Verify LTL with Fairness Assumptions Efficiently

Yong Li∗†, Lei Song∗, Yuan Feng‡, Lijun Zhang∗†
∗State Key Laboratory of Computer Science, Institute of Software, CAS, China
†University of Chinese Academy of Sciences, China
‡Centre for Quantum Computation and Intelligent Systems, University of Technology Sydney, Australia

Abstract—This paper deals with model checking problems with respect to LTL properties under fairness assumptions. We first present an efficient algorithm to deal with a fragment of fairness assumptions and then extend the algorithm to handle arbitrary ones. Notably, by making use of some syntactic transformations, our algorithm avoids constructing corresponding Büchi automata for the whole fairness assumptions, which can be very large in practice. We implement our algorithm in NuSMV and consider a large selection of formulas. Our experiments show that in many cases our approach exceeds the automata-theoretic approach up to several orders of magnitude, in both time and memory.

I. INTRODUCTION

Linear Temporal Logic (LTL) [24] has been shown to be a proper specification language. As a result, for verifying reactive systems, model checkers for LTL, like Spin [18] and NuSMV [7], have been applied in practice successfully. To verify whether or not a system satisfies an LTL formula, the classical automata-theoretic approach [32] is usually adopted: Firstly, a Büchi automaton is built which accepts all executions violating the LTL formula; Secondly, a product system is built from the original system and the Büchi automaton; Finally, the problem is reduced to finding an accepting path in the product system. Since in the worst case the constructed Büchi automaton can be exponentially larger than the LTL formula, both time and space complexity of the algorithm in [32] is exponential with respect to the size of the LTL formula. The complexity of this algorithm is shown to be PSPACE-complete in [30]. Even if we restrict to a small subset of LTL formulas (those only containing eventual modality F), it is still NP-complete. On the other side, due to the popularity of LTL, many ideas have been proposed optimizing the construction of Büchi automata, see e.g. [10], [14], [16], [31], [19], [28].

The classification of properties into different categories is pivotal for efficient verification of reactive systems. In the seminal paper [22], Lamport introduced the notions of safety and liveness properties, where “safety” properties assert something “bad” never happens, while “liveness” properties require something “good” will eventually happen. These notions were later formalized by Alpern and Schneider in [1]. Properties were further classified into strong safety and absolute liveness in [29], and fair properties. The notion of fairness is important for verifying liveness in reactive systems to remove unrealistic behaviors [15].

In practice, fairness assumptions can have a great impact on the performance in many cases. For instance in the binary semaphore protocol [17], the fairness assumption that whenever a process is ready, it will have a chance to enter the critical section, is given by: \( \bigwedge_{1 \leq i \leq n}(GF_{enter_i} \rightarrow GF_{critical}) \) (G and F denote “always” and “eventually”, respectively) with \( n \) being the number of processes. When \( n = 5 \), the corresponding Büchi automaton generated by LTL3BA [16] has more than 300 states and 1 million transitions\(^1\). Therefore, model checking formulas under such an assumption will be time and memory consuming even when given formulas are simple.

In this paper we propose a novel algorithm to verify fairness as well as general properties with fairness assumptions. We do not only consider simple fairness formulas as mentioned above, but also consider more complex fairness with nested modalities like \( FG(a \lor Fb) \). Moreover, we extend the notion of fairness assumptions to full LTL formulas, which allows us to specify some fairness assumption like “a and X(bUc) holds infinitely often”. Notably, our algorithm relies on a syntactic transformation and avoids constructing a Büchi automaton for the whole fairness. The approach is presented in three steps:

- We first restrict to fairness with only F and G modalities, for which our syntactic transformation can completely avoid Büchi automata construction. For this setting our approach achieves a speedup of four orders of magnitudes on some examples.
- We then extend the algorithm to deal with fair formulas of full LTL. The idea is to transform a fair formula into an equivalent one in disjunctive norm form, each sub-formula of which can be handled by specific and efficient algorithms. Even though we may still resort to the automata-theoretic approach for some sub-formulas, they are often much smaller than the original one.
- Finally, we show our approach can be adapted to accelerate the verification of generic LTL formulas under fairness assumptions.

We have implemented the algorithm in NuSMV and compared it with the classical algorithm. Our experimental results show that for many cases while NuSMV runs out of time or memory quickly, our algorithm terminates within seconds using memory less than 100 MB. The main reason is that after the syntactical transformation, we can avoid constructing Büchi automata for many sub-formulas. Even for those where Büchi automata construction is inevitable, their corresponding

\(^1\)Interested readers can try the online LTL translator available at: http://spot.lip6.fr/ltl2tgba.html
automata are relatively small and can be constructed efficiently.

It should be pointed out, however, that the syntactical transformation may also cause exponential blow-ups in the length of given formulas. Hence, as the experimental results show, our algorithm may be much slower than NuSMV in some cases. We then further discuss and characterize the formulas for which our approach provides better performance.

a) Related Work: There is a plenty of work on optimizing verification of LTL (or its sub-logic). Here we only briefly recall a few closely related works. In [4], specialized algorithms are proposed to deal with LTL properties, which can be represented by either terminal or weak automata. Compared to general algorithms, specialized algorithms improve the worst-case time complexity by a constant factor. This result is further formalized in [6], where it is shown that terminal and weak automata correspond to guarantee properties (something happens eventually) and persistence properties (something always happens eventually), respectively. For guarantee properties, model checking algorithm reduces to the reachability of an accepting state, while for persistence properties, it reduces to finding a fully accepting cycle, i.e., all states on it are accepting. Furthermore, a decision algorithm is proposed in [6] to check whether an LTL formula is a guarantee or persistence property. For properties which are neither guaranteed nor persistent, the general algorithm has to be used. One exception is [26], where a decomposition algorithm is proposed for strong automata, which are neither terminal nor weak. The idea is to decompose a strong automaton into three sub-automata, which are terminal, weak, and strong, respectively. Then specialized algorithms can be used to check the terminal and weak sub-automata. Since the strong sub-automaton is smaller than the original automaton, decomposition always speeds up the verification according to the experiment in [26].

Differently, our algorithm performs decomposition syntactically on given formulas, hence we do not need to build their corresponding Büchi automata at the beginning. While the specialized algorithm in [4], [6], [26] is automata-based, Büchi automata have to be built beforehand, which may take a significant amount of time and memory, especially when the given formula is long [17]. Moreover, our algorithm works for arbitrary fairness including those which are neither guaranteed nor persistent.

b) Organization of the paper: Section II introduces some definitions and notations used throughout the paper. The algorithm is presented in Section III. We demonstrate the efficiency of our algorithm via experiment in Section V. Finally, we conclude our paper in Section VI.

All missing proofs can be found in [?].

II. Preliminaries

We shall first introduce some preliminary definitions and notations and then present the syntax and semantics of LTL.

