Model Checking Propositional Projection Temporal Logic Based on SPIN[⋆]

Cong Tian and Zhenhua Duan

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

Abstract. This paper investigates a model checking algorithm for Propositional Projection Temporal Logic (PPTL) with finite models. To this end, a PPTL formula is transformed to a Normal Form Graph (NFG), and then a Nondeterministic Finite Automaton (NFA). The NFA precisely characterizes the finite models satisfying the corresponding formula and can be equivalently represented as a Deterministic Finite Automaton (DFA). When the system to be verified can be modeled as a DFA A_s, and the property of the system can be specified by a PPTL formula P, then $\neg P$ can be transformed to a DFA A_p. Thus, whether the system satisfies the property or not can be checked by computing the product automaton of A_s and A_p, and then checking whether or not the product automaton accepts the empty word. Further, this method can be implemented by means of the verification system SPIN.

Keywords: Model Checking, Propositional Projection Temporal Logic, Automaton, SPIN, Verification.

1 Introduction

Model checking is an important approach for verification of the properties of hardware, softwares, multi-agent systems, communication protocols, embedded systems and so forth. In the last two decades, several model checkers such as SPIN [15] and SMV [16] were developed with success. In particular, as a software verification system, SPIN has attracted a fairly broad group of users in both academia and industry. SPIN can be used as a full Propositional Linear Temporal Logic (PLTL) [2] model checking system, supporting checks of all correctness requirements expressible in linear time temporal logic. However, PLTL is not powerful enough to describe all the ω-regular properties which can be verified in SPIN [15]. 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) [20,15]. Thus, to capture a property that is not expressible in PLTL we need encode it directly into a Never Claim, but this is an error-prone process. Fortunately, it has been proved that these properties can be specified by more

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

© Springer-Verlag Berlin Heidelberg 2007

powerful logics with chop operator [20]. Within Propositional Projection Temporal Logic (PPTL) [6,7] and Propositional Interval Temporal Logic (PITL) [3], chop and projection operators are introduced. Thus, the above property can be specified. It has also been proved that the logic with chop operator has the expressive power of full regular expressions [20,2]. Therefore, we are motivated to investigate a model checking algorithm and the corresponding verification technique based on Spin for PPTL. Note that PPTL is an extension of PITL [6,7,9], our method can also be applied to PITL.

Within PPTL, many logic laws have been formalized and proved [6,7], and a decision procedure for checking satisfiability of PPTL formulas with infinite models has been given in [9]. Thus, model checking PPTL is decidable. The method presented in this paper is mainly inspired by our previous work [9]. For simplicity, we consider only PPTL formulas defined over finite intervals. The full logic will further be studied in the near future.

With our method, the model of the system to be verified is specified by a DFA A_s, and the property of the system is described by a PPTL formula P. Further, $\neg P$ is transformed to a Normal Form Graph (NFG), and then a Nondeterministic Finite Automaton (NFA). The NFA precisely characterizes the finite models satisfying P and can be equivalently represented as a Determined Finite Automaton (DFA) A_p. Thus, whether the system satisfies property P or not can be checked by computing the product automaton of A_s and A_p, and then checking whether or not the product automaton accepts the empty word. When implemented in Spin, the system is described in terms of ProMeLa which produces the automaton A_s when executed by the ProMeLa interpreter within Spin. The automaton A_p of $\neg P$ is also described as Never Claim in the syntax of ProMeLa. Thus, Spin can be employed to implement the model checking procedure.

Our method has several advantages. For instance, first, the method is based on the verification tool Spin. As known, Spin is a successful and widely used software model checking tool. So we can benefit from Spin; secondly, our method extends the function of Spin since specification language PPTL can be used in Spin. This enables us to verify systems with properties specified in PLTL and PPTL; finally, all the properties which can be verified in Spin can now be specified by PPTL, since logic with chop operator has the expressive power of full regular expressions.

