LTL_f Satisfiability Checking via Formula Progression

Tong Niu[1], Yicong Xu[1], Shengping Xiao[1], Lili Xiao[2], Yanhong Huang[1], Jianwen Li[1]
1. East China Normal University
2. Donghua University

Abstract

Linear Temporal Logic over finite traces, or LTL_f*, is a popular logic to describe specifications with finite behaviors in AI scenarios such as motion planning. Satisfiability is one of the fundamental problems of* LTL_f *and extensive studies have been conducted to speed up the process to check whether a given* LTL_f *formula is satisfiable. This paper presents a new approach, namely* LSCFP*, to solve the problem of* LTL_f *satisfiability checking by leveraging the* formula progression *technique. Compared to previous work,* LSCFP *utilizes formula progression to gather more information propagated along with the search path such that it can find satisfiable models more quickly if the input formula is satisfiable. A comprehensive experimental evaluation has been conducted to show the efficiency of* LSCFP*, and the results suggest that* LSCFP *is able to gain at least 15% performance improvement on checking satisfiable formulas when compared to the state-of-the-art* LTL_f *satisfiability checker aaltaf.*

1. Introduction

Linear Temporal Logic over Finite Traces, abbreviated as LTL_f, is a kind of logic that has emerged as a popular specification language in the AI domain to formalize and validate system behaviors [8]. Unlike standard Linear Temporal Logic (LTL) which is interpreted over infinite traces [16], LTL_f is interpreted over finite traces. While LTL is typically used in formal-verification settings, where we are interested in nonterminating computations, cf. [20], LTL_f is more attractive in AI scenarios focusing on finite behaviors, such as planning [15, 5] and user preferences [4, 19]. Due to the wide spectrum of applications of LTL_f in the AI community, the fundamental problems of LTL_f, e.g., satisfiability [13, 12] and synthesis [11, 22], 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 [7, 6, 1, 2, 21], which makes the logic very attractive in this domain.

This paper focuses on the problem of LTL_f satisfiability checking. Given an LTL_f formula, the satisfiability problem asks whether there is a finite trace that satisfies the formula. The existing solutions for LTL_f satisfiability checking fall into two categories. The first is to reduce the LTL_f satisfiability problem to that of LTL satisfiability, and then leverage the state-of-the-art LTL checkers, cf. [17], to solve LTL_f satisfiability. Thus, LTL_f satisfiability checking can benefit from progress in LTL satisfiability checking. There is, however, an inherent drawback that an extra cost has to be paid when checking LTL formulas, as the tool searches for a "lasso" (a lasso consists of a finite path plus a cycle, representing an infinite trace), whereas models of LTL_f formulas are just finite traces.

Based on this motivation, the second solution is proposed to reduce LTL_f satisfiability checking to the language emptiness check over the finite automaton that accepts the same languages as the input formula. Essentially, one can construct such an NFA (Non-deterministic Finite Automaton) from the given formula, such that the formula is satisfiable if and only if there is a finite trace that can be accepted by the NFA. [13] is the first attempt to implement this solution, and the subsequent work [14] improves the performance significantly by introducing the SAT technique to achieve the *on-the-fly* checking procedure, which is considered the most advanced approach so far.

This paper proposes to check the satisfiability of LTL_f formulas on the fly based on DFA (Deterministic Finite Automata), which is constructed by leveraging the *formula progression* technique [3], and shows that this new approach, namely LSCFP, can potentially achieve a better performance, in particular on satisfiable formulas than that based on NFA [14]. Even though constructing DFA is a much harder task than constructing NFA from an LTL_f formula[1], the DFA state contains more formula information than the NFA state such that a satisfiable model may be detected more quickly.

Take the formula $\varphi = \bigcirc(a\,\mathcal{U}\,b \wedge \mathcal{G}\neg b) \vee \bigcirc a$ as an ex-

[1]In theory, the translation to NFA is exponential blow-up while the translation to DFA is double exponential blow-up.

ample. The CDLSC approach, which is presented in [14], may first construct a successor $\varphi_1 = a \mathcal{U} b \wedge \mathcal{G}\neg b$ of φ (φ_1 is an NFA state) and then determine φ_1 is unsatisfiable. Then the algorithm backtracks to φ and constructs another successor $\varphi_2 = a$ of φ, after which it can return satisfiable. The whole checking process may invoke at least two SAT calls. Meanwhile, the LSCFP approach presented in this paper uses formula progression to construct the successor $\varphi_3 = (a \mathcal{U} b \wedge \mathcal{G}\neg b) \vee a$, which is a DFA state and can immediately be determined as an accepting state. Therefore, LSCFP is able to check the satisfiability of φ by invoking only one SAT call.

