LTL Satisfiability Checking Revisited

Jianwen Li*, Lijun Zhang†, Geguang Pu*, Moshe Y. Vardi‡ and Jifeng He*
*Software Engineering, East China Normal University
†DTU Informatics, Technical University of Denmark
‡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 φ over a set $Prop$ of atomic propositions. If a model M is *universal*, that is, it contains all possible traces over $Prop$, then φ 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 check 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.

Rozier and Vardi reached two major conclusions. First, most LTL translation tools are research prototypes and cannot be considered industrial quality tools. Among all the tools tested, only SPOT can be considered an industrial quality tool. Second, when it comes to LTL satisfiability checking, the symbolic approach is clearly superior to the explicit approach. Even SPOT, the best LTL translator in our experiments, was rarely able to compete effectively against the symbolic tools.

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 detail our experimental methodology. We describe our experimental results in Section V. Section VI discusses related work and Section VII concludes the paper.

II. Preliminaries

A. Linear Temporal Logic

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

$$\varphi \ ::= \ \mathtt{tt} \mid \mathtt{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi U \varphi \mid \varphi R \ \varphi \mid X\varphi$$

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

We say φ is a propositional formula if it does not contain temporal operators X, U or R. We say φ 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, α, β, γ to denote propositional formulas, and φ and ψ 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 Σ^ω. For ξ and $k \geq 1$ we use $\xi^k = \omega_0\omega_1 \ldots \omega_{k-1}$ to denote the prefix of ξ up to its k-th element, and $\xi_k = \omega_k\omega_{k+1} \ldots$ to denote the suffix of ξ from its $(k+1)$-th element. Thus, $\xi = \xi^k\xi_k$. We use $\eta, \eta_0 \ldots$ to denote finite sequences in Σ^*. 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 $\bigwedge a \not\equiv \mathtt{ff}$. A trace $\xi = \omega_0\omega_1 \ldots$ is consistent iff ω_i is consistent for all i.

Let $\omega \in \Sigma$ be a consistent set of literal, and α a propositional formula. We define $\omega \models \alpha$ in the standard way: if α is a literal then $\omega \models \alpha$ iff $\alpha \in \omega$, $\omega \models \alpha_1 \wedge \alpha_2$ iff $\omega \models \alpha_1$ and $\omega \models \alpha_2$, and $\omega \models \alpha_1 \vee \alpha_2$ iff $\omega \models \alpha_1$ or $\omega \models \alpha_2$. Moreover, $\omega \models \mathtt{tt}$ and $\omega \not\models \mathtt{ff}$.

The semantics of temporal operators with respect to a consistent trace ξ is given by:

- $\xi \models \alpha$ iff $\xi^1 \models \alpha$,
- $\xi \models X \ \varphi$ iff $\xi_1 \models \varphi$.
- $\xi \models \varphi_1 \ U \ \varphi_2$ iff there exists $i \geqslant 0$ such that $\xi_i \models \varphi_2$ and for all $0 \leqslant j < i, \xi_j \models \varphi_1$;
- $\xi \models \varphi_1 \ R \ \varphi_2$ iff either $\xi_i \models \varphi_2$ for all $i \geq 0$, or there exists $i \geq 0$ with $\xi_i \models \varphi_1 \wedge \varphi_2$ and $\xi_j \models \varphi_2$ for all $0 \leq j < i$;

According to the semantics, it holds $\varphi R\psi = \neg(\neg\varphi U \neg\varphi)$. We use the usual abbreviations $\mathtt{tt} = a \vee \neg a$, $Fa = \mathtt{tt}Ua$ and $Ga = \mathtt{ff}Ra$.

Definition 2 (Satisfiability): We say φ is satisfiable, denoted by $SAT(\varphi)$, if there exists a consistent trace ξ such that $\xi \models \varphi$.

In the remaining 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 φ, we use $cl(\varphi)$ to denote the set of subformulas of φ. We denote by AP_φ the set of atoms appearing in φ, by L_φ the set of literals over AP_φ, and by Σ_φ the set of consistent literal sets over AP_φ.
- Let $\varphi = \bigwedge_{i\in I} \varphi_i$ such that the root operator of φ_i is not a conjunctive. We define the set of conjuncts of φ is $CF(\varphi) := \{\varphi_i \mid i \in I\}$. When φ does not include a conjunctive as a root operator, $CF(\varphi)$ includes only φ itself. The set of disjuncts $DF(\varphi)$ is defined in an analogous way.
- For a formula φ of the form $\varphi_1 U \varphi_2$ or $\varphi_1 R \varphi_2$, let $left(\varphi)$ ($right(\varphi)$) be left (right) subformulas of φ.
- For all propositional formula α appearing in the paper, we always check first whether α is satisfiable. If not, we shall replace α by $\mathtt{ff}$.

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 φ, denoted as $NF(\varphi)$, is a set defined as follows:

1) $NF(\varphi) = \{\varphi \wedge X(\mathtt{tt})\}$ if $\varphi \not\equiv \mathtt{ff}$ is a propositional formula. If $\varphi \equiv \mathtt{ff}$, we define $NF(\mathtt{ff}) = \emptyset$;
2) $NF(X\varphi) = \{\mathtt{tt} \wedge X(\psi) \mid \psi \in DF(\varphi)\}$;
3) $NF(\varphi_1 U \varphi_2) = NF(\varphi_2) \cup NF(\varphi_1 \wedge X(\varphi_1 U \varphi_2))$;
4) $NF(\varphi_1 R \varphi_2) = NF(\varphi_1 \wedge \varphi_2) \cup NF(\varphi_2 \wedge X(\varphi_1 R \varphi_2))$;
5) $NF(\varphi_1 \vee \varphi_2) = NF(\varphi_1) \cup NF(\varphi_2)$;
6) $NF(\varphi_1 \wedge \varphi_2) = \{(\alpha_1 \wedge \alpha_2) \wedge X(\psi_1 \wedge \psi_2) \mid \forall i = 1, 2.\ \alpha_i \wedge X(\psi_i) \in NF(\varphi_i)\}$;

From the definition it is obvious that if $\alpha \wedge X(\psi) \in NF(\varphi)$, then α is a conjunctive clause, namely a conjunction of literals.

For a formula φ, our algorithm will detect sucessor formulas based on the set $NF(\varphi)$. First, we shall show that the formula φ is logically equivalent to $\bigvee NF(\varphi)$, here $\bigvee NF(\varphi)$ represents the formula $\bigvee_{1\leq j\leq k}(\alpha_j \wedge X\varphi_j)$ with $\alpha_j \wedge X\varphi_j \in NF(\varphi)$ and $k = |NF(\varphi)|$. We note that the empty disjunction (OR-ing over an empty set of operands) is defined as $\mathtt{ff}$. Then, we establish the equivalence property:

Lemma 1: For the formula φ, it holds $\varphi \equiv \bigvee NF(\varphi)$.

