Acta Informatica (2008) 45:43–78
DOI 10.1007/s00236-007-0062-z

ORIGINAL ARTICLE

A decision procedure for propositional projection temporal logic with inPnite models

Zhenhua Duan · Cong Tian · Li Zhang

Received: 25 August 2006 / Revised: 11 October 2007 / Accepted: 15 October 2007 /
Published online: 20 November 2007
© Springer-Verlag 2007

Abstract This paper investigates the satis ability of Propositional Projection Temporal Logic (PPTL) with in nite models. A decision procedure for PPTL formulas is given. To this end, Normal Form (NF) and Labeled Normal Form Graph (LNFG) for PPTL formulas are de ned, and algorithms for transforming a formula to its normal form and constructing the LNFG for the given formula are presented. Further, the niteness of LNFGs is proved in details. Moreover, the decision procedure is extended to check the satis ability of the formulas of Propositional Interval Temporal Logic. In addition, examples are also given to illustrate how the decision procedure works.

1 Introduction

Interval Temporal Logic (ITL) [1,25] is a useful formalism for speci cation and veri cation of concurrent systems. In the past two decades, a number of axiom systems have been proposed [2–7] to verify properties of concurrent systems. Although veri cation of programs can be handled by complete deductive systems of ITL, the model checking approach which is an automatic method based on model checking algorithms has not extensively been studied in ITL.

Satis ability and validity of formulas are fundamental issues in the model theory of a logic. Moreover, satis ability plays an important role in the model checking approach since the negation of a property described in a temporal logic has to be satis able. Within ITL community, several researchers have looked at decision procedures. Halpern and Moszkowski

This research is supported by the NSFC Grant No. 60373103 and 60433010, and Defence Pre-Research Project of China, No. 51315050105.

Z. Duan (B) · C. Tian · L. Zhang
Institute of Computing Theory and Technology, Xidian University,
Xi'an 710071, People's Republic of China
e-mail: zhhduan@mail.xidian.edu.cn

C. Tian
e-mail: ctian@mail.xidian.edu.cn

[1,8] proved the decidability of Quanti er Propositional ITL (QPITL) over nite time. Kono presented a tableaux-based decision procedure for QPITL with projection [9]. Bowman and Thompson [10] presented the rst tableaux-based decision procedure for quanti er-free propositional ITL (PITL) over nite intervals with projection. To the best of our knowledge, all the existing decision procedures for ITL are con ned in nite intervals. However, many reactive systems are designed not to terminate. Accordingly, to verify those systems using model checking, a decision procedure with in nite models is required.

Projection Temporal Logic (PTL) [11–15,26,27] is a variation of ITL. It extends ITL to include in nite models and a new projection construct $(P_1, \ldots, P_m)\ prj\ Q$. The new projection construction can be treated as a combination of the parallel ($P \| Q$) and original projection construct ($P\ proj\ Q$ [16]. Formulas $P_1,\ldots,P_m$ 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 $P_1,\ldots,P_m$ (; denotes the chop operator, see Sect. 2), the communication between them is only at rendezvous states, and the formulas may terminate at different time points.

Compared with the $proj$ construct ($P\ proj\ Q$) de ned in [1,25], prj is more powerful. Firstly, $(P_1, \ldots, P_m)\ prj\ Q$ is able to handle terminal formulas (see Sect. 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 [5,10], Bowman and Thompson changed this by con ning P not to be a terminal formula. However, this restriction causes loss of exibility 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 de nition, formulas $P_1,\ldots, P_m$ and Q are autonomous; they are interpreted independently and may terminate at different time points. Thirdly, $P\ proj\ Q$ is only de ned over nite intervals and hard to be extended to in nite intervals. In contrast, our projection construct is de ned over both nite and in nite intervals. Finally, prj can subsume the central operator chop in ITL, $P; Q \quad (P, Q)\ prj\ \varepsilon$, but $proj$ cannot. The above points are explained with two motivation examples in Sect. 3.

ITL has also been extended into in nite time by Moszkowski [16]. However, the chop construct is yet different from the one in PTL in the case of in nite 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 rst part and Q holds on the second part (the interval can be nite or in nite), or P holds over the interval and the interval is in nite; whereas within PTL, $(P; Q)$ is true only for the rst case. That is, in PTL, P is only interpreted over a nite interval, while in ITL, P can be interpreted over an in nite interval. P; (Q) is not satis able in PTL if P has only in nite 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,

$$P ;_m Q \quad (P ;_d Q) \quad P \quad \Box\overset{\S}{\varepsilon}$$
$$P ;_d Q \quad (P \quad \Diamond\varepsilon) ;_m Q$$

where $\Box$ (always) and $\Diamond$ (sometimes) are modal operators and $\overset{\S}{\varepsilon}$ means the current interval is not over. In the sense of *Until* construct in Linear Temporal Logic (LTL) [17,29], $P ;_d Q$ 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 [12, 13], and a decision procedure for checking satis ability of Propositional Projection Temporal Logic (PPTL) formulas with nite models has been given in [18]. Nevertheless, to check the satis ability of the underlying logic with in nite models, a decision procedure is also required.

Therefore, we are motivated to investigate the satis ability of PPTL formulas with in nite models. To this end, Normal Form (NF) and and Labeled Normal Form Graph (LNFG) for PPTL formulas are de ned. Further, algorithms for transforming a formula to its normal form and constructing LNFG for the given formula are also presented. Basically, the normal form is the same as we gave in [11, 12, 15] for Tempura programs. The LNFG is a useful formalism for constructing models of a PPTL formula.

Accordingly, a decision procedure based on LNFGs for checking the satis ability of PPTL formulas with in nite models is presented in the paper. With this method, for a given formula, an LNFG can be constructed by means of its normal form. A nite path from the root node to the ε node (i.e., terminating node) in the LNFG of the formula corresponds to a nite model of the formula while an in nite path emanating from the root corresponds to an in nite model of the formula. Further, we can prove that a formula is satis able if and only if there exists a nite or in nite path in its LNFG. In addition, the decision procedure can also be used to check the satis ability of PITL formulas with minor changes (see Sect. 7).

Compared with Bowman and Thompson's work [5, 10], our method has some advantages. For instance, it can be applied to check the satis ability of PITL formulas with both nite and in nite models, while Bowman's method can deal with only nite models. In addition, our approach is based on LNFGs whereas Bowman's approach is based on Tableaux graphs. In a sense, our approach is more concise than Bowman's since an LNFG for a formula is much smaller than the Tableaux graph (see Sect. 7).

The paper is organized as follows. The next section brie y presents the syntax, semantics and some logic laws of the underlying logic. Two motivation examples are demonstrated in Sect. 3. Section 4 gives the de nition of the normal form of PPTL formulas, and the algorithm NF for transforming a PPTL formula to its normal form. In Sect. 5, the normal form graph is introduced; further, labeled normal form graph is de ned and an algorithm for constructing LNFGs is described, and the niteness of LNFGs is proved in details. A decision procedure for checking the satis ability of PPTL formulas with in nite models is demonstrated in Sect. 6. Section 7 is devoted to developing a decision procedure for PITL formulas. Finally, conclusions are drawn in Sect. 8.

2 Propositional projection temporal logic

Our underlying logic is Propositional Temporal Logic with projection [11–13]. It is a variation of Propositional Interval Temporal Logic [1].

2.1 Syntax

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

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

where $p \in Prop$, $P_1, \ldots, P_m$ and P are all well-formed PPTL formulas, $\bigcirc$ (next) and prj (projection) are basic temporal operators.

2.2 Semantics

1. States
Following the de nition of Kripke's structure [19], we de ne a state s over $Prop$ to be a mapping from $Prop$ to $B = \{true, false\}$, $s : Prop \to B$. We will use $s[p]$ to denote the valuation of p at state s.

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

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 $0 \leq r_1 \leq r_2 \leq \cdots \leq r_h \leq |\sigma|$. The projection of σ onto $r_1, \ldots, r_h$ is the interval (namely projected interval)

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

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

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

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

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

- $\mathcal{I} \models p$ iff $s_k[p] = true$, for any given proposition p
- $\mathcal{I} \models \neg P$ iff $\mathcal{I} \not\models P$
- $\mathcal{I} \models P \vee Q$ iff $\mathcal{I} \models P$ or $\mathcal{I} \models Q$
- $\mathcal{I} \models \bigcirc P$ iff $k < j$ and $(\sigma, k+1, j) \models P$
- $\mathcal{I} \models (P_1, \ldots, P_m)\ prj\ Q$ if there exist integers $k = r_0 \leq r_1 \leq \cdots \leq r_m \leq j$ such that $(\sigma, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_l) \models P_l, 1 < l \leq m$, and $(\sigma\downarrow, 0, |\sigma\downarrow|) \models Q$ for one of the following $\sigma\downarrow$:
  (a) $r_m < j$ and $\sigma\downarrow = \sigma\downarrow(r_0, \ldots, r_m) \cdot \sigma_{(r_m+1 \ldots j)}$ or
  (b) $r_m = j$ and $\sigma\downarrow = \sigma\downarrow(r_0, \ldots, r_h)$ for some $0 \leq h \leq m$

Figure 1 shows the possible semantics of $(P_1, P_2)\ prj\ Q$. Here Q and P_1 start to be interpreted at state s_0; subsequently, P_1 and P_2 are interpreted sequentially, Q is interpreted in parallel with $(P_1; P_2)$ over the interval consisting of endpoints of subintervals over which P_1 and P_2 are interpreted. The possible three cases are given: (a) P_2 terminates before Q; (b) Q and P_2 terminate at the same state; (c) Q terminates before P_2.

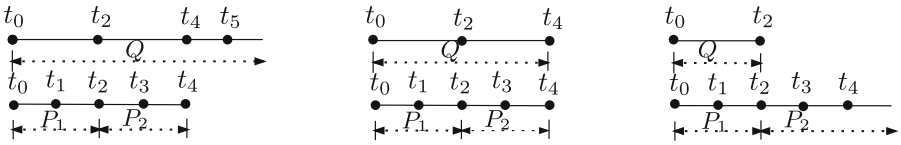


Fig. 1 Semantics of $(P_1, P_2)\ prj\ Q$

Compared with the original projection operator $P\ proj\ Q$ in [1], terminal formulas (i.e., formulas holding only at singleton sates, see below for details) can be handled with our projection construct. For instance, the semantics of $(\varepsilon, \varepsilon, len\ 2, \varepsilon, \varepsilon, len\ 1)\ prj\ len\ 2$ can be depicted as shown in Fig. 2. Actually, for projection construct $(P_1, \ldots, P_m)\ prj\ Q$, if there exist terminal formulas in $P_1, \ldots, P_m$, it cannot be expressed by $proj$ construction.

2.3 Abbreviations

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

$$\begin{array}{ll}
\varepsilon \stackrel{\text{def}}{=} \neg\ true & \check{S}\varepsilon \stackrel{\text{def}}{=} \neg\ \varepsilon \\
^0 P \stackrel{\text{def}}{=} P & ^n P \stackrel{\text{def}}{=} (\ ^{n\check{S}1} P) \\
len\ n \stackrel{\text{def}}{=}\ ^n \varepsilon & skip \stackrel{\text{def}}{=} len\ 1 \\
\odot P \stackrel{\text{def}}{=} \varepsilon \wedge P & P;\ Q \stackrel{\text{def}}{=} (P,\ Q)\ prj\ \varepsilon \\
\Diamond P \stackrel{\text{def}}{=} true;\ P & \Box P \stackrel{\text{def}}{=} \neg \Diamond \neg P
\end{array}$$

where $\odot$ (weak next), $\Box$ (always), $\Diamond$ (sometimes), and $;$ (chop) are derived temporal operators; ε (empty) denotes an interval with zero length, and $\check{S}$ (more) means the current state is not the nal one over an interval.

With projection construct $(P_1, \ldots, P_m)\ prj\ Q$, in some circumstances, there may exist some parts, such as $P_i, \ldots, P_j$, that can repeatedly appear in $P_1, \ldots, P_m$ for several times. In this case, for concise, the projection construct can be described as follows,

$$\begin{array}{l}
(P_1, \ldots, (P_i, \ldots, P_j)^k, \ldots, P_m)\ prj\ Q \\
\stackrel{\text{def}}{=} (P_1, \ldots, \underbrace{(P_i, \ldots, P_j), \ldots, (P_i, \ldots, P_j)}_{k \text{ times}}, \ldots, P_m)\ prj\ Q
\end{array}$$

Fig. 2 A projected interval

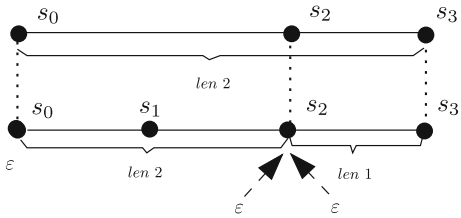


Table 1 Precedence rules

Precedence	Operators
1	¬
2	, ⊙, ◇, □
3	,
4	,
5	prj, ;

2.4 Precedence rules

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

2.5 Satisfaction and validity

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

Two formulas, P and Q, are equivalent if $\models \Box(P \equiv Q)$, and we denote this by $P \equiv Q$. A formula P is called a state formula if it contains no temporal operators, a terminal formula if $P \equiv P \frown \varepsilon$, a non-local formula if $P \neq P \frown \varepsilon$, and a local formula if P is a state or terminal formula.

2.6 Logic laws

Table 2 shows some useful logic laws. Note that w is a state formula.

The proofs of the logic laws can be found in [11] [12].

3 Applications of projection construct

In this section, two examples are given to demonstrate how the projection construct works in practice.