We compare LSCFP to CDLSC by conducting a comprehensive experiment on the widely used formulas for benchmarking LTL$_f$ satisfiability. The results indeed affirm our conjecture that LSCFP can perform better than CDLSC on checking satisfiable formulas by achieving a 15% speed-up on average. Notably, LSCFP cannot compete to CDLSC on checking unsatisfiable formulas, the result of which is consistent with the fact that constructing DFA is much more costly than constructing NFA.

The rest of this paper is organized as follows. Section 2 introduces definitions for LTL$_f$ and its satisfiability problem; Section 3 introduces the LSCFP approach in detail; Section 4 presents the experimental results, and finally, Section 5 concludes the paper.

2. Preliminaries

2.1. LTL over Finite Traces (LTL$_f$)

Linear Temporal Logic over finite traces, or LTL$_f$ [8], extends propositional logic with finite-horizon temporal connectives. In particular, LTL$_f$ can be considered as a variant of Linear Temporal Logic (LTL) [16]. Distinguished with LTL which is interpreted over infinite traces, LTL$_f$ is interpreted over finite traces. Given a set of atomic propositions $\mathcal{P}$, the syntax of LTL$_f$ is identical to LTL, and defined as:

$$\varphi ::= tt \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi$$

where tt represents the *true* formula, $p \in \mathcal{P}$ is an atomic proposition, $\neg$ is the *negation*, $\wedge$ is the *and*, $\bigcirc$ is the *strong Next* and $\mathcal{U}$ is the *Until* operator. We also have the corresponding dual operators ff (*false*) for tt, $\vee$ (or) for $\wedge$, $\bullet$ (weak Next) for $\bigcirc$ and $\mathcal{R}$ (Release) for $\mathcal{U}$. Moreover, we use the notation $\mathcal{G}\varphi$ (Global) and $\mathcal{F}\varphi$ (Future) to represent $ff \mathcal{R} \varphi$ and $tt \mathcal{U} \varphi$, respectively. Notably, $\bigcirc$ is the standard *Next* operator, while $\bullet$ is *weak Next*; $\bigcirc$ requires the existence of a successor instance, while $\bullet$ does not. Thus $\bullet\phi$ is always true in the last instance of a finite trace, since no successor exists there.

A finite *trace* $\rho = \rho[0], \rho[1], \ldots, \rho[n]$ ($n \geq 0$) is a sequence of propositional interpretations (sets), in which $\rho[m] \in 2^{\mathcal{P}}$ ($0 \leq m < |\rho|$) is the m-th interpretation of ρ, and $|\rho| = n + 1$ represents the length of ρ. Intuitively, $\rho[m]$ is interpreted as the set of propositions that are $true$ at instant m. We denote ρ_i to represent $\rho[i], \rho[i+1], \ldots, \rho[n]$, which is the suffix of ρ from position i.

LTL$_f$ formulas are interpreted over finite traces. For a finite trace ρ and an LTL$_f$ formula φ, we define the satisfaction relation $\rho \models \varphi$ (i.e., ρ is a model of φ) as follows:

- $\rho \models tt$;
- $\rho \models p$ iff $p \in \rho[0]$, where p is an atomic proposition;
- $\rho \models \neg\varphi$ iff $\rho \not\models \varphi$;
- $\rho \models \varphi_1 \wedge \varphi_2$ iff $\rho \models \varphi_1$ and $\rho \models \varphi_2$;
- $\rho \models \bigcirc\varphi$ iff $|\rho| > 1$ and $\rho_1 \models \varphi$;
- $\rho \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists i with $0 \leq i < |\rho|$ such that $\rho_i \models \varphi_2$, and for every j with $0 \leq j < i$ it holds that $\rho_j \models \varphi_1$.

