An Improved Decision Procedure for Propositional Projection Temporal Logic*

Zhenhua Duan and Cong Tian**

Institute of Computing Theory & Technology and ISN Laboratory
Xidian University, Xi'an, 710071, China
{zhhduan, ctian}@mail.xidian.edu.cn

Abstract. A new decision procedure for Propositional Projection Temporal Logic (PPTL) is proposed which is an improvement to the decision procedure given in [4]. The main contribution of the paper is as follows: (1) the relationship between paths in the NFG of a formula R and its models is established and proved; (2) a new Labeled NFG (LNFG) with a set of labels (propositions) is defined; (3) given a formula R, an LNFG of R can be generated by the new decision algorithm, and all models of R can be found; (4) based on the new decision procedure, an improved model checking algorithm is presented and implemented.

Keywords: Propositional Projection Temporal Logic, Satisfiability, Decision Procedure, Model Checking, Verification.

1 Introduction

Propositional Projection Temporal Logic (PPTL) [3] is an extension of Propositional Interval Temporal Logic (PITL) [2]. It is a useful logic for specification and verification of concurrent systems. The advantages of using PPTL are in three folds: (1) PPTL has the expressiveness of full regular expressions [12]. (2) Intervals are useful in specifying state sensitive properties. For example, p holding at the 6^{th} state can be described by $len(6); p$, and $len(10); len(6) \wedge \Diamond p; true$ means p holds between the 10^{th} and 16^{th} states. It is cumbersome to specify these properties by Propositional Linear Temporal Logic (PLTL) [11], Computation Tree Logic (CTL) [10] and their variations. (3) Chop and projection constructs are useful in the specification of sequential and iterative behaviors respectively. These properties cannot (or with difficulty) be described by PLTL or CTL.

Decidability of temporal logics is a fundamental issue in verification, especially model checking. Therefore, it is important to investigate the decidability of PPTL and PITL so that model checking PPTL and PITL can be done. Bowman and Thompson presented a tableaux-based decision procedure for PITL over finite models in 2003 [8]. Later in [4], a decision procedure for PPTL with infinite models was given and the complexity was proved to be non-elementary [13]. The decision procedure can easily

* This research is supported by the NSFC Grant No. 61003078, 60433010, 60873018 and 60910004, National Program on Key Basic Research Project of China (973 Program) Grant No.2010CB328102 and SRFDP Grant 200807010012.

** Corresponding author.

J.S. Dong and H. Zhu (Eds.): ICFEM 2010, LNCS 6447, pp. 90–105, 2010.

be implemented and also applied to PITL with minor modification. With this decision procedure, normal form, Normal Form Graph (NFG) and Labeled NFG (LNFG) play important roles for constructing models of a formula. Generally, given a formula P, all models of P are contained in its NFG, and determined by its LNFG. That is, an NFG can be treated as an automaton structure while an LNFG can be viewed as an omega automaton with a specified accepting condition. The decision procedure works well for most of PPTL formulas. Nevertheless, accepting conditions on LNFG was only simply expressed in an informal way. When further implementing the decision procedure, we found that to precisely define accepting conditions is neither a trivial nor an intuitive work. What is more, only by the unprecise accepting condition, some formulas cannot be handled in a proper way. For instance, the LNFG of formula, $q \wedge (\bigcirc empty; q) \wedge \Box(p \wedge \bigcirc \bigcirc empty; q)$, can be constructed by the old algorithm as depicted in Fig.1. By the accepting condition, the formula is unsatisfiable. However, in fact, the formula is satisfiable since the finiteness of chop formula $\bigcirc empty; q$ can be satisfied.

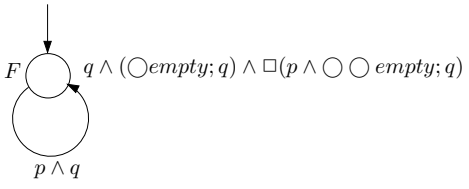


Fig. 1. LNFG of $q \wedge (\bigcirc empty; q) \wedge \Box(p \wedge \bigcirc \bigcirc empty; q)$

The problem is caused by the case in which a circle path occurs, and the circle is composed of nodes (formulas) involving different chop formulas but labeled by the same mark F in the LNFG. To deal with this kind of formulas, only one label F is not sufficient. As a result, we further give an improved decision procedure by means of using distinguishing labels and defining formal accepting conditions in this paper. The improvements focus on four folds: (1) chop formulas and chop components of PPTL formulas are formally defined; (2) finite satisfactory condition, FSC_Property, over infinite paths in the NFG of a formula is explicitly defined to capture the essential property dwelling in the chop construct; (3) models of a formula are precisely specified in its NFG by applying FSC_Property; (4) finally, LNFG of a formula with l_k labels is constructed to explicitly illustrate whether or not FSC_Properties are satisfied over infinite paths. By the above improvements, an improved decision procedure for PPTL with infinite models are formalized. This decision procedure enables us to handle not only basic PPTL formulas but also extended projection star construct. Further, based on the new decision procedure, an improved model checking algorithm to the one presented in [17] is also proposed and implemented. A model checker based on SPIN has been developed recently. By our experience, the decision procedure and the model checking algorithm work well. In addition to verification, the decision procedure is also significant in the theory concerning complementing infinite objections, which is a hard issue [15], since complementing infinite words is implicitly involved in the decision procedure.

The paper is organized as follows. The next section briefly presents the preliminaries, including the syntax and semantics of the underlying logic and the definition of normal form. In section 3, normal form graph is presented in details. Further, the enhanced decision procedure is illustrated in section 4. In section 5, based on the improved decision procedure, an improved model checking algorithm is also presented. Finally, conclusions are drawn in section 6.

2 Preliminaries

2.1 Propositional Projection Temporal Logic

Let *Prop* be a countable set of atomic propositions. The formula P of PPTL is given by the following grammar:

$$P ::= p \mid \bigcirc P \mid \neg P \mid P_1 \vee P_2 \mid (P_1, ..., P_m)\ prj\ P \mid (P_1, ..., (P_i, ..., P_j)^{\circledast}, ..., P_m)\ prj\ Q$$

where $p \in Prop$, $P_1, ..., P_m$, P and Q are all well-formed PPTL formulas. $\bigcirc$ (*next*), prj (*projection*) and $prj^{\circledast}$ (*projection star*) are basic temporal operators.

