Abstract. We consider here Linear Temporal Logic (LTL) formulas interpreted over finite traces. We denote this logic by LTL_f. The existing approach for LTL_f satisfiability checking is based on a reduction to standard LTL satisfiability checking. We describe here a novel direct approach to LTL_f satisfiability checking, where we take advantage of the difference in the semantics between LTL and LTL_f. While LTL satisfiability checking requires finding a 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 [13]. 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 in planning [1, 2, 5, 14], plan constraints [2, 9], and user preferences [3, 4, 19].

In a recent paper [10], De Giacomo and Vardi argued that while standard LTL is interpreted over infinite traces, cf. [13], AI applications are typically interested only in 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 [20].

De Giacomo and Vardi, therefore, introduced LTL_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. [21]. On the other hand, mature software tools have been developed, such as SPOT [8]. Extensive research has been conducted to evaluate these tools, cf. [16]. While the basic theory for LTL_f was presented at [10], no tool has yet to be developed for LTL_f, to the best of our knowledge.

Our main focus here is on the 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 property assurance, which seeks to eliminate errors when writing LTL properties, cf. [16] [13].

2 Preliminaries

2.1 LTL over Finite Traces

The logic LTL_f is a variant of LTL. Classical LTL formulas are interpreted on infinite traces, whereas LTL_f formulas are defined over the finite traces. Given a set P of atomic propositions, an LTL_f formula φ has the form:

\[ \phi ::= p \mid \neg \phi \mid \phi \lor \phi \mid \phi \land \phi \mid X \phi \mid X_w \phi \mid \phi U \phi \mid \phi R \phi \]

where X (strong Next), X_w (weak Next), U (Until), and R (Release) are temporal operators. We have X_w \phi \equiv \neg X \neg \phi and \phi R \phi_2 \equiv \neg (\neg \phi \lor \neg \phi_2). Note that in LTL_f, X \phi \equiv X_w \phi is not true, which is the case in LTL.

For an atom a \in 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 = P \cup \{\neg a \mid a \in 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 true as tt and false as ff. Moreover, we use the notations G\phi
(Global) and $F \phi$ (Eventually) to represent $ffR\phi$ and $ttU\phi$. We use $\phi, \psi$ to represent LTL$_f$ or LTL formulas, and $\alpha, \beta$ for propositional formulas.

Note that standard LTL$_f$ has the same syntax as LTL, see [10]. Here, however, we introduce the $X_\infty$ operator, as we consider LTL$_f$ formulas in NNF (Negation Normal Form), which requires all negations to be pushed all the way down to atoms. So a dual operator for $X$ is necessary. In LTL the dual of $X$ is $X$ itself, while in LTL$_f$ it is $X_\infty$.

**Proviso:** In the rest of paper we assume that all formulas (both LTL and LTL$_f$) are in NNF, and thus there are types of formulas, based on the primary connective: $tt$, $ff$, literal, $\land$, $\lor$, $X$ (and $X_\infty$ in LTL$_f$), $U$ and $R$.

The semantics of LTL$_f$ formulas is interpreted over finite traces, which is referred to as the LTL$_f$ interpretations [10]. Given an atom set $P$, we define $\Sigma := 2^P$. Let $\eta \in \Sigma^*$ with $\eta = \omega_0\omega_1 \ldots \omega_n$, we use $|\eta| = n + 1$ to denote the length of $\eta$. Moreover, for $1 \leq i \leq n$, we use the notation $\eta_i$ to represent $\omega_0\omega_1 \ldots \omega_{i-1}$, which is the prefix of $\eta$ before position $i$ ($i$ is not included). Similarly, we also use $\eta_j$ to represent $\omega_{j+1}\omega_{j+2} \ldots \omega_n$, which is the suffix of $\eta$ from position $i$.

