\textbf{LTL}_f Satisfiability Checking

\textbf{Jianwen Li}\textsuperscript{1} and \textbf{Lijun Zhang}\textsuperscript{2} and \textbf{Geguang Pu}\textsuperscript{1} and \textbf{Moshe Y. Vardi}\textsuperscript{3} and \textbf{Jifeng He}\textsuperscript{1}

\textbf{Abstract.} We consider here Linear Temporal Logic (LTL) formulas interpreted over \textit{finite} traces. We denote this logic by LTL\textsubscript{f}. The existing approach for LTL\textsubscript{f} satisfiability checking is based on a reduction to standard LTL satisfiability checking. We describe here a novel direct approach to LTL\textsubscript{f} satisfiability checking, where we take advantage of the difference in the semantics between LTL and LTL\textsubscript{f}. While LTL satisfiability checking requires finding a \textit{fair cycle} in an appropriate transition system, here we need to search only for a finite trace. This enables us to introduce specialized heuristics, where we also exploit recent progress in Boolean SAT solving. We have implemented our approach in a prototype tool and experiments show that our approach outperforms existing approaches.

1 Introduction

Linear Temporal Logic (LTL) was first introduced into computer science as a property language for the verification for non-terminating reactive systems [9]. Following that, many researches in AI have been attracted by LTL's rich expressiveness. Examples of applications of LTL in AI include temporally extended goals [3], plan constraints [1], and user preferences [13].

In a recent paper [5], De Giacomo and Vardi argued that while standard LTL is interpreted over \textit{infinite} traces, cf. [9], AI applications are typically interested only in \textit{finite} traces. For example, temporally extended goals are viewed as finite desirable sequences of states and a plan is correct if its execution succeeds in yielding one of these desirable sequences. Also in the area of business-process modeling, temporal specifications for declarative workflows are interpreted over finite traces [14]. De Giacomo and Vardi, therefore, introduced LTL\textsubscript{f}, which has the same syntax as LTL but is interpreted over finite traces.

In the formal-verification community there is by now a rich body of knowledge regarding automated-reasoning support for LTL. On one hand, there are solid theoretical foundations, cf. [15]. On the other hand, mature software tools have been developed, such as SPOT [4]. Extensive research has been conducted to evaluate these tools, cf. [10]. While the basic theory for LTL\textsubscript{f} was presented at [5], no tool has yet to be developed for LTL\textsubscript{f}, to the best of our knowledge. Our goal in this paper is to address this gap.

Our main focus here is on the \textit{satisfiability problem}, which asks if a given formula has satisfying model. This most basic automated-reasoning problem has attracted a fair amount of attention for LTL over the past few years as a principled approach to \textit{property assurance}, which seeks to eliminate errors when writing LTL properties, cf. [10, 8].

De Giacomo and Vardi studied the computational complexity of LTL\textsubscript{f} satisfiability and showed that it is \textit{PSPACE}-complete, which is the same complexity as for LTL satisfiability [12]. Their proof of the upper bound uses a reduction of LTL\textsubscript{f} satisfiability to LTL satisfiability. That is, for an LTL\textsubscript{f} formula \( \phi \), one can create an LTL formula \( \phi' \) such that \( \phi \) is satisfiable if \( \phi' \) is satisfiable; furthermore, the translation from \( \phi \) to \( \phi' \) involves only a linear blow-up. The reduction to LTL satisfiability problem can, therefore, take advantage of existing LTL satisfiability solvers [11, 8]. On the other hand, LTL satisfiability checking requires reasoning about infinite traces, which is quite nontrivial algorithmically, cf. [2], due to the required fair-cycle test. Such reasoning is not required for LTL\textsubscript{f} satisfiability. A reduction to LTL satisfiability, therefore, may add unnecessary overhead to LTL\textsubscript{f} satisfiability checking.

This paper approaches the LTL\textsubscript{f} satisfiability problem directly. We develop a direct, and more efficient, algorithm for checking satisfiability of LTL\textsubscript{f}, leveraging the existing body of knowledge concerning LTL satisfiability checking. The finite-trace semantics for LTL\textsubscript{f} is fully exploited, leading to considerable simplification of the decision procedure and significant performance boost. The finite-trace semantics also enables several heuristics that are not applicable to LTL satisfiability checking. We also leverage the power of advanced Boolean SAT solvers in our decision procedure. We have implemented the new approach and experiments show that this approach significantly outperforms the reduction to LTL satisfiability problems.

The paper is organized as follows. We first introduce the definition of LTL\textsubscript{f}, the satisfiability problem, and the associated transition system in Section 2. We then propose a direct satisfiability-checking framework in Section 3. We discuss various optimization strategies in Section 4, and present experimental results in Section 5. Section 6 concludes the paper.

