$ satisfies ϕ. Let $Y = g(\epsilon)$, and based on Lemma 2, we have that $Y \cup X_0 \models \phi|_{\mathcal{X}\cup\mathcal{Y}}$ for every $X_0 \in 2^{\mathcal{X}}$. As the consequence, it is required that $Y \models \phi|_{\mathcal{X}\cup\mathcal{Y}}$ is true. However, we already know that $Y \models \phi|_{\mathcal{X}\cup\mathcal{Y}}$ does not hold for any $Y \in 2^{\mathcal{Y}}$, which causes the contradiction. Therefore, we prove that ϕ is unrealizable. □

Consider formula $\phi = \Box(a \wedge b) \wedge \Box(c \wedge d)$ with $\mathcal{X} = \{a, c\}$ and $\mathcal{Y} = \{b, d\}$ as an example. From Definition 3, we have that $\phi|_{\mathcal{P}} = (a \wedge b \wedge c \wedge d)$. Obviously, $Y \models \phi|_{\mathcal{P}}$ does not hold for any $Y \in 2^{\mathcal{Y}}$. We can conclude that ϕ is unrealizable based on Theorem 3. In general, the performance of the traditional synthesis approach for $\phi = \Box(a \wedge b) \wedge \Box(c \wedge d) \wedge \psi$ with $\mathcal{X} = \{a, c\} \cup \mathcal{P}_\psi$ and $\mathcal{Y} = \{b, d\}$ (assume $\mathcal{P}_\psi$ is the atomic set of ψ and $\mathcal{P}_\psi \cap \{a, b, c, d\} = \emptyset$) can decrease exponentially w.r.t. the size of ψ, while that of the synthesis based on Theorem 3 can escape from the drawback.

On-the-fly Synthesis via TDFA Games

In this section, we first introduce the theoretic foundation of the on-the-fly LTL_f synthesis approach, which is based on solving a TDFA (Transition-based DFA) game. We present an algorithm that is able to generate TDFA on the fly to implement the synthesis. TDFA is a variant of DFA that is better for performing on-the-fly construction. The definition of TDFA is shown below.

Definition 4 (Transition-based DFA). *A transition-based DFA (TDFA) is a tuple $\mathcal{A} = (\Sigma, S, s_0, \delta, T)$, where*

- Σ *is a set of alphabet;*
- S *is a set of states;*
- $s_0 \in S$ *is the initial state;*
- $\delta : S \times \Sigma \hookrightarrow S$ *is the transition function, which is a partial function, i.e.* $\delta(s, \omega) \in S$ *or* $\delta(s, \omega)$ *is undefined for* $s \in S$ *and* $\omega \in \Sigma$*;*
- $T \subseteq \delta$ *is the set of accepting transitions.*

The run r of $\mathcal{A}$ on a finite trace $\eta = \omega_0, \omega_1, \cdots, \omega_n \in \Sigma^+$ is a finite state sequence $r = s_0, s_1, \ldots, s_{n+1}$ such that s_0 is the initial state and $\delta(s_i, \omega_i) = s_{i+1}$ is true for $0 \leq i \leq n$. The trace η is accepted by $\mathcal{A}$ iff the corresponding run r ends with an accepting transition, i.e., $\delta(s_n, \omega_n) = s_{n+1}$ is in T. We denote transition $\delta(s_1, \omega) = s_2$ as $s_1 \xrightarrow{\omega} s_2$.

Lemma 3. *TDFA are equally expressive with DFAs.*

Proof. (⇐:) A DFA can be trivially converted to its equivalent TDFA by marking all transitions leading to accepting states as accepting transitions. Therefore, every word that is accepted by the DFA is also accepted by this TDFA.

(⇒:) We first convert a TDFA to an NFA by adding a new state ns, which is marked as the unique accepting state of this NFA. Later, we create a transition $s \xrightarrow{\omega} ns$ if transition $s \xrightarrow{\omega} s'$ is an accepting transition of the TDFA. Therefore, every trace that is accepted by the TDFA is also accepted by this NFA. From the constructed NFA, one can trivially generate the equivalent DFA by the subset construction. □

According to (De Giacomo and Vardi 2013), every LTL_f formula can be converted to a DFA that accepts exactly the same language as the LTL_f formula. Therefore, it is straightforward that there is also a TDFA for every LTL_f formula such that they accept the same language.