Following the definition of Kripke's structure [1], we define a state s over *Prop* to be a mapping from *Prop* to $B = \{true, false\}$, $s : Prop \longrightarrow B$. We will use $s[p]$ to denote the valuation of p at state s. An interval σ is a non-empty sequence of states, which can be finite or infinite. The length, $|\sigma|$, of σ 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 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 $\leq$ as $\leq -\{(\omega, \omega)\}$. To simplify definitions, we will denote σ by $\langle s_0, ..., s_{|\sigma|}\rangle$, 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 $\langle s_i, ..., s_j\rangle$, $\sigma^{(k)}$ $(0 \leq k \leq |\sigma|)$ denotes $\langle s_k, ..., s_{|\sigma|}\rangle$, i.e., k^{th} suffix of σ, and σ^k $(0 \leq k \leq |\sigma|)$ denotes $\langle s_0, ..., s_k\rangle$, i.e., k^{th} prefix of σ. Note that for an infinite interval σ, the ω^{th} prefix σ^ω and suffix $\sigma^{(\omega)}$ are undefined and denoted by $\perp$. Further, the concatenation of a finite σ with another interval (or empty string) σ' denoted by $\sigma \cdot \sigma'$ is defined as follows.

$$\sigma_1 \cdot \sigma_2 = \begin{cases} \sigma_1 & \text{if } \sigma_2 = \epsilon \\ \sigma_2 & \text{if } \sigma_1 = \epsilon \\ \langle s_0, ..., s_i, s_{i+1}, ...\rangle & \text{if } \sigma_1 = \langle s_0, ..., s_i\rangle \text{ and } \sigma_2 = \langle s_{i+1}, ...\rangle \text{ and } i \in N_0 \\ \perp & \text{otherwise} \end{cases}$$

Let $\sigma = \langle s_0, s_1, \ldots, s_{|\sigma|}\rangle$ be an interval and $r_1, \ldots, r_h$ be integers $(h \geq 1)$ such that $0 \leq r_1 \leq r_2 \leq \ldots \leq r_h \leq |\sigma|$. The projection of σ onto $r_1, \ldots, r_h$ is the interval (namely projected interval)

$$\sigma \downarrow (r_1, \ldots, r_h) = \langle s_{t_1}, s_{t_2}, \ldots, s_{t_l}\rangle$$

where $t_1, \ldots, t_l$ is obtained from $r_1, \ldots, r_h$ by deleting all duplicates. That is, $t_1, \ldots, t_l$ is the longest strictly increasing subsequence of $r_1, \ldots, r_h$. For instance,

$$\langle s_0, s_1, s_2, s_3, s_4\rangle \downarrow (0, 0, 2, 2, 2, 3) = \langle s_0, s_2, s_3\rangle$$

This is convenient to define an interval obtained by taking the endpoints (rendezvous points) of the intervals over which $P_1, \ldots, P_m$ are interpreted in the projection construct.

We need also to generalize the notation of $\sigma \downarrow (r_1, ..., r_h)$ to allow r_h to be ω. For an interval $\sigma =< s_0, s_1, ..., s_{|\sigma|} >$ and $0 \le r_1 \le r_2 \le ... \le r_h \le |\sigma|$, we define

$$\sigma \downarrow (r_1, ..., r_{h-1}, r_h) = \sigma \downarrow (r_1, ..., r_{h-1}, r_h) \text{ if } r_h \text{ is not } \omega$$
$$\sigma \downarrow (r_1, ..., r_{h-1}, r_h) = \sigma \downarrow (r_1, ..., r_{h-1}) \text{ if } r_h \text{ is } \omega$$

For instance,

$$< s_0, ... > \downarrow (0, 1, 3, 4, \omega) =< s_0, s_1, s_3, s_4 >$$

An interpretation is a tuple $\mathcal{I} = (\sigma, k, j)$, where σ is an interval, k is an integer, and j an integer or ω such that $k \le j \le |\sigma|$. We use the notation $(\sigma, k, j) \models P$ to denote that formula P is interpreted and satisfied over the subinterval $\langle s_k, ..., s_j \rangle$ of σ with the current state being s_k. The satisfaction relation ($\models$) is inductively defined as follows:

I – prop $\mathcal{I} \models p$ iff $s_k[p] = true$, for any given proposition p
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 $k < j$ and $(\sigma, k+1, j) \models P$
I – prj $\mathcal{I} \models (P_1, ..., P_m)\ prj\ Q$ if there exist integers $k = r_0 \le r_1 \le ... \le r_m \le j$ such that $(\sigma, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_l) \models P_l, 1 < l \le m$, and $(\sigma', 0, |\sigma'|) \models Q$ for one of the following σ' :
(a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, ..., r_m) \cdot \sigma_{(r_m+1..j)}$ or
(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, ..., r_h)$ for some $0 \le h \le m$
I – prj$^{\circledast}$ $\mathcal{I} \models (P_1, ..., (P_i, ..., P_s)^{\circledast}, ..., P_m)\ prj\ Q$ iff :
$1 \le i \le s \le m$ and $\exists n \in N_0, \mathcal{I} \models (P_1, ..., (P_i, ..., P_s)^{(n)}, ..., P_m)\ prj\ Q$
or: $s = m$ and there exist infinitely many integers $k = r_0 \le r_1 \le ...$
$\le r_h \le \omega$, such that
$(\sigma, r_{l-1}, r_l) \models P_l, 0 < l < i$,
$(\sigma, r_{l-1}, r_l) \models P_t, l \ge i, t = i + ((l - i)\ mod\ (s - i + 1))$,
and $(\sigma', 0, |\sigma'|) \models Q$, where $\sigma' = \sigma \downarrow (r_0, r_1, ...)$.

Fig.2 shows the possible semantics of $(P_1, P_2)\ prj\ Q$. Here Q and P_1 start to be interpreted at state t_0; subsequently, P_1 and P_2 are interpreted sequentially; Q is interpreted in parallel with $(P_1; P_2)$ over the interval consisting of endpoints of subintervals over which P_1 and P_2 are interpreted. The possible three cases are given: (a) P_2 terminates before Q; (b) Q and P_2 terminate at the same state; (c) Q terminates before P_2. Projection construction is useful in the specification of concurrent system.

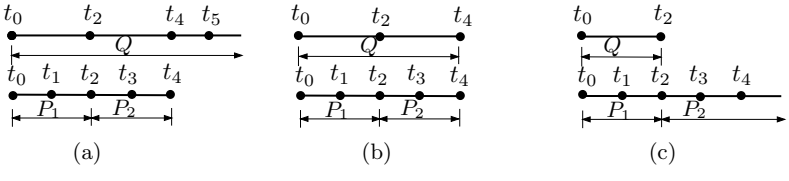


Fig. 2. Semantics of $(P_1, P_2)\ prj\ Q$

In order to avoid an excessive number of parentheses, the following precedence rules are used as shown in Table 1, where 1 = highest and 5 = lowest.