2 Preliminaries

2.1 LTL over Finite Traces

The logic LTL\textsubscript{f} is a variant of LTL. Classical LTL formulas are interpreted on infinite traces, whereas LTL\textsubscript{f} formulas are defined over the finite traces. Given a set \( \mathcal{P} \) of atomic propositions, an LTL\textsubscript{f} formula \( \phi \) has the form:

\[
\phi ::= \text{tt} \mid \text{ff} \mid p \mid \neg \phi \mid \phi \lor \phi \mid \phi \land \phi \mid X\phi \mid X_w\phi \mid \phi U \phi \mid \phi R \phi
\]

where \( X \) (strong Next), \( X_w \) (weak Next), \( U \) (Until), and \( R \) (Release) are temporal operators. We have \( X\phi \equiv \neg X^\omega \phi \) and \( \phi_1 R \phi_2 \equiv \neg(X \phi_1 U \neg X \phi_2) \). Note that in LTL\textsubscript{f}, \( X \phi \equiv X_w \phi \) is not true, which is however the case in LTL.

For an atom \( a \in \mathcal{P} \), we call it or its negation \( \neg a \) a literal. We use the set \( L \) to denote the set of literals, i.e., \( L = \mathcal{P} \cup \{\neg a | a \in \mathcal{P} \} \). Other boolean operators, such as \( \rightarrow \) and \( \leftrightarrow \), can be represented by the combination \( \neg \lor \) or \( \neg \land \), respectively, and we denote the constant \textit{true} as \texttt{tt} and \textit{false} as \texttt{ff}. Moreover, we use the notations \( G \phi \).
Then we define \( \phi \) that in LTL, where neither \( X \) unsatisfiable, while \((Xw_\phi \equiv \neg X \neg \phi)\). Theorem 1: \( mua \phi \) LTL defined analogously.

Proof Sketch: It is easy to reduce the LTL satisfiability to LTL satisfiability:

1. Introduce a proposition “Tail”;
2. Require that Tail holds at position 0;
3. Require also that Tail stays tt until it turns into ff, and after that stays ff forever (TailU(G~Tail)).

4. The LTL formula \( \phi \) is translated into a corresponding LTL formula in the following way:
   - \( t(p) \rightarrow p \), where \( p \) is a literal;
   - \( t(\neg \phi) = \neg t(\phi) \);
   - \( t(\phi_1 \land \phi_2) \rightarrow t(\phi_1) \land t(\phi_2) \);
   - \( t(\phi_1 \lor \phi_2) \rightarrow t(\phi_1) \lor t(\phi_2) \);
   - \( t(X\psi) \rightarrow X(Tail \land t(\psi)) \);
   - \( t(\phi_1 U \phi_2) \rightarrow \phi_1 U (Tail \land t(\phi_2)) \);

(The translation here does not require \( \phi \) in NNF. Thus the \( Xw_\phi \) and \( R \) operators can be handled by the rules \( Xw_\phi \equiv \neg X \neg \phi \) and \( \phi_1 R \phi_2 \equiv \neg (\neg \phi_1 U \neg \phi_2) \).) Finally one can refer to [5] that \( \phi \) is satisfiable iff \( Tail \land U(TailU(G~Tail)) \) and \( t(\phi) \) is satisfiable. Also, a PSPACE lower bound is shown in [5] by reduction from STRIPS Planning.

The reduction approach can take advantage of existing LTL satisfiability solvers. But, there may be an overhead as we need to find a fair cycle during LTL satisfiability checking, which is not necessary in LTL checking.

2.3 LTL\(_f\) Transition System

In [8], Li et al. have proposed using transition systems for checking satisfiability of LTL formulas. Here we adapt this approach to LTL\(_f\). First, we define the normal form for LTL\(_f\) formulas.

Definition 2 (Normal Form). The normal form of an LTL\(_f\) formula \( \phi \), denoted as NF(\( \phi \)), is a formula set defined as follows:

- \( NF(\phi) = \{ \phi \land X(t) \} \) if \( \phi \) is a literal. If \( \phi = ff \), we define \( NF(\phi) = \emptyset \);
- \( NF(Xw_\phi / Xw_\psi) = \{ \phi \land X(\psi) \mid \psi \in DF(\phi) \} \);
- \( NF(\phi_1 U \phi_2) = NF(\phi_1) \cup NF(\phi_1 \land X(\phi_1 U \phi_2)) \);
- \( NF(\phi_1 R \phi_2) = NF(\phi_1 \land \phi_2) \cup NF(\phi_2 \land X(\phi_1 R \phi_2)) \);
- \( NF(\phi_1 \lor \phi_2) = NF(\phi_1) \lor NF(\phi_2) \);
- \( NF(\phi_1 \land \phi_2) = \{ \phi_1 \land \phi_2 \land X(\psi_1 \land \psi_2) \mid \forall i = 1, 2, \phi_i \land X(\psi_i) \in NF(\phi_i) \} \);
- For each \( \alpha_i \land X \phi_i \in NF(\phi) \), we say it a clause of \( NF(\phi) \).

(Although the normal forms of X and Xw formulas are the same, we do distinguished between them through the accepting conditions introduced below.) Intuitively, each clause \( \alpha_i \land X \phi_i \) of \( NF(\phi) \) indicates that the propositional formula \( \alpha_i \) should hold now and then \( \phi_i \) should hold in the next state. For \( \phi_i \), we can also compute its normal form. We can repeat this procedure until no new states are required.

