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: $\bigwedge_{1\leq i\leq n}(\mathsf{GF}\mathit{enter}_i \rightarrow \mathsf{GF}\mathit{critical}_i)$ (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 $\mathsf{FG}(a \vee \mathsf{F}b)$. Moreover, we extend the notion of fairness assumptions to full LTL formulas, which allows us to specify some fairness assumption like "a and $\mathsf{X}(b\mathsf{U}c)$ 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 $\xi = x_0 x_1 \ldots \in X^*$ with $X^* = \cup_{i\geq 0} X^i$ 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 ξ' for infinite times. Let $\xi[i] = x_i$ denote the $(i+1)$-th element on ξ if it exists. We shall write $\xi|^i$ to denote the prefix of ξ ending at the $(i+1)$-th element, while $\xi|_i$ the suffix of ξ 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 ξ' to the end of ξ.

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:

$$\varphi, \psi ::= a \mid \neg a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \mid \varphi_1 \mathsf{W} \varphi_2$$

where $a \in AP$ and φ, ψ, φ_1, and φ_2 range over LTL formulas. As usual, we introduce some abbreviations: $\mathbf{1} = a \vee \neg a$ and $\mathbf{0} = a \wedge \neg a$ denote *True* and *False* respectively, while $\mathsf{F}\varphi = \mathbf{1}\mathsf{U}\varphi$ (eventually φ), $\mathsf{G}\varphi = \varphi \mathsf{W} \mathbf{0}$ (always φ), and $(\varphi_1 \to \varphi_2) = (\neg\varphi_1 \vee \varphi_2)$. Let $l, l_1, l_2, \ldots$ range over *propositional formulas*, i.e., formulas defined by: $l ::= a \mid \neg a \mid l_1 \wedge l_2 \mid l_1 \vee l_2$.

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

$$\begin{array}{rcl}
\rho \models a & \text{iff} & a \in \rho[0] \\
\rho \models \mathsf{X}\varphi & \text{iff} & \rho|_1 \models \varphi \\
\rho \models \varphi_1 \mathsf{U} \varphi_2 & \text{iff} & \exists i \geq 0.(\rho|_i \models \varphi_2 \wedge \forall 0 \leq j < i.\rho|_j \models \varphi_1) \\
\rho \models \varphi_1 \mathsf{W} \varphi_2 & \text{iff} & (\forall i \geq 0.\rho|_i \models \varphi_1) \vee (\rho \models \varphi_1 \mathsf{U} \varphi_2)
\end{array}$$

All other connectives are defined in a standard way. For formulas φ and ψ, we say that φ and ψ 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 $\neg(\mathsf{X}\psi) \equiv \mathsf{X}(\neg\psi)$ and the following *duality laws*:

$$\begin{array}{rcl}
\neg(\varphi_1 \mathsf{U} \varphi_2) & \equiv & (\varphi_1 \wedge \neg\varphi_2) \mathsf{W} (\neg\varphi_1 \wedge \neg\varphi_2) \\
\neg(\varphi_1 \mathsf{W} \varphi_2) & \equiv & (\varphi_1 \wedge \neg\varphi_2) \mathsf{U} (\neg\varphi_1 \wedge \neg\varphi_2)
\end{array}$$

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 φ is a* fairness *iff for any $\rho \in (2^{AP})^\omega$,*

1) *the set of sequences satisfying φ 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 φ is closed under prefixes, i.e., $\rho \models \varphi$ implies $\rho_1 \cdot \rho \models \varphi$ for any $\rho_1 \in (2^{AP})^*$.*

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. *φ is a fairness iff $\varphi \equiv \mathsf{G}\varphi$ and $\varphi \equiv \mathsf{F}\varphi$.*

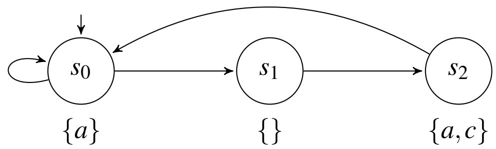

Fig. 1. An example of Kripke structure

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 $\mathsf{F}a \vee \mathsf{G}\neg a$ is equivalent to $\mathsf{GF}(\mathsf{F}a \vee \mathsf{G}\neg a)$.

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

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

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

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

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

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

Proof. Since φ is a fairness, $\varphi \equiv \mathsf{G}\varphi$. Thus $\pi \models \varphi$ implies $\pi|_j \models \varphi$ for any index j. On the other hand, $\pi|_j \models \varphi$ implies $\pi \models \mathsf{F}\varphi$. Then we conclude $\pi \models \varphi$ by $\mathsf{F}\varphi \equiv \varphi$. □

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 φ, $\mathcal{K} \models \varphi$ means that for all infinite paths π starting from initial state $\bar{s}$, $\pi \models \varphi$. Conversely, if $\neg(\mathcal{K} \models \varphi)$, then there exists an infinite path π 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 $fnf(\neg\varphi)$, which has the form $\vee_{i=1}^{m} \varphi_i$, with each φ_i being a fair formula. Then, the problem reduces to checking whether there exists an infinite path π such that $\pi \models \varphi_i$. In other words, whether there exists an SCC B *satisfying* for φ_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 $\mathsf{LTL(F, G)}$, then we extend the algorithm to deal with general fairness. Finally, we handle all LTL formulas with fairness assumptions.

A. Fairness in $\mathsf{LTL(F, G)}$

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

$$\varphi ::= a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi.$$

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

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

$$fnf(\varphi) := \bigvee_{i=1}^{m} \left(\mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{ij}) \right)$$

where l_i and $l_{i,j}$ are propositional formulas.

