Bounded Model Checking for Propositional Projection Temporal Logic ⋆

Zhenhua Duan[1], Cong Tian[1,⋆⋆] Mengfei Yang[2], and Jia He[1]

[1] ICTT and ISN Lab, Xidian University, Xi'an, 710071, P.R. China
[2] China Academy of Space Technology, Beijing, 100094, P.R. China

Abstract. This paper presents a bounded model checking approach for propositional projection temporal logic (PPTL). To this end, first PPTL is briefly introduced. Then, bounded semantics of PPTL is defined according to its semantics in logic theory. Further, a reduction method from BMC to SAT is given in detail. In addition, an example is presented to illustrate how the approach works. Our experience shows that BMC approach for PPTL proposed in the paper is useful and feasible.

1 Introduction

Model checking was firstly proposed by Clarke and Emerson [7] as well as Quielle and Sifakis [11] independently. As a trusted, strong and automatic verification technique, model checking has been widely used in many fields such as verification of hardware, software and communication protocols. Model checking is a procedure used to check whether a property specified by a temporal logic formula ϕ is satisfied by a given system M defined in a transition structure ($M \models \phi$?). With temporal logics, linear-time temporal logic (LTL)[2] and branching-time temporal logic (CTL)[7] are popular in practice. In particular, they are usually used as description logics to specify properties of systems in model checking. SPIN [9] based on LTL and SMV [13] depended on CTL are two well-known model checkers. However, for a given system, the number of states of the system increases exponentially with the increasing of the number of components in the system, leading to state explosion problem. To combat this problem, many optimization methods are put forward. Symbolic model checking [12,13] aims at reduction in memory needed and high efficiency in model checking. Compositional model checking [17] optimizes model checking by dividing the model under consideration into several parts and conducts the verification of each part respectively to reduce the complexity of the checking process. Bounded model checking (BMC) is an important progress in formalized verification after symbolic model checking [3]. It proves to be an efficient and successful approach. The main idea of BMC is to check whether a property is satisfied by a system with limitation that the searching length is bounded by a given integer k. If the property is not satisfied, an error is found. Otherwise, we cannot tell whether the

⋆ This research is supported by the NSFC Grant No. 61003078, 61133001, and 61272117, 973 Program Grant No. 2010CB328102 and ISN Lab Grant No. ISN1102001.
⋆⋆ Corresponding author.

D.-Z. Du and G. Zhang (Eds.): COCOON 2013, LNCS 7936, pp. 591–602, 2013.
© Springer-Verlag Berlin Heidelberg 2013

system satisfies the property or not. In this case, we can consider increasing k, and perform the process of BMC again. In the procedure, BMC is reduced into a SAT problem to find a solution. Although SAT is proved to be an NP-complete problem, however, SAT solver works well in many practical applications with the DPLL algorithm[14]. BMC applies temporal logics as its underlying logics to specify properties of systems as the basic model checking does. Theories and supporting tool NuSMV2 [1] based on LTL is now available. Other tools that can conducts BMC are Bounded Model Checker of CMU [10], Thunder of Intel [8] and so on.

With model checking and bounded model checking, the mostly used temporal logics are LTL, CTL and their variations. An LTL formula is assigned along the path. If all paths from a given state s satisfies an LTL formula f, we say s satisfies f. Thus, LTL implicitly limits all the paths with a universal quantifier. Therefore, assertions that whether or not there exists a path satisfied the property cannot be specified with LTL. Branching-time temporal logic solved this problem by allowing the use of path quantifier explicitly. From this point, we could say CTL is more expressive than LTL. However, CTL cannot choose a range of paths by means of a formula, which indicates LTL is more expressive than CTL. In addition, there exist two limitations in the grammar of CTL. Boolean combination of path formulas is not allowed in CTL as well as the nestification of path modal operators X, F and G. CTL* is a logic which is of the expressiveness of LTL and CTL. Since LTL lacks the path quantifiers while CTL lacks the ability of LTL to specify a single path elaborately [15], CTL* is more expressive than both of the logics. However, the cost of computation for CTL* is quite high.

Since the expressiveness of LTL and CTL is not powerful enough, actually, not full regular, therefore, there are at least two types of properties in practice which cannot be specified by LTL and CTL: (1) some time duration related properties such as property P holds at 100^{th} state or P holds after 100^{th} time unites and before 200^{th} time unites; (2) some repeated properties (kleen closure)P. Propositional projection temporal logic (PPTL)[21] is an extension of interval temporal logic (ITL)[16]. The expressiveness of PPTL is full regular[18] which allows us to verify full regular properties and time duration related properties of systems in a convenient way. For instance, the above mentioned time duration properties can be defined as $len100; P$ and $len100; (\Diamond P \wedge len100)$ respectively while the closure property can be defined by P^*.