Definition 3 (LTL\(_f\) Transition System). Let \( \phi \) be the input formula. The labeled transition system \( T_{\phi} \) is a tuple \( \langle Act, S_\phi, \rightarrow, \phi \rangle \), where: 1. \( \phi \) is the initial state; 2. Act is the set of conjunctive formulas over \( L_\phi \); 3. the transition relation \( \rightarrow \subseteq S_\phi \times Act \times S_\phi \) is defined by: \( \psi_1 \rightarrow \psi_2 \) iff there exists \( \alpha \land X(\psi_1) \in NF(\psi_1) \); and 4. \( S_\phi \) is the smallest set of formulas such that \( \psi_1 \in S_\phi \) and \( \psi_1 \rightarrow \psi_2 \) implies \( \psi_2 \in S_\phi \).

Note that in LTL transition systems the ff state can be deleted, as it can never be part of a fair cycle. This state must be kept in LTL\(_f\) transition systems: a finite trace that reach ff may be accepted in
ŁTₐ, cf. Xₐ ff. Nevertheless, ff edges are not allowed both in ŁTₐ and ŁTL transition systems.

A run of T₀ on finite trace η = ω₀ω₁...ωₙ ∈ Σ⁺ is a sequence s₀ ω₀s₁ ω₁...sn ωnsn₊₁ such that s₀ = φ and for every 0 ≤ i ≤ n it holds ωᵢ = αᵢ. We say ψ is reachable from φ iff there is a run of T₀ such that the final state is ψ.

3 ŁTₐ Satisfiability-Checking Framework

In this section we present our framework for checking satisfiability of ŁTₐ formulas. First we show a simple lemma concerning finite sequences of length 1.

**Lemma 1.** For a finite trace η ∈ Σ⁺ and ŁTₐ formula φ, if |η| = 1 then η |= φ holds iff:

- η = tt and η ≠ ff;
- If φ = p is a literal, then return true if φ ∈ η. otherwise return false;
- If φ = φ₁ ∧ φ₂, then return η |= φ₁ and η |= φ₂;
- If φ = φ₁ ∨ φ₂, then return η |= φ₁ or η |= φ₂;
- If φ = X φ₂, then return false;
- If φ = Xw φ₂, then return true;
- If φ = φ₁ U φ₂ or φ = φ₁ R φ₂, then return η |= φ₂.

**Proof.** This lemma can be directly proven from the semantics of ŁT₁ formulas by fixing |η| = 1.

Now we characterize the satisfaction relation for finite sequences:

**Lemma 2.** For a finite trace η = ω₀ω₁...ωₙ ∈ Σ⁺ and ŁTₐ formula φ,

1. If n = 0, then η |= φ iff there exists α₀ ∈ Xφ₁ ∈ NF(φ) such that ω₀ |= α₀ and CF(α₀) |= φ;
2. If n ≥ 1, then η |= φ iff there exists α₀ ∈ Xφ₁ ∈ NF(φ) such that ω₀ |= α₀ and η₁ |= φ;
3. η |= φ iff there exists a run φ = ω₀ → α₀φ₁ → α₁φ₂ → α₂... → αₙ₊₁φₙ₊₁ in T₀ such that for every 0 ≤ i ≤ n it holds that ωᵢ |= αᵢ and ηᵢ |= φ.

**Proof.** 1. CF(α₀) is treated to be a finite trace whose length is 1. We prove the first item by structural induction over φ.

- If φ = p, then η |= φ iff ω₀ |= p and CF(φ) |= φ hold, where p ∧ X tt is actually in NF(φ);
- If φ = φ₁ ∧ φ₂, then η |= φ holds iff η |= φ₁ and η |= φ₂ hold, and if by induction hypothesis, there exists βᵢ ∈ Xψᵢ in NF(φ₁) such that ω₀ |= βᵢ and CF(βᵢ) |= φ₁ (i = 1, 2). Let α₁ = β₁ ∧ β₂ and φ' = ψ₁ ∨ ψ₂, then according to Definition 2 we know α₁ ∈ Xφ₁ ∈ NF(φ), and ω₀ |= α₀ and CF(α₁) |= φ hold; The proof for the case when φ = φ₁ ∨ φ₂ is similar;
- Note that η |= Xψ is always false, and if φ = Xwψ then from Lemma 1 it is always true that η |= Xwψ if tt ∧ Xψ ∈ NF(φ) and tt |= Xψ;
- If φ = φ₁ U φ₂, then η |= φ holds iff η |= φ₂ holds from Lemma 1, and if by induction hypothesis, there exists α₁ ∈ Xφ₁ ∈ NF(φ₂) such that ω₀ |= α₁ and CF(α₁) |= φ₂, and thus CF(α₁) |= φ according to ŁTₐ semantics. From Definition 2 we know as well that α₁ ∈ Xφ₁ is in NF(φ), thus the proof is done; The proof for the case when φ = φ₁ R φ₂ is similar;
- If φ = Xφ₂ or φ = Xwφ₂, since |η| > 1 so it is obviously true that η |= φ iff ω₀ |= tt and η₁ |= φ₂ hold according to ŁTₐ semantics, and obviously tt ∧ Xφ₂ is in NF(φ);
- If φ = φ₁ ∧ φ₂, then η |= φ iff η |= φ₁ and η |= φ₂, and if by induction hypothesis, there exists β₁ ∧ Xψ₁ in NF(φ₁) such that ω₀ |= β₁ and η₁ |= ψ₁ hold, and if ω₀ |= β₁ ∧ β₂ and η₁ |= ψ₁ ∧ ψ₂ hold, in which (β₁ ∧ β₂ ∧ X(ψ₁ ∧ ψ₂)) is indeed in NF(φ); The case when φ = φ₁ ∨ φ₂ is similar;
- If φ = φ₁ U φ₂, then η |= φ iff η |= φ₂ or η |= (φ₁ ∧ Xφ). If η |= φ₂ holds, then by induction hypothesis iff there exists α₁ ∧ Xφ₁ ∈ NF(φ₂) such that ω₀ |= α₁ and η₁ |= φ₁. According to Definition 2 we know α₁ ∧ Xφ₁ is also NF(φ₂). On the other hand, if η |= φ₁ ∧ X φ holds, the proofs for ∧ formulas are already done. Thus, it is true that η |= φ iff there exists α₁ ∧ Xφ₁ ∈ NF(φ₂) such that ω₀ |= α₁ and η₁ |= φ₁; The case when φ = φ₁ R φ₂ is similar to prove.

