Decidability of Propositional Projection Temporal Logic with Infinite Models[⋆]

Zhenhua Duan and Cong Tian

Institute of Computing Theory and Technology
Xidian University, Xi'an, 710071, P.R. China
{zhhduan, ctian}@xidian.edu.cn

Abstract. This paper investigates the satisfiability of Propositional Projection Temporal Logic (PPTL) with infinite models. A decision procedure for PPTL formulas is formalized. To this end, Normal Form (NF) and Normal Form Graph (NFG) for PPTL formulas are defined and an algorithm constructing NFG for PPTL formulas is presented. Further, examples are also given to illustrate how the decision algorithm works.

Keywords: interval temporal logic; satisfiability; decidability; infinite model; model checking.

1 Introduction

Interval Temporal Logic (ITL) [3,4] is a useful formalism for the specification and verification of concurrent systems. In the past two decades, a number of axiom systems have been proposed [2, 4, 5, 13, 14, 18] to verify properties of concurrent systems. Although verification of programs can be managed by complete deductive systems of ITL, Model Checking approach which is an automatic method based on model checking algorithms has not extensively been studied in ITL.

Satisfiability and validity of formulas are fundamental issues in the model theory of a logic. Moreover, satisfiability plays an important role in the model checking approach. Within ITL community, several researchers have looked at decision procedures. Halpern and Moszkowski [3] proved the decidability of Quantifier Propositional ITL (QPITL) over finite time. Kono presented a tableaux-based decision procedure for QPITL with projection [12]. Bowman and Thompson [13] presented the first tableaux-based decision procedure for quantifier-free propositional ITL (PITL) over finite intervals with projection. To the best of our knowledge, all the existing decision procedures for ITL are confined in finite intervals. However, many reactive systems are designed not to terminate. So, to verify those systems using model checking, a decision procedure with infinite models is required.

Projection Temporal Logic (PTL) [6,8,11,19] is an extension of ITL. It extends ITL to include infinite models and a new projection construct, $(P_1, ..., P_m)\ prj\ Q$. The new projection construction can be treated as a combination of the parallel

[⋆] This research is supported by the NSFC Grant No. 60373103 and 60433010.

$(P \mid\mid Q)$ and original projection construct, $(P\ proj\ Q)$ [3]. Formulas P_1, ..., P_m and Q are autonomous; each formula has the right to specify its own interval over which it is interpreted. Although formula Q is interpreted in a parallel way with formula $(P_1\ ;\ ...\ ;\ P_m)$ (; denotes the chop operator, see Section 2), the communication between them is only at rendezvous states, and the formulas may terminate at different time points.

Compared with the *proj* construct $(P\ proj\ Q)$ defined in [3], *prj* is more powerful. Firstly, $(P_1, ..., P_m)\ prj\ Q$ is able to handle terminal formulas (see Section 2) while $(P\ proj\ Q)$ construct cannot. Within $(P\ proj\ Q)$, the formula P is interpreted repeatedly over a series of consecutive subintervals whose endpoints form the interval over which Q is interpreted. This may result in repeating the same global state in the interpretation of Q several times if P is interpreted over a subinterval of zero length. In [13], Bowman and Thompson changed this by confining P not to be a terminal formula. However, this restriction causes the loss of flexibility since the formula P is necessary to be interpreted in a point state in some circumstances. Secondly, in $(P\ proj\ Q)$, the series of Ps and Q terminate at the same time. In practice, the formulas Ps and Q are not so regularly interpreted. In our definition, formulas P_1, ..., P_m and Q are autonomous; they are interpreted independently and may terminate at different time points. Thirdly, $(P\ proj\ Q)$ is only defined over finite intervals and hard to be extended to infinite intervals. In contrast, our projection construct is defined over both finite and infinite intervals. Finally, *prj* can subsume the central operator chop in ITL, $P; Q \equiv (P, Q)\ prj\ \varepsilon$, but *proj* cannot.