Then we define $\eta$ models $\phi$, i.e. $\eta \models \phi$ in the following way:

- $\eta \models tt$ and $\eta \models ff$;
- If $\phi \models p$ is a literal, then $\eta \models \phi$ iff $p \in \eta$;
- If $\phi = X \psi$, then $\eta \models \phi$ iff $|\eta| > 1$ and $\eta_1 \models \psi$;
- If $\phi = X \psi$, then $\eta \models \phi$ iff $|\eta| > 1$ and $\eta_1 \models \psi$, or $|\eta| = 1$;
- If $\phi = \phi_1 \phi_2$ is an Until formula, then $\eta \models \phi$ iff there exists $0 \leq i < |\eta|$ such that $\eta_i \models \phi_2$ and for every $0 \leq j < i$ it holds $\eta_j \models \phi_1$ as well;
- If $\phi = \phi_1 R \phi_2$ is a Release formula, then $\eta \models \phi$ iff either for every $0 \leq i < |\eta|$ such that $\eta_i \models \phi_2$ holds, or there exists $0 \leq i < |\eta|$ such that $\eta_i \models \phi_1$ and for all $0 \leq j < i$ it holds $\eta_j \models \phi_2$ as well;
- If $\phi = \phi_1 \land \phi_2$, then $\eta \models \phi$ iff $\eta \models \phi_1$ and $\eta \models \phi_2$;
- If $\phi = \phi_1 \lor \phi_2$, then $\eta \models \phi$ iff $\eta \models \phi_1$ or $\eta \models \phi_2$.

The difference between the strong Next ($X$) and the weak Next ($X_\infty$) operators is $X$ that requires a next state in the following while $X_\infty$ may not. Thus $X_\infty \phi$ is always true in the last state of a finite trace, since no next state is provided. As a result, in LTL$_f$, $X \phi$ is unsatisfiable, while $X \phi$ is satisfiable, which is quite different with that in LTL, where neither $Xf$ nor $X \neg \phi$ are satisfiable.

Let $\phi$ be an LTL$_f$ formula, we use $CF(\phi)$ to represent the set of conjuncts in $\phi$, i.e. $CF(\phi) = \{ \phi_i | \phi_i \in I \}$ if $\phi = \bigwedge_{i \in I} \phi_i$, where the root of $\phi_i$ is not a conjunction. $DF(\phi)$ (the set of disjuncts) is defined analogously.

### 2.2 The LTL$_f$ Satisfiability Problem

The satisfiability problem is to check whether, for a given LTL$_f$ formula $\phi$, there is a finite trace $\eta \in \Sigma^*$ such that $\eta \models \phi$.

**Definition 1 (LTL$_f$ Satisfiability Problem).** Given an LTL$_f$ formula $\phi$ over the alphabet $\Sigma$, we say $\phi$ is satisfiable if there is a finite trace $\eta \in \Sigma^*$ such that $\eta \models \phi$.

One approach is to reduce the LTL$_f$ satisfiability problem to that of LTL.

**Theorem 1 ([10]).** The Satisfiability problem for LTL$_f$ formulas is PSPACE-complete.

**Proof Sketch:** It is easy to reduce the LTL$_f$ 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 ($Tail \uparrow (G-Tail)$);
4. The LTL$_f$ formula $\phi$ is translated into a corresponding LTL formula in the following way:
   - $t(p) \rightarrow p$, where $p$ is a literal;
   - $\neg t(\phi)$;
   - $t(\phi_1 \land \phi_2) \rightarrow t(\phi_1) \land t(\phi_2)$;
   - $t(\phi_1 \lor \phi_2) \rightarrow t(\phi_1) \lor t(\phi_2)$;
   - $t(X_\psi) \rightarrow X(Tail \land t(\psi))$;
   - $t(\phi_1 \phi_2) \rightarrow t(\phi_1 \phi_2)$;

(The translation here does not require $\phi$ in NNF. Thus the $X_\infty$ and $R$ operators can be handled by the rules $X_\infty \phi \equiv \neg X \neg \phi$ and $R \phi_2 \equiv \neg (\neg \phi \neg \phi_2)$.) Finally one can prove that $\phi$ is satisfiable iff $Tail \uparrow Tail(G-Tail) \land t(\phi)$ is satisfiable.

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$_f$ checking.