In this paper, we present a dedicated SAT-based LTL_f-to-TDFA construction technique for on-the-fly synthesis. The SAT-based LTL_f-to-automata construction technique was first introduced in (Li et al. 2019), and we follow the methodology presented in the literature. Given an LTL_f formula ϕ, this technique is able to generate a *propositional assignment* A of ϕ such that A includes the information of a transition in the corresponding NFA. For more details, we refer to (Li et al. 2019) and here we just use $\mathsf{SAT}(\phi)$ to denote such process. Assume $A = \mathsf{SAT}(\phi)$ is a *propositional assignment* returned by the SAT-based technique, we use $L(A)$ to denote the transition label, which is represented as a Boolean formula, and $\mathbf{next}(A)$ to denote the successor state of ϕ. ϕ represents the current state. The TDFA construction can be achieved as follows.

Definition 5 (LTL_f-to-TDFA). *Given an LTL_f formula ϕ, the corresponding TDFA $\mathcal{A}_\phi$ is a tuple $(\Sigma, S, \delta, s_0, T)$ s.t.*

- $\Sigma = 2^{\mathcal{L}}$ *is the alphabet, where* $\mathcal{L}$ *is the literal set of* ϕ*;*
- $S \subseteq 2^{2^{cl(\phi)}}$ *is the set of states;*
- $\delta : S \times \Sigma \hookrightarrow S$ *is the partial transition function, where* $s_2 = \delta(s_1, \omega)$ *holds iff* $s_2 = \{\mathbf{next}(A) | A \in \{\mathsf{SAT}(s_1)\}$ *and* $\omega \models L(A)\}$ *for* $\omega \in \Sigma$*;*

- $s_0 = \{\{\phi\}\}$ *is the initial state;*
- $T \subseteq \delta$ *is the set of accepting transitions. A transition* $s_1 \xrightarrow{\omega} s_2$ *is in* T *iff* $\omega \models s_1$ *holds.*

By Definition 5, a TDFA state s is a set of sets of subformulas of the input formula ϕ. We identify such as state with a DNF formula: a state s represents the formula $\bigvee_{q\in s}\bigwedge_{\psi\in q}\psi$, and vice versa. The following theorem guarantees the correctness of the TDFA construction shown in Definition 5.

Theorem 4. *Given an* LTL_f *formula* ϕ *and the* TDFA $\mathcal{A}_\phi$ *constructed by Definition 5, a finite trace* $\eta \models \phi$ *holds iff* η *is accepted by* $\mathcal{A}_\phi$*.*

The proof of this theorem relies on details from (Li et al. 2019).

Once we obtain the TDFA of the given LTL_f formula, the synthesis problem can be reduced to a TDFA game.

Definition 6 (TDFA Game)**.** *A* TDFA *game is a two-player game over a* TDFA $\mathcal{A} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \delta, T)$ *such that*

- $2^{\mathcal{X}\times\mathcal{Y}}$ *is the alphabet of the game, where* $\mathcal{X}$ *and* $\mathcal{Y}$ *are two disjoint sets of variables that are controlled by the environment and system respectively;*
- S *is the set of states;*
- s_0 *is the initial state;*
- $\delta : S \times 2^{\mathcal{X}\times\mathcal{Y}} \hookrightarrow S$ *is the partial transition function, i.e.* $\delta(s, X \cup Y)$ *is in* S *or undefined for* $s \in S$, $X \in 2^{\mathcal{X}}$ *and* $Y \in 2^{\mathcal{Y}}$*;*
- $T \subseteq \delta$ *is the set of winning transitions of the game; the system wins once it takes a transition in* T*.*

To coordinate with LTL_f synthesis, we focus here **System-first** TDFA games. We say a TDFA game $\mathcal{A}$ is *winning* for the system iff there is a system winning strategy $g : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for an arbitrary infinite environment sequence $X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, there is $k > 0$, such that the corresponding run r on the finite trace $(X_0 \cup g(\epsilon)), (X_1 \cup g(X_0)), \ldots, (X_k \cup g(X_{k-1}))$ is winning. In the following, we define system *winning* and *failure* states of a TDFA game.