With the community of projection temporal logic (PTL), plenty of logic laws [24] have been formalized and proved, and a decision procedure for checking satisfiability of PPTL formulas has also been given [22,23]. Further, a model checker based on SPIN for PPTL has been developed [5]. Therefore, full regular properties and time duration properties of systems can be automatically verified with SPIN. However, since the state explosion problem is a common problem for model checking, the model checker based on SPIN for PPTL also suffers from this problem. To combat it, we are motivated to investigate bounded model checking for PPTL in this paper. To do so, the bounded semantics of PPTL is defined, and some lemmas and theorems for building relationship between bounded model checking and the basic model checking are proved. Based on these basic theories, BMC for PPTL is reduced into SAT problem. To illustrate how our approach can be used to verify full regular properties of systems, an example for verifying feasibility of rate monotonic scheduling (RMS) algorithm [4] is presented

in details. To realize our bounded model checking approach, a prototype of bounded model checker based on NuSMV2 has also been developed recently.

The rest of the paper is organized as follows. PPTL is briefly introduced in section 2. In section 3, bounded model checking for PPTL is formalized in detail. Further, an example for verifying feasibility of RM scheduling algorithm is demonstrated. Some related work are reviewed in section 4. Finally, the conclusion is drawn in section 5.

2 Propositional Projection Temporal Logic

To study bounded model checking of interval based temporal logic (IBTL), we use proposition projection temporal logic (PPTL) as the underlying logic. The syntax and semantics of PPTL is briefly introduced in the following. The details of the logic can refer to [22].

2.1 Syntax

Let $Prop$ be a countable set of atomic propositions. PPTL formulas can be inductively defined as below:

1) Every atomic proposition $p \in Prop$ is a formula;

2) If $P, P_1, \cdots, P_m$ are PPTL formulas, so are the following constructs: $\neg P$, $P_1 \vee P_2$, $\bigcirc P$, and $(P_1, \cdots, P_m)\ prj\ P$.

A formula without temporal operators ($\bigcirc$, prj) is called a state formula, otherwise it is a temporal formula.

2.2 Semantics

States: Following the definition of Kripke's structure, we define a state s over $Prop$ as a mapping from $Prop$ to $B = \{true, false\}$, $s : Prop \rightarrow B$. We use $s[p]$ to represent the valuation of p at state s.

Intervals: An interval σ is a non-empty sequence of states, which can be finite or infinite. The length of σ, $|\sigma|$, is ω if σ is infinite, and the number of states minus 1 if σ is finite. To have a uniform notation for both finite and infinite intervals, we will use extended integers as indices. That is, we consider the set N_0 of non-negative integers and ω, $N_\omega = N_0 \cup \{\omega\}$, and extend the comparison operators, $=, <, \leq$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0, i < \omega$. Moreover, we define $\preceq$ as $\preceq\ -(\omega, \omega)$. To simplify definitions, we will denote σ by $< s_0, \cdots, s_{|\sigma|} >$, where $s_{|\sigma|}$ is undefined if σ is infinite. With such a notation, $\sigma_{(i..j)}(0 \leq i \leq j \leq |\sigma|)$ denotes the sub-interval $< s_i, \cdots, s_j >$ and $\sigma^{(k)}(0 \leq k \leq |\sigma|)$ denotes $< s_k, \cdots, s_{|\sigma|>}$. The concatenation of a finite σ with another interval (or empty string) σ' is denoted by $\sigma \cdot \sigma'$.

Let $\sigma =< s_0, \cdots, s_{|\sigma|} >$ be an interval and $r_1, \cdots, r_h$ be integers ($h \geq 1$) such that $0 \leq r_1 \leq r_2 \leq ... \leq r_h \leq |\sigma|$. The projection of σ onto $r_1, \cdots, r_h$ is the interval (namely projected interval) $\sigma \downarrow (r_1, \cdots, r_h) =< s_{t_1}, s_{t_2}, \cdots, s_{t_l} >$, where $t_1, \cdots, t_l$ is obtained from $r_1, \cdots, r_h$ by deleting all duplicates. That is, $t_1, \cdots, t_l$ is the longest strictly increasing subsequence of $r_1, \cdots, r_h$. For instance, $< s_0, s_1, s_2, s_3, s_4 >\downarrow (0, 0, 2, 2, 2, 3) =< s_0, s_2, s_3 >$.

Interpretations: An interpretation is a triple $\mathcal{I} = (\sigma, i, j)$, where σ is an interval, i is an integer, and j an integer or ω such that $i \leq j \leq |\sigma|$. We use the notation $(\sigma, i, j) \models P$ to denote that formula P is interpreted and satisfied over the subinterval $< s_i, \cdots, s_j >$ of σ with the current state being s_i. The satisfaction relation ($\models$) is inductively defined as follows:

$I - prop$ $\mathcal{I} \models p$ iff $s_i[p] = true$, and $p \in Prop$ is an proposition
$I - not$ $\mathcal{I} \models \neg P$ iff $\mathcal{I} \not\models P$
$I - or$ $\mathcal{I} \models P \vee Q$ iff $\mathcal{I} \models P$ or $\mathcal{I} \models Q$
$I - next$ $\mathcal{I} \models \bigcirc P$ iff $i < j$ and $(\sigma, i+1, j) \models P$
$I - prj$ $\mathcal{I} \models (P_1, \cdots, P_m)\ prj\ P$, if there exist integers $r_0 \leq r_1 \leq \cdots \leq r_m \leq j$ such that $(\sigma, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_l) \models P_l, 1 < l \leq m$, and $(\sigma', 0, |\sigma'|) \models Q$ for one of the following σ' :
(*a*) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \cdots, r_m) \cdot \sigma_{(r_m+1..j)}$, or
(*b*) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \cdots, r_h)$ for some $0 \leq h \leq m$.