The rest of the paper is organized as follows. The next section briefly presents the syntax and semantics of PPTL. Section 3 introduces the normal form of PPTL formulas. In Section 4, the definition of NFG and the algorithm for constructing NFG are given. Further, the upper bound of the number of nodes in NFGs is proved in details. Section 5 is devoted to the transformation from NFG to NFA. Further, in Section 6, the model checking method for PPTL is illustrated and how the method can be implemented in Spin are presented. In addition, simple examples are given to show how our method works. Finally, conclusions are drawn in Section 7.

2 Propositional Projection Temporal Logic

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

2.1 Syntax

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

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

where $p \in Prop$, P_1 ,..., P_m and P are all well-formed PPTL formulas. $\bigcirc$ (next) and *prj* (projection) are basic temporal operators. The abbreviations $true$, $false$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $true \overset{\text{def}}{=} P \vee \neg P$ and $false \overset{\text{def}}{=} P \wedge \neg P$ for any formula P. Also we have the following derived formulas:

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

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

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

$$\begin{array}{rl}
 & (P_1, ..., (P_i, ..., P_j)^k, ..., P_m)\ prj\ Q \\
\overset{\text{def}}{=} & (P_1, ..., \underbrace{(P_i, ..., P_j), ..., (P_i, ..., P_j)}_{k\ \text{times}}, ..., P_m)\ prj\ Q
\end{array}$$

where $1 \leq i \leq j \leq m, k \geq 0$. When $i = 1$ and $j = m$, we have,

$$(P_1, ..., P_m)^k\ prj\ Q \overset{\text{def}}{=} (\ \underbrace{(P_1, ..., P_m), ..., (P_1, ..., P_m)}_{k\ \text{times}}\ )\ prj\ Q$$

Further, the following formulas can be derived,

$$\begin{aligned} \varepsilon\ prj\ Q &\overset{\text{def}}{=} (P_1, ..., P_m)^k\ prj\ Q \quad \text{if } k = 0 \\ (P_1, ..., P_m)^+\ prj\ Q &\overset{\text{def}}{=} (P_1, ..., P_m)^k\ prj\ Q \quad \text{if } k > 0 \\ (P_1, ..., P_m)^*\ prj\ Q &\overset{\text{def}}{=} (P_1, ..., P_m)^k\ prj\ Q \quad \text{if } k \geq 0 \end{aligned}$$

In particular, when $m = 1$, let $P_1 \equiv P$, we have,

$$\begin{aligned} \varepsilon\ prj\ Q &\overset{\text{def}}{=} P^k\ prj\ Q \quad \text{if } k = 0 \\ P^+\ prj\ Q &\overset{\text{def}}{=} P^k\ prj\ Q \quad \text{if } k > 0 \\ P^*\ prj\ Q &\overset{\text{def}}{=} P^k\ prj\ Q \quad \text{if } k \geq 0 \end{aligned}$$

Accordingly, in PITL, if $P\ proj\ Q$ [3] holds for some P and Q, then we can express it using prj construction in PPTL,

$$(P^*\ prj\ (Q; r \wedge \varepsilon)) \wedge halt(r)$$

where $r \in Prop$ does not appear in P and Q.

2.2 Semantics

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

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

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

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

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

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

An interpretation is a quadruple $\mathcal{I} = (\sigma, i, k, j)$[1], where σ is an interval, i, k are integers, and j an integer or ω such that $i \leq k \preceq j \leq |\sigma|$. We use the notation $(\sigma, i, k, j) \models P$ to denote that formula P is interpreted and satisfied over the subinterval $< s_i, ..., s_j >$ of σ with the current state being s_k. The satisfaction relation ($\models$) is inductively defined as follows:

$\mathcal{I} \models p$ iff $s_k[p] = \mathit{true}$, for any given atomic 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 $(\sigma, i, k+1, j) \models P$
$\mathcal{I} \models (P_1, ..., P_m)\ \mathit{prj}\ Q$ if there exist integers $k = r_0 \leq r_1 \leq ... \leq r_m \leq j$ such that $(\sigma, 0, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_{l-1}, r_l) \models P_l, 1 < l \leq m,$ and $(\sigma', 0, 0, |\sigma'|) \models Q$ for one of the following σ' :
(a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, ..., r_m) \cdot \sigma_{(r_m+1..j)}$ or
(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, ..., r_h)$ for some $0 \leq h \leq m$

2.3 Satisfaction and Validity

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

Two formulas, P and Q, are equivalent, denoted by $P \equiv Q$, if $\models \Box(P \leftrightarrow Q)$. A formula P is called a state formula if it contains no temporal operators, a terminal formula if $P \equiv P \wedge \varepsilon$, a non-local formula if $P \equiv P \wedge \mathit{more}$, and a local formula if P is a state or terminal formula.

