SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2018, Vol. 61 052106:1–052106:23
https://doi.org/10.1007/s11432-017-9152-x

Symbolic model checking for discrete real-time systems

Xiangyu LUO[1,2], Lijun WU[3], Qingliang CHEN[4*], Haibo LI[1], Lixiao ZHENG[1] & Zuxi CHEN[1]

[1]*College of Computer Science & Technology, Huaqiao University, Xiamen* 361021*, China;*
[2]*Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin* 541004*, China;*
[3]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu* 611731*, China;*
[4]*Department of Computer Science, Jinan University, Guangzhou* 510632*, China*

Received 10 January 2017/Revised 7 May 2017/Accepted 29 June 2017/Published online 26 December 2017

Abstract A considerably large class of critical applications run in distributed and real-time environments, and most of the correctness requirements of such applications must be expressed by time-critical properties. To enable the specification and verification of these properties in both qualitative and quantitative manners, we propose a new real-time temporal logic RTCTL*, by incorporating both the quantitative (bounded) future and past temporal operators from the qualitative temporal logic CTL*. First, we propose a symbolic method for constructing the temporal tester for arbitrary principally temporal formulas. A temporal tester is constructed as a non-deterministic transducer with a fresh boolean output variable, such that at any position the output variable is set to be true if and only if the corresponding formula holds starting from that position. Then we propose a symbolic model checking method for RTCTL* over finite-state transition systems with weak fairness constraints based on the compositionality of testers. The soundness and completeness of the model checking method, the expressiveness of RTCTL*, and the complexity of the tester construction are described and proven. We have already implemented an efficient model checking prototype for the real-time linear temporal logic RTLTL, which is a quantifier-free version of RTCTL*, by building upon the NuSMV model checker. The theoretical and the experimental results from the prototype both confirm that for checking bounded temporal formulae of the form $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$, our method performs exponentially better than the translation-based method in NuSMV.

Keywords symbolic model checking, temporal tester, real-time temporal logic, just discrete system, OBDDs

Citation Luo X Y, Wu L J, Chen Q L, et al. Symbolic model checking for discrete real-time systems. Sci China Inf Sci, 2018, 61(5): 052106, https://doi.org/10.1007/s11432-017-9152-x

1 Introduction

The formal verification of reactive systems constitutes one of the main technical challenges in computer science. Model checking [1] turns out to be the most popular and powerful verification technique for addressing this issue, where desired behavioral properties can be automatically verified with respect to a given system. For specifying the expected properties of behaviors in a reactive system, temporal logics [1, 2] are widely employed as formal modeling languages. The majority of state-of-the-art model

* Corresponding author (email: tpchen@jnu.edu.cn)

checkers for discrete systems are based on temporal logics such as LTL (linear temporal logic), branching-time temporal logics such as CTL (computation tree logic) and CTL* (a superset of CTL and LTL), or their fragments and extensions. For example, SPIN [3] is based on LTL, and NuSMV [4] and SMV [5] are based on CTL and LTL. In addition, JTLV [6] and the model checker MCTK [7] developed by the present authors embrace all three of these logics. These temporal logics can be seen as mathematical formalisms of qualitative reasoning concerning the evolution of a system over time, and only deal with "before and after" properties, without an explicit reference to time. For example, the LTL formula $\mathtt{F}f$ means that the subformula f will eventually hold, but LTL cannot formalize quantitative properties such as "the exact time at which f will take place".

For many critical applications running in distributed and real-time environments, such as network communication protocols and embedded real-time control systems, it is very important to guarantee the time-critical properties that relate the occurrences of events. For such applications (quantitative) temporal logics are a critical requirement for expressing quantitative properties relating to time evolution during the computations of a given system. The real-time temporal logics adopted in most state-of-the-art model checkers for real-time systems (e.g., Uppaal [8], HyTech [9], Kronos [10], and FSMT-MC [11]) are based on CTL and interpreted over timed transition systems (automata). Uppaal supports a subset of CTL, such that its expressive power is restricted to specify reachability, safety, and liveness. HyTech, Kronos, and FSMT-MC utilize timed CTL [12], which is an extension of CTL that allows quantitative temporal reasoning over dense times. Meanwhile, some other real-time temporal logics are based on LTL, including MTL (metric temporal logic) [13] and TPTL (timed temporal logic) [14], which are defined over integer-time semantics, and MITL (metric interval temporal logic) [15], which employs the nonnegative real numbers as the time domain. However, the major drawback of existing real-time logics based on timed automata is that the computational overhead for model checking is high in such cases [12, 16].

In order to address this challenge, the work presented in this paper aims at striking a good balance between the expressive power of real-time temporal logics and the computational complexity. More specifically, we first propose a new real-time temporal logic RTCTL* (real-time computation tree logic) by extending CTL* with bounded future and past temporal operators so that its expressive power is stronger than those real-time logics that are based mainly on LTL or CTL. Then, we develop an efficient model checking algorithm for RTCTL*. We restrict the computation models for RTCTL* to be discrete transition systems, in which each transition is assumed to run for a single time unit, so that we can measure the elapse of time between events by counting the number of transition steps between those events. Discrete transition systems provide a reasonable method for specifying synchronous systems, such as digital circuits, protocols [17]. Furthermore, the tight timing constraints and predictability required by real-time systems can easily be satisfied using synchronous design techniques. Therefore, real-time systems can often be verified by model checking techniques based on discrete time [1]. To demonstrate the significance of RTCTL*, consider the scenario of a network communication protocol that should satisfy the following property: whenever process P1 receives a request signal (denoted by req) from process P2, it is possible for P1 to send acknowledgment signals (denoted by ack) back to P2 at least once every k steps. This property can be specified through the following RTCTL* formula:

$$\mathtt{AG}(\mathrm{req} \rightarrow \mathtt{E}(\mathtt{F}_{[0,\mathrm{k}]}\mathrm{ack} \wedge \mathtt{G}(\mathrm{ack} \rightarrow \mathtt{F}_{[1,\mathrm{k}]}\mathrm{ack}))), \tag{1}$$

where there are two path quantifiers $\mathtt{A}$ (for all paths) and $\mathtt{E}$ (for some path), an unbounded temporal operator $\mathtt{G}$ (always), and a bounded temporal operator $\mathtt{F}_{[a,b]}$ (eventually within the interval $[a, b]$). We say that $\mathtt{F}_{[a,b]}f$ is true for some path starting from state s_0 if f holds in some future state s on that path, and the distance from s_0 to s is within $[a, b]$. Such properties are very useful for verifying periodic real-time tasks, but they cannot be specified using any other real-time temporal logics based on the proper subsets (CTL or LTL) of CTL*. The main contributions of this paper are summarized as follows:

(1) We propose a new real-time temporal logic RTCTL*, which is an extension of CTL* with future and past time-bounded temporal operators, such that one can express and analyze the real-time behaviors of a system both over the computation tree and single computation;

(2) We construct the symbolic temporal testers for (bounded) future and past temporal operators, and propose a symbolic model checking algorithm for RTCTL* based on testers;

(3) We prove that RTCTL* is exponentially more succinct than the equivalent CTL*. For each bounded temporal formula of the form $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$ (where $\mathtt{U}$ denotes "Until" and $\mathtt{S}$ denotes "Since"), only a linear number of fresh boolean variables are introduced when constructing the tester using our method. Therefore, the state space of our tester can be made exponentially smaller than for the tester constructed using the translation-based method in NuSMV;

(4) We implement an efficient symbolic model checking prototype for the real-time linear temporal logic RTLTL, which is a quantifier-free version of RTCTL*, by building upon NuSMV 2.6.0. This prototype enhances the real-time expressive power of the extended LTL in NuSMV, and the experimental results show that our method performs significantly better than NuSMV for the verification of bounded temporal formulae, especially for those of the form $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$;

(5) Thanks to the compositionality of testers, an important advantage of our tester-based verification method is that it can also easily be incorporated into any other existing temporal model checker as well as NuSMV without requiring any modification, as long as the checker supports fairness constraints and the synchronous parallel composition of modules, such as the temporal model checkers SMV, JTLV, SPIN, and the model checkers MCTK and MCMAS for multi-agent systems [18].

This paper is structured as follows. First, we introduce the computational model in Section 2. The syntax and semantics of RTCTL* are presented in Section 3. The symbolic model checking algorithm for RTCTL* is then presented in Section 4. In Section 5, we implement a symbolic model checking prototype for RTLTL, and present an experimental comparison between NuSMV and our method. In Section 6, we discuss related work and compare this with our approach. Finally, we conclude the paper and describe directions for future work in Section 7.

2 Computational model

We adopt a just discrete system (JDS) as our computational model, which is a finite-state transition system with weak fairness constraints.

Definition 1 (Just discrete system)**.** A just discrete system $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ consists of the following components:

- $V = \{v_1, \ldots, v_n\}$ is a finite set of typed state variables over finite domains. We define a state s to be a type-consistent interpretation of V, assigning to each variable $v \in V$ a value $s[v]$ in its domain. By S, we denote the set of all states.
- Θ is the initial condition. This is an assertion (state formula over V) characterizing all of the initial states of $\mathcal{D}$. A state is called initial if it satisfies Θ; that is, $\Theta(s)$ holds.
- R is a transition relation. This is an assertion $R(V, V')$ relating a state $s \in S$ to its $\mathcal{D}$-successor $s' \in S$, by referring to both V and V'. The transition relation $R(V, V')$ identifies the state s' as a $\mathcal{D}$-successor of the state s if $R(s, s')$ holds under the joint interpretation that interprets $v \in V$ as $s[v]$ and $v' \in V'$ as $s'[v']$. Without loss of generality, we require that $R(V, V')$ must be total; that is, for every state $s \in S$ there exists a state $s' \in S$ such that $R(s, s')$ holds.
- $\mathcal{J} = \{J_1, \ldots, J_k\}$ is a set of assertions expressing the justice (weak fairness) constraints. The justice constraint $J \in \mathcal{J}$ requires that every computation contains infinitely many J-states (states satisfying J).

Given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$, let $r : s_0, s_1, \ldots$ be a sequence of states in $\mathcal{D}$, φ be an assertion, and $i \geqslant 0$ be a natural number. We say that i is a φ-position of r if s_i is a φ-state, i.e., s_i satisfies φ. A run of $\mathcal{D}$ is a sequence of states $r : s_0, s_1, \ldots$ satisfying the following two requirements: (1) Initiality. $\Theta(s_0)$ holds; that is, s_0 is the initial state. (2) Consecution. For each $i \geqslant 0$, $R(s_i, s_{i+1})$ holds; that is, s_{i+1} is a $\mathcal{D}$-successor of s_i. By $\mathrm{runs}(\mathcal{D})$, we denote the set of runs of $\mathcal{D}$. From the requirement that the transition relation must be total, it is easy to infer that every run in $\mathrm{runs}(\mathcal{D})$ is infinitely long. An infinite run of $\mathcal{D}$ is called fair if for each $J \in \mathcal{J}$, the run contains infinitely many J-states. Let $r : s_0, s_1, \ldots$ be a run of $\mathcal{D}$. To facilitate the definition of the semantics of the temporal logic over runs, we first denote each

state s_i by $r(i)$. Then, the run $r : s_0, s_1, \ldots$ can be rewritten as $r : r(0), r(1), \ldots$. We denote the suffix $r(i), r(i+1), \ldots$ by r^i. We refer to a pair (r, i) consisting of a run r and position i as a point. We use the terms path and fair path as synonymous to run and fair run, respectively. Let X be a set of variables. We define $|X|$ as the number of bits representing the variables X in binary. We use $|\mathcal{D}|$ to denote the size of $\mathcal{D}$, and define $|\mathcal{D}| = |S| + |R|$, where $|S|$ is the number of reachable states of $\mathcal{D}$ and $|R|$ is the number of transitions of $\mathcal{D}$. Because $|S| = O(2^{|V|})$ and R is over $V \cup V'$, we have that $|\mathcal{D}| = O(2^{2|V|})$.

Given two JDSs $\mathcal{D}_1 = (V_1, \Theta_1, R_1, \mathcal{J}_1)$ and $\mathcal{D}_2 = (V_2, \Theta_2, R_2, \mathcal{J}_2)$, we define the synchronous parallel composition of $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathcal{D}_1 \parallel \mathcal{D}_2$, as a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$, where $V = V_1 \cup V_2$, $\Theta = \Theta_1 \wedge \Theta_2$, $R = R_1 \wedge R_2$, and $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2$. We can view a run of $\mathcal{D}$ as the synchronous combination of two runs respectively in $\mathrm{runs}(\mathcal{D}_1)$ and $\mathrm{runs}(\mathcal{D}_2)$. The synchronous parallel composition is mainly used to combine a JDS to be checked with a tester T_φ for a given formula φ.

3 Real-time temporal logic RTCTL*

To facilitate the analysis of real-time temporal properties over both the computation tree or a linear computation path of a system, we propose a new real-time temporal logic RTCTL*, which is an extension of the computation tree logic CTL* with (bounded) future and past temporal operators. We assume a finite set of variables V over finite domains, and an underlying assertion language $\mathcal{L}$ that is a first-order language over the integers, and contains interpreted symbols for expressing the standard operations and relations over integers. An assertion is a formula in $\mathcal{L}$. For example, the formula $x < y + 5$ is an assertion that includes two variables x and y, one constant 5, the "+" operation, and the "<" relation over integers.

An RTCTL* formula is constructed of assertions, to which we apply boolean operators, basic (bounded) future/past temporal operators, and path quantifiers. The (bounded) future temporal operators are $\mathtt{X}$ (neXt), $\mathtt{U}$ (Until), and $\mathtt{U}_{[a,b]}$ (bounded Until). The (bounded) past temporal operators are $\mathtt{Y}$ (previously), $\mathtt{S}$ (Since), and $\mathtt{S}_{[a,b]}$ (bounded Since). The path quantifier is $\mathtt{E}$ (for some computation path). For convenience, we add the dual path quantifier $\mathtt{A}$ (for all computation paths) to RTCTL*.

3.1 Syntax of RTCTL*

There are two types of formulae in RTCTL*: state formulae and path formulae. State formulae are interpreted over states, and path formulae are interpreted over paths. Let φ be a state formula and ψ a path formula. Then, the syntax of RTCTL* is defined inductively as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{E}\psi \mid \mathtt{A}\psi \quad \text{and} \quad \psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathtt{X}\psi \mid \psi\mathtt{U}\psi \mid \psi\mathtt{U}_{[a,b]}\psi \mid \mathtt{Y}\psi \mid \psi\mathtt{S}\psi \mid \psi\mathtt{S}_{[a,b]}\psi,$$

where p is an assertion in $\mathcal{L}$, and a and b are nonnegative integers such that $0 \leqslant a \leqslant b$. RTCTL* comprises the set of state formulae generated by the above syntax. Let f and g be RTCTL* formulae. We employ standard abbreviations from propositional logic, such as $\bot$ for $f \wedge \neg f$, $\top$ for $\neg\bot$, $f \vee g$ for $\neg(\neg f \wedge \neg g)$, $f \to g$ for $\neg f \vee g$, and $f \leftrightarrow g$ for $(f \to g) \wedge (g \to f)$.

We write $\psi \in \varphi$ to denote that ψ is a subformula of φ. A formula ψ is called principally temporal if its main operator is temporal. A subformula of the form $\mathtt{E}f$ or $\mathtt{A}f$ in φ is called a maximal state subformula of φ iff it is not a strict subformula of any strict subformula of the form $\mathtt{E}g$ or $\mathtt{A}g$ in φ. To facilitate the analysis of the model checking algorithm for RTCTL*, we first define $\mathrm{vars}(\varphi)$, also denoted by φ-variables, to be the set of variables on which the formula φ depends. We then consider the length (or size) of a formula as the length of the string used to write the formula down in a sufficiently succinct manner. We define the length of a formula using Definition 2.