Satisfaction and Validity: A formula P is satisfied by an interval σ, denoted by $\sigma \models P$, if $(\sigma, 0, |\sigma|) \models P$. A formula P is called satisfiable if $\sigma \models P$ for some σ. A formula P is valid, denoted by $\models P$, if $\sigma \models P$ for all σ.

2.3 Normal Form

Normal Form (NF) [22] is a useful notation in our methods. We assume Q_p is the set of atomic propositions appearing in the PPTL formula Q. Then, the NF can be defined as follows:

Definition 1. *Normal Form of Q:* $NF(Q) \equiv \vee_{j=1}^{n_0}(Q_{ej} \wedge empty) \vee \vee_{i=1}^{n}(Q_{ci} \wedge \bigcirc Q'_i)$.

To simplify the proof and expressiveness, we sometimes use $Q_e \wedge empty$ instead of $\vee_{j=1}^{n_0}(Q_{ej} \wedge empty)$ and apply $\vee_{i=1}^{r}(Q_i \wedge \bigcirc Q'_i)$ to replace $\vee_{i=1}^{n}(Q_{ci} \wedge \bigcirc Q'_i)$. Thus, $NF(Q) \equiv Q_e \wedge empty \vee \vee_{i=1}^{r}(Q_i \wedge \bigcirc Q'_i)$, where Q_e and Q_i are state formulas.

Theorem 1. *Any PPTL formula P can be rewritten into its Normal Form.*

The proof of this theorem can be found in [22].

3 Bounded Model Checking for PPTL

In this section, we investigate the bounded model checking of PPTL based on the theory presented in section 2. To do this, we first establish the basic theory, bounded semantics, of bounded model checking of PPTL. Then we introduce a new approach of bounded model checking for PPTL. Since each PPTL formula can be transformed into an equivalent formula in NF, unlike LTL and CTL formulas [3,19], we do not need to deal with all types of PPTL formulas in the bounded semantics. As a result, we need only consider formulas with $\vee$,$\neg$, and $\bigcirc$ operators as well as *empty* and *more*. In the following, we define the bounded semantics of PPTL formulas. Let σ be an interval and P a PPTL formula.

3.1 Bounded Semantics

In this subsection, we first define Kripke structure, then we define bounded semantics of PPTL formulas.

Definition 2. *A Kripke structure M is a quadruple $M = (S, I, T, L)$ where S is the set of sates, $I \subseteq S$ is the set of initial states, $T \subseteq S \times S$ is the transition relation and $L : S \to P(A)$ is the labeling function, where A is the set of atomic propositions and $P(A)$ denotes the powerset over A.*

We use Kripke structure to model systems, and PPTL formulas to define properties of systems. A path in a Kripke structure can be treated as an interval. Since a path with a Kripke structure can be finite or infinite so an interval can also be finite or infinite. Therefore, in the following, we give the definition of $k - loop$ in an interval.

Definition 3. ***(k-loop)*** *For $l, k \in N_0$ and $l \leq k$, if there is a translation from s_k to s_l in σ and $\sigma = (s_0, \cdots, s_{l-1}) \cdot (s_l, \cdots, s_k)^\omega$, we call interval σ a (k, l)-loop. If there exist k, l, $k \geq l \geq 0$ such that σ is a (k, l)-loop, we call σ a k-loop.*

Obviously, if σ is an infinite interval with loop, it must be a $k - loop$ for some $k \in N_0$. An interval with k-loop is shown in Fig.1. Now we can define the successor of state s_i over an interval σ by $succ(i)$ below:

$$succ(i) = \begin{cases} i+1 & \text{if } i < k \\ l & \text{if } i = k \text{ and } \sigma \text{ is an infinite interval with } (k,l) - loop \\ k+1 & \text{if } i = k \text{ and } \sigma \text{ is a finite interval} \end{cases}$$

To define bounded semantics of PPTL formulas, given a bound $k \in N_0$, we define an interpretation to be a pair $\mathcal{I}' = (\sigma, i)$, where σ is an interval, i an integer and $0 \leq i \leq k \leq |\sigma|$. We use the notation $(\sigma, i) \models_k P$ to represent that formula P is interpreted and satisfied over either the whole interval if σ is a k-loop, or subinterval $< s_i, \cdots, s_k >$ of σ otherwise, with the current state being s_i. In terms of the semantics of PPTL formulas given in section 2, in general, $(\sigma, i) \models_k P$ is not equal to $(\sigma, i, k) \models P$.