3 Normal Form of PPTL

Definition 1. Let Q be a PPTL formula and Q_p denote the set of atomic propositions appearing in Q. The normal form of Q is defined as follows,

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

where $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$, $l = |Q_p|$, $1 \leq n$ (also n_0) $\leq 3^l$, $1 \leq m$ (also m_0) $\leq l$; $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, $\dot{r}$ denotes r or $\neg r$; Q_i' is an arbitrary PPTL formula[2].

In some circumstances, for convenience, we write $Q_e \wedge \varepsilon$ instead of $\bigvee_{j=1}^{n_0}(Q_{ej} \wedge \varepsilon)$ and $\bigvee_{i=1}^{r}(Q_i \wedge \bigcirc Q_i')$ instead of $\bigvee_{i=1}^{n}(Q_{ci} \wedge \bigcirc Q_i')$. Thus,

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

where Q_e and Q_i are state formulas or true.

[1] Parameter i is used to handle past operators and redundant with the current version of the underlying logic. However, to be consistent with previous expositions, it is kept in the interpretation.

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

Definition 2. In a normal form, if $\bigvee_i Q_i \equiv true$ and $\bigvee_{i \neq j}(Q_i \wedge Q_j) \equiv false$, then this normal form is called a complete normal form.

The complete normal form plays an important role in transforming the negation of a PPTL formula into its normal form. For example, if P has been rewritten to its complete normal form:

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

then we have,

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

The normal form enables us to rewrite the formula into two parts: the terminating part $\bigvee_{j=1}^{n_0}(Q_{ej} \wedge \varepsilon)$ and the future part $\bigvee_{i=1}^{n}(Q_{ci} \wedge \bigcirc Q_i')$. For any PPTL formula P, P can be rewritten into its normal form and complete normal form. The details of the proofs and the algorithms for transforming PPTL formulas into normal forms and complete normal forms can be found in [8,9].

4 Normal Form Graph of PPTL

To transform a PPTL formula to an automaton that accepts precisely the sequences of sets of propositions satisfying the formula, we first construct a directed graph, called a Normal Form Graph (NFG), for the formula according to the normal form.

4.1 Definition of NFG

For a PPTL formula P, the NFG of P is a labeled 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 $CL(P)$, each node is specified by a formula in PPTL, while in $EL(P)$, each edge is identified by a triple (Q, Q_e, R). Where Q and R are nodes and Q_e is the label of the directed arc from Q to R. $CL(P)$ and $EL(P)$ of G can be inductively defined below.

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

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

$CL(P)$ and $EL(P)$ are only generated by 1 and 2. The NFG of formula P is the directed graph $G = (CL(P), EL(P))$.

In the NFG of P, the root node P is denoted by a double circle, ε node by a small black dot, and each of other nodes by a single circle. Each of the edges is denoted by a directed arc connecting two nodes. Fig.1 shows an example of NFG.