The first example is a pulse generator for boolean variable x which can assume two values: 0 (low) and 1 (high). We first define two types of processes: The first one is $hold(i)$ $(i \geq 1)$ which is interpreted over an interval of length i and ensures that the value of x remains constant in all but the final state:

$$hold(i) \stackrel{\text{def}}{=} len\ i \wedge \Box(more \rightarrow (x = x))$$

The other is $switch(j)$ which ensures that the value of x is first set to 0 and then changed at every subsequent state:

$$switch(j) \stackrel{\text{def}}{=} (x = 0) \wedge len\ j \wedge \Box(more \rightarrow (x = \neg x))$$

Having defined $hold(i)$ and $switch(j)$ we can define pulse generators with varying number and length of low and high intervals for x:

$$pulse(i_1, \ldots, i_k) \stackrel{\text{def}}{=} (hold(i_1), \ldots, hold(i_k))\ prj\ switch(k)$$

Table 2 Logic laws for PPTL

L_1	$\Box(P \quad Q) \quad \Box P \quad \Box Q$
L_2	$\Diamond(P \quad Q) \quad \Diamond P \quad \Diamond Q$
L_3	$(P \quad Q) \quad P \quad Q$
L_4	$(P \quad Q) \quad P \quad Q$
L_5	$R;(P \quad Q) \quad (R; P) \quad (R; Q)$
L_6	$(P \quad Q); R \quad (P; R) \quad (Q; R)$
L_7	$\Diamond P \quad P \quad \Diamond P$
L_8	$\Box P \quad P \quad \odot \Box P$
L_9	$\overset{\check{S}}{\varepsilon} \neg \quad P \quad \overset{\check{S}}{\varepsilon} \neg \quad P$
L_{10}	$\neg \odot P \quad \neg \quad P$
L_{11}	$P; Q \quad (P ; Q)$
L_{12}	$w \quad (P ; Q) \quad (w \quad P) ; Q$
L_{13}	$Q \ prj \ \varepsilon \quad Q$
L_{14}	$\varepsilon \ prj \ Q \quad Q$
L_{15}	$(P_1, \ldots P_m) \ prj \ \varepsilon \quad P_1; \ldots; P_m$
L_{16}	$(P, \varepsilon) \ prj \ Q \quad (P \quad \Diamond\varepsilon) \ prj \ Q$
L_{17}	$(P_1, \ldots, P_t, w \quad \varepsilon, P_{t+1}, \ldots, P_m) \ prj \ Q \quad (P_1, \ldots, P_t, w \quad P_{t+1}, \ldots, P_m) prj Q$
L_{18}	$(P_1, \ldots, (P_i \quad P_i), \ldots, P_m) \ prj \ Q \quad ((P_1, \ldots, P_i, \ldots, P_m) \ prj \ Q) \quad ((P_1, \ldots, P_i, \ldots, P_m) \ prj \ Q)$
L_{19}	$(P_1, \ldots, P_m) \ prj \ (P \quad Q) \quad ((P_1, \ldots, P_m) \ prj \ P) \quad ((P_1, \ldots, P_m) \ prj \ Q)$
L_{20}	$(P_1, \ldots, P_m) \ prj \quad Q \quad (P_1 \ \overset{\check{S}}{\varepsilon}; (P_2, \ldots, P_m) \ prj \ Q) \quad (P_1 \quad \varepsilon ; (P_2, \ldots, P_m) \ prj \quad Q)$
L_{21}	$(\quad P_1, \ldots, P_m) \ prj \quad Q \quad (P_1 ; (P_2, \ldots, P_m) \ prj \ Q)$
L_{22}	$(w \quad P_1, \ldots, P_m) \ prj \ Q \quad w \quad ((P_1, \ldots, P_m) \ prj \ Q)$
L_{23}	$(P_1, \ldots, P_m) \ prj \ (w \quad Q) \quad w \quad ((P_1, \ldots, P_m) \ prj \ Q)$

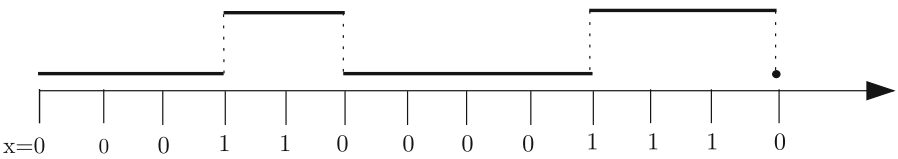


Fig. 3 The pulse generator

Let $k = 4, i_1 = 3, i_2 = 2, i_3 = 4$ and $i_4 = 3$, the generated pulse is illustrated as shown in Fig. 3 and the model is depicted in Fig. 4.

The second example is a logical circuit for the $3x + 1$ conjecture. Let x be an integer and the function $T(x)$ equal to $3x + 1$ if x is odd and $x/2$ if x is even. The conjecture asserts that starting from any positive integer x, the repeated iteration of $T(x)$ eventually produces 1. Figure 5 illustrates the logical circuit for the $3x + 1$ conjecture. The whole circuit is composed of a decider (Decider-O/E) for checking whether an integer is odd or even, a calculator (Mul3&Add1) for multiplying an integer by 3 and then plus 1, a divisor (Div-2) for dividing a number by 2, and another decider (Decider-1) to check whether an integer is

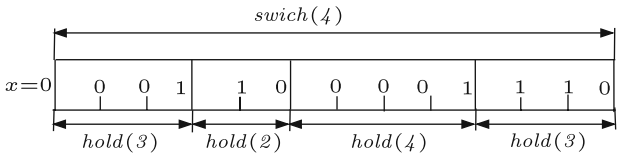


Fig. 4 The model of the pulse generator

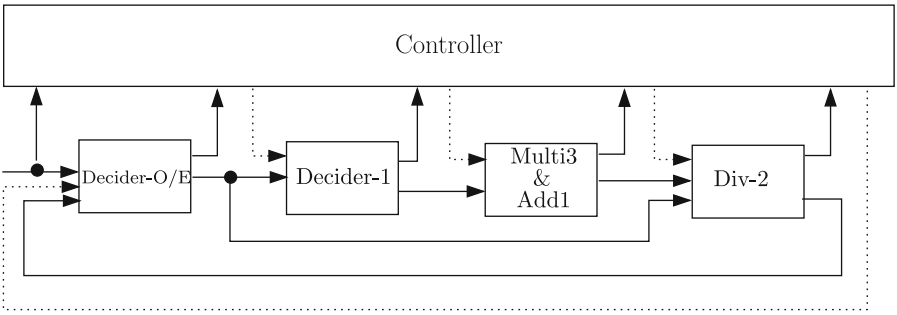


Fig. 5 Logical circuit for $3x + 1$ conjecture

1 or not. Also a controller is employed to make the Decider-O/E, Mul3&Add1, Div-2, and Decider-1 work cooperatively.

If an integer is input to the circuit, the Decider-O/E is activated while the controller is active at the same time. If the input integer is even, 0 is returned to the controller otherwise 1 is returned. When the controller receives 1 from Decider-O/E, Decider-1 is activated to check whether the integer is 1 or not otherwise Div-2 is enabled. When Decider-1 works, it sends 1 to the controller if its input is 1, otherwise its input is output and 0 is returned to the controller. Receiving 1 from Decider-1, the controller terminates the computation normally otherwise Multi3&Add1 is active. If Multi3&Add1 works, it outputs the result and sends the over ow information to the controller; if no over ow happens, Div-2 is enabled. Similar to Multi3&Add1, it outputs the result and returns the over ow information. In the normal case, Decider-O/E is enabled again by the controller. During the process at any stage, if an over ow occurs the controller terminates the computation abnormally.

For example, when 5 is input to the circuit, Decider-O/E and the controller are enabled at the same time; then 1 is returned to the controller; receiving 1 from Decider-O/E, the controller activates Decider-1; Decider-1 outputs 5 and returns 0 to the controller; then Multi3&Add1 works, 16 is returned and Div-2 is enabled; subsequently, 8, 4, 2, and 1 are output from Div-2 and then Decider-E/O is enabled and 1 is returned to the controller; nally, 1 is input to Decider-1 and the computation is completed.

Let Q, P_1, P_2, P_3 and P_4 denote the controller, Decider-O/E, Decider-1, Multi3&Add1 and Div-2 respectively. The above computation can be speci ed as follows:

$$(P_1, P_2, P_3, P_4, (P_1, P_4)^4, P_2)\ prj\ Q$$

In practice, the controller, Decider-O/E, Multi3&Add1 and Div-2 can be realized by multi gates with large delay while Decider-1 can be realized by a single gate with very small delay. So, the latter delay can be ignored. Thus, Q, P_1, P_3 and P_4 can be treated as non-terminal formulas and P_2 a terminal formula. Without loss generality, we assume that P_1, P_2, P_3 and

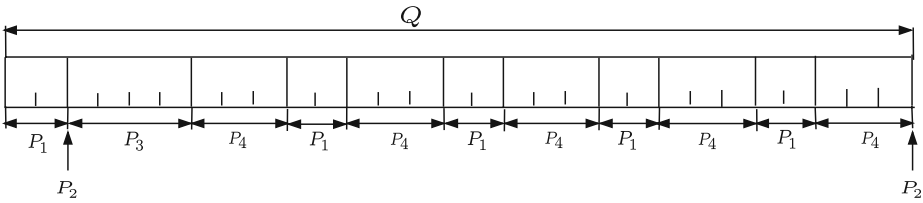


Fig. 6 The computational model for 3 + 1 with input 5

P_4 have delays len 2, len 4 and len 3 respectively. Q works over the projected interval with length len 11. The computation can be depicted as shown in Fig. 6.

4 Normal form of PPTL

In this section, we present a normal form for the purpose of a decision procedure for checking the satis ability of PPTL formulas.

Recall that in classical propositional logic, a basic product (resp. sum) is a conjunction (resp. disjunction) of literals, where a literal is either p or $\neg p$ for some atomic proposition p. A state formula is said to be in disjunctive (resp. conjunctive) normal form (DNF) (resp. CNF) if it is the sum, "or" in logic (resp. product, "and" in logic) of products (resp. sum) of literals. A basic product (resp. sum) is called min-product (resp. max-sum) if, for all atomic propositions, either a proposition or its negation but not both appears in the product (resp. sum). If there are n atomic propositions then we have 2^n min-products, $m_0, m_1, \ldots, m_{2^n-1}$ and 2^n max-sums, $M_0, M_1, \ldots, M_{2^n-1}$. The conjunction of any two different min-products is $false$ and the disjunction of all min-products is $true$ (in this sense, we say that the set of all min-products is complete). That is,

$$\bigvee_i m_i \equiv true \tag{4.1}$$

$$\bigvee_{i=j} (m_i \wedge m_j) \equiv false \tag{4.2}$$

Thus, for any formula Q_i, it is not dif cult to see the following property:

$$\neg \bigvee_i (m_i \wedge Q_i) \equiv \bigvee_i (m_i \wedge \neg Q_i) \tag{4.3}$$

The property can be easily generalized to a set of formulas $< P_i >_i = \{P_1, \ldots, P_i\}$. We call $< P_i >_i$ complete if $< P_i >_i$ satis es the properties,

$$\bigvee_i P_i \equiv true$$

$$\bigvee_{i=j} (P_i \wedge P_j) \equiv false$$

In this case, it is readily to show that (4.3) is still valid,

$$\neg \bigvee_i (P_i \wedge Q_i) \equiv \bigvee_i (P_i \wedge \neg Q_i) \tag{4.4}$$

This idea has been used by Kono [8], [Bowman and Thompson 10]. For convenience, let $Q_p \subseteq Prop$ be the set of all atomic propositions in formula Q. We now define the normal form of PPTL formulas as follows.

Definition 1 Let Q be a PPTL formula and Q_p denote the set of atomic propositions appearing in Q. Q is in normal form if Q has been rewritten as

$$Q \equiv \bigvee_{j=1}^{n_0}(Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{n}(Q_{ci} \wedge \bigcirc Q_i) \tag{4.5}$$

where $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} q_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} q_{ih}$, $l = |Q_p|$, $1 \le n \le 3^l$, $1 \le n_0 \le 3^l$ (it is an exercises to prove the upper bound), $1 \le m \le l$, $1 \le m_0 \le l$; $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, r means r or $\neg r$; Q_i is a general PPTL formula. 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,

$$Q \equiv (Q_e \wedge \varepsilon) \vee \bigvee_{i=1}^{r}(Q_i \wedge \bigcirc Q_i) \tag{4.6}$$

where Q_e and Q_i are state formulas.

Definition 2 A normal form, $Q \equiv (Q_e \wedge \varepsilon) \vee \bigvee_{i=1}^{n}(Q_i \wedge \bigcirc Q_i)$, for a PPTL formula Q is called a complete normal form if $\bigvee_i Q_i \equiv true$ and $\bigvee_{i \ne j}(Q_i \wedge Q_j) \equiv false$.

Theorem 1 *If a formula P has been rewritten to its normal form, then P can be rewritten to its complete normal form.*

Proof Suppose P has been rewritten to its normal form, $P \equiv P_e \wedge \varepsilon \vee \bigvee_{i=1}^{n}(P_i \wedge \bigcirc P_i)$. We can treat P_i as atoms and constructed 2^n min-products, $m_0, \ldots, m_{2^n-1}$, whereas we can treat P_i as atoms and construct 2^n max-sum $M_0, \ldots, M_{2^n-1}$. From these max-sums, we can obtain 2^n basic sums namely $M_0, \ldots, M_{2^n-1}$. Here M_j denotes $M_{2^n-1} \setminus M_j$ by deleting negative disjuncts and M_{2^n-1} is $false$. Thus, it is ready to prove that,

$$\begin{aligned} P &\equiv P_e \wedge \varepsilon \vee \bigvee_{i=1}^{n}(P_i \wedge \bigcirc P_i) \\ &\equiv P_e \wedge \varepsilon \vee \bigvee_{i=0}^{2^n-1}(m_i \wedge \bigcirc M_i) \end{aligned} \tag{4.7}$$

Obviously, P is now in its complete normal form.

Theorem 1 illustrates how a normal form can be rewritten to its complete normal form that plays an important role for transforming a formula in the negation form into its normal form. The following is an example for rewriting a normal form into its complete normal form.

Example 1 Rewrite normal form $(p \wedge \bigcirc P) \wedge (q \wedge \bigcirc Q)$ to its complete normal form.

$$\begin{aligned} &(p \wedge \bigcirc P) \wedge (q \wedge \bigcirc Q) \equiv ((p \wedge q) \wedge \bigcirc (P \wedge Q)) \vee ((p \wedge \neg q) \wedge \bigcirc P) \\ &\qquad \vee ((\neg p \wedge q) \wedge \bigcirc Q) \vee ((\neg p \wedge \neg q) \wedge false) \end{aligned}$$

In the following, Lemma 1 is concerned with how to transform the projection construct into the normal form.

