A practical decision procedure for Propositional Projection Temporal Logic with infinite models

Zhenhua Duan, Cong Tian

ICTT and ISN Laboratory, Xidian University, Xi’an, 710071, PR China

ARTICLE INFO
Available online 18 February 2014

Keywords:
Decision procedure
Propositional Projection Temporal Logic
Büchi accepting condition
Automata
Algorithm

This paper presents a practical decision procedure for Propositional Projection Temporal Logic with infinite models. First, a set \( \text{Prop}_R \) of labels \( l_i, 0 \leq i \leq n \in \mathbb{N}_0 \), is used to mark nodes of an LNFG of a formula, and a node with \( l_i \) is treated as an accepting state as in an automaton. Further, the generalized Büchi accepting condition for automata is employed to identify a path (resulting a word) in an LNFG as a model of the formula. In addition, the implementation details of the decision procedure and relevant algorithms including preprocessing, LNFG, circle finding algorithms are presented; as a matter of fact, all algorithms are implemented by C++ programs.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

A decision procedure for Propositional Projection Temporal Logic (PPTL) with infinite models was given in [1], and further improved in [2]. The decision procedure can also be applied to Propositional Interval Temporal Logic (PITL) [5] and Propositional Linear Temporal Logic (PLTL) [9] with minor modification. With this decision procedure, Normal Form (NF), 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. Nevertheless, accepting conditions on LNFG was only simply expressed in an informal way. Also, the decision procedure and the relative algorithms are only written in pseudo code. When implementing the decision procedure, we found that to precisely define accepting conditions is neither a trivial nor an intuitive work. This paper focuses on the practical aspects of the implementation of the decision procedure such as the acceptance condition, data structures, implementation details of the relevant algorithms. The main contributions of the papers are as follows: (1) A set \( \text{Prop}_R \) of labels \( l_i, 0 \leq i \leq n \in \mathbb{N}_0 \), is used to mark nodes of NFG of a formula \( R \), producing LNFG of \( R \), and a node with \( l_i \) is treated as an accepting state as in an automaton. Further, the generalized Büchi accepting condition for an automaton is employed to identify a path (resulting a word) in an LNFG of \( R \) as a model of the formula \( R \). (2) The decision procedure and relevant algorithms are all implemented by C++ programs.

The paper is organized as follows: in the following section, Propositional Projection Temporal Logic is briefly presented. Section 3 reviews the Normal Form (NF) and Normal Form Graph (NFG); a set of labels is introduced to mark the nodes...
of NFG; thus, Labeled NFG (LNFG) with many labels are constructed. In Section 4, accepting conditions with LNFG are
generalized, and an improved and more practical decision procedure is formalized. In Section 5, some implementation
details are explained. Further, examples are given to illustrate how the decision procedure works. Conclusions are drawn in
Section 6.

2. Propositional Project Temporal Logic

Propositional Project Temporal Logic (PPTL) ⁸, ³ is an extension of Propositional ITL (PITL) ⁵ with infinite models
and a new projection construct ⁴. Let Prop be a countable set of atomic propositions and B = {true, false} the boolean
domain. Usually, we use small letters, possibly with subscripts, like p, q, r to denote atomic propositions and capital letters,
possibly with subscripts, like P, Q, R to represent general PPTL formulas. Then the formulas of PPTL are defined by the
following grammar:

\[ P ::= p \mid \neg P \mid P_1 \land P_2 \mid \bigcirc P \mid (P_1, \ldots, P_m) \text{prj} P \mid P^+ \]

where \( p \in \text{Prop} \), \( \bigcirc \) (next), + (chop-pluss) and \( \text{prj} \) (projection) are temporal operators, and \( \neg, \land \) are similar as that in the classical propositional logic.

We define a state \( s \) over Prop to be a mapping from Prop to B, \( s: \text{Prop} \rightarrow B \). We write \( s[p] \) to denote the valuation of \( p \) at state \( s \). An interval \( \sigma = (s_0, s_1, \ldots) \) is a non-empty sequence of states, which can be finite or infinite. The length of \( \sigma \), \( |\sigma| \),
is the number of states in \( \sigma \) minus one if \( \sigma \) is finite; otherwise it is \( \omega \). Let \( N_0 \) denote the set of non-negative integers. To have a uniform notation for both finite and infinite intervals, we will use extended integers as indices, that is \( N_\omega = N_0 \cup \{\omega\} \), and extend the comparison operators, \( =, \prec, \leq \), to \( N_\omega \) by considering \( \omega = \omega \) and for all \( i \in N_0, i < \omega \). Moreover, we write \( \prec \) as \( \leq \). To simplify definitions, we will denote \( s[\sigma] \) by \( (s_0, \ldots, s_{|\sigma|}] \), where \( s[\sigma] \) is undefined if \( \sigma \) is infinite. With such a notation, \( \sigma_{(i,j)} = (0 \leq i \leq j \leq |\sigma|) \) denotes the sub-interval \( (s_i, \ldots, s_j] \).

To formalize the semantics of the projection construct, we need an auxiliary operator \( \downarrow \). Let \( \sigma = (s_0, s_1, \ldots) \) be an interval and \( r_1, \ldots, r_h \) be integers (\( h \geq 1 \)) such that \( 0 \leq r_1 \leq \cdots \leq r_h \leq |\sigma| \). The projection of \( \sigma \) onto \( r_1, \ldots, r_h \) is the projected interval,

\( \sigma \downarrow (r_1, \ldots, r_h) \) def \( (s_{r_1}, s_{r_2}, \ldots, s_{r_h}) \), where \( t_1, \ldots, t_h \) are attained from \( r_1, \ldots, r_h \) by deleting all duplicates. In other words, \( t_1, \ldots, t_h \) is the longest strictly increasing subsequence of \( r_1, \ldots, r_h \). For instance, \( (s_0, s_1, s_2, s_3) \downarrow (0, 2, 2, 3) = (s_0, s_2, s_3) \). The concatenation \( (\cdot) \) of a finite interval \( \sigma = (s_0, s_1, \ldots, s_{|\sigma|}] \) with another interval \( \sigma' = (s_0', s_1', \ldots, s_{|\sigma'|}] \) is represented by \( \sigma \cdot \sigma' = (s_0, s_1, \ldots, s_{|\sigma|}], s_0', s_1', \ldots, s_{|\sigma'|}] \) (not sharing any states).

An interpretation is a tuple \( I = (\sigma, k, j) \), where \( \sigma = (s_0, s_1, \ldots) \) is an interval, \( k \) is a non-negative integer, and \( j \) is an integer or \( \omega \), such that \( 0 \leq k \leq j \leq |\sigma| \). We write \( (\sigma, k, j) \models \) to mean that a formula is interpreted over a subinterval \( \sigma_{(k, j)} \) with the current state being \( s_k \). We utilize \( I^k_{prp} \) to stand for the state interpretation at state \( s_k \). The satisfaction relation \( I \models \) for formulas is given as follows:

\[
I \models p \quad \text{iff } s_k[p] = I^k_{prp}[p] = \text{true}
\]

\[
I \models \neg p \quad \text{iff } I \not\models p
\]

\[
I \models P_1 \land P_2 \quad \text{iff } I \models P_1 \text{ and } I \models P_2
\]

\[
I \models \bigcirc P \quad \text{iff } k < j \text{ and } (\sigma, k + 1, j) \models P
\]

\[
I \models (P_1, \ldots, P_m) \text{prj} P \quad \text{iff there exist integers } r_0, \ldots, r_m, \text{ and } k = r_0 \leq \cdots \leq r_m \leq j
\]

such that \( (\sigma, r_{1-1}, r_1) \models P_1 \) for all \( 1 \leq l \leq m \) and \( (\sigma', 0, |\sigma'|] \models P \) for \( \sigma' \) given by:

\[
(1) \ r_m < j \text{ and } \sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma(r_{m+1}, \ldots)
\]

\[
(2) \ r_m = j \text{ and } \sigma' = \sigma \downarrow (r_0, \ldots, r_n) \text{ for some } 0 \leq h \leq m
\]

\[
I \models P^+ \quad \text{iff there are finitely many integers } r_0, \ldots, r_n \text{ and }
\]

\[
k = r_0 \leq r_1 \leq \cdots \leq r_{n-1} \leq r_n = j \text{ (} n \geq 1 \text{) such that }
\]

\[
(\sigma, r_{1-1}, r_1) \models P \text{ for all } 1 \leq l \leq n; \text{ or } j = \omega \text{ and there are }
\]

infinitely many integers \( k = r_0 \leq r_1 \leq r_2 \leq \cdots \text{ such that }
\]

\[
\lim_{l \to \infty} r_l = \omega \text{ and } (\sigma, r_{1-1}, r_1) \models P \text{ for all } l \geq 1.
\]

For convenience, some derived formulas from elementary PPTL formulas are shown below, which are explained in ³, ⁸. The abbreviations true, false, \( \lor, \rightarrow \text{ and } \leftrightarrow \) are defined as usual.
\[ \varepsilon \equiv \neg \bigcirc \text{true} \quad \text{more} \equiv \neg \varepsilon \]

\[ \Diamond P \equiv (\text{true}, \text{prj} \varepsilon) \quad \Box P \equiv \neg \Diamond \neg P \]

\[ \text{fin}(P) \equiv \Box(\varepsilon \rightarrow P) \quad \text{halt}(P) \equiv \Box(\varepsilon \leftrightarrow P) \]

\[ \text{keep}(P) \equiv \Box(\neg \varepsilon \rightarrow P) \quad \text{rem}(P) \equiv \Box(\text{more} \rightarrow \Box P) \]

\[ P : Q \equiv (P, Q) \text{prj} \varepsilon \quad P ; w Q \equiv (P ; Q) \lor (P \land \text{more}) \]

\[ \text{fin} \equiv \Diamond \varepsilon \quad \text{inf} \equiv \Box \text{more} \]

\[ P^* \equiv p^+ \lor \varepsilon \quad \text{len}(n) \equiv \begin{cases} \varepsilon & \text{if } n = 0 \\ \bigcirc \text{len}(n - 1) & \text{if } n > 1 \end{cases} \]

Usually, \( \models \Box(P \leftrightarrow Q) \) is represented by \( P \equiv Q \) (strong equivalence), meaning that \( P \) and \( Q \) have the same truth value at all states of any models while \( \models \Box(P \rightarrow Q) \) is denoted by \( P \supset Q \) (strong implication), stating that \( P \rightarrow Q \) is true at all states of any models. The following are some useful logic laws. Here \( w \) is a state formula. The proofs of the logic laws can be found in [3].