4.2 Constructing NFG

In the following, algorithm NFG for constructing the NFG of a PPTL formula is presented. It is actually a sketch of the implementation of Definition 3. The

```
function NFG(P):
/* precondition: $P$ is a PPTL formula*/
/* postcondition: NFG(P) computes NFG of $P$, $G = (CL(P), EL(P))$*/
begin function
   Create root node $P$;
   $Mark(P)$=0; AddE = AddN =0;
   while there exists node $Q$ (not $\varepsilon$ and $false$) in the NFG and $Mark(Q) == 0$
         Mark(Q)=1;                                   /*marking $R$ is decomposed*/
         Rewrite $Q$ to its normal form;
         case
           $Q$ is $\bigvee_{j=1}^{h} Q_{ej} \wedge \varepsilon$:  AddE=1;            /*need to add first part of NF*/
           $Q$ is $\bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$ : AddN=1;      /*second part of NF needs added*/
           $Q$ is $\bigvee_{j=1}^{h} Q_{ej} \wedge \varepsilon \vee \bigvee_{i=1}^{k} Q_i \wedge \bigcirc Q'_i$: AddE=AddN=1;
                                                    /*both parts of NF needs added*/
         end case
         if  AddE == 1                                  /*add first part of NF*/
             if  there exists no $\varepsilon$ node
                 create node $\varepsilon$;
             for $1 \leq j \leq h$,
                   create edge $(Q, Q_{ej}, \varepsilon)$;
             end for
             AddE=0;
         if  AddN == 1                                 /*add second part of NF*/
             for $1 \leq i \leq k$
                   if $Q'_i \notin CL(P)$
                       create node $Q'_i$;
                        if  $Q'_i$ is $false$
                             $mark(Q'_i)$=1;              /*$Q'_i$ not decomposed*/
                        else $mark(Q'_i)$=0;                /*$Q'_i$ needs to be considered*/
                    create edge $(Q, Q_i, Q'_i)$;
              end for
              AddN=0;
      end while
      return $G$;
end function
```

algorithm uses *mark*[] to indicate whether or not a formula needs to be decomposed. If $mark[P] == 0$ (unmarked), P needs further to be decomposed, otherwise $mark[P] == 1$ (marked), thus P has been decomposed or needs not to be precessed. Note that algorithm NF employs algorithm NF [8] to transform a formula into its normal form. Further, in the algorithm, two global boolean variables AddE and AddN are employed to indicate whether or not terminating and future parts in the normal form are encountered respectively. Note also that the algorithm only deals with formulas in a pre-prepared form in which only $\vee$, $\wedge$ and $\neg$, as well as temporal operators $\bigcirc$, ; and prj are contained. Others such as $\rightarrow$, $\leftrightarrow$, $\Diamond$, $\Box$, $\neg\neg$ etc. can be eliminated since they can be expressed by the basic operators. Algorithm NFG is slightly different from the one we gave in [9], since only finite models are considered in this paper.

Example 1. Construct the NFG of formula $\neg(true;\neg\bigcirc q)\vee p\wedge\bigcirc q$ by algorithm NFG.

As depicted in Fig.1, initially, the root node $\neg(true;\neg\bigcirc q)\vee p\wedge\bigcirc q$ is created and denoted by v_0; rewrite $\neg(true;\neg\bigcirc q)\vee p\wedge\bigcirc q$ to its normal form, $\neg(true;\neg\bigcirc q)\vee p\wedge\bigcirc q\equiv\bigcirc(q\wedge\neg(true;\neg\bigcirc q))\vee p\wedge\bigcirc q$, nodes $q\wedge\neg(true;\neg\bigcirc q)$ and q

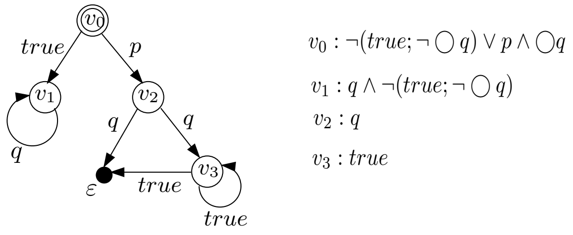


Fig. 1. NFG of formula $\neg(true;\neg\bigcirc q)\vee p\wedge\bigcirc q$

are created and denoted by v_1 and v_2 respectively; also edges $(v_0, true, v_1)$ and (v_0, p, v_2) are created; further, rewrite $q\wedge\neg(true;\neg\bigcirc q)$ to its normal form, $q\wedge\neg(true;\neg\bigcirc q)\equiv q\wedge\bigcirc(q\wedge\neg(true;\neg\bigcirc q))$, edge (v_1, q, v_1) is created; subsequently, rewrite q to its normal form, $q\equiv q\wedge\varepsilon\vee q\wedge\bigcirc true$, node $true$, denoted by v_3 and ε node are created, also edges (v_2, q, ε) and (v_2, q, v_3) are created; finally, rewrite $true$ to its normal form, $true\equiv\varepsilon\vee\bigcirc true$, edges $(v_3, true, \varepsilon)$ and $(v_3, true, v_3)$ are created.