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

Example 2. *Take $\varphi = \neg(\mathsf{FG}(a \vee (\mathsf{F}b \wedge \mathsf{G}c)))$ for example, the fair normal form of $\neg\varphi$ is $fnf(\neg\varphi) = \mathsf{FG}a \vee (\mathsf{FG}c \wedge \mathsf{GF}b)$. Consider model checking the Kripke structure $\mathcal{K}$ in Example 1 against fair formula φ. 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 $\mathsf{FG}a$ or $\mathsf{FG}c \wedge \mathsf{GF}b$. Consider fair formula $\mathsf{FG}a$, 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 φ 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 $\mathsf{F}a \vee \mathsf{G}\neg a$. 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 $\mathsf{GF}a \vee \mathsf{FG}\neg a$.

Given a fairness $\varphi \in \mathsf{LTL}(\mathsf{F},\mathsf{G})$, 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 φ. 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 $\vee$ and $\wedge$.

Suppose $\varphi_1, \varphi_2 \in \mathrm{LTL}$, our goal is pushing GF and FG inside such that they appear only before propositional formulas. We consider the following four cases:

1) $\mathsf{GFG} \equiv \mathsf{FG}, \mathsf{GFF} \equiv \mathsf{GF}, \mathsf{FGG} \equiv \mathsf{FG}$ and $\mathsf{FGF} \equiv \mathsf{GF}$ 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 φ.
2) $\mathsf{GF}(\varphi_1 \vee \varphi_2) \equiv \mathsf{GF}\varphi_1 \vee \mathsf{GF}\varphi_2$ and $\mathsf{FG}(\varphi_1 \wedge \varphi_2) \equiv \mathsf{FG}\varphi_1 \wedge \mathsf{FG}\varphi_2$ hold since GF and FG are distributive over $\vee$ and $\wedge$ operator respectively.
3) $\mathsf{GF}(\varphi_1 \wedge \mathsf{F}\varphi_2) \equiv \mathsf{GF}\varphi_1 \wedge \mathsf{GF}\varphi_2$, $\mathsf{FG}(\varphi_1 \vee \mathsf{G}\varphi_2) \equiv \mathsf{FG}\varphi_1 \vee \mathsf{FG}\varphi_2$, $\mathsf{GF}(\varphi_1 \wedge \mathsf{G}\varphi_2) \equiv \mathsf{GF}\varphi_1 \wedge \mathsf{FG}\varphi_2$ and $\mathsf{FG}(\varphi_1 \vee \mathsf{F}\varphi_2) \equiv \mathsf{FG}\varphi_1 \vee \mathsf{GF}\varphi_2$. 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) $\mathsf{GF}(\varphi_1 \wedge \varphi_2)$ and $\mathsf{FG}(\varphi_1 \vee \varphi_2)$. This is the most challenging part since GF (FG) is not distributive over $\wedge$ ($\vee$) 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 $\mathsf{FG}(a \vee (\mathsf{F}b \wedge \mathsf{G}c))$, we transform $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ 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 *fnf* operator via an example as follows:

Example 3. *Let* $\varphi = \mathsf{FG}(a \vee (\mathsf{F}b \wedge \mathsf{G}c))$. *We show how to flatten* φ *step by step.*

- *Add* GF *in front of* φ *which gives us* φ *since* $\mathsf{GFFG} \equiv \mathsf{FG}$*;*
- *Since* φ *is an instance of case 4), we first transform* $a \vee (\mathsf{F}b \wedge \mathsf{G}c)$ *into a CNF, which results in* $\mathsf{FG}((a \vee \mathsf{F}b) \wedge (a \vee \mathsf{G}c))$*;*
- *According to case 2),* FG *is distributive over* $\wedge$ *operator, we therefore directly push* FG *inside, which gives us* $\mathsf{FG}(a \vee \mathsf{F}b) \wedge \mathsf{FG}(a \vee \mathsf{G}c)$*;*
- *The resulting formula is an instance of case* 3)*, we get* $(\mathsf{FG}a \vee \mathsf{GF}b) \wedge (\mathsf{FG}a \vee \mathsf{FG}c)$ *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* $\mathsf{FG}a \vee (\mathsf{FG}c \wedge \mathsf{GF}b)$ *of* φ*.*

Intuitively, it means whenever $\pi \models \varphi$*, it must be the case that* π *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.*

Algorithm 1 The procedure `fairMC` for checking whether $\mathcal{K} \models \varphi$, where φ is a fair formula in $\mathsf{LTL}(\mathsf{F},\mathsf{G})$. $\mathtt{fairMC}(\varphi, \mathcal{K})$ returns *True* if $\mathcal{K} \models \varphi$, and *False* otherwise.

procedure $\mathtt{fairMC}(\varphi, \mathcal{K})$
  $\mathit{fnf}(\neg\varphi) \equiv \vee_{i=1}^{m} \varphi_i = \vee_{i=1}^{m} (\mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j}))$;
  for all $(1 \leq i \leq m)$ **do**
    $B \leftarrow \{s \in S \mid s \models l_i\}$;
    if $B \neq \emptyset$ **then**
      for all (SCC $B' \subseteq B$) **do**
        if (B' is accepting for φ_i) **then**
          return *False*;
  return *True*;

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* $\varphi = \mathsf{FG}l \wedge (\bigwedge_{j=1}^{m} \mathsf{GF}l_j)$ *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* φ*.*

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

Theorem 2. *For any* $\varphi = \mathsf{FG}l \wedge (\bigwedge_{j=1}^{m} \mathsf{GF}l_j)$*, there exists an infinite path* π *in* $\mathcal{K}$ *such that* $\pi \models \varphi$ *iff there exists a reachable SCC B such that B is accepting for* φ*.*

Proof. $\Rightarrow$ Since $\mathcal{K}$ is finite, for any $\pi \in \mathit{Paths}^{\omega}(\mathcal{K})$, 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 φ is a fairness. For convenience, let $\pi_1 = \pi|_k$. Let B_1 be the set of states on π_1. Obviously, all states in B_1 are connected since all states will be visited by infinite times. $\pi_1 \models \mathsf{FG}l$ means $s \models l$ for each $s \in B_1$ and $\pi \models \mathsf{GF}l_j$ 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 φ.