\[ L_1 \quad \Box(P \land Q) \equiv \Box P \land \Box Q \]

\[ L_2 \quad \Diamond(P \lor Q) \equiv \Diamond P \lor \Diamond Q \]

\[ L_3 \quad \bigcirc(P \lor Q) \equiv \bigcirc P \lor \bigcirc Q \]

\[ L_4 \quad \bigcirc(P \land Q) \equiv \bigcirc P \land \bigcirc Q \]

\[ L_5 \quad R ; (P \land Q) \equiv (R ; P) \lor (R ; Q) \]

\[ L_6 \quad (P \lor Q) ; R \equiv (P ; R) \lor (Q ; R) \]

\[ L_7 \quad \Diamond P \equiv P \lor \bigcirc \Diamond P \]

\[ L_8 \quad \Box P \equiv P \land \bigcirc \Box P \]

\[ L_9 \quad \text{more} \land \neg \bigcirc P \equiv \text{more} \land \bigcirc \neg P \]

\[ L_{10} \quad \neg \bigcirc P \equiv \bigcirc \neg P \]

\[ L_{11} \quad \bigcirc P ; Q \equiv \bigcirc(P ; Q) \]

\[ L_{12} \quad w \land (P ; Q) \equiv (w \land P) ; Q \]

\[ L_{13} \quad Q \text{prj} \varepsilon \equiv Q \]

\[ L_{14} \quad \varepsilon \text{prj} Q \equiv Q \]

\[ L_{15} \quad (P_1, \ldots, P_m) \text{prj} \varepsilon \equiv P_1; \ldots; P_m \]

\[ L_{16} \quad (P, \varepsilon) \text{prj} Q \equiv (P \land \bigcirc \varepsilon) \text{prj} Q \]

\[ L_{17} \quad (P_1, \ldots, w \land \varepsilon, P_1, \ldots, P_m) \text{prj} Q \equiv (P_1, \ldots, w \land P_1, \ldots, P_m) \text{prj} Q \]

\[ L_{18} \quad (P_1, \ldots, P_i \land P'_i, \ldots, P_m) \text{prj} Q \equiv (P_1, \ldots, P_i, \ldots, P_m) \text{prj} Q \lor (P_1, \ldots, P'_i, \ldots, P_m) \text{prj} Q \]

\[ L_{19} \quad (P_1, \ldots, P_m) \text{prj} (P \land Q) \equiv (P_1, \ldots, P_m) \text{prj} P \land (P_1, \ldots, P_m) \text{prj} Q \]

\[ L_{20} \quad (P_1, \ldots, P_m) \text{prj} \bigcirc Q \equiv (P_1 \land \text{more} ; (P_2, \ldots, P_m) \text{prj} Q) \lor (P_1 \land \varepsilon ; (P_2, \ldots, P_m) \text{prj} \bigcirc Q) \]

\[ L_{21} \quad \bigcirc(P_1, \ldots, P_m) \text{prj} \bigcirc Q \equiv \bigcirc(P_1 ; (P_2, \ldots, P_m) \text{prj} Q) \]

\[ L_{22} \quad (w \land P_1, \ldots, P_m) \text{prj} Q \equiv w \land ((P_1, \ldots, P_m) \text{prj} Q) \]

\[ L_{23} \quad (P_1, \ldots, P_m) \text{prj} (w \land Q) \equiv w \land ((P_1, \ldots, P_m) \text{prj} Q) \]
By the derived formulas and logic laws, we can further prove the following conclusions [1,3]:

\[
\begin{align*}
\text{fin}(P) & = P \land \varepsilon \lor \bigcirc \text{fin}(P) & \text{keep}(P) & = \varepsilon \lor P \land \bigcirc \text{keep}(P) \\
\text{halt}(P) & = P \land \varepsilon \lor \neg P \land \bigcirc \text{halt}(P) & \text{rem}(P) & = \varepsilon \lor \bigcirc (P \land \text{rem}(P)) \\
\text{inf} & = \text{true}; w = \text{false} & \text{true} & = \varepsilon \lor \text{true}
\end{align*}
\]

3. Normal Form and Normal Form Graph

3.1 Normal Form (NF) and Normal Form Graph (NFG) are useful in constructing LNFGs of PPTL formulas. The details of these concepts can be found in [1]. In the following, only a brief introduction is given.

**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 as follows.

\[
Q = \bigvee_{j=0}^{n_0} (Q_{ej} \land \varepsilon) \lor \bigvee_{i=0}^{n_1} (Q_{ci} \land \bigcirc Q'_{i})
\]

where \( Q_{ej} = \bigwedge_{k=1}^{m_n} q_{jk} \), \( Q_{ci} = \bigwedge_{h=1}^{m_i} q_{ih} \), \( q_{jk}, q_{ih} \in Q_p \), for any \( r \in Q_p \), \( r \) denotes \( r \lor \neg r \); \( Q'_{i} \) is a PPTL formula without “\( \lor \)" being the main operator. \( \square \)

According to the definition, in a normal form, \( p \land \varepsilon \lor \bigcirc (\bigcirc p \lor q) \) must be written as \( p \land \varepsilon \lor \bigcirc p \land \varepsilon \lor q \) since “\( \lor \)" is the main operator of \( \bigcirc p \lor q \). Implicitly, \( Q'_{i} \) is also not permitted to be the form of \( \neg \bigwedge_{k=2}^{n} P_k \). Further, for convenience, we call \( \bigvee_{j=0}^{n_0} (Q_{ej} \land \varepsilon) \) the terminating part whereas \( \bigvee_{i=0}^{n_1} (Q_{ci} \land \bigcirc Q'_{i}) \) the non-terminating part of the normal form. The reduction process for obtaining a normal form of a formula is known as normal form reduction.

**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,

\[
Q = \bigvee_{j=0}^{n_0} (Q_{ej} \land \varepsilon) \lor \bigvee_{i=0}^{n_1} (Q_{ci} \land \bigcirc Q'_{i})
\]

where like in the normal form, \( Q_{ej} = \bigwedge_{k=1}^{m_n} q_{jk} \), \( Q_{ci} = \bigwedge_{h=1}^{m_i} q_{ih} \), \( q_{jk}, q_{ih} \in Q_p \), for any \( r \in Q_p \), \( r \) denotes \( r \lor \neg r \); further \( \bigvee_i Q_{ci} = \text{true} \) and \( \bigvee_{i \neq j} (Q_{ci} \land Q_{cj}) = \text{false} \); \( Q'_{i} \) is an arbitrary PPTL formula. \( \square \)

Note that a complete normal form may be not a normal form, since “\( \lor \)" is possibly the main operator of \( Q'_{i} \).

For a PPTL formula \( P \), Normal Form Graph (NFG for short) 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. Each node in \( CL(P) \) is specified by a formula in PPTL, while each edge in \( EL(P) \) is a directed arc from a node such as \( Q \) to another node such as \( R \), labeled with a state formula such as \( Q_e \), and identified by a triple, \( (Q, Q_e, R) \). Accordingly, for convenience sometimes, a node in an NFG or Labeled NFG is called a formula. The 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 the set \( EL(P) \) of edges connecting nodes in \( CL(P) \) are inductively defined as follows:

1. Initially, let \( V_0 = CL(P) = EL(P) = \emptyset \);
2. Let \( P = \bigvee_i P_i \). For each \( i \in P \), \( P_i \in CL(P) \);
3. For all \( Q \in CL(P) \setminus \{\varepsilon, \text{false}\} \), if \( Q \) is rewritten into its normal form \( \bigvee_{j=0}^{n_0} (Q_{ej} \land \varepsilon) \lor \bigvee_{i=0}^{n_1} (Q_{ci} \land \bigcirc Q'_{i}) \), then \( \in CL(P) \), \( (Q, Q_{ej}, \varepsilon) \in EL(P) \) 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 \( G = (CL(P), EL(P), V_0) \).

In an NFG, any root node in \( V_0 \) is denoted by a circle with an incoming edge without a source, \( \varepsilon \) 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, \( \pi = \langle n_0, e_0, n_1, e_1, \ldots, \varepsilon \rangle \) from a root node to the \( \varepsilon \) node, while an infinite path is an infinite alternating sequence of nodes and edges, \( \pi = \langle n_0, e_0, n_1, e_1, \ldots, n_i, e_i, n_j, e_j, n_k, e_k, \ldots \rangle \) departing from the root node with some nodes, e.g. \( n_i, \ldots, n_j \), occurring for infinitely many times. For convenience, we use \( \text{inf}(\pi) \) to denote the set of nodes which infinitely often occur.
in the infinite path \( \pi \). In some circumstances, in a path of NFG of formula \( Q \), a node \( n_i \) can be replaced by a formula \( Q_j \in CL(Q) \) 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 a formula \( Q \) and paths of NFG \( G \) of \( Q \), functions \( P2M(\pi, G) \) and \( M2P(\sigma, G) \) are formally defined below. Let \( \Pi_{pptl} \) be the set of all ppt formulas, \( \Delta \) the set of NFGs of all formulas in \( \Pi_{pptl} \), \( \Sigma_G \) the set of all paths in an NFG \( G \), and \( \Gamma \) the set of all intervals. Given a path \( \pi = \langle Q, Q_{0e}, Q_1, Q_1e, \ldots \rangle \) in an NFG \( G \), an interval \( \sigma_\pi \) can be obtained by function \( P2M : \Sigma_G \times \Delta \longrightarrow \Gamma \) defined as follows.

\[
\sigma_\pi = P2M(\pi, G) = \begin{cases} 
\{ (s_0, s_1, \ldots, s_n), \text{ for } 1 \leq i \leq n \} & \text{if } \pi = \langle Q, Q_{0e}, Q_1, Q_1e, \ldots, Q_{n}, \epsilon \rangle \text{ is a finite path} \\
\{ (s_0, s_1, \ldots, (s_j), \epsilon) \}, \text{ for } 1 \leq k \leq j & \text{if } \pi = \langle Q, Q_{0e}, Q_1, Q_1e, \ldots, Q_{j}, \epsilon \rangle \text{ is an infinite path} 
\end{cases}
\]

Correspondingly, for a model \( \sigma = \langle s_0, s_1, \ldots \rangle \models Q \), a path \( \pi_\sigma \) w.r.t. the NFG \( G \) of \( Q \) can be obtained by function \( M2P : \Gamma \times \Delta \longrightarrow \Sigma_G \) defined as follows:

\[
\pi_\sigma = M2P(\sigma, G) = \begin{cases} 
\{ (Q, Q_{0e}, Q_1, Q_1e, \ldots, Q_{n}, \epsilon), \text{ for } 1 \leq i \leq n \} & \text{if } \pi = \langle Q, Q_{0e}, Q_1, Q_1e, \ldots, Q_{n}, \epsilon \rangle \text{ is a finite path} \\
\{ (Q, Q_{0e}, Q_1, Q_1e, \ldots, (Q_i, Q_{ie}, \ldots, Q_{j}, Q_{je})_{\epsilon}) \}, \text{ for } 1 \leq h \leq j & \text{if } \pi = \langle Q, Q_{0e}, Q_1, Q_1e, \ldots, (Q_i, Q_{ie}, \ldots, Q_{j}, Q_{je})_{\epsilon} \rangle \text{ is an infinite interval} 
\end{cases}
\]

Although in the above an interval \( \sigma_\pi \) can be defined for a given path \( \pi \) of the NFG of formula \( Q \), whether or not \( \sigma_\pi \models Q \) needs to be proved (see Lemmas 2, 6 and 9). Similarly, given a model \( \sigma \) of formula \( Q \), \( \sigma \models Q \), although a path \( \pi_\sigma \) can be constructed, however, whether or not the path can be found in \( G \) also needs to be proved (see Lemmas 3 and 5). Before the proofs of theorems and lemmas, we need the following notations: given an interval \( \sigma = \langle s_0, s_1, \ldots \rangle \), the \( i \)th prefix of \( \sigma \) denoted by \( \sigma^i \) is the subsequence \( \langle s_0, \ldots, s_i \rangle \) while the \( i \)th suffix of \( \sigma \) denoted by \( \sigma^{(i)} \) is the subsequence \( \langle s_j, \ldots, s_n \rangle \). In a similar way, for a given path \( \pi \), we can define the \( i \)th prefix \( \pi^i \) and the \( i \)th suffix \( \pi^{(i)} \) of \( \pi \).