Definition 7 (System Winning/Failure State)**.** *For a* TDFA *game over* $\mathcal{A} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \delta, T)$*,* $s \in S$ *is a system winning state iff there is* $Y \in 2^{\mathcal{Y}}$ *such that for every* $X \in 2^{\mathcal{X}}$*, either* $\delta(s, X \cup Y) = s'$ *is an accepting transition or* s' *is a winning state. Moreover, we say* s *is a system failure state iff it is not a system winning state.*

The following lemma is deducible from Definition 7 instantly and can be used as an easy check in the algorithm.

Lemma 4. *For a* TDFA *game* $\mathcal{A} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \delta, T)$ *and state* $s \in S$*,*

1. *s is a system winning state if there is* $Y \in 2^{\mathcal{Y}}$ *such that for every* $X \in 2^{\mathcal{X}}$*,* $\delta(s, X \cup Y) = s'$ *is an accepting transition;*
2. *s is a failure state if for every* $Y \in 2^{\mathcal{Y}}$*, there is* $X \in 2^{\mathcal{X}}$ *such that* $\delta(s, X \cup Y)$ *is undefined.*

Informally speaking, a system winning state is a state if all of its out-going transitions are defined, and are either accepting or leading to some winning state. To the opposite, a system failure state is a state which has some out-going transitions undefined, or has a part of the transitions leading to the failure states. The next theorem shows how to use the winning/failure state to determine the TDFA game.

Theorem 5. *Given a* TDFA *game* $\mathcal{A} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \delta, T)$*,* s_0 *is a system winning state iff the system wins the game.*

The theorem can be proved by induction over the states of TDFA based on Definition 6 and 7. Now, we present the main theorem for our synthesis approach.

Theorem 6. *For an* LTL_f *formula* ϕ *with* $\mathcal{X}$ *and* $\mathcal{Y}$*, let* $\mathcal{A} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \delta, T)$ *be the corresponding* TDFA *game description.* s_0 *is a system winning state iff* ϕ *is realizable.*

Proof. ($\Rightarrow$) Since s_0 is a system winning state, there is a system winning strategy $g : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for an arbitrary infinite sequence $X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, there exists $k > 0$ such that the run of $\mathcal{A}$ on the finite trace $\rho = (X_0 \cup g(\epsilon)), (X_1 \cup g(X_0)), \ldots, (X_k \cup g(X_{k-1}))$ is an accepting run. Therefore, $\rho \models \phi$ holds and g is the system winning strategy.

($\Leftarrow$) If ϕ is realizable, then there is a system winning strategy $g : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for an arbitrary infinite sequence $X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, there exists $k > 0$ such that the finite trace $\rho = (X_0 \cup g(\epsilon)), (X_1 \cup g(X_0)), \ldots, (X_k \cup g(X_{k-1}))$ satisfies ϕ. The run of $\mathcal{A}$ on ρ is thus an accepting run. Therefore, s_0 is a system winning state with winning strategy g. $\square$

An important question is raised, which is how to determine whether the initial state is a *winning/failure state*? The classical synthesis approach (Giacomo and Vardi 2015) first constructs the whole DFA w.r.t. the input LTL_f formula, and then performs a backward fixpoint computation. The fixpoint is initialized as the set of accepting states, visiting which the system obviously wins the game. The fixpoint computation then iteratively collects new states from which the system is able to reach an already-defined winning state, no matter how the environment behaves. The computation terminates as soon as we reach the fixpoint, i.e., no more winning states can be collected. The system wins the game if the initial state s_0 is in the winning set. The main drawback of this approach is that the winning states can only be collected after the full DFA is obtained.

We present in this paper a new technique that is able to collect the set of system winning states and synthesize the system winning strategy on the fly. The algorithm Synthesis, which is shown in Algorithm 1, describes the main procedure of this technique. Comments are blue-colored for better understanding. We summarize the crucial parts of the technique as follows.