$\Leftarrow$ This direction is trivial, since we can always construct a path π_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 π_1 which starts from the initial state and reaches the first state of π_2. Let $\pi = \pi_1 \cdot \pi_2$. Obviously $\pi \models \mathsf{FG}l \wedge (\bigwedge_{j=1}^{m} \mathsf{GF}l_j)$, 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 φ in $\mathsf{LTL}(\mathsf{F},\mathsf{G})$. For this, the algorithm first syntactically transforms $\neg\varphi$ into an equivalent formula of the form

$\bigvee_{i=1}^{m} \mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j})$. For each $1 \le i \le m$, we then try to find an *accepting SCC* B' such that all states in B' satisfy l_i and at least one state in B' satisfies $l_{i,j}$ for each $1 \le j \le n_i$. In case an accepting SCC is found, there exists a path in $\mathscr{K}$ violating φ, hence $\mathscr{K} \not\models \varphi$; otherwise we conclude that $\mathscr{K} \models \varphi$.

By Theorem 2, the soundness and completeness of Algorithm 1 immediately follows.

Note the checking of whether a SCC B' is accepted by φ_i in line 7 of Algorithm 1 can be done easily in linear time with respect to $|B'|$. Let $|\mathscr{K}|$ denote the size of the given model, i.e., the total number of states and transitions. The complexity of Algorithm 1 is shown in the following theorem.

Theorem 3. *Algorithm 1 runs in time* $\mathscr{O}(|\mathscr{K}| \times 2^{|\varphi|})$ *and in space* $\mathscr{O}(|\mathscr{K}| + |\varphi| \times 2^{|\varphi|})$.

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 $\mathscr{O}(2^{|\varphi|})$ 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 $n_1 + n_2 \in \mathscr{O}(|\varphi|)$. We then have 2^{n_1} options for $\mathsf{FG}l$ formulas and 2^{n_2} for $\wedge_k \mathsf{GF}l_k$ since the number of l_k is n_2, so we will at most have $2^{n_1+n_2}$ formulas have the form $\mathsf{FG}l \wedge (\bigwedge_{k=1}^{m} \mathsf{GF}l_k)$ and each formula of that form at most has n_2+1 propositional formulas, which means that formula length can be $|\varphi| \times 2^{\mathscr{O}(|\varphi|)}$ in the worst case. That is, we will at most have $2^{n_1+n_2}$ formulas with the form $\mathsf{FG}l \wedge (\bigwedge_{k=1}^{m} \mathsf{GF}l_k)$, and the time for model checking $\mathsf{FG}l \wedge (\bigwedge_{k=1}^{m} \mathsf{GF}l_k)$ will be $|\mathscr{K}|$ 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 $\mathscr{O}(|\mathscr{K}| \times 2^{\mathscr{O}(|\varphi|)})$ to $\mathscr{O}(|\mathscr{K}| + 2^{\mathscr{O}(|\varphi|)})$ for fairness in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$.

B. Expressiveness of fairness in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$

We have presented an efficient algorithm to handle the fairness in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$. The question then arises whether fair formulas in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$ are expressive enough to encode all fair formulas in LTL? First, one can easily verify that the fairness LTL formula $\mathsf{FG}(a\mathsf{U}b)$ is equivalent to $\mathsf{FG}(a \vee b) \wedge \mathsf{GF}b$. Intuitively, eventually there is a looping path that satisfies $a\mathsf{U}b$ at every position is equivalent to that eventually there is a loop path that every state satisfies $a \vee b$ 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 $\varphi = \mathsf{FG}(a \vee \mathsf{X}(b\mathsf{U}c))$ can not be expressed by any fairness in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$. It is easy to see that φ is a fairness by Lemma 1. But it is impossible to find an equivalent formula in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$ to represent φ since the order of states in SCC matters. We show that φ can not be represented as a fairness in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$ by an example in the following.

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

By Theorem 1, suppose $\varphi \equiv \bigvee_{i=1}^{m}(\mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j}))$ holds, we have $\eta \models \bigvee_{i=1}^{m}(\mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j}))$. In other words, there exists $1 \le i \le m$ such that $\eta \models \mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j})$. Further, we conclude that for any $k \ge 0$, $\eta|_k \models l_i$ 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, $(\{a\}\{a,c\}\{\})^{\omega} \models \mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j})$, which follows that $(\{a\}\{a,c\}\{\})^{\omega} \models \varphi$. Contradiction.

Thus we conclude that fairness in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$ 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 $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$. More notations are needed. Given $B \subseteq S$ and $s \in B$, let $\mathscr{K}_B^s := (B, s, \mathsf{T}_B, L_B)$ where $\mathsf{T}_B = \mathsf{T} \cap (B \times B)$ and $L_B : B \to 2^{AP}$ such that $L_B(t) = L(t)$ for any $t \in B$. In other words, $\mathscr{K}_B^s$ is a sub-model of $\mathscr{K}$ where only states in B and transitions between states in B are kept. Moreover, let $\mathsf{LTL}(\mathsf{U}, \mathsf{X})$ denote the fragment of LTL only containing U and X modalities, namely, it is defined by the following grammar:

$$\varphi ::= a \mid \neg a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2.$$

Formulas in $\mathsf{LTL}(\mathsf{U}, \mathsf{X})$ 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 $\mathsf{LTL}(\mathsf{U}, \mathsf{X})$. 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 $\varphi \in LTL$ be a fair formula. Then, it can be transformed into the following equivalent formula, referred to also as its fair normal form:*

$$fnf(\varphi) := \bigvee_{i=1}^{m} \left(\varphi_{i0} \wedge \mathsf{FG}\varphi_{i1} \wedge (\bigwedge_{j=2}^{n_i} \mathsf{GF}\varphi_{ij}) \right)$$

where $\varphi_{i0} \in \mathsf{LTL}(\mathsf{F}, \mathsf{G})$ and $\varphi_{ij} \in \mathsf{LTL}(\mathsf{U}, \mathsf{X})$ for all $1 \le i \le m$ and $1 \le j \le n_i$.