Lemma 1 *Let* $R \quad (P_1, \ldots, P_m)\ prj\ Q$. *Suppose* $P_1, \quad , P_m$ *and* Q *have been rewritten to their normal forms, then* R *can be rewritten to its normal form.*

Proof The proof proceeds by induction on m. Suppose normal forms of P_1 and Q are presented as follows:

$$P_1 \quad P_{1e} \quad \varepsilon \quad \bigvee_{i=1}^{n} (P_{1i} \qquad P_{1i})$$

$$Q \quad Q_e \quad \varepsilon \quad \bigvee_{k=1}^{n} (Q_k \qquad Q_k)$$

Then,

$$P_1\ prj\ Q \quad P_{1e} \quad Q_e \quad \varepsilon \quad \bigvee_{i=1}^{n} (P_{1i} \quad Q_e \qquad P_{1i})$$

$$\bigvee_{i=1}^{n} \bigvee_{k=1}^{n} (P_{1i} \quad Q_k \qquad (P_{1i};\ Q_k))$$

$$\bigvee_{k=1}^{n} (P_{1e} \quad Q_k \qquad Q_k)$$

Suppose $(P_2, \ldots, P_m)\ prj\ Q$ has been rewritten to its normal form,

$$(P_2, \ldots, P_m)\ prj\ Q \quad R_e \quad \varepsilon \quad \bigvee_{j=1}^{t} (R_j \qquad R_j)$$

Thus,

$$(P_1, \ldots, P_m)\ prj\ Q$$

$$P_{1e} \quad R_e \quad \varepsilon \quad \bigvee_{j=1}^{t} (P_{1e} \quad R_j \qquad R_j)$$

$$\bigvee_{i=1}^{n} \bigvee_{k=1}^{n} (P_{1i} \quad Q_k \qquad (P_{1i};\ ((P_2, \ldots, P_m)\ prj\ Q_k)))$$

$$\bigvee_{i=1}^{n} (P_{1i} \quad Q_e \qquad (Q_{1i};\ P_2;\ \ldots;\ P_m))$$

Theorem 2 *For any PPTL formula* R, R *can be rewritten to its normal form.*

Proof The proof proceeds by induction on the structure of PPTL formulas.

Base: If $R \quad p$ and p is an atomic proposition, we have,

$$R \quad p \quad (\varepsilon \quad \overset{\check{S}}{\varepsilon}) \quad p \quad \varepsilon \quad p \qquad true$$

Induction: Suppose P and Q can be rewritten to their normal forms as follows,

$$P \quad P_e \quad \varepsilon \quad \bigvee_{i=1}^{n} (P_i \quad P_i)$$

$$Q \quad Q_e \quad \varepsilon \quad \bigvee_{j=1}^{n} (Q_j \quad Q_j)$$

Then,

1. If $R \quad P \quad Q$, then the proof is straightforward.
2. If $R \quad P$, R has been in its normal form.
3. If $R \quad (P_1, \ldots, P_m)\ prj\ Q$, by Lemma 1, the conclusion holds.
4. If the formula R is in the form $\neg P$, by Theorem 1, we assume P is in the complete normal form,

$$P \quad P_e \quad \varepsilon \quad \bigvee_{i=1}^{n} (P_i \quad P_i)$$

Thus,

$$\neg P \quad \neg \quad P_e \quad \varepsilon \quad \bigvee_{i=1}^{n} (P_i \quad \neg \quad P_i) \tag{4.8}$$

We now give the algorithm NF for transforming a PPTL formula into its normal form. The basic idea of the algorithm is the same as the above proofs. Accordingly, the above theorem and lemmas ensure the correctness of algorithm NF. Given a formula R as input, we first do some preprocesses by PRE(R) to eliminate all implications, equivalences, double negations, $skip$, $\hat{S}\varepsilon$, $len(n)$ and $\Diamond$ in R by replacing all sub-formulas of the form $P \quad Q$ by $\neg P \quad Q$, $P \quad Q$ by $\neg P \quad \neg Q \quad P \quad Q$, $\neg\neg P$ by P, $skip$ by ε, $\hat{S}\varepsilon$ by $true$, $len\ n$ by $^{n}\varepsilon$ and $\Diamond P$ by $true; P$. Algorithm DNF is used to translate a state formula into its disjunctive normal form. And algorithm NFDNF will transform a disjunction normal form by replacing all basic products $\bigwedge_{j=1}^{m} q_{ij}$ by $\bigwedge_{j=1}^{m} q_{ij} \quad \varepsilon \quad \bigwedge_{j=1}^{m} q_{ij} \quad true$. We say that a formula R is in PRE if P has been processed by PRE(P). In the following algorithms, w denotes a state formula and μ a basic product. We can now write pseudo codes for algorithms as shown in Table 3.

In algorithm NF(R), the basic constructs such as atomic propositions, the next, disjunction, negation and projection constructs need to be considered. However, to improve the efficiency, other constructs such as always, and chop are also taken into account. Intuitively, if R is $true$, $false$ or a state formula, the transformation is straightforward; if R is $\mu \quad P$ or $\mu \quad \varepsilon$, R is already in its normal form; if R is $P \quad Q$ the algorithm only needs to transform P and Q into the normal form; if R is $(P_1, \ldots, P_m)\ prj\ Q$ (resp. $P; Q$) the algorithm calls function PRJ (resp. function CHOP) to rewrite it; in addition, $\Box P \quad P \quad \varepsilon \quad P \quad \Box P$, so the algorithm needs to transform $P \quad \varepsilon$, $P \quad \Box P$ into the normal forms; for $P \quad Q$, the algorithm transforms P and Q into the normal forms at first, then rewrites $P \quad Q$ into its normal form by algorithm AND.

Algorithms PRJ and CHOP are straightforward. They consider all possible cases and use the corresponding logic laws to rewrite them as shown in Tables 4 and 5 respectively.

For formula $\neg P$, we first rewrite formula P into its normal form, then we convert it into its complete normal form by algorithm CONF which is formalized based on (4.7). After that,

Table 3 Algorithm for translating a PPTL formula to its normal form

Function NF(R)
/* precondition: R is a PPTL formula and in PRE* /
/* postcondition: NF(R) computes an equivalent normal form for formula R */

begin function
  case
    R is $true$: return $\varepsilon \quad true$;
    R is $false$: return $false$;
    R is a state formula: return NFDNF(DNF(R));
    R is $\mu \quad P$: return R;
    R is $\mu \quad ^{n} P$: return $\mu \quad (\ ^{n\check{S}1}) P$;
    R is $\mu \quad \varepsilon$: return R;
    R is $P \quad Q$: return NF(P) NF(Q);
    R is $(P_1, \ldots, P_m)\ prj\ Q$: return PRJ($R$);
    R is $\neg P$: return NEG(CONF(NF(P)));
    R is $\Box P$: return NF($P \quad \varepsilon$) NF($P \quad \Box P$);
    R is $P \quad Q$: return AND(NF(P),NF(Q));
    R is $P; Q$: return CHOP(R);
  end case
end function

Table 4 Algorithm for translating projection constructs to normal form

Function PRJ(R)
/* precondition: R is $(P_1, \ldots, P_m)\ prj\ Q$ */
/* postcondition: PRJ(R) computes an equivalent NF for formula R */

begin function
  case
    R is $\varepsilon\ prj\ \varepsilon$: return ε;
    R is $\varepsilon\ prj\ Q$: return NF(Q);
    R is $P\ prj\ \varepsilon$: return NF(P);
    R is $P\ prj \quad Q$: return $(P; Q)$;
    R is $(\ P_1, \ldots, P_m)\ prj \quad Q$: return $(P_1; ((P_2, \ldots, P_m)\ prj\ Q))$;
    R is $(\varepsilon, P_2, \ldots, P_m)\ prj\ Q$: return PRJ($(P_2, \ldots, P_m)\ prj\ Q$);
    R is $(P_1, \ldots, P_m)\ prj\ \varepsilon$: return $(P_1; (P_2; (\ldots; P_m)))$;
    R is $(w \quad P_1, \ldots, P_m)\ prj\ Q$: return NF($w \quad$ PRJ($(P_1, \ldots, P_m)\ prj\ Q$));
    R is $(P_1, \ldots, P_m)\ prj\ (w \quad Q)$: return NF($w \quad$ PRJ($(P_1, \ldots, P_m)\ prj\ Q$));
    R is $(P_1 \quad P_1, \ldots, P_m)\ prj\ Q$: return PRJ($(P_1, \ldots, P_m)\ prj\ Q$)
      PRJ($(P_1, \ldots, P_m)\ prj\ Q$);
    R is $(P_1, \ldots, P_m)\ prj\ (Q \quad Q\ )$: return PRJ($(P_1, \ldots, P_m)\ prj\ Q$)
      PRJ($(P_1, \ldots, P_m)\ prj\ Q\ )$;
    otherwise: return PRJ((NF(P_1), $\ldots, P_m)\ prj$ NF(Q));
  end case
end function

Table 5 Algorithm for translating chop constructs to normal form

Function CHOP(R)
/* precondition: R is P; Q* /
/* postcondition: CHOP(R) computes an equivalent NF for formula R* /

begin function
  case
    R is ε; Q: return NF(Q);
    R is P; Q: return (P; Q);
    R is w P; Q: return NF(w CHOP(P; Q));
    R is P_1 P_2; Q: return CHOP(P_1; Q) CHOP(P_2; Q);
    otherwise: return CHOP(NF(P); Q);
  end case
end function

Table 6 Algorithm for transforming a normal form to its complete normal form

Function CONF(R)
/* precondition: R is in normal form, m_i and M_i are de ned as in Theorem 3.1* /
/* postcondition: CONF(R) computes an equivalent complete normal form for R* /

begin function
  case
    R is P P : return P P $\neg$ P $false$;
    R is P P $\bigvee_{i=1}^{n}(m_i$ $M_i)$: return $\bigvee_{i=1}^{n}(P$ m_i $(P$ $M_i))$;
      $\bigvee_{i=1}^{n}(\neg P$ m_i $M_i)$;
    R is P_e ε $\bigvee_{i=1}^{n}(P_i$ $P_i)$: return P_e ε CONF(P_1 P_1
      CONF($\bigvee_{i=2}^{n}(P_i$ $P_i)$));
  end case
end function

Table 7 Algorithm for rewriting the negation of a complete normal form

Function NEG(R)
/* precondition: R R_e ε $\bigvee_{i=1}^{n}(m_i$ $M_i)$, is in normal form* /
/* postcondition: NEG(R) computes an equivalent normal form for $\neg R$* /

begin function
  return $\neg R_e$ ε $\bigvee_{i=1}^{n}(m_i$ $\neg$ $M_i)$
end function

according to (4.8), $\neg P$ can be rewritten to its normal form. The ideas are shown in algorithm CONF and NEG respectively (see Tables 6 and 7).

In algorithm AND, we assume that P and Q are in the normal forms. So P (resp. Q) is in the form of P_1 P_2, P_e ε, or P_i P_i. If P is in the form of P_1 P_2, then algorithm AND deals with P_1 Q P_2 Q and transforms P_1 Q and P_2 Q by calling itself. If P P_e ε, Q Q_e ε or P P_i P_i, Q Q_i Q_i then the transforming

Table 8 Algorithm for transforming *and construct* to the normal form

```
Function AND(P, Q)
/* precondition: P and Q are in normal form */
/* postcondition: AND(P, Q) computes an equivalent NF for P ∧ Q */
begin function
  case
    P is P1 ∨ P2: return AND(P1, Q) ∨ AND(P2, Q);
    Q is Q1 ∨ Q2: return AND(P, Q1) ∨ AND(P, Q2);
    P is Pe ∧ ε and Q is Qe ∧ ε: return Pe ∧ Qe ∧ ε;
    P is Pi ∧ Pi and Qj ∧ Qj: return Pi ∧ Qj ∨ (Pi ∧ Qj);
    P is Pe ∧ ε, Q is Qj ∧ Qj or P is Pi ∧ Pi, Q is Qe ∧ ε: return false;
  end case
end function
```

is straightforward. If $P = P_e \wedge \varepsilon$, $Q = Q_i \wedge Q_i$ or $P = P_i \wedge P_i$, $Q = Q_e \wedge \varepsilon$ then $P \wedge Q = false$. The algorithm in pseudo code is illustrated in Table 8.

Obviously, the normal form enables us to rewrite a formula into two parts: the present and future ones. The present part is a state formula while the future part is either a next formula. This standard generic form inspires us to construct a graph for describing the models of a formula. The following is an example to illustrate how a PPTL formula can be transformed to its normal form by algorithm NF.

Example 2 Transform formula $^2(\Box p; q) \wedge (p; \Box q)$ to its normal form by means of algorithm NF.

$$
\begin{aligned}
&\text{NF}(^2(\Box p; q) \wedge (p; \Box q))\\
&\quad \text{NF}(^2(\Box p; q)) \wedge \text{NF}(p; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge \text{CHOP}(p; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge \text{CHOP}(\text{NF}(p); \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge \text{CHOP}(p \wedge \varepsilon \vee p \wedge true; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge \text{CHOP}(p \wedge \varepsilon; \Box q) \vee \text{CHOP}(p \wedge true; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge p \wedge \text{CHOP}(\varepsilon; \Box q) \vee p \wedge \text{CHOP}(true; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge p \wedge (\text{NF}(q \wedge \varepsilon) \vee \text{NF}(q \wedge \Box q))\\
&\quad p \wedge (true; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge p \wedge (false \vee (q \wedge \Box q)) \vee p \wedge (true; \Box q)\\
&\quad\quad (\text{NF}(\Box p; q)) \wedge p \wedge (q \wedge \Box q) \vee p \wedge (true; \Box q)
\end{aligned}
$$

5 Labeled Normal Form Graph

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

5.1 Normal Form Graph

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 the graph. In $CL(P)$,

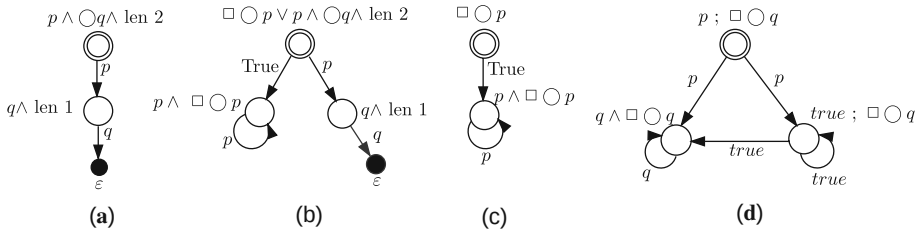


Fig. 7 Examples of NFGs

each node is speci ed by a formula in PPTL, while in $EL(P)$, each edge is a directed arc labeled with a state formula Q_e from node Q to node R and identi ed by a triple (Q, Q_e, R). $CL(P)$ and $EL(P)$ of G can be inductively de ned as in De nition 3. Note that the normal form employed in this de nition is normal form (4.5).