Applying the first item if n = 0 and recursively applying the second item if n ≥ 1, we can prove the third item.

**Theorem 2.** Given an ŁT₁ formula φ and a finite trace η = ω₀...ωₙ (n ≥ 0), we have that η |= φ holds iff there exists a run of T₀ on η which ends at the transition ψ₁ → ψ₂ satisfying CF(α) |= ψ₁.

**Proof.** Combine the first and third items in Lemma 2, and we can easily prove this theorem.

We say the state ψ₁ in T₀ is accepting, if there exists a transition ψ₁ → ψ₂ such that CF(α) |= ψ₁. Theorem 2 implies that, the formula φ is satisfiable if and only if there exists an accepting state ψ₁ in T₀ which is reachable from the initial state φ. Based on this observation, we now propose a simple on-the-fly satisfiability-checking framework for ŁTₐ as follows:

1. If φ equals tt, return φ is satisfiable;
2. The checking is processed on the transition system T₀ on-the-fly, i.e. computing the reachable states step by step with the DFS (Depth First Search) manner, until an accepting one is reached: Here we return satisfiable;
3. Finally we return unsatisfiable if all states in the whole transition system are explored.

The complexity of our algorithm mainly depends on the size of constructed transition system. The system construction is the same as the one for ŁTL proposed in [8]. Given an ŁT₁ formula φ, the constructed transition system T₀ has at worst the size of 2^{CF(φ)}, where CF(φ) is the set of subformulas of φ.

4 Optimizations

In this section we propose some optimization strategies by exploiting SAT solvers. First we study the relationship between the satisfiability problems for ŁTₐ and ŁTL formulas.
4.1 Relating to LTL Satisfiability

In this section we discuss some connections between \( LTL_f \) and LTL formulas. We say an \( LTL_f \) formula \( \phi \) is \( X_w \)-free iff \( \phi \) does not have the \( X_w \) operator. Note that \( LTL_f \) formulas may contain the \( X_w \) operator, while standard LTL ones do not. Here we consider \( X_w \)-free formulas, in which \( LTL_f \) and LTL have the same syntax. First, the following lemma shows how to extend a finite trace into an infinite one but still preserve the satisfaction from \( LTL_f \) to LTL:

**Lemma 3.** Let \( \eta = \omega_0 \) and \( \phi \) an \( LTL_f \) formula which is \( X_w \)-free, then \( \eta \models \phi \) implies \( \eta^n \models \phi \) when \( \phi \) is considered as an LTL formula.

**Proof.** We prove it by structural induction over \( \phi \):

- If \( \phi \) is a literal \( p \), then \( \eta \models p \) implies \( p \in \eta \). Thus \( \eta^n \models \phi \) is true;
- If \( \phi = \phi_1 \land \phi_2 \), then \( \eta \models \phi \) implies \( \eta \models \phi_1 \) and \( \eta \models \phi_2 \).
- By induction hypothesis we have \( \eta^n \models \phi_1 \) and \( \eta^n \models \phi_2 \). So \( \eta^n \models \phi \). The proof is similar when \( \phi = \phi_1 \lor \phi_2 \);
- If \( \phi = X\psi \), then according to Lemma 1 we know \( \eta \models \phi \) cannot happen; and since \( \phi \) is \( X_w \)-free, so cannot be a \( X_w \) formula;
- If \( \phi = \phi_1 U \phi_2 \), then \( \eta \models \phi \) implies \( \eta \models \phi_2 \) according to Lemma 1. By induction hypothesis we have \( \eta^n \models \phi_2 \). Thus \( \eta^n \models \phi \) is true from the LTL semantics; Similarly when \( \phi = \phi_1 R \phi_2 \), we know for every \( i \geq 0 \) it is true that \( (\xi_i = \eta^n) \models \phi_2 \). Thus \( \eta^n \models \phi \) holds from the LTL semantics; The proof is done.