Let $X$ be a finite set of elements and $\xi = x_0 x_1 \ldots \in X^\omega$ with $X^\omega = \bigcup_{i \geq 0} X^i$ a finite sequence of elements in $X$. For each $\xi \in X^i$, we let $|\xi| = i + 1$ denote its length. An infinite sequence $\xi \in X^\omega$ is cyclic if there exists $\xi' \in X^i$ for some $i$ such that $\xi = (\xi')^\omega$, i.e., repeating $\xi'$ for infinite times. Let $\xi[i] = x_i$ denote the $(i+1)$-th element on $\xi$ if it exists. We shall write $\xi[i]$ to denote the prefix of $\xi$ ending at the $(i+1)$-th element, while $\xi[i]$ the suffix of $\xi$ starting from the $(i+1)$-th element. Let $\xi \in X^*$ and $\xi' \in X^\omega$. Then $\xi \cdot \xi'$ denotes the infinite sequence obtained by attaching $\xi'$ to the end of $\xi$.

We will fix a finite set of atomic propositions, denoted $AP$ and ranged over by $a, b, c, \ldots$, throughout the remainder of the paper. The syntax of LTL is given by the following grammar:

$$\varphi, \psi ::= a | \neg a | \varphi_1 \land \varphi_2 | \varphi_1 \lor \varphi_2 | X \varphi | \varphi_1 U \varphi_2 | \varphi_1 W \varphi_2$$

where $a \in AP$ and $\varphi, \psi, \varphi_1, \varphi_2$ range over LTL formulas. As usual, we introduce some abbreviations: $1 = a \lor \neg a$ and $0 = a \land \neg a$ denote True and False respectively, while $F \varphi = 1 U \varphi$ (eventually $\varphi$), $G \varphi = \varphi W 0$ (always $\varphi$), and $(\varphi_1 \rightarrow \varphi_2) = \neg (\varphi_1 \lor \varphi_2)$. Let $l_1, l_2, \ldots$ range over propositional formulas, i.e., formulas defined by: $l ::= a | \neg a | l_1 \lor l_2 | l_1 \land l_2$.

Given an infinite sequence of sets of atomic propositions $\rho = A_0 A_1 \ldots \in (2^AP)^\omega$ and an LTL formula $\varphi$, we say $\rho$ satisfies $\varphi$, written as $\rho \models \varphi$, if:

$$\rho \models a \iff a \in \rho[0]$$

$$\rho \models X \varphi \iff \rho[1] \models \varphi$$

$$\rho \models \varphi_1 U \varphi_2 \iff \exists i \geq 0 (\rho[i] \models \varphi_2 \land \forall j < i \rho[j] \models \varphi_1)$$

$$\rho \models \varphi_1 W \varphi_2 \iff (\forall i \geq 0 \rho[i] \models \varphi_1) \lor (\rho \models \varphi_1 U \varphi_2)$$

All other connectives are defined in a standard way. For formulas $\varphi$ and $\psi$, we say that $\varphi$ and $\psi$ are semantically equivalent, denoted $\varphi \equiv \psi$, if $\rho \models \varphi$ iff $\rho \models \psi$ for any $\rho \in (2^AP)^\omega$.

Here we only define LTL formulas in positive normal form, in the sense that the negation operator can only be applied to atomic propositions. However, it is well-known that any LTL formula can be transformed into an equivalent one in positive normal form, using $\neg (X \psi) \equiv X (\neg \psi)$ and the following duality laws:

$$\neg (\varphi_1 U \varphi_2) \equiv (\varphi_1 \land \neg \varphi_2) W (\neg \varphi_1 \land \varphi_2)$$

$$\neg (\varphi_1 W \varphi_2) \equiv (\varphi_1 \land \neg \varphi_2) U (\neg \varphi_1 \land \varphi_2)$$

Fairness assumptions are critical to rule out unrealistic behaviors when performing verification; see for instance [25], [15]. Formally, fairness is a fragment of LTL, which can be defined as follows:

**Definition 1** ([29]). An LTL formula $\varphi$ is a fairness iff for any $\rho \in (2^AP)^\omega$,

1. the set of sequences satisfying $\varphi$ is closed under suffixes, 
   i.e., $\rho \models \varphi$ implies $\rho[i] \models \varphi$ for any $i \geq 0$;
2. the set of sequences satisfying $\varphi$ is closed under prefixes, 
   i.e., $\rho \models \varphi$ implies $\rho[i] \models \varphi$ for any $\rho[i] \in (2^AP)^\omega$.

We shall refer properties defined in Definition 1 as fair formulas or fairness in the following. According to Definition 1, the following lemma is straightforward:

**Lemma 1.** $\varphi$ is a fairness iff $\varphi \equiv G \varphi$ and $\varphi \equiv F \varphi$. 

As a result of Lemma 1, we can add any number of F and G in front of a fairness without changing its semantics. For instance, fairness $Fa \lor G\neg a$ is equivalent to $GF(Fa \lor G\neg a)$.

As usual we consider models given as Kripke structures, which are formally defined as follows:

**Definition 2.** A Kripke structure is a tuple $\mathcal{K} := (S, s, T, AP, L)$ where $S$ is a finite set of states, $s \in S$ is the initial state, $T \subseteq S \times S$ is a set of transitions, and $L : S \rightarrow 2^{AP}$ is a labeling function. We assume that for each $s \in S$, there exists $s' \in S$ such that $(s, s') \in T$.

We fix a Kripke structure $\mathcal{K} = (S, s, T, AP, L)$ throughout the remainder of the paper. Let $r, s, t, \ldots$ range over $S$. Let $Paths^0(s) \subseteq S^0$ denote the set of all finite paths starting from $s$ such that $\pi \in Paths^0(s)$ iff $\pi[0] = s$ and for any $i \geq 0$, $(\pi[i], \pi[i+1]) \in T$. Similarly, we can define $Paths^*(s)$, i.e., finite paths in $\mathcal{K}$ starting from $s$. Let $Paths^0(\mathcal{K}) = Paths^0(s)$ and $Paths^*(\mathcal{K}) = Paths^*(s)$. Given $\pi \in S^0$, we let $\text{trace}(\pi)$ denote the trace of $\pi$ such that $\text{trace}(\pi)[i] = L(\pi[i])$ for all $i \geq 0$, i.e., $\text{trace}(\pi)$ denotes the sequence of labels of states in $\pi$. For an LTL formula $\varphi$, we write $\pi \models \varphi$ iff $\text{trace}(\pi) \models \varphi$; $s \models \varphi$ iff $\pi \models \varphi$ for all $\pi \in Paths^0(s)$; $\mathcal{K} \models \varphi$ iff $\bar{s} \models \varphi$. Given an LTL formula $\varphi$ and a fairness $\varphi_j$, $\mathcal{K}$ satisfies $\varphi$ under the assumption $\varphi_j$ iff $\mathcal{K} \models (\varphi_j \rightarrow \varphi)$.

**Example 1.** An example for Kripke structure is $\mathcal{K} = (\{s_0, s_1, s_2\}, s_0, T, \{a, b, c\}, L)$, where $T$ and $L$ are depicted in Figure 1, for instance $L(s_0) = \{a\}$. Obviously, traces in $\mathcal{K}$ can be represented as $\{\{a\}^* \{\{a, c\}^* \{a\}^0 \{(\{a\}^* \{\{a, c\}^0\}\}^0\}\}^\omega$.

Moreover, we directly conclude the corollary below from Lemma 1:

**Corollary 1.** Let $\pi \in Paths^0(\mathcal{K})$ and $\varphi$ a fairness. Then for any index $j \geq 0$, $\pi \models \varphi$ iff $\pi|_j \models \varphi$.