Definition 2 (Length of a formula)**.** The length of a formula φ, denoted by $|\varphi|$, is calculated as follows:

- If φ is an assertion, then $|\varphi|$ is the sum of the number of atomic propositions that encode each variable in $\mathrm{vars}(\varphi)$ and the number of arithmetic and relational operators;
- If φ is of the form $\neg f$, $\mathtt{X}f$, $\mathtt{Y}f$, $\mathtt{E}f$, or $\mathtt{A}f$, then $|\varphi| = |f| + 1$;
- If φ is of the form $f \wedge g$, $f\mathtt{U}g$, or $f\mathtt{S}g$, then $|\varphi| = |f| + |g| + 1$;
- If φ is of the form $f\mathtt{U}_{[a,b]}g$ or $f\mathtt{S}_{[a,b]}g$, then $|\varphi| = |f| + |g| + \lceil \log_2 a \rceil + \lceil \log_2 b \rceil + 1$, where $\lceil \log_2 0 \rceil = 1$;

bounded Until:	(bounded) Finally:	(bounded) Globally:
$f\mathtt{U}_{[a,\infty]}g \equiv f\mathtt{U}_{[a,a]}(f\mathtt{U}g)$	$\mathtt{F}f \equiv \top\mathtt{U}f$	$\mathtt{G}f \equiv \neg\mathtt{F}\neg f$
$f\mathtt{U}_{[a,b]}g \equiv f\mathtt{U}_{[a,a]}(f\mathtt{U}_{[0,b-a]}g)$	$\mathtt{F}_{[a,b]}f \equiv \top\mathtt{U}_{[a,b]}f$	$\mathtt{G}_{[a,b]}f \equiv \neg\mathtt{F}_{[a,b]}\neg f$
	$\mathtt{F}_{[a,\infty]}f \equiv \top\mathtt{U}_{[a,a]}(\mathtt{F}f)$	$\mathtt{G}_{[a,\infty]}f \equiv \top\mathtt{U}_{[a,a]}(\mathtt{G}f)$
not previous state not: $\mathtt{Z}f \equiv \neg\mathtt{Y}\neg f$	(bounded) Once:	(bounded) Historically:
bounded Since:	$\mathtt{O}f \equiv \top\mathtt{S}f$	$\mathtt{H}f \equiv \neg(\top\mathtt{S}\neg f)$
$f\mathtt{S}_{[a,\infty]}g \equiv f\mathtt{S}_{[a,a]}(f\mathtt{S}g)$	$\mathtt{O}_{[a,b]}f \equiv \top\mathtt{S}_{[a,b]}f$	$\mathtt{H}_{[a,b]}f \equiv \neg(\top\mathtt{S}_{[a,b]}\neg f)$
$f\mathtt{S}_{[a,b]}g \equiv f\mathtt{S}_{[a,a]}(f\mathtt{S}_{[0,b-a]}g)$	$\mathtt{O}_{[a,\infty]}f \equiv \mathtt{O}_{[a,a]}\mathtt{O}f$	$\mathtt{H}_{[a,\infty]}f \equiv \mathtt{H}_{[a,a]}\mathtt{H}f$

Figure 1 The (bounded) future and past temporal operators derived from RTCTL*.

- Otherwise, $|\varphi| = 0$.

3.2 Semantics of RTCTL*

The semantics of RTCTL* is inductively defined with respect to a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$. We define the notion of an RTCTL* formula φ holding at a point (r, i) of $\mathcal{D}$, denoted by $(\mathcal{D}, r, i) \models \varphi$, as follows:

$(\mathcal{D}, r, i) \models p$, iff p is an assertion over V in $\mathcal{L}$, and p holds at the state $r(i)$.
$(\mathcal{D}, r, i) \models \neg f$, iff $(\mathcal{D}, r, i) \not\models f$.
$(\mathcal{D}, r, i) \models f \wedge g$, iff $(\mathcal{D}, r, i) \models f$, and $(\mathcal{D}, r, i) \models g$.
$(\mathcal{D}, r, i) \models \mathtt{E}f$, iff $\exists r' \in \mathrm{run}s(\mathcal{D})$ and $\exists i' \geqslant 0$. $(r'(i') = r(i)$ and $(\mathcal{D}, r', i') \models f)$.
$(\mathcal{D}, r, i) \models \mathtt{A}f$, iff $\forall r' \in \mathrm{run}s(\mathcal{D})$ and $\forall i' \geqslant 0$. $(r'(i') = r(i)$ implies $(\mathcal{D}, r', i') \models f)$.
$(\mathcal{D}, r, i) \models \mathtt{X}f$, iff $(\mathcal{D}, r, i+1) \models f$.
$(\mathcal{D}, r, i) \models f\mathtt{U}g$, iff $\exists i' \geqslant i$. $((\mathcal{D}, r, i') \models g$ and $\forall i'' \in [i, i')$. $(\mathcal{D}, r, i'') \models f)$.
$(\mathcal{D}, r, i) \models f\mathtt{U}_{[a,b]}g$, iff $\exists i' \in [i+a, i+b]$. $((\mathcal{D}, r, i') \models g$ and $\forall i'' \in [i, i')$. $(\mathcal{D}, r, i'') \models f)$.
$(\mathcal{D}, r, i) \models \mathtt{Y}f$, iff $i > 0$, and $(\mathcal{D}, r, i-1) \models f$. This implies that $(\mathcal{D}, r, 0) \not\models \mathtt{Y}f$.
$(\mathcal{D}, r, i) \models f\mathtt{S}g$, iff $\exists i' \leqslant i$.$((\mathcal{D}, r, i') \models g$, and $\forall i'' \in (i', i]$. $(\mathcal{D}, r, i'') \models f)$.
$(\mathcal{D}, r, i) \models f\mathtt{S}_{[a,b]}g$, iff $\exists i' \in [i-b, i-a]$.$((\mathcal{D}, r, i') \models g$, and $\forall i'' \in (i', i]$. $(\mathcal{D}, r, i'') \models f)$.

Sometimes, we denote the notion of a state formula φ holding at a state $r(i)$ of $\mathcal{D}$ as $(\mathcal{D}, r(i)) \models \varphi$, where $r(i)$ is the state located at the position i of the run r of $\mathcal{D}$. From the semantics, it is easy to have that $\mathtt{A}f \equiv \neg\mathtt{E}\neg f$. In fact, many other useful standard future/past temporal operators can be derived from RTCTL* by rewriting rules. We list some equations in Figure 1, which can be viewed as the rewriting rules that convert the additional formula on the left hand side of "$\equiv$" into the formula on the right hand side. Therefore, any formula can be converted into RTCTL* by applying the rewriting rules to each additional subformula recursively. For conciseness, we do not list the semantics of these additional operators. One can understand these through the equations and semantics of RTCTL*.

The subset of RTCTL* in which each (bounded) temporal operator is immediately preceded by a path quantifier is called RTCTL. The subset of RTCTL without any bounded temporal operators is exactly CTL. The subset of RTCTL* without any path quantifiers or bounded temporal operators is exactly LTL. We denote the subset of RTCTL* without path quantifiers by RTLTL.

3.3 The model checking problem for RTCTL*

Consider a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ and an RTCTL* formula φ. If φ is a state formula, then we say that φ holds over a state s of $\mathcal{D}$, denoted by $(\mathcal{D}, s) \models \varphi$, if s satisfies φ. If φ is a path formula, then we say that φ holds over a path r, denoted by $(\mathcal{D}, r) \models \varphi$, if r satisfies the semantics of φ. We say that φ holds on the JDS $\mathcal{D}$, denoted by $\mathcal{D} \models \varphi$, if $(\mathcal{D}, s) \models \varphi$ for every initial state s satisfying Θ. Note that if φ is a path formula, then the model checking problem of whether φ holds on $\mathcal{D}$ is equivalent to $\mathcal{D} \models \mathtt{A}\varphi$. φ is

called satisfiable if it holds on some JDS. φ is called valid if it holds on all JDSs. Our goal is to verify whether or not all runs of $\mathcal{D}$ satisfy φ, i.e., $\mathcal{D} \models \varphi$.

4 Tester-based symbolic model checking for RTCTL*

For a composite RTCTL* formula, it is essential for the model checking methods for various operators to be compositional to each other. This compositionality will also contribute to the further extension of the logic language, because this is sufficient for constructing the testers for only newly introduced operators. To achieve this compositionality, the core problem is to determine whether any suffix of a run starting from any position satisfies the given formula. To solve this problem, given a JDS $\mathcal{D}$ and a principally temporal formula φ, we wish to construct another JDS T_φ called a temporal tester for φ. This can be viewed as a non-deterministic transducer that keeps observing a run r of $\mathcal{D}$, and at each position $i \geqslant 0$ sets a fresh boolean variable x_φ to be true iff $(\mathcal{D}, r, i) \models \varphi$. x_φ is also called the output variable of T_φ, because the output value of x_φ can be viewed as representing the satisfiability of φ.

By taking advantage of the compositionality of testers, we are able to reduce the model checking problem of a composite formula over a given JDS to that of a state formula over the synchronous parallel composition of the given JDS and the testers for the principally temporal subformulae of the composite formula, where the state formula is generated by replacing each principally temporal subformula with the corresponding fresh output variable. Next, we define the so-called temporal tester T_φ for a principally temporal RTCTL* formula φ.

Definition 3 (Temporal tester)**.** Let φ be a principally temporal RTCTL* formula. A temporal tester T_φ for φ is a JDS with a fresh boolean output variable x_φ, such that

- Soundness: For every fair run ρ of T_φ and each position $i \geqslant 0$, $(T_\varphi, \rho, i) \models \varphi$ iff $(T_\varphi, \rho, i) \models x_\varphi$.
- Completeness: Let $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ be a JDS. Then, for every fair run r in $\mathcal{D}$, there is a corresponding fair run ρ of T_φ such that for each $i \geqslant 0$, $(T_\varphi, \rho, i) \models x_\varphi$ iff $(\mathcal{D}, r, i) \models \varphi$.

For convenience, a temporal tester will simply be referred to as a tester from this point on. A tester is independent of any JDS, and accepts all possible runs of a JDS. For every suffix of these runs, T_φ guarantees that the output variable x_φ is true at the starting state of the suffix if and only if the suffix satisfies φ. Let $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ be a JDS and φ an RTCTL* formula. After constructing the tester T_φ for φ and mapping φ to its corresponding state formula φ', the model checking problem $\mathcal{D} \models \varphi$ is reduced to the model checking problem $\mathcal{D} \parallel T_\varphi \models \varphi'$, which is easy to tackle using the symbolic model checking method. Occasionally, we will simply denote $\mathcal{D} \parallel T_\varphi$ as $\mathcal{D}_\varphi$.

4.1 Mapping RTCTL* formulae to state ones

Given a JDS $\mathcal{D}$ and an RTCTL* formula φ, the basic problem for model checking φ over $\mathcal{D}$ is checking the subformula of the form $\mathtt{E}f$ in φ, where f is a path formula that may include its own maximal state subformula(e), which are also of the form $\mathtt{E}g$. Therefore, we can design a top-down model checking procedure to check φ on its syntax tree. Thanks to the symbolic model checking method, we can obtain the symbolic representation (OBDDs-ordered binary decision diagrams [19]) of the set of states that satisfies a state formula. For an RTCTL* formula $\mathtt{E}f$, assume that all maximal state subformulae of f have already been symbolically model checked. Then, f can be converted into an RTLTL formula f' by accordingly replacing the maximal state subformulae in f with the resulting OBDDs. Thus, the RTCTL* model checking problem of $\mathcal{D} \models \mathtt{E}f$ is reduced to the RTLTL model checking problem of $\mathcal{D} \models \mathtt{E}f'$.

Furthermore, for the RTLTL path formula f' we can also design a top-down procedure on the syntax tree of f' to convert f' into a state formula f'', by replacing each principally temporal subformula g in f' with the fresh boolean variable of the tester for g. Finally, the model checking problem of an RTLTL formula f' over $\mathcal{D}$ can be reduced to that of the state formula f'' over the synchronous parallel composition of $\mathcal{D}$ and the testers constructed for the principally temporal subformulae of f'. This problem can easily be solved using the symbolic model checking method. To achieve this, in the following we design a function χ

to map an RTCTL* formula φ to the corresponding state formula $\chi(\varphi)$, where ψ is a subformula of φ.

$$\chi(\psi)=\begin{cases}\psi, & \psi \text{ is an assertion, or } \psi=\mathtt{E}f;\\ \neg\chi(f), & \psi=\neg f;\\ \chi(f)\wedge\chi(g), & \psi=f\wedge g;\\ x_\psi, & \psi=\mathtt{X}f, f\mathtt{U}g, f\mathtt{U}_{[0,b]}g, \mathtt{Y}f, f\mathtt{S}g \text{ or } f\mathtt{S}_{[0,b]}g, \text{ where } b>0;\\ \chi(g), & \psi=f\mathtt{U}_{[0,0]}g \text{ or } f\mathtt{S}_{[0,0]}g;\\ \chi(\{f\wedge\mathtt{X}(\}^a f\mathtt{U}_{[0,b-a]}g\{)\}^a), & \psi=f\mathtt{U}_{[a,b]}g \text{ and } 0\leqslant a\leqslant b;\\ \chi(\{f\wedge\mathtt{Y}(\}^a f\mathtt{S}_{[0,b-a]}g\{)\}^a), & \psi=f\mathtt{S}_{[a,b]}g \text{ and } 0\leqslant a\leqslant b,\end{cases}$$

where x_ψ is the fresh boolean (output) variable that represents the tester created for the principally temporal subformula ψ of φ. Note that for a maximal state subformula $\mathtt{E}f$ of φ, we preserve this in the resulting formula such that it can be handled in a bottom-up approach.

To construct a tester for an RTCTL* formula φ, not only must the fresh boolean variable for each principally temporal subformula in φ be created, but also the integer variable for counting the transition steps for each bounded principally temporal subformula in φ. Note that the temporal subformulae in any maximal state subformula of φ are excluded. We define the set of the fresh variables created for the tester T_φ as $X_\varphi=X_\varphi^1\cup X_\varphi^2$. One can better understand the set X_φ after consulting the tester construction method described in Subsection 4.2.

$$X_\varphi^1=\left\{x_\psi\,\middle|\,\begin{array}{l}x_\psi \text{ is the fresh boolean variable for } \psi.\ \psi \text{ is of the form } \mathtt{X}f, f\mathtt{U}g, \mathtt{Y}f \text{ or } f\mathtt{S}g, \text{ which is a}\\ \text{subformula of } \varphi \text{ but not in any maximal state subformula of } \varphi.\end{array}\right\},$$

$$X_\varphi^2=\left\{x_\psi,\ t_\psi\,\middle|\,\begin{array}{l}x_\psi \text{ is the fresh boolean variable for } \psi, t_\psi \text{ is the fresh integer variable in interval}\\ [0,b-1] \text{ for } \psi.\ \psi \text{ is of the form } f\mathtt{U}_{[0,b]}g \text{ or } f\mathtt{S}_{[0,b]}g, \text{ where } b>0.\ \psi \text{ is a subformula}\\ \text{of } \varphi \text{ but not in any maximal state subformula of } \varphi.\end{array}\right\}.$$

As an example, consider $\varphi_2=\neg(f\mathtt{U}_{[a,b]}g\vee\mathtt{EX}h)\wedge p\mathtt{U}(q\vee\mathtt{X}r)$. Assume that x_1,x_2,x_3 are the fresh boolean variables of the testers $T_{f\mathtt{U}_{[a,b]}g}$, $T_{\mathtt{X}r}$ and $T_{p\mathtt{U}(q\vee\mathtt{X}r)}$, respectively, and the fresh integer variable $t\in[0,b-a-1]$ represents $T_{f\mathtt{U}_{[a,b]}g}$. Then, we have that $\chi(\varphi_2)=\neg(x_1\vee\mathtt{EX}h)\wedge x_3$, $\mathrm{vars}(\chi(\varphi_2))=\{x_1,x_3,h\}$, and $X_{\varphi_2}=\{x_1,x_2,x_3,t\}$.