### 2.3 LTL$_f$ Transition System

In [13], Li et al. have proposed using transition systems for checking satisfiability of LTL$_f$ 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:

- $NF(\phi) = \{ \phi \land X(tt) \}$ if $\phi \not\equiv ff$ is a propositional formula. If $\phi \equiv ff$, we define $NF(\phi) = \emptyset$;
- $NF(X\phi) = \{ tt \land X(\psi) | \psi \in DF(\phi) \}$;
- $NF(\phi \lor \phi_2) = NF(\phi_2) \cup NF(\phi_1 \land X(\phi_1 \phi_2))$;
- $NF(\phi_1 R \phi_2) = NF(\phi_1 \land \phi_2) \cup NF(\phi_1 \land X(\phi_1 R \phi_2))$;
- $NF(\phi_1 \lor \phi_2) = NF(\phi_1) \cup NF(\phi_2)$;
- $NF(\phi_1 \land \phi_2) = \{(\chi_1 \land \phi_2) \land X(\chi_1 \land \phi_2) | \chi_1 = 1 \land \alpha_1 \land X(\chi_1) \in NF(\phi_1) \}$.

For each $\alpha_1 \land X(\phi_2) \in NF(\phi)$, we say it a clause of $NF(\phi)$.

(Although the normal forms of $X$ and $X_\infty$ formulas are the same, we did distinguish between them through the accepting conditions introduced below.) Intuitively, each clause $\alpha_1 \land X(\phi_i)$ of $NF(\phi)$ indicates that the propositional formula $\alpha_1$ 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 $L_\infty$; 3. the transition relation $\rightarrow \subseteq S_\phi \times Act \times S_\phi$ is defined by: $\psi_1 \rightarrow \psi_2$ iff there exists $\alpha \land X(\psi_2) \in NF(\psi_1)$ 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 LTL$_f$. For LTL$_f$, X_\infty ff. Nevertheless, ff edges are not allowed both in LTL$_f$ and LTL transition systems.
A run of $T_0$ on finite trace $\eta = \omega_0|\omega_1| \ldots |\omega_n \in \Sigma^*$ is a sequence $s_0 \xrightarrow{\omega_0} s_1 \xrightarrow{\omega_1} \ldots \xrightarrow{\omega_n} s_n \xrightarrow{\omega_{n+1}}$ such that $s_0 = \phi$ and for every $0 \leq i \leq n$ it holds $\omega_i = |\alpha_i$. We say $\psi$ is reachable from $\phi$ iff there is a run of $T_0$ such that the final state is $\psi$.

3 \textit{LTL}_f Satisfiability-Checking Framework

In this section we present our framework for checking satisfiability of \textit{LTL}_f formulas. First we show a simple lemma concerning finite sequences of length 1.

\textbf{Lemma 1.} For a finite trace $\eta \in \Sigma^*$ and \textit{LTL}_f formula $\phi$, if $|\eta| = 1$ then $\eta \models \phi$ holds if:

- $\eta \models tt$ and $\eta \not\models \mathbf{ff}$;
- If $\phi = p$ is a literal, then return true if $\phi \in \eta$, otherwise return false;
- If $\phi = \phi_1 \wedge \phi_2$, then return $\eta \models \phi_1$ and $\eta \models \phi_2$;
- If $\phi = \phi_1 \vee \phi_2$, then return $\eta \models \phi_1$ or $\eta \models \phi_2$;
- If $\phi = X\phi_0$, then return true;
- If $\phi = \phi_0 U\phi_2$ or $\phi = \phi_0 R\phi_2$, then return $\eta \models \phi_2$.

Proof. This lemma can be directly proven from the semantics of \textit{LTL}_f formulas by fixing $|\eta| = 1$.

Now we characterize the satisfaction relation for finite sequences:

\textbf{Lemma 2.} For a finite trace $\eta = \omega_0|\omega_1| \ldots |\omega_n \in \Sigma^*$ and \textit{LTL}_f formula $\phi$,

1. $|\mathit{n} = 0$, then $\eta \models \phi$ iff there exists $\alpha_i \wedge X\phi_i \in NF(\phi)$ such that $\omega_0 \models \alpha_i$ and $CF(\alpha_i) \models \phi$;
2. $|\mathit{n} \geq 1$, then $\eta \models \phi$ iff there exists $\alpha_i \wedge X\phi_i \in NF(\phi)$ such that $\omega_0 \models \alpha_i$ and $\eta_1 \models \phi_1$;
3. $\eta \models \phi$ iff there exists a run $\phi = \phi_0 \xrightarrow{\omega_0} \phi_1 \xrightarrow{\omega_1} \ldots \xrightarrow{\omega_n} \phi_{n+1}$ in $T_0$ such that for every $0 \leq i \leq n$ it holds that $\omega_i \models \alpha_i$ and $\eta_i \models \phi_i$.