ITL has also been extended into infinite time by Moszkowski [4]. However, the chop construct is yet different from the one in PTL in the case of infinite models. Within ITL, $(P; Q)$ holds over an interval if and only if either the interval can be split into two parts and P holds on the first part and Q holds on the second part (the interval can be finite or infinite), or P holds over the whole interval and the interval is infinite; whereas within PTL, $(P; Q)$ is true only for the first case. That is, in PTL, P is only interpreted over a finite interval, while in ITL, P can be interpreted over an infinite interval. $(P; Q)$ is not satisfiable in PTL if P has only infinite models. However, the two constructs can express each other directly. Let $;_m$ and $;_d$ denote the chop operators in ITL and PTL respectively. Then we have,

$$P\ ;_m Q \equiv (P\ ;_d Q) \vee P \wedge \Box\ \overline{\varepsilon}$$
$$P\ ;_d Q \equiv (P \wedge \Diamond\varepsilon)\ ;_m Q$$

where $\Box$ (always) and $\Diamond$ (sometimes) are modal operators and $\overline{\varepsilon}$ means the current interval is not over. In the sense of *Until* construct in Linear Temporal Logic (LTL) [15], $P;_d Q$ can be viewed as the strong version while $P;_m Q$ can be thought of as the weak version of the chop construction.

Within PTL, plenty of logic laws have been formalized and proved [6, 11], and a decision procedure for checking satisfiability of Propositional Projection Temporal Logic (PPTL) formulas with finite models has been given in [9]. Nevertheless, to check the satisfiability of the underlying logic with infinite models, a decision procedure is also required.

Therefore, we are motivated to investigate the decidability of PPTL formulas with infinite models. To this end, Normal Form (NF) and Normal Form Graph (NFG) for PPTL formulas are defined. Further, an algorithm constructing NFGs for PPTL formulas is also formalized. Basically, the normal form is the same as we gave in [6, 8] for Tempura programs. An NFG is a useful formalism for constructing models of a PPTL formula.

Accordingly, a decision procedure based on NFGs for checking the satisfiability of PPTL formulas with infinite models is formalized in the paper. With this method, for a given formula, an NFG can be constructed by means of its normal form. A finite path from the root node to the ε node (i.e., terminating node, see Section 2) in the NFG of the formula corresponds to a finite model of the formula while an infinite path emanating from the root corresponds to an infinite model of the formula. It is clear that a formula is satisfiable if and only if there exists a finite or infinite path in its NFG. This decision procedure can also be used to check the satisfiability of PITL with minor changes [10].

The paper is organized as follows. The next section briefly presents the syntax, semantics and some logic laws of the underlying logic. Section 3 gives the definition of the normal form of PPTL formulas. In Section 4, the normal form graph is defined; further, an algorithm constructing NFGs is formalized, and the finiteness of NFGs is proved. A decision algorithm for checking the satisfiability of PPTL formulas with infinite models is demonstrated in Section 5. Conclusions are drawn in Section 6.

2 Propositional Projection Temporal Logic

Our underlying logic is Propositional Projection Temporal Logic (PPTL) [6, 11, 7, 19]. It is an extension of Propositional Interval Temporal Logic (PITL) [3].

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

$$P \ ::= \ p \mid \bigcirc P \mid \neg P \mid P_1 \vee P_2 \mid (P_1, ..., P_m)\ prj\ P$$

where $p \in Prop$, P_1 , ..., P_m and P are all well-formed PPTL formulas. A formula is called a state formula if it contains no temporal operators. The abbreviations $true$, $false$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $true \overset{\text{def}}{=} P \vee \neg P$ and $false \overset{\text{def}}{=} P \wedge \neg P$. Also we have the following derived formulas:

$$\begin{array}{llll llll}
A_1 & \varepsilon & \overset{\text{def}}{=} & \neg \bigcirc true & A_2 & \bar{\varepsilon} & \overset{\text{def}}{=} & \neg\varepsilon \\
A_3 & \bigcirc^0 P & \overset{\text{def}}{=} & P & A_4 & \bigcirc^n P & \overset{\text{def}}{=} & \bigcirc(\bigcirc^{n-1} P) \\
A_5 & len\ n & \overset{\text{def}}{=} & \bigcirc^n \varepsilon & A_6 & skip & \overset{\text{def}}{=} & len\ 1 \\
A_7 & \odot P & \overset{\text{def}}{=} & \varepsilon \vee \bigcirc P & A_8 & \Diamond P & \overset{\text{def}}{=} & true\ ;\ P \\
A_9 & \Box P & \overset{\text{def}}{=} & \neg\Diamond\neg P & A_{10} & P\ ;\ Q & \overset{\text{def}}{=} & (P,\ Q)\ prj\ \varepsilon
\end{array}$$