Example 4. *Take $\varphi = \neg(\mathsf{FG}(a \vee (\mathsf{X}(b\mathsf{U}c) \wedge \mathsf{F}\neg b)))$, then the fair normal form of $\neg\varphi$ is $\mathsf{FG}a \vee (\mathsf{FG}(a \vee \mathsf{X}(b\mathsf{U}c)) \wedge \mathsf{GF}\neg b)$.*

As before, the model checking of an LTL formula φ is essentially reduced to the problem of finding a path in $\mathscr{K}$ satisfying $fnf(\neg\varphi)$. Thus, we shall focus on the procedure of finding a path in $\mathscr{K}$ satisfying the given formula $\psi = \varphi_0 \wedge \mathsf{FG}\varphi_1 \wedge \mathsf{GF}\varphi_2 \wedge \ldots \wedge \mathsf{GF}\varphi_n$ with $\varphi_0 \in \mathsf{LTL}(\mathsf{F}, \mathsf{G})$ and $\varphi_j \in \mathsf{LTL}(\mathsf{U}, \mathsf{X})$ for all $1 \le j \le n$. Note by Theorem 4, $fnf(\neg\varphi)$ is a disjunction of such formulas.

We show how to optimize the procedure of finding a path satisfying ψ or not. Case $\varphi_1 \equiv \mathbf{1}$: hence the sub-formula $\mathsf{FG}\varphi_1$ can be omitted from ψ. The formal procedure for checking

Algorithm 2 The procedure $\texttt{accPath}(\psi, \mathscr{K})$ for checking whether there exists $\pi \in Paths^{\omega}(\mathscr{K})$ such that $\pi \models \psi$, where $\psi = \varphi_0 \wedge \mathsf{GF}\varphi_2 \wedge \ldots \wedge \mathsf{GF}\varphi_n$ with $\varphi_0 \in \mathsf{LTL}(\mathsf{F}, \mathsf{G})$ and $\varphi_j \in \mathsf{LTL}(\mathsf{U}, \mathsf{X})$ for each $2 \leq j \leq n$. $\texttt{accPath}(\psi, \mathscr{K})$ returns *True* if a path satisfying ψ is found, and *False* otherwise.

1: **procedure** $\texttt{accPath}(\psi, \mathscr{K})$
2: $\quad$ $Acc \leftarrow$ {all accepting SCCs with respect to φ_0};
3: $\quad$ **for all** ($B \in Acc$) **do**
4: $\qquad$ $A \leftarrow \emptyset$;
5: $\qquad$ **for all** ($2 \leq j \leq n$ and $t \in B$) **do**
6: $\qquad\quad$ **if** (not ($\mathscr{K}_B^t \models \neg\varphi_j$)) **then**
7: $\qquad\qquad$ $A \leftarrow A \cup \{a_j\}$;
8: $\qquad$ **if** ($A = \{a_j\}_{2 \leq j \leq n}$) **then return** *True*;
9: $\quad$ **return** *False*;

whether there exists a path in $\mathscr{K}$ satisfying ψ is presented in Algorithm 2. As ψ is a fair formula, we can easily show that φ_0 must be also a fair formula. Since $\varphi_0 \in \mathsf{LTL}(\mathsf{F}, \mathsf{G})$, a simple modification of Algorithm 1 can be applied to find all accepting SCCs with respect to φ_0 in $\mathscr{K}$ (line 2). If no accepting SCC exists, we can terminate, as no path in $\mathscr{K}$ can satisfy φ; Otherwise, for each accepting SCC B and φ_j with $2 \leq j \leq n$, add a fresh atomic proposition a_j to A (line 7) iff there exists a state $t \in B$ and $\pi \in Paths^{\omega}(\mathscr{K}_B^t)$ such that $\pi \models \varphi_j$ (line 6). This step can be done by launching classical algorithms: A path $\pi \in Paths^{\omega}(\mathscr{K}_B^t)$ exists such that $\pi \models \varphi_j$ iff $\mathscr{K}_B^t$ does not satisfy $\neg\varphi_j$. Finally, an SCC B is accepted by ψ if at least one state in B is marked by a_j for each $2 \leq j \leq n$, namely, $A = \{a_j\}_{2 \leq j \leq n}$ (line 8).

The key point behind Algorithm 2 is that φ_j ($2 \leq j \leq n$) is in $\mathsf{LTL}(\mathsf{U}, \mathsf{X})$, the corresponding Büchi automaton of which is terminal [4]. Therefore, once a path π satisfies φ_j, we can always find a finite fragment of π which suffices to conclude that $\pi \models \varphi_j$ regardless of the remainder of π. In other words, whenever $\pi \models \varphi_j$, there exists $i \geq 0$ such that $(\pi|^i \cdot \pi') \models \varphi_j$ 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 π_1 be a finite path in $\mathscr{K}_B^t$ for any t such that all states in B appear in π_1 for at least once. Traversing all states in B is useful to witness $\varphi_0 \in \mathsf{LTL}(\mathsf{F}, \mathsf{G})$.
2) Continue from the last state of π_1 and go to a state t_2 by following any path, where t_2 is a state in B, from which a path satisfying φ_2 exists. Let π_1' be the resultant path ending at t_2. Expand π_1' by following the path satisfying φ_2 and stop whenever φ_2 is for sure satisfied. Denote the resultant finite path by π_2.
3) Keep extending π_2 by repeating step 2 for each $3 \leq j \leq n$. Let π_n denote the resulting path.
4) Let π_n' denote an arbitrary extension of π_n such that t is a direct successor of the last state of π_n', namely, $(\pi_n')^{\omega}$ is a cyclic path in $\mathscr{K}_B^t$.

