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 [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 in planning [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 infinite traces, cf. [9], AI applications are typically limited only to 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 $LTL_f$, 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 $LTL_f$ was presented at [5], no tool has yet to be developed for $LTL_f$, to the best of our knowledge. Our goal in this paper is to address this gap.

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. [10, 8].

De Giacomo and Vardi studied the computational complexity of $LTL_f$ satisfiability and showed that it is PSPACE-complete, which is the same complexity as for LTL satisfiability [12]. Their proof of the upper bound uses a reduction of $LTL_f$ satisfiability to LTL satisfiability. That is, for an $LTL_f$ formula $\phi$, one can create an LTL formula $\phi'$ such that $\phi$ is satisfiable iff $\phi'$ 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 $LTL_f$ satisfiability. A reduction to LTL satisfiability, therefore, may add unnecessary overhead to $LTL_f$ satisfiability checking.

The paper is organized as follows. We first introduce the definition of $LTL_f$, 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 $LTL_f$ 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 $\phi$ has the form:

$\phi := tt | ff | p | ¬\phi | \phi \lor \psi | \phi \land \psi | X\phi | X\omega\phi | U\phi | R\phi$

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

For an atom $a \in P$, we call it or its negation ($¬a$) a literal. We use the set $L$ to denote the set of literals, i.e., $L = P \cup \{¬a | a \in P\}$. Other boolean operators, such as $→$ and $↔$, can be represented by the combination ($\lor$) or ($\lor$, $∧$, 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 \(\mathit{LTL}_f\) or \(\mathit{LTL}\) formulas, and \(\alpha, \beta\) for propositional formulas.

Note that standard \(\mathit{LTL}_f\) has the same syntax as \(\mathit{LTL}\), see [5]. Here, however, we introduce the \(X_w\) operator, as we consider \(\mathit{LTL}_f\) 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 \(\mathit{LTL}\) the dual of \(X\) is \(\bar{X}\) itself, while in \(\mathit{LTL}_f\) it is \(\bar{X}_{w}\).

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

The semantics of \(\mathit{LTL}_f\) formulas is interpreted over finite traces, which is referred to as the \(\mathit{LTL}_f\) interpretations [5]. 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 \(\eta_i\) to represent \(\omega_i\omega_{i+1} \ldots \omega_n\), which is the suffix of \(\eta\) from position \(i\).

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

- \(\eta \models tt\) and \(\eta \not\models ff\);  
- If \(\phi = p\) is a literal, then \(\eta \models \phi\) iff \(p \in \eta^1\);  
- If \(\phi = X\psi\), then \(\eta \models \phi\) iff \(|\eta^i| > 1\) and \(\eta_{i+1} \models \psi\);  
- If \(\phi = X_w\psi\), then \(\eta \models \phi\) iff \(|\eta| > 1\) and \(\eta_1 \models \psi\); or \(|\eta| = 1\);  
- If \(\phi = \phi_1 U \phi_2\) 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 \(0 \leq i < |\eta|\) \(\eta_i \models \phi_2\) holds, or there exists \(0 \leq i < |\eta|\) such that \(\eta_i \models \phi_1\) and for all \(0 \leq j \leq i\) 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 (\(X_w\)) operators is that \(X\) requires a next state in the following while \(X_w\) may not. Thus \(X_w\phi\) is always true in the last state of a finite trace, since no next state is provided. As a result, in \(\mathit{LTL}\) \(X\psi\) is unsatisfiable, while in \(\mathit{LTL}_f\) operators can be handled by the rules \(X_w\phi \equiv \neg \neg X \neg \phi\) and \(\phi_1 R \phi_2 \equiv \neg \neg (\phi_1 \neg \neg \phi_2)\). Finally one can refer to [5] that \(\phi\) is satisfiable iff \(\text{Tail} \supset \text{TailU}(\mathit{G} \rightarrow \text{Tail})\) and \(\text{Tail}\) is satisfiable. Also, a \(\text{PSPACE}\) lower bound is shown in [5] by reduction from STRIPS Planning.

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

### 2.3 \(\mathit{LTL}_f\) Transition System

In [8], Li et al. have proposed using transition systems for checking satisfiability of \(\mathit{LTL}_f\) formulas. Here we adapt this approach to \(\mathit{LTL}_f\).