4.3 Upper Bound of NFGs

For an arbitrary PPTL formula Q, if Q is rewritten into its normal form as follows,

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

then ε or each of Q_i' is called a succ-formula of Q. The set of succ-formulas of Q is denoted by $succ(Q)$.

The length of a PPTL formula Q is denoted by $length(Q)$ (or $|Q|$), and is inductively defined in Definition 4. Note that we need consider only the operators $\neg$, $\bigcirc$, $\vee$, $\wedge$, ; and prj supported in algorithm NFG.

Definition 4. Let θ be an atomic proposition, or derived formula $true$, $false$ or ε, $length(\theta)$=1. Suppose P_i, $1\leq i\leq m$, and Q are PPTL formulas, $length(P_i)=n_i$ and $length(Q)=n$, then

- For unary operators $\neg$ or $\bigcirc$ denoted by Θ_1, $length(\Theta_1 Q) = n+1$
- For binary operators ;, $\vee$ or $\wedge$ denoted by Θ_2, $length(P_1\Theta_2 P_2) = n_1+n_2+1$
- For operator prj, $length((P_1,...,P_m)\ prj\ Q) = n_1+...+n_m+n+m$

Roughly speaking, the length of a formula P is the number of the symbols appearing in P.

Lemma 1. Suppose for each formula P_i and Q, $0 \leq i \leq m$, the length of each succ-formula of P_i (or Q) is not larger than the length of P_i (or Q), then the length of each succ-formula of $(P_1,...,P_m)\ prj\ Q$ is not larger than the length of $(P_1,...,P_m)\ prj\ Q$.

Proof. The proof proceeds by induction on m. Suppose P_1 and Q are rewritten into their normal forms,

$$\begin{aligned} P_1 &\equiv P_{1e}\wedge\varepsilon \vee \bigvee_{i=1}^{n}(P_{1i}\wedge\bigcirc P'_{1i}) \\ Q &\equiv Q_e\wedge\varepsilon\vee\bigvee_{k=1}^{n'}(Q_k\wedge\bigcirc Q'_k) \end{aligned}$$

By hypothesis, $|\varepsilon| \leq |P_1|$, $|\varepsilon| \leq |Q|$, for each i, $1 \leq i \leq n$, $|P'_{1i}| \leq |P_1|$, and for each k, $1 \leq k \leq n'$, $|Q'_k| \leq |Q|$. Since,

$$\begin{aligned} P_1\ prj\ Q \equiv\ & P_{1e}\wedge Q_e\wedge\varepsilon\vee\bigvee_{i=1}^{n}(P_{1i}\wedge Q_e\wedge\bigcirc P'_{1i}) \\ &\vee\bigvee_{i=1}^{n}\bigvee_{k=1}^{n'}(P_{1i}\wedge Q_k\wedge\bigcirc(P'_{1i};Q'_k)) \\ &\vee\bigvee_{k=1}^{n'}(P_{1e}\wedge Q_k\wedge\bigcirc Q'_k) \end{aligned}$$

So, $succ(P_1\ prj\ Q) = \{\varepsilon\}\cup\bigcup_{i=1}^{n}(P'_{1i})\cup\bigcup_{i=1}^{n}\bigcup_{k=1}^{n'}(P'_{1i};Q'_k)\cup\bigcup_{k=1}^{n'}(Q'_k)$. Obviously, $|\varepsilon| \leq |P_1\ prj\ Q|$; for each P'_{1i} and Q'_k, $|P'_{1i}| \leq |P_1| \leq |P_1\ prj\ Q|$ and $|Q'_k| \leq |Q| \leq |P_1\ prj\ Q|$; for each $P'_{1i};Q'_k$, $|P'_{1i};Q'_k| = |P'_{1i}\ prj\ Q'_k| \leq |P_1\ prj\ Q|$. Suppose $(P_2,...,P_m)\ prj\ Q$ has been rewritten to its normal form,

$$(P_2,...,P_m)\ prj\ Q \equiv R_e\wedge\varepsilon\vee\bigvee_{j=1}^{t}(R_j\wedge\bigcirc R'_j)$$