By construction, it is easy to check that $(\pi_n')^{\omega} \models \psi$, which also shows the soundness and completeness of Algorithm 2.

Case $\varphi_1 \not\equiv \mathbf{1}$: we have to make sure that an accepting path also satisfies $\mathsf{FG}\varphi_1$. For this purpose, we first transform $\mathsf{FG}\varphi_1$ to a Büchi automaton, denoted $\mathscr{A}_1$, and then build a product model $\mathscr{K} \times \mathscr{A}_1$ as in the classical algorithm. Let a_1 be a fresh atomic proposition such that a_1 holds at a state iff the state is accepting in $\mathscr{K} \times \mathscr{A}_1$. The remainder of the procedure is similar as the case when $\varphi_1 \equiv \mathbf{1}$ except $\mathsf{FG}\varphi_1$ is replaced by $\mathsf{GF}a_1$ in ψ and the model under checked will be $\mathscr{K} \times \mathscr{A}_1$.

Example 5. *Consider to verify φ from Example 4 over $\mathscr{K}$ in Example 1. As we already have the fair normal form for $\neg\varphi$ by Example 4, we need to check whether there is a path π such that $\pi \models \mathsf{FG}a$ or $\pi \models \mathsf{FG}(a \vee \mathsf{X}(b\mathsf{U}c)) \wedge \mathsf{GF}\neg b$). Note that if we first check $\mathsf{FG}a$, then we employ Algorithm 1 and terminate here with a counterexample $(s_0)^{\omega}$.*

*To further illustrate the algorithm, we continue with formula $\mathsf{FG}(a \vee \mathsf{X}(b\mathsf{U}c)) \wedge \mathsf{GF}\neg b$. Since $a \vee \mathsf{X}(b\mathsf{U}c) \not\equiv \mathbf{1}$, we construct an automaton $\mathscr{A}$ for $\mathsf{FG}(a \vee \mathsf{X}(b\mathsf{U}c))$ with Büchi accepting condition GFaccepting where accepting is a new atomic proposition. Then we construct the product of $\mathscr{K}$ and $\mathscr{A}$ and find an SCC accepted by $\mathsf{GF}accepting \wedge \mathsf{GF}\neg b$, in this case, say $\{s_0, s_1, s_2\}$. We therefore construct a counterexample $(s_0 s_1 s_2)^{\omega}$. Detailed information of $\mathscr{A}$ and the product can be found in [**?**].*

a) Discussions: As mentioned before, formulas in $\mathsf{LTL}(\mathsf{U}, \mathsf{X})$ 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 φ_f and an LTL formula φ, the model checking problem of φ under the assumption φ_f reduces to checking whether $\mathscr{K} \models (\varphi_f \to \varphi)$. In order to make use of our specific algorithm for fairness, the procedure can be divided into two steps:

1) $\neg\varphi$ is first transformed into a Büchi automaton, denoted $\mathscr{A}_{\neg\varphi}$, and the product of $\mathscr{A}_{\neg\varphi}$ and $\mathscr{K}$ is then constructed, where all accepting states are marked by a fresh atomic proposition *accepting*;
2) Then $\mathscr{K} \models (\varphi_f \to \varphi)$ iff there is no path in the product satisfying $\varphi_f \wedge \mathsf{GF}accepting$. Note $\varphi_f \wedge \mathsf{GF}accepting$ is still a fair formula, for which our efficient algorithm can be applied.

Note that we can specify some fairness assumption like $\mathsf{FG}(a \vee (\mathsf{X}(b\mathsf{U}c) \wedge \mathsf{F}\neg b))$ in Example 4 which is not in $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$. Moreover, by making use of our algorithm for fairness, we gain some speed up in the model checking procedure if we choose to check $\mathsf{FG}a$ 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 lengths.

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

$$\varphi_0 ::= \varphi_1 \mid \mathsf{F}\varphi_0 \mid \mathsf{G}\varphi_0 \mid \varphi_0 \wedge \varphi_0$$
$$\varphi_f ::= \varphi_0 \mid \varphi_f \vee \varphi_f$$
$$\varphi_e ::= \varphi_1 \mid \varphi_e \vee \varphi_e \mid \varphi_e \wedge \varphi_e \mid \mathsf{F}\varphi_e \mid \mathsf{G}\varphi_e$$

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

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

Example 6. *Let*

$$\begin{aligned}\varphi = \psi_1 \mathsf{U} \psi_2 = &((\mathsf{GF}a_1 \wedge \mathsf{GF}a_2) \vee \cdots \vee (\mathsf{GF}a_{p-1} \wedge \mathsf{GF}a_p)) \\ &\mathsf{U}((\mathsf{GF}b_1 \vee \mathsf{GF}b_2) \wedge \cdots \wedge (\mathsf{GF}b_{q-1} \vee \mathsf{GF}b_q))\end{aligned}$$

Clearly, $|\varphi| = \mathscr{O}(p+q)$. We need first get all F and G modalities out of the scope of U. To this end, by rules of $(\varphi_1 \wedge \varphi_2)\mathsf{U}\varphi_3 \equiv \varphi_1 \mathsf{U}\varphi_3 \wedge \varphi_2 \mathsf{U}\varphi_3$ and $\varphi_1 \mathsf{U}(\varphi_2 \vee \varphi_3) \equiv \varphi_1 \mathsf{U}\varphi_2 \vee \varphi_1 \mathsf{U}\varphi_3$, it requires us to transform ψ_1 to CNF form and ψ_2 to DNF form. After that, we get a formula which is of size $\mathscr{O}(2^{|\varphi|})$.