We showed earlier that \( LTL_f \) satisfiability can be reduced to LTL satisfiability problem. We show that the satisfiability of some \( LTL_f \) formulas implies satisfiability of LTL formulas:

**Theorem 3.** Let \( \phi \) be an \( X_w \)-free formula. If \( \phi \) is satisfiable as an \( LTL_f \) formula, then \( \phi \) is also satisfiable as an LTL formula.

**Proof.** Assume \( \phi \) is a \( X_w \)-free \( LTL_f \) formula, and is satisfiable. Let \( \eta = \omega_0 \ldots \omega_n \) such that \( \eta \models \phi \). Now we interpret \( \phi \) as an LTL formula. Combining Lemma 2 and Lemma 3, we get that \( \xi \models \phi \) where \( \xi = \omega_0 \ldots \omega_n (\omega_i) \).

Equivalently, if \( \phi \) is an LTL formula and \( \phi \) is unsatisfiable, then the \( LTL_f \) formula \( \phi \) is also unsatisfiable. Note here the \( LTL_f \) formula \( \phi \) is \( X_w \)-free since it can be considered as an LTL formula.

**Example 1.** Consider the \( X_w \)-free formula \( \phi = GFa \land GF\neg a \), whose transition system is shown in Figure 1. If \( \phi \) is treated as an \( LTL_f \) formula, then we know that the infinite trace \( \{ (a) \{ \neg a \} \}^\omega \) satisfies \( \phi \). However, if \( \phi \) is considered to be an \( LTL_f \) formula, then we know from that no accepting state exists in the transition system, so it is unsatisfiable. It is due to the fact that no transition \( \psi_1 \xrightarrow{a} \psi_2 \) in \( T_\phi \) satisfies the condition \( CF(a) \models \psi_1 \).

- Consider another example formula \( \phi = G(aUb) \), whose transition system is shown in Figure 2. Here we can find an accepting state \( \phi \), as \( \psi \xrightarrow{a} \phi \) and \( CF(b) \models \phi \) hold. Thus we know that \( \phi \) is satisfiable, interpreted over both finite or infinite traces.

4.2 Obligation Formulas

For an LTL formula \( \phi \), Li et al. [7] have defined its obligation formula \( of (\phi) \) and show that if \( of (\phi) \) is satisfiable then \( \phi \) is satisfiable. Since \( of (\phi) \) is essentially a boolean formula, so we can check it efficiently using modern SAT solvers. However this cannot apply to \( LTL_f \) directly, which we illustrate in the following example.

**Example 2.** Consider \( \phi = GXa \), where \( \alpha \) is a satisfiable propositional formula. It is easy to see that it is satisfiable if it is an LTL formula (with respect to some word \( \alpha^n \)), while unsatisfiable when it is an \( LTL_f \) formula (because no finite trace can end with the point satisfying \( Xa \)). From [7], the obligation formula of \( \phi \) is of \( \phi = a \), which is obviously satisfiable. So the satisfiability of obligation formula implies the satisfiability of LTL formulas, but not that of \( LTL_f \) formulas.

We now show how to handle of Next operators (\( X \) and \( X_w \)) after the Release operators. For a formula \( \phi \), we define three obligation formulas:

**Definition 4 (Obligation Formulas).** Given an \( LTL_f \) formula \( \phi \), we define three kinds of obligation formulas: global obligation formula, release obligation formula, and general obligation formula—denoted as \( ofg (\phi) \), \( ofr (\phi) \) and \( off (\phi) \), by induction over \( \phi \). (We use \( ofx \) as a generic reference to \( ofg \), \( ofr \), and \( off \).)

- \( ofx (\phi) = tt \) if \( \phi = tt \); and \( ofx (\phi) = ff \) if \( \phi = ff \);
- If \( \phi = p \) is a literal, then \( ofx (\phi) = p \);
- If \( \phi = \phi_1 \land \phi_2 \), then \( ofx (\phi) = ofx (\phi_1) \land ofx (\phi_2) \);
- If \( \phi = \phi_1 \lor \phi_2 \), then \( ofx (\phi) = ofx (\phi_1) \lor ofx (\phi_2) \);
- If \( \phi = X_\phi \), then \( ofx (\phi) = off (\phi) \);\( ofr (\phi) = ff \) and \( ofg (\phi) = ff \);
- \( ofx (\phi) = X_{\phi_2} \), then \( off (\phi) = off (\phi_2) \);\( ofr (\phi) = ff \) and \( ofg (\phi) = tt \);
- \( ofx (\phi) = \phi_1 U \phi_2 \), then \( ofx (\phi) = ofx (\phi_2) \).

For example in the third item, the equation represents actually three:\( off (\phi) = ofr (\phi_1) \land ofr (\phi_2) \);\( ofr (\phi) = ofr (\phi_1) \land ofr (\phi_2) \); and \( ofg (\phi) = ofg (\phi_2) \).

For \( off (\phi) \), the changes in comparison to [7] are the definition for release formulas, and introducing the \( X_w \) operator. For example, we have that \( off (GXa) \) is \( ff \) rather than \( tt \). Moreover, since the \( LTL_f \) formula \( GX_{\phi} \) is satisfiable, the definition of \( ofg (\phi) \) is required to identify this situation. (Below we show a fast satisfiability-checking strategy that uses global obligation formulas.)