And for ε and each R'_j, $|\varepsilon| \leq |(P_2,...,P_m)\ prj\ Q|$, $|R'_j| \leq |(P_2,...,P_m)\ prj\ Q|$. Since,

$$\begin{aligned} &(P_1,...,P_m)\ prj\ Q \\ \equiv\ & P_{1e}\wedge R_e\wedge\varepsilon\vee\bigvee_{j=1}^{t}(P_{1e}\wedge R_j\wedge\bigcirc R'_j) \\ &\vee\bigvee_{i=1}^{n}\bigvee_{k=1}^{n'}(P_{1i}\wedge Q_k\wedge\bigcirc(P'_{1i};((P_2,...,P_m)\ prj\ Q'_k))) \\ &\vee\bigvee_{i=1}^{n}(P_{1i}\wedge Q_e\wedge\bigcirc(Q'_{1i};P_2;...;P_m)) \end{aligned}$$

So, $succ((P_1,...,P_m)\ prj\ Q) = \{\varepsilon\}\cup\bigcup_{j=1}^{t}(R'_j)\cup\bigcup_{i=1}^{n}\bigcup_{k=1}^{n'}(P'_{1i};((P_2,...,P_m)\ prj\ Q'_k))\cup\bigcup_{i=1}^{n}(Q'_{1i};P_2;...;P_m)$. Obviously, $|\varepsilon| \leq |(P_1,...,P_m)\ prj\ Q|$; for each R'_j, $|R'_j| \leq |(P_2,...,P_m)\ prj\ Q| \leq |(P_1,...,P_m)\ prj\ Q|$; for each $P'_{1i};((P_2,...,P_m)\ prj\ Q'_k)$, $|P'_{1i};((P_2,...,P_m)\ prj\ Q'_k)| \leq |(P_1,...,P_m)\ prj\ Q|$; for each $Q'_{1i};P_2;...;P_m$, $|Q'_{1i};P_2;...;P_m| \leq |(P_2,...,P_m)\ prj\ Q| \leq |(P_1,...,P_m)\ prj\ Q|$. Thus, the lemma holds.

Lemma 2. For any PPTL formula P, when rewritten into its normal form, the length of each succ-formula of P is not larger than the length of P.

Proof. The proof proceeds by induction on the structure of PPTL formulas composed of the operators $\neg$, $\bigcirc$, $\wedge$, $\vee$, ; and prj which are supported in algorithm NFG.

Base case: P is an atomic proposition p. Rewrite p to its normal form, $p \equiv p \wedge \varepsilon \vee p \wedge \bigcirc true$. For the succ-formulas ε and $true$, $|\varepsilon| \leq |p|$, $|true| \leq |p|$.

Induction step: Suppose for each formula P_i (or Q), $0 \leq i \leq m$, when rewritten into its normal form, the length of each succ-formula of P_i (or Q) will be not larger than the length of P_i (or Q). Then,

(1) $P \equiv \bigcirc P_1$: $|P_1| < 1 + |P_1| = |P|$.

(2) $P \equiv \neg P_1$: If the complete normal form of P_1 is as follows,

$$P_1 \equiv (P_{1e} \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (P_{1i} \wedge \bigcirc P'_{1i})$$

then,

$$\neg P_1 \equiv (\neg P_{1e} \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (P_{1i} \wedge \bigcirc \neg P'_{1i})$$

By hypothesis, $|\varepsilon| \leq |P_1|$, $|P'_{1i}| \leq |P_1|$, $1 \leq i \leq r$. Thus, we have $|\varepsilon| \leq |P_1| < 1 + |P_1| = |\neg P_1|$, $|\neg P'_{1i}| = 1 + |P'_{1i}| \leq 1 + |P_1| = |\neg P_1|$, $1 \leq i \leq r$.