**Proof.** Since $\varphi$ is a fairness, $\varphi \equiv G\varphi$. Thus $\pi \models \varphi$ implies $\pi|_j \models \varphi$ for any index $j$. On the other hand, $\pi|_j \models \varphi$ implies $\pi \models \varphi$. Then we conclude $\pi \models \varphi$ by $\varphi \equiv G\varphi$. \qed

Intuitively, we can safely consider only the suffixes of the paths when the given formula is a fairness.

### III. Model Checking Fairness

In this section, we present an algorithm for model checking fair formulas. We first describe the overall idea. For fair formula $\varphi$, $\mathcal{K} \models \varphi$ means that for all infinite paths $\pi$ starting from initial state $\bar{s}$, $\pi \models \varphi$. Conversely, if $\neg(\mathcal{K} \models \varphi)$, then there exists an infinite path $\pi$ such that $\pi \models \neg \varphi$. Thus, we first construct the negation $\neg \varphi$, which is also a fair formula. Then, we construct a fair normal form of $\neg \varphi$, denoted by $\text{fnf}(\neg \varphi)$, which has the form $\bigvee_{i=1}^m \varphi_i$, with each $\varphi_i$ being a fair formula. Then, the problem reduces to checking whether there exists an infinite path $\pi$ such that $\pi \models \varphi_i$. In other words, whether there exists an SCC $B$ satisfying $\varphi_i$. The satisfaction here can be checked by analysing the SCC $B$. A strongly connected component (SCC) $B$ of $\mathcal{K}$ is a state set such that for any $s, t \in B$, there exists a path from $s$ to $t$. We here do not consider trivial SCCs which are single states without self loops.

We start with treating fairness formulas in LTL($F$, $G$), then we extend the algorithm to deal with general fairness. Finally, we handle all LTL formulas with fairness assumptions.

#### A. Fairness in LTL($F$, $G$)

In this subsection, we focus on a fragment of LTL formulas, denoted LTL($F$, $G$), which only contains $F$ and $G$ modalities, i.e., it is defined by the following grammar:

$$\varphi ::= a \mid \neg (\varphi \land \varphi) \mid F \varphi \lor G \varphi$$

For each fairness in LTL($F$, $G$), we shall show that it can be transformed into an equivalent formula where all propositional formulas are directly preceded by precisely two modalities, either $F$ or $G$. Such a transformation is purely syntactical: we call the transformation procedure the flatten operation, denoted by $\text{fnf}$.

**Theorem 1.** Let $\varphi \in \text{LTL}(F, G)$ be a fairness. Then, it can be transformed into the following equivalent formula, referred to also as its fair normal form:

$$\text{fnf}(\varphi) := \bigvee_{i=1}^m (FG_{l_i} \land (\bigwedge_{j=1}^{n_i} GF_{l_{ij}}))$$

where $l_i$ and $l_{ij}$ are propositional formulas.

Note that $m$ and $n_i$ are nonnegative integers and we omit $FG_{l_i}$ and $GF_{l_{ij}}$ in the fair normal form whenever $l_i$ and $l_{ij}$ are 1. The syntactic transformation $\text{fnf}$ is the key of the algorithm: $\text{fnf}(\varphi)$ can be checked directly on $\mathcal{K}$ without constructing the product automaton. We first give an example to illustrate the main steps of verifying fair formulas in LTL($F$, $G$).

**Example 2.** Take $\varphi = \neg(FG(a \lor (Fb \land Gc)))$ for example, the fair normal form of $\neg \varphi$ is $\text{fnf}(\neg \varphi) = FAG \lor (FGc \land GFb)$. Consider model checking the Kripke structure $\mathcal{K}$ in Example 1 against fair formula $\varphi$. We already have the fair normal form of $\neg \varphi$ above, so we only need to check whether there exists an SCC satisfying fair formula $FAG$ or $FGc \land GFb$. Consider fair formula $FAG$, we find that there exists an SCC $\{s_0\}$ reachable from initial state $s_0$ that fulfils the formula. We therefore conclude that $\mathcal{K}$ does not satisfy $\varphi$ and give a counterexample $\pi = (s_0)^\omega$ such that $\pi \models \neg \varphi$, thus $\pi \not\models \varphi$.

We below give the intuition behind the syntactic transformation.

First, we have to deal with trivial fair formula such as $Fa \lor G\neg a$. Due to Lemma 1, we first add $GF$ in front of the original formula and then apply the flatten operation, which gives us the fair normal form $GFa \lor FG\neg a$.
Given a fairness \( \varphi \in \text{LTL}(F, G) \), our goal is to obtain an equivalent formula of the fair normal form. To that end, we first make sure that there exists at least one FG or GF in front of every propositional formula, which is guaranteed by safely adding GF in front of \( \varphi \). After that, we are going to push every FG and GF directly in front of all propositional formulas. To achieve this, one needs to discuss the distributivity of GF and FG over \( \lor \) and \( \land \).

Suppose \( \varphi_1, \varphi_2 \in \text{LTL} \), our goal is pushing GF and FG inside such that they appear only before propositional formulas.

We consider the following four cases:

1. \( \text{GF} \equiv \text{FG}, \text{GFF} \equiv \text{FG}, \text{GGF} \equiv \text{FG} \) and \( \text{FGF} \equiv \text{GF} \) are trivial according to the semantics of LTL. This insures that we have only GF and FG modalities since we first add GF in front of \( \varphi \).

2. \( \text{GF}(\varphi_1 \lor \varphi_2) \equiv \text{GF}\varphi_1 \lor \text{GF}\varphi_2 \) and \( \text{FG}(\varphi_1 \land \varphi_2) \equiv \text{FG}\varphi_1 \land \text{FG}\varphi_2 \) hold since GF and FG are distributive over \( \lor \) and \( \land \) operator respectively.

3. \( \text{GF}(\varphi_1 \land \text{FG}\varphi_2) \equiv \text{GF}\varphi_1 \land \text{GF}\varphi_2 \), \( \text{FG}(\varphi_1 \lor \text{GF}\varphi_2) \equiv \text{FG}\varphi_1 \lor \text{GF}\varphi_2 \) and \( \text{FG}(\varphi_1 \land \text{FG}\varphi_2) \equiv \text{FG}\varphi_1 \land \text{FG}\varphi_2 \). Intuitively, if the operands of GF and FG are not propositional formulas, they must be the four cases we listed here after we go through case 2) and case 4).

4. \( \text{GF}(\varphi_1 \lor \varphi_2) \) and \( \text{FG}(\varphi_1 \lor \varphi_2) \). This is the most challenging part since GF (FG) is not distributive over \( \land \) or \( \lor \) operator. The following procedure relies on the structure of the formula. If the operand of GF or FG is propositional formula, then it is already the formula we desire. Otherwise, if they are not case 3) such as the formula \( \text{FG}(a \lor (Fb \land Gc)) \), we transform \( \varphi_1 \lor \varphi_2 \) and \( \varphi_1 \lor \varphi_2 \) to disjunctive normal form (DNF) and conjunctive normal form (CNF) respectively. After that, we apply case 2) and may use case 3) for further processing.

