LTL Satisfiability Checking Revisited

Jianwen Li*, Lijun Zhang†, Geguang Pu*, Moshe Y. Vardi‡ and Jifeng He*

*Software Engineering, East China Normal University
†State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
‡Computer Science, Rice University

Abstract—We propose a novel algorithm for the satisfiability problem for Linear Temporal Logic (LTL). Existing approaches first transform the LTL formula into a Büchi automaton and then perform an emptiness checking of the resulting automaton. Instead, our approach works on-the-fly by inspecting the formula directly, thus enabling finding a satisfying model quickly without constructing the full automaton. This makes our algorithm particularly fast for satisfiable formulas. We report on a prototype implementation, showing that our approach significantly outperforms state-of-the-art tools.

I. INTRODUCTION

Model-checking tools are successfully used for checking whether systems have desired properties [CGP99]. The application of model-checking tools to complex systems involves a nontrivial step of creating a mathematical model of the system and translating the desired properties into a formal specification expressed by means of temporal assertions. When the model does not satisfy a given assertion, model-checking tools accompany this negative answer with a counterexample, which points to an inconsistency between the system and the desired behaviors.

The success of model checking led to the emergence of assertion-based design, where one starts the design process by formalizing designer intent by means of temporal assertions [FKL04]. Since at that early stage of the design process model checking cannot be employed, there is a need for techniques that would debug these assertions, as it is quite likely that such assertions contain errors [PSC+06]. A basic check is that of satisfiability [SC85]: checking that each temporal assertion can be satisfied and the full set of assertions be satisfied together. (A stronger test is that of realizability [ALW89], but is out of the scope of this paper.)

An in-depth empirical study of LTL satisfiability was undertaken by Rozier and Vardi [RV07, RV10]. A basic observation underlying their work is that LTL satisfiability checking can be reduced to model checking. Consider an LTL formula \( \varphi \) over a set \( Prop \) of atomic propositions. If a model \( M \) is universal, that is, it contains all possible traces over \( Prop \), then \( \varphi \) is satisfiable precisely when the model \( M \) does not satisfy \( \neg \varphi \). Thus, it is easy to add a satisfiability-checking feature to LTL model-checking tools.

LTL model checkers can be classified as explicit or symbolic. Explicit model checkers, such as SPIN [Hol97] or SPOT [DLP04], construct the state-space of the model explicitly and search for a trace falsifying the assertion [CVWY92]. In contrast, symbolic model checkers, such as CadenceSMV [McM99] or NuSMV [CCGR00], represent the model and analyze it symbolically using binary decision diagrams (BDDs) [BCM+92]. LTL model checkers follow the automata-theoretic approach [VW86], in which the complemented LTL assertion is explicitly or symbolically translated to a Büchi automaton, which is then composed with the model under verification; see also [Var07]. The model checker checks for nonemptiness, by searching for a trace of the model that is accepted by the automaton.

Rozier and Vardi [RV07, RV10] carried out an extensive experimental investigation of LTL satisfiability checking via a reduction to model checking. By using large LTL formulas, they offered challenging model-checking benchmarks to both explicit and symbolic model checkers. For symbolic model checking, they used CadenceSMV and NuSMV. For explicit model checking, they used SPIN as the search engine, and tested essentially all publicly available LTL translation tools. They used a wide variety of benchmark formulas, either generated randomly, as in [DGV99], or using scalable patterns.

The evidence marshalled by Rozier and Vardi for the conclusion in favor of the symbolic approach is quite compelling, but a close examination shows that it applies only to satisfiability checking via model checking. That is, if one chooses to perform satisfiability checking via a reduction to model checking, then the symbolic approach offers superior performance. It is conceivable, however, that a direct explicit approach to satisfiability checking would outperform the symbolic approach. In explicit model checking, it is possible to perform the nonemptiness test on the fly, that is, by letting the search algorithm drive the construction of the automaton [CVWY92]. (In fact, the on-the-fly approach was proposed also for model checking, but was not adopted by SPIN due to its software architecture [Hol97]).

In this paper we revisit the LTL satisfiability problem to examine the advantage of the on-the-fly approach. The driving intuition is that the on-the-fly approach may be quite advantageous in satisfiability checking, since it enables finding a model quickly without constructing the full automaton. Furthermore, the sole focus on satisfiability checking may be amenable to various heuristics that are not applicable in the context of model checking. We report here on a novel LTL satisfiability checking tool, Aalta, and demonstrate that it outperforms both SPOT and CadenceSMV.

To substantiate our results we also revisit the experimental
methodology of Rozier and Vardi. Their focus has been on
testing satisfiability of large LTL formulas, either scalable
patterns or random. But typical temporal assertions are rather
small [BAC98]. What makes the LTL satisfiability problem
hard is the fact that we need to check large conjunctions
of small temporal formulas. We describe here a new class
of challenging benchmarks, which are random conjunctions
of specification patterns from [BAC98]. Our conclusions on
the superiority of Aalta are based both on the benchmarks of
Rozier and Vardi and the newly introduced benchmarks.

The organization of the paper is as follows. We provide
preliminary material in Section II. In Section III we describe
the novel algorithm underlying Aalta. In Section IV we describe
the superiority of small temporal formulas. We describe here a new class
patterns or random. But typical temporal assertions are rather

II. Preliminaries

A. Linear Temporal Logic

Let $AP$ be a set of atomic properties. The syntax of LTL
formulas is defined by:

\[ \varphi ::= \text{tt} | \text{ff} | a | \neg a | \varphi \land \varphi | \varphi \lor \varphi | \varphi U \varphi | \varphi R \varphi | X \varphi \]

where $a \in AP$, $\varphi$ is an LTL formula. We use the usual abbreviations: $Fa = \text{tt} U a$, and $Ga = \text{ff} Ra$.

We say $\varphi$ is a propositional formula if it does not contain
temporal operators $X, U$ or $R$. We say $\varphi$ is a literal if it is an
atomic proposition or its negation. We use $L$ to denote the set
of literals. We use lower case letters $a, b, c$ to denote literals,
$\alpha, \beta, \gamma$ to denote propositional formulas, and $\varphi, \psi$ to denote
LTL formulas.

Note that, w.l.o.g., we are considering LTL formulas in
negation normal form (NNF) – all negations are pushed
down to literal level. LTL formulas are often interpreted over
$(2^{AP})^\omega$. Since we consider LTL in NNF forms, formulas are
interpreted on infinite literal sequences $\Sigma := (2^L)^\omega$.

A trace $\xi = \omega_0\omega_1\omega_2 \ldots$ is an infinite sequence over $\Sigma$. For $\xi$ and $k \geq 1$ we use $\xi^k = \omega_0\omega_1\ldots\omega_{k-1}$ to denote
the prefix of $\xi$ up to its $k$-th element, and $\xi_k = \omega_k\omega_{k+1}\ldots$ to
denote the suffix of $\xi$ from its $(k + 1)$-th element. Thus, $\xi = \xi_0\xi_k$. We use $n, n_0, \ldots$ to denote finite sequences in $\Sigma^*$. First, we
need the notion of consistent traces:

Definition 1 (Consistent Trace): We say a literal set $A$ is
consistent iff for all $a \in A$ we have that $\bigwedge a \neq \text{ff}$. A trace
$\xi = \omega_0\omega_1\ldots$ is consistent iff $\omega_i$ is consistent for all $i$.