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 $\varphi = \mathsf{FG}(\neg a \vee (\neg b \mathsf{U} \neg c))$ for example, after negating φ, it gives us $\mathsf{GF}(a \wedge ((\neg b \wedge c)\mathsf{W}(b \wedge c)))$, which is equivalent to $\mathsf{GF}(a \wedge (\mathsf{G}(\neg b \wedge c) \vee ((\neg b \wedge c)\mathsf{U}(b \wedge c))))$. After applying the formula transformation, the resulting formula becomes $(\mathsf{FG}(\neg b \wedge c) \wedge \mathsf{GF}a) \vee \mathsf{GF}(a \wedge ((\neg b \wedge c)\mathsf{U}(b \wedge c)))$. 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.

TABLE I
Number of reachable states

Model	PD6	PD9	PD12	BS4	BS8	BS12	BS16
Size	566	13605	324782	80	2304	53248	1114110

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 $\varphi \in \mathsf{LTL}(\mathsf{F}, \mathsf{G})$ such that $fnf(\varphi) = \bigvee_{i=1}^{m} \left(\mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j})\right)$, the fair cycle detection algorithm can be applied to determine whether there exists an SCC in $\mathscr{K}$ satisfying $\mathsf{FG}l_i \wedge (\bigwedge_{j=1}^{n_i} \mathsf{GF}l_{i,j})$ for some $1 \le i \le m$. 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 $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$ 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:

$$Spec_1 = \left(\bigwedge_{i=1}^{n} (\mathsf{GF}ready_i \rightarrow \mathsf{GF}eat_i)\right) \rightarrow \mathsf{F}eat_1$$

For BS model, we consider the following two formulas:

$$\begin{aligned} Spec_2 &= \left(\bigwedge_{i=1}^{n} (\mathsf{GF}enter_i \rightarrow \mathsf{GF}critical_i)\right) \\ &\quad \rightarrow \mathsf{F}critical_1 \\ Spec_3 &= \left(\bigwedge_{i=1}^{n} (\mathsf{GF}enter_i \rightarrow \mathsf{GF}critical_i)\right) \\ &\quad \rightarrow ((\neg critical_1 \wedge \neg critical_3)\mathsf{U}critical_2) \end{aligned}$$

$Spec_2$ 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 $\mathsf{LTL}(\mathsf{F}, \mathsf{G})$.

In the following we write NuSMV to represent the automata-theoretic approach implemented in NuSMV. Table II

TABLE II
TIME (SECOND) AND MEMORY USAGE (MB) FOR FORMULAS IN THE FIRST CATEGORY

Formula	Model	Time (second)		Memory (MB)	
		Ours	NuSMV	Ours	NuSMV
$Spec_1$	PD6	**2.65**	19.49	**63.90**	136.98
	PD9	**1373.41**	T-O	**113.07**	T-O
	PD12	T-O	T-O	T-O	T-O
$Spec_2$	BS4	0.30	**0.15**	**11.97**	20.67
	BS8	**0.04**	172.63	**14.01**	141.04
	BS12	**0.11**	T-O	**33.85**	T-O
	BS16	**1.06**	M-O	**319.58**	M-O
$Spec_3$	BS4	**0.02**	0.10	**12.22**	18.67
	BS8	**0.03**	101.25	**14.82**	136.64
	BS12	**0.12**	T-O	**36.31**	T-O
	BS16	**1.14**	M-O	**337.55**	M-O

TABLE III
FORMULAS IN SCALABLE PATTERNS GENERATED BY "GENLTL".

Pattern	genltl arguments	Formula
p1	–and-fg = n	$\bigwedge_{i=1}^{n} \mathsf{FG}a_i$
p2	–and-gf = n	$\bigwedge_{i=1}^{n} \mathsf{GF}a_i$
p3	–gh-r = $n-1$	$\bigwedge_{i=1}^{n-1} (\mathsf{GF}a_i \vee \mathsf{FG}a_{i+1})$
p4	–ccj-xi = n, –or-fg = n	$\bigvee_{i=1}^{n} \mathsf{FG}a_i$

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 $\mathsf{LTL}(\mathsf{F},\mathsf{G})$ 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

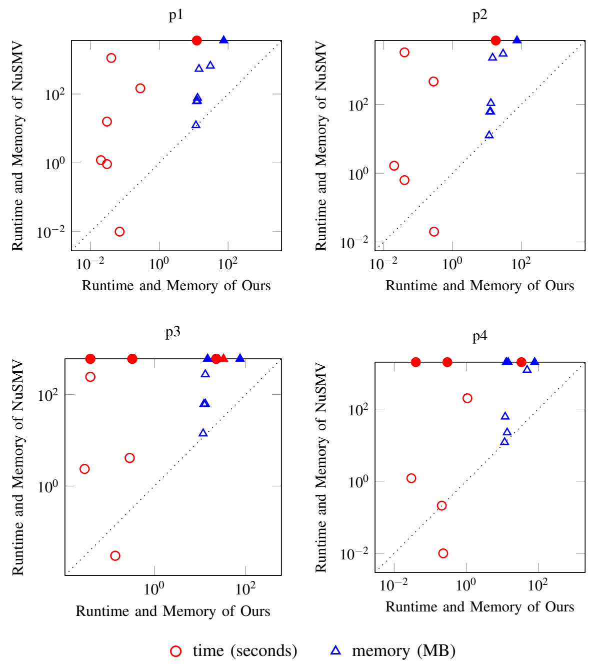

Fig. 2. Comparison With NuSMV for Generated Formulas

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 $\mathsf{LTL}(\mathsf{F},\mathsf{G})$, 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 $\mathsf{LTL}(\mathsf{F},\mathsf{G})$ follows the same idea except that we first conduct a formula transformation so that we can handle fairness like $\mathsf{GF}(a \wedge \mathsf{G}b)$. More importantly, our treatment of fair $\mathsf{LTL}(\mathsf{F},\mathsf{G})$ 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.