Proof. $CF(\alpha_i)$ is treated to be a finite trace whose length is 1. We prove the first item by structural induction over $\phi$.

- If $\phi = p$, then $\eta \models \phi$ iff $\omega_0 \models p$ and $CF(p) \models \phi$ hold, where $p \wedge \mathit{tt}$ is actually in NF$(\phi)$;
- If $\phi = \phi_1 \wedge \phi_2$, then $\eta \models \phi$ holds iff $\eta \models \phi_1$ and $\eta \models \phi_2$ hold, and if by induction hypothesis, there exists $\beta_1 \wedge X\psi_i$ in NF$(\phi_i)$ such that $\omega_0 \models \beta_1$ and $CF(\beta_1) \models \phi$ ($i = 1, 2$). Let $\alpha_i = \beta_1 \wedge \beta_2$ and $\phi_i = \psi_1 \wedge \psi_2$, then according to Definition 2 we know $\alpha_i \wedge X\phi_i$ is in NF$(\phi)$, and $\omega_0 \models \alpha_i$ and $CF(\alpha_i) \models \phi$ hold; the proof for the case when $\phi = \phi_1 \vee \phi_2$ is similar;
- Note that $\eta \models X\psi$ is always false, and if $\phi = X\psi$ then from Lemma 1 it is always true that $\eta \models X\psi$ iff $\mathit{tt} \wedge X\psi \in \phi$ in $T_0$ and $\mathit{tt} \models X\psi$;
- If $\phi = \phi_0 U\phi_2$, then $\eta \models \phi$ holds iff $\eta \models \phi_2$ holds from Lemma 1 and if by induction hypothesis, there exists $\alpha_i \wedge X\phi_i \in NF(\phi_2)$ such that $\omega_0 \models \alpha_i$ and $CF(\alpha_i) \models \phi_2$, and thus $CF(\alpha_i) \models \phi$ according to \textit{LTL}_f semantics. From Definition 2 we know as well that $\alpha_i \wedge X\phi_i$ is in NF$(\phi)$, thus the proof is done; the proof for the case when $\phi = \phi_0 R\phi_2$ is similar;

2. The second item is also proven by structural induction over $\phi$.

- If $\phi = \mathbf{tt}$ or $\phi = p$, then $\eta \models \phi$ iff $\omega_0 \models \phi$ and $\eta_1 \models \mathit{tt}$ hold, where $\phi \wedge \mathit{tt}$ is actually in NF$(\phi)$;
- If $\phi = X\phi_2$ or $\phi = X\omega_{i+1}$, then $\eta \models \phi$ iff $\eta \models \phi_2$ and $\eta \models \phi$ hold according to \textit{LTL}_f semantics, and obviously $\mathit{tt} \wedge X\phi_2$ is in NF$(\phi)$;
- If $\phi = \phi_1 \wedge \phi_2$, then $\eta \models \phi$ iff $\eta \models \phi_1$ and $\eta \models \phi_2$, and iff by induction hypothesis, there exists $\beta_1 \wedge X\psi_i \in NF(\phi_i)$ ($i = 1, 2$) such that $\omega_0 \models \beta_1$ and $\eta_1 \models \psi_i$ hold, and iff $\omega_0 \models \beta_1 \wedge \beta_2$ and $\eta_1 \models \psi_1 \wedge \psi_2$ hold, in which $(\beta_1 \wedge \beta_2) \wedge X(\psi_1 \wedge \psi_2)$ is indeed in NF$(\phi)$; the case when $\phi = \phi_1 \vee \phi_2$ is similar;
- If $\phi = \phi_0 U\phi_2$, then $\eta \models \phi$ iff $\eta \models \phi_2$ or $\eta \models (\phi_0 \wedge X\phi)$.