The obligation-acceleration-optimization works as follows:

**Theorem 4 (Obligation Acceleration).** For an \( LTL_f \) formula \( \phi \), if \( off (\phi) \) is satisfiable then \( \phi \) is satisfiable.

**Proof.** Since \( off (\phi) \) is satisfiable, there exists \( A \in \Sigma \) such that \( A \models off (\phi) \). We prove that there exists \( \eta = \alpha^n \) where \( n \geq 1 \) such that \( \eta \models \phi \), by structural induction over \( \phi \). Note the cases \( \phi = tt \) or \( \phi = p \) are trivial. For other cases:
If $\phi = \phi_1 \land \phi_2$, then $\mathtt{off}(\phi) = \mathtt{off}(\phi_1) \land \mathtt{off}(\phi_2)$ from Definition 4. So $\mathtt{off}(\phi)$ is satisfiable implies that there exists $A \models \mathtt{off}(\phi_1)$ and $A \models \mathtt{off}(\phi_2)$. By induction hypothesis there exists $\eta_1 = A^{n_1}$ ($n_1 \geq 0$) such that $\eta_1 \models \phi_i$ ($i = 1, 2$). Assume $n_1 \geq n_2$, then let $\eta = \eta_1$. Then, $\eta \models \phi_1 \land \phi_2$. The case when $\phi = \phi_1 \lor \phi_2$ can be proved similarly;

If $\phi = X\phi_2$ or $\phi = X_w \phi_2$, then $\mathtt{off}(\phi)$ is satisfiable iff $\mathtt{off}(\phi_2)$ is satisfiable. So there exists $A$ models $\phi_2$. By induction hypothesis, there exists $n$ such that $A^n \models \phi_2$, thus according to $\mathtt{LT}_{L}$ semantics, we know $A^{n+1} \models \phi$;

If $\phi = \phi_1 R \phi_2$, then $\mathtt{off}(\phi) = \mathtt{off}(\phi_2)$. Thus $\mathtt{off}(\phi_2)$ is also satisfiable. So there exists $A \models \mathtt{off}(\phi_2)$, based on which we can show that $A \models \phi_2$ by structural induction over $\phi_2$ by a similar proof. Thus Let $\eta = A$ and according to Lemma 1 we know $\eta \models \phi_2$ implies $\eta \models \phi$. The case for Until can be treated in a similar way, thus the proof is done.

4.3 A Complete Acceleration Technique for Global Formulas

The obligation-acceleration technique (Theorem 4) is sound but not complete, see the formula $\phi = a \land \mathtt{GF}(\neg a)$, in which $\mathtt{off}(\phi)$ is unsatisfiable, while $\phi$ is, in fact, satisfiable. In the following, we prove that both soundness and completeness hold for the global $\mathtt{LT}_{L}$ formulas, which are formulas of the form of $G \psi$, where $\psi$ is an arbitrary $\mathtt{LT}_{L}$ formula.

Theorem 5 (Obligation Acceleration for Global Formulas). For a global $\mathtt{LT}_{L}$ formula $\phi = G \psi$, we have that $\phi$ is satisfiable iff $\mathtt{off}(\phi)$ is satisfiable.

Proof. For the forward direction, assume that $\phi$ is satisfiable. It implies that there is a finite trace $\eta$ satisfying $\phi$. According to Theorem 2, $\eta$ can run on $T_0$ and reaches an accepting state $\psi_1$, i.e., $\psi_1 \overset{\omega}{\rightarrow} \psi_2$ and $\mathtt{CF}(\alpha) \models \psi_1$. Since $\phi$ is a global formula and $\psi_1$ is reachable from $\phi$, it is not hard to prove that $\mathtt{CF}(\phi) \subseteq \mathtt{CF}(\psi_1)$ from Definition 3. So $\mathtt{CF}(\phi) \models \phi$ is also true. Since $\phi$ is a global formula so $\mathtt{CF}(\psi) \models \psi$ holds from Lemma 1. Then one can prove that $\mathtt{CF}(\alpha) \models \mathtt{off}(\psi)$ by structural induction over $\psi$ (it is left to readers here), which implies that $\mathtt{off}(\psi)$ is satisfiable.

For the backward direction, assume $\mathtt{off}(\psi)$ is satisfiable. So there exists $A \subseteq \Sigma$ such that $A \models \mathtt{off}(\psi)$. Then one can prove $A \models \phi$ is also true by structural induction over $\phi = G \psi$. For paper limit, this proof is left to readers. So $\phi$ is satisfiable. The proof is done.

4.4 Acceleration for Unsatisfiable Formulas

Theorem 3 indicates that if an $\mathtt{LT}_{L}$ formula $\phi$ (of course $X_w$-free) is unsatisfiable, then the $\mathtt{LT}_{L}$ formula $\phi$ is also unsatisfiable. As a result, optimizations for unsatisfiable $\mathtt{LT}_{L}$ formulas, for instance those in [7], can be used directly to check unsatisfiable $X_w$-free $\mathtt{LT}_{L}$ formulas.