Once we get a formula where all propositional formulas are adjacent to GF or FG, we transform it into DNF, which gives us the fair normal form in Theorem 1. We illustrate the procedure of \( \text{fnf} \) operator via an example as follows:

**Example 3.** Let \( \varphi = \text{FG}(a \lor (Fb \land Gc)) \). We show how to flatten \( \varphi \) step by step:

- Add GF in front of \( \varphi \) which gives us \( \varphi \) since \( \text{GFG} \equiv \text{GF} \);
- Since \( \varphi \) is an instance of case 4), we first transform \( a \lor (Fb \land Gc) \) into a CNF, which results in \( \text{FG}(a \lor Fb) \land (a \lor Gc) \);
- According to case 2), FG is distributive over \( \land \) operator, we therefore directly push FG inside, which gives us \( \text{FG}(a \lor Fb) \land \text{FG}(a \lor Gc) \);
- The resulting formula is an instance of case 3), we get \( \text{FGa} \lor (\text{FGa} \land \text{FGFb}) \) after we apply the equations in case 3).
- Since all FG are adjacent to propositional formulas, we transform above formula to DNF, which gives us the fair normal form \( \text{FGa} \lor (\text{FGc} \land \text{GFb}) \) of \( \varphi \).

**Algorithm 1** The procedure \( \text{fairMC} \) for checking whether \( \mathcal{K} \models \varphi \), where \( \varphi \) is a fair formula in LTL(F,G).

\begin{verbatim}
1: procedure fairMC(\varphi, \mathcal{K}) returns True if \mathcal{K} \models \varphi, and False otherwise.
2: \text{fnf}(\neg \varphi) \equiv \bigvee_{i=1}^m \varphi_i = \bigvee_{i=1}^m (\text{FG}l_i \land (\bigwedge_{j=i}^{m} \text{GF}l_j));
3: for all (1 \leq i \leq m) do
4: \text{B} \leftarrow \{s \in S \mid s \models l_i\};
5: if B \neq \emptyset then
6: for all (SCC B' \subseteq B) do
7: if (B' is accepting for \varphi) then
8: return False;
9: return True;
\end{verbatim}

Intuitively, it means whenever \( \pi \models \varphi \), it must be the case that \( \pi \) ends up with a loop such that either all states on the loop satisfy \( a \) or all states satisfy \( c \) and at least one state satisfies \( b \). This also can be verified by applying the semantics of LTL.

By Corollary 1, we only need to consider the infinite suffixes of the paths that all states will be visited infinitely often. That is to say, we only need to consider all the SCCs of \( \mathcal{K} \) that can be reached.

**Definition 3** (Accepting SCC). Given a formula \( \varphi = \text{FG}l \land (\bigwedge_{j=1}^{m} \text{GF}l_j) \) and an SCC \( B \). If \( 1 \) for every state \( s \in B \), \( s \models l \) and \( 2 \) for each \( j \), there exists \( s \in B \), such that \( s \models l_j \), then we say SCC \( B \) is accepting for \( \varphi \).

With the definition of accepting SCC, we have the following theorem:

**Theorem 2.** For any \( \varphi = \text{FG}l \land (\bigwedge_{j=1}^{m} \text{GF}l_j) \), there exists an infinite path \( \pi \) in \( \mathcal{K} \) such that \( \pi \models \varphi \) if and only if there exists a reachable SCC \( B \) such that \( B \) is accepting for \( \varphi \).

**Proof.** \( \Rightarrow \) Since \( \mathcal{K} \) is finite, for any \( \pi \in \text{Paths}^\omega(\mathcal{K}) \), there exists a smallest index \( k \), such that all states in \( \pi_k \) will be visited by infinite times. By Corollary 1, it suffices to show that \( \pi \models \varphi \) iff \( \pi_k \models \varphi \) since one can check that \( \varphi \) is a fairness. For convenience, let \( \pi_1 = \pi_k \). Let \( B_1 \) be the set of states on \( \pi_1 \). Obviously, all states in \( B_1 \) are connected since all states will be visited by infinite times. \( \pi_1 \models \text{FG}l \) means \( s \models l \) for each \( s \in B_1 \) and \( \pi \models \text{GF}l_j \) means that there exists \( s \in B_1 \) such that \( s \models l_j \), for each \( l_j \). Let \( B = B_1 \subseteq S \), then \( B \) is an SCC and is accepting for \( \varphi \).

\( \Leftarrow \) This direction is trivial, since we can always construct a path \( \pi_2 \) that starts from any \( s \in B \) and visits all states in \( B \) by infinite times. Since \( B \) is reachable, we can find a finite path \( \pi_1 \) which starts from the initial state and reaches the first state of \( \pi_2 \). Let \( \pi = \pi_1 \cdot \pi_2 \). Obviously \( \pi \models \text{FG}l \land (\bigwedge_{j=1}^{m} \text{GF}l_j) \), thus we complete the proof.

Based on Theorem 2, Algorithm 1 describes the procedure to determine whether all paths in \( \mathcal{K} \) satisfy a given fair formula \( \varphi \) in LTL(F,G). For this, the algorithm first syntactically transforms \( \neg \varphi \) into an equivalent formula of the form
formulas in LTL(F, G)

fairness in B. Expressiveness of fairness in LTL(F, G).

are three kinds of letters, namely in SCC matters. We show that ϕ at every position is equivalent to that eventually there is a loop

Algorithm 1 runs in time $O(|\mathcal{X}| \times 2^{\ell(\phi)})$ and in space $O(|\mathcal{X}| + |\phi| \times 2^{\ell(\phi)})$.

Due to case 4) in the explanation of Theorem 1 and the transformation that gives a formula of DNF, the resulting formula length can be $O(2^{\ell(\phi)})$ in the worst case. Suppose $n_1$ is the number of propositional formulas first preceded by G, and $n_2$ for number of propositional formulas first preceded by F, obviously $n_1 + n_2 \in O(|\phi|)$. We then have 2$^{n_1}$ options for FG/ formulas and 2$^{n_2}$ for $\wedge_{i=1}^{n_1} \text{GF}l_i$ since the number of $l_k$ is $n_2$, so we will at most have 2$^{n_1+n_2}$ formulas have the form $FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j)$ and each formula of that form at most has $n_2 + 1$ propositional formulas, which means that formula length can be $|\phi| \times 2^{O(|\phi|)}$ in the worst case. That is, we will at most have 2$^{n_1+n_2}$ formulas with the form $FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j)$, and the time for model checking $FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j)$ will be $|\mathcal{X}|$ to traverse all SCCs. Comparing to the classical algorithm presented in [32], Algorithm 1 has the same time complexity. However, experiment shows that our algorithm achieves much better performance comparing to the classical one. Furthermore, Algorithm 1 reduces the space complexity from $O(|\mathcal{X}| \times 2^{\ell(\phi)})$ to $O(|\mathcal{X}| + 2^{\ell(\phi)})$ for fairness in LTL(F, G).