De nition 3 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 de ned as follows:

1. $P \in CL(P)$;
2. For all $Q \in CL(P) \setminus \{\varepsilon, false\}$, if $Q = \bigvee_{j=1}^{h}(Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^{k}(Q_{ci} \wedge \bigcirc Q_i)$, then $\varepsilon \in CL(P), (Q, Q_{ej}, \varepsilon) \in EL(P)$ for each j, $1 \le j \le h$; $Q_i \in CL(P), (Q, Q_{ci}, Q_i) \in EL(P)$ for all i, $1 \le i \le k$;

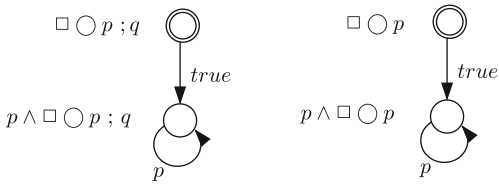
The NFG of formula P is the directed graph $G = (CL(P), EL(P))$.

Note that NFGs will be needed later in Section 7 for handling PITL. 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. Each edge is denoted by a directed arc connecting two nodes. Examples of NFGs are shown in Fig. 7.

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 generated by De nition 3 might wrongly describe models of P with the chop construct. In some circumstances, isomorphic NFGs can be constructed by De nition 3 for two non-equivalent formulas such as $\Box\, p ; q$ and $\Box\, p$ as shown in Fig. 8.

To solve the problem, two cases need to be considered for chop construct $Q_1; Q_2$: (1) Q_1 has only in nite models; in this case $Q_1; Q_2$ has no models. For instance, $\Box\, p; q$ is $false$ but an NFG shown in Fig. 8a can be generated by De nition 3; (2) Q_1 has both nite and in nite models; in this case (see Fig. 7d), we need to eliminate all in nite models of Q_1. Based on the above analysis, we have formalized two algorithms, SIMPLIFY, to generate a subgraph of the NFG, called Labeled Normal Form Graph, for a given formula so that models of the formula can correctly be generated. In algorithm SIMPLIFY, a label F is employed to indicate that a node in a cycle can only be repeated for nitely many times.

Fig. 8 NFGs of $\Box\, p ; q$ and $\Box\, p$



In PPTL, for prj construction $(P_1, \ldots, P_m)\ prj\ Q$ (or chop $P; Q$), each of P_i (or P), $1 \quad i \quad m$ Š 1, is interpreted in nite intervals. However, in the de nition of NFG, P_i (or P) is possibly interpreted over an in nite interval. To mark this type of nodes and to con ne its occurrences to nitely many times in a cycle, label F is required. For instance, in Fig. 7d, without label F on node $(true\ ;\ \Box \quad q)$, an in nite interval consisting of the rst state, interpreting p as $true$, and other states, interpreting p as $true$ or $false$, could be generated in the NFG. Obviously, this interval corresponds to an in nite model for formula p rather than $(p\ ;\ \Box \quad q)$. With label F, node $q \quad \Box \quad q$ can always be reached from node $(true\ ;\ \Box \quad q)$. However, $(P_1, \ldots, P_m)\ prj\ Q$ will never be in a loop since, by algorithm NF, whenever the formula is rewritten to its normal form the prj operator will eventually be replaced by the chop operator in the decomposed formulas. For example, if

$$P_1 \quad P_{1e} \quad \varepsilon \quad \bigvee_{i=1}^{r} (P_{1i} \quad P_{1i})$$

$$P_2 \quad P_{2e} \quad \varepsilon \quad \bigvee_{l=1}^{s} (P_{2l} \quad P_{2l})$$

$$Q \quad Q_e \quad \varepsilon \quad \bigvee_{j=1}^{k} (Q_j \quad Q_j)$$

then,

$$\begin{aligned}
(P_1, P_2)\, prj\ Q \quad & Q_e \quad P_{1e} \quad P_{2e} \quad \varepsilon \\
& \bigvee_{l=1}^{s} (Q_e \quad P_{1e} \quad P_{2l} \quad P_{2l}) \\
& \bigvee_{j=1}^{k} (P_{1e} \quad P_{2e} \quad Q_j \quad Q_j) \\
& \bigvee_{j=1}^{k} \bigvee_{l=1}^{s} (P_{1e} \quad Q_j \quad P_{2l} \quad (P_{2l}; Q_j)) \\
& \bigvee_{i=1}^{r} (Q_e \quad P_{1i} \quad (P_{1i}; P_2)) \\
& \bigvee_{i=1}^{r} \bigvee_{j=1}^{k} (Q_j \quad P_{1i} \quad (P_{1i}; (P_2\, prj\, Q_j)))
\end{aligned}$$

The above explanation shows us that a node (formula) with prj as its main operator does not need label F.

5.2 Labeled Normal Form Graph

We now give the de nition of LNFG, which precisely characterize the models of the corresponding formulas.

De nition 4 The Labeled NFG of formula P is a tuple, $G = (CL(P), EL(P), v_0, V_f)$, where $CL(P)$ denotes the set of nodes and $EL(P)$ is the set of directed edges among $CL(P)$, $v_0 \quad V$ is the initial (or root) node, and V_f denotes the set of nodes with nite label F.

Table 9 Algorithm for constructing LNFG of a PPTL formula

```
Function LNFG(P)
/* precondition: P is a PPTL formula in pre-prepared form */
/* postcondition: LNFG(P) computes LNFG of P, G = (CL(P), EL(P), v_0, V_j) */
begin function
  /*initialization*/
  CL(P) = {P}; EL(P) = φ; v_o = P; V_F = φ; mark[P] = 0; AddE = AddN = 0;
  while there exists R  CL(P) \ {ε, false}, and mark[R] == 0
    do mark[R] = 1;                          /*marking R is decomposed*/
    if R is Q_1 ; Q_2                        /*computing LNFG of Q_1*/
    then add label F in node R; G = (CL(Q_1), EL(Q_1), v_o, V_F) = LNFG(Q_1);
         if ε  CL(Q_1) then continue;
                          /* R needs not decomposed, jump to while*/
    Q = NF(R);
    case
      Q is ⋁_{j=1}^{h} Q_ej   ε: AddE=1;     /* rst part of NF needs added*/
      Q is ⋁_{i=1}^{k} Q_i    Q_i : AddN=1; /*second part of NF needs added*/
      Q is ⋁_{j=1}^{h} Q_ej   ε  ⋁_{i=1}^{k} Q_i    Q_i : AddE=AddN=1;
                              /*both parts of NF needs added*/
    end case
    if AddE == 1 then                        /*add rst part of NF*/
                  CL(P) = CL(P)  {ε};
                  EL(P) = EL(P)  ⋃_{j=1}^{h} {(R, Q_ej, ε)};
                  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  CL(P)
                    then mark[Q_i]=0;
                         /* Q_i needs decomposed*/
         CL(P) = CL(P)  ⋃_{i=1}^{k} {Q_i};
         EL(P) = EL(P)  ⋃_{i=1}^{k} {(R, Q_i, Q_i)};
         AddN=0;
  end while
  return G;
end function
```

Actually, given a formula P, the LNFG of P is generated by means of algorithm LNFG. The algorithm not only produces the nodes and edges of the graph but also identifies some nodes with label F. In Table 9, algorithm LNFG for constructing the LNFG of a PPTL formula is presented. 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 LNFG employs algorithm NF to transform a formula into its normal form. In the algorithm, we consider the chop construct in the form of $Q_1; Q_2$ carefully since Q_1 may only have in nite models and cause R to be $false$. Therefore, if Q_1 has no nite models it needs not further to be decomposed and marked with $mark[R] = 1$ otherwise it needs further to be dealt with and marked with $mark[R] = 0$. Also label F is employed to denote the chop formula cannot be repeated in nitely many times. Further, in the algorithm, two global boolean variables AddE and AddN are employed to indicate whether ε and next the next formulas in the normal form are encountered respectively.

5.3 Finiteness of LNFG

In the LNFG of a PPTL formula P generated by algorithm LNFG, 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 forms. 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 $CL(P)$ is nite, that is, $|CL(P)| = k \quad N_0$. The proof proceeds by induction on the structure of formula P. Throughout this subsection, for simplicity, normal form (4.6) is used.

Lemma 2 *For any PPTL formula* P, $|CL(\quad P)| \quad | \quad CL(P)| + 1$.

Proof By algorithm LNFG, $CL(\quad P) = \{ \quad P\} \quad CL(P)$, so $|CL(\quad P)| \quad | \quad CL(P)| + 1$. The number of nodes in the LNFG of P is at most the number of nodes in the LNFG of P plus one. Figure 9 shows us the relationship between LNFGs of P and P.

Lemma 3 *If* Q *is a PPTL formula and* P_e *is a state formula, then* $|CL(P_e \quad Q)| \quad | \quad CL(Q)| + 1$.

Proof By normal form (4.6), let $Q \quad Q_e \quad \varepsilon \quad \bigvee_{i=1}^{k}(Q_i \quad Q_i)$. Then, $P_e \quad Q \quad P_e \quad Q_e \quad \varepsilon \quad \bigvee_{i=1}^{k}(P_e \quad Q_i \quad Q_i)$. Since $CL(Q) = \{ Q, \varepsilon\} \quad \bigcup_{i=1}^{k} CL(Q_i)$, and $CL(P_e \quad Q) = \{ P_e \quad Q, \varepsilon\} \quad \bigcup_{i=1}^{k} CL(Q_i)$, so we have,

$$
\begin{aligned}
|CL(P_e \quad Q)| &= |\{ P_e \quad Q, \varepsilon\} \quad \bigcup_{i=1}^{k} CL(Q_i)| \\
&\quad |\{ P_e \quad Q\} \ \{ Q, \varepsilon\} \quad \bigcup_{i=1}^{k} CL(Q_i)| \\
&= |\{ P_e \quad Q\} \quad CL(Q)| \\
&\quad | \ CL(Q)| + 1
\end{aligned}
$$

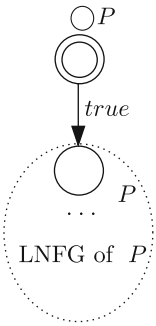
The lemma indicates that the number of nodes in the LNFG of a formula being conjunctive with a state formula is no more than the number of nodes in its own LNFG plus one. In Fig. 10, LNFGs of Q and $P_e \quad Q$ are shown in (a) and (b) respectively. It is obviously that two LNFGs contain the same nodes except for the roots.

Lemma 4 *For PPTL formulas* P *and* Q, $|CL(P \quad Q)| \quad | \quad CL(P)| + | \ CL(Q)| + 1$.

Proof Let $P \quad (P_e \quad \varepsilon) \quad \bigvee_{i=1}^{r}(P_i \quad P_i)$ and $Q \quad (Q_e \quad \varepsilon) \quad \bigvee_{j=1}^{k}(Q_j \quad Q_j)$. Then,

$$
P \quad Q \quad (P_e \quad Q_e) \quad \varepsilon \quad \bigvee_{i=1}^{r}(P_i \quad P_i) \quad \bigvee_{j=1}^{k}(Q_j \quad Q_j)
$$

Fig. 9 LNFG of P



Thus, $CL(P) = \{P, \varepsilon\} \bigcup_{i=1}^{r} CL(P_i)$, $CL(Q) = \{Q, \varepsilon\} \bigcup_{j=1}^{k} CL(Q_j)$, and $CL(P \mathcal{S} Q) = \{P \mathcal{S} Q, \varepsilon\} \bigcup_{i=1}^{r} CL(P_i) \bigcup_{j=1}^{k} CL(Q_j)$. We have,

$$
\begin{aligned}
|CL(P \mathcal{S} Q)| &= \left| \{P \mathcal{S} Q, \varepsilon\} \bigcup_{i=1}^{r} CL(P_i) \bigcup_{j=1}^{k} CL(Q_j) \right| \\
&\quad \left| \{P \mathcal{S} Q, \varepsilon\} \{P\} \bigcup_{i=1}^{r} CL(P_i) \{Q, \varepsilon\} \bigcup_{j=1}^{k} CL(Q_j) \right| \\
&\quad |\{P \mathcal{S} Q\} CL(P) CL(Q)| \\
&\quad |CL(P)| + |CL(Q)| + 1
\end{aligned}
$$

This lemma tells us that the number of nodes in the LNFG of formula Q is no more than the sum of numbers of nodes in LNFGs of formulas P and Q plus one. The LNFG of $P \mathcal{S} Q$ is shown in Fig. 11. Obviously, except for the root, the nodes in the LNFG of $P \mathcal{S} Q$ are descendants of the LNFG of either P or Q.

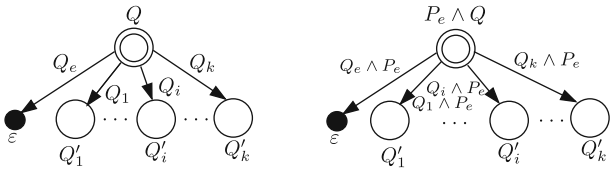
Lemma 5 *If P and Q are PPTL formulas, then* $|CL(P ; Q)| \leq |CL(P)| + |CL(Q)| + 1$.

Proof We consider the following cases. (1) $CL(P) = \emptyset$

This means P has only finite models and thus by algorithm LNFG, the LNFG of $P ; Q$ has only one node. Hence, $|CL(P ; Q)| = |\{P ; Q\}| = 1$. Since $|CL(P)| \geq 1$ and $|CL(Q)| \geq 1$, we have, $|CL(P ; Q)| \leq |CL(P)| + |CL(Q)| + 1$.