Table 1. Precedence Rules

1 $\neg$	2 $\bigcirc$, $\odot$, $\Diamond$, $\Box$	3 $\wedge$, $\vee$
4 $\rightarrow$, $\leftrightarrow$	5 prj, ;	

The abbreviations $true$, $false$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $true \stackrel{\text{def}}{=} P \vee \neg P$ and $false \stackrel{\text{def}}{=} P \wedge \neg P$. Also we have the following derived formulas,

$$
\begin{array}{rcl@{\qquad}rcl}
empty & \stackrel{\text{def}}{=} & \neg \bigcirc true & more & \stackrel{\text{def}}{=} & \neg empty \\
\bigcirc^0 P & \stackrel{\text{def}}{=} & P & \bigcirc^n P & \stackrel{\text{def}}{=} & \bigcirc(\bigcirc^{n-1} P) \\
len(0) & \stackrel{\text{def}}{=} & empty & len(n) & \stackrel{\text{def}}{=} & \bigcirc len(n-1), n \geq 1 \\
skip & \stackrel{\text{def}}{=} & len(1) & \odot P & \stackrel{\text{def}}{=} & empty \vee \bigcirc P \\
P; Q & \stackrel{\text{def}}{=} & (P, Q)\ prj\ empty & \Diamond P & \stackrel{\text{def}}{=} & true; P \\
\Box P & \stackrel{\text{def}}{=} & \neg\Diamond\neg P & keep(P) & \stackrel{\text{def}}{=} & \Box(\neg empty \rightarrow P) \\
halt(P) & \stackrel{\text{def}}{=} & \Box(empty \leftrightarrow P) & fin(P) & \stackrel{\text{def}}{=} & \Box(empty \rightarrow P) \\
P^* & \stackrel{\text{def}}{=} & (P^{\circledast})\ prj\ empty & P^+ & \stackrel{\text{def}}{=} & P; P^*
\end{array}
$$

where $n \geq 1$, $\odot$ (weak next), $\Box$ (always), $\Diamond$ (sometimes), ; (chop) and + (plus) are derived temporal operators; *empty* denotes an interval with zero length, and *more* means the current state is not the final one over an interval; $halt(P)$ is true over an interval if and only P is true at the final state, $fin(P)$ is true as long as P is true at the final state and $keep(P)$ is true if P is true at every state ignoring the final one.

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.

In some circumstances, an undefined interval may be involved in the interpretations. Let Σ be the set of all intervals, and $\Sigma_\perp = \Sigma \cup \{\perp\}$, where $\perp$ is an undefined interval. We can extend the satisfaction relation to $\Sigma_\perp$. For an undefined interval $\perp$, and any formula P, we define $\perp \models P$.

Theorem 1. Let P be a PPTL formula. We have,

(1) $fin(P) \equiv P \wedge empty \vee \bigcirc fin(P)$
(2) $keep(P) \equiv empty \vee P \wedge \bigcirc keep(P)$
(3) $halt(P) \equiv P \wedge empty \vee \neg P \wedge \bigcirc halt(P)$ □

2.2 Normal Form of PPTL

Normal form is an important notation for constructing NFGs of PPTL formulas. In the following, we briefly give its definition and relevant concepts. The detailed explanation can be found in [4].

Definition 1 (Normal Form). Let Q_p be the set of atomic propositions appearing in a PPTL formula Q. The normal form of Q can be defined by,

$$Q \equiv \bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty) \vee \bigvee_{i=0}^{n_1}(Q_{ci} \wedge \bigcirc Q'_i)$$

where $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$, $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, $\dot{r}$ denotes r or $\neg r$; Q'_i is a PPTL formula without " $\vee$ " being the main operator. □

According to the definition, in a normal form, $p \wedge q \wedge \bigcirc(\Box p \vee q)$ must be written as $p \wedge q \wedge \bigcirc(\Box p) \vee p \wedge q \wedge \bigcirc q$ since " $\vee$ " is the main operator of $\Box p \vee q$. Implicitly, Q'_i is also not permitted to be the form of $\neg \bigwedge_{k=1}^{n\geq 2} P_k$. Further, for convenience, we call $\bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty)$ the terminating part whereas $\bigvee_{i=0}^{n_1}(Q_{ci} \wedge \bigcirc Q'_i)$ the non-terminating part of the normal form.

Definition 2 (Complete Normal Form). Let Q_p be the set of atomic propositions appearing in a PPTL formula Q. The complete normal form of Q is defined by,

$$Q \equiv \bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty) \vee \bigvee_{i=0}^{n_1}(Q_{ci} \wedge \bigcirc Q'_i)$$

where like in the normal form, $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$, $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, $\dot{r}$ denotes r or $\neg r$; further $\bigvee_i Q_{ci} \equiv true$ and $\bigvee_{i \neq j}(Q_{ci} \wedge Q_{cj}) \equiv false$; Q'_i is an arbitrary PPTL formula. □

Note that a complete normal form may be not a normal form, since "$\vee$" is possibly the main operator of Q'_i.

Notice that if Q is transformed into complete normal form, $Q \equiv \bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty) \vee \bigvee_{i=0}^{n_1}(Q_{ci} \wedge \bigcirc Q'_i)$, then $\neg Q$ can be transformed into its normal form, $\neg Q \equiv \bigwedge_{j=0}^{n_0} \neg Q_{ej} \wedge empty \vee \bigvee_{i=0}^{n_1}(Q_{ci} \wedge \bigcirc \neg Q'_i)$. Therefore, it is useful in transforming negation constructs into their normal forms. An important conclusion is that any PPTL formula can be transformed into its normal form [4].

3 Normal Form Graph

Normal Form Graph (NFG) was introduced in [4] for the purpose of obtaining models of PPTL formulas. For convenience, here we also briefly introduce its definition. For a PPTL formula P, NFG of P is a directed graph, $G = (CL(P), EL(P), V_0)$, where $CL(P)$ denotes the set of nodes, $EL(P)$ the set of edges , and $V_0 \subseteq CL(P)$ the set of root nodes in the graph. In $CL(P)$, each node is specified by a formula in PPTL, while in $EL(P)$, each edge is a directed arc, labeled with a state formula Q_e, from node Q to node R and identified by a triple, (Q, Q_e, R). Accordingly, for convenience sometimes, a node in an NFG or LNFG is called a formula. NFG of a PPTL formula is inductively defined in Definition 3.

Definition 3 (Normal Form Graph, NFG). For a PPTL formula P, the set $CL(P)$ of nodes and $EL(P)$ of edges connecting nodes in $CL(P)$ are inductively defined as follows:

1. If P is not in the form of $\bigvee_i P_i$, $P \in CL(P)$, $V_0 = \{P\}$, otherwise for each i $P_i \in V_0$, $P_i \in CL(P)$;
2. For all $Q \in CL(P) \setminus \{\varepsilon, false\}$, if Q is rewritten into its normal form $\bigvee_{j=0}^{h}(Q_{ej} \wedge empty) \vee \bigvee_{i=0}^{k}(Q_{ci} \wedge \bigcirc Q'_i)$, then $\varepsilon \in CL(P)$, $(Q, Q_{ej}, \varepsilon) \in EL(P)$ for each j, $1 \leq j \leq h$; $Q'_i \in CL(P)$, $(Q, Q_{ci}, Q'_i) \in EL(P)$ for all i, $1 \leq i \leq k$;

The NFG of formula P is the directed graph $G = (CL(P), EL(P), V_0)$. □

It is important that for any PPTL formula P, the number of $CL(P)$ is finite [4]. An algorithm for constructing NFGs of a formula is given in Algorithm NFG. Although the algorithm is similar as the one given in [4], the proofs of Theorem 2, Lemma 3 and Lemma 4 are totally new, and based on the algorithm.