We say the state $\psi_1$ in $T_0$ is \textit{accepting}, if there exists a transition $\psi_1 \xrightarrow{\mathit{tt}} \psi_2$ such that $CF(\alpha_i) \models \psi_1$. Theorem 2 implies that, the formula $\phi$ is satisfiable if and only if there exists an accepting state $\psi_1$ in $T_0$ which is reachable from the initial state $\phi$. Based on this observation, we now propose a simple on-the-fly satisfiability-checking framework for \textit{LTL}_f as follows:

1. If $\phi$ equals $\mathbf{tt}$, return $\phi$ is satisfiable;
2. The checking is processed on the transition system $T_0$ 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 \textit{LTL} proposed in [13]. Given an \textit{LTL}_f formula $\phi$, the constructed transition system $T_0$ has at worst the size of $2^{cl(\phi)}$, where $cl(\phi)$ is the set of subformulas of $\phi$.

4 \textit{Optimizations}

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

In this section we discuss some connections between LTLf and LTL formulas. We say an LTLf formula φ is Xw-free iff φ does not have the Xw operator. Note that LTLf formulas may contain the Xw operator, while standard LTL ones do not. Here consider Xw-free formulas, in which LTLf 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 LTLf to LTL:

**Lemma 3.** Let η = ω₀ and φ an LTLf formula which is Xw-free, then η |= φ implies η² |= φ when φ is considered as an LTL formula.

**Proof.** We prove it by structural induction over φ:

- If φ is a literal p, then η |= p φ ∈ η. Thus η² |= φ is true;
- If φ is τt, then η² |= τt is obviously true;
- If φ = φ₁ ∧ φ₂, then η |= φ implies η |= φ₁ and η |= φ₂. By induction hypothesis we have η² |= φ₁ and η² |= φ₂. So η² |= φ₁ ∧ φ₂. The proof is similar when φ = φ₁ ∨ φ₂;
- If φ = Xψ, then according to Lemma 1 we know η |= φ cannot happen; And since φ is Xw-free, so φ cannot be an Xw formula;
- If φ = φ₁ Uφ₂, then η |= φ implies η |= φ₂ according to Lemma 4. By induction hypothesis we have η² |= φ₂. Thus η² |= φ holds from the LTL semantics; The proof is done.

We showed earlier that LTLf satisfiability can be reduced to LTL satisfiability problem. We show that the satisfiability of some LTLf formulas implies satisfiability of LTL formulas:

**Theorem 3.** Let φ be an Xw-free formula. If φ is satisfiable as an LTLf formula, then φ is also satisfiable as an LTL formula.

**Proof.** Assume φ is a Xw-free LTLf formula, and is satisfiable. Let η = ω₀...ωₙ₊₁ such that η |= φ. Now we interpret φ as an LTL formula. Combining Lemma 2 and Lemma 5, we get that ζ |= φ where ζ = ω₀...ωₙ⁻¹ (ωₙ)².

Equivalently, if φ is an LTL formula and φ is unsatisfiable, then the LTLf formula φ is also unsatisfiable. Note here the LTLf formula φ is Xw-free since it can be considered as an LTL formula.

**Example 1.** Consider the Xw-free formula φ = GFa ∧ GF¬a, whose transition system is shown in Figure 1. If φ is treated as an LTL formula, then we know that the infinite trace {a} {¬a}² satisfies φ. However, if φ is considered to be an LTLf 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 ψ₁  → ψ₂ in T φ satisfies the condition CF(α) |= ψ₁.

**Example 2.** Consider φ = GXa, where α is a satisfiable propositional formula. It is easy to see that if α is satisfiable if it is an LTL formula (with respect to some word aⁿ), while unsatisfiable when it is an LTL formula (because no finite trace can end with the point satisfying Xa). From [12], the obligation formula of φ is of φ = a, which is obviously satisfiable. So the satisfiability of obligation formula implies the satisfiability of LTL formulas, but not that of LTLf formulas.