Two LTL$_f$ formulas φ_1 and φ_2 are semantically equivalent, denoted as $\varphi_1 \equiv \varphi_2$, iff for every finite trace ρ, $\rho \models \varphi_1$ iff $\rho \models \varphi_2$. According to the semantics of LTL$_f$ formulas, it is trivial to have that $ff \equiv \neg tt$, $\bigcirc\phi \equiv \neg\bullet\neg\phi$, $(\phi_1 \mathcal{U} \phi_2) \equiv \neg(\neg\phi_1 \mathcal{R} \neg\phi_2)$ and $\mathcal{G}\phi \equiv \neg\mathcal{F}\neg\phi$. A *literal* is an atom $p \in \mathcal{P}$ or its negation ($\neg p$). We say an LTL$_f$ formula is in *Negation Normal Form* (NNF) if the negation operator appears only in front of an atom. Every LTL$_f$ formula can be converted into its equivalent NNF in linear time. We now introduce below the concept of satisfiability for LTL$_f$ formulas.

Definition 1 (LTL$_f$ Satisfiability)**.** *An* LTL$_f$ *formula φ is satisfiable iff there exists a finite trace $\rho \in (2^{\mathcal{P}})^*$ such that $\rho \models \varphi$; otherwise, it is unsatisfiable.*

Theorem 1 ([8])**.** *Checking the satisfiability of an* LTL$_f$ *formula is PSPACE-complete.*

Notations. We use $cl(\phi)$ to denote the set of subformulas of ϕ. Let A be a set of LTL$_f$ formulas, we denote $\bigwedge A$ to be the formula $\bigwedge_{\psi \in A} \psi$. We say an LTL$_f$ formula ϕ is in *Tail Normal Form* (TNF) if ϕ is in *Negated Normal Form* (NNF) and $\mathcal{N}$-free. Assume ϕ is in NNF, $\mathsf{tnf}(\phi)$ is defined as $t(\phi) \wedge \mathcal{F}Tail$, where $Tail$ is a new atom to identify the last state of satisfying traces (Motivated from [8]), and $t(\phi)$ is an LTL$_f$ formula defined recursively as follows:

1. $t(\phi) = \phi$ if ϕ is $true$, ff or a literal;
2. $t(\bigcirc\psi) = \neg Tail \wedge \bigcirc(t(\psi))$;
3. $t(\bullet\psi) = Tail \vee \bigcirc(t(\psi))$;

4. $t(\phi_1 \wedge \phi_2) = t(\phi_1) \wedge t(\phi_2)$;

5. $t(\phi_1 \vee \phi_2) = t(\phi_1) \vee t(\phi_2)$;

6. $t(\phi_1 \mathcal{U} \phi_2) = (\neg Tail \wedge t(\phi_1)) \mathcal{U}\, t(\phi_2)$;

7. $t(\phi_1 \mathcal{R} \phi_2) = (Tail \vee t(\phi_1)) \mathcal{R}\, t(\phi_2)$.

Theorem 2 ([12]). *ϕ is satisfiable iff $\mathsf{tnf}(\phi)$ is satisfiable.*

In the rest of the paper, unless clearly specified, the input LTL_f formula is in TNF.

2.2. Transition-based Deterministic Finite Automaton (TDFA)

The Transition-based Deterministic Finite Automaton (TDFA) is a variant of the Deterministic Finite Automaton (DFA), which identifies the accepting condition on the transitions instead of states.

Definition 2 (Transition-based DFA [18]). *A transition-based DFA (TDFA) is a tuple $\mathcal{A} = (2^{\mathcal{P}}, S, s_0, \delta, T)$ where*

- *$2^{\mathcal{P}}$ is the alphabet;*
- *S is the set of states;*
- *$s_0 \in S$ is the initial state;*
- *$\delta : S \times 2^{\mathcal{P}} \to S$ is the transition function;*
- *$T \subseteq \delta$ is the set of accepting transitions.*

For simplicity, we use the notation $s_1 \xrightarrow{\omega} s_2$ to denote $\delta(s_1, \omega) = s_2$. The run r of a TDFA $\mathcal{A}$ on a finite trace $\rho = \rho[0], \rho[1], \cdots, \rho[n] \in (2^{\mathcal{P}})^+$ is a finite state sequence $r = s_0, s_1, \cdots, s_n$ such that s_0 is the initial state, $s_i \xrightarrow{\rho[i]} s_{i+1}$ is true for $0 \leq i < n$. Note that runs of TDFA do not need to include the destination state of the last transition, which is implicit $s_{n+1} = \delta(s_n, \rho[n])$, since the starting state (s_n) together with the labels of the transition $(\rho[n])$ are sufficient to determine the destination. r is called *acyclic* iff $(s_i = s_j) \Leftrightarrow (i = j)$ for $0 \leq i, j < n$. Also, we say that ρ *runs across* s_i iff s_i is in the corresponding run r. The trace ρ is accepted by $\mathcal{A}$ iff the corresponding run r ends with an accepting transition, i.e., $\delta(s_n, \rho[n]) \in T$. The set of finite traces accepted by a TDFA $\mathcal{A}$ is the language of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$.