**Definition 2 (Normal Form).** The normal form of an \(\mathit{LTL}_f\) 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 literal. If \(\phi \equiv ff\), we define \(NF(ff) = \emptyset\);  
- \(NF(X\phi/X_w\phi) = \{tt \land X(\psi) \mid \psi \in DF(\phi)\};  
- \(NF(\phi_1 U \phi_2) = NF(\phi_2) \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) \cup NF(\phi_2);  
- \(NF(\phi_1 \land \phi_2) = \{\alpha_1 \land \alpha_2 \land X(\psi_1 \land \psi_2) \mid \forall i = 1, 2, \alpha_i \land X(\psi_i) \in NF(\phi_i)\}.

For each \(\alpha_i \land X(\psi_i) \in NF(\phi)\), we say it a clause of \(NF(\phi)\).

(Although the normal forms of \(X\) and \(X_w\) formulas are the same, we do distinguished between them through the accepting conditions introduced below.) Intuitively, each clause \(\alpha_i \land X(\psi_i) \in NF(\phi)\) indicates that the propositional formula \(\alpha_i\) should hold now and then \(\psi_i\) 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 (\(\mathit{LTL}_f\) 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_2) \in NF(\psi_1);  
(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 \(\mathit{LTL}_f\) transition systems the \(ff\) state can be deleted, as it can never be part of a fair cycle. This state must be kept in \(\mathit{LTL}_f\) transition systems: a finite trace that reach \(ff\) may be accepted in
LTL, cf. X. D. ff. Nevertheless, ff edges are not allowed both in LTL 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 $s_0 \xrightarrow{\omega_0} s_1 \xrightarrow{\omega_1} \ldots s_n \xrightarrow{\omega_n} s_{n+1}$ 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. LTL Satisfiability-Checking Framework

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

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

- $\eta = \tt$ and $\eta \not\models \ff$;
- $\phi$ is a literal, then return true if $\phi \in \eta$, otherwise return false;
- If $\phi = \phi_1 \land \phi_2$, then return $\eta \models \phi_1$ and $\eta \models \phi_2$;
- If $\phi = \phi_1 \lor \phi_2$, then return $\eta \models \phi_1$ or $\eta \models \phi_2$;
- $\phi = X\phi_2$, then return false;
- $\phi = X_\omega \phi_2$, then return true;
- If $\phi = \phi_1 U \phi_2$ or $\phi = \phi_1 R \phi_2$, then return $\eta \models \phi_2$.

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

Now we characterize the satisfaction relation for finite sequences:

**Lemma 2.** For a finite trace $\eta = \omega_0\omega_1 \ldots \omega_n \in \Sigma^*$ and LTL formula $\phi$.

1. If $n = 0$, then $\eta \models \phi$ iff there exists $\alpha_0 \land X\phi_1 \in NF(\phi)$ such that $\omega_0 \models \alpha_0$ and $CF(\alpha_0) \models \phi$;
2. If $n \geq 1$, then $\eta \models \phi$ iff there exists $\alpha_0 \land X\phi_1 \in NF(\phi)$ such that $\omega_0 \models \alpha_0$ and $\eta_1 \models \phi_1$;
3. $\eta \models \phi$ iff there exists a run $\phi = \phi_0 \xrightarrow{\omega_0} \phi_1 \xrightarrow{\omega_1} \phi_2 \ldots \xrightarrow{\omega_n} \phi_{n+1}$ 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. 1. $CF(\alpha_0)$ 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 $p \land X\tt$ is actually in NF($\phi$);
- If $\phi = \phi_1 \land \phi_2$, then $\eta \models \phi$ holds iff $\eta \models \phi_1$ and $\eta \models \phi_2$ hold, and by induction hypothesis, there exists $\beta_i \land X\psi_i \in NF(\phi_i)$ such that $\omega_0 \models \beta_i$ and $CF(\beta_i) \models \phi_i$ ($i = 1, 2$). Let $\alpha_0 = \beta_1 \land \beta_2$ and $\eta_1 = \psi_1 \land \psi_2$, then according to Definition 2 we know $\alpha_0 \land X\phi_1 \in NF(\phi)$, and $\omega_0 \models \alpha_0$ and $CF(\alpha_0) \models \phi$ hold; The proof for the case when $\phi = \phi_1 \lor \phi_2$ is similar;
- Note that $\eta \models X\psi$ is always false, and if $\phi = X_\omega \psi$ then from Lemma 1 it is always true that $\eta \models X_\omega \psi$ iff $\tt \land X\psi \in NF(\phi)$ and $\tt \models X_\omega \psi$;
- If $\phi = \phi_1 U \phi_2$, then $\eta \models \phi$ holds iff $\eta \models \phi_2$ holds from Lemma 1, and by induction hypothesis, there exists $\alpha_0 \land X\phi_1 \in NF(\phi_2)$ such that $\omega_0 \models \alpha_0$ and $CF(\alpha_0) \models \phi_2$, and thus $CF(\alpha_0) \models \phi$ according to LTL semantics. From Definition 2 we know as well that $\alpha_0 \land X\phi_1 \in NF(\phi)$, thus the proof is done; The proof for the case when $\phi = \phi_1 R \phi_2$ is similar;