4.2 Construction of testers for principally temporal formulae

In this section, we present a symbolic method for constructing the tester T_φ for a principally temporal RTCTL* formula φ. Then, we will present a compositional method for constructing the tester for arbitrary RTCTL* formulae. We first construct the testers for the basic future temporal operators $\mathtt{X}$, $\mathtt{U}$, and $\mathtt{U}_{[a,b]}$.

4.2.1 *Tester for* Xf

For $\varphi=\mathtt{X}f$, it follows from the semantics of $\mathtt{X}f$ that the tester $T_{\mathtt{X}f}=(V_\varphi,\Theta_\varphi,R_\varphi,\mathcal{J}_\varphi)\parallel T_f$ is defined by the (2), where x_φ is the fresh boolean variable for φ. Here, $\chi'(f)$ is the prime version of $\chi(f)$. That is, $\chi'(f)$ is the formula resulting by replacing each occurrence of $v\in\mathrm{vars}(\chi(f))$ in $\chi(f)$ with the prime version v' of v. We have that $\chi(\varphi)=x_\varphi$ and $X_\varphi=\{x_\varphi\}\cup X_f$:

$$T_{\mathtt{X}f}:\begin{cases}V_\varphi:\mathrm{vars}(\chi(f))\cup\{x_\varphi\};\\ \Theta_\varphi:\top;\\ R_\varphi:x_\varphi\leftrightarrow\chi'(f);\\ \mathcal{J}_\varphi:\emptyset,\end{cases}\tag{2}$$

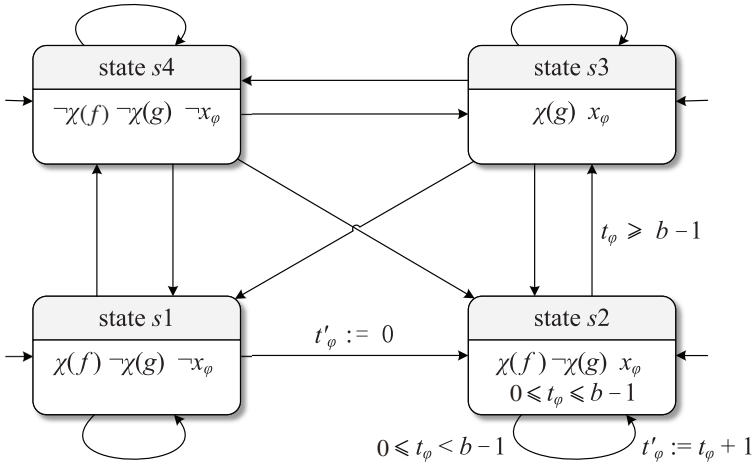

Figure 2 The tester $T_{f\mathtt{U}_{[0,b]}g}$.

$$T_{f\mathtt{U}g}: \begin{cases} V_\varphi : \mathrm{vars}(\chi(f)) \cup \mathrm{vars}(\chi(g)) \cup \{x_\varphi\}; \\ \Theta_\varphi : \top; \\ R_\varphi : x_\varphi \leftrightarrow (\chi(g) \vee (\chi(f) \wedge x'_\varphi)); \\ \mathcal{J}_\varphi : \{x_\varphi \rightarrow \chi(g)\}. \end{cases} \tag{3}$$

4.2.2 *Tester for* $f\mathrm{U}g$

For $\varphi = f\mathtt{U}g$, it follows from the expansion equation $f\mathtt{U}g \equiv g \vee (f \wedge \mathtt{X}(f\mathtt{U}g))$ that the tester $T_{f\mathtt{U}g} = (V_\varphi, \Theta_\varphi, R_\varphi, \mathcal{J}_\varphi) \parallel T_f \parallel T_g$ is defined by (3), where x_φ is the fresh boolean variable for φ and x'_φ is the prime version of x_φ. We have that $\chi(\varphi) = x_\varphi$ and $X_\varphi = \{x_\varphi\} \cup X_f \cup X_g$. Note that without the justice constraints $\mathcal{J}_\varphi$, the transition relation R_φ cannot guarantee the correctness of the tester, because it accepts infinite runs along which $x_\varphi \wedge \chi(f) \wedge \neg\chi(g)$ holds forever, and in this case the runs do not satisfy $f\mathtt{U}g$. These runs can be ruled out by the justice constraint $x_\varphi \rightarrow \chi(g)$.

4.2.3 *Tester for* $f\mathrm{U}_{[a,b]}g$

In this section, we want to construct the testers for bounded future temporal operators. However, from Figure 1 we have that $\mathtt{U}$ and $\mathtt{U}_{[a,b]}$ are the two basic operators for expressing arbitrary bounded future temporal operators. The tester for $\mathtt{U}$ is presented in the previous section, and so in this section it is adequate to construct the tester for $f\mathtt{U}_{[a,b]}g$. From the semantics of RTCTL*, we first define 4, which can be viewed as a rewriting rule such that we can convert $f\mathtt{U}_{[a,b]}g$ into a formula that only includes the temporal operators $\mathtt{X}$ and $\mathtt{U}_{[0,b]}$.

$$f\mathtt{U}_{[a,b]}g = \begin{cases} \{f \wedge \mathtt{X}(\}^a\ f\mathtt{U}_{[0,b-a]}g\ \{)\}^a, & \text{if } 0 \leqslant a < b; \\ \{f \wedge \mathtt{X}(\}^a\ g\ \{)\}^a, & \text{if } 0 \leqslant a = b, \end{cases} \tag{4}$$

where $\{f \wedge \mathtt{X}(\}^a$ denotes the string formed by repeatedly concatenating the word "$f \wedge \mathtt{X}($" a times, so $\{f \wedge \mathtt{X}(\}^0$ denotes an empty string. By applying (4), we can rewrite $f\mathtt{U}_{[a,b]}g$ with $a < b$ as $\{f \wedge \mathtt{X}(\}^a f\mathtt{U}_{[0,b-a]}g\{)\}^a$, the tester of which can be constructed using the two ways synchronous parallel composition of the two testers for $f\mathtt{U}_{[0,b-a]}g$ and $\{f \wedge \mathtt{X}(\}^a x\{)\}^a$, where x is the fresh boolean variable of the tester for $f\mathtt{U}_{[0,b-a]}g$.

Without loss of generality, we construct the tester $T_{f\mathtt{U}_{[0,b]}g}$ as shown in Figure 2, where $b \geqslant 1$.

Formally, for a formula $\varphi = f\mathtt{U}_{[0,b]}g$ the tester $T_{f\mathtt{U}_{[0,b]}g} = (V_\varphi, \Theta_\varphi, R_\varphi, \mathcal{J}_\varphi) \parallel T_f \parallel T_g$ is defined as

follows:

$$T_{f\mathtt{U}_{[0,b]}g}:\begin{cases} V_\varphi:\ \mathrm{vars}(\chi(f))\cup\mathrm{vars}(\chi(g))\cup\{x_\varphi,t_\varphi\};\\ \Theta_\varphi:\ 0\leqslant t_\varphi\leqslant b-1;\\ R_\varphi:\ (0\leqslant t_\varphi\leqslant b-1)\wedge(0\leqslant t'_\varphi\leqslant b-1)\wedge R_{f\mathtt{U}_{[0,b]}g};\\ \mathcal{J}_\varphi:\ \emptyset, \end{cases} \tag{5}$$

where x_φ is the fresh boolean variable, and t_φ is the fresh integer variable in the interval $[0,b-1]$. We have that $\chi(\varphi)=x_\varphi$ and $X_\varphi=\{x_\varphi,t_\varphi\}\cup X_f\cup X_g$. We define $s1:=(\chi(f)\wedge\neg\chi(g)\wedge\neg x_\varphi)$, $s2:=(\chi(f)\wedge\neg\chi(g)\wedge x_\varphi)$, $s3:=(\chi(g)\wedge x_\varphi)$, and $s4:=(\neg\chi(f)\wedge\neg\chi(g)\wedge\neg x_\varphi)$, and define $s1',s2',s3'$, and $s4'$ as the prime versions of $s1,s2,s3$, and $s4$, respectively. For example, $s1':=(\chi'(f)\wedge\neg\chi'(g)\wedge\neg x'_\varphi)$. According to Figure 2, the assertion $R_{f\mathtt{U}_{[0,b]}g}$ is defined as follows:

$$\begin{aligned} R_{f\mathtt{U}_{[0,b]}g}:=\ & (s1\wedge s1')\vee(s1\wedge s2'\wedge t'_\varphi=0)\vee(s1\wedge s4')\\ & \vee(s2\wedge 0\leqslant t_\varphi<b-1\wedge s2'\wedge t'_\varphi=t_\varphi+1)\vee(s2\wedge t_\varphi\geqslant b-1\wedge s3')\\ & \vee(s3\wedge s1')\vee(s3\wedge s2')\vee(s3\wedge s3')\vee(s3\wedge s4')\\ & \vee(s4\wedge s1')\vee(s4\wedge s2')\vee(s4\wedge s3')\vee(s4\wedge s4'). \end{aligned}$$

In the following we start constructing the testers for (bounded) past temporal operators $\mathtt{Y}$, $\mathtt{S}$ and $\mathtt{S}_{[a,b]}$.

4.2.4 *Tester for* $\mathrm{Y}f$

It follows from the semantics of $\varphi=\mathtt{Y}f$ that the tester $T_{\mathtt{Y}f}=(V_\varphi,\Theta_\varphi,R_\varphi,\mathcal{J}_\varphi)\parallel T_f$ is defined by (6), where x_φ is the fresh boolean variable for φ and x'_φ is the prime version of x_φ. Note that the initial condition $\Theta_\varphi=\neg x_\varphi$ corresponds to the semantics stating that for any subformula f, $\mathtt{Y}f$ must be false over any initial state; that is, $(\mathcal{D},r,0)\not\models\mathtt{Y}f$. We have that $\chi(\varphi)=x_\varphi$ and $X_\varphi=\{x_\varphi\}\cup X_f$.

$$T_{\mathtt{Y}f}:\begin{cases} V_\varphi:\ \mathrm{vars}(\chi(f))\cup\{x_\varphi\};\\ \Theta_\varphi:\ \neg x_\varphi;\\ R_\varphi:\ x'_\varphi\leftrightarrow\chi(f);\\ \mathcal{J}_\varphi:\ \emptyset, \end{cases} \tag{6}$$

$$T_{f\mathtt{S}g}:\begin{cases} V_\varphi:\ \mathrm{vars}(\chi(f))\cup\mathrm{vars}(\chi(g))\cup\{x_\varphi\};\\ \Theta_\varphi:\ x_\varphi\leftrightarrow\chi(g);\\ R_\varphi:\ x'_\varphi\leftrightarrow(\chi'(g)\vee(\chi'(f)\wedge x_\varphi));\\ \mathcal{J}_\varphi:\ \emptyset. \end{cases} \tag{7}$$

4.2.5 *Tester for* $f\mathrm{S}g$

For $\varphi=f\mathtt{S}g$, it follows from the expansion equation $f\mathtt{S}g\equiv g\vee(f\wedge\mathtt{Y}(f\mathtt{S}g))$ that the tester $T_{f\mathtt{S}g}=(V_\varphi,\Theta_\varphi,R_\varphi,\mathcal{J}_\varphi)\parallel T_f\parallel T_g$ is defined by (7), where x_φ is the fresh boolean variable for φ and x'_φ is the prime version of x_φ. We have that $\chi(\varphi)=x_\varphi$ and $X_\varphi=\{x_\varphi\}\cup X_f\cup X_g$. Consider an initial state s_0 satisfying $\neg\chi(g)$. Owing to the fact that there is no predecessor for s_0, it follows from the semantics we have that there is no run to s_0 satisfying $f\mathtt{S}g$. Furthermore, if s_0 satisfies $\chi(g)$, then we immediately have that any run to s_0, i.e., s_0 itself, satisfies $f\mathtt{S}g$. Therefore, we set the initial condition Θ_φ to be $x_\varphi\leftrightarrow\chi(g)$. We set the justice constraints $\mathcal{J}_\varphi$ to be the empty set, because any run from the state under consideration back to any initial state is finite, so that the initial condition Θ_φ and the transition relation R_φ are adequate for implementing the semantics of $f\mathtt{S}g$.

4.2.6 *Tester for* $f\mathrm{S}_{[a,b]}g$

In this section, we want to construct the testers for bounded past temporal operators. From Figure 1, we have that $\mathtt{S}$ and $\mathtt{S}_{[a,b]}$ are the two basic operators for expressing arbitrary bounded past temporal

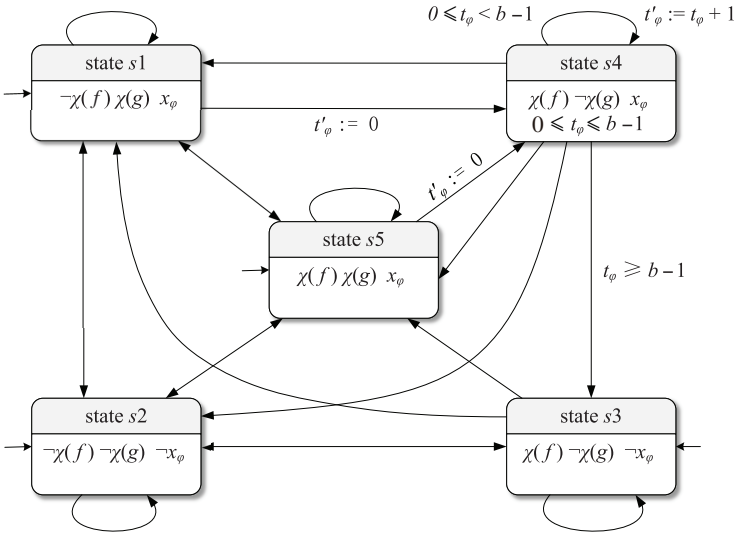

Figure 3 The tester $T_{f\mathtt{S}_{[0,b]}g}$.

operators. The tester for $\mathtt{S}$ was presented in the previous section, and so in this section it is adequate to construct the tester for $f\mathtt{S}_{[a,b]}g$. From the semantics of RTCTL*, we first define (8), which can be viewed as a rewriting rule such that we can convert $f\mathtt{S}_{[a,b]}g$ into a formula that only includes the temporal operators $\mathtt{Y}$ and $\mathtt{S}_{[0,b]}$:

$$f\mathtt{S}_{[a,b]}g = \begin{cases} \{f \wedge \mathtt{Y}(\}^a\ f\mathtt{S}_{[0,b-a]}g\ \{)\}^a, & \text{if } 0 \leqslant a < b; \\ \{f \wedge \mathtt{Y}(\}^a\ g\ \{)\}^a, & \text{if } 0 \leqslant a = b, \end{cases} \tag{8}$$

where $\{f \wedge \mathtt{Y}(\}^a$ denotes the string formed by repeatedly concatenating the word "$f \wedge \mathtt{Y}($" a times. By recursively applying (8), we can rewrite $f\mathtt{S}_{[a,b]}g$ $(a < b)$ as $\{f \wedge \mathtt{Y}(\}^a f\mathtt{S}_{[0,b-a]}g\{)\}^a$, which is a formula that only includes the temporal operators $\mathtt{Y}$ and $\mathtt{S}_{[0,b]}$. Furthermore, its tester can be constructed using the two ways synchronous parallel composition of the two testers for $f\mathtt{S}_{[0,b-a]}g$ and $\{f \wedge \mathtt{X}(\}^a x\{)\}^a$, where x is the fresh boolean variable of the tester for $f\mathtt{S}_{[0,b-a]}g$. Without loss of generality, we construct the tester $T_{f\mathtt{S}_{[0,b]}g}$ as shown in Figure 3, where $b \geqslant 1$. A bidirectional edge denotes two transitions with reverse directions. An edge without a starting node indicates that the node the edge points to is one of the initial states, so that $\{s1, s2, s3, s5\}$ is the set of initial states.