The lemma below states that along the expansion the set of subformulas is decreasing:

Lemma 2: If $\alpha \wedge X\psi \in NF(\varphi)$, then $CF(\psi) \subseteq cl(\varphi) \cup \{\mathsf{tt}\}$.

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*. 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 $\bigvee_{1\leq i\leq n} a_iUb_i$ the obligation set is $\{\{b_1\},\{b_2\},\ldots,\{b_n\}\}$;
- for the formula $\bigwedge_{1\leq i\leq n} a_iUb_i$ the obligation set is $\{\{b_1,b_2,\ldots,b_n\}\}$.

For a formula φ, if one of its obligation $O \subseteq 2^L$ is consistent, i.e. $\bigwedge_{a\in O} a \not\equiv \mathsf{ff}$, 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_φ for φ *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 show that the formula is satisfiable if an accepting 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 (detailed later) 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 φ:

Definition 4 (Obligation Set): For a formula φ, we define its obligation set, denoted by $Olg(\varphi)$, as follows:

1) $Olg(\mathsf{tt}) = \{\emptyset\}$ and $Olg(\mathsf{ff}) = \{\{\mathsf{ff}\}\}$;
2) If φ is a literal, $Olg(\varphi) = \{\{\varphi\}\}$;
3) If $\varphi = X\psi$, $Olg(\varphi) = Olg(\psi)$;
4) If $\varphi = \psi_1 \vee \psi_2$, $Olg(\varphi) = Olg(\psi_1) \cup Olg(\psi_2)$;
5) If $\varphi = \psi_1 \wedge \psi_2$, $Olg(\varphi) = \{O_1 \cup O_2 \mid O_1 \in Olg(\psi_1) \wedge O_2 \in Olg(\psi_2)\}$;
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 φ. Moreover, we say O is a consistent obligation iff $\bigwedge a \not\equiv \mathsf{ff}$ holds, where $a \in O$.

The obligation set $Olg(\varphi)$ enumerates all obligations the given formula φ 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 $\{\mathsf{ff}\}$ 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 φ. 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^ω satisfies $G(aRb)$.
- Consider the formula $\varphi := GF(a \wedge b) \wedge F(\neg a)$: first, the obligation $\{a, b, \neg a\}$ is not consistent. Further, the normal form $NF(\varphi)$ contains $\neg a \wedge X(GF(a \wedge b)))$. Thus we can reach the formula $GF(a \wedge b)$ along $\neg a$. Moreover, $GF(a \wedge b)$ has a consistent obligation set $O = \{a, b\}$. Theorem 1 then provides a trace ξ with: $\xi := \neg a O^\omega \models \varphi$.
- The opposite direction of Theorem 1 does not hold. Consider for example the formula $F(a) \wedge G(X\neg a)$. It has a single obligation $\{a, \neg a\}$ which is not consistent. However, $a(\neg a)^\omega$ is a satisfying trace. Consider another formula $F(a) \wedge G(\neg a)$ 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. This can be problematic as the left subformula does play a role in our construction. Thus, we need first *tag* the input formula in our approach such that they can be differentiated.

For an given input formula φ under consideration. For each atom a appearing in φ, we enumerate all occurrences of a by $S_a := \{a_1, a_2, \ldots, a_n\}$, provided a appears n times in φ. The most 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, thus, this tagging is inefficient. Below we give an improved tagging function.

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

Definition 5 (Tagging Formula): Let $a \in AP$ be an atom appearing in φ. Then, the tagging function $F_a : S_a \to 2^{U(\varphi)}$ is defined as: $\psi \in F_a(a_i)$ iff a_i appears in $right(\psi)$.

We define the *tagged formula* φ_t as the formula obtained by replacing a_i by $a_{F_a(a_i)}$ for each $a_i \in S_a$.

Thus, after tagging AP_{φ_t} will contain more atoms. Note that all these new copies are semantically equivalent to a, i.e., $a_{F_a(a_i)} \equiv a$ for all $a_{F_a(a_i)}$. 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, $a_1 = a_2 \Leftrightarrow F_a(a_1) = F_a(a_2)$.

As an example, consider $\varphi = aU(a \wedge aU\neg a)$. Let $\psi_u = aU\neg a$, and $S_a = \{a_1, a_2, a_3, a_4\}$. From Definition 5 we know $F_a(a_1) = \emptyset, F_a(a_2) = F_a(a_3) = \{\varphi\}$, and $F_a(a_4) = \{\varphi, \varphi_u\}$. So the tagging function will introduce three syntactically different copies of a, and we denote φ_t by $a_1U(a_2 \wedge a_2U\neg a_4)$. Here even a_1, a_2, a_4 are syntactically different, they are semantically equivalent. Thus it holds for example $a_2 \wedge \neg a_4 \equiv \mathtt{ff}$.

Note the size of subformulas may increase after tagging. According to Definition 5, the following lemma is obvious:

Lemma 3 (Tagging Cost): Let φ be the input formula and φ_t the formula obtained after tagging φ. 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 φ, it holds $\varphi \equiv \varphi_t$. This implies $SAT(\varphi)$ iff $SAT(\varphi_t)$. As our approach will work with the tagged formula φ_t, in the remaining of the paper:

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

For a given formula φ, we shall define below a labelled transition system T_φ for it:

Definition 6 (LTL Transition System): Let φ be the input formula and φ_t the tagged formula. The labeled transition system T_φ is a tuple $\langle Act, S_\varphi, \to, \varphi_t\rangle$ where:

1) φ_t is the initial state,
2) Act is the set of conjunctive formulas over L_{φ_t}.
3) the transition relation $\to \subseteq S_\varphi \times Act \times S_\varphi$ is defined by: $\psi_1 \xrightarrow{\alpha} \psi_2$ iff there exists $\alpha \wedge X(\psi_2) \in NF(\psi_1)$;
4) S_φ 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 φ is defined by starting from the *tagged* formula φ_t. The set of states is the set of formulas reachable from φ_t, with φ_t as the initial state. Note between two states there can be more transitions. A state φ has no outgoing transitions iff for all $\alpha \wedge X\psi \in NF(\varphi)$ and α is equivalent to ff. In this case φ is not satisfiable. Now we introduce the notion of accepting traces:

Definition 7: A *run* of T_φ is a (finite or infinite) path $r = \varphi \xrightarrow{\alpha_0} \psi_1 \xrightarrow{\alpha_1} \psi_2 \xrightarrow{\alpha_2} \dots$ in T_φ. A trace $\xi = \omega_0\omega_1 \dots \in \Sigma^\omega$ is accepted by the run r if $\omega_i \models \alpha_i$ for all i.

For $\omega \in \Sigma$, we write $\varphi \xrightarrow{\omega} \psi$ if there exists $\varphi \xrightarrow{\alpha} \psi$ such that $\omega \models \alpha$. For a finite sequence $\eta = \omega_0\omega_1..\omega_k$, we write $\varphi \xrightarrow{\eta} \psi$ iff $\varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_1} \psi_2 \xrightarrow{\omega_2} \dots \xrightarrow{\omega_k} \psi_{k+1} = \psi$. More specially, we write $\varphi \xrightarrow{\xi} \varphi$ iff ξ can be written as $\xi = \eta_0\eta_1\eta_2 \dots$ such that $\varphi \xrightarrow{\eta_i} \varphi$ for all $i \geq 0$.