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

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

- ofx = tt φ = tt; and ofx(φ) = ff if φ = ff;
- If φ = p is a literal, then ofx(φ) = p;
- If φ = φ₁ ∧ φ₂, then ofx(φ) = ofx(φ₁) ∧ ofx(φ₂);
- If φ = φ₁ ∨ φ₂, then ofx(φ) = ofx(φ₁) ∨ ofx(φ₂);
- If φ = Xφ₂, then off(φ) = off(φ₂), ofr(φ) = ff, and ofg(φ) = ff;
- If φ = Xwφ₂, then off(φ) = off(φ₂), ofr(φ) = ff, and ofg(φ) = tt; If φ = φ₁ Uφ₂, then off(φ) = ofx(φ₂).
- If φ = φ₁ Rφ₂, then off(φ) = ofr(φ₂), ofr(φ) = ofx(φ₂), and ofg(φ) = ofg(φ₂)

For example in the third item, the equation represents actually three: off(φ) = ofg(φ₁) ∧ ofg(φ₂), ofr(φ) = ofr(φ₁) ∧ ofr(φ₂) and ofg(φ) = ofg(φ₁) ∧ ofg(φ₂).

For off(φ), the changes in comparison to [12] are the definition for release formulas, and introducing the Xw operator. For example, we have that off(Xa) is ff rather than a. Moreover, since the LTLf formula GXa is satisfiable, the definition of ofg(φ) 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 LTLf formula φ, if off(φ) is satisfiable then φ is satisfiable.

**Proof.** Since off(φ) is satisfiable, there exists A ∈ Σ such that A |= off(φ). We prove that there exists η = Aⁿ such that η |= φ, by structural induction over φ. Note the cases φ = tt or φ = p are trivial. For other cases:

- If φ = φ₁ ∧ φ₂, then off(φ) = off(φ₁) ∧ off(φ₂) from Definition 4. So off(φ) is satisfiable implies that there exists A |= off(φ₁) and A |= off(φ₂). By induction hypothesis there exists η₁ = Aⁿ⁻¹ (n₁ ≥ 0) such that η₁ |= φ₁ (i = 1, 2). Assume n₁ ≥ n₂, then let η = η₁. Then, η |= φ₁ ∧ φ₂.
4.3 A Complete Acceleration Technique for Global Formulas

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

**Theorem 5** (Obligation Acceleration for Global Formulas). For a global LTL formula \( \phi = G\psi \), we have that \( \phi \) is satisfiable iff \( \alpha \phi \psi \) 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 4, \( \eta \) can run on \( T_\phi \) and reaches an accepting state \( \psi_2 \), i.e., \( \psi_1 \xrightarrow{a} \psi_2 \) and \( CF(\alpha) \models \psi_1 \). Since \( \phi \) is a global formula and \( \psi_1 \) is reachable from \( \phi \), it is not hard to prove that \( CF(\phi) \subseteq CF(\psi_1) \) from Definition 3. So \( CF(\alpha) \models \phi \) is also true. Since \( \phi \) is a global formula so \( CF(\alpha) \models \psi \) holds from Lemma 3. Then one can prove that \( CF(\alpha) \models \alpha \phi \psi \) by structural induction over \( \psi \) (it is left to readers here), which implies that \( \phi \) is satisfiable.

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

4.4 Acceleration for Unsatisfiable Formulas

Theorem 2 indicates that if an LTL formula \( \phi \) (of course \( X_\omega \)-free) is unsatisfiable, then the LTL formula \( \phi \) is also unsatisfiable. As a result, optimizations for unsatisfiable LTL formulas, for instance those in [12], can be used directly to check unsatisfiable \( X_\omega \)-free LTL formulas.

5 Experiments

In this section we present an experimental evaluation. The algorithms are implemented in the LfSat tool 4. We have implemented three optimization strategies. They are 1) \( \alpha \phi \): the obligation acceleration technique for LTL formulas (Theorem 2), 2) \( \alpha \phi g \psi \): the obligation acceleration for global LTL formula (Theorem 5), 3) \( \alpha \phi f \psi \): 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 LTL satisfiability. We choose the tool Polsat, a portfolio LTL solver, which was introduced in [11]. One main feature of Polsat is that it integrates most existing LTL satisfiability solvers; consequently, it is currently the best-of-breed LTL satisfiability solver. The input of LfSat is directly an LTL formula \( \phi \), while that of Polsat should be \( Tail \land \neg Tail \land \phi \), which is the LTL formula that is equi-satisfiable with the LTL formula \( \phi \).