The algorithm Synthesis takes a tuple $(\phi, \mathcal{X}, \mathcal{Y})$ as the input and returns a set Ω which represents the winning strategy if $(\phi, \mathcal{X}, \mathcal{Y})$ is realizable, or an empty set if unrealizable. Each element of Ω is in the form of $\langle s, Y, \{\langle X, s'\rangle\}\rangle$, where $s, s' \in S$, $X \in 2^{\mathcal{X}}$ and $Y \in 2^{\mathcal{Y}}$. Each element indicates

Algorithm 1 Synthesis: Compute the winning strategy on the fly

Require: LTL_f formula ϕ with $\mathcal{X}$ and $\mathcal{Y}$;
Ensure: $\Omega = \{\langle s, Y, \{\langle X, s'\rangle\}\rangle\}$ if ϕ is realizable, otherwise return $\emptyset$;{$s, s' \in S$, $X \in 2^{\mathcal{X}}$ and $Y \in 2^{\mathcal{Y}}$}
1: Let $\langle ret, \Omega\rangle := preprocess(\phi, \mathcal{X}, \mathcal{Y})$;{Check by the pre-processing techniques at first}
2: **if** $ret \neq unknown$ **then**
3:    **return** Ω;
4: **while** true{Enumerate Y inside} **do**
5:    $\langle \omega, s\rangle := getTransition(\phi, \emptyset)$;{Get $\phi \xrightarrow{\omega} s$}
6:    **if** $\langle \omega, s\rangle = \emptyset$ **then**
7:       **return** $\emptyset$;
8:    Let $\phi' := \phi$ and $r := \langle \phi, Y = \omega|_{\mathcal{Y}}, Q = \{\langle \omega|_{\mathcal{X}}, s\rangle\}\rangle$;
9:    Push r into Ω;
10:    **while** true{Fix $Y = \omega|_{\mathcal{Y}}$, enumerate X inside} **do**
11:       **if** s has been generated{A loop is detected} **then**
12:          **break**;{This Y cannot be a move in a winning strategy}
13:       **if** $\omega \models \phi'${An accepting transition is detected} **then**
14:          $\phi' := \phi' \wedge (\neg(\omega|_{\mathcal{X}}))$;{Enumerate X}
15:       **else**
16:          Let $\Omega' :=$ Synthesis$(s, \mathcal{X}, \mathcal{Y})$;
17:          **if** $\Omega' \neq \emptyset${s is a winning state} **then**
18:             $\phi' := \phi' \wedge (\neg(\omega|_{\mathcal{X}}))$;{Enumerate X}
19:          **else**
20:             **break**;{The chosen Y is not a move of a winning strategy}
21:       $\langle \omega', s'\rangle := getTransition(\phi', \omega|_{\mathcal{Y}})$;{Get the transition $\phi \xrightarrow{\omega'} s'$ such that $\omega|_{\mathcal{Y}} \subseteq \omega'$}
22:       **if** $\langle \omega', s'\rangle = \emptyset${Enumerating X is finished} **then**
23:          **return** Ω;
24:       **else**
25:          Update $r = \langle \phi, Y, Q\rangle$ by pushing $\langle \omega'|_{\mathcal{X}}, s'\rangle$ into Q;
26:    Remove r from Ω;
27:    $\phi := \phi \wedge (\neg(\omega|_{\mathcal{Y}}))$;{Enumerate Y}

that from state s, the system can take Y as a move, such that no matter how the environment chooses $X \in 2^{\mathcal{X}}$, the next state s' is also a system winning state, i.e., $s \xrightarrow{X \cup Y} s'$ is a transition of the TDFA and s' is a winning state as well.

Synthesis starts with the preprocessing techniques from Line 1 to 3. The *preprocess* function implements Theorem 1-3, and Synthesis returns immediately if the function succeeds to determine ϕ is a winning or failure state. The loop from Line 4 to 27 aims to enumerate $Y \in 2^{\mathcal{Y}}$ inside, and it can terminate as long as a winning move Y for the system is found (at Line 23). Meanwhile, the loop from Line 10 to 25 has to enumerate every $X \in 2^{\mathcal{X}}$ before it can conclude the fixed Y is indeed a winning move for the system. If the enumeration on X is not successful, another Y has to be chosen (at Line 27) and the above process repeats.

There are two points that need to be clarified in the algorithm. Firstly, if the new transition $\phi \xrightarrow{\omega} s$ is not an accepting transition, Synthesis will be invoked on s recursively to determine whether s is a winning state (at Line 16). Secondly, if a loop is detected before s can be determined as the winning state for the system (at Line 11), the current chosen Y cannot be a move in a winning strategy. Assume the run of the TDFA is $r = s, s_1, \ldots, s$. Starting from s, the environment can have the option to induce the same run as r, in which case the system can never win.