Let $\omega \in \Sigma$ be a consistent set of literals, and $\alpha$
a propositional formula. We define $\omega \models \alpha$ in the standard way:
if $\alpha$ is a literal then $\omega \models a$ iff $a \in \omega$, $\omega \models \alpha_1 \land \alpha_2$ iff
$\omega \models \alpha_1$ and $\omega \models \alpha_2$, and $\omega \models X \alpha$ if $\omega \models \alpha_1$ or $\omega \models \alpha_2$. Moreover, $\omega \models \text{tt$\neq$ff}$. The semantics of
temporal operators with respect to a consistent trace $\xi$ is given by:

- $\xi \models \alpha$ iff $\xi^i \models \alpha$;
- $\xi \models X \varphi$ iff $\xi_1 \models \varphi$;
- $\xi \models U \varphi$ iff there exists $i \geq 0$ such that $\xi_i \models \varphi$ and
for all $0 \leq j < i, \xi_j \models \varphi$;
- $\xi \models R \varphi$ iff either $\xi_i \models \varphi_2$ for all $i \geq 0$, or
there exists $i \geq 0$ with $\xi_i \models \varphi_1 \land \varphi_2$ and $\xi_j \models \varphi_2$ for all $0 \leq j < i$;

According to the semantics, it holds $\varphi R \psi = \neg(\neg \varphi U \neg \psi)$.

Definition 2 (Satisfiability): We say $\varphi$ is satisfiable, de-
noted by $\text{SAT}(\varphi)$, if there exists a consistent trace $\xi$ such that $\xi \models \varphi$.

In the remainder of this paper, if not stated explicitly, all
traces considered are assumed to be consistent.

Notation. We define some notation that we use throughout
this paper.

- For a formula $\varphi$, we use $cl(\varphi)$ to denote the set of
subformulas of $\varphi$. We denote by $AP$ the set of
atoms appearing in $\varphi$, by $L$, the set of literals over $AP$,
and by $\Sigma$ the set of consistent literal sets over $AP$.
- Let $\varphi = \bigwedge_{i \in I} \varphi_i$ such that the root operator of $\varphi_i$ is
not a conjunctive. We define the set of conjuncts of $\varphi$ as
$CF(\varphi) := \{ \varphi_i | i \in I \}$. When $\varphi$ does not include
a conjunctive as a root operator, $CF(\varphi)$ includes only
$\varphi$ itself. The set of disjuncts $DF(\varphi)$ is defined in an
analogous way.
- For a formula $\varphi$ of the form $\varphi_1U \varphi_2$ or $\varphi_1R \varphi_2$, let
$\text{left}(\varphi)$ (right$(\varphi)$) be left (right) subformulas of $\varphi$.
- For all propositional formula $\alpha$ appearing in the paper,
we always check first whether $\alpha$ is satisfiable. If not, we
shall replace $\alpha$ by $\text{ff}$.

B. Normal Form Expansion

Our algorithm will extend the given formula based on the
notion of normal form for LTL formulas defined as follows:

Definition 3 (Normal Form): The normal form of an LTL
formula $\varphi$, denoted as $\text{NF}(\varphi)$, is a set defined as follows:

1) $\text{NF}(\varphi) = \{ \varphi \land X(\text{tt}) \}$ if $\varphi \neq \text{ff}$ is a propositional
formula. If $\varphi = \text{ff}$, we define $\text{NF}(\text{ff}) = \emptyset$;
2) $\text{NF}(X(\varphi)) = \{ \text{tt} \land X(\psi) | \psi \in \text{DF}(\varphi) \}$;
3) $\text{NF}(\varphi_1U\varphi_2) = \text{NF}(\varphi_2) \cup \text{NF}(\varphi_1 \land X(\varphi_1U\varphi_2))$;
4) $\text{NF}(\varphi_1R\varphi_2) = \text{NF}(\varphi_2) \cup \text{NF}(\varphi_1 \land X(\varphi_1R\varphi_2))$;
5) $\text{NF}(\varphi_1 \land \varphi_2) = \text{NF}(\varphi_1) \land \text{NF}(\varphi_2)$;
6) $\text{NF}(\varphi_1 \lor \varphi_2) = \{ (\alpha_1 \land \alpha_2) \land X(\psi_1 \lor \psi_2) | \forall i \in 1, 2, \alpha_i \land X(\psi_i) \in \text{NF}(\varphi_i) \}$;

From the definition it is obvious that if $\alpha \land X(\varphi) \in \text{NF}(\varphi)$,
then $\alpha$ is a conjunctive clause, namely a conjunction of
literals.

For a formula $\varphi$, our algorithm will detect successor formu-
las based on the set $\text{NF}(\varphi)$. First, we shall show that the
formula $\varphi$ is logically equivalent to $\bigvee \text{NF}(\varphi)$, here $\bigvee \text{NF}(\varphi)$
represents the formula $\bigvee_{i < k} (\alpha_j \land X\varphi_j)$ with $\alpha_j \land X\varphi_j \in
\text{NF}(\varphi)$ and $k = |\text{NF}(\varphi)|$. We note that the empty disjunction
(OR-ing over an empty set of operands) is defined as $\text{ff}$. Then,
we establish the equivalence property:

Lemma 1: For the formula $\varphi$, it holds $\varphi \equiv \bigvee \text{NF}(\varphi)$. 
The lemma below states that along the expansion the set of subformulas is decreasing (except the constant $tt$):

**Lemma 2:** If $\alpha \land X\psi \in NF(\varphi)$, then $CF(\psi) \subseteq d(\varphi) \cup \{tt\}$.

### III. New Satisfiability Checking Algorithm

We first illustrate the main idea of our methodology. The key of our on-the-fly-approach is the notion of obligation set. As we show below, satisfying an obligation set gives a sufficient condition for satisfying a given formula. For a given formula, the obligation set contains several possible obligations, each obligation consists of some literals that characterize a possible way of satisfying the formula. We give the flavor of this notion in term of a few examples:

- For the formula $aUb$ or $aRb$ the obligation set is $\{\{b\}\}$; for the formula $\bigvee_{1 \leq i \leq n} a_i U b_i$ the obligation set is $\{\{b_1\}, \{b_2\}, \ldots, \{b_n\}\}$; for the formula $\bigwedge_{1 \leq i \leq n} a_i U b_i$ the obligation set is $\{\{b_1, b_2, \ldots, b_n\}\}$.

For a formula $\varphi$, if one of its obligations $O \subseteq 2^L$ is consistent, i.e. $\bigwedge_{a \in O} a \not\equiv \text{ff}$, the trace $\xi = O^\omega$ is consistent, and moreover, it satisfies the corresponding formula. If there is no consistent obligation, we construct a Transition System $T_\varphi$ for $\varphi$ on-the-fly. States consist of reachable formulas, and transitions are obtained by unrolling the current formula according to the normal form expansion. In our construction we need first to tag the subformulas such that all the literals are identified by their positions in until subformulas. We need the notion of accepting SCCs:

**Definition 4:** For an SCC $B$ of a transition system (here it is viewed as a directed graph) we denote by $L(B)$ the set of all literals appearing in transitions between states in $B$. We say $B$ is accepting if $L(B)$ is a superset of some obligation $O \in Olg(\psi)$ and $\psi \in B$.

We show that the formula is satisfiable if an accepting strongly connected component (SCC) is found that contains a consistent obligation. Summarizing, combining with the trivial on-the-fly checking, our approach works as follows:

1. If a consistent obligation is found in the processed states so far, then the formula is satisfiable;
2. If an accepting SCC is found during the generation of the transition system, then the formula is also satisfiable;
3. In the worst case, the formula is unsatisfiable after exploring on the whole transition system.

Now we present our approach in the following subsections.