5 Experiments

In this section we present an experimental evaluation. The algorithms are implemented in the aalta tool\(^4\). We have implemented three optimization strategies. They are 1). $\mathtt{off}$: the obligation acceleration technique for $\mathtt{LT}_{L}$ (Theorem 4); 2). $\mathtt{off}$: the obligation acceleration for global $\mathtt{LT}_{L}$ formula (Theorem 5); 3). $\mathtt{off}$: the acceleration for unsatisfiable formulas (Section 4.4). Note that all three optimizations can benefit from the power of modern SAT solvers.

We compare our algorithm with the approach using off-the-shelf tools for checking $\mathtt{LT}_{L}$ satisfiability. We choose the tool Polsat, a portfolio LTL solver, which was introduced in [6]. One main feature of Polsat is that it integrates most existing $\mathtt{LT}_{L}$ satisfiability solvers (see [6]); consequently, it is currently the best-of-breed $\mathtt{LT}_{L}$ satisfiability solver. The input of aalta is directly an $\mathtt{LT}_{L}$ formula $\phi$, while that of Polsat should be $\mathtt{Tail} \land \mathtt{Tail}(\neg \mathtt{Tail}) \land t(\phi)$, which is the $\mathtt{LT}_{L}$ formula that is equi-satisfiable with the $\mathtt{LT}_{L}$ formula $\phi$.

The experimental platform of this paper is the BlueBiou cluster\(^5\) at Rice university. The cluster consists of 47 IBM Power 755 nodes, each of which contains four eight-core POWER7 processors running at 3.86GHz. In our experiments, both aalta and Polsat occupy a unique node, and Polsat runs all its integrated solvers in parallel by using independent cores of the node. The time is measured by Unix time command, and each test case has the maximal limitation of 60 seconds.

Since $\mathtt{LT}_{L}$ formulas are also $\mathtt{LT}_{L}$ formulas, we use existing $\mathtt{LT}_{L}$ benchmarks to test the tools. We compare the results from both tools, and no inconsistency occurs.

5.1 Schuppen-collected Formulas

We consider first the benchmarks introduced in previous works [11]. The benchmark suite there include earlier benchmark suites (e.g., [10]), and we refer to this suite as Schuppen-collected. The Schuppen-collected suite has a total amount of 7448 formulas. The different types of benchmarks are shown in the first column of Table 1.

<table>
<thead>
<tr>
<th>Formula type</th>
<th>aalta/sec.</th>
<th>Polsat/sec.</th>
<th>Polsat/bench</th>
</tr>
</thead>
<tbody>
<tr>
<td>raccacia/example</td>
<td>1.5</td>
<td>3.3</td>
<td>2.2</td>
</tr>
<tr>
<td>raccacia/demo-v3</td>
<td>1.4</td>
<td>3.0</td>
<td>434.9</td>
</tr>
<tr>
<td>raccacia/demo-v22</td>
<td>1.0</td>
<td>1.6</td>
<td>1.05</td>
</tr>
<tr>
<td>alaska/lift</td>
<td>23.0</td>
<td>7191.6</td>
<td>318.2</td>
</tr>
<tr>
<td>ratak/zywinka</td>
<td>1.2</td>
<td>1.0</td>
<td>81.1</td>
</tr>
<tr>
<td>rataulal smash</td>
<td>21.0</td>
<td>212026.9</td>
<td>979.7</td>
</tr>
<tr>
<td>rataulal/globol</td>
<td>5066.9</td>
<td>21371.7</td>
<td>1.5</td>
</tr>
<tr>
<td>rote/counter</td>
<td>1850.3</td>
<td>4009.4</td>
<td>1.6</td>
</tr>
<tr>
<td>rote/fortress</td>
<td>552.9</td>
<td>167.3</td>
<td>0.8</td>
</tr>
<tr>
<td>rote/founder</td>
<td>22.9</td>
<td>46.9</td>
<td>2.1</td>
</tr>
<tr>
<td>schuppen/alaska/formulas</td>
<td>5.7</td>
<td>3.1</td>
<td>2.5</td>
</tr>
<tr>
<td>schuppen/o2/formulas</td>
<td>2.0</td>
<td>1265.0</td>
<td>408.1</td>
</tr>
<tr>
<td>schuppen/pilch</td>
<td>226.3</td>
<td>808.5</td>
<td>2.8</td>
</tr>
<tr>
<td>npp/N5a</td>
<td>10.5</td>
<td>42.0</td>
<td>4.0</td>
</tr>
<tr>
<td>npp/N12x</td>
<td>2764.9</td>
<td>27777.4</td>
<td>1.0</td>
</tr>
<tr>
<td>npp/N12y</td>
<td>22.8</td>
<td>24061.3</td>
<td>1055.3</td>
</tr>
<tr>
<td>fialat</td>
<td>15244.2</td>
<td>500038.2</td>
<td>3.2</td>
</tr>
</tbody>
</table>

Table 1 shows the experimental results on Schuppen-collected benchmarks. The fourth column of the table shows the speed-up of aalta relative to Polsat. One can see that the results from aalta outperforms those from Polsat, often by several orders of magnitudes. We explain some of them.