Semantics. 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 the 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 the set N_0 of non-negative integers and ω, $N_\omega = N_0 \cup \{\omega\}$, and extend the comparison operators, $=, <, \leq$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Moreover, we define $\preceq$ as $\leq - \{(\omega, \omega)\}$. To simplify definitions, we will denote σ as $< s_0, ..., s_{|\sigma|} >$, where $s_{|\sigma|}$ is undefined if σ is infinite. With such a notation, $\sigma_{(i..j)}$ $(0 \leq i \preceq j \leq |\sigma|)$ denotes the sub-interval $< s_i, ..., s_j >$ and $\sigma^{(k)}$ $(0 \leq k \preceq |\sigma|)$ denotes $< s_k, ..., s_{|\sigma|} >$. The concatenation of a finite σ with another interval (or empty string) σ' is denoted by $\sigma \cdot \sigma'$.

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

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

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,

$$< s_0, s_1, s_2, s_3, s_4 >\downarrow (0, 0, 2, 2, 2, 3) =< s_0, s_2, s_3 >$$

An interpretation is a quadruple $\mathcal{I} = (\sigma, i, k, j)^1$, where σ is an interval, i, k are integers, and j an integer or ω such that $i \leq k \preceq j \leq |\sigma|$. We use the notation $(\sigma, i, k, j) \models P$ to denote that formula P is interpreted and satisfied over the subinterval $< s_i, ..., s_j >$ 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 atomic 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 $(\sigma, i, k+1, j) \models P$

$I - prj$ $\mathcal{I} \models$ $(P_1, ..., P_m)\ prj\ Q$ if there exist integers $k = r_0 \leq r_1 \leq ... \leq r_m \leq j$ such that $(\sigma, 0, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_{l-1}, r_l) \models P_l$ $1 < l \leq m$, and $(\sigma', 0, 0, |\sigma'|) \models Q$ 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 $0 \leq h \leq m$

Satisfaction and Validity. A formula P is satisfied by an interval σ, denoted by $\sigma \models P$, if $(\sigma, 0, 0, |\sigma|) \models P$. 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.

Two formulas, P and Q, are equivalent, denoted by $P \equiv Q$, if $\models \Box(P \leftrightarrow Q)$. A formula P is called a terminal formula if $P \equiv P \wedge \varepsilon$, a non-local formula if $P \equiv P \wedge \bar{\varepsilon}$, and a local formula if P is a state or terminal formula.

Precedence Rules. In order to avoid an excessive number of parentheses, the following precedence rules are used:

1 Parameter i is used to handle past operators and redundant with the current version of the underlying logic. However, to keep the consistency with our previous research, it is kept in the interpretation.

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

where 1=highest and 5=lowest.

3 Normal Form of PPTL

Let Q be a PPTL formula and Q_p denote the set of atomic propositions appearing in Q. The normal form of Q can be defined as follows.

$$Q \equiv \bigvee_{j=1}^{n_0} (Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{n} (Q_{ci} \wedge \bigcirc Q'_i) \tag{1}$$

where $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$, $l = |Q_p|$, $1 \leq n$ (also n_0) $\leq 3^l$, $1 \leq m$ (also m_0) $\leq l$; $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, $\dot{r}$ denotes r or $\neg r$; Q'_i is a general PPTL formula[2]. In some circumstances, for convenience, we write $Q_e \wedge \varepsilon$ instead of $\bigvee_{j=1}^{n_0} (Q_{ej} \wedge \varepsilon)$ and $\bigvee_{i=1}^{r} (Q_i \wedge \bigcirc Q'_i)$ instead of $\bigvee_{i=1}^{n} (Q_{ci} \wedge \bigcirc Q'_i)$. Thus,

$$Q \equiv (Q_e \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (Q_i \wedge \bigcirc Q'_i) \tag{2}$$

where Q_e and Q_i are state formulas or *true*. Further, in a normal form, if $\bigvee_i Q_i \equiv$ *true* and $\bigvee_{i \neq j} (Q_i \wedge Q_j) \equiv false$, then this normal form is called a complete normal form. The complete normal form plays an important role in transforming the negation of a PPTL formula into its normal form. For example, if P has been written to its complete normal form:

$$P \equiv P_e \wedge \varepsilon \vee \bigvee_{i=1}^{r} (P_i \wedge \bigcirc P'_i) \tag{3}$$

then we have,

$$\neg P \equiv \neg P_e \wedge \varepsilon \vee \bigvee_{i=1}^{r} (P_i \wedge \bigcirc \neg P'_i)$$

For any PPTL formula P, P can be rewritten to its normal form and complete normal form. The details of the proofs and the algorithms transforming PPTL formulas into normal forms and complete normal forms can be found in [9].