B. Expressiveness of fairness in LTL(F, G)

We have presented an efficient algorithm to handle the fairness in LTL(F, G). The question then arises whether fair formulas in LTL(F, G) are expressive enough to encode all fair formulas in LTL? First, one can easily verify that the fairness LTL formula FG(aUb) is equivalent to FG(a ∨ b) ∨ GFb. Intuitively, eventually there is a looping path that satisfies aUb at every position is equivalent to that eventually there is a loop path that every state satisfies a ∨ b and there exists at least one state on the loop that satisfies b.

The transformation does not work in general. In the following, we show that $\phi = FG(a \vee X(b\text{U}c))$ cannot be expressed by any fairness in LTL(F, G). It is easy to see that $\phi$ is a fairness by Lemma 1. But it is impossible to find an equivalent formula in LTL(F, G) to represent $\phi$ since the order of states in SCC matters. We show that $\phi$ can not be represented as a fairness in LTL(F, G) by an example in the following.

For the trace $\eta = \{\{a\}\{a,c\}\}$ of $\mathcal{X}$ in Example 1, there are three kinds of letters, namely $\{a\}$, $\{\}$ and $\{a,c\}$. It is trivial that $\{a\} = a$ or $\{a,c\} = a$. For the word starting from letter $\{\}$, we have $\{\}\{a,c\} \cdots = X(b\text{U}c)$ since every letter $\{\}$ is directly followed by the letter $\{a,c\}$. Thus we conclude that $\eta \models \phi$.

By Theorem 1, suppose $\phi \equiv \bigvee_{i=1}^{m} (FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j))$ holds, we have $\eta \models \bigvee_{i=1}^{m} (FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j))$. In other words, there exists $1 \leq i \leq m$ such that $\eta \models FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j)$. Further, we conclude that for any $k \geq 0$, $\eta[k] = l_i$ and for every $l_{i,j}$, there is at least one out of letters $\{a\}$ and $\{a,c\}$ must satisfy $l_{i,j}$. As a result, $\{(a)\{a,c\}\}^\omega \models FGl_i \wedge (\wedge_{j=1}^{n_i} \text{GF}l_j)$, which follows that $\{(a)\{a,c\}\}^\omega \models \phi$.

Thus we conclude that fairness in LTL(F, G) is not powerful enough to express all fairness in LTL.

C. Fairness in LTL

In this subsection we deal with arbitrary fair formulas including those not expressible in LTL(F, G). More notations are needed. Given $B \subseteq S$ and $s \in B$, let $\mathcal{X}^B_s := (B,s,T_B,L_B)$ where $T_B = T \cap (B \times B)$ and $L_B : B \rightarrow 2^B$ such that $L_B(i) = L(i)$ for any $i \in B$. In other words, $\mathcal{X}^B_s$ is a sub-model of $\mathcal{X}$ where only states in $B$ and transitions between states in $B$ are kept. Moreover, let $LTL(U, X)$ denote the fragment of LTL only containing $U$ and $X$ modalities, namely, it is defined by the following grammar:

$$\psi ::= a | \neg a | \psi_1 \wedge \psi_2 | \psi_1 \vee \psi_2 | X \psi | \psi_1 U \psi_2.$$  

Formulas in LTL(U, X) are also known as co-safety in literature [20], [3], [21].

Similar as in Section III-A, we shall show that any fair formula can be transformed into an equivalent one, where all U and X modalities can be separated from F and G such that the innermost formulas are all in LTL(U, X). Such a transformation is syntactical as well, after which a formula in DNF will be obtained and moreover, each sub-formula can be handled individually by specific and efficient algorithms.

Theorem 4. Let $\phi \in LTL$ be a fair formula. Then, it can be transformed into the following equivalent formula, referred to also as its fair normal form:

$$fnf(\phi) := \bigvee_{i=1}^{m} (\phi_0 \land FG\phi_1 \land (\bigwedge_{j=2}^{n_i} \text{GF}\phi_j)),$$

where $\phi_0 \in LTL(F, G)$ and $\phi_j \in LTL(U, X)$ for all $1 \leq i \leq m$ and $1 \leq j \leq n_i$.

Example 4. Take $\phi = \neg(FG(a \lor X(b\text{U}c) \land F \neg b))$, then the fair normal form of $\neg \phi$ is $FGa \land (FG(a \lor X(b\text{U}c)) \land F \neg b)$.

As before, the model checking of an LTL formula $\phi$ is essentially reduced to the problem of finding a path in $\mathcal{X}$ satisfying $fnf(\neg \phi)$. Thus, we shall focus on the procedure of finding a path in $\mathcal{X}$ satisfying the given formula $\psi = \phi_0 \land FG\phi_1 \land GF\phi_2 \land \ldots \land GF\phi_n$ with $\phi_0 \in LTL(F, G)$ and $\phi_j \in LTL(U, X)$ for all $1 \leq j \leq n$. Note by Theorem 4, $fnf(\neg \phi)$ is a disjunction of such formulas.

We show how to optimize the procedure of finding a path satisfying $\psi$ or not. Case $\phi_1 \equiv 1$: hence the sub-formula $FG\phi_1$ can be omitted from $\psi$. The formal procedure for checking
Algorithm 2 The procedure accPath(ψ, ℋ) for checking whether there exists π ∈ Paths♭(ℋ) such that π |= ψ, where ψ = φ0 ∧ GFφ1 ∧ ... ∧ GFφn with φi ∈ LTL(F, G) and φj ∈ LTL(U, X) for each 2 ≤ j ≤ n, accPath(ψ, ℋ) returns True if a path satisfying ψ is found, and False otherwise.

1: procedure accPath(ψ, ℋ)
2:   Acc ← {all accepting SCCs with respect to φ0};
3: for all (B ∈ Acc) do
4:     A ← Ø;
5:     for all (2 ≤ j ≤ n and t ∈ B) do
6:         if (not (ℋB') |= −φj)) then
7:             A ← A ∪ {aj};
8:     if (A = {aj}2≤j≤n) then return True;
9: return False;

whether there exists a path in ℋ satisfying ψ is presented in Algorithm 2. As ψ is a fair formula, we can show that φ0 must be also a fair formula. Since φ0 ∈ LTL(F, G), a simple modification of Algorithm 1 can be applied to find all accepting SCCs with respect to φ0 in ℋ (line 2). If no accepting SCC can exist, terminate, as no path in ℋ can satisfy ψ; Otherwise, for each accepting SCC B and φj with 2 ≤ j ≤ n, add a fresh atomic proposition aj to A (line 7) iff there exists a state t ∈ B and π ∈ Paths♭(ℋB') such that π |= φj (line 6). This step can be done by launching classical algorithms: A path π in Paths♭(ℋB') exists such that π |= φj iff ℋB' does not satisfy −φj. Finally, an SCC B is accepted by ψ if at least one state in B is marked by aj for each 2 ≤ j ≤ n, namely, A = {aj}2≤j≤n (line 8).