Lemma 2 implies the following properties of $|S_\varphi|$:

Corollray 1: For any formula φ, it holds:

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

D. On-the-fly Satisfiability Algorithm

First, we introduce some notations:

- For notational convenience, we fix λ as our input formula in this section. Let T_λ be the transition system for the tagged formula λ_t.
- For all $\varphi \in S_\lambda$, we denote by ST_φ the subsystem of T_λ consisting all states reachable from ψ.
- For a strongly connected component (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.

Now we present our main theorem:

Theorem 2: Let $\varphi \in S_\lambda$. Then, $SAT(\varphi)$ iff there exists a SCC B of ST_φ and a state ψ 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 $\xi = \omega_0\omega_1 \dots$ be a (consistent) trace such that $\xi \models \varphi$. Then, there is a run in ST_φ accepting ξ, i.e., we have $\varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_1} \dots$. After some prefix ξ^n, since there are only finitely many states reachable, we will be able to partition the suffix into $\eta_1\eta_2 \dots$ where all η_i are finite sequences, and all η_i lead from ψ to ψ itself. Such formula ψ will be referred to as a *looping formula*.

Looping formulas arising from U and R operators must be treated differently. For instance $aRb \xrightarrow{b} aRb$ resolves the obligation $\{b\}$, however $aUb \xrightarrow{a} aUb$ 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 side 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, thus any run in the transition system induces a *consistent* trace, which can be proven to satisfy the formula iff

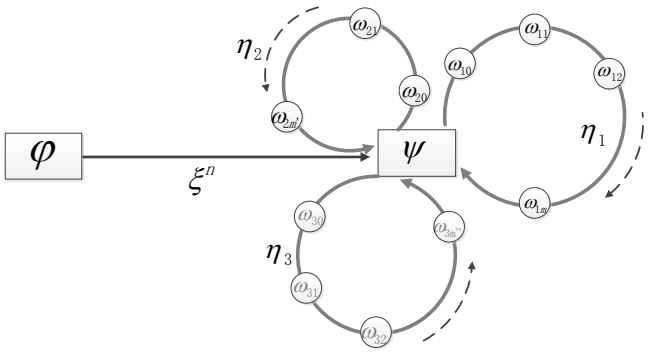

Fig. 1. A snapshot illustrating the relation $\xi \models \varphi$

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 $O \subseteq L(B)$. ■

Example 2: Consider $\varphi := (a \vee b)U(Ga)$, 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 φ.

On the other side, our algorithm first tags the formula to $\varphi_t = (a_1 \vee b)U(Ga_2)$. 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 the infinite path through SCC $\varphi \xrightarrow{a_1} \varphi \xrightarrow{b} \varphi$ can not satisfy φ.

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

1) We first tag the formula and get λ_t. Then we construct T_λ, where we explore the states in an on-the-fly manner, by performing for instance standard search method [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 $O \in Olg(\varphi)$, we return true,
4) If all SCCs are explored, and all do not have the property in step 3, we return false.

We remark that the worst case senario happens if all extended formulas do not contain any consistent sets, which happens for instance for the formula $GF(a) \wedge GF(\neg a)$.

IV. Experimental Methodology

We have implemented our algorithms in a tool called *Aalta*[1]. 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:

- OF: On-the-fly checking without obligation acceleration,
- OFOA: On-the-fly checking with obligation acceleration (default)

A. Testing tools

We have implemented our algorithms in a tool called *Aalta*, whose default is on-the-fly plus obligation acceleration (OFOA). In this paper we compare the performance of *Aalta* with two other LTL satisfiability solvers: PANDA+CadenceSMV [RV11] and SPOT [DLP04]. SPOT is considered to be the best explicit LTL-to-Büchi translator [RV07], [RV10]. Its most recent version (1.0.2) has an integrated the emptiness checking implementation (with "-e" flag) and is 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.)

B. Platform

We conducted our benchmarking on the SUG@R cluster in Rice University[2]. 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. Are time measurements are "end-to-end"; we measure the time starting from formula input to the satisfiability-checking result (SAT or UNSAT).

C. 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].

1) Random Conjunction Formulas: Formally, a random conjunction formula $RC(n)$ has the form: $RC(n) = \bigwedge_{1 \le i \le n} P_i(v_1, v_2, \ldots, v_k)$, where n is the number of conjuncts elements and $P_i (1 \le i \le m)$ 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]Tool to be released upon publication.

[2]http://www.rcsg.rice.edu/sugar/

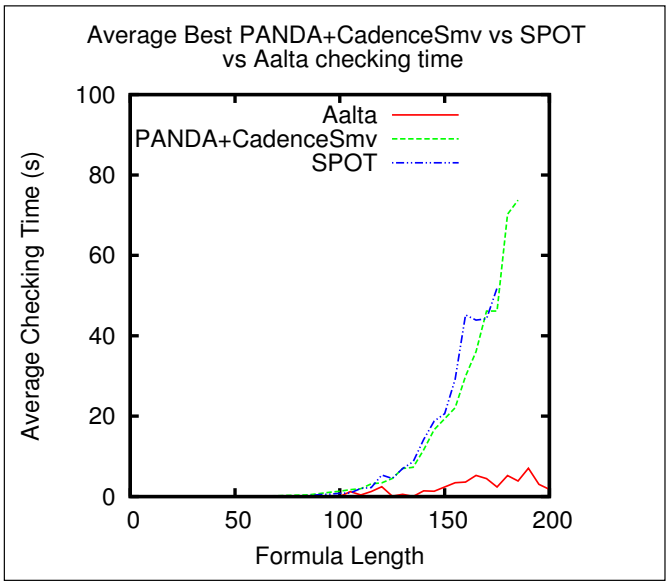

Fig. 2. Experimental results for random formulas with 3 variables.

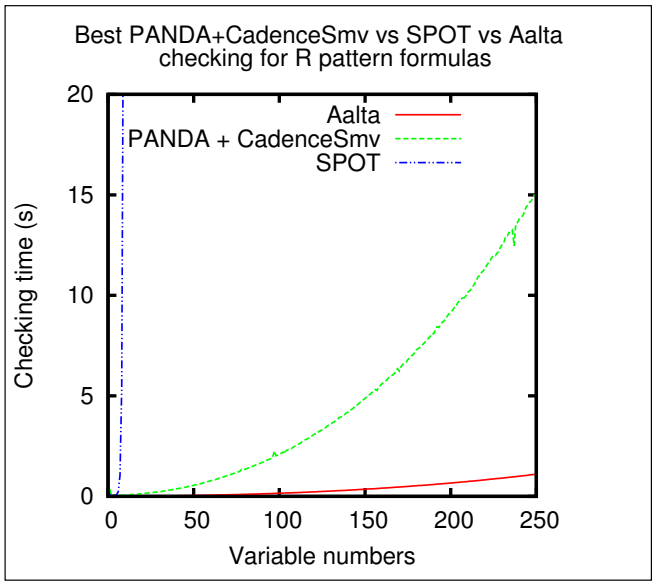

Fig. 3. Experimental results for $R(n) = \bigwedge_{i=1}^{n}(GFp_i \vee FGp_{i+1})$.