Algorithm 2 Implementation of function *getTransition*

Require: LTL_f formula ϕ and a set of literals $assumption$;
Ensure: $\langle label, next\rangle$ such that $\phi \xrightarrow{label} next$ is a transition of the TDFA, otherwise return $\emptyset$;
1: Let $label := null$, $next := null$;
2: Let $\phi := \phi \wedge (\bigwedge assumption)$;
3: **while** true{Find $label$ such that $Y = label|_{\mathcal{Y}}$ satisfies $Y \models \phi|_{\mathcal{X} \cup \mathcal{Y}}$} **do**
4:    Let $A =$ SAT(ϕ);
5:    **if** $A = \emptyset$ **then**
6:       **break**;
7:    **if** $A|_{\mathcal{Y}} \models \phi|_{\mathcal{X} \cup \mathcal{Y}}$ **then**
8:       $label := L(A)$;
9:       $next := \mathbf{next}(A)$;
10:       **break**;{$Y = A|_{\mathcal{Y}}$ is found}
11:    $\phi := \phi \wedge (\neg \bigwedge A|_{\mathcal{Y}})$;
12: $\phi := \phi \wedge label \wedge (\neg next)$;
13: **while** true{Fix $label$ and generate the successor state of ϕ in the TDFA} **do**
14:    Let $A =$ SAT(ϕ);
15:    **if** $A = \emptyset$ **then**
16:       **break**;
17:    $next := next \vee \mathbf{next}(A)$;
18:    $\phi := \phi \wedge (\neg(\mathbf{next}(A)))$;
19: **if** $label = null$ or $next = null$ **then**
20:    **return** $\emptyset$;
21: **return** $\langle label, next\rangle$;

The Algorithm 2 is used to calculate a feasible transition in TDFA. To enumerate X, we set the parameter *assumption* that represents the current fixed Y. The first while loop (from Line 3 to Line 11) aims to obtain a feasible label of a TDFA transition. According to Theorem 3, when we get an assignment A from the SAT solver, we check whether $A|_{\mathcal{Y}} \models \phi|_{\mathcal{X} \cup \mathcal{Y}}$ holds (Line 7) to determine whether this transition is feasible. The second while loop (from Line 13 to Line 18) is to calculate all possible NFA successors when fixing the labels of the transition so as to generate the TDFA successor. In this loop, we fix the *labels* each time and enumerate the *next* part of the NFA transition. If the first loop cannot find a feasible TDFA transition, the $getTransition$ function returns $\emptyset$.

Let $\Omega =$ Synthesis$(\phi, \mathcal{X}, \mathcal{Y})$ and we now define the strategy generator based on Ω. Notably, our strategy generator only returns one winning strategy due to the on-the-fly construction, while the approach in (Giacomo and Vardi 2015) is able to return all possible winning strategies.

Definition 8 (Strategy Generator)**.** *The strategy generator*

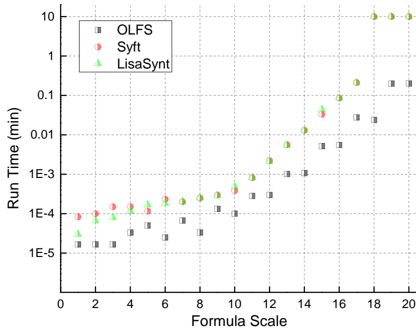

Figure 2: Results on the pattern formula $U(n) = p_1\mathcal{U}(p_2\mathcal{U}(...\mathcal{U}p_n))$.

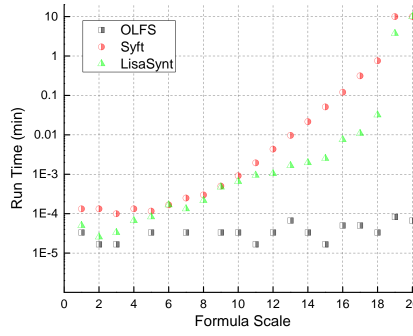

Figure 3: Results on the pattern formula $GF(n) = \Box p_1 \wedge (\bigwedge_{i=2\cdots n} \Diamond p_i)$.