The formulas in “Schuppen-collected/alaska/lift” are mostly unsatisfiable, which can be handled by the $\mathtt{off}$ technique of aalta. On the other side, Polsat needs more than 300 times to finish the checking. The same happens on the “Schuppen-collected/trp/N12x” patterns, in which aalta is more than 1000 times faster. For the “Schuppen-collected/schuppen/O2formula” pattern formulas, aalta scales better due to the $\mathtt{off}$ technique.

\(^4\) www.lab205.org/aalta

\(^5\) http://www.rcsg.rice.edu/sharecore/bluebiou/
5.2 Random Conjunction Formulas

Random conjunction formulas have the form of $\bigwedge_{1 \leq i \leq n} P_i$, where $P_i$ is randomly selected from typical small pattern formulas widely used in model checking [8]. By randomly choosing the that atoms the small patterns use, a large number of random conjunction formulas can be generated. More specially, to evaluate the performance on global formulas, we also fixed the selected $P_i$ by a random global pattern, and thus create a set of global formulas. In our experiments, we test 10,000 cases each for both random conjunction and global random conjunction formulas, with the number of conjunctions varying from 1 to 20 and 500 cases for each number.

Figure 3 shows the comparison results on random conjunction formulas. On average aalta earns about 10% improving performance on this kind of formulas. Among all the 10,000 cases, 8059 of them are checked by the off technique; 1105 of them are obtained by the ofg technique; 508 are acquired by the ofp technique; and another 107 are from an accepting state. There are also 109 formulas equivalent to tt or ff, which can be directly checked. In the worst case, 76 formulas are finished by exploring the whole system in the worst time, which requires further improvement. Overall, we can see Polsat is three times slower on this benchmark suite than aalta.

Figure 4 shows the comparison results on random conjunction formulas. On average LfSat and Polsat are from an accepting state. There are also 109 formulas equivalent to $\bigwedge_{1 \leq i \leq n} P_i$, where $P_i$ is randomly selected from typical small pattern formulas widely used in model checking [8]. Randomly choosing the that atoms the small patterns use, a large number of random conjunction formulas can be generated. More specially, to evaluate the performance on global formulas, we also fixed the selected $P_i$ by a random global pattern, and thus create a set of global formulas. In our experiments, we test 10,000 cases each for both random conjunction and global random conjunction formulas, with the number of conjunctions varying from 1 to 20 and 500 cases for each number.

Figure 3 shows the comparison results on random conjunction formulas. On average aalta earns about 10% improving performance on this kind of formulas. Among all the 10,000 cases, 8059 of them are checked by the off technique; 1105 of them are obtained by the ofg technique; 508 are acquired by the ofp technique; and another 107 are from an accepting state. There are also 109 formulas equivalent to tt or ff, which can be directly checked. In the worst case, 76 formulas are finished by exploring the whole system in the worst time, which requires further improvement. Overall, we can see Polsat is three times slower on this benchmark suite than aalta.

5.2 Random Conjunction Formulas

Random conjunction formulas have the form of $\bigwedge_{1 \leq i \leq n} P_i$, where $P_i$ is randomly selected from typical small pattern formulas widely used in model checking [8]. By randomly choosing the that atoms the small patterns use, a large number of random conjunction formulas can be generated. More specially, to evaluate the performance on global formulas, we also fixed the selected $P_i$ by a random global pattern, and thus create a set of global formulas. In our experiments, we test 10,000 cases each for both random conjunction and global random conjunction formulas, with the number of conjunctions varying from 1 to 20 and 500 cases for each number.

Figure 3 shows the comparison results on random conjunction formulas. On average aalta earns about 10% improving performance on this kind of formulas. Among all the 10,000 cases, 8059 of them are checked by the off technique; 1105 of them are obtained by the ofg technique; 508 are acquired by the ofp technique; and another 107 are from an accepting state. There are also 109 formulas equivalent to tt or ff, which can be directly checked. In the worst case, 76 formulas are finished by exploring the whole system in the worst time, which requires further improvement. Overall, we can see Polsat is three times slower on this benchmark suite than aalta.

Among the results from aalta, totally 5879 out of 7448 formulas in the benchmark are checked by using the off technique. This indicates the off technique is very efficient. Moreover, 84 of them are finished by exploring whole system in the worst time, which requires further improvement. Overall, we can see Polsat is three times slower on this benchmark suite than aalta.

6 Conclusion

In this paper we have proposed a novel $LTL_f$ satisfiability-checking framework based on the $LTL_f$ transition system. Meanwhile, three different optimizations are introduced to accelerate the checking process by using the power of modern SAT solvers, in which particularly the ofg optimization plays the crucial role on checking global formulas. The experimental results show that, the checking approach proposed in this paper is clearly superior to the reduction to $LTL$ satisfiability checking.

7 Acknowledgement

We thank anonymous reviewers for the useful comments. Geguang Pu is partially supported by Shanghai Knowledge Service Platform No. ZF1213. Jianwen Li is partially supported by SHEITC Project 130407 and NSFC Project No. 91118007. Jifeng He is partially supported by NSFC Project No. 61021004. Lijun Zhang is supported by NSFC project No. 61361136002. Moshe Vardi is supported in part by NSF grants CNS 1049862 and CCF-1139011, by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering", by Bsf grant 9800096, and by gift from Intel.

REFERENCES