In an NFG, any root node in V_0 is denoted by a circle with an input edge originating from none nodes, ε node is marked by a small black dot, and each of other nodes by a single circle. Each edge is denoted by a directed arc connecting two nodes. A finite path is a finite alternating sequence of nodes and edges, $\pi = \langle n_0, e_0, n_1, e_1, ..., \varepsilon\rangle$ from a root node to the ε node, while an infinite path is an infinite alternating sequence of nodes and edges, $\pi = \langle n_0, e_0, n_1, e_1 ..., n_i, e_i ..., n_j, e_j, n_i, e_i, ..., n_j, e_j, ...\rangle$ departing from the root node with some nodes, e.g. $n_i, ..., n_j$, occurring for infinitely many times. For convenience, we use $\mathsf{Inf}(\pi)$ to denote the set of nodes which infinitely often occur in the infinite path π. In some circumstances, in a path of NFG of formula Q, a node n_i can be replaced by a formula $Q_i \in CL(Q)$ and an edge e_i can be replaced by a state formula $Q_{ie} \in EL(Q)$.

Intuitively, all models of a formula are implicitly contained in its NFG. For easily expressing the relationship between models of formula Q and paths of NFG G of Q, functions $P2M(\pi, G)$ and $M2P(\sigma, G)$ are formally defined bellow. Given a path $\pi = \langle Q, Q_{0e}, Q_1, Q_{1e}, ...\rangle$ in G, an interval σ_π can be obtained by,

$$\sigma_\pi = P2M(\pi, G) = \begin{cases} \langle s_0, s_1, ..., s_n\rangle, \text{ for } 1 \leq i \leq n \\ \quad s_i[q] = true \text{ if } q \in Q_{ie}, \text{ and } s_i[q] = false \text{ if } \neg q \in Q_{ie}, \\ \quad \text{if } \pi = \langle Q, Q_{0e}, Q_1, Q_{1e}, ..., Q_{ne}, \varepsilon\rangle \text{ and } \pi \in G \\ \langle s_0, s_1, ..., (s_i, ..., s_j)^\omega\rangle, \text{ for } 1 \leq k \leq j \\ \quad s_k[q] = true \text{ if } q \in Q_{ke}, \text{ and } s_k[q] = false \text{ if } \neg q \in Q_{ke}, \\ \quad \text{if } \pi = \langle Q, Q_{0e}, Q_1, Q_{1e}, ..., (Q_i, Q_{ie}, ..., Q_j, Q_{je})^\omega\rangle \text{ and } \pi \in G \end{cases}$$

Correspondingly, for a model $\sigma = \langle s_0, s_1, ...\rangle \models Q$, a path π_σ w.r.t the NFG G of Q can be obtained by,

$$\pi_\sigma = M2P(\sigma, G) = \begin{cases} \langle Q, Q_{0e}, Q_1, Q_{1e}, ..., Q_{ne}, \varepsilon\rangle, \text{ for } 1 \leq i \leq n \\ \quad Q_{ie} = \bigwedge_k \dot{q}_k, \dot{q}_k \equiv q_k, \text{ if } s_i[q_k] = true, \\ \quad \text{and } \dot{q}_k \equiv \neg q_k, \text{ if } s_i[q_k] = false, Q_i \in CL(Q) \\ \quad \text{if } \sigma =< s_0, s_1, ..., s_n > \\ \langle Q, Q_{0e}, Q_1, Q_{1e}, ..., (Q_i, Q_{ie}, ..., Q_j, Q_{je})^\omega\rangle, \text{ for } 1 \leq h \leq j \\ \quad Q_{he} = \bigwedge_k \dot{q}_k, \dot{q}_k \equiv q_k, \text{ if } s_h[q_k] = true, \\ \quad \text{and } \dot{q}_k \equiv \neg q_k, \text{ if } s_h[q_k] = false, Q_h \in CL(Q) \\ \quad \text{if } \sigma = \langle s_0, s_1, ..., (s_i, ..., s_j)^\omega\rangle \end{cases}$$

Although in the above an interval σ_π can be defined for a given path π of the NFG of formula Q, whether or not $\sigma_\pi \models Q$ needs to be proved (see Lemma 6 and 13). Similarly, given a model σ of formula Q, $\sigma \models Q$, a path π_σ can be constructed, however, whether or not the path can be found in G also needs to be proved (See Lemma 7 and 9).

Theorem 2. Finite paths in the NFG of PPTL formula P precisely characterize finite models of P.
Proof: It is a consequence of Lemma 3 and 4. □

Lemma 3. For a finite path π in the NFG G of P, $\sigma_\pi \models P$. □

Lemma 4. For a finite interval $\sigma \models Q$, π_σ can be found in the NFG G of Q. □

For infinite models, it is much more intricate because of the involvement of chop and projection operators. In fact, not all of infinite paths in the NFG of a formula are infinite models of the corresponding formula. We investigate the case carefully in the following.

A formula R is called a chop formula if $R \equiv P \wedge Q$ and $P \equiv P_1; P_2$, where P_1, P_2 and Q are any PPTL formulas. Further, $P_1; P_2$ is called a chop component of chop formula R. Note that a chop component is also a chop formula but the reverse may be not true. Formally, a chop formula R_c can be defined as follows

$$R_c ::= P; Q \mid R_c \wedge R_c \mid R_c \wedge R$$

where P, Q and R are any PPTL formulas.

For instance, $(\bigcirc p; q)$, $(p; p^*)$, $(\bigcirc p; q) \wedge \bigcirc r$ are chop formulas while $\bigcirc(\bigcirc p; q)$, $\neg(\bigcirc p; q)$ and p^* are not, where p, q, r are atomic propositions. Note that $P; Q$ is a chop formula but $\neg(P; Q)$ is not. This will be formally analyzed late.

For chop construct $P; Q$, an infinite model $\sigma = \langle s_0, s_1, ..., s_k, ...\rangle \models P; Q$ if and only if there exists $i \in N_0$, such that $\sigma^i \models P$ and $\sigma^{(i)} \models Q$. This implies that if infinite model $\sigma = \langle s_0, s_1, ..., s_k, ...\rangle \models P$ and there is no finite prefix $\sigma^i \models P$ ($i \in N_0$), it fails to satisfy $P; Q$. Note that in the weak version of the chop construct, $P; Q$ is still satisfied in the case. For convenience, we say finiteness of P in strong chop construct $P; Q$ (FSC_Property for short) is satisfiable over an infinite model σ, denoted by predicate $fsc(\sigma, P; Q) = true$, if there exists $i \in N_0$ such that $\sigma^i \models P$ and $\sigma^{(i)} \models Q$. Actually, an infinite path of the NFG of $P; Q$ presents an infinite model of either $P; Q$ or P. So, FSC_Property needs to be considered if a node (formula) is a chop formula.

Correspondingly, in NFGs, when constructing NFG of $P; Q$, initially, $P; Q$ is transformed into its normal form,

$$\begin{aligned} P; Q &\equiv (\bigvee_{j=0}^{n_0} P_{ej} \wedge empty \vee \bigvee_{i=0}^{n_1}(P_{ci} \wedge \bigcirc P'_i)); Q \\ &\equiv \bigvee_{j=0}^{n_0}(P_{ej} \wedge empty; Q) \vee \bigvee_{i=0}^{n_1} P_{ci} \wedge \bigcirc(P'_i; Q) \\ &\equiv \bigvee_{j=0}^{n_0}(P_{ej} \wedge Q) \vee \bigvee_{i=0}^{n_1} P_{ci} \wedge \bigcirc(P'_i; Q) \end{aligned}$$

Subsequently, the new generated formula $P'_i; Q$ needs to be repeatedly transformed into its normal form in the same way. Whenever a final state of P is reached, i.e. $P_e \wedge empty; Q$ is encountered, where P_e is a state formula, FSC_Property of $P; Q$ is satisfied over the path π departing from the root node.