According to [18], TDFA has the same expressiveness as the normal DFA, and for an LTL_f formula φ, there is a TDFA $\mathcal{A}_\varphi$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. As a result, the LTL_f satisfiability-checking problem can be solved on the corresponding TDFA.

3. LTL_f Satisfiability Checking via Formula Progression (LSCFP)

In this section, we first introduce the concept of *formula progression* for LTL_f formulas and how to construct the TDFA via formula progression. Then we produce an on-the-fly satisfiability-checking framework along with the TDFA construction.

3.1 LTL_f-to-TDFA via Formula Progression

The *formula progression* technique originates in [3] for goal planning with temporal logic. A definition of LTL_f progression has been used in [10], and here we adapt the definition to a finite trace instead of a single proposition.

Definition 3 (Formula Progression for LTL_f). *Given an LTL_f formula φ and a non-empty finite trace ρ, the progression formula $\mathsf{fp}(\varphi, \rho)$ is recursively defined as follows:*

- *$\mathsf{fp}(tt, \rho) = tt$ and $\mathsf{fp}(ff, \rho) = ff$;*
- *$\mathsf{fp}(p, \rho) = tt$ if $p \in \rho[0]$; $\mathsf{fp}(p, \rho) = ff$ if $p \notin \rho[0]$;*
- *$\mathsf{fp}(\neg\varphi, \rho) = \neg\mathsf{fp}(\varphi, \rho)$;*
- *$\mathsf{fp}(\varphi_1 \wedge \varphi_2, \rho) = \mathsf{fp}(\varphi_1, \rho) \wedge \mathsf{fp}(\varphi_2, \rho)$;*
- *$\mathsf{fp}(\varphi_1 \vee \varphi_2, \rho) = \mathsf{fp}(\varphi_1, \rho) \vee \mathsf{fp}(\varphi_2, \rho)$;*
- *$\mathsf{fp}(\bigcirc\varphi, \rho) = \varphi$ if $|\rho| = 1$; Else $\mathsf{fp}(\bigcirc\varphi, \rho) = \mathsf{fp}(\varphi, \rho_1)$;*
- *$\mathsf{fp}(\bullet\varphi, \rho) = \varphi$ if $|\rho| = 1$; Else $\mathsf{fp}(\bullet\varphi, \rho) = \mathsf{fp}(\varphi, \rho_1)$;*
- *$\mathsf{fp}(\varphi_1 \mathcal{U} \varphi_2, \rho) = \mathsf{fp}(\varphi_2, \rho) \vee (\mathsf{fp}(\varphi_1, \rho) \wedge \mathsf{fp}(\bigcirc(\varphi_1 \mathcal{U} \varphi_2), \rho))$;*
- *$\mathsf{fp}(\varphi_1 \mathcal{R} \varphi_2, \rho) = \mathsf{fp}(\varphi_2, \rho) \wedge (\mathsf{fp}(\varphi_1, \rho) \vee \mathsf{fp}(\bullet(\varphi_1 \mathcal{U} \varphi_2), \rho))$.*

The following lemmas are not hard to obtain based on Definition 3, whose proofs are omitted here.

Lemma 1. *Given an LTL_f formula φ and two non-empty finite traces ρ_1 and ρ_2, $\rho_2 \models \mathsf{fp}(\varphi, \rho_1)$ implies $\rho_1 \cdot \rho_2 \models \varphi$.*

Lemma 2. *Given an LTL_f formula φ and two non-empty finite traces ρ_1 and ρ_2, it holds that $\mathsf{fp}(\mathsf{fp}(\varphi, \rho_1), \rho_2) = \mathsf{fp}(\varphi, \rho_1 \cdot \rho_2)$.*

Lemma 3. *Given an LTL_f formula and a non-empty finite trace ρ, $\rho \models \varphi$ implies $\rho_i \models \mathsf{fp}(\varphi, \rho^i)$ for every $0 \leq i < |\rho|$.*

Now we re-construct the TDFA for an LTL_f formula.