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

- If $\phi = \tt$ or $\phi = p$, then $\eta \models \phi$ iff $\omega_0 \models \phi$ and $\eta_1 \models \tt$ hold, where $\phi \land X\tt$ is actually in NF($\phi$);
- If $\phi = X\phi_2$ or $\phi = X_\omega \phi_2$, since $|\eta| > 1$ so it is obviously true that $\eta \models \phi$ iff $\omega_0 \models \tt$ and $\eta_1 \models \phi_2$ hold according to LTL semantics, and obviously $\tt \land X\phi_2$ is in NF($\phi$);
- If $\phi = \phi_1 \land \phi_2$, then $\eta \models \phi$ iff $\eta \models \phi_1$ and $\eta \models \phi_2$, and by induction hypothesis, there exists $\beta_i \land X\psi_i \in NF(\phi_i)$ ($i = 1, 2$) such that $\omega_0 \models \beta_i$ and $\eta_1 \models \psi_i$, hold, and if $\omega_0 \models \beta_i \land \beta_2$ and $\eta_1 \models \psi_1 \land \psi_2$ hold, in which $\beta_i \land \beta_2 \land X(\psi_1 \land \psi_2)$ is indeed in NF($\phi$); The case when $\phi = \phi_1 \lor \phi_2$ is similar;
- If $\phi = \phi_1 U \phi_2$, then $\eta \models \phi$ iff $\eta \models \phi_2$ or $\eta \models (\phi_1 \land X\phi)$.

Theorem 2. Given an LTL formula $\phi$ and a finite trace $\eta = \omega_0 \ldots \omega_n (n \geq 0)$, we have that $\eta \models \phi$ holds iff there exists a run of $T_0$ on $\eta$ which ends at the transition $\psi_1 \rightarrow \psi_2$ satisfying $CF(\alpha) \models \psi_1$.

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

We say the state $\psi_1$ in $T_0$ is accepting, if there exists a transition $\psi_1 \rightarrow \psi_2$ such that $CF(\alpha) \models \psi_1$. 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 LTL as follows:

1. If $\phi$ equals $\tt$, 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 LTL proposed in [8]. Given an LTL formula $\phi$, the constructed transition system $T_0$ has at worst the size of $2^{2^{|\phi|}}$, where $cl(\phi)$ is the set of subformulas of $\phi$.

4. Optimizations

In this section we propose some optimization strategies by exploiting SAT solvers. First we study the relationship between the satisfiability problems for LTL and LTL formulas.
Example 1. \( LTL \) formula \( \eta = \phi \land \phi \). We say an \( LTL \) formula is \( \eta \)-free if \( \eta \) does not have the \( X \) operator. Note that \( LTL \) formulas may contain the \( X \) operator, while standard \( LTL \) ones do not. Here we consider \( \eta \)-free \( LTL \) formulas, in which \( LTL \) and \( \eta \) 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 \( LTL \) to \( LTL \):

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

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

- If \( \phi \) is a literal \( p \), then \( \eta \models \phi \) implies \( \eta^* \models \phi \); And if \( \phi \) is \( tt \), then \( \eta^* \models \phi \); And if \( \phi = \phi_1 \land \phi_2 \), then \( \eta \models \phi_1 \land \phi_2 \) implies \( \eta^* \models \phi_1 \land \phi_2 \).
- By induction hypothesis we have \( \eta^* \models \phi_1 \) and \( \eta^* \models \phi_2 \). So \( \eta^* \models \phi_1 \land \phi_2 \).
- The proof is similar when \( \phi = \phi_1 \lor \phi_2 \).
- Similarly when \( \phi = \phi_1 R \phi_2 \), we know for every \( i \geq 0 \) it is true that \((\xi_i = \eta^*) \models \phi_2 \). Thus \( \eta^* \models \phi \) holds from the \( LTL \) semantics; The proof is done.

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

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

**Proof.** Assume \( \phi \) is an \( \eta \)-free \( LTL \) 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 \( \xi = \omega_0 \ldots \omega_n \). Equivalently, if \( \phi \) is an \( LTL \) formula and \( \phi \) is unsatisfiable, then the \( LTL \) formula \( \phi \) is also unsatisfiable. Note here the \( LTL \) formula \( \phi \) is \( \eta \)-free since it can be considered as an \( LTL \) formula.