**Theorem 1.** Finite paths in the NFG of PPTL formula \( Q \) precisely characterize finite models of \( Q \).

**Proof.** It is a consequence of Lemmas 2 and 3. \( \Box \)

**Lemma 2.** For a finite path \( \pi \) in the NFG \( G \) of \( Q \), \( \sigma_\pi \models Q \).

**Proof.** Let \( \pi = \langle Q, Q_{0e}, Q_1, Q_1e, \ldots, Q_{n-1}, Q_{n-1}e, Q_n, Q_{ne}, \epsilon \rangle \) be a finite path of the NFG \( G \) of \( Q \), \( \sigma_\pi = \langle s_0, s_1, \ldots, s_{n-1}, s_n \rangle (n \geq 0) \) can be obtained by function \( P2M(\pi, G) \), where for each \( i \in N_0 \), \( s_i[q] = true \) if proposition \( q \) in \( Q_{ie} \), otherwise \( s_i[q] = false \) if \( q \) is not in \( Q_{ie} \). We need to prove \( \sigma_\pi \) is a model of formula \( Q \).

Case 1: \( n = 0 \), the path is \( \pi = \langle Q, Q_{0e}, \epsilon \rangle \) and the interval is \( \sigma_\pi = \langle s_0 \rangle \). By the construction of \( s_0 \), \( \langle s_0 \rangle \models Q_{0e} \). Thus, \( \sigma_\pi \) is a model of \( Q \).

Case 2: \( n > 0 \), in this case, the proof proceeds by induction on the length of the prefix \( \sigma^i_\pi \) of \( \sigma_\pi \).

**Base:** \( \sigma^0_\pi = \langle s_0 \rangle \). By the construction of \( s_0 \), \( \langle s_0 \rangle \models Q_{0e} \). That is \( \sigma^0_\pi \) is a prefix of a model of \( Q \).

**Induction:** Suppose for all \( j < n \), prefix \( \sigma^j_\pi \) of \( \sigma_\pi \) is a prefix of a model. We prove \( \sigma^n_\pi \) is a prefix of a model of \( Q \). By hypothesis, \( \sigma^{n-1}_\pi = \langle s_0, \ldots, s_{n-1} \rangle \) is a prefix of the model. According to Algorithm NFG, there must be \( Q_{n} \equiv Q_{ne} \wedge \epsilon \) such that a new node \( \epsilon \) and a new edge \( \langle Q_n, Q_{ne}, \epsilon \rangle \) are added to the NFG of \( Q \). By the construction of \( s_n \), \( \langle s_n \rangle \models Q_{ne} \wedge \epsilon \). Since \( \sigma^{n-1}_\pi \) is a prefix of the model, so \( \sigma^n_\pi = \sigma_\pi \) is a model of \( Q \). \( \Box \)

**Lemma 3.** For a finite interval \( \sigma \models Q \), \( \pi_\sigma \) can be found in the NFG \( G \) of \( Q \).

**Proof.** For a finite interval \( \sigma = \langle s_0, s_1, \ldots, s_{n-1}, s_n \rangle \), \( \sigma = \sigma[0, |\sigma|] \models Q \), a path \( \pi_\sigma = \langle Q, Q_0, Q_1, Q_1e, \ldots, Q_{n-1}, Q_{n-1}, Q_n, Q_{ne}, \epsilon \rangle \) can be obtained by function \( M2P(\sigma, G) \), where \( Q_i = \bigcup \emptyset q_k, q_k = q_k \) if \( s_i[q_k] = true \), and \( q_k = \neg q_k \) if \( s_i[q_k] = false \) for each \( i, 0 \leq i \leq n \). We need to prove that \( \pi_\sigma \) can be found in the NFG of \( Q \).
Case 1: \( n = 0 \), the model \( \sigma = (s_0) \), and the path \( \pi_{\sigma} = (Q, Q_0, e) \). By the construction of NFG, there must exist a normal form of \( Q \), \( Q \equiv (Q_0 \land e) \lor \bigvee_{j=0}^{k}(Q_0j \land Q_{0j}) \) such that there exists an edge from root \( Q \) to node \( e \) and labeled by \( Q_0 \) in \( G \), and \( (\sigma, 0, [\sigma]) \equiv (Q_0 \land e) \).

Case 2: \( n > 0 \), in this case, the proof proceeds by induction on the length of the prefix \( \pi_{\sigma}^i \) of \( \pi_{\sigma} \).

**Base:** \( \pi_{\sigma}^0 \equiv (Q, Q_0, Q_1^i) \). By the construction of NFG, there must exist a normal form of \( Q \), \( Q \equiv (Q_0e \land e) \lor \bigvee_{j=0}^{k}(Q_0j \land Q_{0j}) \) such that there exists an edge from root \( Q \) to node \( Q_0 \) and labeled by \( Q_0 \) in \( G \), and \( (\sigma, 0, [\sigma]) \equiv (Q_0h \land Q_{0h}) \). Here \( Q_0h \equiv \land \hat{q}_k. \hat{q}_k \equiv q_k \), if \( s_i[q_k] \) is true, and \( \hat{q}_k \equiv \neg q_k \) if \( s_i[q_k] \) is false for each \( i, 0 \leq i \leq n \). So, prefix \( \pi_{\sigma}^0 \) of \( \pi_{\sigma} \) has been found in \( G \).

**Induction:** Suppose we have found a prefix \( \pi_{\sigma}^{n-1} \equiv (Q, Q_0, Q_1, \ldots, Q_{n-1}, Q_n) \) of \( \pi_{\sigma} \) in \( G \) w.r.t. the prefix \( \pi_{\sigma}^{n-1} \equiv (s_0, \ldots, s_{n-1}) \) of \( \pi_{\sigma} \). At this point, we can rewrite \( Q_n \) to its normal form, \( Q_n \equiv (Q_{ne} \land e) \lor \bigvee_{j=0}^{k}(Q_{nj} \land Q_{0j}) \), such that there exists an edge from \( Q_n \) to \( e \) and labeled by \( Q_n \equiv Q_{ne} \) in \( G \) and \( (\sigma, n, [\sigma]) \equiv Q_{ne} \land e \). Here \( Q_{ne} \equiv \land r_i \), and \( r_i \) is an atomic proposition \( r_i \) or \( \neg r_i \). Thus, \( (s_n) \equiv Q_{ne} \land e \). That is, \( s_n[r_i] = true \) if \( r_i \) is \( r_i \), and \( s_n[r_i] = false \) if \( r_i \) is \( \neg r_i \). So we have found a path \( \pi_{\sigma} = (Q, Q_0, Q_1, \ldots, Q_{n-1}, Q_n, Q_n, e) \) in \( G \) w.r.t. the model \( \sigma \).

For infinite models, the relationship between models of a formula and paths of the NFG of the formula 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 \( R = P \land Q \) and \( P = P_1 \lor P_2 \), where \( P_1, P_2 \), and \( Q \) are any PPTL formulas. Further, \( P_1 \lor P_2 \) 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

\[ R_{c} ::= R \lor Q \lor R_{c} \land R \]

where \( P, Q \), and \( R \) are any PPTL formulas.

For instance, \((\bigcirc p \land q), (p \lor p'), (\bigcirc p \land q) \land r)\) are chop formulas while \((\bigcirc p \land q), \neg (\bigcirc p \land q) \land p'\) are not, where \( p, q \), and \( r \) are atomic propositions. Note that \( P ; Q \) is a chop formula but \( \neg (P ; Q) \) is not. This will be formally analyzed later.

For chop construct \( P ; Q \), an infinite model \( \sigma = (s_0, s_1, \ldots, s_k, \ldots) \models P ; Q \) 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 an infinite model \( \sigma = (s_0, s_1, \ldots, s_k, \ldots) \models P \) but there are no finite prefixes \( \sigma^i \models P \) (\( i \in N_0 \)), it fails to satisfy \( P ; Q \). Note that in the weak version of the chop construct, \( P ; Q \) is still satisfied in the case. In some contexts, for convenience, we say this type of requirement for \( P \) is the finiteness (or terminating) property of \( P \). Formally, the finiteness of \( P \), called FSC-Property, is a strong chop construct \( P ; Q \), means that \( P ; Q \) can be reduced to some formula \( P_1 \land e \); \( Q \) by means of repeatedly using normal form reduction and \( P_1 \land Q \) is satisfiable. More precisely, FSC-Property of \( P \) in \( P ; Q \) is satisfiable over an infinite model \( \sigma \) if function \( fsc' (\sigma, P ; Q) = true \). Formally, function \( fsc' \) is defined as \( fsc' : \Gamma \times \Pi_{ppl} \rightarrow \{true, false\} \), where \( \Pi_{ppl} \) is the set of all PPTL formulas, and \( fsc' (\sigma, P ; Q) = true \) 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 \( Q \). So, FSC-Property of \( P \) needs to be considered if it is 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,

\[
P ; Q \equiv \left( \bigcup_{j=0}^{n_0} P_{ej} \land e \lor \bigcup_{i=0}^{n_1} (P_{ci} \land \bigcirc P_{i}^j) \right) ; Q
\]

\[
= \bigcup_{j=0}^{n_0} \left( P_{ej} \land e \lor Q \lor \bigcup_{i=0}^{n_2} (P_{ci} \land \bigcirc P_{i}^j) \right)
\]

Subsequently, the new generated formula \( P_{i}^j ; Q \) needs to be repeatedly transformed into its normal form in the same way. Whenever a final state of \( P \) is reached, i.e. \( P_{e} \land e \lor Q \) is encountered, where \( P_{e} \) is a state formula, FSC-Property of \( P ; Q \) is satisfied over the path \( \pi \) departing from the root node.

To formally define FSC-Property of \( P ; Q \) over a path \( \pi \) by means of function \( fsc' \) we could transfer \( \pi \) into a model \( \sigma_{\pi} \) by function \( P2M \). However, for clarity and simplicity, we here formally define functions \( FSC \) and \( fsc \). Let \( \pi = (R, R_{0e}, R_1, R_{1e}, \ldots) \) be an infinite path in the NFG \( G \) of a formula \( R \), and \( \pi^{(k)} = (R_k, R_{ke}, \ldots) \) be the \( k \)th suffix of \( \pi \). Whether or not FSC-Property of \( R \) is satisfiable over \( \pi \), denoted by function \( FSC(\pi, R) \), FSC (also \( fsc \)) : \( \Sigma \times \Pi_{ppl} \rightarrow \{true, false\} \), can be defined based on \( fsc(\pi, R) \) as follows, where \( \Pi_{ppl} \) stands for all PPTL formulas while \( \Sigma \) denotes all paths in NFGs of all formulas of \( \Pi_{ppl} \).
Accordingly, 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 \(R_i \equiv P_{ie} \land e; Q\), or
3. \(R \equiv P \land Q\) and \(\text{fsc}(\pi, P) = \text{true}\) and \(\text{fsc}(\pi, Q) = \text{true}\), or
4. \(R \equiv P \lor Q\) and \(\text{fsc}(\pi, P) = \text{true}\) or \(\text{fsc}(\pi, Q) = \text{true}\).