The key point behind Algorithm 2 is that φj (2 ≤ j ≤ n) is in LTL(U, X), the corresponding Büchi automaton of which is terminal [4]. Therefore, once a path π satisfies φj, we can always find a finite fragment of π which suffices to conclude that π |= φj regardless of the remainder of π. In other words, whenever π |= φj, there exists i ≥ 0 such that (π|i·π') |= φj for any infinite path π'. Whenever Algorithm 2 returns True and finds an accepting B for ψ, we can construct a path satisfying ψ as follows:

1) Let π0 be a finite path in ℋ0 for any t such that all states in B appear in πt for at least once. Traversing all states in B is useful to witness φ0 ∈ LTL(F, G).
2) Continue from the last state of π1 and go to a state t2 by following any path, where t2 is a state in B, from which a path satisfying φ2 exists. Let π1' be the resultant path ending at t2. Expand π1' by following the path satisfying φ2 and stop whenever φ2 is for sure satisfied. Denote the resultant finite path by π2.
3) Keep extending π2 by repeating step 2 for each 3 ≤ j ≤ n. Let πn denote the resulting path.
4) Let πn' denote an arbitrary extension of πn such that t is a direct successor of the last state of πn', namely, (πn')♭ is a cyclic path in ℋn'.

By construction, it is easy to check that (πn')♭ |= ψ, which also shows the soundness and completeness of Algorithm 2.

Case φt ⊨ 1: we have to make sure that an accepting path also satisfies FGφt. For this purpose, we first transform FGφt to a Büchi automaton, denoted ⁰Sand then build a product model ℋ × ⁰Sand as in the classical algorithm. Let ⁰S and be a fresh atomic proposition such that ⁰S holds at a state iff the state is accepting in ℋ × ⁰Sand. The remainder of the procedure is similar as the case when φt ⊨ 1 except FGφt is replaced by GF⁰Sand the model under checked will be ℋ × ⁰Sand.

Example 5. Consider to verify ψ from Example 4 over ℋ in Example 1. As we already have the fair normal form for ¬φ by Example 4, we need to check whether there is a path π such that π |= FGa or π |= FG(a ∨ X(bUc)) ∧ GF¬b). Note that if we first check FGa, then we employ Algorithm 1 and terminate here with a counterexample (s0)a.

To further illustrate the algorithm, we continue with formula FG(a ∨ X(bUc)) ∧ GF¬b. Since a ∨ X(bUc) ⊨ 1, we construct an automaton ⁰Sand find an SCC accepted by GFaccepting ∧ GF¬b, in this case, say {s0, s1, s2}. We therefore construct a counterexample (s0s1s2)a. Detailed information of ⁰Sand the product can be found in [7].

a) Discussions: As mentioned before, formulas in LTL(U, X) are guarantee properties according to the classification in [6]. Their corresponding Büchi automata are terminal, for which specific and efficient algorithms exist [4]. By separating a fair formula, we can identify sub-formulas belonging to different fragments, each of which will be handled by specific and efficient algorithms.

D. General Formulas with Fairness Assumptions

In this subsection we show how the model checking problem for general LTL formulas with fairness assumptions can be accelerated by the specific algorithms for fair formulas introduced in the above subsections.

Given a fair formula φt and an LTL formula φ, the model checking problem of φ under the assumption φt reduces to checking whether ℋ |= (φt → φ) in order to make use of our specific algorithm for fairness, the procedure can be divided into two steps:

1) ¬φ is first transformed into a Büchi automaton, denoted ⁰Sand the product of ⁰Sand ℋ is then constructed, where all accepting states are marked by a fresh atomic proposition accepting;
2) Then ℋ |= (φt → φ) iff there is no path in the product satisfying φt ∧ GFaccepting. Note φt ∧ GFaccepting is still a fair formula, for which our efficient algorithm can be applied.

Note that we can specify some fairness assumption like FG(a ∨ X(bUc) ∧ F¬b)) in Example 4 which is not in LTL(F, G). Moreover, by making use of our algorithm for fairness, we gain some speed up in the model checking procedure if we choose to check FGa in the fair normal form as discussed in Example 5.
E. Formula Characterization

In this section, we specify some formula sets which are favourable to our algorithm as well as some formula sets for which our syntactic transformation leads to dramatic blow up of the formula length.

We first characterize some formula sets to which applying our transformation does not lead to dramatic growth of formula length, and we call them the fast LTL formulas.

Definition 4. Let $\Sigma_f$ be a subset of LTL formulas which is constructed by following rules. Then $\varphi_f, \varphi_e \in \Sigma_f$ where $\varphi_1 \in \text{LTL}(U, X)$.

\[
\begin{align*}
\varphi_0 & := \varphi_1 \mid F\varphi_0 \mid G\varphi_0 \mid \varphi_0 \land \varphi_0 \\
\varphi_f & := \varphi_0 \mid \varphi_f \lor \varphi_f \\
\varphi_e & := \varphi_1 \mid \varphi_e \lor \varphi_e \mid \varphi_e \land \varphi_e \mid F\varphi_e \mid G\varphi_e 
\end{align*}
\]

By induction on the structure of formulas defined in Definition 4 and similar analysis from Theorem 3, it is straightforward to show that:

Corollary 2. Let $\varphi_f(\varphi_e)$ be a formula defined in Definition 4 and $\varphi'_f (\varphi'_e)$ be the resulting formula after the transformation defined in Theorem 4. Then $|\varphi'_f| = O(|\varphi_f|)$. Similarly, we have $|\varphi'_e| = O(2^{|\varphi_e|})$.

In the following, we give the intuition why the transformation increase the formula length by the following example.

Example 6. Let

\[
\varphi = \psi_1 U \psi_2 = ((G F a_1 \land G F a_2) \lor \cdots \lor (G F a_{p-1} \land G F a_p)) \\
U ((G F b_1 \lor G F b_2) \land \cdots \land (G F b_{q-1} \lor G F b_q))
\]

Clearly, $|\varphi| = O(p + q)$. We need first get all F and G modalities out of the scope of U. To this end, by rules of ($\varphi_1 \land \varphi_2)U\varphi_3 \equiv \varphi_1 U \varphi_3 \lor \varphi_2 U \varphi_3$ and $\varphi_1 U (\varphi_2 \land \varphi_3) \equiv \varphi_1 U \varphi_2 \land \varphi_1 U \varphi_3$, it requires us to transform $\psi_1$ to CNF form and $\psi_2$ to DNF form. After that, we get a formula which is of size $O(2^{|\varphi|})$.

We remark that our transformation does not work when the formula contains W modalities, so we replace W with G and U modalities. As a result, it may increase the number of modalities after negating a formula. Take $\varphi = G F (\neg a \lor (\neg b \lor c))$ for example, after negating $\varphi$, it gives us $G F (a \lor ((\neg b \land c)W(b \land c)))$, which is equivalent to $G F (a \lor (\neg b \land c) \lor ((\neg b \land c)U(b \land c)))$. After applying the formula transformation, the resulting formula becomes $(G F (\neg b \land c) \lor G F a) \lor G F (\neg b \lor c)(U(b \land c))$. We notice that the reduction for W modality contributes to the growth of the formula length.

IV. Experiment

In this section we first illustrate briefly how our algorithm is implemented symbolically in NuSMV and then compare the experiment results with existing algorithms. NuSMV is a Symbolic Model Verifier extending the first BDD-based model checker SMV [5]. Compared to tools based on explicit representations, NuSMV is able to handle relatively more complex formulas [27], which is the main reason for choosing NuSMV in our experiment.