Formally, let $\pi = \langle R, R_{0e}, R_1, R_{1e}, ...\rangle$ be an infinite path in the NFG of formula R, and $\pi^{(k)} = \langle R_k, R_{ke}, ...\rangle$ be the k^{th} suffix of π. Whether or not FSC_Property of R is satisfiable over π, denoted by predicate $FSC(\pi, R)$, can be defined based on $fsc(\pi, R)$ as follows.

Definition 4. $fsc(\pi, R) = true$ if

1. R is not a chop formula, or
2. $R \equiv P; Q$ and there exists $i \in N_0$ such that $R_i \equiv P_{ie} \wedge empty; Q$, or
3. $R \equiv P \wedge Q$ and $fsc(\pi, P) = true$ and $fsc(\pi, Q) = true$, or
4. $R \equiv P \vee Q$ and $fsc(\pi, P) = true$ or $fsc(\pi, Q) = true$

Finally, $FSC(\pi, R) = true$ if $fsc(\pi^{(k)}, R_k) = true$ for all $k \in N_0$. □

In particular, notice that $fsc(\pi, \neg R) = true$ if $R \equiv P; Q$, since in $\neg(P; Q)$, FSC_Property of $P; Q$ needs not to be considered.

Lemma 5. For an infinite model, $\sigma \models \neg(P; Q)$ iff $\forall k \in N_\omega \wedge (\sigma^k \models \neg P \vee \sigma^{(k)} \models \neg Q)$. □

For projection construct, $(P_1, ..., P_m)\ prj\ Q$, it is eventually treated as a chop formula when constructing NFGs according to the following transforming rule. Suppose

$$\begin{aligned} P_1 &\equiv P_{1e} \wedge empty \vee \bigvee_{i=1}^{r}(P_{1i} \wedge \bigcirc P'_{1i}) \\ P_2 &\equiv P_{2e} \wedge empty \vee \bigvee_{l=1}^{s}(P_{2l} \wedge \bigcirc P'_{2l}) \\ Q &\equiv Q_e \wedge empty \vee \bigvee_{j=1}^{k}(Q_j \wedge \bigcirc Q'_j) \end{aligned}$$

then,

$$\begin{aligned} (P_1, P_2)\ \text{prj}\ Q \equiv\ & Q_e \wedge P_{1e} \wedge P_{2e} \wedge empty \\ & \vee \bigvee_{l=1}^{s}(Q_e \wedge P_{1e} \wedge P_{2l} \wedge \bigcirc P'_{2l}) \\ & \vee \bigvee_{j=1}^{k}(P_{1e} \wedge P_{2e} \wedge Q_j \wedge \bigcirc Q'_j) \\ & \vee \bigvee_{j=1}^{k} \bigvee_{l}^{s}(P_{1e} \wedge Q_j \wedge P_{2l} \wedge \bigcirc(P'_{2l}\ ;\ Q'_j)) \\ & \vee \bigvee_{i=1}^{r}(Q_e \wedge P_{1i} \wedge \bigcirc(P'_{1i}\ ;\ P_2)) \\ & \vee \bigvee_{i=1}^{r} \bigvee_{j}^{k}(Q_j \wedge P_{1i} \wedge \bigcirc(P'_{1i}\ ;\ (P_2\ \text{prj}\ Q'_j))) \end{aligned}$$

The above explanation shows that the fitness property for projection constructs and $\neg(P; Q)$ needs not to be considered, and only for chop formulas requires to be treated in a special way. To do so, we need the fixed point theory and Scott's fixed point induction.

Fixed point theory: Every monotonic function F over a complete lattice $\langle A, \subseteq \rangle$ has a unique least fixed point $\bigcup_i F^i(\bot)$ and a unique greatest fixed point $\bigcap_i F^i(\top)$ [14]. □

Scott's fixed-point induction: Suppose D is a complete lattice with bottom $\bot$, $F : D \to D$ is a continuous function, and P is an inclusion subset of D. If $\bot \in P$ and $\forall x \in D.x \in P \Rightarrow F(x) \in P$, then $fix(F) \in P$ [14]. □

In the following, Lemma 6, 7, 8, 10 and Theorem 11 represent the relationship between infinite models and paths in the NFG of a formula.

Lemma 6. For an infinite model σ, $\sigma \models Q$, a π_σ with $FSC(\pi_\sigma, Q) = true$ can be found in the NFG G of Q. □

Lemma 7. In the NFG of R, for an infinite path π, if there exist no chop formulas over π, $\sigma_\pi \models R$. □

In the following, we further discuss the general case of path π for a given formula R. The final conclusion is presented in Theorem 11.

Lemma 8. In the NFG G of formula R, for an infinite path $\pi = \langle R, R_{e0}, R_1, R_{e1}, R_2, ...\rangle$, if $\sigma_\pi^{(i)} \models R_i$, then $\sigma_\pi^{(i-1)} \models R_{i-1}$. □

Corollary 9. In the NFG of formula R, for an infinite path $\pi = \langle R, R_{e0}, R_1, R_{e1}, R_2, ...\rangle$, $\sigma_\pi \models R$ if there exists $i \in N_0$ such that $\sigma_\pi^{(i)} \models R_i$. □

Lemma 10. If π is an infinite path in the NFG G of formula R with $FSC(\pi, R) = true$, and $\sigma_\pi = P2M(\pi, G)$, then $\sigma_\pi \models R$. □

Theorem 11. In the NFG of PPTL formula R, an infinite path π with $FSC(\pi, R) = true$ precisely characterizes infinite models of R.

Proof: The theorem is a direct consequence of Lemma 6 and 10. □

4 Decision Procedure Based on LNFG

To explicitly display whether or not the FSC_Property of a chop formula is satisfied, extra propositions l_k, $k \in N_0$ and $k > 0$, are introduced. Let $Prop_l = \{\mathsf{l}_1, \mathsf{l}_2, ...\}$ be the set of extra propositions. $Prop \cap Prop_l = \emptyset$. Note that these extra propositions are merely employed to mark nodes and are not allowed to appear in a PPTL formula. When constructing NFGs by normal forms, for any chop formula $P; Q$, we equivalently rewrite it as $P \wedge fin(\mathsf{l}_k); Q$. Formally, we have,

$$
\begin{aligned}
& P \wedge fin(\mathsf{l}_k); Q \\
\equiv\ & (\bigvee_{j=0}^{n_0} P_{ej} \wedge empty \vee \bigvee_{i=0}^{n_1}(P_{ci} \wedge \bigcirc P'_i)) \wedge (\mathsf{l}_k \wedge empty \vee \bigcirc fin(\mathsf{l}_k)); Q \\
\equiv\ & (\bigvee_{j=0}^{n_0} P_{ej} \wedge \mathsf{l}_k \wedge empty \vee \bigvee_{i=0}^{n_1}(P_{ci} \wedge \bigcirc(P'_i \wedge fin(\mathsf{l}_k)))); Q \\
\equiv\ & \bigvee_{j=0}^{n_0}(P_{ej} \wedge \mathsf{l}_k \wedge empty; Q) \vee \bigvee_{i=0}^{n_1}(P_{ci} \wedge \bigcirc(P'_i \wedge fin(\mathsf{l}_k); Q)) \\
\equiv\ & \bigvee_{j=0}^{n_0}\underline{(P_{ej} \wedge \mathsf{l}_k \wedge Q)} \vee \bigvee_{i=0}^{n_1}(P_{ci} \wedge \bigcirc(P'_i \wedge fin(\mathsf{l}_k); Q))
\end{aligned}
$$