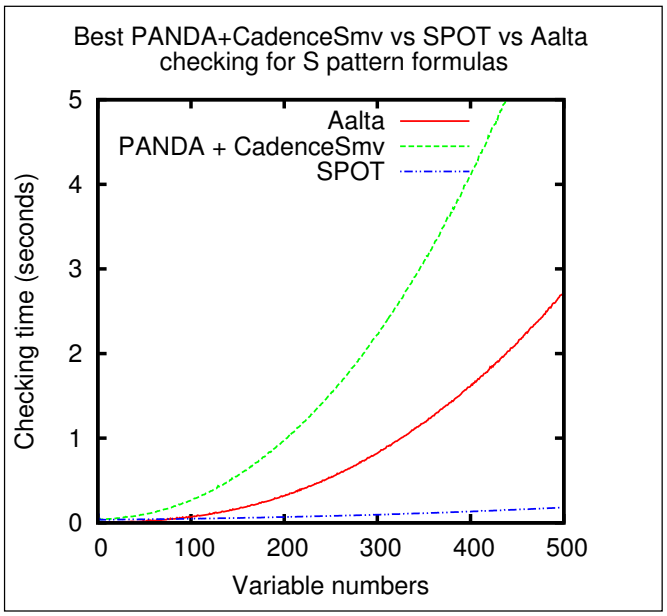

Fig. 4. Experimental results for $S(n) = \bigwedge_{1 \leq i \leq n} Gp_i$.

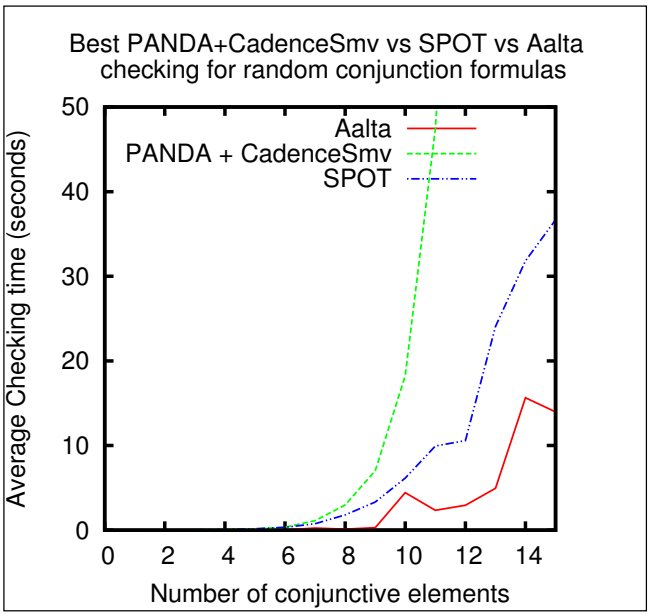

Fig. 5. Experimental results for random conjunctive formulas.

1) We extract all pattern formulas[3].
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.

D. 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.

[3]http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml

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

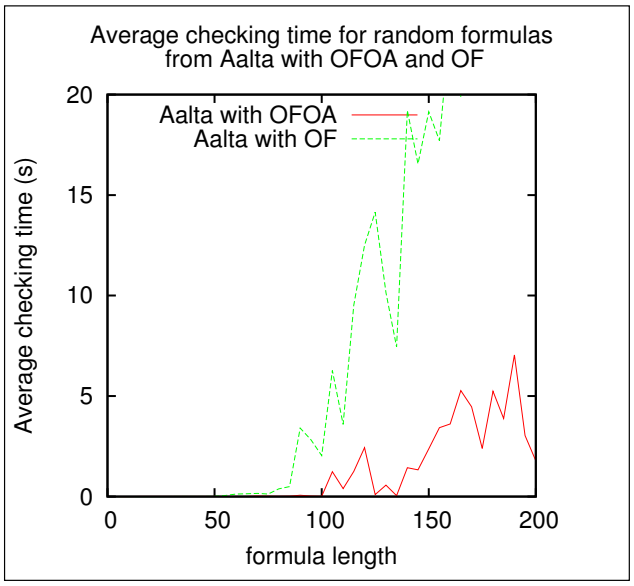

Fig. 6. Experimental results for 3-variable random formulas from *Aalta* with OFOA and OF.

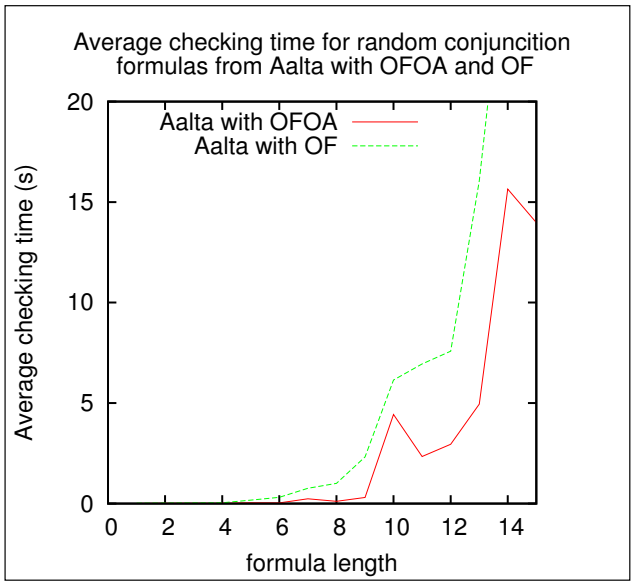

Fig. 7. Experimental results for random conjunction formulas from *Aalta* with OFOA and OF.

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 20,00 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*, but, overall, *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* is 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 textitobligation 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'd be interesting to combine *Aalta* with the abstraction technique of [CRST07].

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

One of the effective heuristic of *Aalta* is the OF technique. For many formulas, a consistent obligation implies satisfiability directly. We explore its performance in satisfiability checking by the following experimental results:

1) Fig. 6 and Fig. 7 indicate that, the *obligation acceleration* technique also plays a key role in checking random and random conjunction formulas. We compare here the results from *Aalta* implemented with the OFOA and pure OF strategies. For random formulas the OFOA strategy performs at least five times faster than the pure OF strategy. Moreover, the OFOA strategy can be even exponential 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 performance for the explicit-state approach, but symbolic tools such as CadenceSMV [McM99] or NuSMV [CCG+02] yield better performance. In follow-up work [RV11], Rozier and Vardi studied several symbolic encodings of automata for LTL formulas and described a tool, PANDA, built on top of CadenceSMV, which implements a portfolio approach, running many symbolic encodings in parallel and selecting always the best performing one.

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 [GKS10] 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].