(2) $\varepsilon \in CL(P)$.

We first prove a fact that whenever a non-node R is added to $CL(P)$, a node $R ; Q$ is added to $CL(P ; Q)$.



(a) LNFG of Q **(b)** LNFG of $P_e \wedge Q$

Fig. 10 LNFGs of Q and $P_e \wedge Q$

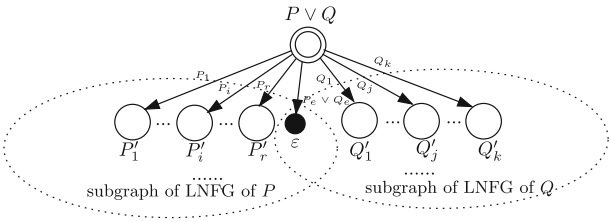


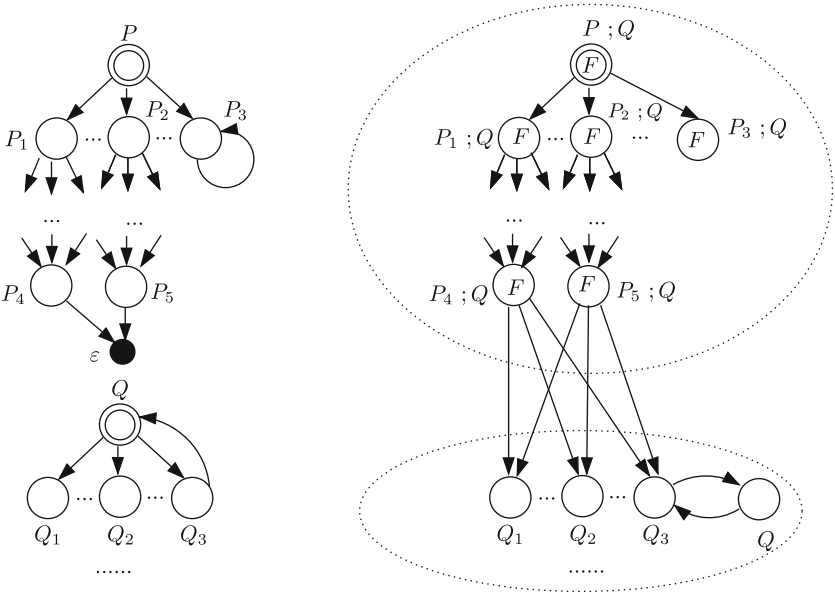
Fig. 11 LNFG of $P \quad Q$

By algorithm LNFG, initially $CL(P) = \{P\}$ and $CL(P ; Q) = \{P ; Q\}$. Suppose in the jth step, node R is added to $CL(P)$, and $R ; Q$ is added to $CL(P ; Q)$. When the algorithm proceeds to the $(j + 1)$th step, if $R \quad R_i \quad R_i$ then $R ; Q \quad (R_i \quad R_i) ; Q$ $R_i \quad (R_i ; Q)$. Thus, while formula R_i is added to $CL(P)$, $R_i ; Q$ is added to $CL(P ; Q)$.

Thus, when a non-node R in $CL(P)$ is rewritten to its normal form $R \quad R_e \quad \varepsilon$ and ε is added to $CL(P)$, according to the fact proved above, there must exist $R_e ; Q$ in $CL(P ; Q)$ and $R ; Q \quad ((R_e \quad \varepsilon) ; Q) \quad R_e \quad Q$ such that the LNFG of $R_e \quad Q$ is constructed, and $CL(R_e \quad Q)$ is eventually added to $CL(P ; Q)$ by algorithm LNFG.

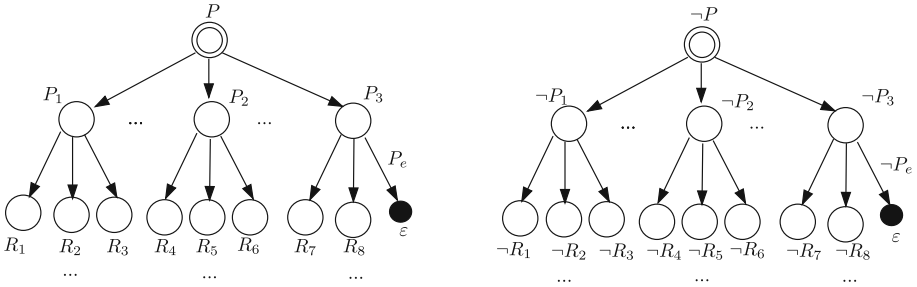
So $|CL(P ; Q)| \quad | \quad CL(P) \backslash \{\varepsilon\}| + | \quad CL(R_e \quad Q)|$, leading to $|CL(P ; Q)| \quad | \quad CL(P)| + |CL(R_e \quad Q)| \quad | \quad CL(P)| + | \quad CL(Q)| + 1$ (Lem 3).

Similar to Lemma 4, this lemma shows us that the number of nodes in the LNFG of $P ; Q$ is no more than the sum of numbers of nodes in the LNFGs of P and Q plus one. To show the relationship among the LNFGs of P, Q and $P ; Q$ clearly, the LNFGs of P and Q are presented in Fig. 12a, and the LNFG of $P ; Q$ is depicted in Fig. 12b. Obviously, the LNFG of $P ; Q$ is constructed from the LNFGs of P and Q.



(a) LNFGs of P and Q (b) LNFG of $P ; Q$

Fig. 12 LNFGs of P, Q and $P ; Q$



(a) LNFG of P (b) LNFG of $\neg P$

Fig. 13 LNFGs of P and $\neg P$

Lemma 6 *For a PPTL formula* P, $|CL(\neg P)| \mid CL(P)| + 1$.

Proof We first prove a fact that whenever a non-ε node Q is added to $CL(P)$, a node $\neg Q$ is added to $CL(\neg P)$.

The proof proceeds by induction on the number of steps for generating normal form of $\neg P$. To generate the normal form of P, the complete normal form of P is employed. Thus, initially P is added to $CL(P)$, and $\neg P$ is added to $CL(\neg P)$. Suppose in the i^{th} step, a node Q (not ε) is added to $CL(P)$, and $\neg Q$ is added to $CL(\neg P)$. When the algorithm proceeds to the $(i+1)^{th}$ step, we consider Q in its complete normal form. That is, Q $(Q_e \ \varepsilon) \ \bigvee_{i=1}^{k}(Q_i \ Q_i)$, thus $\neg Q \ (\neg Q_e \ \varepsilon) \ \bigvee_{i=1}^{k}(Q_i \ \neg Q_i)$ [Sect. 4 (4.8)]. Hence, while node Q_i is added to $CL(P)$, node $\neg Q_i$ is added to $CL(\neg P)$.

So the numbers of non-ε nodes in $CL(\neg P)$ and $CL(P)$ are identical. Therefore, $|CL(\neg P) \ \{\varepsilon\}| = |CL(P) \ \{\varepsilon\}|$, leading to $|CL(\neg P)| \mid CL(P) \ \{\varepsilon\}| \mid CL(P)| + 1$.

This lemma indicates the relationship between node numbers of the LNFGs of P and $\neg P$. As we can see, each of non-ε nodes in the LNFG of P is replaced by the negation of the formula in the LNFG of $\neg P$ as shown in Fig. 13. Further, both LNFGs can contain an ε node.

Lemma 7 *If* $|CL(P_i)| = k_i \ N_0 \ (1 \ i \ m)$ *and* $|CL(Q)| = k_0 \ N_0$, *then* $|CL((P_1, \ldots, P_m) \ prj \ Q)| = k \ N_0$

Proof The proof proceeds by induction on m.

Base case $m = 1$, let

$$P_1 \ (P_{1e} \ \varepsilon) \ \bigvee_{i=1}^{r}(P_{1i} \ P_{1i})$$

$$Q \ (Q_e \ \varepsilon) \ \bigvee_{j=1}^{h}(Q_j \ Q_j)$$

Then, $CL(P_1) = \{P_1, \varepsilon\} \cup \bigcup_{i=1}^{r} CL(P_{1i})$, $CL(Q) = \{Q, \varepsilon\} \cup \bigcup_{j=1}^{h} CL(Q_j)$. Since

$$P_1 prj Q \quad P_{1e} \quad Q_e \quad \varepsilon$$
$$\bigvee_{j=1}^{h} (P_{1e} \quad Q_j \quad Q_j)$$
$$\bigvee_{i=1}^{r} (Q_e \quad P_{1i} \quad P_{1i})$$
$$\bigvee_{j=1}^{h} \bigvee_{i=1}^{r} (Q_j \quad P_i \quad (P_{1i}; Q_j))$$

So,

$$CL(P_1 \ prj \ Q) = \{P_1 \ prj \ Q, \varepsilon\} \cup \bigcup_{j=1}^{h} CL(Q_j) \cup \bigcup_{i=1}^{r} CL(P_{1i})$$
$$\bigcup_{i=1}^{r} \bigcup_{j=1}^{h} CL(P_{1i} ; Q_j)$$

We have,

$$|CL(P_1 \ prj \ Q)| \quad |\{P_1 \ prj \ Q, \varepsilon\}| + \left|\bigcup_{i=1}^{r} CL(P_{1i})\right| + \left|\bigcup_{j=1}^{h} CL(Q_j)\right|$$
$$+ \left|\bigcup_{i=1}^{r} \bigcup_{j=1}^{h} CL(P_{1i} ; Q_j)\right|$$
$$2 + k_1 + k_0 + \sum_{i=1}^{r} \sum_{j=1}^{h} (|CL(P_{1i})| + |CL(Q_j)| + 1) \text{Lem 4}$$
$$2 + k_1 + k_0 + rh(k_1 + k_0 + 1) \quad N_0$$

Inductive step: Suppose $|CL((P_2, \ldots, P_m) \ prj \ Q)| = k \quad N_0$.

$$(P_1, \ldots, P_m) \ prj \ Q$$
$$\left(\left(P_{1e} \quad \varepsilon \quad \bigvee_{i=1}^{r} (P_{1i} \quad P_{1i})\right), P_2, \ldots, P_m\right) \ prj \ Q$$
$$((P_{1e} \quad \varepsilon, P_2, \ldots, P_m) \ prj \ Q) \quad \bigvee_{i=1}^{r} ((P_{1i} \quad P_{1i}, P_2, \ldots, P_m) \ prj \ Q) \ L_{18}$$
$$P_{1e} \quad ((P_2, \ldots, P_m) \ prj \ Q) \quad \bigvee_{i=1}^{r} P_{1i} \quad ((\quad P_{1i}, P_2, \ldots, P_m) \ prj \ Q) \ L_{17}, L_{22}$$
$$P_{1e} \quad ((P_2, \ldots, P_m) \ prj \ Q)$$
$$\bigvee_{i=1}^{r} P_{1i} \quad \left((\quad P_{1i}, P_2, \ldots, P_m) \ prj \left(Q_e \quad \varepsilon \quad \bigvee_{j=1}^{h} (Q_j \quad Q_j)\right)\right)$$

$$P_{1e} \quad ((P_2, \ldots, P_m)\ prj\ Q) \quad \bigvee_{i=1}^{r} P_{1i} \quad Q_e \qquad (P_{1i}; P_2; \ldots; P_m)$$

$$\bigvee_{i=1}^{r}\bigvee_{j=1}^{h} P_{1i} \quad Q_j \qquad \left(P_{1i}\ ; \left((P_2, \ldots, P_m)\ prj\ Q_j\right)\right)\ L_{18}, L_{23}, L_{15}, L_{21}$$

So,

$$\begin{aligned}
&|CL((P_1, \ldots, P_m)\ prj\ Q)| \\
&\quad |\ CL(P_{1e} \quad ((P_2, \ldots, P_m)\ prj\ Q))| \\
&\quad + \sum_{i=1}^{r} |CL(P_{1i} \quad Q_e \qquad (P_{1i}; P_2; \ldots; P_m))| \\
&\quad + \sum_{i=1}^{r}\sum_{j=1}^{h} |CL(P_{1i} \quad Q_j \qquad (P_{1i}\ ; ((P_2, \ldots, P_m)\ prj\ Q_j)))| + r + r \cdot h \ \text{Lem 4} \\
&\quad |\ CL((P_2, \ldots,\ P_m)\ prj\ Q)| + 1 + \sum_{i=1}^{r} (|CL(P_{1i}; P_2; \ldots; P_m)| + 1 + 1) \\
&\quad + \sum_{i=1}^{r}\sum_{j=1}^{h} (|CL(P_{1i}\ ; ((P_2, \ldots, P_m)\ prj\ Q_j))| + 1 + 1) + r + r \cdot h \ \text{Lem 2,3} \\
&\quad k\ +\ 1 + r(k_1 + k_2 + \cdots + k_m + m \text{ Š } 1 + 2) + r \cdot h(|CL(P_{1i})| \\
&\quad +|\ CL((P_2, \ldots, P_m)\ prj\ Q_j)| + 1 + 2) + r + r \cdot h \ \text{Lem 5} \\
&\quad k\ +\ 1 + r(k_1 + k_2 + \cdots + k_m + m + 1) + r \cdot h(k_1 + k\ +\ 3) + r + r \cdot h \\
&= k \quad N_0
\end{aligned}$$

This lemma shows us only that if the numbers of nodes in the LNFGs of $P_1, \ldots, P_m$ and Q are finite then the number of nodes in the LNFG of $(P_1, \ldots, P_m)\ prj\ Q$ is also finite.

Theorem 3 *For any PPTL formula P, $|CL(P)|$ is finite.*

Proof The proof proceeds by induction on the structure of PPTL formulas.

Base case: P is an atomic proposition p. Rewrite p to its normal form, $p \quad p\ \varepsilon\ p \quad true$. Further, $true$ can be rewritten to its normal form, $true \quad true\ \varepsilon\ true \quad true$. So in the LNFG of p (see Fig. 14), $CL(p) = \{p, true, \varepsilon\}$. Thus, $|CL(p)| = |\{p, true, \varepsilon\}| = 3$.