#### A. Obligation Set

The key of our on-the-fly satisfiability algorithm is the notion of obligation set, defined for the input formula $\varphi$:

**Definition 5 (Obligation Set):** For a formula $\varphi$, we define its obligation set, denoted by $Olg(\varphi)$, as follows:

1) $Olg(tt) = \emptyset$ and $Olg(ff) = \{\{ff\}\}$;
2) If $\varphi$ is a literal, $Olg(\varphi) = \{\{\varphi\}\}$;
3) If $\varphi = X\psi$, $Olg(\varphi) = Olg(\psi)$;
4) If $\varphi = \psi_1 \lor \psi_2$, $Olg(\varphi) = Olg(\psi_1) \cup Olg(\psi_2)$;
5) If $\varphi = \psi_1 \land \psi_2$, $Olg(\varphi) = \{O_1 \cup O_2 \mid O_1 \in Olg(\psi_1) \land O_2 \in Olg(\psi_2)\}$;
6) If $\varphi = \psi_1 U \psi_2$ or $\psi_1 R \psi_2$, $Olg(\varphi) = Olg(\psi_2)$.

For $O \in Olg(\varphi)$, we refer to it as an obligation of $\varphi$. Moreover, we say $O$ is a consistent obligation iff $\bigwedge a \not\equiv \text{ff}$ holds, where $a \in O$.

The obligation set $Olg(\varphi)$ enumerates all obligations the given formula $\varphi$ is subject to. Each obligation $O \in Olg(\varphi)$ characterizes a possible way to resolve the obligations proposed by the formula, in the sense that a formula is satisfiable if one of its obligation can be resolved accordingly. The particular obligation $\{\text{ff}\}$ can never be resolved.

The power of this characterization is best explained by the following theorem:

**Theorem 1:** Assume $O \in Olg(\varphi)$ is a consistent obligation. Then, $O^\omega \models \varphi$.

The theorem can be proven by simple structural induction over $\varphi$. We illustrate the usefulness of the theorem by the following example:

**Example 1:**

- Consider $G(aRb)$. It has only one obligation $\{b\}$ which is consistent. Thus, the trace $\{\{b\}\}$ satisfies $G(aRb)$.

- Consider the formula $\varphi := GF(a \land b) \land F(\neg a)$: first, the obligation $\{a, b, \neg a\}$ is not consistent. Further, the normal form $NF(\varphi)$ contains $\neg a \land X(GF(a \land b))$. Thus we can reach the formula $GF(a \land b)$ along $\neg a$. Moreover, $GF(a \land b)$ has a consistent obligation set $O = \{a, b\}$. Then the trace $\xi := \{\neg a\} O^\omega \models \varphi$.

- The opposite direction of Theorem 1 does not hold. Consider for example the formula $F(a) \land G(\neg a)$. It has a single obligation $\{a, \neg a\}$ which is not consistent. However, $\{a\} \{\neg a\}^\omega$ is a satisfying trace. Consider another formula $F(a) \land G(\neg a)$ which has the same obligation. This formula is obviously not satisfiable.

Theorem 1 is indeed very useful: it returns an affirmative answer as far as a consistent obligation is found for the current candidate. This is often the case for satisfiable formulas. In the following sections, we exploit this notion to derive an on-the-fly algorithm for all formulas.

#### B. Tagging Input Formulas

First, from the discussions and definitions above, we observe that the obligation set ignores the left subformulas of until and release operators. If Theorem 1 does not give an affirmative answer, the left subformulas then play a role in our construction. As a preparation for the general case, we need first tag the atoms in the input formula in our approach such that they can be differentiated. The example below illustrates why tagging is useful in our construction:

**Example 2:** Consider $\varphi := (a \lor b) U (Ga)$, in which the atom $a$ appears twice. Without tagging, we can see there exists a transition $\varphi \xrightarrow{a} \varphi \xrightarrow{b} \varphi$ which forms a SCC $B$, and $L(B) = \{a, b\}$ is a superset of the obligation $\{a\}$. However, obviously, the infinite path through this SCC can not satisfy $\varphi$. 

On the other side, our algorithm first tags the formula to \( \varphi_t = (a_1 \lor b)U(Ga_2) \). Then the transition system for the tagged formula will be constructed. The tagged SCC \( B \) has label \( L(B) = \{a_1, b\} \) which is not a superset of the obligation \( \{a_2\} \). Thus \( B \) is not an accepting SCC, and the infinite path through SCC \( \varphi \xrightarrow{a_1} \varphi \xrightarrow{b} \varphi \) can not satisfy \( \varphi \).

We need some notations to formalize the tagging process. For a given input formula \( \varphi \) under consideration. For each atom \( a \) appearing in \( \varphi \), we enumerate all occurrences of \( a \) by \( S_a := \{a_1, a_2, \ldots, a_n\} \), provided \( a \) appears \( n \) times in \( \varphi \). The easiest tagging function is the identity function, i.e., we consider all \( a_i \) syntactically different, but semantically equivalent. The complexity of our approach will depend on the number of syntactically different atoms. This tagging is inefficient: below we give an improved tagging function.

Given a formula \( \varphi \) we denote \( U(\varphi) \) the set of until subformulas of \( \varphi \). Then:

**Definition 6 (Tagging Formula):** Let \( a \in AP \) be an atom appearing in \( \varphi \). Then, the tagging function \( F_a : S_a \to 2^{U(\varphi)} \) is defined as: \( \psi \in F_a(a_i) \) if \( a_i \) appears in \( right(\psi) \).

We define the tagged formula \( \varphi_t \) as the formula obtained by replacing \( a_i \) by \( aF_a(a_i) \) for each \( a_i \in S_a \).

Thus, after tagging \( AP_\varphi \), will contain more atoms. Note that all these new copies are semantically equivalent to \( a \), i.e., \( aF_a(a_i) \equiv a \) for all \( aF_a(a_i) \). Given a tagging function \( F_a \), two copies \( a_i, a_j \) are syntactically equivalent iff \( F_a(a_i) = F_a(a_j) \). More explicitly, \( a_1 \equiv a_2 \iff F_a(a_1) = F_a(a_2) \).

As an example, consider \( \varphi = aU(a \land aU \neg a) \). Let \( \psi_u = aU(a \land aU \neg a) \), and \( S_a = \{a_1, a_2, a_3, a_4\} \). From Definition 6 we know \( F_a(a_1) = \emptyset, F_a(a_2) = F_a(a_3) = \{\varphi\}, \) and \( F_a(a_4) = \{\varphi, \neg \varphi\} \). So the tagging function will introduce three syntactically different copies of \( a \), and we denote \( \varphi_t \) by \( aU(a_2 \land a_2U \neg a_4) \). Here even \( a_1, a_2, a_4 \) are syntactically different, they are semantically equivalent. Thus it holds for example \( a_2 \land \neg a_4 \equiv \emptyset \).

Note the size of subformulas may increase after tagging. According to Definition 6 the following lemma is obvious:

**Lemma 3 (Tagging Cost):** Let \( \varphi \) be the input formula and \( \varphi_t \) the formula obtained after tagging \( \varphi \). Then, \( |cl(\varphi_t)| \leq 2^{m \cdot |cl(\varphi)|} \), where \( m = |U(\varphi)| \).

**C. LTL Transition System**

First, we note that for all formula \( \varphi \), it holds \( \varphi \equiv \varphi_t \). This implies \( SAT(\varphi) \) iff \( SAT(\varphi_t) \). As our approach will work with the tagged formula \( \varphi_t \), in the remaining of the paper:

- Syntactically: for a given input formula \( \varphi \), all atoms are ranging over the tagged atoms appearing in \( \varphi_t \), thus \( AP = AP_{\varphi_t} \), \( L = L_{\varphi_t} \), and \( \Sigma = \Sigma_{\varphi_t} = 2^{L_{\varphi_t}} \).
- Semantically: tagged atoms are equivalent to the original atom. Thus, the notion of consistent traces and consistent obligations are defined by taking the semantical equivalences of tagged atoms into consideration.