The bounded satisfaction relation $\models_k$ is inductively defined as follows:

$$\begin{array}{lll} (\sigma, i) \models_k p & \text{iff} & s_i[p] = true \text{ if } p \in Prop \text{ is an atomic proposition} \\ (\sigma, i) \models_k \neg P & \text{iff} & (\sigma, i) \not\models_k P \\ (\sigma, i) \models_k P_1 \vee P_2 & \text{iff} & (\sigma, i) \models_k P_1 \text{ or } (\sigma, i) \models_k P_2 \\ (\sigma, i) \models_k \bigcirc P & \text{iff} & succ(i) \leq k \text{ and } (\sigma, succ(i)) \models_k P \\ (\sigma, i) \models_k empty & \text{iff} & i = k \\ (\sigma, i) \models_k more & \text{iff} & i < k \end{array}$$

A formula P is satisfied with bound k by an interval σ, denoted by $\sigma \models_k P$, if $(\sigma, 0) \models_k P$. It is not difficult to see the following fact:

Fact 1. *For a PPTL formula P, if σ is finite, $(\sigma, i) \models_k P \Longleftrightarrow (\sigma, i, k) \models P$*

As any PPTL formula P can be transformed into its normal form, the above definition about satisfaction relation in bounded semantics of PPTL is sound.

3.2 Bounded Model Checking

According to the structure of interval σ, two basic lemmas are given below:

Lemma 1. *Let $k \in N_0$, f be a PPTL formula and σ a $k-loop$. Then $\sigma \models_k f$ iff $\sigma \models f$.*

Fig. 1. σ is a $k-loop$

Note that we do not need to deal with *empty* and *more* in the above inductive cases since $empty \equiv \neg \bigcirc true$ and $more \equiv \bigcirc true$.

Lemma 2. *Let f be a PPTL formula, and σ a finite interval. Then $\sigma \models f \Rightarrow \exists k, k \geq 0, \sigma \models_k f$.*

Theorem 2. *Suppose a system is described by a Kripke structure* **M***, and some property is defined by a PPTL formula f. let M be the set of all intervals generated in* **M***. We define $M \models \exists f$ iff $\exists \sigma \in M$ and $\sigma \models f$ as well as $M \models_k \exists f$ iff $\exists \sigma \in M$ and $\sigma \models_k f$. Then we have $M \models \exists f$ iff $\exists k, k \geq 0, M \models_k \exists f$.*

Proof: We need prove that $\exists \sigma \in M$ *and* $\sigma \models f$ iff $\exists \sigma \in M$ *and* $\sigma \models_k f$. To do so, we consider intervals in M as infinite or finite ones respectively.

1. Suppose σ is infinite, there must be an integer $k \geq 0$ that makes σ a $k-loop$. By Lemma 1: $\sigma \models_k f$ iff $\sigma \models f$. We have, $M \models \exists f$ iff $\exists k, k \geq 0, M \models_k \exists f$.
2. Suppose interval σ is finite, i.e., $|\sigma| = m \in N_0$.

   " $\Rightarrow$ " : $M \models \exists f \Leftrightarrow \exists \sigma \in M \wedge \sigma \models f$
   $\Rightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge \sigma \models_k f$
   $\Leftrightarrow \exists k, k \geq 0, \Sigma \models_k \exists f$ (Lemma 2)

   " $\Leftarrow$ " : $\exists k, k \geq 0, M \models_k \exists f$
   $\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge \sigma \models_k f$
   $\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge (\sigma, 0) \models_k f$
   $\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge (\sigma, 0, k) \models f$ ($Fact$ 1)

   Let $\sigma' =< s_0, s_1, \cdots, s_k >$, we have $(\sigma', 0, k) \models f$. Since σ' is a subinterval of σ and $|\sigma'| = k$, hence, $\exists \sigma' \in M$ and $\sigma' \models f$, leading to $M \models \exists f$. As a result, $\exists k, k \geq 0$, $M \models_k \exists f \Rightarrow M \models \exists f$. □

3.3 Reducing BMC of PPTL to SAT

Given a finite state transition system M (a Kripke structure or NFG), the property of the system in terms of a PPTL formula f, and a bound k, then the procedure of BMC can be described as a process for constructing a proposition formula $[M, f]_k$. Let $(s_0, \cdots, s_k)$ be a subinterval of interval σ. The definition of formula $[M, f]_k$ consists

of two components: M_k, the first component, is a propositional formula that constrains $(s_0, \cdots, s_k)$ to be a valid interval starting from an initial state; X_k, the second component, a propositional formula that constrains σ to satisfy f with bound k. To define the second component, we first give the definition of *loop condition*, which is a propositional formula that is evaluated to true only if the interval σ contains a loop; then we give some translation rules for formulas.

Definition 4. *For a Kripke structure M, and k≥0,* $M_k \stackrel{\text{def}}{=} I(s_0) \wedge \wedge_{i=0}^{k-1} T(s_i, s_{i+1})$.

Loop condition: $L_{(k,l)} \stackrel{\text{def}}{=} T(s_k, s_l)\ \ (0 \le l \le k)$ and $L_k \stackrel{\text{def}}{=} \vee_{l=0}^{k} L_{(k,l)}$.