The experimental platform of this paper is a cluster that consists of 47 IBM Power 755 nodes, each of which contains four eight-core POWER7 processors running at 3.86GHz. In our experiments, both LfSat 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 LTL formulas are also LfSat formulas, we use existing LTL benchmarks to test the tools. We compare the results from both tools, and no inconsistency occurs.

5.1 Schuppan-collected Formulas

We consider first the benchmarks introduced in previous works [17]. The benchmark suite there include earlier benchmark suites (e.g., [16]), and we refer to this suite as Schuppan-collected. The Schuppan-collected suite has a total amount of 7448 formulas. The different types of benchmarks are shown in the first column of Table 1.

Table 1: Experimental results on Schuppan-collected formulas.

| Formula Type | LfSat(sec.) | Polsat(sec.) | Polsat/LfSat |
|--------------|-------------|--------------|--------------|
| /acacia/simple | 1.5 | 3.3 | 2.2 |
| /acacia/demo-v1 | 1.4 | 504.4 | 351.9 |
| /acacia/demo-v22 | 2.0 | 1.4 | 0.66 |
| /alaska/lift | 23.1 | 7317.6 | 318.2 |
| /alkaska/ymashi | 1.2 | 7.3 | 6.1 |
| /anza/ramle | 3120.9 | 2052.9 | 1.5 |
| /anza/geibat | 3066.9 | 3171.9 | 1.0 |
| /alloca/counters | 1840.3 | 3009.3 | 1.6 |
| /alloca/formulas | 522.9 | 467.0 | 0.9 |
| /alloca/pattern | 22.9 | 49.9 | 2.1 |
| /schuppan/O1formula | 2.9 | 7.1 | 2.4 |
| /schuppan/O2formula | 3.1 | 1253.9 | 0.021 |
| /schuppan/pHll | 226.3 | 602.5 | 2.6 |
| /tmp/N5 | 10.5 | 42.0 | 4.0 |
| /tmp/N5 | 29.0 | 2717.4 | 94.3 |
| /tmp/N12 | 22.8 | 2406.1 | 105.5 |
| /tmp/N12 | 400.2 | 4049.2 | 1.0 |
| Total | 15224.2 | 50358.2 | 3.3 |

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

The formulas in “Schuppan-collected/alaska/lift” are mostly unsatisfiable, which can be handled by the \( \alpha \phi g \psi \) technique of LfSat. On the other side, Polsat needs more than 300 times to finish the checking. The same happens on the “Schuppan-collected/trp/N12a” patterns, in which LfSat is more than 1000 times faster. For the “Schuppan-collected/schuppan/O2formula” pattern formulas, LfSat scales better due to the \( \alpha \phi f \) technique.

Among the results from LfSat, totally 5879 out of 7448 formulas in the benchmark are checked by using the \( \alpha \phi f \) technique. This indicates the \( \alpha \phi f \) 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 LfSat.

---

4 Tool will be released upon paper publication.
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 [13]. 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 LfSat earns about 10% improving performance on this kind of formulas. Among all the 10,000 cases, 8059 of them are checked by the ofg technique; 1105 of them are obtained by the ofg technique; 508 are acquired by the ofg 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 transition system. About 36 formulas fail to be checked within 60 seconds by LfSat. Statistics above show the optimizations are very useful.

Moreover, one can conclude from Figure 3 that, LfSat dominates Polsat when performing on the global random conjunction formulas. As the ofg technique is both sound and complete for global formulas and invokes SAT solvers only once, so LfSat performs almost constant time for checking both satisfiable and unsatisfiable formulas. Compared with that, Polsat takes an ordinary checking performance for this kind of special formulas. Indeed, the ofg technique is considered to play the crucial role on checking global $LTL_f$ formulas.

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.