TABLE IV
FORMULA PATTERNS USED IN OUR EXPERIMENT

Pattern	Formula
p5	$\bigvee_{1\le i\le n}((\mathsf{FG}a_i\vee\mathsf{GF}b_i)\wedge(\mathsf{FG}c_i\vee\mathsf{GF}d_i))$
p6	$\bigwedge_{1\le i\le n}((\mathsf{FG}a_i\vee\mathsf{GF}b_i)\wedge(\mathsf{FG}c_i\vee\mathsf{GF}d_i))$
p7	$\bigwedge_{1\le i\le n}((\mathsf{GF}(a_i\wedge\mathsf{XX}b_i)\vee\mathsf{FG}b_i)\wedge\mathsf{FG}(c_i\vee(\mathsf{X}d_i\wedge\mathsf{XX}b_i)))$
p8	$\bigvee_{1\le i\le n}((\mathsf{GF}(a_i\wedge\mathsf{XX}b_i)\vee\mathsf{FG}b_i)\wedge\mathsf{FG}(c_i\vee(\mathsf{X}d_i\wedge\mathsf{XX}b_i)))$
p9	$\bigwedge_{1\le i\le n}(\mathsf{FG}(a_i\vee c_i\vee(a_i\mathsf{U}b_i)\vee(c_i\mathsf{U}d_i))$
p10	$\bigvee_{1\le i\le n}(\mathsf{FG}(a_i\vee c_i\vee(a_i\mathsf{U}b_i)\vee(c_i\mathsf{U}d_i))$
p11	$\bigvee_{1\le i\le n}(\mathsf{FG}(a_i\vee(a_i\mathsf{U}b_i))\vee\mathsf{GF}(c_i\wedge(c_i\mathsf{U}d_i)))$
p12	$\bigwedge_{1\le i\le n}(\mathsf{FG}(a_i\vee(a_i\mathsf{U}b_i))\vee\mathsf{GF}(c_i\wedge(c_i\mathsf{U}d_i)))$
p13	$\bigwedge_{1\le i\le n}(\mathsf{FG}((a_i\wedge\mathsf{XX}b_i\wedge\mathsf{GF}b_i)\mathsf{U}(\mathsf{G}(\mathsf{XX}\neg c_i\vee\mathsf{XX}(a_i\wedge b_i)))))$
p14	$\bigwedge_{1\le i\le n}(\mathsf{G}(\mathsf{F}\neg a_i\wedge\mathsf{F}(b_i\wedge\mathsf{X}\neg c_i)\wedge\mathsf{GF}(a_i\mathsf{U}d_i))\wedge\mathsf{GF}((\mathsf{X}d_i)\mathsf{U}(b_i\vee\mathsf{G}c_i)))$
p15	negations of formulas in p13
p16	negations of formulas in p14

TABLE V
TIME USAGE (SECOND)

model	p5		p6		p7		p8		p9		p10	
	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV
PD6	**0.11**	T-O	**0.17**	T-O	**0.27**	2216.90	**43.65**	T-O	**0.49**	3.88	4353.34	**3.90**
PD9	**1.09**	M-O	**0.38**	M-O	**1.74**	M-O	T-O	T-O	**18.54**	M-O	T-O	M-O
PD12	**41.45**	M-O	**12.79**	M-O	**129.50**	M-O	T-O	M-O	**1344.55**	M-O	M-O	M-O
BS4	**0.07**	10.96	**0.23**	26.67	**0.08**	116.65	**0.64**	14.61	**0.06**	0.07	6.49	**0.06**
BS8	**0.14**	M-O	**0.02**	M-O	**0.03**	T-O	**1072.92**	T-O	**0.08**	5.36	T-O	**3.34**
BS12	**4.55**	M-O	**0.25**	M-O	**0.05**	M-O	T-O	M-O	**0.32**	2519.18	M-O	**243.44**
BS16	**377.33**	M-O	**1.07**	M-O	**0.41**	M-O	T-O	M-O	**0.94**	T-O	M-O	M-O

TABLE VI
MEMORY USAGE (MB)

model	p5		p6		p7		p8		p9		p10	
	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV
PD6	**14.10**	T-O	**13.26**	T-O	**14.63**	507.14	**90.35**	T-O	**40.30**	95.27	1877.45	**159.64**
PD9	**55.77**	M-O	**35.61**	M-O	**48.17**	M-O	T-O	T-O	**73.23**	M-O	T-O	M-O
PD12	**98.70**	M-O	**76.07**	M-O	**92.81**	M-O	T-O	M-O	**141.55**	M-O	M-O	M-O
BS4	**11.77**	62.47	**11.70**	63.75	**12.01**	61.32	**19.64**	61.70	**13.37**	20.36	89.45	**19.57**
BS8	**16.44**	M-O	**12.16**	M-O	**12.68**	T-O	**508.83**	T-O	**17.21**	62.23	T-O	**61.89**
BS12	**74.95**	M-O	**12.94**	M-O	**14.28**	M-O	T-O	M-O	**40.95**	683.38	M-O	**394.22**
BS16	**423.19**	M-O	**14.39**	M-O	**16.84**	M-O	T-O	M-O	**63.17**	T-O	M-O	M-O