Definition 4 (LTL_f to TDFA). *Given an LTL_f formula φ, the* TDFA *$\mathcal{A}_\varphi$ is a tuple $(2^{\mathcal{P}}, S, \delta, s_0, T)$ such that*

- $2^{\mathcal{P}}$ *is the alphabet, where* $\mathcal{P}$ *is the set of atoms of* φ*;*
- $S = \{\varphi \cup \mathsf{fp}(\varphi, \rho) \mid \forall \rho \in (2^{\mathcal{P}})^+\}$ *is the set of states;*
- $s_0 = \varphi$ *is the initial state;*
- $\delta : S \times 2^{\mathcal{P}} \to S$ *is the transition function such that* $\delta(s, \sigma) = \mathsf{fp}(s, \sigma)$ *for* $s \in S$ *and* $\sigma \in 2^{\mathcal{P}}$ *(Here* σ *is considered a trace with length 1);*
- $T = \{s_1 \xrightarrow{\sigma} s_2 \in \delta \mid \sigma \models s_1\}$ *is the set of accepting transitions.*

Theorem 3. *Given an* LTL$_f$ *formula* φ *and the* TDFA $\mathcal{A}_\varphi$ *constructed by Definition 4, it holds that* $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$.

Proof. Let $|\rho| = n + 1$ ($n \geq 0$) and the corresponding run r of $\mathcal{A}_\varphi$ on ρ is $s_0, s_1, \ldots, s_n$, where $s_0 = \varphi$.

($\Leftarrow$) According to Definition 4, ρ is accepted by $\mathcal{A}_\varphi$ implies $(\rho_n = \rho[n]) \models s_n$ and $s_n = \mathsf{fp}(\varphi, \rho^n)$. Then from Lemma 1, we have $(\rho^n \cdot \rho_n = \rho) \models (s_0 = \varphi)$.

($\Rightarrow$) First from Definition 4, every $\mathsf{fp}(\varphi, \rho^i)$ for $0 \leq i \leq n$ is a state of $\mathcal{A}_\varphi$. Secondly, $\delta(\mathsf{fp}(\varphi, \rho^i), \rho[i]) = \mathsf{fp}(\varphi, \rho^{i+1})$ is true for $0 \leq i \leq n$, because $\mathsf{fp}(\varphi, \rho^{i+1}) = \mathsf{fp}(\mathsf{fp}(\varphi, \rho^i), \rho[i])$ is true (Lemma 2). Therefore, let $s_i = \mathsf{fp}(\varphi, \rho^i)$ $(0 \leq i \leq n)$ and the state sequence $r = s_0, s_1, \ldots, s_n$ is a run of $\mathcal{A}_\varphi$ on ρ. Finally, $\rho \models \varphi$ implies that $\rho_n \models (s_n = \mathsf{fp}(\varphi, \rho^n))$ is true because of Lemma 3. So ρ is accepted by $\mathcal{A}_\varphi$. □

3.2 On-the-fly Checking Framework

The CDLSC approach [12] is an on-the-fly satisfiability checking framework along with the NFA construction. Compared to that, LSCFP completes the satisfiability checking over the on-the-fly TDFA construction instead. In a high-level description, LSCFP combines SAT computation together with formula progression to construct TDFA states instead of NFA ones.

Given an LTL$_f$ formula φ whose closure is $cl(\varphi)$, every state s of the corresponding NFA is a conjunction of subformulas of φ, i.e., $s \subseteq cl(\varphi)$. To compute one successor of φ, the SAT-based method proposed in [12] first converts φ to its *neXt Normal Form* (XNF) $\mathsf{xnf}(\varphi)$, in which there are no Until or Release subformulas of ϕ in the atomic level or connected by Boolean operators. For example, $\varphi_1 = ((\neg Tail \wedge a)\mathcal{U}b)$ is not in XNF because there is an Until subformula (φ itself) in the atomic level, while $\varphi_2 = (b \vee (\neg Tail \wedge a \wedge (\mathcal{X}((\neg Tail \wedge a)\mathcal{U}b))))$ is, though φ_1 and φ_2 are semantically equivalent. In fact, every LTL$_f$ formula ϕ has a linear-time conversion to an equivalent formula in XNF. By treating $\mathsf{xnf}(\varphi)$ as a propositional logic, denoted as $\mathsf{xnf}(\varphi)^p$, a Boolean SAT solver is able to return an assignment that indicates the current conditions (label of the NFA transition) and one next state of φ. Consider φ_2 as the example, the SAT solver may return $\{a, b, \neg Tail, \mathcal{X}((\neg Tail \wedge a)\mathcal{U}b)\}$ as an assignment, based on which CDLSC identifies that $\varphi_1 = (\neg Tail \wedge a)\mathcal{U}b$ is a successor of itself under the condition $a \wedge b \wedge \neg Tail$.