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

In particular, notice that \(\text{fsc}(\pi, \neg(P; Q)) = \text{true}\) since \(\neg(P; Q)\) is not a chop formula.

The functions \(\text{FSC}\) and \(\text{fsc}\) facilitate us to check whether or not a node which is a chop formula in particular involved in a circle in a path of NFG of a formula satisfies FSC_property. As a matter of fact, when a chop formula \(Q_i\) is involved in a circle in a path \(\pi\) in NFG \(G\) of a formula \(Q\) the FSC_property of \(Q_i\) can be checked by means of \(\text{fsc}(\pi, G)\) and the whole path, i.e. all nodes of the path, can be checked by means of \(\text{FSC}(\pi, G)\). Although for any chop formula \(X; Y\) involved in a circle we consider its FSC_property, however, for the negation of a chop formula like \(\neg(X; Y)\) involved in a circle in a path we do not need to consider its finiteness property since an infinite path containing \(\neg X\) could generate an infinite model of \(\neg(X; Y)\) (see Lemma 4). This case can happen when we construct the NFG of a formula such as \(\neg(P; Q; R)\).

**Lemma 4.** Let \(\pi\) be an infinite path in the NFG \(G\) of a formula \(Q\) with \(\text{FSC}(\pi, Q) = \text{true}\), and \(R \equiv \neg(X; Y)\) be the \(i\)th node in \(\pi\). For the infinite model \(\sigma_\pi \equiv \text{P2M}(\pi, G)\), \(\sigma_\pi^{(i)} \models \neg(X; Y)\) if and only if \(\forall k(i \leq k \in N_0 \rightarrow (\sigma_\pi^k \models \neg X \lor \sigma_\pi^{(k)} \models \neg Y))\).

**Proof.** Since \(\sigma_\pi\) be an infinite model, and

\[
\sigma_\pi^{(i)} \models X; Y \iff \exists k \leq k \in N_0 \text{ and } \sigma_\pi^k \models X \text{ and } \sigma_\pi^{(k)} \models Y
\]

We have,

\[
\sigma_\pi^{(i)} \models \neg(X; Y)
\]

\[
\iff \neg(\sigma_\pi^{(i)} \models (X; Y))
\]

\[
\iff \neg(\exists k \leq k \in N_0 \land \sigma_\pi^k \models X \land \sigma_\pi^{(k)} \models Y)
\]

\[
\iff \forall k(\neg(i \leq k \in N_0) \lor \sigma_\pi^k \models \neg X \lor \sigma_\pi^{(k)} \models \neg Y)
\]

\[
\iff \forall k(i \leq k \in N_0 \rightarrow (\sigma_\pi^k \models \neg X \lor \sigma_\pi^{(k)} \models \neg Y)) \quad \square
\]

Note that, in Lemma 4, if \(k = \omega\) the implication of the right hand side is trivially satisfied. This means that an infinite model \(\sigma^{(\omega)} \models \neg X\) might hold.

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

\[
P_1 \equiv P_{1e} \land e \lor \bigvee_{i=1}^r (P_{1i} \land \bigcirc P'_{1i})
\]

\[
P_2 \equiv P_{2e} \land e \lor \bigvee_{l=1}^s (P_{2l} \land \bigcirc P'_{2l})
\]

\[
Q \equiv Q_e \land e \lor \bigvee_{j=1}^k (Q_j \land \bigcirc Q'_j)
\]

then,

\[
(P_1, P_2) \text{prj } Q \equiv Q_e \land P_{1e} \land P_{2e} \land e \lor \bigvee_{i=1}^s (Q_e \land P_{1e} \land P_{2l} \land \bigcirc P'_{2l}) \lor \bigvee_{j=1}^k (P_{1e} \land P_{2e} \land Q_j \land \bigcirc Q'_j)
\]

\[
\Rightarrow \bigvee_{i=1}^r (P_{1i} \land P_{2l} \land \bigcirc P'_{2l}) \lor \bigvee_{j=1}^k (P_{1i} \land P_{2e} \land Q_j \land \bigcirc Q'_j)
\]
The above explanation shows that the finiteness property for projection constructs and $\lnot(P;Q)$ needs not to be considered, and only chop formulas require 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 $\langle A, \sqsubseteq \rangle$ has a unique least fixed point $\bigcup_i F^i(\bot)$ and a unique greatest fixed point $\bigcap_i F^i(\top)$ [7].

**Scott’s fixed-point induction.** Suppose $D$ is a complete lattice with bottom $\bot$, $F : D \to D$ a continuous function, and $B$ an inclusion subset of $D$. If $\bot \in B$ and $\forall x \in D. x \in B \Rightarrow F(x) \in B$, then $\text{fix}(F) \in B$ [7].

The inclusion subset is defined as follows:

**Definition 5 (Inclusion subset).** Let $D$ be a complete partial order. A subset $B$ of $D$ is inclusive iff for all $\omega$-chains $d_0 \sqsubseteq d_1 \cdots \sqsubseteq d_n \sqsubseteq \cdots$ in $D$ if $d_n \in B$ for all $n \in N_0$ then $\bigcup_{n \in N_0} d_n \in B$.

We now turn to some prove to turn out the relationship between paths in the NFG and models of a formula $Q$.

**Lemma 5.** For an infinite model $\sigma$ and NFG $G$ of a formula $Q$, if $\pi_\sigma = M2P(\sigma, G)$ then $\pi_\sigma$ with FSC($\pi_\sigma, Q$) = true can be found in $G$.

**Proof.** Let $\sigma = (s_0, s_1, \ldots)$ and $\pi_\sigma = (Q, Q_0e, Q_1, Q_1e, \ldots)$. We first prove that $\pi_\sigma$ can be found in $G$. By the construction of $\pi_\sigma$, for $(\sigma, 0, |\sigma|) \models Q$, there must exist a normal form of $Q$, $Q_0 = (Q_0e \land \epsilon) \lor \bigvee_{i=0}^k (Q_0j \land \square Q_0j')$ such that there exists an edge from root $Q$ to node $Q_1$ and labeled by $Q_0e = Q_0$ for some $0 \leq t \leq k$ in $G$ and $(\sigma, 0, |\sigma|) \models Q_0e \land \square Q_0j$. Here $Q_0 = \bigwedge_i \neg q_i$, and $q_i$ is an atomic proposition $q_i$ or $\neg q_i$. Thus, $(s_0) \models Q_0e$, and $(\sigma, 1, |\sigma|) \models Q_1$. That is, $s_0[q_i] = \text{true}$ if $q_i$ is $q_i$, and $s_0[q_i] = \text{false}$ if $q_i$ is $\neg q_i$. So we have found a prefix $\pi_0 = (Q_0e, Q_1, \ldots)$ of $\pi_\sigma$ w.r.t. the prefix $(s_0, \ldots, s_0)$ of model $\sigma$ such that, for $0 \leq k \leq i$, $(\sigma, k, |\sigma|) \models Q_0e \land \square Q_{1+k}$; here $Q_0e = \bigwedge_j p_{kj}$, and $p_{kj}$ is an atomic proposition $p_{kj}$ or $\neg p_{kj}$, and $(\langle s_0 \rangle) \models Q_0e$, and $(\sigma, k+1, |\sigma|) \models Q_{1+k}$, and $s_0[p_{kj}] = \text{true}$ if $p_{kj}$ is $p_{kj}$, and $s_0[p_{kj}] = \text{false}$ if $p_{kj}$ is $\neg p_{kj}$.

At this point, we can rewrite $Q_{i+1}$ to its normal form, $Q_{i+1} = (Q_{(i+1)e} \land \epsilon) \lor \bigvee_{j=0}^k (Q_{(i+1)j} \land \square Q_{(i+1)j})$, such that there exists an edge from $Q_{i+1}$ to $Q_{(i+2)e}$ and labeled by $Q_{(i+1)e} = Q_{(i+1)e}$ in $G$ for some $0 \leq t \leq k$ and $(\sigma, i+1, |\sigma|) \models Q_{(i+1)e} \land \square Q_{(i+2)e}$. Here $Q_{i+1} = \bigwedge_i \neg r_i$, and $r_i$ is an atomic proposition $r_i$ or $\neg r_i$. Thus, $(s_1, \ldots, s_1) \models Q_{(i+1)e}$, and $(\sigma, (i+2), |\sigma|) \models Q_{(i+2)}$. That is, $s_{i+1}[r_i] = \text{true}$ if $r_i$ is $r_i$, and $s_{i+1}[r_i] = \text{false}$ if $r_i$ is $\neg r_i$. So we have found a prefix $\pi_{i+1} = (Q, Q_{0e}, Q_1, Q_1e, \ldots)$ of $\pi_\sigma$ w.r.t. the prefix $(s_0, \ldots, s_0)$ of model $\sigma$. Now we need to prove $\pi_\sigma = (Q, Q_{0e}, Q_1, Q_1e, Q_2, \ldots, Q_{l_1}, Q_{l_1e}, \ldots)$ is an infinite path in the NFG $G$ of $Q$ w.r.t. $\sigma$. To do so, we define, $\pi_{0} = (Q, Q_0e, Q_1, \ldots, Q_{l_1}, Q_{l_1e}, \ldots)$ is an infinite path in the NFG $G$ of $Q$ w.r.t. $\sigma$. Then we define a set $D = \{\pi_{0}^{-1}, \pi_{0}^0, \ldots, \pi_{0}^{l_1}, \ldots\} \cup \{\pi_{\sigma}\}$. It is readily to prove that $(D, \sqsubseteq)$ is a complete partial order set with bottom $\pi_{0}^{-1} = \epsilon$, so $(D, \subseteq)$ is a complete lattice. We further define a function $R$ over $D$ as follows:

$$R(\pi_k) = \pi_k^{i+1}, \text{ for } k \in \{-1\} \cup N_0$$

It is easily to prove that, for $\pi_{0}^i, \pi_{0}^j \in D$, $\pi_{0}^i \sqsubseteq \pi_{0}^j$ iff $i \leq j$. Furthermore,

$$R(\pi_{\sigma}^i) = \pi_{\sigma}^{i+1} \quad R(\pi_{\sigma}^j) = \pi_{\sigma}^{j+1}$$

Since $i \leq j$, so $i+1 \leq j+1$. Thus, we have $\pi_{\sigma}^{i+1} \sqsubseteq \pi_{\sigma}^{j+1}$. Hence $R$ is monotonic. By Tarsky’s fixed point theorem,

$$\text{fix}(R) = \bigcup_i R^i(\pi_{0}^{-1}) = \pi_{\sigma}$$

Further, we have
\[
\mathcal{R}\left( \bigcup_{i \in N_0} \pi^i_\sigma \right) = \pi_\sigma = \bigcup_{i \in N_0} \mathcal{R}(\pi^i_\sigma)
\]

So, \( \mathcal{R} \) is also continuous. We can now construct a subset \( B \) of \( D \) as follows:

\[
B = \{ \pi^i_\sigma \mid \pi^i_\sigma \in D \text{ and } \pi^i_\sigma \text{ is the ith prefix of a path } \pi_1 \text{ of } G \}
\]

In the previous step, we have shown that, for \( i \in N_0 \), ith prefix \( \pi^i_\sigma \) of \( \pi_\sigma \) is ith prefix of a path in \( G \). Hence, we can reasonably assume that this path in \( G \) is \( \pi_1 \). So, for any \( \omega \)-chain \( \pi^0_\sigma \subseteq \pi^1_\sigma \subseteq \cdots \subseteq \pi^i_\sigma \subseteq \cdots \) in \( D \), this \( \omega \)-chain is in fact in \( B \) since \( \pi^i_\sigma \in B \). Moreover, we have \( \bigcup_{i \in N_0} \pi^i_\sigma = \pi_\sigma \in B \), namely, \( \bigcup_{i \in N_0} \pi^i_\sigma \in B \). Therefore, \( B \) is an inclusion subset of \( D \).