Let p be an atomic proposition, and f, g be PPTL formulas, σ an interval with a $(k,l) - loop$, s_i the current state, and k the bound, where $0 \le i, l, k \le |\sigma|,\ i, l \le k$. The translation of a PPTL formula is given below:

k-Loop: Translation $f(i,k,l)$ of a PPTL formula f over an infinite interval σ with $(k,l)-loop$ is inductively defined as follows.

$$p_{(i,k,l)} \stackrel{\text{def}}{=} p(s_i) \text{ if } p \in Prop \text{ is an atomic proposition, meaning } s_i[p]$$

$$(\neg f)_{(i,k,l)} \stackrel{\text{def}}{=} \neg f_{(i,k,l)} \qquad (f \vee g)_{(i,k,l)} \stackrel{\text{def}}{=} f_{(i,k,l)} \vee g_{(i,k,l)} \qquad (\bigcirc f)_{(i,k,l)} \stackrel{\text{def}}{=} f_{(succ(i),k,l)}$$

$$empty_{(i,k,l)} \stackrel{\text{def}}{=} \begin{cases} false & i < k \\ true & i = k \end{cases} \qquad more_{(i,k,l)} \stackrel{\text{def}}{=} \begin{cases} true & i < k \\ false & i = k \end{cases}$$

For the translation presented above, a new propositional variable is introduced for each intermediate formula $e_{(i,k,l)}$, where e is a sub-formula of PPTL formula f and $0 \le i \le k$ and l indicates the existence of $(k,l) - loop$. If e is an atomic proposition or its negation, $e_{(i,k,l)}$ is substituted by a boolean variable with the truth-value at state s_i. Otherwise, $e_{(i,k,l)}$ indicates a propositional formula whose satisfiability should be considered over the interval $(s_i, \cdots, s_k)$.

No Loop: Translation $f(i,k)$ of a PPTL formula f for a finite interval σ (with no loop).

$$p_{(i,k)} \stackrel{\text{def}}{=} p(s_i) \qquad (\neg f)_{(i,k)} \stackrel{\text{def}}{=} \neg f_{(i,k)} \qquad (f \vee g)_{(i,k)} \stackrel{\text{def}}{=} f_{(i,k)} \vee g_{(i,k)}$$

$$(\bigcirc f)_{(i,k)} \stackrel{\text{def}}{=} f_{(i+1,k)} \qquad f_{(k+1,k)} \stackrel{\text{def}}{=} false$$

$$empty_{(i,k)} \stackrel{\text{def}}{=} \begin{cases} false & i < k \\ true & i = k \end{cases} \qquad more_{(i,k)} \stackrel{\text{def}}{=} \begin{cases} true & i < k \\ false & i = k \end{cases}$$

When the intermediate formula $e_{(i,k)}$ appears in a PPTL formula for a finite interval (with no loop), the explanations are similar to the ones for a PPTL formula with an interval with $(k,l) - loop$ and omitted here.

General translation (BMC to SAT):

$X_k \stackrel{\text{def}}{=} (\neg L_k \wedge f_{(0,k)}) \vee \vee_{l=0}^{k}(L_{(k,l)} \wedge f_{(0,k,l)})$ and $[M,f]_k \stackrel{\text{def}}{=} M_k \wedge X_k$, i.e.,

$$[M,f]_k \stackrel{\text{def}}{=} I(s_0) \wedge \wedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge [(\neg L_k \wedge f_{(0,k)}) \vee \vee_{l=0}^{k}(L_{(k,l)} \wedge f_{(0,k,l)})]$$

As we can see, the right side of the definition can be divided into two parts: the first part $I(s_0) \wedge \wedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge (\neg L_k \wedge f_{(0,k)})$ indicates an interval with no loop and the translation without loop is used; the second part $I(s_0) \wedge \wedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge (\vee_{l=0}^{k}(L_{(k,l)} \wedge f_{(0,k,l)}))$ presents that an interval with a $k - loop$ and all possible starting points l of a loop and the translation for a $k - loop$ is conjoined with the corresponding loop condition $L_{(k,l)}$.

Lemma 3. *Let $k \in N_0$, f be a PPTL formula and σ an interval. The translation $f_{(i,k,l)}$ or $f_{(i,k)}$ is defined as above. Then*
(1) $\sigma \models_k f$ iff $\sigma \models_k f_{(i,k,l)}$ if σ is infinite with a $(k,l)-loop$ or
(2) $\sigma \models_k f$ iff $\sigma \models_k f_{(i,k)}$ if σ is finite.

Now we come to prove an important conclusion formalized in Theorem 3.

Theorem 3. *$[M,f]_k$ is satisfiable iff $M \models_k \exists f$.*

Proof: " $\Rightarrow$ ": Suppose $[M,f]_k$ is satisfiable. That is, $\exists k \geq 0, \sigma \in M, \sigma \models [M,f]_k \Leftrightarrow \sigma \models M_k \wedge [(\neg L_k \wedge f_{(0,k)}) \vee \vee_{l=0}^{k}(L_{(k,l)} \wedge f_{(0,k,l)})]$.