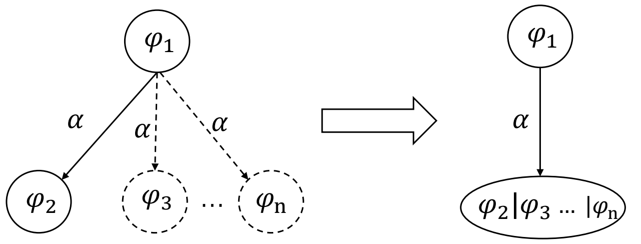

Figure 1. A schema to illustrate CDLSC (left) vs. LSCFP (right) on state construction. CDLSC needs to invoke up to n SAT calls to compute the successors of φ_1, while LSCFP needs only one SAT call plus one formula progression to obtain all the state information.

Algorithm 1 LSCFP: LTL$_f$ Satisfiability Checking via Formula Progression.

```
Require: An LTL_f formula $\varphi$.
Ensure: SAT or UNSAT.
 1: if $Tail \wedge \mathsf{xnf}(\varphi)^p$ is satisfiable then
 2:     return SAT;
 3: Let $\psi = \mathsf{xnf}(\varphi) \wedge \neg\mathcal{X}(\varphi)$;
 4: while $\psi^p$ is satisfiable do
 5:     Let $A$ be a propositional assignment for $\psi^p$;
 6:     Let $L(A)$ be the set of literals extracted from $A$, i.e.,
        $L(A)$ represents the current conditions;
 7:     Let $\varphi' = \mathsf{fp}(\psi \wedge \varphi, L(A))$, i.e., be the DFA successor
        state of $\varphi$ computed via formula progression;
 8:     if LSCFP ($\varphi'$) returns SAT then
 9:         return SAT;
10:     Let $\psi = \mathsf{xnf}(\phi) \wedge \neg\mathcal{X}(\varphi')$;
11: return UNSAT;
```

Compared to CDLSC, LSCFP takes the current conditions as the input to compute $\mathsf{fp}(\varphi, \{a, b, \neg Tail\})$, which from Definition 3 equals $\varphi_1 \vee tt \equiv tt$. Therefore, LSCFP identifies tt as a TDFA successor state of φ under the conditions $a \wedge b \wedge \neg Tail$. In general, CDLSC computes only one successor under the fixed conditions, while LSCFP is able to compute all possible successors under the same conditions. Figure 1 shows the differences between LSCFP and CDLSC on constructing automata states.

The implementation of LSCFP is shown in Algorithm 1. Line 1 first checks whether the input formula φ is a *final state* of the TDFA, which is equivalent to checking whether $Tail \wedge (\mathsf{xnf}(s))^p$ is satisfiable [12]. If φ is not a final state, the while loop (Line 4-10) searches the successors of φ recursively to identify whether they can be final. Line 3 and 10 block the states already checked during the search. If no new state (assignment) can be generated from the SAT solver, the algorithm terminates with an unsatisfiable result. In the while loop, Line 5 obtains the propositional assignment A from the SAT solver, Line 6 extracts the conditions (label) of the current transition indicated by A, and then Line utilizes the formula progression technique to compute the corresponding TDFA successor.

The theorem below provides the theoretical guarantee to the correctness of Algorithm 1.

Theorem 4. *The input* LTL$_f$ *formula is satisfiable if and only if Algorithm 1 returns* SAT.