Obviously, the normal form of a formula enables us to rewrite the formula into two parts: the present and future ones. The present part is a state formula while the future part is either ε or a next formula. This normal form inspires us to construct a graph for describing the models of the formula.

4 Normal Form Graph

Our decision algorithm is based on a so called Normal Form Graph (NFG). It is constructed according to the normal form.

4.1 Definition of NFG

For a PPTL formula P, the NFG of P is a directed graph, $G = (CL(P), EL(P))$, where $CL(P)$ denotes the set of nodes and $EL(P)$ denotes the set of edges in

[2] It is an exercise to prove $n, n_0 \leq 3^l$.

the graph. In $CL(P)$, each node is specified by a formula in PPTL, while in $EL(P)$, each edge is identified by a triple (Q, Q_e, R). Where Q and R are nodes and Q_e is labeling of the directed arc from Q to R. $CL(P)$ and $EL(P)$ of G can be inductively defined as in Definition 1. Note that the normal form employed in this definition is normal form (1).

Definition 1. For a PPTL formula P, set of nodes, $CL(P)$, and set of of edges, $EL(P)$, connecting nodes in $CL(P)$ are inductively defined as follows:

1. $P \in CL(P)$;
2. For all $Q \in CL(P) \setminus \{\varepsilon, false\}$, if $Q \equiv \bigvee_{j=1}^{h} (Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{k} (Q_{ci} \wedge \bigcirc Q'_i)$, 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, $1 \leq i \leq k$;
3. $CL(P)$ and $EL(P)$ are only generated by 1 and 2; thus, the NFG of P can be defined as a directed graph G given by
4. $G = (CL(P), EL(P))$.

In the NFG of P, the root node P is denoted by a double circle, ε node by a small black dot, and each of other nodes by a single circle. If the main operator of a node (formula) is chop (;), a letter F is placed in the circle or double circle. Each of the edges is denoted by a directed arc connecting two nodes. Examples of NFGs are depicted in Fig.1.

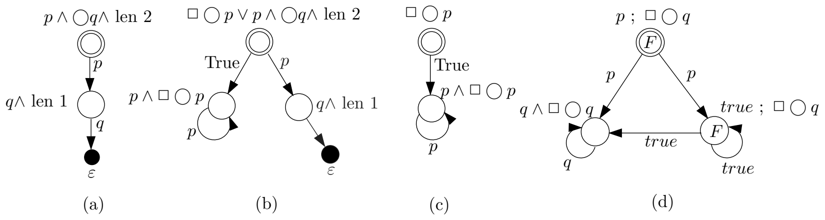
$p \wedge \bigcirc q \wedge$ len 2
p
$q \wedge$ len 1
q
ε
(a)
$\Box \bigcirc p \vee p \wedge \bigcirc q \wedge$ len 2
True
p
$p \wedge \Box \bigcirc p$
p
$q \wedge$ len 1
ε
(b)
$\Box \bigcirc p$
True
$p \wedge \Box \bigcirc p$
p
(c)
p ; $\Box \bigcirc q$
F
p
p
$q \wedge \Box \bigcirc q$
$true$; $\Box \bigcirc q$
$true$
F
q
$true$
(d)

Fig. 1. Examples of NFGs

Intuitively, the NFG of formula P describes models of formula P since it is constructed according to the normal form. However, the NFG of formula P generated by Definition 1 might wrongly describe models of P with the chop construct, $Q_1; Q_2$, when Q_1 has infinite models.