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, similarly 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 is 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 state-of-the-art tools.

References

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 25th Int. Colloq. on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1989.

[BAC98] M. Dwyer B, G.S. Avrunin, and J.C. Corbett. Property specification patterns for finite-state verification. In *Proc. 2nd workshop on Formal methods in software practice*, pages 7–15. ACM, 1998.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[CCG+02] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proc. 14th Int'l Conf. on Computer Aided Verification*, Lecture Notes in Computer Science 2404, pages 359–364. Springer, 2002.

[CCGR00] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *It'l J. on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[CRST07] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean abstraction for temporal logic satisfiability. In *Proc. 15th Int'l Conf. on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 532–546. Springer, 2007.

[CVWY92] 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.

[DGV99] N. Daniele, F. Guinchiglia, and M.Y. Vardi. Improved automata generation for linear temporal logic. In *Proc. 11th Int. Conf. on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 1999.

[DLP04] A. Duret-Lutz and Denis Poitrenaud. SPOT: An extensible model checking library using transition-based generalized büchi automata. In *Proc. 12th Int'l Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 76–83. IEEE Computer Society, 2004.

[FDP01] M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001.

[FKL04] H.D. Foster, A. Krolnik, and D.J. Lacey. *Assertion-Based Design*. Springer, 2004.

[GKS10] Valentin Goranko, Angelo Kyrilov, and Dmitry Shkatov. Tableau tool for testing satisfiability in ltl: Implementation and experimental analysis. *Electr. Notes Theor. Comput. Sci.*, 262:113–125, 2010.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[McM99] K. McMillan. The SMV language. Technical report, Cadence Berkeley Lab, 1999.

[PSC+06] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *Proc. 43rd Design Automation Conference*, pages 821–826. ACM, 2006.

[RV07] K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. In *Proc. 14th International SPIN Workshop*, volume 4595 of *Lecture Notes in Computer Science*, pages 149–167. Springer, 2007.

[RV10] K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. *Int'l J. on Software Tools for Technology Transfer*, 12(2):1230–137, 2010.

[RV11] K.Y. Rozier and M.Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *Proc. 17th Int'l Symp. on Formal Methods*, volume 6664 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 2011.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.

[Sch98] Stefan Schwendimann. A new one-pass tableau calculus for pltl. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, TABLEAUX '98, pages 277–292. Springer-Verlag, 1998.

[SD11] Viktor Schuppan and Luthfi Darmawan. Evaluating ltl satisfiability solvers. In *Proceedings of the 9th international conference on Automated technology for verification and analysis*, AVTA'11, pages 397–413. Springer-Verlag, 2011.

[Var07] M.Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 2007.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to

automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[Wol85] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 110–111:119–136, 1985.

APPENDIX

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

A. Proofs of Lemma 1

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 $\varphi = \varphi_1 \vee \varphi_2$, then applying induction hypothesis we have $\varphi \equiv \varphi_1 \vee \varphi_2 \equiv \bigvee NF(\varphi_1) \vee \bigvee NF(\varphi_2) \equiv \bigvee(NF(\varphi_1) \cup NF(\varphi_2)) \equiv \bigvee NF(\varphi_1 \vee \varphi_2)$.
- If $\varphi = \varphi_1 \wedge \varphi_2$, then applying induction hypothesis we have $\varphi \equiv \varphi_1 \wedge \varphi_2 \equiv (\bigvee NF(\varphi_1)) \wedge (\bigvee NF(\varphi_2))$. By inspection, this is equivalent to $\bigvee NF(\varphi_1 \wedge \varphi_2)$.
- If $\varphi = \varphi_1 U \varphi_2$, then by Definition 3 we know $NF(\varphi) = NF(\varphi_2) \cup NF(\varphi_1 \wedge X\varphi)$, and that $\varphi \equiv \varphi_2 \vee (\varphi_1 \wedge X\varphi)$. By induction hypothesis we have that $\varphi_2 \equiv \bigvee NF(\varphi_2)$. Moreover, we also proved previously that $\varphi_1 \wedge X\varphi \equiv \bigvee NF(\varphi_1 \wedge X\varphi)$. Thus we can prove that $\varphi \equiv \varphi_2 \vee (\varphi_1 \wedge X\varphi) \equiv \bigvee NF(\varphi_2) \vee \bigvee NF(\varphi_1 \wedge X\varphi) \equiv \bigvee NF(\varphi)$;
- The case $\varphi = \varphi_1 R \varphi_2$ is similar to the case when φ is a Until formula.

■

B. Proof of Lemma 2

Proof: First, $CF(\psi) \subseteq cl(\varphi)$ by structural induction over φ. The base cases $\varphi = \mathtt{tt}, \mathtt{ff}$ and propositional formulas are trivial. Otherwise:

1) If $\varphi = \varphi_1 \vee \varphi_2$. Then $NF(\varphi) = NF(\varphi_1) \cup NF(\varphi_2)$. So $\alpha \wedge X\psi \in NF(\varphi_i)$ with $i = 1$ or $i = 2$. By induction hypothesis we have $CF(\psi) \subseteq cl(\varphi_i) \subseteq cl(\varphi)$.
2) If $\varphi = X\varphi_1$. In this case we have $NF(\varphi) = \{\mathtt{tt} \wedge X\varphi' \mid \varphi' \subseteq DF(\varphi_1)\}$. Since $CF(\varphi') \subseteq cl(\varphi_1) \subseteq cl(\varphi)$, so we have $CF(\psi) \subseteq cl(\varphi)$.
3) If $\varphi = \varphi_1 \wedge \varphi_2$, then we know for every $\alpha \wedge X\psi \in NF(\varphi)$ there exists $\alpha_1 \wedge X\psi_1 \in NF(\varphi_1)$ and $\alpha_2 \wedge X\psi_2 \in NF(\varphi_2)$ such that $\alpha = \alpha_1 \wedge \alpha_2$ and $\psi = \psi_1 \wedge \psi_2$. Since by induction hypothesis we know $CF(\psi_1) \subseteq cl(\varphi_1)$ and $CF(\psi_2) \subseteq cl(\varphi_2)$, so $CF(\psi) \subseteq cl(\varphi)$ holds.
4) If $\varphi = \varphi_1 U \varphi_2$. We have two cases. Either we have the right expansion $\alpha \wedge X\psi \in NF(\varphi_2)$, in which case $CF(\psi) \subseteq cl(\varphi_2) \subseteq cl(\varphi)$ follows directly by induction hypothesis. For the left expansion case, we have $\alpha \wedge X\psi \in NF(\varphi_1 \wedge X\varphi)$, implying that there exists $\alpha_1 \wedge X\psi_1 \in NF(\varphi_1)$ such that $\psi = \psi_1 \wedge \varphi$. So $CF(\psi) \subseteq cl(\varphi)$ follows by exploiting the induction hypothesis that $CF(\psi_1) \subseteq cl(\varphi_1) \subseteq cl(\varphi)$.
5) If $\varphi = \varphi_1 R \varphi_2$, then we can prove similarly as the case when $\varphi = \varphi_1 U \varphi_2$.