For a given formula \( \varphi \), we shall define below a labelled transition system \( T_{\varphi} \) for it:

**Definition 7 (LTL Transition System):** Let \( \varphi \) be the input formula and \( \varphi_t \) the tagged formula. The labeled transition system \( T_{\varphi} \) is a tuple \( \langle Act, S_{\varphi}, \rightarrow, \varphi_t \rangle \) where:

1) \( \varphi_t \) is the initial state,
2) \( Act \) is the set of conjunctive formulas over \( L_{\varphi_t} \),
3) the transition relation \( \rightarrow \subseteq S_{\varphi} \times Act \times S_{\varphi} \) is defined by: \( \psi_1 \xrightarrow{\alpha} \psi_2 \) iff there exists \( \alpha \lor X(\psi_2) \in NF(\psi_1) \);
4) \( S_{\varphi} \) is the smallest set of formulas such that \( \psi_1 \in S_{\varphi} \), and \( \psi_1 \xrightarrow{\alpha} \psi_2 \) implies \( \psi_2 \in S_{\varphi} \).

Again, note that the transition system for \( \varphi \) is defined by starting from the tagged formula \( \varphi_t \). The set of states is the set of formulas reachable from \( \varphi_t \), with \( \varphi_t \) as the initial state. Note between two states there can be more transitions. A state \( \varphi \) has no outgoing transitions iff for all \( \alpha \lor X \psi \in NF(\varphi) \) and \( \alpha \) is equivalent to \( \emptyset \) in this case \( \varphi \) is not satisfiable. Now we introduce the notion of accepting traces:

**Definition 8: A run of \( T_{\varphi} \) is a (finite or infinite) path \( r = \varphi \xrightarrow{\alpha_1} \psi_1 \xrightarrow{\alpha_2} \psi_2 \xrightarrow{\alpha_3} \ldots \) in \( T_{\varphi} \). A trace \( \xi = \omega_0 \omega_1 \ldots \in \Sigma^* \) is accepted by the run \( r \) if \( \omega_i = \alpha_i \) for all \( i \).

For \( \omega \in \Sigma \), we write \( \varphi \xrightarrow{\omega} \psi \) if there exists \( \varphi \xrightarrow{\xi} \psi \) such that \( \omega \equiv \xi \). For a finite sequence \( \eta = \omega_0 \omega_1 \ldots \omega_k \), we write \( \varphi \xrightarrow{\bar{\omega}} \psi \) if \( \varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_1} \psi_2 \xrightarrow{\omega_2} \ldots \xrightarrow{\omega_k} \psi_{k+1} = \psi \).

More specially, we write \( \varphi \xrightarrow{\xi} \varphi \) iff \( \xi \) can be written as \( \xi = \eta_0 \eta_1 \eta_2 \ldots \) such that \( \eta_i \) is a finite sequence and \( \varphi \xrightarrow{\bar{\omega}} \varphi \) for all \( i \geq 0 \).

Lemma 2 implies the following properties of \( |S_{\varphi}| \):

**Corollary 1:** For any formula \( \varphi \), it holds:

1) for all \( \psi \in S_{\varphi} \), it holds \( CF(\psi) \subseteq cl(\varphi_t) \cup \{tt\} \),
2) \( |S_{\varphi}| \leq 2^n + 1 \) where \( n \) denotes the number of subformulas of \( \varphi_t \).

**D. On-the-fly Satisfiability Algorithm**

First, we introduce some notations:

- For notational convenience, we fix \( \lambda \) as our input formula in this section. Let \( T_\lambda \) be the transition system for the tagged formula \( \lambda_t \).
- For all \( \varphi \in S_\lambda \), we denote by \( ST_{\varphi} \) the subsystem of \( T_\lambda \) consisting all states reachable from \( \varphi \).

Now we present our main theorem:

**Theorem 2:** Let \( \varphi \in S_\lambda \). Then, \( SAT(\varphi) \) iff there exists a SCC \( B \) of \( ST_{\varphi} \) and a state \( \psi \) in \( B \) such that \( L(B) \) is a superset of some obligation \( O \in Olg(\psi) \).

**Sketch:** The full proof is given in the appendix. We sketch the proof idea here, which is best illustrated in Figure 1. Let \( \xi = \omega_0 \omega_1 \ldots \) be a (consistent) trace such that \( \xi \models \varphi \). Then, there is a run in \( ST_{\varphi} \) accepting \( \xi \), i.e., we have \( \varphi \xrightarrow{\bar{\omega}} \psi_1 \xrightarrow{\bar{\omega}} \psi_2 \xrightarrow{\bar{\omega}} \ldots \) After some prefix \( \xi^n \), since there are only finitely many states reachable, we will be able to partition the suffix into \( \eta_1 \eta_2 \ldots \) where all \( \eta_i \) are finite sequences, and all \( \eta_i \) lead from \( \psi \) to itself. Such formula \( \psi \) will be referred to as a looping formula.
Looping formulas arising from $U$ and $R$ operators must be treated differently. For instance $aRb \Rightarrow aRb$ resolves the obligation $\{b\}$, however $aUb \Rightarrow aUb$ does not. To characterize this difference, we shall memorize atoms appearing along the edges and check whether the obligation $\{b\}$ is met. Interestingly, $R$ operators are easy to handle, but things get more involved if the same atom appears on both sides of $U$ operators, such as $aUa$. Here we make use the fact that we are working on the tagged formula, and our transition system is labelled with tagged atoms. Thus we can efficiently check whether appearing atoms correspond to those obligations for $U$ formulas or not. With these notions, the theorem can be proven by the following idea: any edge label is a propositional formula that is not ff, but any run in the transition system induces a consistent trace, which can be proven to satisfy the formula if the collected atoms along the trace can produce an obligation. Thus, the formula is satisfiable if and only if we can find an SCC $B$ such that $O \subseteq L(B)$.

The above theorem states that the satisfiability of an LTL formula $\lambda$ can be checked directly on the transition system $T$. Together with Theorem 1, we arrive at the following on-the-fly algorithm, which we refer to as OFOA($\lambda$):

1) We first tag the formula and get $\lambda_1$. Then we construct $T_{\lambda_1}$, where we explore the states in an on-the-fly manner, by performing nest depth-first [CVWY92].

2) Whenever a formula is found, we compute the obligation set. In case that it contains a consistent obligation set, we return true because of Theorem 1.

3) If a SCC $B$ is reached, $\varphi \in B$, and $L(B)$ is a superset of some obligation set $O \in \text{Olg}(\varphi)$, we return true.

4) If all SCCs are explored, and all do not have the property in step 3, we return false.

We discuss briefly the complexity of the proposed algorithm. First, we remark that the worst case scenario happens if all extended formulas do not contain any consistent sets, which happens for instance for the formula $GF(a) \land GF(\neg a)$. Given the input formula $\lambda$, we first construct $\lambda_1$. By Lemma 5 we have $|d(\lambda_1)| \leq 2^m \cdot |d(\lambda)|$, where $m = |U(\lambda)|$ is the number of until subformulas of $\lambda$. By Corollary 1 the number of states is bounded by $|S_\varphi| \leq 2^n + 1$, where $n = 2^m \cdot |d(\lambda)|$. In addition, for each reachable state $\varphi$, we compute the obligation $\text{Olg}(\varphi)$, which is exponential in the number of conjunctions in $\varphi$, but linear in other operators.

IV. EXPERIMENTAL METHODOLOGY

We have implemented our algorithms in a tool called Aalta1. We denote Theorem 1 as the obligation acceleration technique (OA, for short). Similarly, we refer to the technique that underlie Theorem 2 as the on-the-fly technique (OF, for short). In the tool we have the following two configurations: (i) OF: On-the-fly checking without OA, (ii) OFOA: On-the-fly checking with OA (default).