To solve the problem, two cases need to be considered for chop construct $Q_1; Q_2$. 1) Q_1 has only infinite models. In this case, $Q_1; Q_2$ has no models, so edges might not depart from it. 2) Q_1 has both finite and infinite models. In this case, we need to eliminate all infinite models of Q_1. Based on the above analysis, we have formalized two algorithms, *NFG* and *Simplify* to generate a subgraph of the NFG for a given formula so that models of the formula can correctly be generated. In algorithm *NFG*, a label F is employed to indicate that a node in a cycle can only be repeated for finitely many times. Therefore, whenever the chop operator is the main operator of a formula, F is placed in the node as a mark.

4.2 Algorithm for Constructing NFG

In the following, algorithm *NFG* for constructing the NFG of a PPTL formula is presented. It is merely a sketch of the implementation of Definition 1. The algorithm uses *mark*[] to indicate whether or not a formula needs to be decomposed. If $mark[P] = 0$ (unmarked), then P needs further to be decomposed, otherwise $mark[P] = 1$ (marked), thus P has been decomposed or needs not to be done. Note that algorithm *NFG* employs algorithm *NF* [9] to transform a formula into its normal form. In the algorithm, we consider the chop construct in the form of $R \equiv Q_1; Q_2$ carefully since Q_1 may only have infinite models and cause R to be $false$. Therefore, if Q_1 has no finite models it needs not further to be decomposed and marked with 1 ($mark[R] = 1$) otherwise it needs further to be dealt with and marked with 0 ($mark[R] = 0$). Also label F is employed to denote the chop formula cannot be repeated infinitely many times. Further, in the algorithm, two global boolean variables AddE and AddN are employed to indicate whether or not ε and the next formulas in the normal form are encountered respectively. Note also that the algorithm only deals with formulas in a pre-prepared form in which only $\vee$, $\wedge$ and $\neg$, as well as temporal operators $\bigcirc$, ;, $\square$ and prj are contained. Others such as $\rightarrow$, $\leftrightarrow$, $\Diamond$, $\neg\neg$ etc. can be eliminated since they can be expressed by the basic operators.

```
function NFG(P)
/* precondition: P is a PPTL formula in pre-prepared form*/
/* postcondition: NFG(P) computes NFG of P, G = (CL(P), EL(P))*/
begin function
                                                              /*initialization*/
    $CL(P) = \{P\}$; $EL(P) = \phi$; $mark[P] = 0$; AddE = AddN = 0;
    while there exists $R \in CL(P) \setminus \{\varepsilon, false\}$, and $mark[R]$ ==0
      do  $mark[R]$ =1;                                /*marking R is decomposed*/
      if  $R$ is $Q_1 ; Q_2$                                /*computing NFG of $Q_1$*/
      then  add label $F$ in node $R$; $G' = (CL(Q_1), EL(Q_1)) = NFG(Q_1)$;
            if $\varepsilon \notin CL(Q_1)$ then continue;
                                           /*R needs not decomposed, jump to while*/
      $Q = NF(R)$;
      case
         $Q$ is $\bigvee_{j=1}^{h} Q_{ej} \wedge \varepsilon$:  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 $\bigvee_{j=1}^{h} Q_{ej} \wedge \varepsilon \vee \bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$: AddE=AddN=1;
                                                     /*both parts of NF needs 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                              /*add second part of NF*/
              for $i = 1$ to $k$ do if $Q'_i$ is $false$
                          then $mark[Q'_i]$=1;
                                            /*$Q'_i$ needs not decomposed*/
                          else if $Q'_i \notin CL(P)$
                             then $mark[Q'_i]$=0;
                                            /*$Q'_i$ needs decomposed*/
              $CL(P) = CL(P) \cup \bigcup_{i=1}^{k} \{Q'_i\}$;
              $EL(P) = EL(P) \cup \bigcup_{i=1}^{k} \{(R, Q_i, Q'_i)\}$;
              AddN=0;
    end while
    return $G$;
end function
```

4.3 Finiteness of NFG

In the NFG of PPTL formula P generated by algorithm NFG, the set $CL(P)$ of nodes and the set $EL(P)$ of edges are inductively produced by repeatedly rewriting the unmarked nodes into their normal form. So one question we have to answer is whether or not the rewriting process terminates. Fortunately, we can prove that, for any PPTL formula P, the number of nodes in $CL(P)$ are finite, that is, $|CL(P)| = k \in N_0$. To prove this, Lemma 1 $\sim$ 6 are needed. The proofs of these lemmas and theorems can be found in [10].