For $\varphi = f\mathtt{S}_{[0,b]}g$, the tester $T_{f\mathtt{S}_{[0,b]}g} = (V_\varphi, \Theta_\varphi, R_\varphi, \mathcal{J}_\varphi) \parallel T_f \parallel T_g$ is formally defined as follows:

$$T_{f\mathtt{S}_{[0,b]}g} : \begin{cases} V_\varphi : \mathrm{vars}(\chi(f)) \cup \mathrm{vars}(\chi(g)) \cup \{x_\varphi, t_\varphi\}; \\ \Theta_\varphi : \neg(\chi(f) \wedge \neg\chi(g) \wedge x_\varphi \wedge 0 \leqslant t_\varphi \leqslant b-1) \wedge (x_\varphi \leftrightarrow \chi(g)); \\ R_\varphi : (0 \leqslant t_\varphi \leqslant b-1) \wedge (0 \leqslant t'_\varphi \leqslant b-1) \wedge R_{f\mathtt{S}_{[0,b]}g}; \\ \mathcal{J}_\varphi : \emptyset, \end{cases} \tag{9}$$

where x_φ is the fresh boolean variable, and t_φ is the fresh nonnegative integer variable ($t_\varphi \in [0..b-1]$) for $\varphi = f\mathtt{S}_{[0,b]}g$. We have that $\chi(\varphi) = x_\varphi$ and $X_\varphi = \{x_\varphi, t_\varphi\} \cup X_f \cup X_g$. We define $s1 := (\neg\chi(f) \wedge \chi(g) \wedge x_\varphi)$, $s2 := (\neg\chi(f) \wedge \neg\chi(g) \wedge \neg x_\varphi)$, $s3 := (\chi(f) \wedge \neg\chi(g) \wedge \neg x_\varphi)$, $s4 := (\chi(f) \wedge \neg\chi(g) \wedge x_\varphi)$, and $s5 := (\chi(f) \wedge \chi(g) \wedge x_\varphi)$, and define $s1', s2', s3'$, $s4'$, and $s5'$ as the prime versions of $s1, s2, s3, s4$, and $s5$, respectively. According to Figure 3, the assertion $R_{f\mathtt{S}_{[0,b]}g}$ is defined as follows:

$$\begin{aligned} R_{f\mathtt{S}_{[0,b]}g} := {} & (s1 \wedge s1') \vee (s1 \wedge s2') \vee (s1 \wedge s4' \wedge t'_\varphi = 0) \vee (s1 \wedge s5') \\ & \vee (s2 \wedge s1') \vee (s2 \wedge s2') \vee (s2 \wedge s3') \vee (s2 \wedge s5') \\ & \vee (s3 \wedge s1') \vee (s3 \wedge s2') \vee (s3 \wedge s3') \vee (s3 \wedge s5') \\ & \vee (s4 \wedge s1') \vee (s4 \wedge s2') \vee (s4 \wedge t_\varphi \geqslant b-1 \wedge s3') \vee (s4 \wedge 0 \leqslant t_\varphi < b-1 \wedge s4' \wedge t'_\varphi = t_\varphi + 1) \\ & \vee (s4 \wedge s5') \vee (s5 \wedge s1') \vee (s5 \wedge s2') \vee (s5 \wedge s4' \wedge t'_\varphi = 0) \vee (s5 \wedge s5'). \end{aligned}$$

4.3 Correctness of the tester construction for principally temporal formulae

Lemma 1 (Soundness of the tester for a principally temporal formula)**.** Let φ be of the form $\mathtt{X}f$, $\mathtt{Y}f$, $f\mathtt{U}g$, $f\mathtt{S}g$, $f\mathtt{U}_{[0,b]}g$, or $f\mathtt{S}_{[0,b]}g$, and let x_φ be the fresh boolean output variable for φ. Then, for each fair run r of T_φ and every position i, $(T_\varphi, r, i) \models \varphi$ iff $(T_\varphi, r, i) \models x_\varphi$.

Lemma 2 (Completeness of the tester for a principally temporal formula)**.** Let $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ be a JDS, where φ is of the form $\mathtt{X}f$, $\mathtt{Y}f$, $f\mathtt{U}g$, $f\mathtt{S}g$, $f\mathtt{U}_{[0,b]}g$, or $f\mathtt{S}_{[0,b]}g$, and x_φ is the fresh boolean output variable for φ. Then, for every fair run r in $\mathcal{D}$, there is a corresponding fair run ρ of T_φ such that for each $i \geqslant 0$, $(T_\varphi, \rho, i) \models x_\varphi$ iff $(\mathcal{D}, r, i) \models \varphi$.

The soundness and completeness of the tester for a principally temporal formula can be proven by induction on the structure of the formula. These proofs are tedious and long, and so we omit them here due to limited space.

4.4 Basic symbolic model checking for CTL with OBDDs

Given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$, a state s in $\mathcal{D}$, and an RTCTL* formula $\varphi = \mathtt{E}f$, the model checking problem of $(\mathcal{D}, s) \models \mathtt{E}f$ is reduced to that of $(\mathcal{D} \parallel T_f, s') \models \mathtt{E}\chi(f)$, where s' is the state s augmented with some assignment to the fresh boolean variables in $\mathrm{var}s(\chi(f))$, where $\chi(f)$ is a state formula corresponding to f, which may include some maximal state subformulae in the same form of $\mathtt{E}g$. Such maximal state subformulae can be recursively treated in the same manner presented here.

For the method presented in this paper, the construction of testers for bounded temporal formulae and the computation of fair states are based on the basic symbolic model checking method for CTL with OBDDs. OBDDs represent a canonical representation form for boolean formulae. The following CTL model checking algorithms are called symbolic because both the representations of JDSs and the algorithms themselves can be implemented by OBDDs. Given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ and CTL formulae f, g over V, we first define the preimage of f in $\mathcal{D}$ as $\mathrm{EX}(f, \mathcal{D}) = \exists V'.(R(V, V') \wedge f(V'))$. This follows straightforwardly from the semantics of $\mathtt{EX}f$, which is true in a state if that state has a $\mathcal{D}$-successor in which f is true. From the equation $\mathtt{AX}f \equiv \neg\mathtt{EX}\neg f$, we define $\mathrm{AX}(f, \mathcal{D}) = \forall V'.(R(V, V') \rightarrow f(V'))$, which is true in a state if f is true in all $\mathcal{D}$-successors of that state. The set of states in $\mathcal{D}$ satisfying $\mathtt{E}(f\mathtt{U}g)$ can be calculated through the following least fixpoint computation $\mu Z.\tau(Z)$ of the monotonic predicate transformer $\tau(Z) = g \vee (f \wedge \mathtt{EX}Z)$ on a set Z of states: $\mathrm{EU}(f, g, \mathcal{D}) = \mu Z.(g \vee (f \wedge \mathrm{EX}(Z, \mathcal{D})))$.

We say a run of $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ is fair if each justice constraint in $\mathcal{J}$ holds infinitely often along the run. A state of $\mathcal{D}$ is called fair if it is a state of a fair run. The set of fair states in $\mathcal{D}$ can be calculated through the following greatest fixpoint computation $\nu Z.\tau(Z)$: $\mathrm{Fair}(\mathcal{D}) = \nu \mathrm{Z}.(\bigwedge_{\mathrm{J} \in \mathcal{J}} \mathrm{EX}(\mathrm{EU}(\top, \mathrm{Z} \wedge \mathrm{J}, \mathcal{D}), \mathcal{D}))$. From the calculation of fair states, it is easy to see that any state that can reach a fair state within finite steps is also fair. Note that the calculations above for $\mathtt{EX}$, $\mathtt{AX}$, and $\mathtt{EU}$ are unfair, owing to the lack of a justice constraint. To make these fair under the justice constraints $\mathcal{J}$, we can simply execute them as $\mathrm{EX}(f \wedge \mathrm{Fair}(\mathcal{D}), \mathcal{D})$, $\mathrm{AX}(f \wedge \mathrm{Fair}(\mathcal{D}), \mathcal{D})$, and $\mathrm{EU}(f, g \wedge \mathrm{Fair}(\mathcal{D}), \mathcal{D})$. If $\mathcal{J} = \emptyset$, we simply let $\mathrm{Fair}(\mathcal{D}) = \top$. From now on, we write $[\![\varphi, \mathcal{D}]\!]$ to denote the set of fair states in a JDS $\mathcal{D}$ satisfying φ, where φ is an RTCTL* formula over the set of state variables of $\mathcal{D}$.

4.5 Symbolic model checking for arbitrary RTCTL* formulae

4.5.1 *Symbolic model checking for $\mathtt{E}\psi$*

We say that an RTCTL* formula φ holds on a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ if $(\mathcal{D}, s) \models \varphi$ for every state s satisfying the initial condition Θ. To implement the symbolic model checking by using OBDDs, we first symbolically calculate and store the set of fair states in the JDS $\mathcal{D}$ satisfying φ, denoted by $[\![\varphi, \mathcal{D}]\!]$, as an OBDD. Then, $\Theta \subseteq [\![\varphi, \mathcal{D}]\!]$ implies that φ holds on $\mathcal{D}$, i.e., $\mathcal{D} \models \varphi$.

From the definition of the function $\chi(\varphi)$, each state subformula $\mathtt{E}\psi$ of φ will be model checked, and then replaced by the resulting OBDD. Given an RTCTL* formula $\mathtt{E}\psi$, the set of fair states in $\mathcal{D}$ satisfying $\mathtt{E}\psi$ can be calculated by existentially quantifying the fresh variables in X_ψ from the set of fair states satisfying $\chi(\psi)$ in the augmented JDS $\mathcal{D} \parallel T_\psi$. That is, $[\![\mathtt{E}\psi, \mathcal{D}]\!] = \exists X_\psi.(\chi(\psi) \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\psi))$. Because $\mathtt{E}\psi$ is a

state formula, we construct $T_{\mathtt{E}\psi}$ as a "valid" tester $(\emptyset, \top, \top, \emptyset)$, which is just used for the synchronous parallel composition of testers for other subformulae. The correctness of the computation for $[\![\mathtt{E}\psi, \mathcal{D}]\!]$ is demonstrated by Lemma 3.

Lemma 3. Given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ and an RTCTL* formula $\mathtt{E}\psi$, for each state s of $\mathcal{D}$, $(\mathcal{D}, s) \models \mathtt{E}\psi$ iff $(\mathcal{D}, s) \models \exists X_\psi.(\chi(\psi) \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\psi))$.

4.5.2 *Correctness of the tester construction for arbitrary RTCTL* formulae*

Theorem 1 shows that the tester T_φ for an arbitrary RTCTL* formula φ is sound and complete.

Theorem 1. The tester for any RTCTL* formula is sound and complete.

Proof. Let φ be an RTCTL* formula. We prove this theorem by induction on the structure of φ:

(1) If $\varphi = \neg f$, then $T_\varphi = T_f$. We adopt the induction hypothesis that T_f is sound and complete, which means that T_φ is also sound and complete.

(2) If $\varphi = f \wedge g$, then $T_\varphi = T_f \parallel T_g$. By the induction hypothesis that T_f and T_g are sound and complete and the definition of synchronous parallel composition, we have that T_φ is sound and complete.

(3) If φ is a state formula, i.e., an assertion over V or in the form of $\mathtt{E}f$, then $\chi(\varphi) = \varphi$, $X_\varphi = \emptyset$, and $T_\varphi = (\emptyset, \top, \top, \emptyset)$. The proof for this is trivial.

(4) If $\varphi = \mathtt{X}f, \mathtt{Y}f, f\mathtt{U}g, f\mathtt{S}g, f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$, then we conclude that T_φ is sound and complete directly by applying Lemmas 1 and 2, and the induction hypothesis that T_f and T_g are sound and complete.

(5) If $\varphi = f\mathtt{U}_{[a,b]}g$, then the tester T_φ is exactly constructed for $\{f \wedge \mathtt{X}()^a \ f\mathtt{U}_{[0,b-a]}g \ ()\}^a$. By the induction hypothesis that the testers for each of f and g are sound and complete, and by considering the other cases, we conclude that the tester for $\{f \wedge \mathtt{X}()^a \ f\mathtt{U}_{[0,b-a]}g \ ()\}^a$ is also sound and complete.

(6) If $\varphi = f\mathtt{S}_{[a,b]}g$, then the tester T_φ is exactly constructed for $\{f \wedge \mathtt{Y}()^a \ f\mathtt{S}_{[0,b-a]}g \ ()\}^a$. By the induction hypothesis that the testers for each of f and g are sound and complete, and by considering the other cases, we conclude that the tester for $\{f \wedge \mathtt{Y}()^a \ f\mathtt{S}_{[0,b-a]}g \ ()\}^a$ is also sound and complete.

Corollary 1 provides the method for constructing the set of states in $\mathcal{D}$ satisfying an RTCTL* formula φ.

Corollary 1. Let $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ be a JDS and φ an arbitrary RTCTL* formula. Then, $[\![\varphi, \mathcal{D}]\!]$, the set of fair states of $\mathcal{D}$ satisfying φ, is characterized by $\exists X_\varphi.(\chi(\varphi) \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\varphi))$.

Proof. By Theorem 1, the model checking problem of the RTCTL* formula φ over runs of $\mathcal{D}$ is reduced to that of the corresponding state formula $\chi(\varphi)$ over states of the combined JDS $\mathcal{D} \parallel T_\varphi$. We know that $[\![\chi(\varphi), \mathcal{D} \parallel T_\varphi]\!]$, the set of fair states of $\mathcal{D} \parallel T_\varphi$ satisfying $\chi(\varphi)$, is characterized by $\chi(\varphi) \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\varphi)$. From the fact that $\mathcal{D} \parallel T_\varphi$ constitutes the synchronous parallel composition of $\mathcal{D}$ and T_φ, we have that each state of $\mathcal{D} \parallel T_\varphi$ is a state of $\mathcal{D}$ augmented with an interpretation of the fresh variables in X_φ. Therefore, by existentially quantifying out X_φ from $\chi(\varphi) \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\varphi)$, we obtain the set of fair states of $\mathcal{D}$ satisfying φ, i.e., $[\![\varphi, \mathcal{D}]\!]$.

If we add the path quantifier $\mathtt{A}$ to RTCTL*, then $[\![\mathtt{A}\psi, \mathcal{D}]\!]$ must be calculated directly. For this, we employ the following corollary and $[\![\mathtt{A}\psi, \mathcal{D}]\!] = \forall X_\psi.(\mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\psi) \rightarrow \chi(\psi))$.

Corollary 2. Given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ and an RTCTL* formula $\mathtt{A}\psi$, for each state s of $\mathcal{D}$, $(\mathcal{D}, s) \models \mathtt{A}\psi$ iff $(\mathcal{D}, s) \models \forall X_\psi.(\mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\psi) \rightarrow \chi(\psi))$.

Proof. By the equation $\mathtt{A}\psi \equiv \neg\mathtt{E}\neg\psi$ and Lemma 3, we have that $[\![\mathtt{A}\psi, \mathcal{D}]\!] = \neg[\![\mathtt{E}\neg\psi, \mathcal{D}]\!] = \neg\exists X_\psi.(\chi(\neg\psi) \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_{\neg\psi})) = \forall \mathrm{X}_\psi.(\mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_{\neg\psi}) \rightarrow \neg\chi(\neg\psi))$, which is equivalent to $\forall X_\psi.(\mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\psi) \rightarrow \chi(\psi))$ by the fact that $\chi(\neg\psi) = \neg\chi(\psi)$ and $T_{\neg\psi} = T_\psi$.

4.5.3 *Symbolic model checking algorithm for RTCTL**

In Algorithm 1, given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ and an RTCTL* formula φ, we propose the algorithm $\mathrm{Tester}(\varphi, \mathcal{D}, b_\varphi, T_\varphi)$ to construct the tester T_φ for φ and the OBDD b_φ characterizing $\chi(\varphi)$. Then, $[\![\varphi, \mathcal{D}]\!]$, the set of fair states of $\mathcal{D}$ satisfying φ, is characterized by $\exists X_\varphi.(b_\varphi \wedge \mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\varphi))$.