If $|\sigma| = \omega$ with (k,l)-loop and $l \leq k$, we have,

$$\begin{aligned}
\sigma \models [M,f]_k &\Rightarrow \vee_{l=0}^{k}(L_{(k,l)} \wedge f_{(0,k,l)}) && def\ of\ [M,f]_k \\
&\Rightarrow \sigma \models \exists j\, 0 \leq j \leq k\ f_{(0,k,j)} && \\
&\Rightarrow \sigma \models f_{(0,k,j)}\ (0 \leq j \leq k) && \\
&\Rightarrow \sigma \models_k f_{(0,k,j)}\ (0 \leq j \leq k) && Lem1 \\
&\Rightarrow \sigma \models_k f && Lem3 \\
&\Rightarrow \exists k, k \geq 0, \sigma \in M\ \sigma \models_k f && \\
&\Rightarrow M \models_k \exists f && def
\end{aligned}$$

If $|\sigma| \in N_0$ with no loop, we have,

$$\begin{aligned}
\sigma \models [M,f]_k &\Rightarrow \sigma \models f_{(0,k)} && def\ of\ [M,f]_k \\
&\Rightarrow \exists k, k \geq 0, \sigma \models_k f_{(0,k)} && Lem1 \\
&\Rightarrow \exists k, k \geq 0, \sigma \in M\ \sigma \models_k f && Lem3 \\
&\Rightarrow M \models_k \exists f && def
\end{aligned}$$

"$\Leftarrow$": Suppose $M \models_k \exists f$. Since $M \models_k \exists f \Leftrightarrow \exists \sigma \in M, \sigma \models_k f$, so $k \geq 0$ is the bound, and there exists subinterval $< s_0, ..., s_k >$ of $\sigma \in M$ such that $\sigma \models_k M_k \equiv I(s_0) \wedge \wedge_{i=0}^{k-1} T(s_i, s_{i+1})$.

If $|\sigma| = \omega$ with (k,l)-loop, there is a $l, 0 \leq l \leq k$ and $T(s_k, s_l)$, so $\sigma \models_k T(s_k, s_l)$.

$$\begin{aligned}
\sigma \models_k f &\Rightarrow \sigma \models_k f_{(0,k,l)}) && Lem3 \\
&\Rightarrow \sigma \models_k M_k \wedge L(k,l) \wedge f_{(0,k,l)} && \\
&\Rightarrow \sigma \models_k [M,f]_k && def\ of\ [M,f]_k
\end{aligned}$$

If $|\sigma| = \omega$ without loop, $\sigma \models_k \neg L_k$, leading to $\sigma \models_k M_k \wedge \neg L_k$. Thus, we have,

$$\begin{aligned}
\sigma \models_k f &\Rightarrow \sigma \models_k f_{(0,k)}) && Lem3 \\
&\Rightarrow \sigma \models_k M_k \wedge \neg L_k \wedge f_{(0,k)} && \\
&\Rightarrow \sigma \models_k [M,f]_k && def\ of\ [M,f]_k
\end{aligned}$$

□

3.4 An Example: BMC for RMS

RMS (rate monotonic scheduling) is a classical algorithm for periodic task scheduling proposed by Liu and Layland in 1973. As one of the static priority-driven scheduling

approaches, RMS is optimal. Let $T = \{t_1, t_2, \cdots, t_n\}$ be a set composed of n periodic tasks. A task is defined as a quadruple $t_i = (T_i, D_i, E_i, P_i)$. T_i is the period of t_i, D_i is the deadline of t_i, E_i denotes the execution time of t_i and P_i indicates priority level of t_i. In general, $D_i = T_i$ and $P_i > P_j$ if $T_i < T_j$. Thus, the definition can be simplified as a pair $t_i = (T_i, E_i)$. Since the utilization factor of CPU is defined as $U_i = E_i/T_i$, the utilization factor of CPU for task set T can be defined as $U = \sum_{i=1}^{n} U_i$. It has been proved by Liu and Layland that for any task set T, if $\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1)$, T is schedulable with RMS algorithm. However, this is only a unnecessary and sufficient condition.

If a task set T is schedulable with RMS algorithm, any $t_i \in T$ must be real-time. This indicates that the remaining execution time of t_i is 0 or 1 if t_i is being executed when a new period of t_i is coming in the next moment, which means the execution time of t_i in its every period must equals E_i. To explain this property in detail, we give the following example. Given a task set $T = \{t_1, t_2\}$, where $t_1 = (4, 1)$, $t_2 = (3, 1)$. Since $1/4 + 1/3 < 2 \times (\sqrt{2} - 1)$, so T is schedulable with RMS algorithm. Since $T_1 > T_2$, we have $P_1 < P_2$. The execution of T with RMS algorithm is as shown below in figure 2.

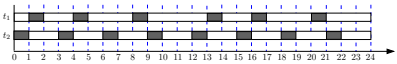

Fig. 2. Execution of T with RMS algorithm

In figure 2, we just simulate 24 time units. From this figure, we can see the task T is schedulable and the real-time property is satisfied. Alternatively, we alter the task set as $T_f = \{t_1, t_2\}$ where $t_1 = (4, 2)$, $t_2 = (3, 2)$. In this case, t_2 is still prior to t_1 . The execution of T_f with RMS algorithm is shown in figure 3.