1) Testing tools: In default Aalta is implelmented with the on-the-fly plus obligation acceleration (OFA) techniques. In this paper we compare the performance of Aalta with two other LTL satisfiability solvers: PANDA+CadenceSMV [RV11] and SPOT [DLPH04]. SPOT is considered as the best explicit LTL-to-Büchi translator [RV07], [RV10]. Its most recent version (1.0.2) has an integrated emptiness checking implementation (with "-e" flag) and it is considerably improved since the benchmarking in [RV07], [RV10]. That benchmarking showed the superiority of CadenceSMV for LTL satisfiability checking, and this has been further improved in PANDA+CadenceSMV [RV11]. Thus, we benchmarked all three tools. (Since PANDA consists of 30 different symbolic encodings, we run all these encodings in parallel and chose the best result among them.)

2) Platform: We conducted our benchmarking on the SUG@R cluster in Rice University2. SUG@R is an Intel Xeon compute cluster. It contains 134 SunFire x4150 nodes from Sun Microsystems. Each node has two quad-core Intel Xeon processors running at 2.83GHz, yielding a system-wide total of 1064 processor cores. In our experiments, each test is run on a single core with a timeout of 10 minutes for each test formula. The OS is Red Hat Enterprise 5 Linux, 2.6.18 kernel. Times are measured using the Unix time command. All time measurements are “end-to-end”; we measure the time starting from formula input to the satisfiability-checking result (SAT or UNSAT).

3) Input Formulas: We use here the benchmarks from [RV07], [RV10], [RV11]. These include random, pattern and counter formulas. We tested over 60,000 random formulas and all eight kinds of pattern (lengths varying from 1 to 1000) and four counter formulas (lengths varying from 1 to 20). These benchmarks are suitable for testing satisfiability of large formulas. Typical temporal assertions are, however, quite small in practice [BAC98]. What makes the LTL satisfiability problem hard is the fact that we need to check large conjunctions of small temporal formulas, as we need to check that the conjunction of all input assertions is also satisfiable. We introduce here a novel class of challenging LTL benchmarks, which are random conjunctions of specification patterns from [BAC98].

Random Conjunction Formulas: Formally, a random conjunction formula $RC(n)$ has the form: $RC(n) = \bigwedge_{1 \leq i \leq n} F_i(v_1, v_2, \ldots, v_k)$, where $n$ is the number of conjuncts and $F_i(1 \leq i \leq n)$ is a randomly chosen property pattern formula used frequently in practice [BAC98]. The propositions $\{v_1, v_2, \ldots, v_k\}$ used in these formulas are also chosen randomly. More precisely, we generate the class of random conjunction formulas in the following way:

1 www.lab205.org/aalta
2 http://www.rcsg.rice.edu/sugar/
Fig. 2. Experimental results for random formulas with 3 variables.

Fig. 3. Experimental results for $R(n) = \bigwedge_{i=1}^{n}(GF_{p_i} \lor FG_{p_{i+1}})$.

Fig. 4. Experimental results for $S(n) = \bigwedge_{1 \leq i \leq n} G_{p_i}$.

Fig. 5. Experimental results for random conjunctive formulas.

1) We extract all pattern formulas.
2) For a formula in $RC(n)$, we conjoin $n$ pattern formulas selected randomly. In each pattern formula, we instantiate the variables as random literals (positive or negative) over a set of six atomic propositions.
3) In our experiments we generated 500 random formulas for each $n$.
4) Correctness: To test Aalta’s correctness, we assume that the results from PANDA+CadenceSMV and SPOT are correct and we compare the results Aalta’s. Aalta successfully passes all the tests.

V. EXPERIMENTAL RESULTS

In this section we analyze the experimental results. Generally speaking, our results demonstrate that Aalta outperforms both SPOT and PANDA+CadenceSMV.

A. Aalta performs best for random formulas.

We first compare the three tools on random benchmarks. We use here three atomic propositions and formula length of up to 200. In total, we tested 20,000 formulas. Fig. 2 shows performance results for the three tools, where for each length we report average running time on 500 formulas. We can see that Aalta outperforms the other tools on random formulas. In fact, Aalta significantly outperforms the other tools; for
60% of the formulas, Aalta returns in a few millisecond, while SPOT and PANDA+CadenceSMV takes tens of seconds. In fact, Aalta completes checking all 20,000 formulas in one hour, where neither SPOT nor PANDA+CadenceSMV were able to complete in 40 hours. The superiority of Aalta stems from the fact that 95% of the test formulas turn out to be satisfiable; furthermore, 80% of them are checked using the obligation acceleration technique. Indeed, on unsatisfiable formulas PANDA+CadenceSMV is faster than Aalta, performing nearly twice as fast as than Aalta. Overall, however, Aalta’s heuristics for quick satisfiability testing do pay off.

B. Aalta performs best for most of the pattern formulas.

Our experiments show that Aalta performs best for all pattern formulas except the S-pattern formula, where SPOT performs best. For example, Fig. 3 displays the comparing results for the R-pattern formulas, where SPOT scales exponentially with formula length, while PANDA+CadenceSMV is quicker, and Aalta performs the best. Here it is clear that SPOT pays the price for not performing the automaton non-emptiness test on the fly, as the automata scale exponentially. In fact, even Aalta scales exponentially for R-pattern formulas without the obligation acceleration technique.

For S-pattern formulas, the results are shown in Fig. 4. Here, all three tools scale polynomially, since automata size scales linearly. SPOT performs better than Aalta, as its automaton construction is faster.

C. Aalta performs best for random conjunction formulas.

Checking satisfiability of random conjunction formulas is quite challenging, but Aalta still performs best. The results are shown in Fig. 5. The number of conjuncts extends only to 15 (with average formula length of 100) and all tools time out for larger formulas. The advantage of Aalta here is less marked; it performs about twice as fast as SPOT and PANDA+CadenceSMV.

1) The number of cases that can be checked by the obligation acceleration technique is much smaller here. For random conjunction, less than 20% cases can be checked by the obligation acceleration, and only about 30% can be checked by finding an accepting SCC;
2) The fraction of unsatisfiable formulas is higher here; about 50% of the formulas are satisfiable, so Aalta’s advantage in quick satisfiability finding is reduced.

Checking satisfiability for random-conjunction formulas emerges as a challenging problem, requiring further research. It would be interesting to combine Aalta with the abstraction technique of [CRST07].

D. The obligation acceleration enhances on-the-fly checking.

One of the effective heuristics of Aalta is the OA technique: a consistent obligation implies satisfiability directly. Now we compare here the results from Aalta implemented with the OFOA and pure OF strategies in checking random and random conjunction formulas. Fig. 6 and Fig. 7 indicate that OA indeed plays a key role. For random formulas the OFOA strategy performs much faster than the pure OF strategy. Moreover, the OFOA strategy can be even exponentially better for special cases, such as the R pattern formulas mentioned above. Although the advantage declines for random conjunction formulas, the OFOA strategy is still twice as fast as the OF strategy.

VI. RELATED WORK

The classical approach to LTL satisfiability checking is by reduction to model checking. This can be implemented using either explicit-state techniques or symbolic techniques. Rozier and Vardi [RV07], [RV10] studied this approach and benchmarked several tools. They concluded that the combination of SPIN [Hol97] and SPOT [DLP04] yields the best
Several authors described direct approaches to LTL satisfiability checking, including Wolper [Wol85] and Schwendimann [Sch98]. Wolper’s algorithm uses multiple-pass incremental tableau procedure, while Schwendimann’s requires only one pass. Although, theoretically, the multiple-pass algorithm works in EXPTIME and the worst complexity of the one-pass works in 2EXPTIME, Goranko, Kyrilov, and Shkatov [VGS10] showed that, in practice, the one-pass procedure is more efficient than the multiple-pass one. Yet another approach to LTL satisfiability is based on temporal resolution [FDP01]. Cimati et al. described a Boolean abstraction technique for LTL satisfiability, which can be combined with different satisfiability-checking techniques [CRST07]. De Wulf et al. [DDMR08] described a semi-symbolic approach based on anti-chains.