Proof. First of all, it is not hard to see that Algorithm 1 constructs exactly the TDFA in Definition 4. Secondly, let $\rho = \omega_0\omega_1 \ldots \omega_n$ $(n \geq 0)$ be a non-empty trace and the corresponding run on the TDFA constructed in Algorithm 1 be $r = s_0(\varphi) \xrightarrow{\omega_0} s_1 \ldots s_n \xrightarrow{\omega_n} s_{n+1}$. Notably, since Algorithm 1 constructs TDFA, the run of the automaton on a given trace is unique. According to Theorem 3, $\rho \models \varphi$ if and only if $\omega_n \models s_n$ holds, which indicates that s_n is a final state. Then, s_n is a final state if and only if Algorithm 1 can return SAT at Line 2. As a result, $\rho \models \varphi$ if and only if Algorithm 1 returns SAT at Line 2 when taking s_n as the input in the recursive checking process. □

4. Experimental Evaluations

Experimental Setup. We implement LSCFP in the state-of-the-art LTL$_f$ satisfiability solver *aaltaf*[2], which implemented CDLSC, and use Minisat 2.2.0 [9] as the SAT engine. To evaluate the performance of LSCFP, we compared it to CDLSC on the widely used LTL$_f$ benchmarks that are presented in [12], in which there are 8645 formulas in total.

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. When we ran the experiments, each tool was run on a dedicated node, which guarantees that no CPU or memory conflict with other jobs can occur. For each tested formula, we set the timeout to 120 seconds, measuring execution time with Unix time. Excluding timeouts, the results from both LSCFP and CDLSC are consistent.

Results. Figure 2 shows the comparison between LSCFP and CDLSC on satisfiable formulas in terms of time cost. LSCFP performs better than CDLSC on more than 60% of the tested satisfiable cases, and for some particular cases, LSCFP can be 10X times faster than CDLSC. LSCFP is able to solve 6380 satisfiable instances within timeout, while CDLSC solves the number of 6419. Notably, there are 39 formulas that cannot be solved by LSCFP within timeout, which slows down its overall performance. However, LSCFP is still able to obtain a 15% improvement on average when compared to CDLSC on checking satisfiable formulas.

[2]https://github.com/lijwen2748/aaltaf

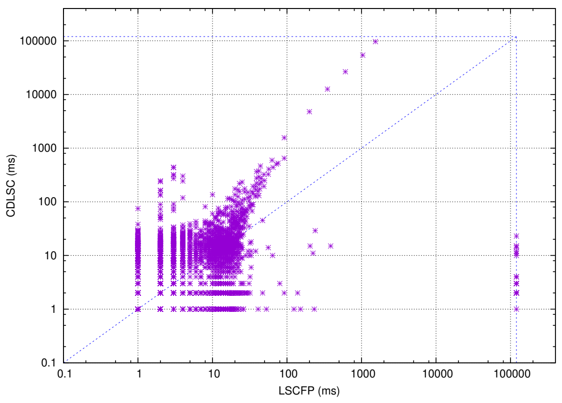

Figure 2. Scatter plots for LSCFP vs. CDLSC on satisfiable formulas.

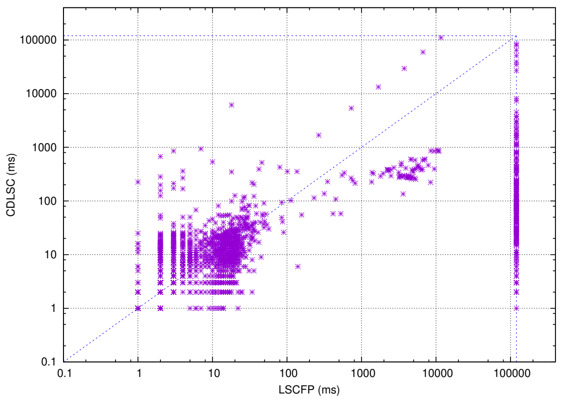

Figure 3. Scatter plots for LSCFP vs. CDLSC on unsatisfiable formulas.

Figure 3 shows the comparison between LSCFP and CDLSC on unsatisfiable formulas in terms of time cost. For instances that can be solved by both approaches, LSCFP gains a similar performance with CDLSC. However, LSCFP has more timeout cases than CDLSC. In total, LSCFP solves the amount of 1904 unsatisfiable formulas, while CDLSC solves the amount of 2141. So there are 237 more unsatisfiable instances that can be timeout for LSCFP than CDLSC, which is the main reason why LSCFP is not competitive in solving unsatisfiable formulas. In our conjecture, LSCFP needs to enumerate all TDFA states in the worst case to check unsatisfiability, and this is a much heavier task than enumerating NFA states. Also, CDLSC leverages more advanced convergent techniques by using the *conflict sequence*, which we plan to explore in LSCFP to speed up the performance.