Thus, by using $fin(\mathsf{l}_k)$, FSC_Property of $P; Q$ is satisfied if there exists an edge where l_k holds. And $fin(\mathsf{l}_k)$ occurring in a node $P \wedge fin(\mathsf{l}_k); Q$ means that FSC_Property of $P; Q$ has not been satisfied at this node. For convenience, for a node in the form of $P \wedge fin(\mathsf{l}_k); Q$ or $\bigwedge_{i=1}^{n} R_i$ with some $R_i \equiv P_i \wedge fin(\mathsf{l}_k); Q_i$, we add an extra label $\tilde{\mathsf{l}}_k$ in this node to mean that the finiteness of some chop formula has not been satisfied at this node.

Accordingly, Labeled Normal Form Graph (LNFG) is defined based on NFG with the usage of l_k propositions.

Definition 5 (Labeled Normal Form Graph, LNFG). For a PPTL formula P, its LNFG is a tuple $G = (CL(P), EL(P), V_0, \mathbb{L} = \{\mathbb{L}_1, ..., \mathbb{L}_m\})$, where $CL(P)$, $EL(P)$ and V_0 are identical to the ones in NFG, while each $\mathbb{L}_k \subseteq CL(P)$, $1 \le k \le m$, is the set of nodes with $\tilde{\mathsf{l}}_k$ labels. □

Algorithm LNFG: Constructing LNFG of a PPTL formula.

Function LNFG(P)
/* precondition: P is a PPTL formula*/
/* postcondition: LNFG(P) computes LNFG of P, $G = (CL(P), EL(P), V_0, \mathbb{L} = \{\mathbb{L}_1, ..., \mathbb{L}_m\})$*/
begin function
  case
    P is $\bigvee_i P_i$: $CL(P) = \{P_i | P_i$ appears in $\bigvee_i P_i\}$; $Mark[P_i] = 0$ for each i;
      $V_0 = \{P_i | P_i$ appears in $\bigvee_i P_i\}$;
    P is not $\bigvee_i P_i$: $CL(P) = \{P\}$; $Mark[P] = 0$; $V_0 = \{P\}$;
  end case
  $EL(P) = \emptyset$; AddE = AddN = 0; $k = 0$; $\mathbb{L} = \emptyset$;
  while there exists $R \in CL(P) \setminus \{\varepsilon, false\}$, and $mark[R] == 0$
    if $R \equiv P; Q$ or $R \equiv \bigwedge_{i=1}^{n} R_i$ and $\exists R_i \equiv P_i; Q_i$ and no $fin(k)$ has been added in the chop construct
      k=k+1; Rewrite R as $P \wedge fin(\mathsf{l_k}); Q$ or $\bigwedge_{i=1}^{n} R_i$ with some R_i being $P_i \wedge fin(\mathsf{l_k}); Q_i$;
        $\mathbb{L}_k = \{R\}$; $\mathbb{L} = \mathbb{L} \cup \{\mathbb{L}_k\}$;
      if there exist $P \wedge fin(\mathsf{l_s}); Q$ or $\bigwedge_{i=1}^{n} R'_i$ with some $R'_i \equiv P \wedge fin(\mathsf{l_s}); Q$, and $P \wedge fin(\mathsf{l_x}); Q$ or $\bigwedge_{i=1}^{n} R'_i$ with some $R'_i \equiv P \wedge fin(\mathsf{l_x}); Q$, $1 \le l < x < k$
        delete the node $P \wedge fin(\mathsf{l_k}); Q$ or $\bigwedge_{i=1}^{n} R'_i$ with some $R'_i \equiv P \wedge fin(\mathsf{l_s}); Q$;
        re-adding the edges to $P \wedge fin(\mathsf{l_s}); Q$ or $\bigwedge_{i=1}^{n} R'_i$; continue;
    Q =NF(R); $mark[R]$ =1; /*marking R is decomposed*/
    case
      Q is $\bigvee_{j=1}^{h} Q_{ej} \wedge empty$: AddE=1; /*first part of NF needs added*/
      Q is $\bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$: AddN=1; /*second part of NF needs added*/
      Q is $\bigvee_{j=1}^{h} Q_{ej} \wedge empty \vee \bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$: AddE=AddN=1; /*both parts need added*/
    end case
    if AddE == 1 **then** /*add first part of NF*/
      $CL(P) = CL(P) \cup \{\varepsilon\}$; $EL(P) = EL(P) \cup \bigcup_{j=1}^{h} \{(R, Q_{ej}, \varepsilon)\}$; AddE=0;
    if AddN == 1 **then for** $i = 1$ **to** k, /*add second part of NF*/
      if $Q'_i \notin CL(P)$,$CL(P) = CL(P) \cup \bigcup_{i=1}^{k} \{Q'_i\}$;
        if Q'_i is not $false$, $mark[Q'_i]$=0; **else** $mark[Q'_i]$=1;
      $EL(P) = EL(P) \cup \bigcup_{i=1}^{k} \{(R, Q_i, Q'_i)\}$; AddN=0;
  end while
  return G;
End function

Algorithm LNFG based on Algorithm NFG is given by further rewriting the chop component $P; Q$ as $P \wedge fin(\mathsf{l_k}); Q$ for some $k \in N_0$ whenever a new chop formula is encountered. The algorithm uses $mark[]$ to indicate whether or not a node needs to be decomposed. If $mark[P] = 0$ (unmarked), P needs further to be decomposed, otherwise if $mark[P] = 1$ (marked), P has been decomposed or needs not to be done. Function NF is used to transform a formula into its normal form. Further, in the algorithm, two global boolean variables AddE and AddN are employed to indicate whether or not terminating part and non-terminating part of the normal form are encountered respectively.

Lemma 12. For an infinite path π in the LNFG, $G = (CL(R), EL(R), V_0, \mathbb{L} = \{\mathbb{L}_1, ..., \mathbb{L}_m\})$, of PPTL formula R, $FSC(\pi, Q) = true$ iff $\mathsf{Inf}(\pi) \not\subseteq \mathbb{L}_i$ for any $1 \le i \le m$. □

Theorem 13. In the LNFG of a formula P, finite paths precisely characterize finite models of P; infinite paths with $\mathsf{Inf}(\pi) \not\subseteq \mathbb{L}_i$, for all $1 \leq i \leq m$ precisely characterize infinite models of P.

Proof: For finite case, it has been proved in Theorem 2. For infinite case, it is a direct consequence of Theorem 11 and Lemma 12. □

Consequently, a decision procedure for checking the satisfiability of a PPTL formula P can be constructed based on the LNFG of P. In the following, a sketch of the procedure, Algorithm CHECK in pseudo code, is demonstrated.