Induction step: Suppose $|CL(P_i)| = k_i \quad N_0$, $1 \quad i \quad m$, and $|CL(Q)| = k_0 \quad N_0$. Then,

(1) $P \qquad P_1$: by Lemma 2, $|CL(\quad P_1)| \quad k_1 + 1 = k \quad N_0$
(2) $P \quad P_1 \quad P_2$: by Lemma 4, $|CL(P_1 \quad P_2)| \quad k_1 + k_2 + 1 = k \quad N_0$

Fig. 14 LNFG of atomic proposition p

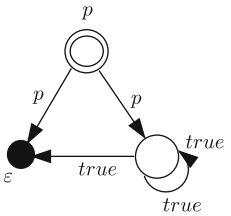


Table 10 Algorithm for simplifying an LNFG

Function SIMPLIFY(G)
/* precondition: $G = (CL(P), EL(P), v_o, V_F)$ is an LNFG of PPTL formula P */
/* postcondition: SIMPLIFY(G) computes an LNFG of P, $G = (CL(P), EL(P))$, which contains no redundant nodes */

```
begin function
    G = G;
    while R ≠ CL(P) and R is not ε and has no edges departing from
          do (CL(P) = CL(P) \ R;
              EL(P) = EL(P) \ ⋃_i (R_i, R_e, R);)
        /* ⋃_i (R_i, R_e, R) denotes the set of edges connecting to node R */
    end while
    return G;
end function
```

(3) $P = \neg P_1$: by Lemma 6, $|CL(\neg P_1)| \leq k_1 + 1 = k \leq N_0$
(4) $P = (P_1, \ldots, P_m)\ prj\ Q$: by Lemma 7, $|CL((P_1, \ldots, P_m)\ prj\ Q)| = k \leq N_0$

The theorem convinces us that for any PPTL formula P, the number of nodes in the LNFG of P is nite. This is critical since it guarantees that algorithm LNFG terminates. Furthermore, this enables us to develop a decision procedure for checking the satis ability of PPTL formulas based on algorithm LNFG. In addition, the LNFG of a formula P contains all models of P, including both nite and in nite ones. Hence, our decision procedure not only gives an algorithm for checking the satis ability of a formula P but also constructs all models of formula P.

In an LNFG constructed by algorithm LNFG, some nodes might have no successors (e.g. $P = \neg P$). These nodes are redundant and can be removed. Algorithm SIMPLIFY is useful for eliminating redundant nodes of an LNFG as shown in Table 10.

6 Decision procedure for PPTL formulas

6.1 Paths and models

In the LNFG constructed by algorithm LNFG and SIMPLIFY for formula Q, a nite path, $\Pi = Q, Q_e, Q_1, Q_{1e}, \ldots, \varepsilon$, is an alternate sequence of nodes and edges from the root to ε node, while an in nite path, $\Pi = Q, Q_e, Q_1, Q_{1e}, \ldots$, is an in nite alternate sequence of nodes and edges emanating from the root. Note that, in an in nite path, there must exist some nodes which appear in the path for in nitely many times, and the nodes with F label can just occur for nitely many times. In fact, a path (nite or in nite) in the LNFG of a formula Q corresponds to a model of Q. The fact is concluded in Theorem 4 and 5.

Theorem 4 *A formula Q can be satis ed by nite models if and only if there exist nite paths in the LNFG of Q.*

Proof To prove this theorem, we can equivalently prove that for any PPTL formula Q, Q can be satis ed by nite models, if and only if node can be found in the LNFG of Q,

$G = (CL(Q), EL(Q), v_o, V_F)$. For convenience, we assume the normal form of Q is shown as follows: $Q \bigvee_{j=1}^{l}(Q_{ej} \varepsilon) \bigvee_{i=1}^{n}(Q_i Q_i)$.

( ) If Q can be satis ed by nite models, then node can be found in the LNFG of Q.

Suppose there is a nite model σ, $\sigma \mid Q$. We try to prove $\varepsilon \in CL(Q)$. The proof proceeds by induction on the length of σ.

Base: If $|\sigma| = 0$, then there must exist a product Q_e, such that $\sigma = < s_0 > \mid Q_e \varepsilon$, where $Q_e \bigwedge_i p_i$, and for each p_i, $s_0[p_i] = true$. Since $Q \bigvee_{j=1}^{l}(Q_{ej} \varepsilon) \bigvee_{i=1}^{n}(Q_i Q_i)$. So Q_e must be some Q_{ej} in $\bigvee_{j=1}^{l}(Q_{ej} \varepsilon)$. By algorithm LNFG, there must exist node Q, ε and edge $< Q, Q_{ej}, \varepsilon >$ in G. Thus, the conclusion holds.

Induction: Suppose for all σ, $|\sigma| = k$ Š 1 $(k \geq 1)$, the conclusion holds. If $|\sigma| = k$, then i, $1 \leq i \leq n$, such that $\sigma \mid Q_i Q_i$, leading to $\sigma \mid Q_i$ and $\sigma \mid Q_i$. That is, $(\sigma, 0, |\sigma|) \mid Q_i$ and $(\sigma, 1, |\sigma|) \mid Q_i$. By algorithm LNFG, there must exist node Q, Q_i and edge $< Q, Q_i, Q_i >$ in G. Let $\sigma = \sigma_{(1..|\sigma|)}$, thus, $(\sigma , 0, |\sigma |) \mid Q_i$. Let $G = (CL(Q_i), EL(Q_i))$ be the LNFG of Q_i. By inductive hypothesis, $\varepsilon \in CL(Q_i)$. It is clear that $CL(Q_i) \subseteq CL(Q)$, so $\varepsilon \in CL(Q)$.

( ) If $\varepsilon \in CL(Q)$, then Q can be satis ed by nite models.

Since $\varepsilon \in CL(Q)$, there must exist a nite path $\Pi = Q, Q_{11}, Q_{11}, Q_{21}, Q_{21}, \ldots, Q_{k1}, \varepsilon$ in G. We need to prove that, for $|\Pi| \geq 1$, an interval σ can be constructed so that $\sigma \mid Q$. The proof proceeds by induction on the length of path Π.

Base: If $|\Pi| = 1$, that is $k = 1$, then there must exist a path $Q, Q_{ej}, \varepsilon >$ in G. This means that there exists $Q_{ej} \varepsilon$ in the normal form of formula Q. We can construct an interval $\sigma = < s_0 >$ such that for any p contained in Q_{ej}, $s_0[p] = true$. Thus obviously, $\sigma \mid Q$.

Induction: Suppose $|\Pi| = k$ Š 1 $(k \geq 1)$ the conclusion is true. If $|\Pi| = k$, then $\Pi = Q, Q_{11}, Q_{11}, Q_{21}, Q_{21}, \ldots, Q_{k1}, \varepsilon$. Let $\Pi = Q_{11}, Q_{21}, Q_{21}, \ldots, Q_{k1}, \varepsilon$, then $|\Pi | = k$ Š 1. By hypothesis, we can construct a model σ, $\sigma \mid Q_{11}$. Then we can construct a new interval as follows $\sigma = < s_0 > \cdot \sigma$, where s_0 is a state such that for all p contained in Q_{11}, $s_0[p] = true$, so $< s_0 > \mid Q_{11}$. Thus, $\sigma \mid Q_{11}$ and $\sigma \mid Q_{11}$, leading to $\sigma \mid Q_{11} Q_{11}$. Since $Q \bigvee_{j=1}^{l}(Q_{ej} \varepsilon) \bigvee_{i=1}^{n}(Q_i Q_i)$, therefore $\sigma \mid Q$.

A similar conclusion regarding in nite paths and models is given in Theorem 5. To prove it, we need the following two lemmas. Lemma 8 is concerned with constructing a model from an in nite path in the LNFG of a formula while Lemma 9 does the reverse. The two lemmas are proved by means of xed point theory and Scott's xed-point induction.

Fixed point theory: Every monotonic function F over a complete lattice A, has a unique least xed point $\bigcup_i F^i( )$ and a unique greatest xed point $\bigcap_i F^i( )$ [20].

ScottÕs Þxed-point induction: D is a complete lattice with bottom, $F : D \to D$ is a continuous function, P is an inclusion subset of D. If $\forall x \in D. x \in P \Rightarrow F(x) \in P$, then $fix(F) \subseteq P$ [20].

Lemma 8 *Given a PPTL formula Q, if there exists an in nite path, where F occurs only nitely often, in the LNFG of Q, a corresponding in nite model of Q can be constructed.*

Proof Let $\Pi = Q, Q_{0e}, Q_0, Q_{1e}, \ldots, Q_{ie}, Q_i, \ldots$ be an in nite path, where F occurs only nitely often, in the LNFG of Q. Then we de ne a sequence $\sigma = s_0, s_1, \ldots, s_i, \ldots$ w.r.t Π. For each $i \in N_0$, if proposition q in Q_{ie} then $s_i[q] = true$, otherwise if $\neg q$ in Q_{ie} then $s_i[q] = false$. We need to prove σ is a model of formula Q. To do so, we need rst to prove that a pre x σ^i of σ is a pre x of a model. The proof proceeds by induction on the length of the pre x.

Base: $\sigma^0 = s_0$. By construction of LNFG, $s_0, s_0 \mid Q_{0e}$.

Induction: Suppose for all $j < i$, pre x σ^j of σ is a pre x of a model. We prove σ^i is also a pre x of the model. By hypothesis, $s_{i-1} = s_0, \ldots, s_{i-1}$ is a pre x of the model. According to algorithm LNFG, there must be $Q_{i-1} \quad Q_{ie} \quad Q_i$ such that a new node Q_i and a new edge (Q_{i-1}, Q_{ie}, Q_i) are added to the LNFG of Q. By the construction of s_i, $s_i \mid Q_{ie}$. Since σ^{i-1} is a pre x of the model σ^i is a pre x of the model. Now we need to prove that σ is a model of Q.

Let $\sigma_s^{i-1} = \phi$ and $\sigma_s^i = \{(0, s_0), (1, s_1), \ldots, (i, s_i)\}$ be a coded set corresponding to σ^i. Then we de ned a set $A = \{\sigma_s^{i-1}, \ldots, \sigma_s^i, \ldots\}$. It is readily to prove that $(A, \quad)$ is a complete partial order set with bottom $\sigma_s^{i-1} = \phi$, so $(A, \quad)$ is a complete lattice. We de ne a function R over A as follows:

$$R(\sigma_s^k) = \sigma_s^{k+1}, \quad k \geq -1$$

So, for $\sigma_s^i, \sigma_s^j \quad A, i \quad j$, we have $\sigma_s^i \quad \sigma_s^j$, and

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

Since $i \quad j$, so $i + 1 \quad j + 1$. Then we have $\sigma_s^{i+1} \quad \sigma_s^{j+1}$. So, R is monotonic. Thus, by xed point theorem,

$$x(R) = \bigcup_i R^i(\sigma_s^{-1}) = R \quad (\sigma_s^{-1}) = \sigma_s$$

Also, we have

$$R\left(\bigcup_{i \quad \omega} \sigma_s^i\right) = \bigcup_{i \quad \omega} R(\sigma_s^i)$$

So R is also continuous. Since w.r.t each σ_s^i, σ^i is a pre x of the model for formula Q, by Scott's xed-point induction, we know that, w.r.t σ_s, σ is a model of formula Q.

Lemma 9 *Given a PPTL formula Q, if there exists an in nite model of Q, a corresponding in nite path, where F occurs only nitely often, in the LNFG of Q can be found.*

Proof If there exists an in nite model $\sigma = s_0, s_1, \ldots, (\sigma, 0, |\sigma|) \mid Q$, then there must exist a normal form of Q, $Q \quad (Q_{0e} \quad \varepsilon) \quad \bigvee_{j=0}^{k}(Q_{0j} \quad Q_{0j})$ such that there exists an edge from root Q to node Q_{00} and labeled by Q_{00} in the LNFG of Q and $(\sigma, 0, |\sigma|) \mid Q_{00} \quad Q_{00}$. Here $Q_{00} \quad \bigwedge_i q_i$, and q_i is an atomic proposition p_i or $\neg p_i$. Thus, $s_0 \mid Q_{00}$, and $(\sigma, 1, |\sigma|) \mid Q_{00}$. That is, $s_0[q_i] = true$ if q_i is p_i, and $s_0[q_i] = false$ if q_i is $\neg p_i$. So we have found a pre x $\Pi^0 = Q, Q_{00}, Q_{00}$ of a path Π w.r.t the pre x s_0 of model σ.

Suppose we have found a pre x $\Pi^i = Q, Q_{00}, Q_{00}, Q_{10}, Q_{10}, \ldots Q_{i0}, Q_{i0}$ of Π w.r.t the pre x $s_0, \ldots, s_i$ of model σ such that, for $0 \quad k \quad i$, $(\sigma, k, |\sigma|) \mid Q_{k0} \quad Q_{k0}$; here $Q_{k0} \quad \bigwedge_j p_{kj}$, and p_{kj} is an atomic proposition p_{kj} or $\neg p_{kj}$, and $< s_k > \mid Q_{k0}$, and $(\sigma, k+1, |\sigma|) \mid Q_{k0}$; that is, $s_k[p_{kj}] = true$ if p_{kj} is p_{kj}, and $s_k[p_{kj}] = false$ if p_{kj} is $\neg p_{kj}$.

At this point, we can rewrite Q_{i0} to its normal form $Q_{i0} \quad (Q_{(i+1)e} \quad \varepsilon) \quad \bigvee_{j=0}^{k}(Q_{(i+1)j} \quad Q_{(i+1)j})$, such that there exists an edge from Q_{i0} to $Q_{(i+1)0}$ and labeled by $Q_{(i+1)0}$ in the LNFG of Q and $(\sigma, i+1, |\sigma|) \mid Q_{(i+1)0} \quad Q_{(i+1)0}$. Here $Q_{(i+1)0} \quad \bigwedge_l r_l$, and r_l is an atomic proposition p_l or $\neg p_l$. Thus, $s_{i+1} \mid Q_{(i+1)0}$ and $(\sigma, (i+2), |\sigma|) \mid Q_{(i+1)0}$.

That is, $s_{i+1}[r_l] = true$ if r_l is r_l, and $s_{i+1}[r_l] = false$ if r_l is $\neg r_l$. So we have found a prefix $\Pi^{i+1} = Q, Q_{00}, Q_{00}, Q_{10}, Q_{10}, \ldots Q_{i0}, Q_{i0}, Q_{(i+1)0}, Q_{(i+1)0}$ of Π w.r.t the prefix $s_0, \ldots, s_i, s_{i+1}$ of model σ.