Schuppan and Darmawan [SD11] performed a comprehensive experimental evaluation of LTL satisfiability solvers. They considered a wide range of solvers implementing three major classes of algorithms, based on model checking, tableau, and temporal resolution. They concluded that no solver dominates or solves all instances, and recommend a portfolio approach, similar to that of [RV11].

Our tool, Aalta, is closest in spirit to the model-checking approach, but it combines automaton generation and nonemptiness checking in an on-the-fly approach. In this paper we demonstrate its performance advantage over model-checking-based tools. We leave comprehensive comparison in the style of [SD11] to future work.

VII. CONCLUSIONS

In this paper, we proposed a novel on-the-fly satisfiability checking approach for LTL formulas. Our approach exploits the notion of obligation set, which provides efficient ways for identifying many satisfiable formulas. We have implemented a tool, Aalta, and run experiments using existing and new benchmarks. In most of the cases, Aalta significantly outperforms existing model-checking-based LTL satisfiability solvers.

REFERENCES


APPENDIX

In this appendix we provide all of the missing proofs in the main paper.

A. Proofs of Lemma 7

Proof: We prove it by structural induction over ϕ:

• The case that ϕ is propositional formula or a next formula is trivial by definition.

• If ϕ = ϕ₁ ∨ ϕ₂, then applying induction hypothesis we have ϕ ≡ ϕ₁ ∨ ϕ₂ ≡ ∨ NF(ϕ₁) ∨ ∨ NF(ϕ₂) ≡ ∨ NF(ϕ₁) ∨ NF(ϕ₂).

• If ϕ = ϕ₁ ∧ ϕ₂, then applying induction hypothesis we have ϕ ≡ ϕ₁ ∧ ϕ₂ ≡ (NF(ϕ₁) ∧ NF(ϕ₂)). By inspection, this is equivalent to NF(ϕ₁ ∧ ϕ₂).

• If ϕ = ϕ₂Uϕ₁, then by Definition 5 we know NF(ϕ) = NF(ϕ₂) U NF(ϕ₁ ∧ ϕ₂), and that ϕ ≡ ϕ₁ ∧ ϕ₂ ∨ (ϕ₁ ∧ Xϕ₂). By induction hypothesis we have that ϕ₁ ≡ ∨ NF(ϕ₁) ∨ NF(ϕ₂). Moreover, we also proved previously that ϕ₁ ∧ Xϕ ≡ ∨ NF(ϕ₁ ∧ Xϕ). Thus we can prove that ϕ ≡ ϕ₁ ∨ (ϕ₁ ∧ Xϕ₂) ≡ ∨ NF(ϕ₁) ∨ NF(ϕ₂).

• The case ϕ = ϕ₁Rϕ₂ is similar to the case when ϕ is a Until formula.

B. Proof of Lemma 2

Proof: First, CF(ψ) ⊆ cl(ϕ) is by structural induction over ϕ. The base cases ϕ = tt, ff and propositional formulas are trivial. Otherwise:

1) If ϕ = ϕ₁ ∨ ϕ₂. Then NF(ϕ) = NF(ϕ₁) U NF(ϕ₂). So α ∧ Xψ ∈ NF(ϕ₁) with i = 1 or i = 2. By induction hypothesis we have CF(ψ) ⊆ cl(ϕ₁) ⊆ cl(ϕ).

2) If ϕ = Xϕ₁. In this case we have NF(ϕ) = {tt ∧ Xψ′ | ϕ′ ⊆ DF(ϕ₁)}. Since CF(ϕ₁) ⊆ cl(ϕ₁) ⊆ cl(ϕ), so we have CF(ψ) ⊆ cl(ϕ).

3) If ϕ = ϕ₁ ∧ ϕ₂, then we know for every α ∧ Xψ ∈ NF(ϕ) there exists α₁ ∧ Xψ₁ ∈ NF(ϕ₁) and α₂ ∧ Xψ₂ ∈ NF(ϕ₂) such that α = α₁ ∧ α₂ and ψ = ψ₁ ∧ ψ₂. Since by induction hypothesis we know CF(ψ₁) ⊆ cl(ϕ₁) and CF(ψ₂) ⊆ cl(ϕ₂), so CF(ψ) ⊆ cl(ϕ) holds.

4) If ϕ = ϕ₁Uϕ₂. We have two cases. Either we have the right expansion α ∧ Xψ ∈ NF(ϕ₂), in which case CF(ψ) ⊆ cl(ϕ₂) ⊆ cl(ϕ) follows directly by induction hypothesis. For the left expansion case, we have α ∧ Xψ ∈ NF(ϕ₁ ∧ Xϕ₂), implying that there exists α₁ ∧ Xψ₁ ∈ NF(ϕ₁) such that ψ = ψ₁ ∧ ϕ₁. So CF(ψ) ⊆ cl(ϕ₁) follows by exploiting the induction hypothesis that CF(ψ₁) ⊆ cl(ϕ₁) ⊆ cl(ϕ).

5) If ϕ = ϕ₁Rϕ₂, then we can proves similarly as the case when ϕ = ϕ₁Uϕ₂.

C. Proof of Theorem 7

Proof: We prove by structural induction over ϕ. The basic cases when ϕ is either tt, ff and or any literal are trivial. For the induction step we consider:

• If ϕ = Xψ then we have O ∈ Olg(ϕ) = Olg(ψ). By inductive hypothesis we have O₂ = ψ. Thus, O₂ = ϕ as well;

• If ϕ = ϕ₁Uϕ₂ then we have O ∈ Olg(ϕ₂), according to the definition of obligation set. By inductive hypothesis we have O₂ = ϕ₂. Moreover according to LTL semantics we know O₂ = ϕ as well;

• The case ϕ = ϕ₁Rϕ₂ is similar to previous case;

• If ϕ = ϕ₁ ∨ ϕ₂, we have O ∈ Olg(ϕ₁) or O ∈ Olg(ϕ₂). Assume O ∈ Olg(ϕ₁) without loss of generality. By inductive hypothesis we have O₂ = ϕ₁, implying O₂ = ϕ as well;

• If ϕ = ϕ₁ ∧ ϕ₂ then there exist O₁ ∈ Olg(ϕ₁) and O₂ ∈ Olg(ϕ₂) such that O = O₁ ∪ O₂. Since O is consistent so both O₁ and O₂ must be consistent. By inductive hypothesis we have O₁₂ = ϕ₁ and O₂₁ = ϕ₂. Again since O is consistent have O₂ = ϕ₁ ∧ ϕ₂.

D. Proof of Corollary 7

Proof: The first clause follows by a simple induction over the path from ϕ₁ to ψ. Note the constant 1 is due to the possibility of producing tt along the expansion.

E. Proof of Theorem 2

This section is devoted to the proof of Theorem 2. We organize the proof as follows. We first introduce the notion of looping formulas and discuss their properties. We then continue with the soundness and completeness proofs of the theorem.

Assumption. Throughout the section, we have the following assumptions:

• λ denotes the fixed input formula, Tλ is the transition system for λ.

• all traces are over Σω ⊆ 2Lλ, i.e., the set of consistent literals over Lλ.

• all formulas appearing in this section are taken from the set of states Sλ, i.e., ϕ, ψ, ..., ∈ Sλ. Thus, all formulas in this appendix will be ranging over tagged atoms appearing in λ.