Lemma 1. For any PPTL formula P, $|CL(\bigcirc P)| \leq |CL(P)| + 1$.

The lemma tells us that the number of nodes in the NFG of $\bigcirc P$ is at most the number of nodes in the NFG of P plus one.

Lemma 2. If Q is a PPTL formula and P_e is a state formula, then $|CL(P_e \wedge Q)| \leq |CL(Q)| + 1$.

The lemma indicates that the number of nodes in the NFG of a formula being conjunctive with a state formula is no more than the number of nodes in its own NFG plus one.

Lemma 3. For PPTL formulas P and Q, $|CL(P \vee Q)| \leq |CL(P)| + |CL(Q)| + 1$.

This lemma shows us that the number of nodes in the NFG of formula $P \vee Q$ is no more than the sum of numbers of nodes in NFGs of formulas P and Q plus one.

Lemma 4. For a PPTL formula P, $|CL(\neg P)| \leq |CL(P)| + 1$.

This lemma indicates the relationship between NFGs of P and $\neg P$.

Lemma 5. If P and Q are PPTL formulas, then $|CL(P\ ;Q)| \leq |CL(P)| + |CL(Q)| + 1$.

Similar to Lemma 3, this lemma shows us that the number of nodes in NFG of $P\ ;Q$ is no more than the sum of numbers of nodes in NFGs of P and Q plus one.

Lemma 6. If $|CL(P_i)| = k_i \in N_0$ $(1 \leq i \leq m)$ and $|CL(Q)| = k_0 \in N_0$, then $|CL((P_1, ..., P_m)\ prj\ Q)| = k \in N_0$.

This lemma predicates only that if the numbers of nodes in the NFGs of $P_1, ..., P_m$ and Q are finite then the number of nodes in the NFG of $(P_1, ..., P_m)\ prj$ Q is also finite.

Theorem 1. For any PPTL formula P, $|CL(P)| = k \in N_0$.

The theorem can be proved by Lemma 1 $\sim$ Lemma 6. It convinces us that for any PPTL formula P, the number of nodes in the NFG of P is finite. This is critical since it guarantees that algorithm *NFG* terminates. Furthermore, this enables us to develop a decision procedure for checking the satisfiability of PPTL formulas based on algorithm *NFG*. In addition, the NFG of a formula P contains all models of P, including both finite and infinite ones. Hence, our decision procedure not only gives an algorithm for checking the satisfiability of a formula P but also constructs all models of formula P.

5 Decision Procedure for PPTL Formulas

5.1 Path and Satisfiability

In the NFG of formula Q, a finite path, $\Pi = \langle Q, Q_e, Q_1, Q_{1e}, ..., \varepsilon\rangle$, is an alternating sequence of nodes and edges from the root to ε node, while an infinite path, $\Pi = \langle Q, Q_e, Q_1, Q_{1e}, ...\rangle$, is an infinite alternating sequence of nodes and edges emanating from the root. In fact, a path (finite or infinite) in the NFG of a formula Q corresponds to a model of Q. The fact is concluded in Theorem 2 and 3.

Theorem 2. A formula Q can be satisfied by finite models if and only if there exist finite paths in the NFG of Q.

Theorem 3. A formula Q can be satisfied by infinite models if and only if there exist infinite paths in the NFG of Q.

The proof of the Theorem 2 can be found in [9]. To prove Theorem 3, Lemma 7 and Lemma 8 are needed. The two lemmas are useful since they tell us the relationship between the models and paths in the NFG of a PPTL formula.

Lemma 7. Given a PPTL formula Q, if there exists an infinite path in NFG of Q, a corresponding infinite model of Q can be constructed.

Lemma 8. Given a PPTL formula Q, if there exists an infinite model of Q, a corresponding infinite path in NFG of Q can be found.

The proofs of Lemma 7 and Lemma 8 can be found in [10]. Theorem 2 and 3 confirm that a PPTL formula is satisfiable if and only if there exist finite or infinite paths in its NFG.