To improve the efficiency of Algorithm 1, in lines 1–11 we apply the RTCTL model checking algorithm once it is applicable. That is, when φ is of the form $\mathcal{Q}\mathcal{T}f$, $\mathcal{Q}(f\mathcal{T}g)$, $\mathcal{Q}\neg\mathcal{T}f$, or $\mathcal{Q}\neg(f\mathcal{T}g)$, where $\mathcal{Q}$ is a path quantifier and $\mathcal{T}$ is a (bounded) future temporal operator. Here, $\varphi[f/b_f, g/b_g]$ is the formula

resulting for φ by replacing f with b_f and g with b_g. We use checkRTCTL$(\varphi, \mathcal{D})$ to denote the symbolic model checking algorithm for RTCTL that outputs the set of fair states of $\mathcal{D}$ satisfying the RTCTL formula φ. After generating b_f (the OBDD of $\chi(f)$) and b_g (the OBDD of $\chi(g)$), the problem of checking the RTCTL* formula φ on $\mathcal{D}$ can be reduced to that of checking the RTCTL formula $\varphi[f/b_f, g/b_g]$ on $\mathcal{D} \parallel T_f \parallel T_g$. Thus, the set of fair states in $\mathcal{D}$ satisfying φ can be characterized by existentially quantifying X_f and X_g from checkRTCTL$(\varphi[f/b_f, g/b_g], \mathcal{D} \parallel T_f \parallel T_g)$, such that the resulting OBDD is only over the set of state variables in $\mathcal{D}$.

Algorithm 1: $Tester(\varphi, \mathcal{D}, b_\varphi, T_\varphi)$

input : (1) φ: an *RTCTL** formula; (2) $\mathcal{D}$: a JDS and $\mathcal{D} = (V, \Theta, R, \mathcal{J})$.
output: (3) b_φ: the OBDD characterizing $\chi(\varphi)$; (4) T_φ: the tester for φ.

1 **if** $\varphi = \mathcal{Q}Xf$ and $\mathcal{Q} \in \{A, E\}$ **then**
2 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $T_\varphi := (\emptyset, \top, \top, \emptyset)$; $b_\varphi := \exists X_f.checkRTCTL(\varphi[f/b_f], \mathcal{D} \parallel T_f)$; destroy T_f;
3 **else if** $\varphi = \mathcal{Q}\neg Xf$ and $\mathcal{Q} \in \{A, E\}$ **then**
4 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $T_\varphi := (\emptyset, \top, \top, \emptyset)$; **if** $\mathcal{Q} = A$ **then** $\mathcal{Q}' := \mathtt{E}$; **else** $\mathcal{Q}' := \mathtt{A}$;
5 $\quad b_\varphi := \exists X_f.checkRTCTL(\neg\mathcal{Q}'\mathtt{X}f[f/b_f], \mathcal{D} \parallel T_f)$; destroy T_f;
6 **else if** $\varphi = \mathcal{Q}(f\mathcal{T}g)$ and $\mathcal{Q} \in \{A, E\}$ and $\mathcal{T} \in \{U, U_{[a,b]}\}$ **then**
7 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $Tester(g, \mathcal{D}, b_g, T_g)$; $T_\varphi := (\emptyset, \top, \top, \emptyset)$;
8 $\quad b_\varphi := \exists X_f.\exists X_g.checkRTCTL(\varphi[f/b_f, g/b_g], \mathcal{D} \parallel T_f \parallel T_g)$; destroy T_f and T_g;
9 **else if** $\varphi = \mathcal{Q}\neg(f\mathcal{T}g)$ and $\mathcal{Q} \in \{A, E\}$ and $\mathcal{T} \in \{U, U_{[a,b]}\}$ **then**
10 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $Tester(g, \mathcal{D}, b_g, T_g)$; $T_\varphi := (\emptyset, \top, \top, \emptyset)$; **if** $\mathcal{Q} = A$ **then** $\mathcal{Q}' := \mathtt{E}$; **else** $\mathcal{Q}' := \mathtt{A}$;
11 $\quad b_\varphi := \exists X_f.\exists X_g.checkRTCTL(\neg\mathcal{Q}'(f\mathcal{T}g)[f/b_f, g/b_g], \mathcal{D} \parallel T_f \parallel T_g)$; destroy T_f and T_g;
12 **else if** $\varphi = Ef$ **then** $Tester(f, \mathcal{D}, b_f, T_f)$; $b_\varphi := \exists X_f.(Fair(\mathcal{D} \parallel T_f) \wedge b_f)$; $T_\varphi := (\emptyset, \top, \top, \emptyset)$; destroy T_f;
13 **else if** $\varphi = Af$ **then** $Tester(f, \mathcal{D}, b_f, T_f)$; $b_\varphi := \forall X_f.(Fair(\mathcal{D} \parallel T_f) \rightarrow b_f)$; $T_\varphi := (\emptyset, \top, \top, \emptyset)$; destroy T_f;
14 **else if** φ is an assertion over V **then** $b_\varphi :=$ OBDD of φ; $T_\varphi := (\emptyset, \top, \top, \emptyset)$; // T_φ is set as a valid tester
15 **else if** $\varphi = \neg f$ **then** $Tester(f, \mathcal{D}, b_f, T_f)$; $b_\varphi := \neg b_f$; $T_\varphi := T_f$;
16 **else if** $\varphi = f \wedge g$ **then**
17 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $Tester(g, \mathcal{D}, b_g, T_g)$; $b_\varphi := b_f \wedge b_g$; $T_\varphi := T_f \parallel T_g$; destroy T_f and T_g;
18 **else if** $\varphi = Xf$ *or* Yf **then**
19 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $\varphi' := \varphi[f/b_f]$; $T_{\varphi'} := (V_{\varphi'}, \Theta_{\varphi'}, R_{\varphi'}, \mathcal{J}_{\varphi'}) \parallel T_f$; // by (2), (6)
20 $\quad b_\varphi := x_{\varphi'}$; destroy T_f; // $x_{\varphi'}$ is the fresh boolean output variable of $T_{\varphi'}$
21 **else if** $\varphi = fUg$, $fU_{[0,b]}g$, fSg *or* $fS_{[0,b]}g$ **then**
22 $\quad Tester(f, \mathcal{D}, b_f, T_f)$; $Tester(g, \mathcal{D}, b_g, T_g)$; $\varphi' := \varphi[f/b_f, g/b_g]$;
23 $\quad T_{\varphi'} := (V_{\varphi'}, \Theta_{\varphi'}, R_{\varphi'}, \mathcal{J}_{\varphi'}) \parallel T_f \parallel T_g$; // by (3), (5), (7), (9)
24 $\quad b_\varphi := x_{\varphi'}$; destroy T_f and T_g; // $x_{\varphi'}$ is the fresh boolean output variable of $T_{\varphi'}$
25 **else if** $\varphi = fU_{[a,b]}g$ and $0 < a \leqslant b$ **then** $\varphi' := \{f \wedge \mathtt{X}(\}^a fU_{[0,b-a]}g\{)\}^a$; $Tester(\varphi', \mathcal{D}, b_\varphi, T_\varphi)$; //by (4)
26 **else if** $\varphi = fS_{[a,b]}g$ and $0 < a \leqslant b$ **then** $\varphi' := \{f \wedge \mathtt{Y}(\}^a fS_{[0,b-a]}g\{)\}^a$; $Tester(\varphi', \mathcal{D}, b_\varphi, T_\varphi)$; //by (8)

Before model checking a given RTCTL* formula φ, it needs to be simplified by distributing negations over logical connectives, path quantifiers, and the temporal operator X so that they are only applied to temporal operators or assertions. For this, we define a function simp(φ) that repeatedly applies the rewriting rules in Figure 1 and the equations $f \equiv \neg\neg f$, $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $\neg\mathtt{X}f \equiv \mathtt{X}\neg f$, $\mathtt{A}f \equiv \neg\mathtt{E}\neg f$, and $\forall V.f \equiv \neg\exists V.\neg f$ to φ, until it is simplified completely such that the output formula does not contain redundant "$\neg$" signs and only includes six basic temporal operators $\mathtt{X}, \mathtt{Y}, \mathtt{U}, \mathtt{S}, \mathtt{U}_{[0,b]}$, and $\mathtt{S}_{[0,b]}$. Thus, $\chi(\text{simp}(\varphi))$ is a state formula in which negations are only applied to assertions and the fresh boolean variables for each principally temporal subformula.

Given a JDS $\mathcal{D} = (V, \Theta, R, \mathcal{J})$ and an RTCTL* formula φ, the main procedure Check$(\varphi, \mathcal{D})$ in Algorithm 2 determines the satisfiability of φ over $\mathcal{D}$, i.e., $\mathcal{D} \models \varphi$. Note that if φ is not a state formula, then $\mathcal{D} \models \varphi$ iff φ holds over any run starting from any initial state, so that line 1 will add a path quantifier A preceding φ when φ is not a state formula. Then, the negation of φ or $\mathtt{A}\varphi$ is simplified to ψ. Clearly, ψ is a state formula such that $X_\psi = \emptyset$ and $\mathcal{D} \parallel T_\psi = \mathcal{D}$. Line 2 obtains the OBDD b_ψ over V characterizing $\chi(\psi)$, i.e., ψ. The condition $(\Theta \wedge b_\psi \wedge \text{Fair}(\mathcal{D})) = \bot$ of line 3 means that there is not any initial fair state of $\mathcal{D}$ satisfying ψ. In other words, every fair run starting from every initial state of $\mathcal{D}$ satisfies φ, i.e.,

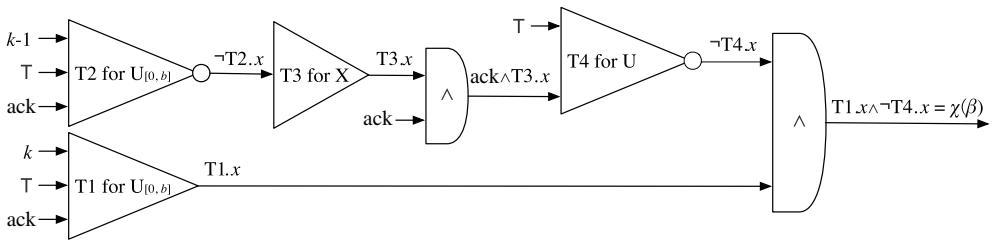

Figure 4 Composition of transducers to form the tester for subformula $\beta = (\top \mathtt{U}_{[0,k]} \mathrm{ack}) \wedge \neg(\top \mathtt{U}(\mathrm{ack} \wedge \mathtt{X}\neg(\top \mathtt{U}_{[0,k-1]} \mathrm{ack})))$.

$\mathcal{D} \models \varphi$. Otherwise, $\mathcal{D} \not\models \varphi$, and the output is "No".

Algorithm 2: Check$(\varphi, \mathcal{D})$

input : (1) φ: an RTCTL* formula; (2) $\mathcal{D}$: a JDS and $\mathcal{D} = (V, \Theta, R, \mathcal{J})$.
output: If $\mathcal{D} \models \varphi$ then outputs "Yes", or else outputs "No".

1 **if** φ is not a state formula **then** $\psi := \mathrm{simp}(\neg \mathtt{A}\varphi)$; **else** $\psi := \mathrm{simp}(\neg\varphi)$;
2 Tester$(\psi, \mathcal{D}, b_\psi, T_\psi)$;
3 **if** $(\Theta \wedge b_\psi \wedge \mathrm{Fair}(\mathcal{D})) = \bot$ **then** output "Yes"; **else** output "No" ;

4.5.4 *Example to demonstrate the algorithm*

We take the RTCTL* formula (1) from Section 1, i.e., $\varphi = \mathtt{AG}(\mathrm{req} \rightarrow \mathtt{E}(\mathtt{F}_{[0,k]}\mathrm{ack} \wedge \mathtt{G}(\mathrm{ack} \rightarrow \mathtt{F}_{[1,k]}\mathrm{ack})))$, as an example formula to demonstrate the model checking process. In the main algorithm Check$(\varphi, \mathcal{D})$, because φ is a state formula we first simplify $\neg\varphi$ as $\psi = \mathrm{simp}(\neg\varphi) = \mathtt{EF}(\mathrm{req} \wedge \neg\mathtt{E}(\mathtt{F}_{[0,k]}\mathrm{ack} \wedge \mathtt{G}(\mathrm{ack} \rightarrow \mathtt{F}_{[1,k]}\mathrm{ack}))) = \mathtt{E}(\top \mathtt{U}(\mathrm{req} \wedge \neg\mathtt{E}\beta))$, where $\beta = (\top \mathtt{U}_{[0,k]}\mathrm{ack}) \wedge \neg(\top \mathtt{U}(ack \wedge \mathtt{X}\neg(\top \mathtt{U}_{[0,k-1]}\mathrm{ack})))$. Then, the algorithm Tester$(\psi, \mathcal{D}, b_\psi, T_\psi)$ is invoked to construct the OBDD b_ψ for $\chi(\psi)$ and the tester T_ψ. We explain the algorithm in a bottom-up manner over the syntax tree of ψ. β is a path subformula without any path quantifier. Tester$(\beta, \mathcal{D}, b_\beta, T_\beta)$ is invoked to construct the OBDD b_β of $\chi(\beta)$ and the tester T_β. The construction process is illustrated in Figure 4. A tester (transducer) is denoted by a triangle. A triangle with one input (on the left side) denotes the tester for $\mathtt{X}f$, and the input is f. A triangle with two inputs denotes the tester for $f\mathtt{U}g$, and the inputs from top to bottom are f and g, respectively. A triangle with three inputs denotes the tester for $f\mathtt{U}_{[0,b]}g$, and the inputs from top to bottom are b, f, and g, respectively. Each tester Ti has only one output (on the right side), which is represented by the fresh boolean output variable (denoted by T$i.x$) of the tester. Therefore, the four testers T1∼T4 are constructed in bottom-up manner over the syntax tree of β: T1 for $\top \mathtt{U}_{[0,k]}$ack, T2 for $\top \mathtt{U}_{[0,k-1]}$ack, T3 for $\mathtt{X}\neg$T2.x, and T4 for $\top \mathtt{U}(\mathrm{ack} \wedge \mathrm{T3}.x)$. Thus, we take T1 ∥ T2 ∥ T3 ∥ T4 as the tester T_β and take the final output T1.$x \wedge \neg$T4.x as $\chi(\beta)$. Each input/output is a formula that contains arithmetic or boolean operations, which can be efficiently encoded as an OBDD.

The OBDD of the set of states in $\mathcal{D}$ satisfying $\mathtt{E}\beta$, i.e., $[\![\mathtt{E}\beta, \mathcal{D}]\!]$, is further encoded as $\exists X_\beta.(\mathrm{Fair}(\mathcal{D} \parallel \mathrm{T}_\beta) \wedge \chi(\beta)) = \exists\{\mathrm{T1.x}, \mathrm{T1.t}, \mathrm{T2.x}, \mathrm{T2.t}, \mathrm{T3.x}, \mathrm{T4.x}\}.(\mathrm{Fair}(\mathcal{D} \parallel \mathrm{T1} \parallel \mathrm{T2} \parallel \mathrm{T3} \parallel \mathrm{T4}) \wedge (\mathrm{T1.x} \wedge \neg\mathrm{T4.x}))$. For $\psi = \mathtt{E}(\top \mathtt{U}(\mathrm{req} \wedge \neg\mathtt{E}\beta))$, the principal path quantifier E is immediately followed by the temporal operator U, so we invoke the CTL algorithm EU$(\top, \mathrm{req} \wedge \neg[\![\mathtt{E}\beta, \mathcal{D}]\!], \mathcal{D})$ to efficiently calculate the OBDD of $[\![\psi, \mathcal{D}]\!]$. That is, b_ψ returned from the initially invoked Tester$(\psi, \mathcal{D}, b_\psi, T_\psi)$. Recall the original formula $\varphi = \neg\psi$. If there does not exist any fair initial state satisfying b_ψ (verified by $(\Theta \wedge b_\psi \wedge \mathrm{Fair}(\mathcal{D})) = \bot$), then we have that $\mathcal{D} \models \varphi$. Otherwise, $\mathcal{D} \not\models \varphi$.