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 VII
TIME USAGE (SECOND)

model	p11		p12		p13		p14		p15		p16	
	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV
PD6	**4.48**	120.16	**0.26**	1793.13	**0.32**	122.76	**0.07**	T-O	125.82	**57.35**	**3.96**	M-O
PD9	M-O	M-O	**1.62**	T-O	**0.95**	M-O	**0.29**	M-O	T-O	T-O	**2912.29**	M-O
PD12	M-O	M-O	**77.01**	M-O	**72.20**	M-O	**10.95**	M-O	T-O	M-O	T-O	M-O
BS4	15.30	**0.26**	**0.50**	3.49	**0.14**	13.13	**0.01**	313.46	**0.46**	10.14	**0.32**	218.21
BS8	T-O	**280.91**	**0.03**	T-O	**0.03**	T-O	**0.01**	M-O	**742.02**	T-O	**0.26**	M-O
BS12	M-O	M-O	**0.05**	M-O	**0.05**	T-O	**0.34**	M-O	T-O	M-O	**1.80**	M-O
BS16	M-O	M-O	**1.00**	M-O	**0.07**	M-O	**0.38**	M-O	M-O	M-O	**13.16**	M-O

TABLE VIII
MEMORY USAGE (MB)

model	p11		p12		p13		p14		p15		p16	
	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV	Ours	NuSMV
PD6	**139.28**	261.5	**14.72**	457.01	**13.76**	90.76	**12.85**	T-O	101.20	**93.16**	**74.48**	M-O
PD9	M-O	M-O	**50.61**	T-O	**36.09**	M-O	**29.83**	M-O	T-O	T-O	**440.11**	M-O
PD12	M-O	M-O	**90.92**	M-O	**89.87**	M-O	**79.64**	M-O	T-O	M-O	T-O	M-O
BS4	95.83	**40.36**	**12.04**	60.96	**12.08**	51.46	**11.68**	465.63	**20.53**	50.18	**14.07**	460.19
BS8	T-O	**143.0**	**12.71**	T-O	**12.60**	T-O	**12.19**	M-O	**214.81**	T-O	**27.05**	M-O
BS12	M-O	M-O	**14.07**	M-O	**13.70**	T-O	**13.03**	M-O	T-O	M-O	**111.46**	M-O
BS16	M-O	M-O	**16.86**	M-O	**15.42**	M-O	**14.50**	M-O	M-O	M-O	**841.90**	M-O

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/SAFEA International Partnership Program for Creative Research Teams.

REFERENCES

[1] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *ICRA*, pages 2689–2696, 2010.

[4] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *CAV*, volume 1633 of *LNCS*, pages 222–235. Springer, 1999.

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

[6] I. Cerná and R. Pelánek. Relating hierarchy of temporal properties to model checking. In *MFCS*, volume 2747 of *LNCS*, pages 318–327. Springer, 2003.

[7] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.

[8] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events (extended abstract). In *ICALP*, pages 336–349, 1990.

[9] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[10] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *CAV*, volume 1633 of *LNCS*, pages 249–26. Springer, 1999.

[11] A. Duret-Lutz and D. Poitrenaud. SPOT: an extensible model checking library using transition-based generalized Büchi automata. In *Proceedings of 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, MASCOTS '04, pages 76–83. IEEE Press, 2004.

[12] E. A. Emerson and C. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Symposium on Logic in Computer Science*, pages 267–278. IEEE Computer Society, 1986.

[13] J. Esparza and J. Kretínský. From LTL to deterministic automata: A safraless compositional approach. In *CAV*, volume 8559 of *LNCS*, pages 192–208. Springer, 2014.

[14] K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In *CONCUR*, volume 1877 of *LNCS*, pages 153–167. Springer, 2000.

[15] N. Francez. *Fairness*. Springer, 1986.

[16] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.

[17] M. Hammer, A. Knapp, and S. Merz. Truly on-the-fly LTL model checking. In *TACAS*, volume 3440 of *LNCS*, pages 191–205. Springer, 2005.

[18] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[19] D. Kini and M. Viswanathan. Limit Deterministic and Probabilistic Automata for LTL\GU. In *TACAS*, LNCS, pages 628–642, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[20] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[21] B. Lacerda, D. Parker, and N. Hawes. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *IROS*, pages 1511–1516, 2014.

[22] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, pages 125–143, 1977.

[23] R. Pelánek. BEEM: benchmarks for explicit model checkers. In *SPIN*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.

[24] A. Pnueli. The temporal logic of programs. In *SFCS*, pages 46–57. IEEE Computer Society, 1977.

[25] J. Queille and J. Sifakis. Fairness and related properties in transition systems - A temporal logic to deal with fairness. *Acta Inf.*, 19:195–220, 1983.

[26] E. Renault, A. Duret-Lutz, F. Kordon, and D. Poitrenaud. Strength-based decomposition of the property Büchi automaton for faster model checking. In *TACAS*, volume 7795 of *LNCS*, pages 580–593. Springer, 2013.

[27] K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. In *Proceedings of the 14th international SPIN conference on Model checking software*, volume 4595 of *LNCS*, pages 149–167. Springer-Verlag, 2007.

[28] S. Sickert, J. Esparza, S. Jaax, and J. Kretínský. Limit-Deterministic Büchi Automata for Linear Temporal Logic. In *CAV*, LNCS, 2016. To appear.

[29] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Asp. Comput.*, 6(5):495–512, 1994.

[30] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[31] F. Somenzi and R. Bloem. Efficient büchi automata from ltl formulae. In *CAV*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.

[32] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.