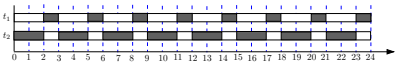

Fig. 3. Execution of T_f with RMS algorithm

Since $2/4 + 2/3 > 2 \times (\sqrt{2} - 1)$, we cannot decide whether T is schedulable with RMS algorithm. As an application of BMC for PPTL, we will check T_f's schedulability with RMS algorithm.To do this, first RMS is described by a Kripke structure M, and the property to be verifies as a PPTL formula P; then an integer $k > 0$ is chosen as the bound; after that, the model checking problem is translated into SAT problem using the approach as described before; finally, a process checking the SAT problem is employed to find a counterexample or to confirm the property is satisfied.

Modeling of RMS: To check the schedulability of T_f using RMS algorithm by means of bounded model checking of PPTL, we first make a model for figure 3 as shown in figure 4, where p indicates in a period of t_1, the execution time equals E_1 and q denotes

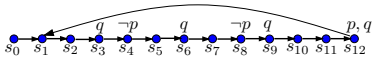

Fig. 4. Kripke structure of T_f with RMS algorithm:M

in a period of t_2, the execution time equals E_2. In Kripke structure M shown in Figure 4, $S = \{s_0, s_1, \cdots, s_{12}\}$, $I = \{s_0\}$ and $T = \{(s_0, s_1), (s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_5), (s_5, s_6), (s_6, s_7), (s_7, s_8), (s_8, s_9), (s_9, s_{10}), (s_{10}, s_{11}), (s_{11}, s_{12}), (s_{12}, s_1)\}$. The atomic proposition set is $A = p, q, x, y$, and labeling function can be defined as: $L(s_3) = \{q\}, L(s_6) = \{q\}, L(s_9) = \{q\}$ and $L(s_{12}) = \{p, q\}$. Note that a state without specified propositions implied that the propositions at the state are negative. For instance, s_4 is unspecified, so $L(s_4) = \phi$, implying $\{\neg p, , \neg q\}$ at state s_4.To encode the system, we have to add 2 additional atomic propositions x and y into the set of atomic proposition, since there are 13 states in M and $2^3 < |S| = 13 < 2^4$. We do not need to specify values for x and y in this example since we do not use them. We can set a fixed order of atomic propositions as (p, q, x, y) , then we can represent every state $s \in S$ with the boolean vector (p, q, x, y).

Definition of Property: The property we concern can be defined as follows:

$$p \text{ is true at state } s_{n*T_1} \text{ and } q \text{ is true at state } s_{n*T_2}, 1 \leq n \in N.$$

This is a typical example for showing the limitation of the expressive power of LTL because this property cannot be described by an LTL formula since the property is full regular. Certainly, we can specify this property by a PPTL formula as follows:

$$F \equiv (\bigcirc^{T_1}(p \wedge \varepsilon))^+ \wedge (\bigcirc^{T_2}(q \wedge \varepsilon))^+$$

Let $T_1 = 4$ and $T_2 = 3$, we have, $F \equiv (\bigcirc^4(p \wedge \varepsilon))^+ \wedge (\bigcirc^3(q \wedge \varepsilon))^+$. By means of normal form of PPTL formulas, we can work out $NF(F)$: $NF(F) \equiv \bigcirc(\bigcirc^3(p \wedge \varepsilon); (\bigcirc^4(p \wedge \varepsilon))^*) \wedge \bigcirc(\bigcirc^2(q \wedge \varepsilon); (\bigcirc^3(q \wedge \varepsilon))^*)$ and $NF(\neg F) \equiv \varepsilon \vee \bigcirc\neg(\bigcirc^3(p \wedge \varepsilon); (\bigcirc^4(p \wedge \varepsilon))^*) \vee \bigcirc\neg(\bigcirc^2(q \wedge \varepsilon); (\bigcirc^3(q \wedge \varepsilon))^*)$

Translation of BMC to SAT: With BMC for PPTL, we are trying to find a counterexample of the property F, which means we are looking for a witness for $\neg F$. The existence of such a witness indicates that property F is false. On the other hand, it means the property holds up to the given bound. Assume the bound $k = 4$. Unfolding the transition relation, we can get the following formula:

$$M_4 \equiv I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3) \wedge T(s_3, s_4)$$

According to Figure 4, the loop condition $L_4 \equiv \vee_{l=0}^4 L_{(4,l)} \equiv \vee_{l=0}^4 T(s_4, s_l)$ is false. We can get the translation for paths with no loops as following:

$$\neg F \equiv [NF(\neg F)]_{(0,4)} \equiv \neg p(s_4) \vee \neg q(s_3)$$

Putting everything together we can get the following boolean formula:

$$\begin{aligned}[M, \neg F]_{(0,4)} &\equiv M_4 \wedge [(\neg L_4 \wedge (\neg F)_{(0,4)}) \vee \vee_{l=0}^4 (L_{(4,l)} \wedge (\neg F)_{(0,4,l)})] \\ &\equiv M_4 \wedge [(true \wedge (\neg F)_{(0,4)}) \vee false] \\ &\equiv M_4 \wedge (\neg F)_{(0,4)}\end{aligned}$$

$$\equiv I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3) \wedge T(s_3, s_4) \wedge (\neg p(s_4) \vee \neg q(s_3))$$

Finding a Witness for the SAT Problem: As shown in figure 4, the assignment, $p(s_4) = 0$ satisfies $[M, \neg F]_{(0,4)}$. This assignment corresponds to an interval from the initial state to s_4 that violates the property.