Since \( \pi^i_\sigma = \pi^i_1 \) for \( i \in N_0 \), \( B = \{ \pi \} \cup \{ \pi^i_1 \} \) is the ith prefix of \( \pi_1 \cup \{ \pi_\sigma \} = D \). Further, \( F \) is a monotonic and continuous function over \( D \). Hence, \( F \) is also a monotonic and continuous function over \( B \) with \( F(\pi^i_1) = \pi^{i+1}_1 \) for \( i \in N_0 \). Now by using Tarsky's theorem, we obtain fix(\( F \)) = \( \pi_1 \). Further, \( \epsilon \in B \), by using Scott's Fix-Point Induction, fix(\( F \)) = \( \pi_1 \in B \). However, \( B \) has only one element \( \pi_\sigma \) with infinite length. It turns out that \( \pi_\sigma = \pi_1 \) is an infinite path of \( G \).

Now we prove FSC(\( \pi_\sigma, Q \)) = true. This can equivalently be proved in terms of fsc(\( \pi^i_\sigma, Q_i \)) = true, for all \( i \geq 0 \). Let \( Q_i \) be an arbitrary node in \( \pi_\sigma \). We need only to prove that for any chop component \( P \) appearing in \( Q_i \), \( \pi(i) \models P \); \( V \). That is, \( \exists \ j \in N_0, j \geq i, \pi(i, j) \models P \) and \( \pi(j, i) \models V \). As the reduction proceeds, a final state of \( P \) can be reached, i.e., \( P\pi(i) \land \epsilon \models V \) can be encountered, where \( P\pi(i) \) is a state formula, FSC Property of \( P \); \( V \) can be satisfied over the path \( \pi_\sigma \). Hence, at state \( s_j \) over path \( \pi_\sigma \) \( Q_j \models P(\epsilon(i,j) \land \epsilon \models V \). By Definition 4, fsc(\( \pi^i_\sigma, Q_i \)) = true. It turns out that FSC(\( \pi_\sigma, Q \) = true.

**Lemma 6.** For an infinite path \( \pi \) in the NFG \( G \) of \( Q \), if there exist no chop formulas over \( \pi, \sigma_\pi = \{ \pi \} \cup \{ \sigma_i \} \) = true.

**Proof.** Let \( \pi = (Q, Q_0, Q_1, Q_2, \ldots, Q_i, Q_{i+1}) \) be an infinite path in the NFG \( G \) of \( Q \). Then we can construct an interval \( \sigma_\pi = (s_0, s_1, s_2, \ldots) \) by function \( P2M(\pi, G) \). For each \( i \in N_0 \), if proposition \( q \) in \( Q_i \) then \( s_i[q] = \text{true} \), otherwise if \( \neg q \) in \( Q_i \) then \( s_i[q] = \text{false} \). We need to prove \( \sigma_\pi \) is a model of formula \( Q \). To do so, we need first to prove that a prefix \( \sigma^i_\pi \) of \( \sigma_\pi \) is a prefix of a model. The proof proceeds by induction on the length of the prefix of the interval.

**Base:** \( \sigma^0_\pi = (s_0) \). By construction of \( s_0 \). \( (s_0) \models Q_0 \).

**Induction:** Suppose for all \( j < i \), prefix \( \sigma^j_\pi \) of \( \sigma^i_\pi \) is a prefix of a model. We prove \( \sigma^j_\pi \) is also a prefix of the model. By hypothesis, \( \sigma^{j-1}_\pi \) is a prefix of the model. According to the construction of NFG, there must be \( Q_{i-1} \models Q_i \land \pi \) in \( Q_{i+1} \) such that a new node \( Q_i \), and a new edge \( (Q_{i-1}, Q_i, Q_j) \) are added to \( G \). By the construction of \( s_i, s_i[q] = Q_i \). Since \( \sigma^{i-1}_\pi \) is a prefix of the model, \( \sigma^{i-1}_\pi \) is also a prefix of the model. We now need to prove that \( \sigma^{i}_\pi \) is a model of \( Q \).

Let \( \sigma^{i-1}_\pi \models \epsilon \) and \( \sigma^i_\pi = (s_0, s_1, s_2, \ldots, s_i) \) for \( i \in N_0 \). Then we define a set \( A = \{ \sigma^{i-1}_\pi \} \cup \{ \sigma_\pi \} \). It is readily to prove that \( (A, \subseteq) \) is a complete partial order set with bottom \( \sigma^{i-1}_\pi \models \epsilon \), so \( (A, \subseteq) \) is a complete lattice. We define a function \( \mathcal{R} \) over \( A \) as follows:

\[
\mathcal{R}(\sigma^k_\pi) = \sigma^{k+1}_\pi, \quad k \geq -1
\]

So, for all \( \sigma^i_\pi \subseteq \sigma^j_\pi \subseteq \sigma^k_\pi \), and

\[
\mathcal{R}(\sigma^i_\pi) = \sigma^{i+1}_\pi \quad \mathcal{R}(\sigma^j_\pi) = \sigma^{j+1}_\pi
\]

Since \( i \leq j \), so \( i + 1 \leq j + 1 \). Then we have \( \sigma^{i+1}_\pi \subseteq \sigma^{j+1}_\pi \). So, \( \mathcal{R} \) is monotonic. Thus, by fixed point theorem,

\[
\text{fix}(\mathcal{R}) = \bigcup \mathcal{R}(\sigma^{i-1}_\pi) = \sigma_\pi
\]

Further, we have

\[
\mathcal{R} \left( \bigcup_{i \in N_0} \sigma^i_\pi \right) = \bigcup_{i \in N_0} \mathcal{R}(\sigma^i_\pi)
\]

So \( \mathcal{R} \) is also continuous. We can now construct a subset \( B \) of \( A \) as follows:

\[
B = \{ \epsilon \} \cup \{ \sigma^i_\pi \mid \sigma^i_\pi \in A \text{ and } \sigma^i_\pi \text{ is the ith prefix of a model } \sigma_0 \text{ of } Q \}
\]

In the previous step, we have shown that the ith prefix \( \sigma^i_\pi \) of \( \sigma_\pi \) is the ith prefix of a model of \( Q \). Hence, we can assume that this model is \( \sigma_1 \). Further, for any \( \omega \)-chain \( \sigma^0_\pi \subseteq \sigma^1_\pi \subseteq \cdots \subseteq \sigma^j_\pi \subseteq \cdots \) in \( A \), this \( \omega \)-chain is in fact in \( B \) since \( \sigma^j_\pi \in B \). Moreover, we have \( \bigcup_{i \in N_0} \sigma^i_\pi = \sigma_\pi \), namely, \( \bigcup_{i \in N_0} \sigma^i_\pi \in B \). Therefore, \( B \) is an inclusion subset of \( A \).

Since \( \sigma^i_\pi = \sigma_1 \) for \( i \in N_0 \), so \( B = \{ \epsilon \} \cup \{ \sigma^i_\pi \} \) is the ith prefix of \( \sigma_1 \) and \( \{ \sigma_\pi \} = A \). Further, \( F \) is a monotonic and continuous function over \( A \). Hence, \( F \) is also a monotonic and continuous function over \( B \) with \( F(\sigma^i_\pi) = \sigma^{i+1}_\pi \) for \( i \in N_0 \). Now by
using Tarsky’s theorem, we obtain $\text{fix}(F) = \sigma_1$. Further, $\epsilon \in B$, by using Scott’s Fix-Point Induction, $\text{fix}(F) = \sigma_1 \in B$. However, $B$ has only one element $\sigma_\pi$ with infinite length. It turns out that $\sigma_\pi = \sigma_1$ is an infinite model of $Q$. \hfill $\Box$

In the following, we further discuss the general case of path $\pi$ for a given formula $R$. The final conclusion is presented in Theorem 10.

**Lemma 7.** For an infinite path $\pi = \langle R, R_{e0}, R_1, R_{e1}, R_2, \ldots \rangle$ in the NFG $G$ of formula $R$, if $\sigma_\pi^{(i)} \models R$, then $\sigma_\pi^{(i-1)} \models R_{i-1}$.

**Proof.** By function $P2M(\pi, G)$, we can obtain an interval $\sigma_\pi = \langle s_0, s_1, \ldots \rangle$. Thus, for $i > 0$, $(s_{i-1}, s_i) \models R_{e_{i-1}} \land \text{skip}$, and $\sigma_\pi^{(i-1)} = \langle s_{i-1}, s_i \rangle \circ \sigma_i \models R_{e_{i-1}} \land \text{skip} \land R_i$. So, $\sigma_\pi^{(i-1)} \models R_{e_{i-1}} \land \langle \text{skip} \land R_i \rangle$. That is, $\sigma_\pi^{(i-1)} \models R_{e_{i-1}} \land \bigcirc R_i$. Further, by the construction of NFG, $R_{i-1} \models R_{e_{i-1}} \land \bigcirc R_i \lor X$, $X$ being a PPTL formula in its normal form. So, $\models R_{e_{i-1}} \land \bigcirc R_i \to R_{i-1}$, leading to $\sigma_\pi^{(i-1)} \models R_{i-1}$. \hfill $\Box$

**Corollary 8.** For an infinite path $\pi = \langle R, R_{e0}, R_1, R_{e1}, R_2, \ldots \rangle$ in the NFG of formula $R$, $\sigma_\pi \models R$ if there exists $i \in \mathbb{N}_0$ such that $\sigma_\pi^{(i)} \models R_i$.

**Lemma 9.** If $\pi$ is an infinite path in the NFG $G$ of formula $R$ with $\text{FSC}(\pi, R) = \text{true}$, and $\sigma_\pi = P2M(\pi, G)$, then $\sigma_\pi \models R$.

**Proof.** Let $\pi = \langle R_0, R_{e0}, R_1, R_{e1}, \ldots \rangle$, where $R_0 \equiv R$, and $\text{Nod}_\pi = \{ R_0, R_1, R_2, \ldots \}$ be the set of all nodes of $\pi$, and $\text{CNod}_\pi = \{ R_i \mid R_i \in \text{Nod}_\pi \}$ and $R_1$ is a chop formula, and $\text{CCT}_\pi = \{ P; Q \mid R \in \text{CNod}_\pi \}$ and $R_1 \equiv (P; Q) \land R'$ be the set of all chop components over $\pi$. The proof proceeds by induction on the number of chop components over $\pi$.

**Base:** there are no any chop components over $\pi$. This implies $\text{CNod}_\pi = \emptyset$. By Lemma 6, $\sigma_\pi \models R$.