1) Looping Formulas and Their Properties: We start with a simple lemma about the relation between satisfiability and the transitions:

Lemma 4: Let ξ ∈ Σω be a trace. Then, for all n ≥ 1, there exists ψ such that ξ ⊨ ϕ ⇔ Cn ⊨ ϕ ∧ ξ_n ⊨ ψ.

Proof: Let ξ = ω₁ω₂ ... ωₙ. We prove the lemma by induction over the number n.
The order "<" is obviously a partial order. For a looping formula \( \varphi \), the set \( CF(\varphi) \) possesses at least one minimal element (w.r.t. the order "<"). Below we prove that all minimal elements of the set expand either to \( tt \) or themselves.

**Lemma 6:** If \( \varphi \not\rightarrow_{<} \) \( \varphi \) then for any minimal element \( \psi \in CF(\varphi) \) we have \( \varphi \rightarrow_{<} \psi \) or \( \psi \).

**Proof:** Let \( \psi \) be a minimal element in \( CF(\varphi) \). Since \( \varphi \not\rightarrow_{<} \) \( \varphi \), and \( \psi \in CF(\varphi) \), there must exist \( \psi' \) such that \( \varphi \rightarrow_{<} \psi' \) and \( CF(\psi') \subseteq CF(\varphi) \) or \( \psi' \equiv \psi \). If \( \psi' \not\equiv \psi \), then according toLemma 2, we know \( CF(\psi') \subseteq cl(\psi) \). However, \( cl(\psi) \cap CF(\varphi) = \{\psi\} \) because of the minimality of \( \psi \). Thus \( \psi' = \psi \).

**2) Soundness Proof of Theorem 2** We first introduce the relation \( \vdash \):

**Definition 11:** Let \( \eta = \omega_{0} \omega_{1} \ldots \omega_{n} (n \geq 0) \). Then, we say the finite sequence \( \eta \) satisfies the formula \( \varphi \), denoted by \( \eta \vdash \varphi \), iff there exists \( O \in Olg(\varphi) \) such that \( O \subseteq \eta \). Here \( O \subseteq \eta \) is an abbreviation for \( O \subseteq \cup_{i=0}^{n} \omega_{i} \).

Please note that the relation is defined by checking syntactic inclusion. Thus, assuming the input formula is \( alUa \), which is \( a_{1}Ua_{2} \) after tagging. According to the above definition, \( \{a_{1}\} \not\vdash_{f} a_{2} \). The reader shall bear this in mind in the remaining of this section.

Below we present some simple properties of the relation \( \vdash \) which will be useful later:

**Lemma 7:**
1. Assume \( \eta \vdash \varphi \) and \( \eta \vdash \varphi \land \psi \).
2. Assume \( \eta \vdash \varphi \) \land \( \eta \vdash \psi \) and \( \eta \vdash \varphi \lor \psi \).

**Proof:**
Let \( \eta = \omega_{0}\omega_{1} \ldots \omega_{n} \). We consider the first case: \( \eta \vdash \varphi \) implies that there exists \( O \in Olg(\varphi) \) such that \( O \subseteq \eta \). Since \( O \subseteq Olg(\varphi) \subseteq Olg(\varphi \lor \psi) \), we have \( \eta \vdash \varphi \lor \psi \). For the second case: \( \eta \vdash \varphi \lor \psi \) implies that there exists \( O_{1} \in Olg(\varphi) \) such that \( O_{1} \subseteq \eta \). Similarly, \( \eta \vdash \varphi \) implies that there exists \( O_{2} \in Olg(\psi) \) such that \( O_{2} \subseteq \eta \). Since we have \( O_{1} \cup O_{2} \subseteq Olg(\varphi \lor \psi) \) and \( O_{1} \cup O_{2} \subseteq \eta \), we have \( \eta \vdash \varphi \lor \psi \).

The following lemma corresponds to Lemma 5 for until formulas with respect to the finite satisfaction relation:

**Lemma 8:** Let \( \varphi = \varphi_{1}U\varphi_{2} \) and \( \varphi \rightarrow_{<} \). Then, \( \eta \not\vdash_{f} \varphi \).

**Proof:**
Let \( \eta = \omega_{0}\omega_{1} \ldots \omega_{n} \) and we rewrite \( \varphi \rightarrow_{f} \varphi \) as \( \rightarrow_{\omega_{0}} \psi_{1} \omega_{2} \rightarrow_{\omega_{2}} \ldots \rightarrow_{\omega_{n}} \). Note it is apparent that \( \rightarrow_{\omega_{i}} \varphi \) with \( \rightarrow_{\omega_{i}} \varphi \lor \varphi'_{i} \) such that \( \rightarrow_{\omega_{i}} \varphi \rightarrow_{\omega_{i}} \varphi'_{i} \). Thus, by induction one can show that along the path \( \rightarrow_{\omega_{0}} \varphi_{1} \rightarrow_{\omega_{2}} \ldots \rightarrow_{\omega_{n}} tt \) it holds \( \varphi \in CF(\varphi_{i}) \) for all \( i \). Since the transition for conjunctive formula is obtained by combining transitions for each \( \psi_{i} \in CF(\psi_{i}) \), the transition for the subformula \( \varphi \in CF(\varphi_{i}) \) must be from left subformula \( \varphi_{1} \), i.e., \( \rightarrow_{\omega_{0}} \psi_{1} \rightarrow_{\omega_{2}} \ldots \rightarrow_{\omega_{n}} \). As a result, for each \( i \), the label \( \omega_{i} \) must be a superset of \( CF(\alpha_{i}) \), and \( CF(\alpha_{i}) \) contains the literals from \( \varphi_{1} \). Moreover, the tagging function has been used to classify the atoms. According to Definition 6 the atoms in \( \varphi_{1} \) and \( \varphi_{2} \) are never tagged the same – this is because the atoms in \( \varphi_{2} \) will tag \( \varphi \) while those in \( \varphi_{1} \) will not. So \( \varphi_{1} \) and \( \varphi_{2} \) still don’t have common atoms, thus \( CF(\alpha_{i}) \) contains no obligation literals from \( \varphi_{2} \).

According to the definition 5 the obligation literals of \( \varphi \) are all from those of \( \varphi_{2} \). Thus, from the definition of \( \vdash \) (Definition 11), we know \( \eta \not\vdash_{f} \varphi \).
The following lemma says that if there exists a partitioning $\xi = \eta_1 \eta_2 \ldots$ that makes $\phi$ expanding to itself by each $\eta_i$ and $\eta_i \models f \phi$ holds, then $\xi \models f \phi$.

**Lemma 9:** Given a looping formula $\phi$ and a trace $\xi$, let $\xi = \eta_1 \eta_2 \ldots$ if $\forall i \geq 1 \cdot \phi \not\models \eta_i$ $\phi$, then $\xi \not\models f \phi$.

**Proof:** We enumerate the set $CF(\phi) = \{\phi_1, \phi_2, \ldots, \phi_n\}$, and we shall prove $\xi \not\models f \phi$. Let $S_0$ denote the minimal elements of $CF(\phi)$ with respect to the partial order $\prec$. Moreover, we define $S_{i+1} = S_i \cup \{\phi_i \in CF(\phi) \mid \exists \psi \in S_i, \psi \not\prec \phi_i\}$. Obviously, there is a finite index $k$ such that $S_k = CF(\phi)$. We proceed with induction over the index:

1) Basic step: Let $\psi \in S_0$. By Lemma 6 if $\phi \models \eta_i \phi$, then $\psi \models \eta_i \psi$ or $\psi \models \eta_i \phi$. If $\eta_i \psi \models \eta_i \psi$ holds, $\xi \models f \psi$ follows from Lemma 4. Otherwise we have $\psi \models \eta_i \psi$ for all $i \geq 1$. According to LTL semantics $\psi$ must be either until or release formula. Applying Lemma 8 we know that $\psi$ cannot be an Until formula, and thus be a Release formula. Then Lemma 5 implies that $\xi \models f \psi$, and therefore $\xi \models f S_{k+1}$.