5.2 Decision Procedure

In an NFG, some nodes might have no successors (e.g. $\Box\bigcirc p\ ; q$). These nodes are redundant and can be removed. Algorithm *Simplify* is useful for eliminating redundant nodes of an NFG.

```
function $Simplify(G)$
/* precondition: $G = (CL(P), EL(P))$ is an NFG of PPTL formula $P$*/
/* postcondition: $Simplify(G)$ computes an NFG of $P$, $G' = (CL'(P), EL'(P))$,
which contains no redundant nodes*/
begin function
    $CL'(P) = CL(P)$; $EL'(P) = EL(P)$;
    while $\exists\ R \in CL'(P)$ and $R$ is not $\varepsilon$ and has no edges departing from
      do $CL'(P) = CL'(P) \setminus R$; $EL'(P) = EL'(P) \setminus \bigcup_i (R_i, R_e, R)$;
          /* $\bigcup_i (R_i, R_e, R)$ denotes the set of edges connecting to node $R$*/
    end while
    return $G'$;
end function
```

The algorithm is terminable since the nodes of an NFG is finite. In the simplified NFG of P, a finite or infinite path can readily be constructed. Obviously, P is satisfiable if and only if there exist finite or infinite paths in the NFG of P. Consequently, a decision procedure for checking the satisfiability of a PPTL formula P can be constructed based on the NFG of P. In the following, a skeleton of the procedure, algorithm *Check* in pseudo code, is given.

```
function $Check(P)$
/* precondition: $P$ is a PPTL formula*/
/* postcondition: $Check(P)$ checks whether formula $P$ is satisfiable or not.*/
begin function
    $G = NFG(P)$;
    $G' = Simplify(G)$;
    if $G'$ is empty, return unsatisfiable;
    else return satisfiable;
end function
```

Example. Check the satisfiability of formula $Q \equiv (skip,\ p\wedge\Box\bigcirc p,\ \varepsilon)\ prj\ \bigcirc^2 q$.

To check the satisfiability of formula $Q \equiv (skip,\ p\wedge\Box\bigcirc p,\ \varepsilon)\ prj\ \bigcirc^2 q$, NFG of formula Q can be constructed according to **function** *NFG*, and then simplified by **function** *Simplify*. The details of the procedure are shown as follows.

Step 1. Build NFG of formula Q (see Fig.2);

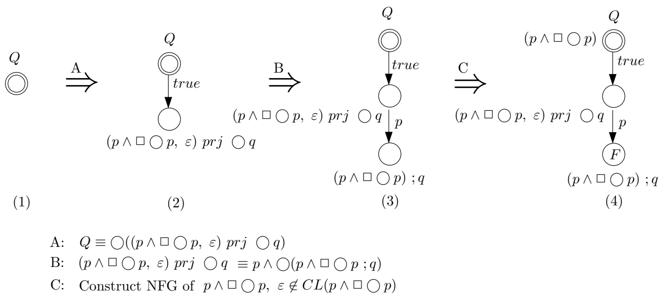
Q
A
true
$(p \wedge \Box \bigcirc p,\ \varepsilon)\ prj\ \bigcirc q$
B
p
$(p \wedge \Box \bigcirc p)\ ;q$
C
$(p \wedge \Box \bigcirc p)$
F
(1)
(2)
(3)
(4)
A: $Q \equiv \bigcirc((p \wedge \Box \bigcirc p,\ \varepsilon)\ prj\ \bigcirc q)$
B: $(p \wedge \Box \bigcirc p,\ \varepsilon)\ prj\ \bigcirc q \equiv p \wedge \bigcirc(p \wedge \Box \bigcirc p\ ;q)$
C: Construct NFG of $p \wedge \Box \bigcirc p,\ \varepsilon \notin CL(p \wedge \Box \bigcirc p)$

Fig. 2. Constructing the NFG of Q

Step 2. Obtain simplified NFG of formula Q (see Fig.3);

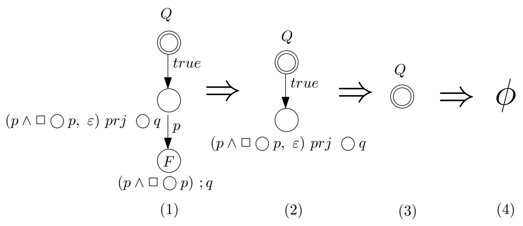
Q
true
$(p \wedge \Box \bigcirc p,\ \varepsilon)\ prj\ \bigcirc q$
p
F
$(p \wedge \Box \bigcirc p)\ ;q$
ϕ
(1)
(2)
(3)
(4)

Fig. 3. Simplifying the NFG of Q

Step 3. The simplified NFG of Q is empty, so formula Q is unsatisfiable.