4.6 Expressiveness and complexity of RTCTL*

Theorem 2 (Expressiveness of RTCTL*)**.** Each RTCTL* formula can be translated into an equivalent CTL* formula, and vice versa.

Proof. The direction from CTL* to RTCTL* is trivial, because any CTL* formula is in RTCTL*. For the opposite direction, owing to the fact that RTCTL* is an extension of CTL* with two basic bounded temporal operators $\mathtt{U}_{[a,b]}$ and $\mathtt{S}_{[a,b]}$, we present the following two expansion equations:

$$f\mathtt{U}_{[a,b]}g \equiv \begin{cases} f \wedge \mathtt{X}(f\mathtt{U}_{[a-1,b-1]}g), & \text{for } a>0, b \geqslant a; \\ g \vee (f \wedge \mathtt{X}(f\mathtt{U}_{[0,b-1]}g)), & \text{for } a=0, b>0; \\ g, & \text{for } a=0, b=0, \end{cases} \tag{10}$$

$$f\mathtt{S}_{[a,b]}g \equiv \begin{cases} f \wedge \mathtt{Y}(f\mathtt{S}_{[a-1,b-1]}g), & \text{for } a>0, b \geqslant a; \\ g \vee (f \wedge \mathtt{Y}(f\mathtt{S}_{[0,b-1]}g)), & \text{for } a=0, b>0; \\ g, & \text{for } a=0, b=0. \end{cases} \tag{11}$$

Then, if an RTCTL* formula φ is of the form $f\mathtt{U}_{[a,b]}g$, then φ can be translated into a pure CTL* formula φ' by the exhaustive application of (10): $\{f \wedge \mathtt{X}(\}^a\{g \vee (f \wedge \mathtt{X}(\}^{b-a}g\{)\}^{2b-a}$. Similarly, if an RTCTL* formula φ is of the form $f\mathtt{S}_{[a,b]}g$, then φ can be translated into a pure CTL* formula φ' by the exhaustive application of (11): $\{f \wedge \mathtt{Y}(\}^a\{g \vee (f \wedge \mathtt{Y}(\}^{b-a}g\{)\}^{2b-a}$. Thus, we conclude that any RTCTL* formula can be translated into CTL* by the exhaustive application of (10) and (11).

By Theorem 2, we conclude that RTCTL* has equal expressive power to CTL*. After translating an RTCTL* formula φ into a CTL* forumula φ', the model checking problem of RTCTL* can be reduced to that of CTL*. We call this model checking method "translation-based."

Theorem 3 (Complexity of the translation from RTCTL* to CTL*)**.** Any RTCTL* formula φ can be translated into an equivalent CTL* formula of length $2^{O(|\varphi|)}$.

As shown in the proof of Theorem 2, any RTCTL* formula φ can be translated into a CTL* formula φ'. We prove the theorem by induction on the structure of φ.

(1) If φ is also a CTL* formula, then the theorem holds immediately.

(2) If $\varphi = f\mathtt{U}_{[a,b]}g$, then we have that $|\varphi| = |f| + |g| + \lceil \log_2 a \rceil + \lceil \log_2 b \rceil + 1$. By the induction hypothesis, there are two CTL* formulae f' and g' that are equivalent to two RTCTL* formulae f and g, respectively. Meanwhile, $|f'| = 2^{O(|f|)}$ and $|g'| = 2^{O(|g|)}$. Then, from the equation $f\mathtt{U}_{[a,b]}g \equiv \{f \wedge \mathtt{X}(\}^a\{g \vee (f \wedge \mathtt{X}(\}^{b-a}g\{)\}^{2b-a}$, we have that $|\varphi'| = a(|f'|+2)+(b-a)(|f'|+|g'|+3)+|g'| = b|f'|+(b-a+1)|g'|+3b-a = b2^{O(|f|)}+(b-a+1)2^{O(|g|)}+3b-a$. In the case that $a=0$, $|\varphi| = |f|+|g|+\lceil \log_2 b \rceil + 2$, and $|\varphi'|$ attains the maximal value $b2^{O(|f|)}+(b+1)2^{O(|g|)}+3b$. Furthermore, we have that $|\varphi'| \leqslant b2^{O(|f|)}+(b+1)2^{O(|g|)}+3b \leqslant (b+1)(2^{O(|f|)}+2^{O(|g|)}+2^2) = (b+1)2^{O(|f|+|g|+2)} = 2^{O(|f|+|g|+\log_2(b+1)+2)} \leqslant 2^{O(|f|+|g|+\lceil \log_2 b \rceil+2)} = 2^{O(|\varphi|)}$.

(3) If $\varphi = f\mathtt{S}_{[a,b]}g$, then the proof is similar to that for the case with $\varphi = f\mathtt{U}_{[a,b]}g$.

However, the translation-based model checking method is often infeasible, even for RTCTL* formulae of small lengths. We know that the best currently known time complexity of a model checking algorithm for the CTL* formula ψ over a JDS $\mathcal{D}$ is $|\mathcal{D}| \cdot 2^{O(|\psi|)}$ [1], which is exponential in the length of the CTL* formula. Furthermore, in light of the fact that $|\varphi'| = 2^{O(|\varphi|)}$, from Theorem 3 we can conclude that checking φ' using an existing CTL* model checking algorithm requires time $|\mathcal{D}| \cdot 2^{O(2^{O(|\varphi|)})}$, which is doubly exponential in the length of the original RTCTL* formula φ.

Theorem 4 (Complexity of the tester construction for RTCTL*)**.** For an arbitrary RTCTL* formula φ, there exists a tester with $2^{O(|\varphi|)}$ fresh boolean variables. If every bounded principally temporal subformula in φ is of the form $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$, then the number of fresh boolean variables is linear in the length of φ.

Proof. We collect the fresh boolean variables introduced in constructing the tester of φ as ν_φ. Assuming that all additional operators have been eliminated by rewriting rules, we have the following assertions:

(1) If $\varphi = \mathtt{E}f$, then we will construct the tester for f. Thus, in the worst case φ should not include the path quantifier $\mathtt{E}$. In the following proof, we exclude this case.

(2) If the temporal operators in φ are restricted to $\mathtt{X}$, $\mathtt{Y}$, $\mathtt{U}$, or $\mathtt{S}$, then we have that $|\nu_\varphi| = O(|\varphi|)$, because one boolean variable is introduced for each principally temporal subformula, and there are only a linear number of subformulae.

(3) If the temporal operators in φ are restricted to $\mathtt{U}_{[0,b]}$ or $\mathtt{S}_{[0,b]}$, then each bounded temporal operator includes $1 + \log_2 0 + \log_2 b = \log_2 b + 2$ characters. The tester construction for each principally temporal

subformula introduces $\log_2 b+1$ boolean variables to encode one boolean output variable and one integer variable in the interval $[0, b-1]$. We also have that $|\nu_\varphi| = O(|\varphi|)$.

(4) If the temporal operators in φ are restricted to $\Omega_{[a,b]}$, where Ω denotes $\mathtt{U}$ or $\mathtt{S}$ and $0 < a \leqslant b$, then we have that $|\varphi| = |f| + |g| + \lceil \log_2 a \rceil + \lceil \log_2 b \rceil + 1$. We first translate each subformula $f\Omega_{[a,b]}g$ into $\beta = \{f \wedge \Delta(\}^a\ f\Omega_{[0,b-a]}g\ \{)\}^a$, where Δ denotes $\mathtt{X}$ or $\mathtt{Y}$ when Ω is $\mathtt{U}$ or $\mathtt{S}$, respectively. When constructing the tester for β, $\lceil \log_2(b-a) \rceil + 1$ fresh boolean variables are introduced for $f\Omega_{[0,b-a]}g$ and $a(|\nu_f|+1)$ fresh boolean variables are introduced for $\{f \wedge \Delta(\}^a\ \ldots\{)\}^a$. We prove that $|\nu_\varphi| = 2^{O(|\varphi|)}$ by induction on the structure of the formula. By the induction hypothesis, we have that $|\nu_f| = 2^{O(|f|)}$ and $|\nu_g| = 2^{O(|g|)}$. Then, $|\nu_\varphi| = a(|\nu_f|+1) + |\nu_f| + |\nu_g| + \lceil \log_2(b-a) \rceil + 1 = (a+1)|\nu_f| + |\nu_g| + a + \lceil \log_2(b-a) \rceil + 1 = (a+1)2^{O(|f|)} + 2^{O(|g|)} + a + \lceil \log_2(b-a) \rceil + 1 = 2^{O(|f|+\log_2(a+1))} + 2^{O(|g|)} + a + \lceil \log_2(b-a) \rceil + 1 \leqslant 2^{O(|f|+|g|+\log_2(a+1))} + 2b \leqslant 2^{O(|f|+|g|+\lceil \log_2 a \rceil + \lceil \log_2 b \rceil + 1)} = 2^{O(|\varphi|)}$.

According to (10) and (11), we translate an RTCTL* bounded temporal formula $f\mathtt{U}_{[a,b]}g/f\mathtt{S}_{[a,b]}g$ to a formula with a $\mathtt{X}/\mathtt{Y}$ unbounded operators and $\mathtt{U}_{[0,b-a]}/\mathtt{S}_{[0,b-a]}$ bounded operators. Furthermore, from Theorems 3 and 4, we can conclude that the state space of our tester can be made exponentially smaller than the tester constructed using fully translation-based methods.

5 Implementation and experimental results

There are two options for implementing model checking algorithms, either direct implementation or reduction to existing tools. Developers often need to devote significant time and effort to a direct implementation. Furthermore, it is not easy to avoid introducing unexpected programming errors. Meanwhile, in a reduction-based approach the proposed model checking method can be reduced to verification using an existing model checker. This method is usually rapid, reliable, and extendable, because it constitutes a mechanical translation process, and does not (or only does rarely) require modification to the original checker. In this paper, we prefer to adopt a reduction-based approach.

To adopt this approach to the implementation of our method by building upon existing model checkers, we should eliminate path quantifiers from RTCTL* and obtain the real-time linear temporal logic RTLTL. As far as we know, this is still not fully supported by any existing model checker. The core problem of symbolic model checking for RTCTL* is to compute the set of states in a JDS satisfying a formula of the form $\mathtt{E}f$, where f is treated as an RTLTL subformula. The reason for this is that in checking an arbitrary RTCTL* formula $\mathtt{E}f$, f is treated as an "RTLTL" formula by allowing not only its state subformulae to be assertions, but also its maximal state subformulae of the form $\mathtt{E}g$, which have already been processed in a bottom-up manner and replaced by their corresponding sets of states. Thus, the model checking procedure for subformula of the form $\mathtt{E}f$ may be invoked more than once, and the resulting set of states must be integrated to obtain the final result. This process cannot be reduced to any existing model checker. This is the reason for tailoring RTCTL* to RTLTL, such that the reduction-based method can be adopted.

5.1 Implementation of symbolic model checking for RTLTL

We have already implemented a symbolic model checking prototype [1)] for pure RTLTL, based on the well-known symbolic model checker NuSMV, with the latest version 2.6.0. Figure 5 shows the model checking framework of the prototype. The prototype is an M4 [2)]-based preprocessor, which produces the tester T_φ for an RTLTL specification φ and maps φ to a semantically-equivalent state formula $\chi(\varphi)$. Thus, the model checking problem of $\mathcal{D} \models \varphi$ is reduced to that of $\mathcal{D} \parallel T_\varphi \models \chi(\varphi)$, which can be treated using the CTL or LTL model checking algorithm in NuSMV. Therefore, the biggest advantage of the framework is that it constitutes a rapid, reliable, extendable, and fully automatic method of implementing the proposed

1) We refer the reader to https://github.com/hovertiger/RTLTL-model-checker to download the source code of the prototype.

2) M4 is a general-purpose macro processor, available on most UNIX platforms.

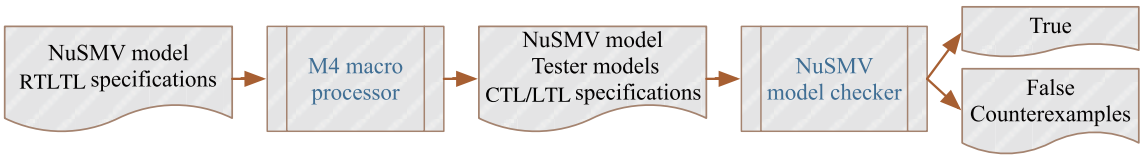

Figure 5 (Color online) Model checking framework for RTLTL.

Table 1 RTLTLSPEC syntax for principally temporal formulae [a)]

RTLTL	syntax	RTLTL	syntax	RTLTL	syntax	RTLTL	syntax
Xf	`NX(f)`	Yf	`NY(f)`	$f\mathtt{U}_{[a,b]}g$	`BU(a,b,f,g)`	$f\mathtt{S}_{[a,b]}g$	`BS(a,b,f,g)`
fUg	`NU(f,g)`	fSg	`NS(f,g)`	$\mathtt{F}_{[a,b]}f$	`BF(a,b,f)`	$f\mathtt{O}_{[a,b]}g$	`BO(a,b,f,g)`
Ff	`NF(f)`	Of	`NO(f)`	$\mathtt{G}_{[a,b]}f$	`BG(a,b,f)`	$f\mathtt{H}_{[a,b]}g$	`BH(a,b,f,g)`
Gf	`NG(f)`	Hf	`NH(f)`	Zf	`NZ(f)`		

a) $b < 0$ denotes ∞

method, without any modification to the source code of NuSMV. The only manual intervention required is the design of the NuSMV input program describing the JDS to be verified.

As shown in Figure 5, we design an extension of the NuSMV input language for RTLTL specifications. The extension is an RTLTL specification defined as "`RTLTL SPEC(f)`", where `f` is an RTLTL formula. In Table 1, we list the `RTLTLSPEC` syntax for principally temporal RTLTL formulae.