2) For induction step we assume $\xi \models f S_k$. Consider arbitrary $\eta_i$: let $\psi \in S_{k+1} \setminus S_k$ and assume $\psi \models \eta_i \psi$. Since $\phi \models \eta_i \phi$, we have $CF(\eta_i \psi) \subseteq CF(\phi)$. By the construction of $S_k$, $CF(\psi)$ does not contain any other elements in $S_{k+1}$, thus we have $CF(\psi') \subseteq \{\psi'\} \cup S_k$. First we assume $CF(\psi') \subseteq S_k$. Then from the induction hypothesis we know $\eta_i \eta_{i+1} \ldots \models f \psi'$ so $\eta_i \eta_{i+1} \ldots \models f \psi$. (From Lemma 4, thus $\xi \models f \psi$. Now consider the case $\psi \in CF(\psi')$: $\phi \not\models \eta_i \phi$ implies that $\psi$ must be a Release formula. Then Lemma 5 implies again that $\xi \models f \psi$, and therefore $\xi \models f S_{k+1}$.

Now we are ready to prove the soundness part of the theorem:

**Lemma 10 (Soundness):** Let $\psi \in S_A$. Assume that there exists a SCC $B$ of $ST_{\phi}$ such that $\psi \in B$ and $L(B)$ is a superset of some obligation set $O \in Olg(\psi)$. Then, $SAT(\phi)$.

**Proof:** As $B$ is a SCC, we have a path $\delta := \psi(\psi_1)$ $\not\models \psi_2$, $\psi_2 \not\models \psi_3$, $\ldots$ $\not\models \psi_k$ such that $\eta$ visits all states in $B$ and all transitions between states in $B$ ($\eta = \omega_1, \omega_2, \ldots, \omega_{k-1}$). Since all states in $ST_{\phi}$ are reachable from $\varphi$, there must exist a finite sequences $\eta_0$ such that $\varphi \models \eta_0 \psi$. We construct $\xi := \eta_0 \eta_1 \ldots$. By assumption there exists $O \in Olg(\psi)$ such that $O \subseteq L(B) = \cup_{i=0}^{k-1} L(\omega_i)$. Since the definition of $f \varphi$, we have $\eta_0 \models f \psi$. Thus from Lemma 5 we have $\xi \models f \psi$. So $\xi$ is satisfiable.

3) Completeness Proof of Theorem 2

**Lemma 11:** $\xi \models f \phi \Rightarrow \exists \eta \cdot \xi \models f \phi$.

**Proof:** We prove it by structural induction over the formula $\phi$. For the base case assume $\phi$ is tt or a literal, $\xi_i \models f \phi$ by definition. Moreover, $\phi$ cannot be ff. Now we consider the induction step:

- If $\phi = X\psi$, then $\xi \models f \phi \Rightarrow \xi_1 \models f \psi$. By induction hypothesis we know $\exists \eta \cdot \xi_1 \models f \psi$ holds, so $\xi_1 \models f \phi$ holds.

- If $\phi = \phi_1 \lor \phi_2$, then $\xi \models f \phi_1 \lor \phi_2$. By induction hypothesis we have $\exists \eta \cdot \xi_1 \models f \phi_1$ and $\exists \eta \cdot \xi_1 \models f \phi_2$ holds. Observe that $\xi_1 \models f \phi$ implies $\xi_1 \models f \phi$ for all $m \geq n$. Now from Lemma 7 we have that $\xi_m \models f \phi$ with $n := \max(n_1, n_2)$.

- If $\phi = \phi_1 \land \phi_2$, then $\xi \models f \phi_1 \land \phi_2$. By induction hypothesis we have $\exists \eta \cdot \xi_1 \models f \phi_1$ or $\exists \eta \cdot \xi_1 \models f \phi_2$ holds. Without loss of generality, assume $\exists \eta \cdot \xi_1 \models f \phi_1$, Lemma 13 implies then $\xi_1 \models f \phi_1 \lor \phi_2$.

- If $\phi = \phi_1 U \phi_2$, $\xi \models f \phi_1 U \phi_2$ implies that there exists $i \geq 0$ such that $\xi_i \models \phi_2$. By induction hypothesis we have $\exists \eta \cdot \xi_i \models f \phi_2$ holds, thus there exists an obligation $O \in Olg(\phi_2)$ such that $O \subseteq \xi_i$. This implies $O \subseteq \xi^{i+n}$, thus $\xi^{i+n+1} \models f \phi$ holds.

- If $\phi = \phi_1 R \phi_2$, we observe first that $\xi \models f \phi_2$ must hold. By induction hypothesis we know $\exists \eta \cdot \xi_n \models f \phi_2$. According to Definition 11 we have that $\xi_n \models f \phi$ holds as well.

**Lemma 12:** Let $\phi$ be a looping formula, and assume $\phi \not\models f \phi$ and $\xi \models f \phi$. Then there exists a partitioning $\xi = \eta_1 \eta_2 \ldots$ and $\forall i \geq 0 \cdot \phi \models \eta_i \phi \land \eta_i \models f \phi$.

**Proof:** Assume $\phi \not\models f \phi \land \xi \models f \phi$. We first prove that there exist $n$ such that $\phi \models \eta_n \phi \land \phi \models f \phi$. From Lemma 11 $\xi \models f \phi$ implies that there exists $k$ such that $\xi_k \models f \phi$. Since $\phi$ is a looping formula with respect to $\xi$, we can find the $n \geq k$ such that $\phi \models \phi \land \phi \models f \phi$. For $n \geq k$, $\xi_n \models f \phi$ holds. Now we apply Lemma 11 $\xi \models f \phi$ implies that $\phi \models f \phi$ and $\xi_n \models f \phi$.

Since $\xi_n \models f \phi \land \xi_n \models f \phi$, applying the arguments above inductively yields the lemma.

The above lemma states that if $\phi \not\models f \phi$ as well $\xi \models f \phi$, we can find a partitioning $\eta_1 \eta_2 \ldots$ that makes $\phi$ expand to itself by each $\eta_i$ and $\eta_i \models f \phi$ holds. Combining Lemma 6, Lemma 7 and Corollary 1, we have our central theorem:

**Lemma 13 (Completeness):** Let $\phi \in S_A$. Then, $SAT(\phi)$ implies that there exists a SCC $B$ of $ST_{\phi}$ and such that $\psi \in B$ and $L(B)$ is a superset of some obligation set $O \in Olg(\psi)$.

**Proof:** Since $SAT(\phi)$, let $\xi$ such that $\xi \models f \phi$. Then by Corollary 2 there exists $n \geq 0$ and $\psi \in S_A$ such that $\phi \models \psi \not\models f \psi$, $\psi \models f \psi$ and $\xi_n \models f \phi$. Moreover, by Lemma 12 there exists a partition $\xi_n = \eta_1 \eta_2 \ldots$ such that for every finite sequence $\eta_i$ we have $\psi \models f \psi$ as well as $\eta_i \models f \phi$. As the state $\psi$ is visited infinitely often, there must be a SCC $B$ such that $\psi \in B$. According to the definition of $f \phi$, we know that there exists $O \in Olg(\psi)$ such that $O \subseteq \eta_i$. Obviously, $\psi \models f \psi$ is contained in some SCC $B$, thus $\eta \subseteq L(B)$, implying $O \subseteq L(B)$. 