REFERENCES

[1] F. Bacchus and F. Kabanza, ‘Planning for temporally extended goals’, Ann. of Mathematics and Artificial Intelligence, 22, 5–27, (1998).
[2] F. Bacchus and F. Kabanza, ‘Using temporal logic to express search control knowledge for planning’, Artificial Intelligence, 116(1–2), 123–191, (2000).
[3] M. Bienveneu, C. Fritz, and S. McIlraith, ‘Planning with qualitative temporal preferences’, in International Conference on Principles of Knowledge Representation and Reasoning (KR06), pp. 134–144, Lake District, UK, (June 2006).
[4] M. Bienveneu, C. Fritz, and S. A. McIlraith, ‘Specifying and computing preferred plans’, Artificial Intelligence, 175(7C8), 1308 – 1345, (2011).
[5] D. Calvanese, G. De Giacomo, and M.Y. Vardi, ‘Reasoning about actions and planning in LTL action theories’, in Proc. 8th Int’l Conf. on the Principles of Knowledge Representation and Reasoning, pp. 593–602, Morgan Kaufmann, (2002).
[6] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis, ‘Memory efficient algorithms for the verification of temporal properties’, Formal Methods in System Design, 1, 275–288, (1992).
[7] G. De Giacomo and M.Y. Vardi, ‘Automata-theoretic approach to planning for temporally extended goals’, in Proc. European Conf. on Planning, Lecture Notes in AI 1809, pp. 226–238, Springer, (1999).
[8] A. Duret-Lutz and D. Poitrenaud, ‘SPOT: An extensible model checking library using transition-based generalized buchi automata’, in Proc. 12th Int’l Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 76–83, IEEE Computer Society, (2004).
[9] A. Gabaldon, ‘Precondition control and the progression algorithm’, in Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR), pp. 634–643, AAAI Press, (2004).
[10] G. De Giacomo and M. Vardi, ‘Linear temporal logic and linear dynamic logic on finite traces’, in Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI’13, pp. 2000–2007, AAAI Press, (2013).
[11] J. Li, G. Pu, L. Zhang, M. Y. Vardi, and J. He, ‘Polsat: A portfolio LTL satisfiability solver’, CoRR, abs/1311.1602, (2013).
[12] J. Li, G. Pu, L. Zhang, M. Y. Vardi, and J. He, ‘Fast LTL satisfiability checking by sat solvers’, CoRR, abs/1401.5677, (2014).
[13] J. Li, L. Zhang, G. Pu, M. Vardi, and J. He, ‘LTL satisfiability checking revisited’, in The 20th International Symposium on Temporal Representation and Reasoning, pp. 91–98, (2013).
[14] F. Patrizi, N. Lipovezsky, G. De Giacomo, and H. Geffner, ‘Computing infinite plans for LTL goals using a classical planner’, in Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI’11, pp. 2003–2008, AAAI Press, (2011).
[15] A. Pnueli, ‘The temporal logic of programs’, in Proc. 18th IEEE Symp. on Foundations of Computer Science, pp. 46–57, (1977).
[16] K.Y. Rozier and M.Y. Vardi, ‘LTL satisfiability checking’, Int’l J. on Software Tools for Technology Transfer, 12(2), 1230–137, (2010).
[17] V. Schuppian and L. Darmawan, ‘Evaluating LTL satisfiability solvers’, in Proceedings of the 9th international conference on Automated technology for verification and analysis, AVTA’11, pp. 397–413, Springer-Verlag, (2011).
[18] A.P. Sistla and E.M. Clarke, ‘The complexity of propositional linear temporal logic’, Journal of the ACM, 32, 733–749, (1985).
[19] S. Sohrabi, J. A. Baier, and S. A. McIlraith, ‘Preferred explanations: Theory and generation via planning’, in Proceedings of the 25th Conference on Artificial Intelligence (AAAI-11), pp. 261–267, San Francisco, USA, (August 2011). Accepted as both oral and poster presentation.
[20] W. M. P. van der Aalst, M. Pesci, and H. Schonenberg, ‘Declarative workflows: Balancing between flexibility and support’, Computer Science - R&D, 99–113, (2009).
[21] M.Y. Vardi, ‘An automata-theoretic approach to linear temporal logic’, in Logics for Concurrency: Structure versus Automata, eds., F. Moller and G. Birtwistle, volume 1043 of Lecture Notes in Computer Science, pp. 258–266, Springer, (1996).