According to Figure 1, we first design some M4 macros to rewrite a formula with an additional principally temporal operator `NF,NG,NO,NH,NZ,BF,BG,BO,` or `BH` into a formula including only `NX,NU,NY,NS,BU,` and `BS` temporal operators. For example, `BG(a,b,f)` is written into `!BU(a,b,TRUE,!f)`. Based on (4)/(8), we design two M4 macros to further reduce each subformula `BU(a,b,f,g)`/`BS(a,b,f,g)` with `a>0` to a formula with subformula `BU(0,b-a,f,g)`/`BS(0,b-a,f,g)`, such that this subformula can be efficiently tackled based on its tester. The M4 macro for reducing `BU(a,b,f,g)` proceeds as (12a)–(12g), in which the code of lines (12c), (12d) and (12f) constitutes the recursive implementation of (4), and the code of lines (12b) and (12e) enacts the recursive implementation of $f\mathtt{U}_{[a,\infty]}g \equiv f\mathtt{U}_{[a,a]}(f\mathtt{U}g)$. Note that we allow ∞ to be denoted by a negative integer, and so $f\mathtt{U}_{[a,\infty]}g$ is denoted by `BU(a,-1,f,g)`. The M4 macro for reducing `BS(a,b,f,g)` is similar, and is omitted here,

$$\texttt{BU(a,b,f,g)} \Rightarrow \begin{cases} \text{“}\texttt{[Syntax error: a<0]}\text{”}, & \text{if } \texttt{a} < 0; \quad (12a)\\ \texttt{NU(f,g)}, & \text{if } \texttt{a} = 0 \text{ and } \texttt{b} < 0; \quad (12b)\\ \texttt{g}, & \text{if } \texttt{a} = 0 \text{ and } \texttt{b} = 0; \quad (12c)\\ \texttt{BU(0,b,f,g)}, & \text{if } \texttt{a} = 0 \text{ and } \texttt{b} > 0; \quad (12d)\\ \texttt{f \& NX(BU(a-1,b,f,g))}, & \text{if } \texttt{a} > 0 \text{ and } \texttt{b} < 0; \quad (12e)\\ \texttt{f \& NX(BU(a-1,b-1,f,g))}, & \text{if } \texttt{a} > 0 \text{ and } \texttt{a} \leqslant \texttt{b}; \quad (12f)\\ \text{“}\texttt{[Syntax error: a>b]}\text{”}, & \text{if } \texttt{a} > 0 \text{ and } \texttt{a} > \texttt{b}. \quad (12g) \end{cases}$$

Given an `RTLTLSPEC` formula φ, we can now obtain a new `RTLTLSPEC` formula φ' that includes only the six basic temporal operators `NX,NY,NU,NS`, `BU`, and `BS`, with a lower bound of 0. After reducing φ to φ', we next construct the tester $T_{\varphi'}$ and the formula $\chi(\varphi')$ via M4. For each of the six basic temporal operators, we then design an M4 macro to (1) declare a new tester module instantiation as the tester T_ψ for the matching principally temporal subformula ψ in φ' in the variable declaration part of the main module, and (2) convert ψ into the output variable `x` of the tester instantiation, i.e., $\chi(\psi)$. Thus, the resulting formula converted from φ' forms the state formula $\chi(\varphi')$, which can be checked as a CTL or LTL specification using NuSMV. The list of newly declared tester module instantiations forms the tester for φ'. Finally, according to the tester construction method presented in Subsection 4.2, we design six NuSMV modules as the tester construction templates for the six basic temporal operators `NX`, `NY`, `NU`, `NS`, `BU`, and `BS` with 0 lower bound, as illustrated in Figure 6. The six tester module templates are appended to the end of the original NuSMV input program. This accomplishes the reduction from RTLTL to

```
MODULE Tester_X(f) -- tester for X f
VAR x : boolean;   -- fresh output variable
TRANS x <-> next(f);

MODULE Tester_Y(f) -- tester for Y f
VAR x : boolean;   -- fresh output variable
INIT !x;
TRANS next(x) <-> f;

MODULE Tester_U(f, g) -- tester for f U g
VAR x : boolean;      -- fresh output variable
TRANS x <-> (g | (f & next(x)));
JUSTICE x -> g;

MODULE Tester_S(f, g) -- tester for f S g
VAR x : boolean;      -- fresh output variable
INIT x <-> g;
TRANS next(x) <-> (next(g) | (next(f) & x));

MODULE Tester_BU0(b, f, g)  --tester for f U [0,b] g
VAR x : boolean;            --fresh output variable
    t : 0..b - 1;           --fresh integer variable
DEFINE s1 := (f & !g & !x); --s1-s4 are states in the tester
       s2 := (f & !g & x);
       s3 := (g & x);
       s4 := (!f & !g & !x);
TRANS -- transition relation of module Tester_BU0
  (s1 & next(s1))|(s1 & next(t)=0 & next(s2))|(s1 & next(s4))|
  (s2 & t>=0 & t<(b-1) & next(s2) & next(t)=t+1)|
  (s2 & t>=b-1 & next(s3))|(s3 & next(s1))|(s3 & next(s2))|
  (s3 & next(s3))|(s3 & next(s4))|(s4 & next(s1))|
  (s4 & next(s2))|(s4 & next(s3))|(s4 & next(s4));

MODULE Tester_BS0(b, f, g) -- tester for f S [0,b] g
VAR x : boolean;           -- fresh output variable
    t : 0..b - 1;          -- fresh integer variable
DEFINE s1 := (!f & g & x); -- s1-s5 are states in the tester
       s2 := (!f & !g & !x);
       s3 := (f & !g & !x);
       s4 := (f & !g & x);
       s5 := (f & g & x);
INIT !(s4 & t>=0 & t<=b-1) & (x<->g);
TRANS
  (s1 & next(s1))|(s1 & next(s2))|(s1 & next(s4) & next(t)=0)|
  (s1 & next(s5))|(s2 & next(s1))|(s2 & next(s2))|
  (s2 & next(s3))|(s2 & next(s5))|(s3 & next(s1))|
  (s3 & next(s2))|(s3 & next(s3))|(s3 & next(s5))|
  (s4 & next(s1))|(s4 & next(s2))|(s4 & t>=b-1 & next(s3))|
  (s4 & t>=0 & t<b-1 & next(s4) & next(t)=t+1)|(s4 & next(s5))|
  (s5 & next(s1))|(s5 & next(s2))|(s5 & next(s4) & next(t)=0)|
  (s5 & next(s5));
```

Figure 6 The NuSMV modules representing the testers for the basic temporal operators X, Y, U, S, $\mathtt{U}_{[0,b]}$, and $\mathtt{S}_{[0,b]}$.

CTL/LTL.

We consider the RTLTL formula $\varphi = f\mathtt{U}_{[20,50]}\mathtt{X}g$ as an example, where f and g are two boolean variables. We add the line "`RTLTLSPEC(BU(20,50,f,NX(g)))`" to the original NuSMV input program. This specification is first rewritten into the formula $\varphi' = \{\mathtt{f\ \&\ NX(}\}^{20}\ \mathtt{BU(0,30,f,NX(g))}\ \{\mathtt{)}\}^{20}$, which only includes basic temporal operators. On the syntax tree of φ', all temporal operators are dealt with in a bottom-up manner, which declares the following tester module instantiations and adds them to the variable declaration part of the `main` module:

```
T1 : Tester_X(g);              -- tester for NX(g)
T2 : Tester_BU0(30, f, T1.x); -- tester for BU(0,30,f,T1.x)
T3 : Tester_X(T2.x);           -- tester for NX(T2.x)
T4 : Tester_X(f & T3.x);       -- tester for NX(f & T2.x)
...
T22 : Tester_X(f & T21.x);     -- tester for NX(f & T21.x)
```

where `T1` and `T2` are the testers for `NX(g)` and `BU(0,30,f,T1.x)`, respectively. `T3-T22` are the testers for the `NX` operators, from right to left respectively, in φ'. Thus, φ' is further translated into $\chi(\varphi') =$ `f & T22.x` by M4. After adding the NuSMV source code in Figure 6 to the end of the original NuSMV input program, the line "`RTLTLSPEC(BU(20,50,f,NX(g)))`" is replaced with "`LTLSPEC f & T22.x;`" (checked as LTL) or "`SPEC f & T22.x;`" (checked as CTL). The counterexample of $f\mathtt{U}_{[20,50]}\mathtt{X}g$ can also be generated by NuSMV when the specification is checked as false under the extended NuSMV input program. One can understand this counterexample by replacing each output variable on the trace with the corresponding principally temporal subformula.

5.2 Experimental results

Our model checking prototype is implemented based on the latest NuSMV version 2.6.0, and supports the verification of full RTLTL formulae. NuSMV 2.6.0 supports BDD-based model checking for an extension of LTL, which is a subset of RTLTL that includes the (bounded) future temporal operators X, U, G, $\mathtt{G}_{[a,b]}$, F, and $\mathtt{F}_{[a,b]}$, and the (bounded) past temporal operators Y, Z, S, H, $\mathtt{H}_{[a,b]}$, O, and $\mathtt{O}_{[a,b]}$. Thus, it is natural to perform experimental comparisons between NuSMV and our method.

We now compare the efficiency of NuSMV with that of our method. Through our analysis of the source code of NuSMV, we find that before verification NuSMV translates each bounded principally temporal

```
MODULE main -- program (1)
DEFINE cb := 300;
       mb := 270;
VAR c : 0..cb;
    p : boolean; -- T1 is the tester for BU(0, mb, TRUE, !p)
    T1 : Tester_BU0(mb, TRUE, !p);
ASSIGN init(c) := 0;
       next(c) := case
         c<cb: c + 1;
         TRUE : c;
       esac;
       init(p) := TRUE;
       next(p) := case
         c<mb: TRUE;
         TRUE : {TRUE,FALSE};
       esac;
LTLSPEC G [0, mb] p; -- for testing NuSMV
LTLSPEC !T1.x;       -- for testing our method
```

```
MODULE main -- program (2)
DEFINE cb := 2000;
       mb := 1800;
VAR c : 0..cb+1; -- T1 is the tester for BS(0, mb, TRUE, p)
    p : boolean; -- T2 is the tester for TRUE U !(c=cb -> T1.x)
    T1 : Tester_BS0(TRUE, p, mb);
    T2 : Tester_U(TRUE, !(c=cb -> T1.x));
ASSIGN init(c) := 0;
       next(c) := case
         c<cb+1: c + 1;
         TRUE : c;
       esac;
       init(p) := TRUE;
       next(p) := case
         c<cb - mb: TRUE;
         TRUE : FALSE;
       esac;
LTLSPEC G(c=cb -> (O [0, mb ] p)); -- for testing NuSMV
LTLSPEC !T2.x;                     -- for testing our method
```

Figure 7 The NuSMV input programs for testing.

subformula into an equivalent LTL formula that includes only the operators `X` or `Y`, by applying rewriting rules. We have that the tester construction methods in NuSMV and in our method for `X`, `Y`, `U`, `S`, $f\mathtt{U}_{[a,a]}g$, and $f\mathtt{S}_{[a,a]}g$ are similar. The experimental results also show that the efficiency of verification for these operators is similar, and so we omit these results here. The main difference between NuSMV and our method lies in the tester constructions for $f\mathtt{U}_{[0,b]}g$ and $f\mathtt{S}_{[0,b]}g$. In the following, we present and compare the experimental results for the verification of the formulae of the two forms by NuSMV and our method.

We design two simple NuSMV input programs, presented in Figure 7, so that we can perform an experimental comparison between NuSMV and our method. Program (1) is designred to test formulae of the form $f\mathtt{U}_{[0,b]}g$, and program (2) is for testing formulae of the form $f\mathtt{S}_{[0,b]}g$. The two programs are verified on a virtual machine (Parallels Desktop 12 for Mac), allocated with two virtual CPUs and 8 GB memory. The virtual machine is installed on an Apple computer, equipped with a 3.3 GB Intel Core i5 CPU and 16 GB memory, which runs 64 bit Ubuntu Linux 16.04 LTS. The model checker employed by our method is also NuSMV 2.6.0.

In Figure 7, program (1) models a system in which the variable `p` is true in the initial state and stays true for the first `mb` steps. The constant `cb` is the upper bound for counting the number of steps. We successfully verified the bounded temporal formula $\mathtt{G}_{[0,mb]}p$ using both NuSMV and our method. For testing NuSMV, the formula is verified as the LTL specification `G [0,mb] p`. For testing our method, the formula is verified as the LTL specification `!T1.x`, where `T1.x` is the output variable of the tester `T1` for `BU(0, mb, TRUE, !p)`. The two specifications are verified independently 10 times, with different values for `cb` and `mb`, using the variable dynamic reordering function of the BDD package CUDD.

Program (2) models a system in which the variable `p` is true initially and stays true for the first `cb-mb` steps. After that, `p` stays false forever. The constant `cb` is the upper bound for counting the number of steps. We successfully verified the bounded temporal formula $\mathtt{G}((c = cb) \rightarrow \mathtt{O}_{[0,mb]}p)$ using both NuSMV and our method. For testing NuSMV, the formula is verified as the LTL specification `G(c=cb -> (O [0, mb] p))`. For testing our method, the formula is verified as the LTL specification `!T2.x`, where `T2.x` is the output variable of the tester `T2` for `TRUE U !(c=cb -> T1.x)`. `T1` is the tester for `BS(0, mb, TRUE, p)`. The two specifications are verified independently 10 times with different values of `cb` and `mb`, using the variable dynamic reordering function of the BDD package CUDD.

Figures 8 and 9 list the experimental results for the two programs, where the total verification time is the elapsed time for the model construction and verification. The memory in use and the peak number of BDD nodes show the space consumed by CUDD. In fact, to test the performance limit we also verified program (1) using NuSMV and our method for `cb` = 50000 and `mb` = 45000. NuSMV collapsed immediately, due to a memory segmentation fault. For our method, the total verification time was only 9 min 43.833 s, the memory in use by CUDD was 46 MB, the peak number of BDD nodes was 1103760, and the number of BDD variables was 69. Clearly, these experimental results demonstrate that in terms of the memory usage and particularly the time consumption, our method performs significantly better than NuSMV.

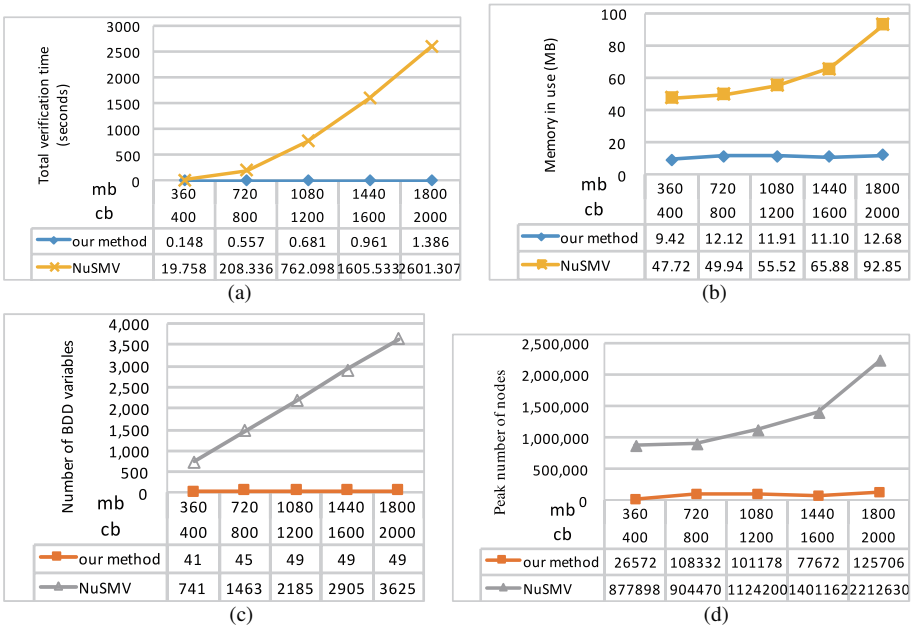

Figure 8 (Color online) Experimental comparison for checking $\mathtt{G}_{[0,mb]}p$ in program (1) using NuSMV and our method. (a) Total verification time; (b) memory in use; (c) number of BDD variables; (d) peak number of nodes.

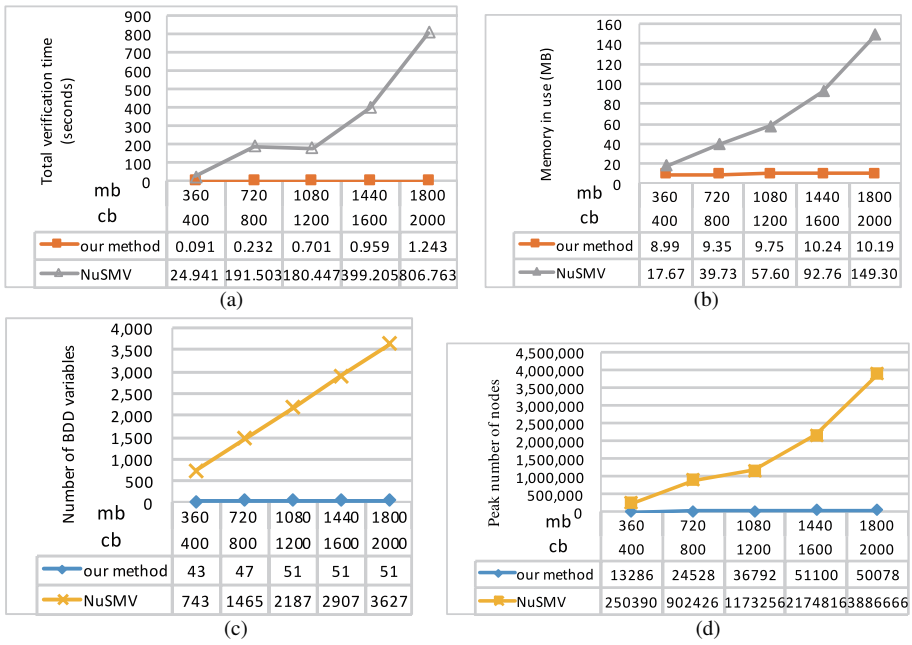

Figure 9 (Color online) Experimental comparison for checking $\mathtt{G}(c = cb \rightarrow \mathtt{O}_{[0,mb]}p)$ in program (2) using NuSMV and our method. (a) Total verification time; (b) memory in use; (c) number of BDD variables; (d) peak number of nodes.