■

C. Proof of Theorem 1

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

- If $\varphi = X\psi$ then we have $O \in Olg(\varphi) = Olg(\psi)$. By inductive hypothesis we have $O^\omega \models \psi$. Thus, $O^\omega \models \varphi$ as well;
- If $\varphi = \varphi_1 U \varphi_2$ then we have $O \in Olg(\varphi_2)$, according to the definition of obligation set. By inductive hypothesis we have $O^\omega \models \varphi_2$. Moreover according to LTL semantics we know $O^\omega \models \varphi$ as well;
- The case $\varphi = \varphi_1 R \varphi_2$ is similar to previous case;
- If $\varphi = \varphi_1 \vee \varphi_2$, we have $O \in Olg(\varphi_1)$ or $O \in Olg(\varphi_2)$. Assume $O \in Olg(\varphi_2)$ without loss of generality. By inductive hypothesis we have $O^\omega \models \varphi_2$, implying $O^\omega \models \varphi$ as well;
- If $\varphi = \varphi_1 \wedge \varphi_2$ then there exist $O_1 \in Olg(\varphi_1)$ and $O_2 \in Olg(\varphi_2)$ such that $O = O_1 \cup O_2$. Since O is consistent so both O_1 and O_2 must be consistent. By inductive hypothesis we have $O_1^\omega \models \varphi_1$ and $O_2^\omega \models \varphi_2$. Again since O is consistent have $O^\omega \models \varphi_1 \wedge \varphi_2$.

■

D. Proof of Corollary 1

Proof: The first clause follows by a simple induction over the path from φ_t to ψ. Note the constant 1 is due to the possibility of producing $\mathtt{tt}$ 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, T_λ is the transition system for λ.
- all traces are over $\Sigma^\omega \subseteq 2^{L_{\lambda_t}}$, i.e., the set of consistent literals over L_{λ_t}.
- all formulas appearing in this section are taken from the set of states S_λ, i.e., $\varphi, \psi ... \in S_\lambda$. Thus, all formulas in this appendix will be ranging over tagged atoms appearing in λ_t.

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

Lemma 4: Let $\xi \in \Sigma^\omega$ be a trace. Then, for all $n \geq 1$, there exists ψ such that $\xi \models \varphi \Leftrightarrow \varphi \xrightarrow{\xi^n} \psi \wedge \xi_n \models \psi$.

Proof: Let $\xi = \omega_0 \omega_1 \ldots$. We prove the lemma by induction over the number n.

- For the base case we let $n = 1$. Then: $\xi \models \varphi \Leftrightarrow \xi \models \bigvee NF(\varphi) \Leftrightarrow \exists \alpha \wedge X\psi \in NF(\varphi) \cdot \xi \models (\alpha \wedge X\psi) \Leftrightarrow \exists \alpha \wedge X\psi \in NF(\varphi) \cdot \xi_1 \models \psi \wedge \omega_0 \models \alpha \Leftrightarrow \exists \psi \cdot \varphi \xrightarrow{\xi^1} \psi \wedge \xi_1 \models \psi$.
- For the induction step, we assume the lemma holds for all $n = 1, 2, \ldots, k$ and prove that it holds for $n = k+1$ as well. Applying the induction hypothesis on k, we have: $\xi \models \varphi \Leftrightarrow \varphi \xrightarrow{\xi^k} \psi \wedge \xi_k \models \psi$ holds. Further, for $\xi_k \models \psi$ we apply the induction hypothesis with respect to the base case and obtain $\xi_k \models \psi \Leftrightarrow \psi \xrightarrow{\xi_k^1} \psi' \wedge \xi_{k+1} \models \psi'$, so we can conclude that $\xi \models \varphi \Leftrightarrow \varphi \xrightarrow{\xi^{k+1}} \psi' \wedge \xi_{k+1} \models \psi'$. The proof is done.

■

Essentially, $\xi \models \varphi$ is equivalent to that we can reach a formula ψ along the prefix ξ^n such that the suffix ξ_n satisfies ψ. Thus, if $\xi \models \varphi$ holds, then ξ will be accepted by a run in ST_φ.

Now we introduce the notion of looping formulas:

Definition 8 (Looping formula): We say φ is a *looping formula* iff there exists a trace $\xi \in \Sigma^\omega$ which can be written as an infinite sequence $\xi = \eta_0 \eta_1 \eta_2 \ldots$ such that η_i is a finite sequence and $\varphi \xrightarrow{\eta_i} \varphi$ for all $i \geq 0$. We write $\varphi \xrightarrow{\xi} \varphi$ in this case, and say φ is a looping formula with respect to ξ.

The following corollary is a direct consequence of Lemma 4 and the fact that we have only finitely many formulas in S_φ:

Corollray 2: If $\xi \models \varphi$, then there exists $n \geq 1$ such that $\varphi \xrightarrow{\xi^n} \psi \wedge \xi_n \models \psi \wedge \psi \xrightarrow{\xi_n} \psi$.

Proof: From Lemma 4 $\xi \models \varphi$ implies there is an infinite expansion path $\sigma = \varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_1} \psi_2 \xrightarrow{\omega_2} \ldots$ such that $\xi_i \models \psi_i$ for all $i \geq 1$. Since we have only finitely states, there must exist a ψ reachable from φ such that it appears infinitely often along this path. Obviously, this formula ψ is a looping formula as required. ■

This corollary gives the hint that after a finite prefix we can focus on the satisfiability of looping formulas. Now we give a lemma stating a nice property for the release operator:

Lemma 5: If $\varphi = \varphi_1 R \varphi_2$ and $\varphi \xrightarrow{\xi} \varphi$, then $\xi \models \varphi$.

Proof: Since $\varphi \xrightarrow{\xi} \varphi$, so we have $\exists n \cdot \varphi \xrightarrow{\xi^n} \varphi \wedge \varphi \xrightarrow{\xi_n} \varphi$. Let $\eta_i = \omega_i \omega_{i+1} \ldots \omega_n (0 \leq i \leq n)$. Here all expansions for the formula φ along the path must be from the right subformula φ_2 of φ, i.e., $\alpha \wedge X\psi \in NF(\varphi_2 \wedge X\varphi)$. Since $\varphi \xrightarrow{\xi^n} \varphi$, we have that $\forall 0 \leq i \leq n \cdot \varphi_2 \xrightarrow{\eta_i} \mathtt{tt}$, which implies $\xi_j \models \varphi_2$ for all $0 \leq j \leq n$. Inductively for $\varphi \xrightarrow{\xi_n} \varphi$ we can get the same property. So $\forall j \geq 0$ we have $\xi \models \varphi_2$, implying $\xi \models \varphi$.

■

Finally, we shall introduce an order on formulas to identify structural properties of looping formulas. We propose the following order for formulas:

Definition 9 (Poset on Formulas): For formulas φ, ψ, we write $\varphi \prec \psi$ iff $\varphi \in cl(\psi)$.