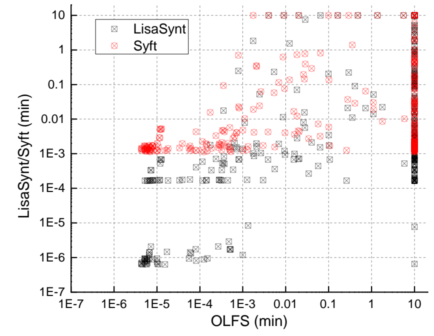

Figure 4: Results on the benchmarks from (Bansal et al. 2020).

for the LTL_f *synthesis problem* $(\phi, \mathcal{X}, \mathcal{Y})$ *is a transducer* $\mathcal{T}_\mathcal{A} = (2^{\mathcal{X}\times\mathcal{Y}}, S, s_0, \xi, \sigma)$ *such that*

- $2^{\mathcal{X}\times\mathcal{Y}}$ *is the alphabet of the transducer;*
- $S = \{s | \langle s, Y, \{\langle X, s' \rangle\}\rangle \in \Omega\} \subseteq 2^{2^{\mathsf{cl}(\phi)}}$ *is the set of states;*
- $s_0 = \{\{\phi\}\}$ *is the initial state;*
- $\xi : S \times 2^{\mathcal{X}} \to 2^S$ *is the transition function such that* $\xi(s, X) = \{s' | \langle s, Y, \{\langle X, s' \rangle\}\rangle \in \Omega\}$*;*
- $\sigma : S \to 2^{\mathcal{Y}}$ *is the output function such that* $\sigma(s) = \{Y | \langle s, Y, \{\langle X, s' \rangle\}\rangle \in \Omega\}$.

Experimental Evaluation

Tools We implemented both the pre-processing and on-the-fly synthesis techniques in the tool OLFS. We compared the results with extant LTL_f-synthesis tools SYFT (Zhu et al. 2017) and LISASYNT (Bansal et al. 2020). Both tools implement the synthesis approach proposed in (Giacomo and Vardi 2015), but are distinct from each other by using different complete DFA constructions, monolithic and compositional, respectively. All three tools were run with their default parameters.

Benchmarks We benchmarked the experiment with the 430 instances presented in (Bansal et al. 2020). We also select two classes of patterns U and GF, which originate from (Rozier and Vardi 2007) and are shown in Figure 2 and Figure 3 respectively, to test the efficiency of pre-processing techniques.

Platform We ran the experiments on a RedHat 6.0 cluster with 2304 processor cores in 192 nodes (12 processor cores per node), running at 2.83 GHz with 48GB of RAM per node. Each tool executed on a dedicated node with a timeout of 10 minutes, measuring execution time with Unix `time`. Excluding timeouts, all solvers found correct verdicts for all formulas.

Results Figure 2 and Figure 3 show the results on the U and GF patterns to evaluate the power of the pre-processing techniques presented in the paper. We carefully divide the atomic sets such that the synthesis on the U pattern formulas are all realizable, while the results for the GF pattern formulas are all unrealizable. By applying Theorem 2 to synthesis the U patterns, OLFS is able to achieve a linear-cost performance. Meanwhile, SYFT and LISASYNT reach the timeout when the pattern length n is greater than 18 (see Figure 2). Analogously, Theorem 3 enables OLFS to gain a linear-cost performance on the synthesis of the GF patterns, while SYFT and LISASYNT perform exponentially worse on such patterns (see Figure 3).

Figure 4 shows the comparison results between on-the-fly synthesis approach and the classical one presented in (Giacomo and Vardi 2015), represented by tools SYFT and LISASYNT, on the 430 instances from (Bansal et al. 2020). The figure shows that the on-the-fly approach cannot outperform the classical one. In total, OLFS solves 149 out of 430 instances, while SYFT and LISASYNT solve the number of 281 and 288 respectively. There are 50 of 149 instances for that OLFS can solve faster than the other two tools, as shown in Figure 4. Therefore, we conclude that the on-the-fly approach can complement rather than replace extant approaches.

The reason why current on-the-fly synthesis cannot perform better, is that a lot of time is consumed by TDFA state generation. From Definition 5, each state of the TDFA belongs to $2^{2^{cl(\phi)}}$, where ϕ is the input formula. The SAT-based technique shown in (Li et al. 2019) is good at computing transitions for the NFA, but the composition of the non-deterministic states so as to obtain the deterministic one is challenging computationally. We observed that most instances that cannot be solved by OLFS fail due to this reason. We leave the improvement of the on-the-fly state generation to future work.