Now we need to prove $\Pi = Q, Q_{00}, Q_{00}, Q_{10}, Q_{10}, \ldots, Q_{i0}, Q_{i0}, Q_{(i+1)0}, Q_{(i+1)}, \ldots$ is an infinite path, where F occurs only finitely often, of the LNFG of Q w.r.t σ. To do so, we define coded sets as follows: $\Pi_s^{\check{S}1} = \phi$, $\Pi_s^0 = \{(0, Q, Q_{00}, Q_{00})\}$, $\Pi_s^i = \{(0, Q, Q_{00}, Q_{00}), \ldots, (i, Q_{(i\check{S}1)0}, Q_{i0}, Q_{i0})\}$ ($i \in N_0$). Then we define a set $A = \{\Pi_s^{\check{S}1}, \Pi_s^0 \ldots, \Pi_s^i, \ldots\}$. It is readily to prove that $(A, \sqsubseteq)$ is a complete partial order set with bottom $\Pi_s^{\check{S}1} = \phi$, so $(A, \sqsubseteq)$ is a complete lattice. We define a function R over A as follows:

$$R(\Pi_s^k) = \Pi_s^{k+1}$$

So, for $\Pi_s^i, \Pi_s^j \in A, i \leq j$, we have $\Pi_s^i \sqsubseteq \Pi_s^j$, and

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

Since $i \leq j$, so $i + 1 \leq j + 1$. Then we have $\Pi_s^{i+1} \sqsubseteq \Pi_s^{j+1}$. So, R is monotonic. Therefore, by Tarsky's fixed point theorem,

$$x(R) = \bigcup_i R^i(\Pi_s^{\check{S}1}) = R \quad (\Pi_s^{\check{S}1}) = \Pi_s$$

Further, we also have

$$R\left(\bigcup_{i\ \omega} \Pi_s^i\right) = \bigcup_{i\ \omega} R(\Pi_s^i)$$

So, R is also continuous.

Further, by Scott's fixed-point induction, we know that, w.r.t σ_s, Π is an infinite path in the LNFG of formula Q.

Now, we need to show that, in Π, F occurs only finitely often. By algorithm LNFG, any infinite path $\Pi = Q, Q_{00}, Q_{00}, Q_{10}, Q_{10}, \ldots, Q_{i0}, Q_{i0}, Q_{(i+1)0}, Q_{(i+1)}, \ldots$ can be expressed as follows,

$$\Pi = Q, Q_{00}, Q_{00}, Q_{10}, Q_{10}, \ldots, Q_{i0}, \underbrace{Q_{i0}, Q_{(i+1)0}, Q_{(i+1)}, \ldots, Q_{j0}, Q_{(j+1)0}}_{\text{in finitely many times}}$$

Suppose Q_{k0}, $i \leq k \leq j + 1$, is labeled with F, then by algorithm LNFG, the main operator of Q_{k0} is chop (let $Q_{k0} \equiv P_1; P_2$). Thus, in the LNFG of Q_{k0}, there exists a cyclic branch as shown in Fig. 5.

Similar to the analysis in Fig. 3, for the prefix of Π, $\Pi = Q_{k0}, Q_{(k+1)0}, Q_{(k+1)}, \ldots$, the corresponding model $\sigma \mid P_1$ but not $Q_{i0} \equiv P_1; P_2$. Thus, corresponding to Π, $\sigma \mid Q$. This contradicts the condition of the lemma. Therefore, Q_{k0} cannot be labeled with F. So if there exists F in Π, it can occur only finitely often.

Theorem 5 *A formula Q can be satisfied by infinite models if and only if there exist infinite path, where F occurs only finitely often, in the LNFG of Q.*

Proof This theorem is a direct consequence of Lemma 8 and 9.

Theorems 4 and 5 confirm that a PPTL formula is satisfiable if and only if there exist finite or infinite paths in its LNFG.

Fig. 15 A cyclic branch in the LNFG of Q_{k0}

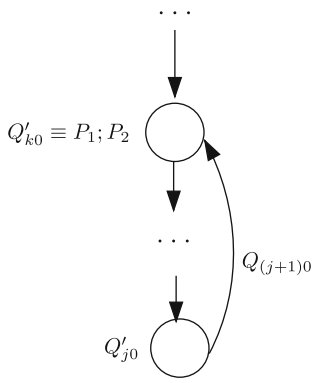


Table 11 Algorithm for checking the satis ability of a PPTL formula

Function CHECK(P)
/* precondition: P is a PPTL formula */
/* postcondition: CHECK(P) checks whether or not formula P is satis able. */

begin function
  G = LNFG(P);
  G = SIMPIFY(G);
  if G is empty, return unsatis able;
  else return satis able;
end function

6.2 Decision procedure

In the simpli ed LNFG of a formula P, a nite or in nite path can readily be constructed. Obviously, P is satis able if and only if there exist nite or in nite paths in the LNFG of P. Consequently, a decision procedure for checking the satis ability 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. Let P be a PPTL formula. The sketch of the procedure is shown in Table 11.

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

6.3 Examples

In this subsection, two examples are given to show how our decision procedure works.

Example 3 Check the satis ability of formula,

$$P \quad ^2(\Box \quad p ; q) \quad (p ; \Box \quad q)$$

Step 1 Build the LNFG of formula P.

As depicted in Fig. 16 (1), initially, the root node is created and named with formula P. To get the successor nodes, formula $^2(\Box \quad p ; q) \quad (p ; \Box \quad q)$ is rewritten to

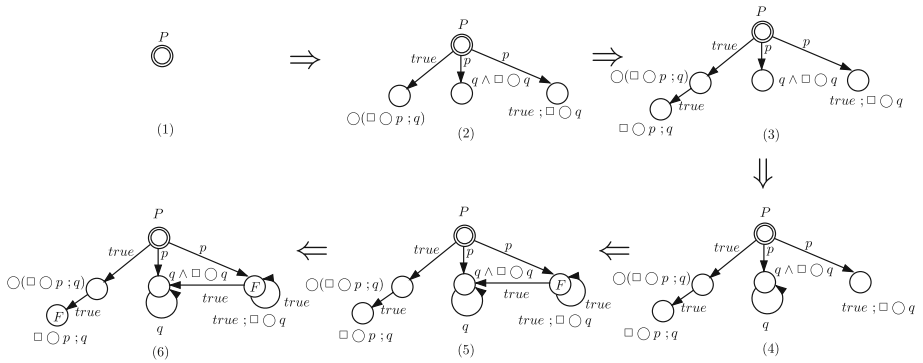


Fig. 16 Constructing the LNFG of P

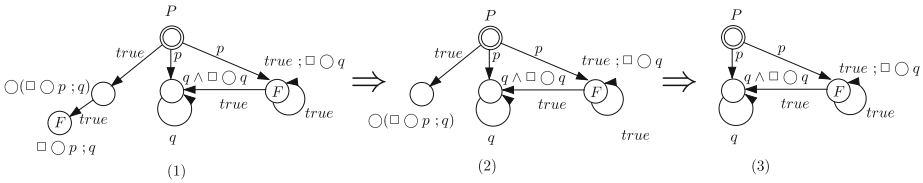


Fig. 17 Simplifying the LNFG of P

its normal form $(\bigcirc(\Box \bigcirc p; q)) \wedge p \wedge (q \wedge \Box \bigcirc q) \wedge p \wedge (true; \Box \bigcirc q)$. Thus nodes $(\Box \bigcirc p; q), q \wedge \Box \bigcirc q$ and $true; \Box \bigcirc q$ are generated from the root and connected by directed arcs labeled with $true$, p and p respectively [Fig. 16 (2)]. Subsequently, node $\Box \bigcirc p; q$ is created from node $(\Box \bigcirc p; q)$ as shown in Fig. 16 (3). Since the normal form of $q \wedge \Box \bigcirc q$ is $q \wedge (q \wedge \Box \bigcirc q)$, an edge labeled with q is emanated from node $q \wedge \Box \bigcirc q$ to itself (see Fig. 16 (4)). Further, as depicted in Fig. 16 (5), since the main operator of formula $true; \Box \bigcirc q$ is chop (;), label F is added in node $true; \Box \bigcirc q$; since ε node is contained in the LNFG of formula $true$, by rewriting $true; \Box \bigcirc q$ to its normal form, $true; \Box \bigcirc q \wedge (q \wedge \Box \bigcirc q) \wedge (true; \Box \bigcirc q)$, two edges both labeled by $true$ are created from node $true; \Box \bigcirc q$ to node $q \wedge \Box \bigcirc q$ and node $true; \Box \bigcirc q$ itself respectively. Finally, chop formula $\Box \bigcirc p; q$ is encountered. We construct the LNFG of $\Box \bigcirc p$ firstly (see Fig. 8b), since there exists no node in the LNFG of $\Box \bigcirc p$, label F is only needed for node $\Box \bigcirc p; q$.

Step 2 Obtain simplified LNFG of formula P.

As shown in Fig. 17 (1), node $\Box \bigcirc p; q$ has no descendant nodes. We remove it and the edges connecting to it. Thus Fig. 17 (2) is obtained. Similarly, node $(\Box \bigcirc p; q)$ and the edges connecting to it are removed. Thus the simplified LNFG is obtained as shown in Fig. 17 (3).

Step 3 The simplified LNFG of P is not empty, so formula P is satisfiable.

Example 4 Check the satisfiability of formula,

$$Q \equiv (skip, p \wedge \Box \bigcirc p, \varepsilon) prj^2 q$$

Step 1 Build the LNFG of formula Q.

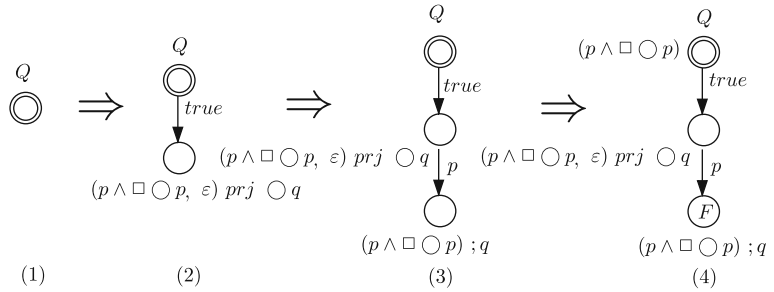


Fig. 18 Constructing the LNFG of Q

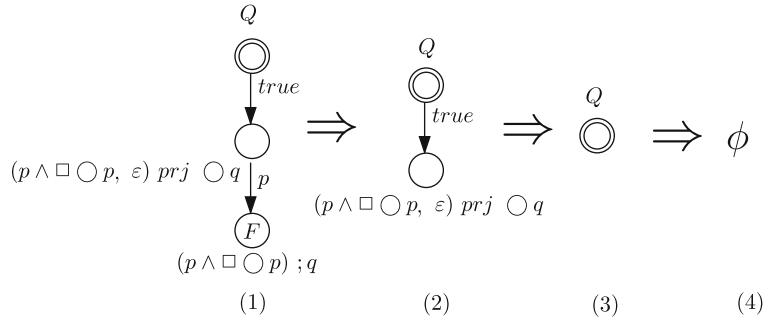


Fig. 19 Simplifying the LNFG of Q

As depicted in Fig. 18 (1), initially, the root node is created and named with formula Q. To get its successor nodes, formula Q is rewritten to its normal form $((p \ \Box \ p, \varepsilon) prj \ q)$. Thus node $(p \ \Box \ p, \varepsilon) prj \ q$ is generated from the root and connected by a directed arc labeled with $true$ (see Fig. 18 (2)). Subsequently, as depicted in Fig. 18 (3), by rewriting $(p \ \Box \ p, \varepsilon) prj \ q$ to its normal form, $p \ (p \ \Box \ p; q)$, node $p \ \Box \ p; q$ is created and connected by an edge with label p from node $(p \ \Box \ p, \varepsilon) prj \ q$. Finally, for chop formula $p \ \Box \ p; q$, we construct the LNFG of $p \ \Box \ p$ rstly [see Fig. 8b], since there exists no node in the LNFG of $p \ \Box \ p$, label F is only needed for node $p \ \Box \ p; q$.

Step 2 Obtain simpli ed LNFG of formula Q.

As depicted in Fig. 19 (1), node $p \ \Box \ p; q$ has no descendant nodes. We remove it and the edges connecting to it, thus Fig. 19 (2) is obtained. Subsequently, nodes $(\Box \ p; q)$ and the edges connecting to it are removed (see Fig. 19 (3)). Now only the root node without edges emanating from it left. It is eventually removed. Thus the simpli ed LNFG is empty as shown in Fig. 19 (4).

Step 3 The simpli ed LNFG of Q is empty, so formula Q is unsatis able.

7 A decision procedure for PITL with in nite models

In this section, we will present a similar decision procedure for PITL with projection [6]. PITL formulas are inductively de ned by the following grammar:

$$P ::= p \mid \quad P \mid \neg P \mid P_1 \quad P_2 \mid P_1 ; P_2 \mid P \quad \mid P \ proj \ Q$$

The states, intervals and interpretations are the same as de ned in PPTL. The satisfaction relation (|) is inductively de ned as follows,

$\mathcal{I} \mid p$ iff $s_k[p]$ = true, for any given proposition p
$\mathcal{I} \mid \neg P$ iff $\mathcal{I} \mid P$
$\mathcal{I} \mid P_1 \quad P_2$ iff $\mathcal{I} \mid P_1$ or $\mathcal{I} \mid P_2$
$\mathcal{I} \mid \quad P$ iff $k < j$ and $(\sigma, k+1, j) \mid P$
$\mathcal{I} \mid P_1; P_2$ iff there exists r such that $k \quad r \quad j$ and $(\sigma, k, r) \mid P_1$ and $(\sigma, r, j) \mid P_2$ or $(\sigma, k, j) \mid P_1, j = \omega$
$\mathcal{I} \mid P^*$ iff there are nitely many integers $s = r_0 \quad r_1 \quad \cdots \quad r_{n-1} \quad r_n$ j and $(\sigma, r_0, r_1) \mid P$, and for all $1 < l \quad n, (\sigma, r_{l-1}, r_l) \mid P$ and also $(\sigma, r_n, j) \mid P$, or there are in nitely many integers, $k = r_0 \quad r_1 \quad r_2 \ldots$ such that $\lim_{i} r_i = \omega$ and $(\sigma, i, r_0, r_1) \mid P$, and for all $1 \quad l \quad n$, $(\sigma, r_{l-1}, r_l) \mid P$
$\mathcal{I} \mid P$ *proj* Q if there exist nitely many integers $s = r_0 < r_1 < \cdots < r_m = j$, $j = \omega$, such that $(\sigma, r_{l-1}, r_l) \mid P, 1 < l \quad m$, and $(\sigma', 0, |\sigma'|) \mid Q$, where

$$\sigma' = \sigma \quad (r_0, \ldots, r_m)$$

PITL formulas can be rewritten to their normal forms in the same way as in PPTL. With nite models, Bowman and Thompson [5, 10] have given a decision procedure for checking the satis ability of PITL formulas with projection. Table 2 illustrates a modi ed version of algorithm LNFG which can be used to construct NFGs for PITL formulas. Note that the models of PITL formulas can be precisely characterized by the NFGs, and nite labels are not needed for PITL. Further, algorithm SIMPLIFY can be executed directly for PITL. Moreover, a similar decision procedure for checking the satis ability of PITL formulas with nite or in nite models is presented based on algorithms LNFG and SIMPLIFY. In the following, an example is given to show how this decision procedure works.