We implement our algorithm in NuSMV symbolically. The algorithm first decomposes a given formula syntactically to the specific form according to Theorems 1 and 4 and then uses the fair cycle detection algorithm proposed by Emerson and Lei [12] to find accepting SCCs. For instance, given a fair formula $\varphi \in \text{LTL}(F, G)$ such that $\text{fnf}(\varphi) = \bigvee_{i=1}^{m} (FG l_i \land (\bigwedge_{j=1}^{n_i} GF l_{i,j}))$, the fair cycle detection algorithm can be applied to determine whether there exists an SCC in $\mathcal{X}$ satisfying $FG l_i \land (\bigwedge_{j=1}^{n_i} GF l_{i,j})$ for some $1 \leq i \leq m$. By doing so, we avoid enumerating all SCCs one by one.

We adopt two well-known and scalable problems as our benchmarks: dining philosopher problem (PD) and binary semaphore protocol (BS). Their sizes are summarized in Table I, where “Size” refers to the number of reachable states for each model, PDx denotes the PD model with x philosophers, and similarly for BSx. All experiment results were obtained on a computer with an Intel(R) Core(TM) i7-2600 3.4GHz CPU running Ubuntu 14.04 LTS. We set time and memory limits to be 2 hours and 3 GB, respectively. The source code and several cases can be downloaded from http://iscasmc.ios.ac.cn/?page_id=984

We consider three categories of formulas.

A. Fair LTL($F, G$) formulas

The first category takes formulas often used in verification tasks. Specifically, for PD model we consider the following formula, saying that the first philosopher will eat eventually if no one will be starved (fairness assumption), namely, whenever a philosopher is ready, he/she will be able to eat eventually:

\[
\text{Spec}_1 = \left( \bigwedge_{i=1}^{n} (GF ready_i \rightarrow GF eat_i) \right) \rightarrow GF eat_1
\]

For BS model, we consider the following two formulas:

\[
\text{Spec}_2 = \left( \bigwedge_{i=1}^{n} (GF enter_i \rightarrow GF critical_i) \right) \rightarrow GF critical_1
\]

\[
\text{Spec}_3 = \left( \bigwedge_{i=1}^{n} (GF enter_i \rightarrow GF critical_i) \right) \rightarrow (\neg GF critical_1 \land \neg GF critical_3) \land GF critical_3
\]

$Spec_2$ denotes a similar specification as $Spec_1$, while $Spec_3$ requires that the second process entering the critical part before the first and third processes. Notice that all given fairness assumptions are simple formulas in LTL($F, G$).

In the following we write NuSMV to represent the automata-theoretic approach implemented in NuSMV. Table II
shows both the time and memory spent by our algorithm and NuSMV to check formulas in the first category on PD and BS models, where T-O and M-O denote “timeout” and “out-of-memory”, respectively.

The above assumptions in formulas Spec$_i$ (i = 1, 2, 3) are fair LTL($F$, $G$) formulas. In this case, our algorithm avoids the product construction entirely. From Table II, we can see that our algorithm outperforms NuSMV in almost all cases. In particular, our algorithm terminates in seconds for some cases, while NuSMV runs out of time or memory.

### B. Fair Pattern Formulas

We consider the second category of fair formulas generated by “genltl” – a tool of Spot library [11] to generate formulas of scalable patterns. These patterns and sample formulas are presented in Table III, where column “genltl arguments” denotes arguments used by “genltl” to generate corresponding formulas and $n$ the number of philosophers in PD or the number of processes in BS. In Table III and the following formulas, we use $a_i, b_i, \ldots$ as placeholders which will be replaced by proper atomic propositions during the experiment. To ease the presentation, we omit the details here. The time and memory usages of our algorithm and NuSMV to model check formulas in Table III are presented in Figure 2 where we mark by circles and triangles the running time and maximal memory consumption respectively. Each circle (triangle) corresponds to the time (memory) consumption of our algorithm and NuSMV. The coordinate values of the $y$ axis and $x$ axis are the corresponding experimental results for NuSMV and our algorithm respectively. We fill the marks with red color when it runs out of time and with blue color for memory out. For all cases, our algorithm consumes a negligible amount of time and memory comparing to NuSMV, which runs out of time and memory in many cases. All points above the main diagonal indicate that our algorithm is faster or consumes less memory than NuSMV, which is the case for all large examples. Moreover, we tried Spin [18] for generating the automata for formulas in Table III, it can not return the answer within 30 minutes for a single formula. We note that we have run experimental results on more generated pattern formulas and observe very similar results as the one presented here.

We remark that all formulas in Table III are simple formulas, actually a subset of LTL($F$, $G$), which can be converted to simple Streett/Rabin fairness conditions. We expect some speedup if optimisations [2] for treating simple fairness are implemented in NuSMV. Our algorithm for fairness in LTL($F$, $G$) follows the same idea except that we first conduct a formula transformation so that we can handle fairness like $GF(a \land Gb)$. More importantly, our treatment of fair LTL($F$, $G$) formulas is also an essential preparation step of handling general LTL fair formulas, as considered below.

### C. General LTL Fairness

We consider some general fair LTL formulas, summarized in Table IV. These formulas are often adopted to evaluate performance of an LTL model checker or planner in the literature; see for instance [31], [14], [23], [13]. The time consumption for checking these formulas is presented in Table V and VII, while the memory consumption is shown in Table VI and VIII. From these results we observe similar phenomena as before for most cases except for “p10”, “p11”, and “p15”, where our algorithm uses more time and/or memory than NuSMV for certain cases, particularly when “p10” and PD models are concerned. We explain such performance differences in details in the following.
As mentioned before, our algorithm relies on syntactical transformations in Theorems 1 and 4. These transformations can decompose a fair formula into smaller subformulas, whose corresponding Büchi automata are usually much smaller than the automaton of the original formula. This is the main reason that our algorithm achieves much better performance than the classical algorithm for most of the instances. However, the syntactic transformations adopted in Theorem 1 and 4 may cause exponential blow-up for certain cases; for instance formulas whose negations are in form of “p10” and “p11”. In order to push all $F$ and $G$ modalities in front of $U$ modality, our transformation may need to transform back and forth between CNF and DNF of some formulas, especially for those formulas where $F$, $G$, and $U$ are alternatively nested for many times. Therefore, for such formulas, the syntactic transformation may be time-consuming and result in formulas of exponentially longer than the original ones.

We note that many formulas we take from the literature are characterized by Definition 4, and transforming the negation of these formulas only leads to a linear increase in the formula length. The exceptions are “p5”, “p8”, “p10”, “p11”, and “p13-p16”. It is worthwhile to mention that even though for formulas such that the transformations result in formulas of exponential length, our algorithm is not necessarily slower than NuSMV, as the corresponding Büchi automata may be exponentially large as well; for instance “p8” and “p10”. Finally, our algorithm outperforms NuSMV for “p14” and its negation “p16”; it is faster for “p13” and is only slightly slower than NuSMV for its negation “p15” for one case.

V. CONCLUSION

We presented a novel model checking algorithm for formulas in LTL with fairness assumptions. Our algorithm does not follow the automata-theoretic approach completely but tries to decompose a fair formula into several sub-formulas, each of which can be handled by specific and efficient algorithms. We showed by experiment that our algorithm in many cases exceeds NuSMV up to several orders of magnitudes.