5.3 Discussion of experimental results

Regarding temporal succinctness, the extended LTL in NuSMV cannot directly express the RTLTL formulae $f\mathtt{U}_{[a,b]}g$ and $f\mathtt{S}_{[a,b]}g$, which are two basic bounded temporal operators in RTLTL supported by our method. To verify an RTLTL formula of either of these forms using NuSMV, the formula must be translated into an equivalent LTL formula. Theorem 3 shows that the length of the translated LTL formula is exponential in the length of the RTLTL formula. Therefore, we can conclude that the temporal expressiveness of RTLTL is exponentially more succinct than that of the equivalent LTL in NuSMV.

Regarding the efficiency of NuSMV and our method, we first analyze the source code of NuSMV, and

learn that NuSMV adopts a fully translation-based method, which translates each bounded temporal formula into an LTL formula including only $\mathtt{X}$ and $\mathtt{Y}$. From Theorem 3, we have that the length of the translated formula is exponential in the length of the original formula. We can further conclude that for a bounded temporal formula, the translated-based model checking method in NuSMV is doubly exponential in the length of the original formula. This means that for verifying a bounded temporal formula, NuSMV is infeasible even for a system of small scale. Meanwhile, our method can be viewed as an important optimization on the tester construction for most bounded temporal formulae, because their testers are constructed based on the tester for $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$. In Theorem 4, we prove that the number of variables in our tester for either formula $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$ is linear in the length of the formula, so that the state space of the tester of our method is exponential in the length of the original formula. Therefore, for a bounded temporal formula the state space of our tester is exponentially smaller than that of NuSMV. This assertion is also justified by the experimental results presented in Figures 8 and 9. Thus, we can strongly recommend NuSMV developers to adopt our method.

6 Related work

Emerson et al. [20] were the first to propose the use of time bounded modalities in the branching time framework. They introduced a real-time computation tree logic RTCTL, which is an extension of CTL with real-time modalities, and developed an RTCTL model checking algorithm for discrete systems. Let $\mathtt{Q}$ be a path quantifier $\mathtt{A}$ or $\mathtt{E}$. Then, the basic real-time (time bounded) formulae of RTCTL include $\mathtt{Q}(f\mathtt{U}^{\leqslant k}g)$, $\mathtt{Q}(f\mathtt{U}^{=k}g)$, and $\mathtt{Q}(f\mathtt{U}^{\geqslant k}g)$, whose semantics are equivalent to the RTCTL* formulae $\mathtt{Q}(f\mathtt{U}_{[0,k]}g)$, $\mathtt{Q}(f\mathtt{U}_{[k,k]}g)$, and $\mathtt{Q}(f\mathtt{U}_{[k,\infty]}g)$, respectively. It is easy to see that the generic real-time formula $\mathtt{Q}(f\mathtt{U}^{[a,b]}g)$ can be expressed by $\mathtt{Q}(f\mathtt{U}^{=a}\mathtt{Q}(f\mathtt{U}^{\leqslant b-a}g))$. However, in RTCTL any (time bounded) temporal operator must be restricted directly by a path quantifier, while the RTCTL* in this paper does not entail such a restriction. Therefore, the expressive power of RTCTL* is stronger than that of RTCTL.

In [21], Fruth proposed runtime verification methods for real-time systems. The system properties are expressed through an extension of future LTL with real-time temporal operators $\mathtt{X}_a$ and $\mathtt{U}_{a,b}$, where the semantics of $\mathtt{U}_{a,b}$ is equal to that of $\mathtt{U}_{[a,b]}$ in this paper. $\mathtt{X}_a$ denotes a successive applications of $\mathtt{X}$. Thus, the extended LTL is a fragment of RTLTL that only supports future temporal operators, and not past temporal operators. The author proposed a tableau construction method to translate an extended LTL formula into a nondeterministic finite automaton (NFA) that accepts precisely the traces satisfying the formula. Then, the truth checking problem of whether a finite trace satisfies a formula can be solved by forward depth-first search algorithms. Unlike testers, NFAs do not possess compositionality, so that for a compound formula an NFA cannot be constructed by composing the NFAs of its subformulae. However, owing to compositionality the testers introduced in this paper are more functionally complex than NFAs, so that we can easily extend existing logics to support more valuable operators just by constructing the testers for these new operators.

Pnueli and Zaks [22] proposed the construction of temporal testers for formulae specified in LTL, PSL, and MITL, and also presented a general overview of the tester methodology. We list some comparisons between [22] and our method.

(1) The efficiency of our method is significantly better than that of [22]. We know that the state space of a JDS/tester is exponential in the number of variables. So for constructing a tester, the fewer number of fresh variables are created, the better efficiency will be obtained. In [22], the authors construct the tester of $f\mathtt{U}_{[a,b]}g$ as that of $\diamondsuit_a[\boxminus_{[0,a]}f \wedge (f\mathtt{U}_{[0,b-a]}g)]$, where $\diamondsuit_a$ is a "shift by a" operator that is equal to $\top\mathtt{U}_{[a,a]}$, and $\boxminus_{[0,a]}f$ is a past analog of $\Box_{[0,a]}f$ that is satisfied iff f has been continuously true for the last a time units. The tester of $f\mathtt{U}_{[a,b]}g$ is constructed as the synchronous parallel composition of the testers of $\boxminus_{[0,a]}f$, $f\mathtt{U}_{[0,b-a]}g$, and $\diamondsuit_a(x_{\boxminus_{[0,a]}f} \wedge x_{f\mathtt{U}_{[0,b-a]}g})$. Via our analysis, one boolean variable and one integer variable in $[0,a]$ should be created for the tester of $\boxminus_{[0,a]}f$, and one boolean variable and one integer variable in $[0,b-a]$ should be created for the tester of $f\mathtt{U}_{[0,b-a]}g$. The tester of $\diamondsuit_a f$ is constructed as the synchronous parallel composition of the testers $U, P, \mathrm{ON}[0], \mathrm{OFF}[0], \ldots, \mathrm{ON}[k-1]$ and $\mathrm{OFF}[k-1]$, where one boolean variable and $2k+1$ integer variables in $[0,a]$ should be produced. The authors assume that

f undergos no more than k changes for each period of length a, but do not tell us how to determine the value of k. However, we know that the value of k is at most $a-1$. Therefore, to construct the tester of $f\mathtt{U}_{[a,b]}g$, that method requires the creation of three boolean variables, one integer variable in $[0, b-a]$, and $2a+2$ integer variables in $[0, a]$. Meanwhile, for $\psi = f\mathtt{U}_{[a,b]}g$ $(a < b)$ our tester T_ψ only needs to create $a+1$ fresh boolean variables and one integer variable in $[0, b-a-1]$.

(2) For $f\mathtt{U}_{[0,b]}g$, our tester is different from that in [22], because we use the standard semantics of $f\mathtt{U}_{[0,b]}g$, which does not restrict f to hold on the state where g holds. In [22], this restriction is added to the semantics.

(3) Besides the (bounded) future temporal formulae, we construct the testers for $\mathtt{Y}$, $f\mathtt{S}g$, and $f\mathtt{S}_{[0,b]}g$, so that our method also fully supports (bounded) past temporal formulae. Meanwhile, Ref. [22] only presents the tester for $\boxminus_{[0,a]}f$ (historically), which is not adequate to express full past temporal formulae. Moreover, the tester for $\boxminus_{[0,a]}f$ is only a special case of our tester for $f\mathtt{S}_{[0,b]}g$, because the former can be constructed as our tester for $\neg(\top\mathtt{S}_{[0,b]}\neg f)$.

As far as we know, to date there does not exist any state-of-the-art temporal model checker supporting complete RTLTL, let alone RTCTL*. The latest version 2.6.0 of NuSMV supports two real-time temporal logics, RTCTL and an extension of LTL, both of which are subsets of RTCTL*. We optimize our method for RTCTL* by applying the RTCTL model checking method, instead of constructing the tester, when checking a principally temporal subformula that is immediately preceded by a path quantifier. Therefore, our method can achieve the same performance as NuSMV when checking RTCTL formulae. The extension of LTL in NuSMV cannot directly support the operators $\mathtt{U}_{[a,b]}$ and $\mathtt{S}_{[a,b]}$, and the state space of the translation-based method used by NuSMV will be exponentially larger than that of our tester-based method for these operators with $a = 0$. This result is justified by the experimental results presented in Section 5.

On the other hand, as shown in Section 1, the existing real-time temporal logics adopted by most model checkers for real-time systems (including HyTech, Uppaal, Kronos, and FSMT-MC) are based on CTL or LTL, and thus will be semantically weaker than RTCTL* for modeling complex scenarios. Recently, substantial research effort has been dedicated to advancing the frontiers of traditional temporal logics [23–26]. Meanwhile, the verification of hybrid systems with discrete and continuous transitions has also been intensively studied, such as in [27, 28]. This paper combines these two strands of research, and has addressed this issue by providing a new modeling language that features flexibility and algorithmic manageability.

7 Conclusion

In this paper, we presented a new real-time temporal logic RTCTL*, as an extension of CTL* with (bounded) future and past temporal operators. By constructing the testers for all temporal operators, we proposed a tester-based symbolic model checking method for RTCTL*. We have already implemented an efficient model checking prototype for real-time linear temporal logic RTLTL, which is a subset of RTCTL* without path quantifiers, by building upon NuSMV. The soundness and completeness of the proposed method, the expressiveness of RTCTL*, and the complexity of the tester construction have been described and proven. Theoretical and experimental results for the prototype both show that for checking bounded temporal formulae of the form $f\mathtt{U}_{[0,b]}g$ or $f\mathtt{S}_{[0,b]}g$, our method performs exponentially better than the fully translation-based one.

For future work, we plan to apply the proposed reduction-based method for RTLTL to other existing model checkers, such as SPIN, MCMAS, and MCTK. To support full RTCTL* model checking, we also plan to directly implement the proposed method for RTCTL* based on MCTK, which is an efficient symbolic model checker for multi-agent systems that we have developed. Furthermore, generating counterexamples is another important issue that siginificantly aids users in debugging verified systems. We will provide a symbolic algorithm to generate tree-like counterexamples/witnesses for RTCTL*, and integrate this into MCTK. Another important application of testers is deductive verification. Based on the compositionality of the proposed testers, it is possible to establish a deductive proof system for RTCTL*,

such that the proof of an RTCTL* formula φ with respect to a JDS $\mathcal{D}$ can be reduced to the proof of $\chi(\varphi)$ with respect to $\mathcal{D} \parallel T_\varphi$. Such a deductive proof system for RTCTL* would be valuable for verifying more expressive real-time temporal properties over infinite-state real-time systems.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61170028, 61572234, 61370072, 71571056), Young Scientists Fund of the National Natural Science Foundation of China (Grant No. 61502184), Program for New Century Excellent Talents in Fujian Province Universities (Grant No. 2013FJ-NCET-ZR03), Natural Foundation Key Program for Young Scholars in the Universities of Fujian Province (Grant No. JZ160409), Natural Science Foundation of Fujian Province (Grant No. 2015J01255), Promotion Program for Young and Middle-aged Teacher in Science and Technology Research of Huaqiao University (Grant No. ZQN-YX109), and Guangxi Key Laboratory of Trusted Software (Grant No. kx201323).

References

1 Clarke E M, Grumberg O, Peled D A. Model Checking. London: The MIT Press, 2000
2 Goranko V, Galton A. Temporal logic. In: The Stanford Encyclopedia of Philosophy. San Francisco: Metaphysics Research Lab, Stanford University, 2015
3 Holzmann G J. The SPIN Model Checker-Primer and Reference Manual. Boston: Addison-Wesley, 2004
4 Cimatti A, Clarke E M, Giunchiglia E, et al. Nusmv 2: an opensource tool for symbolic model checking. In: Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002). Berlin: Springer, 2002. 359–364
5 McMillan K L. Symbolic Model Checking. Norwell: Kluwer Academic Publisher, 1993
6 Pnueli A, Sa'ar Y, Zuck L D. Jtlv: a framework for developing verification algorithms. In: Proceedings of the 22th International Conference on Computer Aided Verification (CAV 2010). Berlin: Springer, 2010. 171–174
7 Su K L, Sattar A, Luo X Y. Model checking temporal logics of knowledge via OBDDs. Comput J, 2007, 50: 403–420
8 Larsen K G, Pettersson P, Wang Y. UPPAAL in a nutshell. Int J Softw Tools Tech Transfer, 1997, 1: 134–152
9 Henzinger T A, Ho P-H, Howard W-T. HYTECH: a model checker for hybrid systems. Int J Softw Tools Tech Transfer, 1997, 1: 110–122
10 Bozga M, Daws C, Maler O, et al. Kronos: a model-checking tool for real-time systems. In: Proceedings of the 10th International Conference on Computer Aided Verification (CAV 1998). London: Springer-Verlag, 1998. 546–550
11 Georges M, Christoph S. Fully symbolic TCTL model checking for complete and incomplete real-time systems. Sci Comput Program, 2015, 111: 248–276
12 Alur R, Courcoubetis C, Dill D. Model-checking in dense real-time. Inf Comput, 1993, 104: 2–34
13 Alur R, Henzinger T A. Real-time logics: complexity and expressiveness. Inf Comput, 1993, 104: 35–77
14 Alur R, Henzinger T A. A really temporal logic. J ACM, 1994, 41: 181–204
15 Alur R, Feder T, Henzinger T A. The benefits of relaxing punctuality. J ACM, 1996, 43: 116–146
16 Aceto L, Laroussinie F. Is your model checker on time? On the complexity of model checking for timed modal logics. J Log Algebr Program, 2002, 52: 7–51
17 Bouyer P, Fahrenberg U, Larsen K G, et al. Model checking real-time systems. In: Handbook of Model Checking. Berlin: Springer-Verlag, 2017
18 Lomuscio A, Qu H Y, Raimondi F. MCMAS: an open-source model checker for the verification of multi-agent systems. Int J Softw Tools Tech Transfer, 2017, 19: 9–30
19 Bryant R E. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv, 1992, 24: 293–318
20 Emerson E A, Mok A K, Sistla A P, et al. Quantitative temporal reasoning. Real-Time Syst, 1992, 4: 331–352
21 Fruth M. Formal verification of embedded real-time systems. Dissertation for Ph.D. Degree. Dresden: TU Dresden, 2005
22 Pnueli A, Zaks A. On the merits of temporal testers. In: 25 Years of Model Checking: History, Achievements, Perspectives. Berlin: Springer, 2008. 172–195
23 Finkbeiner B, Rabe M N, Sánchez C. Algorithms for model checking HyperLTL and HyperCTL*. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV 2015). Berlin: Springer, 2015. 30–48
24 Cimatti A, Griggio A, Mover S, et al. Verifying LTL properties of hybrid systems with k-liveness. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV 2014). Berlin: Springer, 2014. 424–440
25 Cook B, Khlaaf H, Piterman N. On automation of CTL* verification for infinite-state systems. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV 2015). Berlin: Springer, 2015. 13–29.
26 Zhang N, Duan Z H, Tian C. Model checking concurrent systems with MSVL. Sci China Inf Sci, 2016, 59: 118101
27 Immler F. Verified reachability analysis of continuous systems. In: Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015). Berlin: Springer, 2015. 37–51
28 Lin W, Wu M, Yang Z F, et al. Exact safety verification of hybrid systems using sums-of-squares representation. Sci China Inf Sci, 2014, 57: 052101