With bounded model checking for PPTL, we proved T_f is non-schedulable with RMS algorithm.

4 Related work

Bounded model checking for linear-time and branching-time temporal logics, have been studied in recent years [3,19]. As typical representatives of two kinds of temporal logics, LTL and CTL are often used as the logics to specify properties in the bounded model checking. A useful supporting tool NuSMV2 [1] is based on LTL. Other BMC supporting tools including Bounded Model Checker of CMU [10], Thunder of Intel [8] and so on are also available. Further, researchers also studied the completeness and complexity of bounded model checking for LTL and CTL, which are considered as computational challenges in bounded model checking [6]. Computing completeness threshold ($C\mathcal{T}$) may be an efficient way but it only works for some simple, non-nested properties [3]. Other methods are checking for convergence which only works for properties that can be reduced to an invariant one and fixpoint detection. All the above methods have limitations since the expressiveness of LTL or CTL is not full regular. Compared with linear-time and branching-time temporal logics, however, interval based temporal logic such as ITL or PPTL is more powerful since they are both full regular. Therefore, we use PPTL as our underlying logic to define properties. In addition, a prototype of bounded model checker based on NuSMV2 for PPTL has been developed. This allows us to bounded model checking for PPTL in an automatical way. Our experience shows that the bounded model checking approach presented in this paper for PPTL is feasible and useful.

5 Conclusion

We proposed some basic theory of BMC for PPTL including bounded semantics and process of reducing BMC to a SAT problem. We also gave an example to show the process of BMC for PPTL. Although we established the basic theory on bounded model checking for PPTL, however, we have not investigated the completeness and complexity of it. In the future, we will further investigate the completeness and complexity of BMC for PPTL. Moreover, we will also need to improve the prototype of our model checker of BMC for PPTL so that automatical verification can be conducted. In addition, some practical case studies are also indispensable in the future.

References

1. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)

2. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–67. IEEE, New York (1977)
3. Biere, A., Clarke, E., et al.: Bounded Model checking. Advances in Computers, vol. 58, pp. 117–148. Academic Press, London (2003)
4. Liu, C.L., Layland, J.W.: Scheduling algorithm for multiprogramming in a hard real-time environment. Journal of the ACM 20(1), 46–61 (1973)
5. Tian, C., Duan, Z.: Model Checking Propositional Projection Temporal Logic based on SPIN. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 246–265. Springer, Heidelberg (2007)
6. Clarke, E., Kroening, D., et al.: Computational Challenges in Bounded Model Checking. International Journal on Software Tools for Technology Transfer 7(2), 174–183 (2005)
7. Clark, E., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
8. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y.: Benefits of Bounded Model Checking at an Industrial Setting. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 436–453. Springer, Heidelberg (2001)
9. Holzmann, G.J.: SPIN Model Checker: The Primer and Reference Manual (September 4, 2003)
10. http://www.cs.cmu.edu/~modelcheck/bmc.html
11. Quielle, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
12. Burch, J.R., Clarke, E., McMillan, K.L., Dill, D.L., Hwang, J.: Symbolic Model Checking: 10^{20} States and Beyond. Information and Computation 98(2), 142–170 (1992)
13. McMillian, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993) ISBN: 0-7923-9380-5
14. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5, 394–397 (1962)
15. Huth, M., Ryan, M.: Logic in Computer Science: Modeling and Reasoning about Systems, 2nd edn., Part 3. China Machine Press (2007) ISBN:978-7-111-21397-0
16. Moszkowski, B.: Reasoning about digital circuits. Ph.D. Thesis. Stanford University, Stanford (1983)
17. Grumberg, O., Long, D.E.: Model checking and modular verification. Journal ACM Transactions on Programming Languages and Systems TOPLAS Homepage Archive 16(3), 843–871 (1994)
18. Tian, C., Duan, Z.: Propositional projection temporal logic, Büchi automata and ω-regular expressions. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 47–58. Springer, Heidelberg (2008)
19. Zhang, W.: Bounded Semantics of CTL and SAT-Based Verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 286–305. Springer, Heidelberg (2009)
20. http://www.cs.cmu.edu/~modelcheck/bmc.html
21. Duan, Z.: An Extended Interval Temporal Logic and a Framing Technique for Temporal Logic Programming. Ph.D. Thesis, University of Newcastle upon Tyne (May 1996)
22. Duan, Z., Tian, C., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. Acta Informatica 45(1), 43–78 (2008)
23. Duan, Z., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic. Technical Report No.1, Institute of Computing Theory and Technology, Xidian University, Xi'an P.R.China (2005)
24. Duan, Z., Yang, X., Kounty, M.: Framed Temporal Logic Programming. Science of Computer Programming 70(1), 31–61 (2008)