**Example 1.** Consider the \( X \)-free formula \( \phi = GFa \land GF \neg a \), whose transition system is shown in Figure 1. If \( \phi \) is treated as an \( LTL \) formula, then we know that the infinite trace \((\alpha\{\neg a\})^\omega \) satisfies \( \phi \). However, if \( \phi \) is considered to be an \( LTL \) 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 \( \psi_1 \leadsto \psi_2 \) in \( T_n \) satisfies the condition \( CF(\alpha) \models \psi_1 \).

Consider another example formula \( \phi = G(aUb) \), whose transition system is shown in Figure 2. Here we can find an accepting state \((\phi, a \leadsto b) \land CF(b) \models \phi \). 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 \( \phi(\phi) \) is satisfiable then \( \phi \) is satisfiable. Since \( \phi(\phi) \) is essentially a boolean formula, we can check it efficiently using modern SAT solvers. However this cannot apply to \( LTL \) directly, which we illustrate in the following example.

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

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

**Definition 4** (Obligation Formulas). Given an \( LTL \) 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 ofr, ofg, 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 \( \phi = X\phi_2 \), then \( ofx(\phi) = off(\phi_2) \), \( ofr(\phi) = ff \) and \( ofg(\phi) = ff \);
- If \( \phi = X_\omega \phi_2 \), then \( ofx(\phi) = ofx(\phi_2) \), \( ofr(\phi) = ff \) and \( ofg(\phi) = tt \);
- If \( \phi = \phi_1 U \phi_2 \), then \( ofr(\phi) = off(\phi_2) \);
- If \( \phi = \phi_1 R \phi_2 \), then \( ofr(\phi) = off(\phi_2) \), \( ofr(\phi) = ofr(\phi_2) \) and \( ofg(\phi) = ofg(\phi_2) \);

For example in the third item, the equation represents actually three:

- \( ofx(\phi) = off(\phi_1) \land ofx(\phi_2) \), \( ofr(\phi) = off(\phi_1) \land ofr(\phi_2) \) and \( ofg(\phi) = ofg(\phi_2) \).

For off(\phi), the changes in comparison to [7] are the definition for release formulas, and introducing the \( X_\omega \) operator. For example, we have that \( off(GXa) \) is ff rather than a. Moreover, since the \( LTL \) formula \( GX_\omega a \) 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 \( LTL \) 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 \( \eta = A^\omega \) 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 $\phi = \phi_1 \wedge \phi_2$, then $off(\phi) = off(\phi_1) \wedge off(\phi_2)$ from Definition 4. So $off(\phi)$ is satisfiable implies that there exists $A \models off(\phi_1)$ and $A \models off(\phi_2)$. By induction hypothesis there exists $n_i = A^{n_i}$ ($n_i \geq 0$) such that $\psi_i = \phi_i (i = 1, 2)$. Assume $n_1 \geq n_2$, then let $n = n_1$. Then, $\eta = \phi_1 \wedge \phi_2$. 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 $off(\phi)$ is satisfiable iff $off(\phi_2)$ is satisfiable. So there exists a model $\phi_2$. By induction hypothesis, there exists $n$ such that $A^n = \phi_2$, thus according to $\text{LTL}_f$ semantics, we know $A^{n+1} = \phi$;

• If $\phi = \phi_1 R \phi_2$, then $off(\phi) = ofr(\phi_2)$. Thus $ofr(\phi_2)$ is also satisfiable. So there exists $A \models ofr(\phi_2)$, 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 $\eta = \phi_2$ implies $\eta = \phi$. The case for Until can be treated in a similar way, thus the proof is done.

\[\blacksquare\]

4.3 A Complete Acceleration Technique for Global Formulas

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

Theorem 5 (Obligation Acceleration for Global Formulas). For a global $\text{LTL}_f$ formula $\phi = G\psi$, we have that $\phi$ is satisfiable iff $ofg(\psi)$ 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 2, $\eta$ can run on $T_0$ and reaches an accepting state $\psi_1$, i.e., $\eta \rightarrow \psi_2$ and $CF(\alpha) = \psi_1$. 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 $CF(\alpha) = \psi_1$ is also true. Since $\phi$ is a global formula so $CF(\alpha) = \psi_1$ holds from Lemma 1. Then one can prove that $CF(\phi) = ofg(\psi)$ by structural induction over $\psi$ (it is left to readers here), which implies that $ofg(\psi)$ is satisfiable.