5 Concluding Remarks

In this paper, we present LSCFP, a new on-the-fly satisfiability checking approach for LTL$_f$ formulas. Compared to the existing CDLSC method, LSCFP combines SAT invoking together with the formula progression technique to compute the DFA states instead of NFA ones. The benefit is to accelerate the checking performance, in particular for satisfiable instances. Our experimental results affirm our conjecture on such a benefit. Currently, LSCFP cannot perform well on checking unsatisfiable formulas, and we plan to introduce the ideas in CDLSC to enable fast convergent by overapproaximating reachable states in future work.

6 Acknowledgment

We thank anonymous reviewers for their helpful comments. Tong Niu and Yicong Xu have equal contributions to this paper. Jianwen Li is the corresponding author. This work is supported by the National Natural Science Foundation of China (Grant #U21B2015 and #62002118), the Shanghai Collaborative Innovation Center of Trusted Industry Internet Software, and the National Key Research and Development Program (2022YFB3305202).

References

[1] B. Aminof, G. De Giacomo, A. Murano, and S. Rubin. 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*, pages 615–616. AAAI Press, 2018.

[2] B. Aminof, G. De Giacomo, A. Murano, and S. Rubin. 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*, pages 31–39. AAAI Press, 2019.

[3] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[4] M. Bienvenu, C. Fritz, and S. A. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175:1308 – 1345, 2011.

[5] A. Camacho, J. Baier, C. Muise, and A. McIlraith. Bridging the gap between LTL synthesis and automated planning. Technical report, U. Toronto, 2017.

[6] A. Camacho, M. Bienvenu, and S. A. McIlraith. 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*, pages 454–463. AAAI Press, 2018.

[7] A. Camacho, E. Triantafillou, C. J. Muise, J. A. Baier, and S. A. McIlraith. 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*, pages 3716–3724. AAAI Press, 2017.

[8] G. De Giacomo and M. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 2000–2007. AAAI Press, 2013.

[9] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2003.

[10] G. D. Giacomo, M. Favorito, J. Li, M. Y. Vardi, S. Xiao, and S. Zhu. Ltlf synthesis as and-or graph search: Knowledge compilation at work. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 3292–3298. AAAI Press, 2022.

[11] G. D. Giacomo and M. Y. Vardi. Synthesis for ltl and ldl on finite traces. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI 15, pages 1558–1564. AAAI Press, 2015.

[12] J. Li, K. Y. Rozier, G. Pu, Y. Zhang, and M. Y. Vardi. Sat-based explicit ltlf satisfiability checking. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 2946–2953. AAAI Press, 2019.

[13] J. Li, L. Zhang, G. Pu, M. Y. Vardi, and J. He. LTL$_f$ satisfibility checking. In *ECAI*, pages 91–98, 2014.

[14] J. Li, S. Zhu, G. Pu, L. Zhang, and M. Y. Vardi. Sat-based explicit ltl reasoning and its application to satisfiability checking. *Formal Methods in System Design*, pages 1–27, 2019.

[15] F. Patrizi, N. Lipoveztky, G. De Giacomo, and H. Geffner. Computing infinite plans for LTL goals using a classical planner. In *IJCAI*, pages 2003–2008. AAAI Press, 2011.

[16] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, Oct 1977.

[17] K. Rozier and M. Vardi. LTL satisfiability checking. In *SPIN*, volume 4595 of *LNCS*, pages 149–167. Springer, 2007.

[18] Y. Shi, S. Xiao, J. Li, J. Guo, and G. Pu. Sat-based automata construction for ltl over finite traces. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 1–10, 2020.

[19] S. Sohrabi, J. A. Baier, and S. A. McIlraith. Preferred explanations: Theory and generation via planning. In *AAAI*, pages 261–267, August 2011.

[20] M. Vardi. Automata-theoretic model checking revisited. In *VMCAI*, LNCS 4349, pages 137–150. Springer, 2007.

[21] S. Zhu, G. D. Giacomo, G. Pu, and M. Y. Vardi. Ltl$f$ synthesis with fairness and stability assumptions. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3088–3095. AAAI Press, 2020.

[22] S. Zhu, L. Tabajara, J. Li, G. Pu, and M. Vardi. Symbolic ltlf synthesis. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI 17, pages 1362–1369. AAAI Press, 2017.