In order to use our approach, we have developed a tool in C++. With this tool, all the algorithms introduced above have been implemented. For any PPTL formula, the tool can automatically transform it to its normal form, and LNFG, then check whether the formula is satisfiable or not.

Algorithm CHECK: Checking whether or not P is satisfiable.

```
Function CHECK(P)
/* precondition: P is a PPTL formula*/
/* postcondition: CHECK(P) checks whether formula P is satisfiable or not.*/
begin function
    G =LNFG(P);
    if there exists ε node in CL(P),
    return P is satisfiable with finite models;
    if there exists infinite path π with Inf(π) ⊈ L_i, for all 1 ≤ i ≤ m
    return P is satisfiable with infinite models;
    else return unsatisfiable;
end function
```

Example 1. Checking the satisfiability of formula $P \equiv (p \wedge \Box \bigcirc p; \bigcirc \Box q) \wedge (\Box r; \bigcirc \Box q)$.

By Algorithm LNFG, LNFG $G = (CL(P), EL(P), V_0, \mathbb{L} = \{\mathbb{L}_1, ..., \mathbb{L}_m\})$ of formula $(p \wedge \Box \bigcirc p; \bigcirc \Box q) \wedge (\Box r; \bigcirc \Box q)$ is constructed as depicted in Fig.3, where $CL(P) = \{n_0, n_1\}$, $EL(P) = \{(n_0, p \wedge r, n_0), (n_0, p \wedge r, n_1), (n_1, p, n_1)\}$, $V_0 = \{n_0\}$, $\mathbb{L} = \{\mathbb{L}_1, \mathbb{L}_2\}$, $\mathbb{L}_1 = \{n_0, n_1\}$ and $\mathbb{L}_2 = \{n_0\}$. The formula is unsatisfiable since there exists no node without $\tilde{\mathsf{L}_1}$ label which is reachable from n_0 and n_1. □

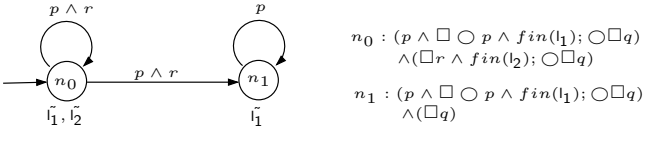
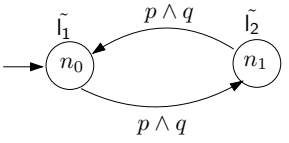


Fig. 3. LNFG of $(p \wedge \Box \bigcirc p; \bigcirc \Box q) \wedge (\Box r; \bigcirc \Box q)$

Example 2. Re-checking the formula given in the introduction.

The example presented in the introduction, which cannot be handled by the old algorithm can now properly be treated. The new LNFG of formula $q \wedge (\bigcirc empty; q) \wedge \Box(p \wedge \bigcirc\bigcirc empty; q)$ is illustrated in Fig.4. The formula is satisfiable since for the only infinite path $Inf(\pi) \not\subseteq \mathbb{L}_i$, $i = 1, 2$. □



$n_0 : q \wedge (\bigcirc empty \wedge fin(l_1); q) \wedge \Box(p \wedge \bigcirc \bigcirc empty; q)$

$n_1 : q \wedge (\bigcirc empty \wedge fin(l_2); q) \wedge \Box(p \wedge \bigcirc \bigcirc empty; q)$

Fig. 4. LNFG of $q \wedge (\bigcirc empty; q) \wedge \Box(p \wedge \bigcirc \bigcirc empty; q)$

Example 3. Checking the satisfiability of formula $(p \wedge skip; q \wedge skip)^+ \wedge \Box more; r$.

The formula is obviously unsatisfiable since $(p \wedge skip; q \wedge skip)^+ \wedge \Box more$ cannot be satisfied with finite models. However, by the old algorithm, the LNFG of $(p \wedge skip; q \wedge skip)^+ \wedge \Box more$ can be constructed as illustrated in Fig.5 (1) which shows the formula is satisfiable. With the new algorithm, the LNFG is depicted in Fig.5 (2) which indicates the formula is unsatisfiable. □

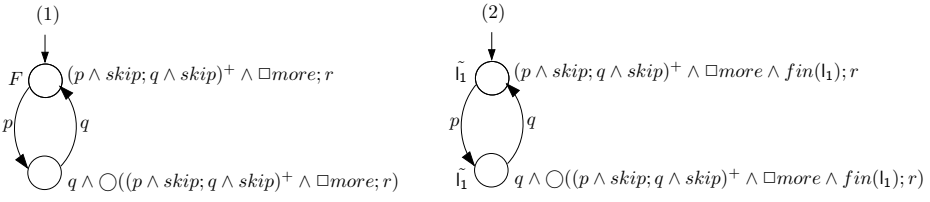


Fig. 5. LNFG of $(p \wedge skip; q \wedge skip)^+ \wedge \Box more; r$

5 Model Checking PPTL

Based on the new decision procedure, the PPTL model checker based on SPIN has also been improved. To this end, a new transformation from LNFGs to Büchi automata is presented. We first transform an LNFG to a Generalized Büchi Automata (GBA) [16]. Factually, an LNFG contains all the information of the corresponding GBA. The set of nodes is in fact the set of locations in the corresponding GBA; each edge (v_i, Q_e, v_j) forms a transition; there exists only one initial location, the root node; the set of accepting locations consists of ε node and the nodes which can appear in infinite paths for infinitely many times. Given an LNFG $G = (CL(P), EL(P), V_0, \mathbb{L} = \{\mathbb{L}_1, ..., \mathbb{L}_m\})$ of formula P, an GBA, $B = (Q, I, \delta, F = \{F_1, ..., F_m\})$, can be constructed as follows.

- Sets of the locations Q and the initial locations I: $Q = V$, and $I = \{v_0\}$.
- Transition δ: Let $\dot{q}_k$ be an atomic proposition or its negation, and we define a function $atom(\bigwedge_{k=1}^{m_0} \dot{q}_k)$ for picking up atomic propositions or their negations appearing in $\bigwedge_{k=1}^{m_0} \dot{q}_k$ as follows,

$$atom(true) = true$$
$$atom(\dot{q}_k) = \begin{cases} \{q_k\}, \text{ if } \dot{q}_k \equiv q_k \ 1 \le k \le l \\ \{\neg q_k\}, \text{ otherwise} \end{cases}$$
$$atom(\bigwedge_{k=1}^{m_0} \dot{q}_k) = atom(\dot{q}_1) \cup atom(\bigwedge_{k=2}^{m_0} \dot{q}_k)$$

For each $e_i = (v_i, Q_e, v_{i+1}) \in E$, there exists $v_{i+1} \in \delta(v_i, atom(Q_e))$. For node ε, $\delta(\varepsilon, \epsilon) = \{\varepsilon\}$.