Example 5 Check the satis ability of formula $\Box \quad p ; \varepsilon$.

The NFG of formula $\Box \quad p ; \varepsilon$ is constructed as shown in Fig. 20. However, no nodes can be deleted by algorithm SIMPLIFY. Therefore, the formula is satis able in PITL (but not in PPTL). In fact, this formula has no nite models but in nite models.

Discussion. The decision procedure given by Bowman and Thompson [5, 10] is a valuable contribution to PITL since it is the rst decision procedure for checking the satis ability of quanti er free PITL formulas with nite models. With this procedure, for a given formula P, the initial node of a Tableaux graph is labeled by P, *empty*; the empty ensures the procedure to terminate over a nite interval; then the graph is repeatedly constructed in two phases: using decomposition rules rst for all non-elementary formulas if any at a node and then the step rule with a node containing only elementary formulas and no empty; after that, unacceptable nodes in the graph are eliminated by using reducing rules repeatedly; nally, P is satis able iff there exists a path from pre-state (i.e. an initial node or son of the node containing only elementary formulas) to a node containing empty. In contrast with this procedure, our procedure has some advantages: for instance, it is able to check the satis ability for PITL formulas with both nite and in nite models; further, our procedure is simpler and more concise; given a formula P, roughly, the number of nodes in the NFG of P is the half of that in the Tableau graph of P. As you can see, the Tableau graph of q is of 12 nodes and 12 edges while the NFG of this formula has only 5 nodes and 6 edges as shown in Fig. 21(a) and (b) respectively.

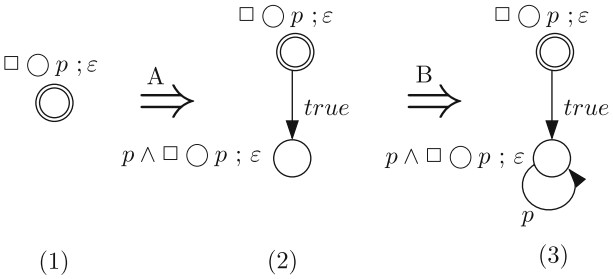
Table 12 Algorithm for constructing the NFG of PITL formulas

function $\mathrm{NFG}(P)$:
/* precondition: P is a PITL formula */
/* postcondition: $\mathrm{NFG}(P)$ computes NFG of P, $G = (CL(P), EL(P))$ */

```
begin function
    /*initialization*/
    CL(P) = { P}; EL(P) = φ; mark[P]=0; AddE = AddN =0;
    while there exists R  CL(P) \ { ε, false}, and mark[R]==0
     do mark[R]=1;                          /*marking R is decomposed*/
      Q = NF(R);
      case
        Q is ⋁_{j=1}^{h} Q_ej   ε:  AddE=1;      /* rst part of NF needs added*/
        Q is ⋁_{i=1}^{k} Q_i     Q_i : AddN=1; /*second part of NF needs added*/
        Q is ⋁_{j=1}^{h} Q_ej   ε   ⋁_{i=1}^{k} Q_i     Q_i: AddE=AddN=1;
                                    /*both parts of NF needs added*/
      end case
      if  AddE == 1 then                    /*add  rst part of NF*/
                  CL(P) = CL(P)  { ε};
                  EL(P) = EL(P)  ⋃_{j=1}^{h}{(R, Q_ej, ε)};
                  AddE=0;
      if  AddN == 1 then                    /*add second part of NF*/
            for i = 1 to k do ( if Q_i is P_1  ¬ P_1 or false
                        then mark[Q_i]=1;
                             /* Q_i needs not decomposed*/
                        else if Q_i  CL(P)
                        then mark[Q_i]=0;
                             /* Q_i needs decomposed*/
            CL(P) = CL(P)  ⋃_{i=1}^{k}{Q_i};
            EL(P) = EL(P)  ⋃_{i=1}^{k}{(R, Q_i, Q_i)};
            AddN=0;)
    end while
    return G;
end function
```

8 Conclusion

In this paper, we have presented decision procedures for checking the satis ability of both PPTL and PITL formulas with in nite models. This enables us to verify properties of concurrent systems with PPTL and PITL by means of model checking [21,22]. However, the famous model checker SPIN [21] cannot directly be used to check PPTL or PITL formulas due to the chop and projection constructs. Furthermore, with SPIN, the speci cation language PLTL is not powerful enough to describe all the regular properties which can be veri ed in SPIN [21,23,24]. For instance, it is impossible to describe the property that proposition p must hold at even states regardless of odd states over a run (sequence of states) [21,23]. Thus, to



A: $\Box \bigcirc p\ ; q \equiv \bigcirc(p \wedge \Box \bigcirc p\ ; q)$

B: $p \wedge \Box \bigcirc p\ ; q \equiv p \wedge (p \wedge \Box \bigcirc p\ ; q)$

Fig. 20 Constructing the NFG of $\Box \quad p\ ;\varepsilon$

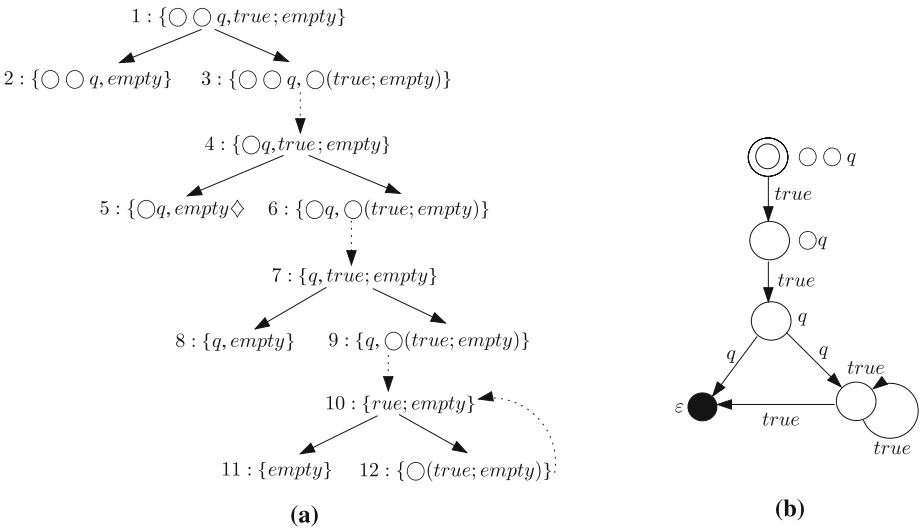


Fig. 21 Tableaux graph and NFG of formula q

capture a property that is not expressible in PLTL we need encode it directly into the Never Claim, but this is an error-prone process. Fortunately, it has been proved that these properties can be speci ed by more powerful logics with the chop operator [32].

Therefore, to check any regular properties automatically, a model checker for PPTL and PITL is required. By our experience, the decision procedures given in this paper are useful in this respect. Accordingly, we have developed a model checker based on SPIN for PPTL in C++ [30]. With this model checker, the negation of a property is rst described by a PPTL formula, and then an LNFG of the formula is constructed by the algorithm given in this paper. If the LNFG is not empty it is satis able and transformed to a Büchi automaton. Subsequently, the automaton is described in Never claim in terms of PROMELA. The model part is still described by PROMELA in the same way as in SPIN. This model checker works well but is merely a prototype at present. More development work is needed to improve it.

Besides the model checker, we are also motivated to develop a practical veri cation environment for PPTL with a set of supporting tools in the near future. Furthermore, it is

necessary for us to investigate the complexity of the decision algorithms so that comparisons can be given between PLTL and PPTL (or PITL) and improvements for the algorithms could be made. In addition, an axiomatization system for PPTL might be helpful for the veri cation of concurrent systems based on PPTL. Finally, temporal logic programming [11, 12, 15, 26, 30] with PTL also needs to be investigated in the future.

References

1. Moszkowski, B.C.: Reasoning about digital circuits. PhD Thesis, Stanford University. TRSTAN-CS-83-970 (1983)
2. Rosner, R., Pnueli, A.: A choppy logic. In: First annual IEEE symposium on logic in computer science, LICS, pp. 306–314 (1986)
3. Moszkowski, B.C.: A complete axiomatization of interval temporal logic with in nite time. In: 15th Annual IEEE symposium on logic in computer science (LICS'00), LICS, p. 241, (2000)
4. Chaochen, Z., Hoare, C.A.R., Ravn, A.P.: A calculus of duration. Inf. Process. Lett. 40(5), 269–275 (1991)
5. Bowman, H., Thompson, S.: A decision procedure and complete axiomatization of interval temporal logic with projection. J. Logic Comput. 13(2), 195–239 (2003)
6. Dutertre, B.: Complete proof systems for rst order interval temporal logic. In: Proceedings of LICS'95, pp. 36–43 (1995)
7. Wang, H., Xu, Q.: Temporal logics over in nite intervals. Technical Report 158, UNU/IIST, Macau (1999)
8. Halpern, J., Manna, Z., Moszkowski, B.: A hardware semantics based on temporal intervals. In: Proceedings of the 10th international conlloquium on automata, Languages and Programming, vol. 154. Springer, LNCS, Barcelona (1983)
9. Kono, S.: A combination of clausal and non-clausal temporal logic programs. In: Lecture notes in arti cial intelligence, vol. 897, pp. 40–57. Springer, Heidelberg (1995)
10. Bowman, H., Thompson, S.: A Tableau method for interval temporal logic with projection. In: de Swart, H. (ed.) TABLEAUX98, LNAI 1397, Springer, Berlin (1998)
11. Duan, Z.: An extended interval temporal logic and a framing technique for temporal logic programming. PhD thesis, University of Newcastle Upon Tyne (1996)
12. Duan, Z.: Temporal Logic and Temporal Logic Programming Language. Science press, Beijing (2006)
13. Duan, Z., Koutny, M., Holt, C.: Projection in temporal logic programming. In: Pfenning, F. (ed.) Proceedings of logic programming and automatic reasoning, Lecture Notes in Arti cial Intelligence, vol. 822, pp. 333–344. Springer, Heidelberg (1994)
14. Duan, Z., Koutny, M.: A framed temporal logic programming language. J. Comput. Sci. Technol. 19(3), 333–344 (2004)
15. Duan, Z., Yang, X., Kounty, M.: Semantics of framed temporal logic programs. In: Proceedings of ICLP 2005, vol. 3668, pp. 256–270. LNCS, Barcelona (2005)
16. Moszkowski, B.C.: Compositional reasoning about projected and in nite time. In: Proceeding of the rst IEEE international conference on engineering of complex computer systems (ICECCS'95), pp. 238–245. IEEE Computer Society Press (1995)
17. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer, Heidelberg (1992)
18. Duan, Z., Zhang, L.: A decision procedure for propositional projection temporal logic. Technical Report No.1, Institute of computing Theory and Technology, Xidian University, Xi'an, People's Republic of China, http://www.paper.edu.cn/en/paper.php?serial$_$number=200612-405 (2006)
19. Kripke, S.A.: Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9, 67–96 (1963)
20. Winskel, G.: The Formal Semantics of Programming Languages. Foundations of Computing. MIT, Cambridge
21. Holzmann, G.J.: The Model Checker Spin. IEEE Trans. Softw. Eng. 23(5), 279–295 (1997)
22. McMillan, K.L.: Symbolic Model Checking. Kluwer (1993)
23. Harel, D., Kozen, D., Parikh, R.: Process logic: expressiveness, decidability, completeness. J. Comput. Syst. Sci. 25(2), 144–170 (1982)
24. Chandra, A., Halpern, J., Meyer, A., Parikh, R.: Equations between regular terms and an application to process logic. SIAM J. Comput. 14(4), 935–942 (1985)
25. Moszkowski, B.: Executing Temporal Logic Programs. Cambridge University Press, Cambridge (1986)
26. Duan, Z.: Modelling and Analysis of Hybrid Systems. Science Press, Beijing (2005)

27. Duan, Z., Holcombe, M., Bell, A.: A logic for biosystems. Biosystems 55(1-3), 93–105 (2000)
28. Paech, B.: Gentzen-systems for propositional temporal logics. In: Borger, E., Kleine Buning, H., Richter, M.M. (eds.) Proceedings of the 2nd workshop on computer science logic, Duisburg (FRG), vol. 385, pp. 240–253. Springer, Heidelberg (1988)
29. Pnueli, A.: The temporal logic of programs. In: Proceedings of 18th IEEE symposium on foundations of computer science, pp. 46–57 (1977)
30. Tian, C., Duan, Z.: Model Checking Propositional Projection Temporal Logic Based on SPIN, ICFEM 2007, LNCS4789, pp. 246-265, Springer, Heidelberg (2007)
31. Kröger, F.: Temporal Logic of Programs. EATCS Monographs on Theoretical Computer Science, vol. 8. Springer, Heidelberg (1987)
32. Wolper, P.L.: Temporal logic can be more expressive. Inf. Control 56(1-2), 72–99 (1983)