**Induction:** Suppose for any path $\pi$, $|\text{CCT}_\pi| < k \in \mathbb{N}_0$, $\sigma_\pi \models R$. When $|\text{CCT}_\pi| = k + 1$, let $R_1$ be the first chop formula over $\pi$. By the definition of chop formulas, $R_i = (P_i; Q_i) \land R'$. Since $\text{FSC}(\pi, R) = \text{true}$, we have $\text{fsc}(\pi^{(i)}, R_i) = \text{true}$. That is, there exists $i_j \in \mathbb{N}_0$ such that $R_{i_j} = P_{ij} \land \epsilon : Q_i = P_{ij} \land Q_i, i_j-l \geq 1$. This means one chop component is eliminated in at least one reduction step. Therefore, $|\text{CCT}_\pi^{(i)}| < k_i$, according to the hypothesis, $\sigma_\pi^{(i)} \models R_{i_j}$. Further, by Lemma 7 and Corollary 8, $\sigma_\pi \models R$. \hfill $\Box$

**Theorem 10.** The set of infinite path $\pi$ with $\text{FSC}(\pi, R) = \text{true}$ in the NFG $G$ of PPTL formula $R$ precisely characterizes the set of infinite models of $R$.

**Proof.** The theorem is a direct consequence of Lemmas 5 and 9. \hfill $\Box$

4. Decision procedure based on LNFG

To explicitly display whether or not the FSC_Property of a chop formula is satisfied, extra propositions $\text{Prop}_k, k \in \mathbb{N}_0$ and $k > 0$, are introduced. Let $\text{Prop}_k = \{ l_1, l_2, \ldots \}$ be the set of extra propositions with $\text{Prop} \cap \text{Prop}_k = \emptyset$. 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 form reductions, for any chop formula $P; Q$, we equivalently rewrite it as $P \land \text{fin}(l_k): \text{CHOP}\, P, Q$. Here, $\text{fin}(P)$ is defined by,

$$
\text{fin}(P) \equiv \Box (\epsilon \rightarrow P)
$$

Equivalently,

$$
\text{fin}(P) \equiv P \land \epsilon \lor \bigcirc \text{fin}(P)
$$

Thus, we have,\n
$$
P \land \text{fin}(l_k); Q \equiv \bigvee_{j=0}^{n_0} P_{ej} \land \epsilon \lor \bigvee_{i=0}^{n_1} (P_{cl} \land \bigcirc P_i) \land (l_k \land \epsilon \lor \bigcirc \text{fin}(l_k)); Q
$$

$$
\equiv \bigvee_{j=0}^{n_0} P_{ej} \lor l_k \land \epsilon \lor \bigvee_{i=0}^{n_1} (P_{cl} \land \bigcirc (P_i \land \text{fin}(l_k)))); Q
$$

$$
\equiv \bigvee_{j=0}^{n_0} (P_{ej} \land l_k \land \epsilon; Q) \lor \bigvee_{i=0}^{n_1} (P_{cl} \land \bigcirc (P_i \land \text{fin}(l_k); Q))
$$
As a result, by using $\text{fin}(l_k)$, $\text{FSC}_\text{Property}$ of $P; Q$ is satisfied if there exists an edge where $l_k$ holds. Furthermore, $\text{fin}(l_k)$ occurring in a node $P \land \text{fin}(l_k); Q$ means that $\text{FSC}_\text{Property}$ of $P; Q$ has not been satisfied at this node. For convenience, for a node in the form of $P \land \text{fin}(l_k); Q$ or $\bigwedge_{i=1}^{n} R_i$ with some $R_i = P_i \land \text{fin}(l_k); Q_i$, we add an extra label $l_k$ 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 by means of $l_k$ propositions.

**Definition 6 (Labeled Normal Form Graph, LNFG).** For a PPTL formula $P$, its LNFG is a tuple $G = (CL(P), EL(P), V_0, L) = \{[l_1, \ldots, l_m]\}$, where $CL(P)$, $EL(P)$ and $V_0$ are identical to the ones in the NFG, while each $L_{lk} \subseteq CL(P), 1 \leq k \leq m$, is the set of nodes with $l_k$ labels.

Algorithm LNFG based on Algorithm NFG is formalized by further rewriting any chop component $P; Q$ as $P \land \text{fin}(l_k); Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

In the following, we use $P_{\text{fin}}$ for $Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

In the following, we use $P_{\text{fin}}$ for $Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

Algorithm LNFG based on Algorithm NFG is formalized by further rewriting any chop component $P; Q$ as $P \land \text{fin}(l_k); Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

Algorithm LNFG based on Algorithm NFG is formalized by further rewriting any chop component $P; Q$ as $P \land \text{fin}(l_k); Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

In the following, we use $P_{\text{fin}}$ for $Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

In the following, we use $P_{\text{fin}}$ for $Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.

Algorithm LNFG based on Algorithm NFG is formalized by further rewriting any chop component $P; Q$ as $P \land \text{fin}(l_k); Q$ for some $k \in N_0$ whenever it is encountered. Basically, for a given PPTL formula $P$, the number of its LNFG $G$ is at most double of number of notes of NFG $G'$ since, in the worst case, a note of $G'$ can be placed in both $CL(P)$ and $CL'(P)$. The key difference between $G$ and $G'$ is that some extra labels $l_k$s appear in some edges and $l_k$s appear in some nodes of $G$.
Algorithm LNFG: Constructing LNFG of a PPTL formula.

Function LNFG(P)

\[*/ preconditions: P = ∨_i \ (0 \leq i) \ is a \ PPTL \ formula */\]
\[*/ postconditions: LNFG \ of \ P, \ G = (CL(P), EL(P), V_0, L = \{L_1, \ldots, L_m\}) */\]

begin function

\[V_0 = CL(P) = \{p_i \mid p_i \ \text{appears in} \ \bigvee_i \ p_i\}; \ \text{Mark}[p_i] = 0 \ \text{for each} \ i;\]
\[EL(P) = \emptyset; \ \text{Add}E = \text{Add}D = 0; \ k = 1; \ L = \emptyset; \ CL(P) = \emptyset; \ /* \ initialization */\]

while \(∃R \in (CL(P) ∪ CL'(P)) \{ε\}\) and \(\text{mark}[R] = 0\) do

\[R = \text{PRE}(R);\]

if \(R \neq false\) then \(*/ R \ is \ false \ if \ r \wedge \neg r \ appears \ in \ it */\)

\[R = (\bigwedge_{i=n} R_j) \wedge Z \ with \ R_j = P_i; Q_j \ and \ Z \ is \ not \ a \ \text{chop} \ \text{formula then}\]

for \(i = 1 \ to \ n \ /* \ adding \ labels \ to \ \text{chop} \ \text{formulas} */\)

if \(R_j\) has been rewritten with a \(\text{fin}()\ (1 \leq j < k)\) then

\[L_j = L_j ∪ \{R\} /* \ \text{incorporating} \ R \ to \ \text{corresponding} \ \text{LNFG */}\]

elseif \(R_j\) is \(CL(P) ∪ CL'(P)\) then \(/* \ \text{rewriting} \ R \ into \ its \ normal \ form */\)

end if

end for

end if

Q = N(F(R); \ \text{mark}[R] = 1; /* \ \text{deciding} \ whether \ \text{future} \ \text{products} \ or \ \text{terminal} \ \text{products} \ are \ contained \ in \ \text{NF */}\)

Q is \(\forall j =\ 1 \ to \ m \ /* \ \text{dealing} \ with \ each \ Q_{et} */\)

if \(Q_{et} \neq false\) then

if \(CE \neq CL(P)\) then \(CL(P) = CL(P) \cup \{ε\}; \ \text{end if} \)

end if

end for

AddE = 0;

end if

if \(AddD = 1\) then

for \(t = 1 \ to \ m \ /* \ \text{dealing} \ with \ each \ Q_{et} */\)

if \(Q_{et} \neq false\) then

if \(Q_{et} \notin CL(P)\) then

if \(\exists P_n \in CL(P)\) such that \(Q_{et} = P_n\) then

end if

else \(CL(P) \cup \{Q_{et}\}; EL(P) = EL(P) \cup \{(R, Q_{et}; P_s)\}; \ \text{mark}[Q_{et}] = 0;\ /* \ \text{a} \ \text{node} \ \text{added} \ \text{with} \ \text{fin}(L_i) \ \text{is} \ \text{forced} \ \text{to} \ \text{point} \ \text{to} \ \text{the} \ \text{first} \ \text{one} \ \text{in} \ \text{CL}(P) */\)

end if

end if

end if

end for

AddD = 0;

end if

end if

\end function

Corollary 13. For a circle \(π_c = ⟨(R_1, R_{e+1}), \ldots, R_j, R_{e+1}⟩^{m} \), with at least one label \(\tilde{L}_i (k \in Prop)\), of an infinite path \(π\) in the LNFG \(G\) of a formula \(R\), if \(FSC(π_c, R_i) = true\), then \(\text{Inf}(π_c) \geq 2\).

Lemma 14. For an infinite path \(π\) in the LNFG \(G = (CL(R), EL(R), V_0, L = \{L_1, \ldots, L_m\})\), of a PPTL formula \(R\), \(FSC(π, R) = true\) iff \(\text{Inf}(π) \nsubseteq L_i\) for all \(i, 1 \leq i \leq m\).

Proof. Since \(π\) is an infinite path in the LNFG \(G\) of a formula \(R\), there is a circle \(π_c = ⟨(R_1, R_{e+1}), \ldots, R_j, R_{e+1}⟩^{m} \) of \(π\). By Lemma 11, we have, \(FSC(π_c, R_i) = true\) iff \(FSC(π_c, R_i) = true\), and \(\text{Inf}(π) \nsubseteq L_i\) iff \(\text{Inf}(π_c) \nsubseteq L_i\). We need only to prove \(FSC(π_c, R_i) = true\) iff \(\text{Inf}(π_c) \nsubseteq L_i\) for all \(i, 1 \leq k \leq j\).
Example 1
LNFG of $P_{\text{inf}}$ means that any node for finite case, it has been proved in Theorem 1. For infinite case, it is a direct consequence of Theorem 10 and all $(\pi_{\text{inf}})$ is not marked by a single label $\tilde{l}_k$. In fact, there exists no node without label which is reachable from any node $R_h \in \text{inf}(\pi_c)$, $R_h$ has no label $\tilde{l}_k$. This guarantees that not all of nodes can be marked by a single label $\tilde{l}_k$. Consequently, a decision procedure for checking the satisfiability of a PPTL formula $P$ is satisfiable since $\text{inf}(\pi_c) \subseteq L_k$ leading to $\text{inf}(\pi_c) \not\subseteq L_k$ for all $1 \leq k \leq m$.

Case 2: Suppose $m \geq 1$ labels $\tilde{l}_1, \tilde{l}_2, \ldots, \tilde{l}_m$ are used in $\pi_c$. By Lemma 12, for all $i$ $(1 \leq i \leq m)$, at least one node $R_h \in \text{inf}(\pi_c)$ is not marked by $\tilde{l}_i$. Therefore, $\text{fsc}(\pi_c, R_h) = true$. Of course, for any node $R_h \in \text{inf}(\pi_c)$ without any labels, we also have $\text{fsc}(\pi_c, R_h) = true$. Therefore $\text{FSC}(\pi_c, R_i) = true$. □

Theorem 15. In the LNFG $G$ of a formula $P$, finite paths precisely characterize finite models of $P$; infinite paths with $\text{Int}(\pi) \not\subseteq L_k$, for all $1 \leq i \leq m$ precisely characterize infinite models of $P$.