The order $\prec$ is obviously a partial order. For a looping formula φ, the set $CF(\varphi)$ posses at least one minimal element (w.r.t. the order $\prec$). Below we prove that all minimal elements of the set expand either to $\mathtt{tt}$ or themselves.

Lemma 6: If $\varphi \xrightarrow{\eta} \varphi$ then for all minimal element $\psi \in CF(\varphi)$ we have $\psi \xrightarrow{\eta} \mathtt{tt}$ or ψ.

Proof: Let ψ be a minimal element in $CF(\varphi)$. Since $\varphi \xrightarrow{\eta} \varphi$ and $\psi \in CF(\varphi)$, there must exist ψ' such that $\psi \xrightarrow{\eta} \psi'$ and $CF(\psi') \subseteq CF(\varphi)$ or $\psi' = \mathtt{tt}$. If $\psi' \neq \mathtt{tt}$, then according to Lemma 2 we know $CF(\psi') \subseteq cl(\psi)$. However, $cl(\psi) \cap CF(\varphi) = \{\psi\}$ because of the minimality of ψ. Thus $\psi' = \psi$. ∎

2) Soundness Proof of Theorem 2: We first introduce the relation $\models_f$:

Definition 10: Let $\eta = \omega_0\omega_1 \dots \omega_n$ $(n \geq 0)$. Then, we say the finite sequence η satisfies the formula φ, denoted by $\eta \models_f \varphi$, 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 aUa, which is a_1Ua_2 after tagging. According to the above definition, $\{a_1\} \not\models_f a_2$. The reader shall bear this in mind in the remaining of this section.

Below we present some simple properties of the relation $\models_f$ which will be useful later:

Lemma 7: 1) Assume $\eta \models_f \varphi$, then $\eta \models_f \varphi \vee \psi$.
2) Assume $\eta \models_f \varphi$ and $\eta \models_f \psi$, then $\eta \models_f \varphi \wedge \psi$.

Proof: Let $\eta = \omega_0\omega_1 \dots \omega_n$. We consider the first case: $\eta \models_f \varphi$ implies that there exists $O \in Olg(\varphi)$ such that $O \subseteq \eta$. Since $O \in Olg(\varphi) \subseteq Olg(\varphi \vee \psi)$, we have $\eta \models_f \varphi \vee \psi$. For the second case: $\eta \models_f \varphi$ implies that there exists $O_1 \in Olg(\varphi)$ such that $O_1 \subseteq \eta$. Similarly, $\eta \models_f \psi$ implies that there exists $O_2 \in Olg(\psi)$ such that $O_2 \subseteq \eta$. Since we have $O_1 \cup O_2 \in Olg(\varphi \wedge \psi)$ and $O_1 \cup O_2 \subseteq \eta$, we have $\eta \models_f \varphi \vee \psi$. ∎

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 $\varphi \xrightarrow{\eta} \varphi$. Then, $\eta \not\models_f \varphi$.

Proof: Let $\eta = \omega_0\omega_1 \dots \omega_n$ and we rewrite $\varphi \xrightarrow{\eta} \varphi$ as $\varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_2} \dots \xrightarrow{\omega_n} \varphi$. Note it is apparent that $\varphi \xrightarrow{\omega_0} (\varphi \wedge \varphi_1')$ with $\varphi_1 \xrightarrow{\omega_0} \varphi_1'$. Thus, by induction one can show that along the path $\varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_1} \psi_2 \dots$ it holds $\varphi \in CF(\psi_i)$ for all i. Since the transition for conjunctive formula ψ_i is obtained by combining transitions for each $\psi' \in CF(\psi_i)$, the transition for the subformula $\varphi \in CF(\psi_i)$ must be from left subformula φ_1, i.e., $\alpha \wedge X\psi \in NF(\varphi_1 \wedge X\varphi)$. As a result, for each i, the label ω_i must be a superset of $CF(\alpha_i)$, and $CF(\alpha_i)$ contains the literals from φ_1. Moreover, the tagging function has been used to classify the atoms. According to Definition 5 the atoms in φ_1 and φ_2 are never tagged the same – this is because the atoms in φ_2 will tag φ while those in φ_1 will not. So φ_1 and φ_2 still don't have common atoms, thus $CF(\alpha_i)$ contains no obligation literals from φ_2;

According to the definition 4, the obligation literals of φ are all from those of φ_2. Thus, from the definition of $\models_f$ (Definition 10), we know $\eta \not\models_f \varphi$. ∎

The following lemma says that if there exists a partitioning $\xi = \eta_1\eta_2...$ that makes φ expanding to itself by each η_i and $\eta_i \models_f \varphi$ holds, then $\xi \models \varphi$.

Lemma 9: Given a looping formula φ and a trace ξ, let $\xi = \eta_1\eta_2 \dots \cdot$ if $\forall i \geq 1 \cdot \varphi \xrightarrow{\eta_i} \varphi \wedge \eta_i \models_f \varphi$, then $\xi \models \varphi$.

Proof: We enumerate the set $CF(\varphi) = \{\varphi_1, \varphi_2, \dots, \varphi_n\}$, and we shall prove $\xi \models \bigwedge CF(\varphi)$. Let S_0 denote the minimal elements of $CF(\varphi)$ with respect to the partial order $\prec$. Moreover, we define $S_{i+1} = S_i \cup \{\psi' \in CF(\varphi) \mid \exists \psi \in S_i . \psi \prec \psi'\}$. Obviously, there is a finite index k such that $S_k = CF(\varphi)$. We proceed with induction over the index:

1) Basic step: Let $\psi \in S_0$. By Lemma 6 if $\varphi \xrightarrow{\eta_i} \varphi$, then $\psi \xrightarrow{\eta_i} \mathtt{tt}$ or $\psi \xrightarrow{\eta_i} \psi$. If $\exists \eta_i \cdot \psi \xrightarrow{\eta_i} \mathtt{tt}$ holds, $\xi \models \psi$ follows from Lemma 4. Otherwise we have $\psi \xrightarrow{\eta_i} \psi$ for all $i \geq 1$. According to LTL semantics ψ must be either until or release formula. Applying Lemma 8 we know that ψ cannot be an Until formula, and thus be a Release formula. Then Lemma 5 implies that $\xi \models \psi$, and therefore $\xi \models \bigwedge S_0$.
2) For induction step we assume $\xi \models \bigwedge S_k$. Consider arbitrary η_i: let $\psi \in S_{k+1} \setminus S_k$ and assume $\psi \xrightarrow{\eta_i} \psi'$. Since $\varphi \xrightarrow{\eta_i} \varphi$, we have $CF(\psi') \subseteq CF(\varphi)$. By the construction of the set S_i, $CF(\psi')$ 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 $\eta_{i+1}\eta_{i+2} \dots \models \psi'$ so $\eta_i\eta_{i+1} \dots \models \psi$ (From Lemma 4), thus $\xi \models \psi$. Now consider the case $\psi \in CF(\psi')$:$\psi \xrightarrow{\eta_i} \psi$ implies that ψ must be a Release formula. Then Lemma 5 implies again that $\xi \models \psi$, and therefore $\xi \models \bigwedge S_{k+1}$.