For the backward direction, assume $ofg(\psi)$ is satisfiable. So there exists $A \in \Sigma$ such that $A \models ofg(\psi)$. 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. \[\blacksquare\]

4.4 Acceleration for Unsatisfiable Formulas

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

5 Experiments

In this section we present an experimental evaluation. The algorithms are implemented in the $aalta$ tool\(^\text{4}\). We have implemented three optimization strategies. They are 1). $off$: the obligation acceleration technique for $\text{LTL}_f$ (Theorem 4); 2). $ofg$: the obligation acceleration for global $\text{LTL}_f$ formula (Theorem 5); 3). $off$: the acceleration for unsatisfiable formulas (Section 4.4). Note that all three optimizations can benefit from the power of modern SAT solvers, while their invoking times are limited according to our theories.

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

The experimental platform of this paper is the BlueBiou cluster \(^\text{5}\) 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 $\text{LTL}$ formulas are also $\text{LTL}_f$ formulas, we use existing $\text{LTL}$ 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 [11]. The benchmark suite there include earlier benchmark suites (e.g., [10]), 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 first column of Table 1.

<table>
<thead>
<tr>
<th>Formula type</th>
<th>aalta/sec (s)</th>
<th>Polsat/sec (s)</th>
<th>Polsat/aalta</th>
</tr>
</thead>
<tbody>
<tr>
<td>macorexunique</td>
<td>1.9</td>
<td>0.3</td>
<td>5.7</td>
</tr>
<tr>
<td>macardino-v3</td>
<td>1.4</td>
<td>0.4</td>
<td>3.5</td>
</tr>
<tr>
<td>macardino-v22</td>
<td>2.0</td>
<td>0.5</td>
<td>4.0</td>
</tr>
<tr>
<td>alaskanlift</td>
<td>4.0</td>
<td>0.5</td>
<td>8.0</td>
</tr>
<tr>
<td>alaskaneurans</td>
<td>1.2</td>
<td>0.3</td>
<td>4.0</td>
</tr>
<tr>
<td>alaskahamburg</td>
<td>210.9</td>
<td>210.9</td>
<td>1.0</td>
</tr>
<tr>
<td>franz/lot</td>
<td>1000.3</td>
<td>1000.3</td>
<td>1.0</td>
</tr>
<tr>
<td>franz/counter</td>
<td>1840.3</td>
<td>1840.3</td>
<td>1.0</td>
</tr>
<tr>
<td>franz/formulas</td>
<td>552.9</td>
<td>552.9</td>
<td>1.0</td>
</tr>
<tr>
<td>franz/pattern</td>
<td>22.3</td>
<td>22.3</td>
<td>1.0</td>
</tr>
<tr>
<td>schuppan/O2formula</td>
<td>2.3</td>
<td>2.3</td>
<td>1.0</td>
</tr>
<tr>
<td>schuppan/O2formula</td>
<td>160.9</td>
<td>160.9</td>
<td>1.0</td>
</tr>
<tr>
<td>schuppan/phil</td>
<td>256.3</td>
<td>256.3</td>
<td>1.0</td>
</tr>
<tr>
<td>gdp05x</td>
<td>10.5</td>
<td>10.5</td>
<td>1.0</td>
</tr>
<tr>
<td>gdp05y</td>
<td>2868.9</td>
<td>2868.9</td>
<td>1.0</td>
</tr>
<tr>
<td>gdp0012x</td>
<td>23.3</td>
<td>23.3</td>
<td>1.0</td>
</tr>
<tr>
<td>gdp0112y</td>
<td>4004.2</td>
<td>4004.2</td>
<td>1.0</td>
</tr>
<tr>
<td>Total</td>
<td>15541.3</td>
<td>15541.3</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 1 shows the experimental results on Schuppan-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.

The formulas in “Schuppan-collected/alaska/lift” are mostly unsatisfiable, which can be handled by the $ofg$ technique of $aalta$. On the other side, $Polsat$ needs more than 300 times to finish the checking. The same happens on the “Schuppan-collected/trp/N12x” patterns, in which $aalta$ is more than 1000 times faster. For the

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

\(^5\) http://www.rcsg.rice.edu/sharecore/bluebiou/
“Schuppan-collected/schuppan/O2formula” pattern formulas, aalta scales better due to the ofg technique. Among the results from aalta, totally 5879 out of 7448 formulas in the benchmark are checked by using the ofg technique. This indicates the ofg 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.

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 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 aalta. Statistics above show the optimizations are very useful. Moreover, one can conclude from Figure 4 that, aalta 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 aalta 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 $LTL_1$ formulas.

6 Conclusion
In this paper we have proposed a novel $LTL_1$ satisfiability-checking framework based on the $LTL_1$ 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 NSF Project No. 61021004. Lijun Zhang is supported by NSFC project No. 6136136002. 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