Proof. For finite case, it has been proved in Theorem 1. For infinite case, it is a direct consequence of Theorem 10 and Lemma 14. □

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.

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 $\varepsilon$ node in $\text{CL}(P)$,
    return $P$ is satisfiable with finite models;
if there exists infinite path $\pi$ with $\text{Int}(\pi) \not\subseteq L_k$, for all $1 \leq i \leq m$
    return $P$ is satisfiable with infinite models;
else return unsatisfiable;
end function

Fig. 1. LNFG of $(p \land \Box p; \Box q) \land (\Box’; \Box q)$.

Fig. 2. LNFG of $q \land (\Box \varepsilon; q) \land (\Box p \land \Box \varepsilon; q)$.

Example 1 (Checking the satisfiability of formula $P \equiv (p \land \Box p; \Box q) \land (\Box’; \Box q)$). By Algorithm LNFG, LNFG $G = (\text{CL}(P), E(P), V_0, L = [L_1, \ldots, L_m])$ of formula $(p \land \Box p; \Box q) \land (\Box’; \Box q)$ is constructed as depicted in Fig. 1, where $\text{CL}(P) = \{n_0, n_1\}$, $E(P) = \{(n_0, p \land r, n_0), (n_0, p \land r, n_1), (n_1, p, n_1)\}$, $V_0 = \{n_0\}$, $L = [L_1, L_2]$, $L_1 = [n_0, n_1]$ and $L_2 = [n_0]$. There are two infinite paths in the LNFG $G$ of $P$: $\pi_1 = (n_0, p \land r, n_0)k, p \land r, n_0)k, p \land r, n_0)k$, $0 \leq k \in N_0$. As a result, $\text{inf}(\pi_1) = [n_0] \subseteq L_2 = [n_0] \subseteq L_1 = [n_0, n_1]$ and $\text{inf}(\pi_2) = [n_0] \subseteq L_1 = [n_0, n_1]$. Hence, the formula $P$ is unsatisfiable. In fact, there exists no node without $l_1$ label which is reachable from $n_0$ and $n_1$. □

Example 2 (Checking the satisfiability of the formula, $R \equiv q \land (\Box \varepsilon; q) \land (\Box p \land \Box \varepsilon; q)$). The new LNFG of formula $R \equiv q \land (\Box \varepsilon; q) \land (\Box p \land \Box \varepsilon; q)$ is illustrated in Fig. 2. As a result, $L_1 = [n_0]$ and $L_2 = [n_1]$. The formula is satisfiable since for the only infinite path $\pi = (\langle n_0, p \land q, n_1, p \land q \rangle)^m$, $\text{inf}(\pi) = [n_0, n_1] \not\subseteq L_i$, $i = 1, 2$. □
Example 3 (Checking the satisfiability of formula \((p \land \text{skip} ; q \land \text{skip})^+ \land \lnot \text{more} \land \text{fin}(l_1); r\)). The formula is obviously unsatisfiable since \((p \land \text{skip} ; q \land \text{skip})^+ \land \lnot \text{more} \land \text{fin}(l_1)\) cannot be satisfied with finite models. With the new decision procedure, the LNFG is depicted in Fig. 3 (2). As you can see, \(L_1 = \{n_0, n_1\}\), and the only infinite path \(\pi = \langle n_0, p, n_1, q \rangle^\omega\). This indicates the formula is unsatisfiable since \(\inf(\pi) = \{n_0, n_1\} \subseteq L_1\). □

5. Implementation of the decision procedure

This section focuses on the implementation aspects of the decision procedure. To do so, implementation details of relevant algorithms including pre-processing algorithms, the circle founding algorithm, the strong connected components algorithm i.e. Tarjan algorithm etc. are provided; the generalized Büchi accepting condition for LNFG is formalized. To show how the program works, three examples are presented.

5.1. Programs related issues

It has been proved that infinite paths with \(\inf(\pi) \not\subseteq L_i\) for all \(1 \leq i \leq m\) precisely characterize infinite models of \(P\). This can be equivalently expressed by “infinite paths with \(\inf(\pi) \not\subseteq L_i\) for all \(1 \leq i \leq m\) precisely characterize infinite models of \(P\)”, where \(L_i\) denotes \(\text{CL}(P) \setminus L_i\). Thus, the generalized Büchi accepting set can be defined by \(F = \{F_1, \ldots, F_m\}\) where \(F_i = L_i\) for each \(i\).

In the LNFG algorithm, when we partition the set of vertices based on \(\equiv\) relation, there are at most 2 nodes in each equivalence class, where

\[
P \equiv_p \text{ iff } p \equiv q \text{ without considering } \text{fin}(l_x) \text{ and } l_x; \quad l_x \not\in \text{prop}
\]

We have proved that the number of nodes of an NFG of a formula is finite [1]. In the same way, we can also prove the number of nodes of an LNFG of a formula is finite. In fact, when an LNFG is produced based on its NFG of a formula, each node of the NFG is at most split into two nodes. Since each node of an LNFG is marked by same label at most twice, so the set of labels is also finite.

To facilitate for producing LNFGs, some pre-processes are needed. We replace \(P \rightarrow Q, P \leftrightarrow Q\) by \(\lnot P \lor Q\) and \(P \land Q \lor \lnot P \land \lnot Q\) respectively; we also replace \(\varepsilon \text{ prj } \varepsilon\) by \(\varepsilon\), \((P_1, \ldots, P_m) \text{ prj } \varepsilon\) by \(P_1; \ldots; P_m\), \((P_1, \ldots, P_1 \lor P'_1, \ldots, P_m) \text{ prj } Q\) by \((P_1, \ldots, P_m) \text{ prj } Q \lor (P_1, \ldots, P_m) \text{ prj } Q\), \((P_1, \ldots, P_m) \text{ prj } (Q \lor Q')\) by \((P_1, \ldots, P_m) \text{ prj } Q \lor (P_1, \ldots, P_m) \text{ prj } Q\)', \((w \land P_1, \ldots, P_m) \text{ prj } Q\) and \((P_1, \ldots, P_m) \text{ prj } Q \land w\) by \(w \land (P_1, \ldots, P_m) \text{ prj } Q\); further, we replace \(\varepsilon; P \equiv P\), and \(\varepsilon \land w; P\) by \(w \land P\), respectively, where \(w\) is a state formula; finally, we replace \(true; P \equiv P\) by \(P \lor \bigcirc\bigcirc P\).

We move negation operators to the front of atomic propositions or chop or projection formulas as possible as we can by means of distributive laws on conjunction and disjunction operations; we also simplify the formula by using logic laws: \(\lnot \lnot P = P\), \(P \land \lnot P = false\) \((\varepsilon \land more = false)\), \(P \lor \lnot P = true\) \((\varepsilon \lor more = true)\), \(P \land false = false\), \(P \lor false = P\), \(P \equiv true = P\), \(P \equiv true = true\), \(false; P = false\), \(P; false = false\), \((P_1, \ldots, false, \ldots, P_m) \text{ prj } Q = false\), \((P_1, \ldots, P_m) \text{ prj } false = false\).

A loop without duplicated vertices is called a simple loop. To find out all simple loops in an LNFG, we use a DFS algorithm to search the LNFG. To do so, we use a global stack \(S\) to store the visited vertices in DFS. For a vertex \(v\), if its successor node \(w\) is not in \(S\), \(w\) is pushed into \(S\) and recursively DFS \(w\); otherwise, we find a loop and record it, then continue depth first searching. If all successor nodes of \(v\) are visited, \(v\) is popped and we return to DFS of the previous node.

Tarjan algorithm is a strongly connected components searching algorithm based on DFS. Define \(\text{DFN}(u)\) as searching number (timestamp) of \(u\) and \(\text{Low}(u)\) as searching number of the earliest vertex that is reachable for \(u\) or subtrees of \(u\). \(\text{Low}(u) = \min\{\text{DFN}(u), \text{DFN}(v), \text{DFN}(v)\}\), where \(u \rightarrow v\) is an edge.
Fig. 4. LNFG of formula (1).

Fig. 5. LNFG of formula (2).

Fig. 6. LNFG of formula (3).
Fig. 7. LNFG of formula (4).
5.2. Examples

The LNFGs of the first three examples are artificially presented in the previous section. In this section, we run the program to produce the LNFGs and the satisfaction results of the formulas automatically.

(1) \( P \equiv (p \land \Box \Diamond p; \Diamond \Box q) \land (\Diamond p; \Diamond \Box q) \).

The LNFG of formula (1) is constructed in Fig. 4.

(2) \( R \equiv q \land (\Diamond e; q) \land (p \land \Diamond \Box e; q) \).

The LNFG of formula (2) is shown in Fig. 5.

(3) \( (p \land \text{skip}; q \land \text{skip}) \land \Diamond \text{more}; r \).

The LNFG of formula (3) is shown in Fig. 6.

(4) \((\Box \Diamond p_0 \rightarrow \Box \Diamond p_1) \land (\Box \Diamond p_2 \rightarrow \Box \Diamond p_6) \land (\Box \Diamond p_3 \rightarrow \Box \Diamond p_2) \land (\Box \Diamond p_5 \rightarrow \Box \Diamond p_3) \land (\Box \Diamond p_6 \rightarrow \Box \Diamond (p_5 \lor p_4)) \land (\Box \Diamond p_7 \rightarrow \Box \Diamond p_6) \land (\Box \Diamond p_1 \rightarrow \Box \Diamond p_7)) \rightarrow \Box \Diamond p_8 \).

This formula is a typical example given in [6]. Our decision procedure can produce the LNFG of the formula as shown in Fig. 7.

6. Conclusion

We have presented a practical decision procedure for PPTL with infinite models. As a matter of fact, this decision procedure can also be used as decision procedures for PITL [5] and PLTL [9] since PPTL subsumes PITL and PLTL. Based on the decision procedure, a model checker for PPTL (can also be used for PITL and PLTL) have also been developed [10]. In the future, we will further investigate the techniques such as symbolic, bounded, abstract, probabilistic model checking for taking over the state space exploring problem. We will also develop practical model checkers for PPTL.

Acknowledgements

We are grateful to Miss Xiaofang Liu for her assistance in implementing the decision procedure in C++.

Appendix A. Algorithm PRE

This algorithm is recursive and basically uses case statements to deal with different formulas. Function PRE(R), ImpEquPre(R), PreRecur(R), not_PRE(R), con_PRE(R), dis_PRE(R), cchop_PRE(R), prj_PRE(R).