∎

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

Lemma 10 (Soundness): Let $\varphi \in S_\lambda$. Assume that there exists a SCC B of ST_φ such that $\psi \in B$ and $L(B)$ is a superset of some obligation set $O \in Olg(\psi)$. Then, $SAT(\varphi)$.

Proof: As B is a SCC, we have a path $\delta := \psi(\psi_1) \xrightarrow{\omega_1} \psi_2 \xrightarrow{\omega_2} \dots \xrightarrow{\omega_{k-1}} \psi$ such that η visits all states in B and all transitions between states in B ($\eta = \omega_1\omega_2 \dots \omega_{k-1}$). Since all states in ST_φ are reachable from φ, there must exist a finite sequences η_0 such that $\varphi \xrightarrow{\eta_0} \psi$. We construct $\xi := \eta_0\eta^\omega$. By assumption there exists $O \in Olg(\psi)$ such that $O \subseteq L(B) = \cup_{i=1}^{k-1} \omega_i$. So according to the definition of $\models_f$ we have $\eta \models_f \psi$. Thus from Lemma 9 we have $\xi \models \varphi$. So φ is satisfiable. ∎

3) Completeness Proof of Theorem 2:

Lemma 11: $\xi \models \varphi \Rightarrow \exists n \cdot \xi^n \models_f \varphi$.

Proof: We prove it by structural induction over the formula φ. For the base case assume φ is $\mathtt{tt}$ or a literal, $\xi^1 \models_f \varphi$ by definition. Moreover, φ can not be $\mathtt{ff}$. Now we consider the induction step:

- If $\varphi = X\psi$, then $\xi \models \varphi \Rightarrow \xi_1 \models \psi$. By induction hypothesis we know $\exists n \cdot {\xi_1}^n \models_f \psi$ holds, so $\xi^{n+1} \models_f \varphi$ holds.

- If $\varphi = \varphi_1 \wedge \varphi_2$, then $\xi \models \varphi_1 \wedge \xi \models \varphi_2$. By induction hypothesis we have $\exists n_1 \cdot \xi^{n_1} \models_f \varphi_1$ and $\exists n_2 \cdot \xi^{n_2} \models_f \varphi_2$ hold. Observe that $\xi^n \models_f \varphi$ implies $\xi^m \models_f \varphi$ for all $m \geq n$. Now from Lemma 7 we have that $\xi^n \models_f \varphi$ with $n := \max(n_1, n_2)$.
- If $\varphi = \varphi_1 \vee \varphi_2$, then $\xi \models \varphi_1 \vee \xi \models \varphi_2$. By induction hypothesis we have $\exists n_1 \cdot \xi^{n_1} \models_f \varphi_1$ or $\exists n_2 \cdot \xi^{n_2} \models_f \varphi_2$ holds. Without loss of generality, assume $\exists n_1 \cdot \xi^{n_1} \models_f \varphi_1$. Lemma 7 implies then $\xi^{n_1} \models_f \varphi_1 \vee \varphi_2$.
- If $\varphi = \varphi_1 U \varphi_2$, $\xi \models \varphi_1 U \varphi_2$ implies that there exists $i \geq 0$ such that $\xi_i \models \varphi_2$. By induction hypothesis we have $\exists n \cdot {\xi_i}^n \models_f \varphi_2$ hold, thus there exists an obligation $O \in Olg(\varphi_2)$ such that $O \subseteq {\xi_i}^n$. This implies $O \subseteq \xi^{i+n}$, thus $\xi^{i+n} \models_f \varphi$ holds.
- If $\varphi = \varphi_1 R \varphi_2$, we observe first that $\xi \models \varphi_2$ must hold. By induction hypothesis we know $\exists n \cdot \xi^n \models_f \varphi_2$. According to Definition 10 we have that $\xi^n \models_f \varphi$ holds as well.

■

Lemma 12: Let φ be a looping formula, and assume $\varphi \xrightarrow{\xi} \varphi$ and $\xi \models \varphi$ hold. Then there exists a partitioning $\xi = \eta_1 \eta_2 \ldots$ and $\forall i \geq 0 \cdot \varphi \xrightarrow{\eta_i} \varphi \wedge \eta_i \models_f \varphi$ holds.

Proof: Assume $\varphi \xrightarrow{\xi} \varphi \wedge \xi \models \varphi$. We first prove that there exist n such that $\varphi \xrightarrow{\xi^n} \varphi \wedge \xi^n \models_f \varphi \wedge (\varphi \xrightarrow{\xi_n} \varphi \wedge \xi_n \models \varphi)$. From Lemma 11 $\xi \models \varphi$ implies that there exists k such that $\xi^k \models_f \varphi$. Since φ is a looping formula with respect to ξ, we can find the $n \geq k$ such that $\varphi \xrightarrow{\xi^n} \varphi$ and $\varphi \xrightarrow{\xi_n} \varphi$ hold. For $n \geq k$, $\xi^n \models_f \varphi$ holds as well. Now we apply Lemma 4: $\xi \models \varphi$ implies that $\varphi \xrightarrow{\xi^n} \varphi$ and $\xi_n \models \varphi$.

Since $\varphi \xrightarrow{\xi_n} \varphi \wedge \xi_n \models \varphi$, applying the arguments above inductively yields the lemma. ■

The above lemma states that if $\varphi \xrightarrow{\xi} \varphi$ as well as $\xi \models \varphi$, we can find a partitioning $\eta_1 \eta_2 \ldots$ that makes φ expend to itself by each η_i and $\eta_i \models_f \varphi$ holds. Combining Lemma 6, Lemma 7 and Corollary 1, we have our central theorem:

Lemma 13 (Completeness): Let $\varphi \in S_\lambda$. Then, $SAT(\varphi)$ implies that there exists a SCC B of ST_φ and such that $\psi \in B$ and $L(B)$ is a superset of some obligation set $O \in Olg(\psi)$.

Proof: Since $SAT(\varphi)$, let ξ such that $\xi \models \varphi$. Then by Corollary 2 there exists $n \geq 0$ and $\psi \in S_\varphi$ such that $\varphi \xrightarrow{\xi^n} \psi$, $\psi \xrightarrow{\xi_n} \psi$ and $\xi_n \models \psi$. Moreover, by Lemma 12, there exists a partition $\xi_n = \eta_1 \eta_2 \ldots$ such that for every finite sequence η_i we have $\psi \xrightarrow{\eta_i} \psi$ as well as $\eta_i \models_f \psi$. As the state ψ is visited infinitely often, there must be a SCC B such that $\psi \in B$. According to the definition of $\models_f$ we know that there exists $O \in Olg(\psi)$ such that $O \subseteq \eta_1$. Obviously, $\psi \xrightarrow{\eta_1} \psi$ is contained in some SCC B, thus $\eta \subseteq L(B)$, implying $O \subseteq L(B)$. ■