Concluding Remarks

In this paper, we presented both pre-processing and on-the-fly techniques to solve the synthesis problem of LTL over finite traces. Compared to other methods, our approach allows us to perform the synthesis by possibly just generating partially the DFA w.r.t. the input LTL_f formula. The experimental results show that, (1) the pre-processing technique can bring an exponential-better performance than existing temporal synthesis approaches; (2) our on-the-fly synthesis approach complements the extant ones by speeding up the synthesis of 50 instances in the benchmark suite.

Acknowledgment

Jianwen Li and Geguang Pu are supported by National Key Research and Development Program (#2020AAA0107800), the Science and Technology Commitment of Shanghai, China (#20PJ1403500 and #19511103602) and NSFC (#62002118, #61632005 and #61532019). Shufang Zhu is supported by ERC Advanced Grant (#834228) and EU ICT-48 2020 project (#952215). Moshe Vardi is supported by NSF grants IIS-1527668, CCF-1704883, IIS-1830549, and an award from the Maryland Procurement Office.

References

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2018. Synthesis under Assumptions. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, 615–616. AAAI Press.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL Environment Specifications. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, 31–39. AAAI Press.

Bansal, S.; Li, Y.; Tabajara, L.; and Vardi, M. 2020. Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 9766–9774. AAAI Press.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2018. Finite LTL Synthesis with Environment Assumptions and Quality Measures. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, 454–463. AAAI Press.

Camacho, A.; Triantafillou, E.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 3716–3724. AAAI Press.

De Giacomo, G.; and Vardi, M. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 2000–2007. AAAI Press.

Duret-Lutz, A.; and Poitrenaud, D. 2004. SPOT: An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata. In *Proc. 12th Int'l Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 76–83. IEEE Computer Society.

Elgaard, J.; Klarlund, N.; and Möller, A. 1998. Mona 1.x: new techniques for WS1S and WS2S. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, 516–520. Springer.

Giacomo, G. D.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI 15, 1558–1564. AAAI Press. ISBN 9781577357384.

Henriksen, J.; Jensen, J.; Jørgensen, M.; Klarlund, N.; Paige, R.; Rauhe, T.; and Sandholm, A. 1995. Mona: Monadic Second-Order Logic in Practice. In *Proc. 1st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, 89–110. Springer.

Kupferman, O.; and Vardi, M. 2001. Model checking of safety properties. *Formal Methods in System Design* 19(3): 291–314.

Li, J.; Rozier, K. Y.; Pu, G.; Zhang, Y.; and Vardi, M. Y. 2019. SAT-Based Explicit LTLf Satisfiability Checking. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, 2946–2953. AAAI Press.

Li, J.; Zhang, L.; Pu, G.; Vardi, M. Y.; and He, J. 2014. LTL_f Satisfibility Checking. In *ECAI*, 91–98.

Meyer, P. J.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit Reactive Synthesis Strikes Back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, 578–586. Springer.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57. ISSN 0272-5428. doi:10.1109/SFCS.1977.32.

Rintanen, J. 2004. Complexity of Planning with Partial Observability. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 345–354. AAAI.

Rozier, K.; and Vardi, M. 2007. LTL Satisfiability Checking. In *SPIN*, volume 4595 of *LNCS*, 149–167. Springer.

Tabajara, L.; and Vardi, M. 2019. Partitioning Techniques in LTLf Synthesis. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI 19, 5599–5606. AAAI Press.

Zhu, S.; Giacomo, G. D.; Pu, G.; and Vardi, M. Y. 2020. $\mathrm{LTL}f$ Synthesis with Fairness and Stability Assumptions. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 3088–3095. AAAI Press.

Zhu, S.; Pu, G.; and Vardi, M. Y. 2019. First-Order vs. Second-Order Encodings for LTLf-to-Automata Translation. In *Theory and Applications of Models of Computation - 15th Annual Conference, TAMC 2019, Kitakyushu, Japan, April 13-16, 2019, Proceedings*, volume 11436 of *Lecture Notes in Computer Science*, 684–705.

Zhu, S.; Tabajara, L.; Li, J.; Pu, G.; and Vardi, M. 2017. Symbolic LTLf Synthesis. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI 17, 1362–1369. AAAI Press. ISBN 9780999241103.