---

**Algorithm PRE(R): preprocessing of a PPTL formula R.**

**Function** PRE(R)

/* precondition: R is a PPTL formula as a syntax tree */

/* postcondition: PRE(R) returns the reduced form of R */

begin function

// reduce operator \( \rightarrow \) and \( \leftrightarrow \)

ImpEquPre(R);

return PreRecur(R);

end function

---

**Algorithm ImpEquPre(R): preprocessing \( \rightarrow \) and \( \leftrightarrow \) of a PPTL formula R.**

**Function** ImpEquPre(R)

/* precondition: R is a PPTL formula as a syntax tree */

/* postcondition: ImpEquPre(R) returns the \( \rightarrow \) and \( \leftrightarrow \) reduced form of R */

begin function

// for \( i = 1 \) to \( m \) /* assume \( R \) has \( m \) subtrees \( R_1, \ldots, R_m \) */

for (i = 1 to m)

if \( R_i \) is \( P \rightarrow Q \), then

ImpEquPre(Pi); ImpEquPre(Qi); \( R_i \) is rewritten as \( \neg Pi \lor Qi \);

else if \( R_i \) is \( P \leftrightarrow Q \), then

ImpEquPre(Pi); ImpEquPre(Qi); \( R_i \) is rewritten as \( P_i \land Q_i \lor \neg P_i \land \neg Q_i \);

end if

end for

return R;

end function
Algorithm `PreRecur(R)`: preprocessing a PPTL formula $R$ without $\rightarrow$ and $\leftrightarrow$.

Function
/** precondition: $R$ is a PPTL formula as a syntax tree without operator $\rightarrow$ and $\leftrightarrow$ */
/** postcondition: PreRecur($R$) returns the reduced form of $R$ */

begin function
  case
    $R$ is $\neg P$: return $\not_{PRE}(R)$;
    $R$ is $P \land Q$: return $\con_{PRE}(R)$;
    $R$ is $P \lor Q$: return $\dis_{PRE}(R)$;
    $R$ is $P_1; \ldots; P_m$: return $\chop_{PRE}(R)$;
    $R$ is $(P_1, \ldots, P_m) \proj Q$: return $\proj_{PRE}(R)$;
    $R$ is $\Box P$: return $\Box_{PRE}(P)$;
    $R$ is $\Diamond P$: return $\Diamond_{PRE}(P)$;
    $R$ is $\text{len}(x)$: if $x=0$: then return $\varepsilon$; else return $\varepsilon^*\varepsilon$; endif
    $R$ is $\text{fin}(P)$: return $\text{fin}_{PRE}(P)$;
    $R$ is $\text{keep}(P)$: return $\text{keep}_{PRE}(P)$;
    $R$ is $\text{halt}(P)$: return $\text{halt}_{PRE}(P)$;
    $R$ is $\text{true}$: return $R$;
    $R$ is $\text{false}$: return $R$;
    $R$ is $p(p \in \text{prop})$: return $R$;
  endcase
end function

Algorithm `not_{PRE}(R)`: preprocessing a PPTL formula $R = \neg P$.

Function
/** precondition: $R$ is a PPTL formula, $R = \neg P$ */
/** postcondition: not_{PRE}(R) returns the reduced form of $R$ */

begin function
  case
    $R$ is $\varepsilon$: return $\text{more}$;
    $R$ is $\text{more}$: return $\varepsilon$;
    $R$ is $\text{true}$: return $\false$;
    $R$ is $\false$: return $\true$;
    $R$ is $\neg\neg P$: return $\text{PreRecur}(P)$;
    $R$ is $\neg P$: break;
  endcase
PreRecur($P$);
end function
Algorithm con_PRE(R): preprocessing a PPTL formula $R = P \land Q$.

Function
/* precondition: $R$ is a PPTL formula, $R = P \land Q$ */
/* postcondition: con_PRE($R$) returns the reduced form of $R$ */
begin function
\begin{algorithmic}
\Function{case}{\Var{R}}
\If {$R$ is $P \land \neg P$} return false; \EndIf
\If {$R$ is $\varepsilon \land \text{more}$} return false; \EndIf
\If {$R$ is $P \land \text{false}$} return false; \EndIf
\If {$R$ is $P \land \text{true}$} \Call{PreRecur}{$P$}; \EndIf
\If {$R$ is $\text{true} \land Q$} \Call{PreRecur}{$P$}; \EndIf
\If {$R$ is $P \land Q$} break; \EndIf
\EndFunction
\begin{algorithmic}
\Function{PreRecur}{$P$};
\EndFunction
\begin{algorithmic}
\Function{case}{\Var{R}}
\EndFunction
\end{algorithmic}
\end{algorithmic}
end function

Algorithm dis_PRE(R): preprocessing a PPTL formula $R = P \lor Q$.

Function
/* precondition: $R$ is a PPTL formula, $R = P \lor Q$ */
/* postcondition: dis_PRE($R$) returns the reduced form of $R$ */
begin function
\begin{algorithmic}
\Function{case}{\Var{R}}
\If {$R$ is $P \lor \neg P$} return true; \EndIf
\If {$R$ is $\varepsilon \lor \text{more}$} return true; \EndIf
\If {$R$ is $P \lor \text{false}$} \Call{PreRecur}{$P$}; \EndIf
\If {$R$ is $\text{false} \lor Q$} return PreRecur($Q$); \EndIf
\If {$R$ is $P \lor Q$} return true; \EndIf
\If {$R$ is $P \lor Q$} break; \EndIf
\EndFunction
\begin{algorithmic}
\Function{PreRecur}{$P$};
\EndFunction
\begin{algorithmic}
\Function{case}{\Var{R}}
\EndFunction
\end{algorithmic}
\end{algorithmic}
end function
Algorithm chop_PRE(R): preprocessing a PPTL formula R = P; Q.

Function
/* precondition: R is a PPTL formula, R = P; Q */
/* postcondition: chop_PRE(R) returns the reduced form of R */
begin function
if R is \( P_1; \ldots; P_m \) and any of \( (P_1; \ldots; P_m) \) is false
then return false
endif
begin case
R is \( e : P \): return PreRecur(P);
R is \( e \land w : P \): return PreRecur(w \land P);
if PreRecur(\( P \)) then return true; return PreRecur(P \lor \lor P); R is \( P; Q \): break;
endcase
end function

Algorithm prj_PRE(R): preprocessing a PPTL formula R = \( P_1, \ldots, P_m \) prj \( Q \).

Function
/* precondition: R is a PPTL formula, R = \( (P_1, \ldots, P_m) \) prj \( Q \) */
/* postcondition: prj_PRE(R) returns the reduced form of R */
begin function
begin case
R is \( e \): return \( e \);
R is \( (P_1, \ldots, P_m) \) prj \( e \): return PreRecur(\( P_1; \ldots; P_m \));
R is \( (P_1, \ldots, P_1 \lor P_1, \ldots, P_m) \) prj \( Q \);
return PreRecur(\( (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \lor (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
R is \( (P_1, \ldots, P_m) \) prj \( Q \lor Q \): return PreRecur(\( (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \lor (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
R is \( (w \land (P_1, \ldots, P_m) \) prj \( Q \): return PreRecur(\( w \land (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
/* where w is a state formula */
R is \( (P_1, \ldots, P_1 \lor P_1, \ldots, P_m) \) prj \( Q \land w \): return PreRecur(\( w \land (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
/* where w is a state formula */
R is \( (P_1, \ldots, P_m) \) prj \( Q \lor w \): return PreRecur(\( w \land (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
/* where w is a state formula */
endcase
for i = 1 to m
PreRecur(P_i);
if \( P_i \) is false then return false; endif
endfor
begin case
R is \( e \): return \( e \);
R is \( (P_1, \ldots, P_m) \) prj \( e \): return PreRecur(\( P_1; \ldots; P_m \));
R is \( (P_1, \ldots, P_1 \lor P_1, \ldots, P_m) \) prj \( Q \);
return PreRecur(\( (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \lor (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
R is \( (P_1, \ldots, P_m) \) prj \( Q \lor Q \): return PreRecur(\( (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \lor (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
/* where w is a state formula */
R is \( (w \land (P_1, \ldots, P_m) \) prj \( Q \): return PreRecur(\( w \land (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
/* where w is a state formula */
R is \( (P_1, \ldots, P_m) \) prj \( Q \lor w \): return PreRecur(\( w \land (P_1, \ldots, P_1, \ldots, P_m) \) prj \( Q \));
/* where w is a state formula */
endcase
end function
Appendix B. DFSLoopSearch Algorithm

**Algorithm** DFSLoopSearch(G): search loops in LNFG G.

**Function**
/* precondition: G an LNFG G = {CL, EL, V₀,..., L = {L₁,..., Lₘ}} */
/* postcondition: DFSLoopSearch(G) returns a boolean variable indicating whether there exist loops or not */

**begin function**

//initialization
dfsStack = Φ; /* global stack of DFS search */
loopSet = Φ; /* global set of simple loops in LNFG */
extistLoop = false; /* indicate whether there exist loops in LNFG */

//kernel
deleteIrrelevantNodes(G);
if CL = Φ then return existLoop; else existLoop = true; endif
for i = 1 to n
  if vi.DFN undefined then Tarjan(G, vi);endif
endfor
for i = 1 to m /* assume there are m subgraphs */
  choose a random vertex v in CL of Gᵢ, and dfsStack.push(v);
dfsOneSubgraph(Gᵢ);
endfor

**end function**

Algorithm deleteIrrelevantNodes(G): delete irrelevant nodes.

**Function**
/* precondition: G an LNFG G = {CL, EL, V₀,..., L = {L₁,..., Lₘ}} */
/* postcondition: DFSLoopSearch(G) delete nodes whose in degree or out degree is zero */

**begin function**
for i = 1 to m
  vi.indgree = 0; vi.outdegree = 0;
endfor
isEmpty = false;
nodeDeleted = false;
for i = 1 to n
  wi.outdegree++; vi.indgree++; /* assume eᵢ is wᵢ → vᵢ */
endfor
for i = 1 to l /* assume there l nodes in CL */
  if vi.indegree = 0 then delete vi and delete edges start with vi; nodeDeleted = true; endif
  if vi.outdegree = 0 then delete vi and delete edges end with vi; nodeDeleted = true; endif
endfor
if CL = Φ then return;
else if nodeDeleted = true then deleteIrrelevantNodes(G); endif

**end function**
Algorithm dfsOneSubgraph(G): DFS a connected graph and obtain its loops.

Function
/* precondition: G a strongly connected graph */
/* postcondition: DFSLoopSearch(G) obtains loops in G */
begin function
    w = dfsStack.front();
    for i=1 to m /* assume there are m edges starting from w and e_i = w → v_i */
        if v_i is in dfsStack
            loopSet+=\{v_i, ..., w\};
        else
            dfsStack.push(v_i); dfsOneSubgraph(G);
        endif
    endfor
    dfsStack.pop();
end function

Appendix C. Tarjan Algorithm

Algorithm Tarjan(G, v): obtain strongly connected graphs associated with v.

Function
/* precondition: G a directed graph, v is a vertex in G */
/* postcondition: Tarjan(G) obtains strongly connected components associated with v */
begin function
    v.DFN = v.Low = ++Index; /* Index is numeral order in Tarjan */
    Stack.push(v);
    foreach w in E.G
        if w is not visited
            then
                Tarjan(G, w);
                v.Low = min{v.Low, w.Low};
            else
                if w is in Stack
                    then
                        v.Low = min{v.Low, w.DFN};
                    endif
                endif
            endif
    endif
    if v.DFN = v.Low then
        subGraph CG;
        do u = Stack.pop(); CG.add(u);
        until (u = v)
    endif
end function

References