– Accepting set $F = \{F_1, ..., F_m\}$: It has been proved that infinite paths with $\mathsf{Inf}(\pi) \nsubseteq \mathbb{L}_i$ for all $1 \le i \le m$ precisely characterize infinite models of P. This can be equivalently expressed by "infinite paths with $\mathsf{Inf}(\pi) \cap \overline{\mathbb{L}_i} \neq \emptyset$ for all $1 \le i \le m$ precisely characterize infinite models of P", where $\overline{\mathbb{L}_i}$ denotes $CL(P) \setminus \mathbb{L}_i$. So, $F_i = \overline{\mathbb{L}_i}$ for each i. In addition, by employing the stutter extension rule, $\{\varepsilon\}$ is also an accepting set.

Formally, algorithm Lnfg-Gba in pseudo code is given for transforming an LNFG to a GBA.

Algorithm Lnfg-Gba: Transforming an LNFG to a GBA.
Function Check(P)
/* precondition: $G = (CL(P), EL(P), V_0, \mathbb{L} = \{\mathbb{L}_1, ..., \mathbb{L}_m\})$ is the LNFG of PPTL formula P*/
/* postcondition: Lnfg-Sba(G) computes an GBA $B = (Q, I, \delta, F = \{F_1, ..., F_n\})$ from G*/
begin function
  $Q = \emptyset$; $F = \{F_1, ..., F_m\}$; $F_i = \emptyset$, $1 \le i \le m$; $I = \emptyset$;
  for each node $v_i \in V$, add a state q_i to Q, $Q = Q \cup \{q_i\}$; **end for**
  for each node $v_i \in \overline{\mathbb{L}_i}$, add a state q_i to F_i, $F_i = F_i \cup \{q_i\}$;**end for**
  if v_i is ε, $F = F \cup \{q_i\}$; $\delta(q_i, \epsilon) = \{q_i\}$;
  if $q_0 \in V_0$, $I = I \cup \{q_0\}$;
  for each edge $e = (v_i, P_e, v_j) \in E$, $q_j \in \delta(q_i, atom(P_e))$;**end for**
  return $B = (Q, \Sigma, I, \delta, F)$
end function

Consequently, the algorithm presented in [16] for transforming a GBA to a Büchi automaton can be applied to further transform the GBAs to Bü chi automata.

By transforming PPTL formulas to Büchi automata which precisely characterize the models of the corresponding formulas, an automaton based model checking algorithm for PPTL can be formalized. The algorithm can further be implemented within the model checker SPIN. With our model checking algorithm for PPTL, the system to be verified is modeled as a B*ü*chi automaton A_s, while the property is specified by a PPTL formula P. To check whether the system satisfies P or not, $\neg P$ is transformed into an LNFG, and further a B*ü*chi automaton A_p. The system can be verified by computing the product automaton of A_s and A_p, and then checking whether the words accepted by the product automaton is empty or not as shown in Fig. 6. If the words accepted by the product automaton is empty, the system can satisfy the property otherwise the system cannot satisfy the property, and a counter-example can be found.

With SPIN, a translator from PLTL to Never Claim, and a B*ü*chi automaton in terms of PROMELA, are realized to automatically transform a formula in linear temporal logic to Never Claim. To implement model checking PPTL in SPIN, we also provide a translator from PPTL formulas to Never Claims as shown in Fig. 7. We have realized the translator in C++ according to the algorithms presented in this paper. Further, the translator has successfully been integrated within the original SPIN.

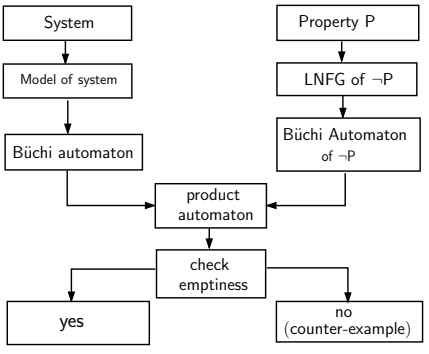


Fig. 6. Model checking PPTL

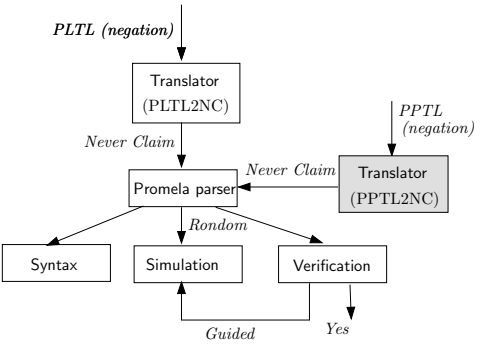


Fig. 7. The model checker SPIN with PPTL

6 Conclusion

An improved decision procedure for checking satisfiability of PPTL formulas is given in this paper. The algorithm has been realized and a model checker based on it has been developed recently. With our experience, the decision procedure and the model checker are useful in practice since full regular properties can be described by PPTL. In the future, we will further investigate the partial order model checking, symbolic model checking, bounded model checking and compositional model checking with PPTL. Furthermore, some supporting tools will also be developed to support our model checking algorithms.

References

1. Kripke, S.A.: Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9, 67–96 (1963)
2. Moszkowski, B.: Reasoning about digital circuits, Ph.D Thesis, Department of Computer Science, Stanford University, TRSTAN-CS-83-970 (1983)

3. Duan, Z.: An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming. PhD thesis, University of Newcastle Upon Tyne (May 1996)
4. Duan, Z., Tian, C., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. Acta Informatica 45(1), 43–78 (2008)
5. Wang, H., Xu, Q.: Temporal logics over infinite intervals. Technical Report 158, UNU/IIST, Macau (1999)
6. Moszkowski, B.C.: A Complete Axiomatization of Interval Temporal Logic with Infinite Time. In: 15th Annual IEEE Symposium on Logic in Computer Science. LICS, p. 241 (2000)
7. Chaochen, Z., Hoare, C.A.R., Ravn, A.P.: A calculus of duration. Information Processing Letters 40(5), 269–275 (1991)
8. Bowman, H., Thompson, S.: A decision procedure and complete axiomatization of interval temporal logic with projection. Journal of logic and Computation 13(2), 195–239 (2003)
9. Dutertre, B.: Complete proof systems for first order interval temporal logic. In: Proceedings of LICS 1995, pp. 36–43 (1995)
10. Clark, M., Gremberg, O., Peled, A.: Model Checking. The MIT Press, Cambridge (2000)
11. Pnueli, A.: The temporal logic of programs. In: Proc. 18th IEEE Symp. Found. of Comp. Sci., pp. 46–57 (1977)
12. Tian, C., Duan, Z.: Propositional Projection Temporal Logic, Büchi Automata and ω-Expressions. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 47–58. Springer, Heidelberg (2008)
13. Tian, C., Duan, Z.: Complexity of Propositional Projection Temporal Logic with Star. Mathematical Structure in Computer Science 19(1), 73–100 (2009)
14. Winskel, G.: The Formal Semantics of Programming Languages. In: Foundations of Computing. The MIT Press, Cambridge
15. Vardi, M.Y.: The Büchi Complementation Saga. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 12–22. Springer, Heidelberg (2007)
16. Katoen, J.-P.: Concepts, Algorithms, and Tools for Model Checking. Lecture Notes of the Course Mechanised Validation of Parrel Systems (1999)
17. 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)