6 Conclusion

In this paper, we have given a decision procedure for PPTL formulas with infinite models. A modified version of the decision algorithm can also be applied to PITL with infinite models [10]. This enables us to verify properties of concurrent systems with PPTL and PITL by means of model checking. However, the existing model checkers such as SPIN [16] and SMV [17] cannot directly be used to check PPTL or PITL formulas since PPTL and PITL are involved both finite and infinite models due to the chop operator while SPIN and SMV are only dealt with infinite models for PLTL and CTL formulas. Therefore, to check PPTL and PITL formulas, a model checker for PPTL and PITL is required. We believe that the decision procedures we give in this paper are useful in this respect. At the present, we have developed a model checker based on SPIN for PPTL. Further, we are also motivated to develop a practical verification environment with a set of supporting tools in the near future.

References

1. S.A. Kripke. Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9, 67-96, 1963.
2. R. Rosner and A. Pnueli. A choppy logic. First Annual IEEE Symposium on Logic In Computer Science, LICS, 306-314, 1986.
3. B.C. Moszkowski. *Reasoning about digital circuits.* Ph.D Thesis, Department of Computer Science, Stanford University. TRSTAN-CS-83-970,1983.
4. B. Moszkowski. A Complete Axiomatization of Interval Temporal Logic with Infinite Time. lics, p. 241, 15 th Annual IEEE Symposium on Logic in Computer Science (LICS'00), 2000.
5. Zhou Chaochen , C.A.R. Hoare and A.P. Ravn. A calculus of duration. Information Processing Letters 40(5): 269-275, 1991.
6. Z. Duan. Temporal Logic and Temporal Logic Programming. Science Press of China, 2006.
7. Z. Duan and M. Koutny. A Framed Temporal Logic Programming Language. Journal of Computer Science and Technology, Vol,19, No.3, pp.341-351, May, 2004.
8. Z. Duan, X. Yang and M. Koutny. Semantics of Framed Temporal Logic Programs. Proceedings of ICLP 2005, Barcelona, Spain, LNCS 3668, pp256-270, Oct. 2005.
9. Z. Duan and L. Zhang. A Decision Procedure for Propositional Projection Temporal Logic. Technical Report No.1, Institute of computing Theory and Technology, Xidian University, Xi'an P.R.China, 2005.
http://www.paper.edu.cn/process/download.jsp?file=200611-427
10. Z. Duan and C. Tian. Decison Prodedure for Propositional Projection Temporal Logic with Infinite Models. Technical Report No.1, Institute of computing Theory and Technology, Xidian University, Xi'an P.R.China, 2006.
http://www.paper.edu.cn/process/download.jsp?file=200611-444
11. Z. Duan, M. Koutny and C. Holt. Projection in temporal logic programming. In F. Pfenning (ed.), Proceedings of Logic Programming and Automatic Reasoning, LNAI, Springer-Verlag, vol 822, 333-344, 1994.
12. S. Kono. A combination of clausal and non-clausal temporal logic programs. In Lecture Notes in Artificial Intelligence, vol. 897, pages 40-57. SpringerVerlag, 1993.
13. H. Bowman and S. Thompson. A decision procedure and complete axiomatization of interval temporal logic with projection. Journal of logic and Computation 13(2),195-239, 2003.
14. Dutertre B. Complete proof systems for first order interval temporal logic. In Proceedings of LICS'95, (1995)36-43.
15. Z. Manna and A. Pnueli. The temporal logic of reactive and concurrent systems. Springer-Verlag, 1992.
16. Gerard J. Holzmann. The Model Checker Spin, IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.
17. K.L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers, 1993.
18. Hanpin Wang and Qiwen Xu, Completeness of Temporal Logics over Infinite Models, Discrete Applied Mathematics 136 (2004) 87-103, Elsevier B.V.
19. X. Yang Z. Duan, Operational Semantics of Framed Temporal Logic Programs, Proceedings of SOFSEM 2007, Harrachov, Czech, LNCS 4362,pp.566-578,Jan. 2007.