### TABLE IV

**Formula patterns used in our experiment**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>p5</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p6</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p7</td>
<td>$\neg a_i \land X c_i \land (F G c_i \lor X \exists X \neg X c_i)$</td>
</tr>
<tr>
<td>p8</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p9</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p10</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p11</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p12</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p13</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p14</td>
<td>$\forall X \neg G (a_i \lor \exists X b_i \land (F G c_i \lor X \exists X \neg X c_i))$</td>
</tr>
<tr>
<td>p15</td>
<td>negations of formulas in p13</td>
</tr>
<tr>
<td>p16</td>
<td>negations of formulas in p14</td>
</tr>
</tbody>
</table>

### TABLE V

**Time Usage (second)**

<table>
<thead>
<tr>
<th>model</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>p8</th>
<th>p9</th>
<th>p10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ours</td>
<td>NuSMV</td>
<td>Ours</td>
<td>NuSMV</td>
<td>Ours</td>
<td>NuSMV</td>
</tr>
<tr>
<td>PD6</td>
<td>0.11</td>
<td>T-O</td>
<td>0.17</td>
<td>T-O</td>
<td>0.27</td>
<td>2215.90</td>
</tr>
<tr>
<td>PD9</td>
<td>1.09</td>
<td>M-O</td>
<td>0.38</td>
<td>M-O</td>
<td>1.74</td>
<td>T-O</td>
</tr>
<tr>
<td>PD12</td>
<td>41.45</td>
<td>M-O</td>
<td>12.79</td>
<td>M-O</td>
<td>129.50</td>
<td>M-O</td>
</tr>
<tr>
<td>BS4</td>
<td>0.07</td>
<td>10.96</td>
<td>0.23</td>
<td>26.67</td>
<td>0.08</td>
<td>116.65</td>
</tr>
<tr>
<td>BS8</td>
<td>0.14</td>
<td>M-O</td>
<td>0.02</td>
<td>M-O</td>
<td>0.03</td>
<td>T-O</td>
</tr>
<tr>
<td>BS12</td>
<td>4.55</td>
<td>M-O</td>
<td>0.25</td>
<td>M-O</td>
<td>0.95</td>
<td>M-O</td>
</tr>
<tr>
<td>BS16</td>
<td>377.33</td>
<td>M-O</td>
<td>1.07</td>
<td>M-O</td>
<td>0.41</td>
<td>M-O</td>
</tr>
</tbody>
</table>

### TABLE VI

**Memory Usage (MB)**

<table>
<thead>
<tr>
<th>model</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>p8</th>
<th>p9</th>
<th>p10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ours</td>
<td>NuSMV</td>
<td>Ours</td>
<td>NuSMV</td>
<td>Ours</td>
<td>NuSMV</td>
</tr>
<tr>
<td>PD6</td>
<td>14.10</td>
<td>T-O</td>
<td>13.26</td>
<td>T-O</td>
<td>14.63</td>
<td>507.14</td>
</tr>
<tr>
<td>PD9</td>
<td>55.77</td>
<td>M-O</td>
<td>35.61</td>
<td>M-O</td>
<td>48.17</td>
<td>M-O</td>
</tr>
<tr>
<td>PD12</td>
<td>98.70</td>
<td>M-O</td>
<td>76.07</td>
<td>M-O</td>
<td>92.81</td>
<td>M-O</td>
</tr>
<tr>
<td>BS4</td>
<td>11.77</td>
<td>62.47</td>
<td>11.70</td>
<td>63.75</td>
<td>12.01</td>
<td>61.32</td>
</tr>
<tr>
<td>BS8</td>
<td>16.44</td>
<td>M-O</td>
<td>12.16</td>
<td>M-O</td>
<td>12.68</td>
<td>M-O</td>
</tr>
<tr>
<td>BS12</td>
<td>74.95</td>
<td>M-O</td>
<td>12.94</td>
<td>M-O</td>
<td>14.28</td>
<td>M-O</td>
</tr>
<tr>
<td>BS16</td>
<td>423.19</td>
<td>M-O</td>
<td>14.39</td>
<td>M-O</td>
<td>16.84</td>
<td>M-O</td>
</tr>
</tbody>
</table>
TABLE VII

<table>
<thead>
<tr>
<th>model</th>
<th>p11</th>
<th>p12</th>
<th>p13</th>
<th>p14</th>
<th>p15</th>
<th>p16</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD9</td>
<td>M-O</td>
<td>M-O</td>
<td>M-O</td>
<td>M-O</td>
<td>M-O</td>
<td>T-O</td>
</tr>
<tr>
<td>PD12</td>
<td>M-O</td>
<td>M-O</td>
<td>77.01</td>
<td>M-O</td>
<td>72.20</td>
<td>M-O</td>
</tr>
<tr>
<td>BS14</td>
<td>15.30</td>
<td>0.26</td>
<td>0.29</td>
<td>3.49</td>
<td>0.14</td>
<td>13.13</td>
</tr>
<tr>
<td>BS18</td>
<td>T-O</td>
<td>280.91</td>
<td>0.03</td>
<td>T-O</td>
<td>0.03</td>
<td>M-O</td>
</tr>
<tr>
<td>BS12</td>
<td>M-O</td>
<td>M-O</td>
<td>0.05</td>
<td>M-O</td>
<td>0.05</td>
<td>M-O</td>
</tr>
<tr>
<td>BS16</td>
<td>M-O</td>
<td>M-O</td>
<td>1.00</td>
<td>M-O</td>
<td>0.07</td>
<td>M-O</td>
</tr>
</tbody>
</table>

TABLE VIII

<table>
<thead>
<tr>
<th>model</th>
<th>p11</th>
<th>p12</th>
<th>p13</th>
<th>p14</th>
<th>p15</th>
<th>p16</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD9</td>
<td>M-O</td>
<td>M-O</td>
<td>50.61</td>
<td>M-O</td>
<td>36.09</td>
<td>M-O</td>
</tr>
<tr>
<td>PD12</td>
<td>M-O</td>
<td>M-O</td>
<td>90.92</td>
<td>M-O</td>
<td>89.87</td>
<td>M-O</td>
</tr>
<tr>
<td>BS14</td>
<td>95.83</td>
<td>40.36</td>
<td>12.04</td>
<td>60.96</td>
<td>12.08</td>
<td>51.46</td>
</tr>
<tr>
<td>BS18</td>
<td>T-O</td>
<td>143.0</td>
<td>12.71</td>
<td>T-O</td>
<td>12.60</td>
<td>T-O</td>
</tr>
<tr>
<td>BS12</td>
<td>M-O</td>
<td>M-O</td>
<td>14.07</td>
<td>M-O</td>
<td>13.70</td>
<td>T-O</td>
</tr>
<tr>
<td>BS16</td>
<td>M-O</td>
<td>M-O</td>
<td>16.86</td>
<td>M-O</td>
<td>15.42</td>
<td>M-O</td>
</tr>
</tbody>
</table>

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grants No. 61532019, 61472473, 61472406), by the National 973 Program (No. 2014CB340701), by the CDZ project CAP (GZ 1023), and by the CAS/SSAFEA International Partnership Program for Creative Research Teams.

REFERENCES