(3) $P \equiv P_1 \vee P_2$: Let

$$P_1 \equiv (P_{1e} \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (P_{1i} \wedge \bigcirc P'_{1i})$$
$$P_2 \equiv (P_{2e} \wedge \varepsilon) \vee \bigvee_{j=1}^{k} (P_{2j} \wedge \bigcirc P'_{2j})$$

Then,

$$P_1 \vee P_2 \equiv (P_{1e} \vee P_{2e}) \wedge \varepsilon \vee \bigvee_{i=1}^{r} (P_{1i} \wedge \bigcirc P'_{1i}) \vee \bigvee_{j=1}^{k} (P_{2j} \wedge \bigcirc P'_{2j})$$

By hypothesis, $|\varepsilon| \leq |P_1|$, $|\varepsilon| \leq |P_2|$, $|P'_{1i}| \leq |P_1|$, $1 \leq i \leq r$, and $|P'_{2j}| \leq |P_2|$, $1 \leq j \leq k$. Thus, we have $|\varepsilon| \leq |P_1| < |P_1| + |P_2| + 1 = |P_1 \vee P_2|$, $|P'_{1i}| \leq |P_1| < |P_1| + |P_2| + 1 = |P_1 \vee P_2|$, $1 \leq i \leq r$, and $|P'_{2i}| \leq |P_2| < |P_1| + |P_2| + 1 = |P_1 \vee P_2|$.

(4) $P \equiv P_1 \wedge P_2$: Let

$$P_1 \equiv (P_{1e} \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (P_{1i} \wedge \bigcirc P'_{1i})$$
$$P_2 \equiv (P_{2e} \wedge \varepsilon) \vee \bigvee_{j=1}^{k} (P_{2j} \wedge \bigcirc P'_{2j})$$

Then,

$$P_1 \wedge P_2 \equiv (P_{1e} \wedge P_{2e}) \wedge \varepsilon \vee \bigvee_{i=1}^{r} \bigvee_{j=1}^{k} (P_{1i} \wedge P_{2j} \wedge \bigcirc (P'_{1i} \wedge P'_{2j}))$$

By hypothesis, $|\varepsilon| \leq |P_1|$, $|\varepsilon| \leq |P_2|$, $|P'_{1i}| \leq |P_1|$, $1 \leq i \leq r$, and $|P'_{2j}| \leq |P_2|$, $1 \leq j \leq k$. Thus, we have $|\varepsilon| \leq |P_1| < |P_1| + |P_2| + 1 = |P_1 \wedge P_2|$, $|P'_{1i} \wedge P'_{2j}| = |P'_{1i}| + |P'_{2j}| + 1 \leq |P_1| + |P_2| + 1 = |P_1 \wedge P_2|$, $1 \leq i \leq r$ and $1 \leq j \leq k$.

(5) $P \equiv P_1 ; P_2$: Let

$$P_1 \equiv (P_{1e} \wedge \varepsilon) \vee \bigvee_{i=1}^{r} (P_{1i} \wedge \bigcirc P'_{1i})$$
$$P_2 \equiv (P_{2e} \wedge \varepsilon) \vee \bigvee_{j=1}^{k} (P_{2j} \wedge \bigcirc P'_{2j})$$

Then,

$$P_1; P_2 \equiv P_{1e} \wedge P_{2e} \wedge \varepsilon \vee P_{1e} \wedge \bigvee_{j=1}^{k}(P_{2j} \wedge \bigcirc P'_{2j}) \vee \bigvee_{i=1}^{r}(P_{1i} \wedge \bigcirc(P'_{1i}; P_2))$$

By hypothesis, $|\varepsilon| \leq |P_1|$, $|\varepsilon| \leq |P_2|$, $|P'_{1i}| \leq |P_1|$, $1 \leq i \leq r$, and $|P'_{2j}| \leq |P_2|$, $1 \leq j \leq k$. Thus, we have $|\varepsilon| \leq |P_1| < |P_1| + |P_2| + 1 = |P_1; P_2|$, $|P'_{2j}| \leq |P_2| < |P_1| + |P_2| + 1 = |P_1; P_2|$, $1 \leq i \leq r$, $|P'_{1i}; P_2| = |P'_{1i}| + |P_2| + 1 \leq |P_1| + |P_2| + 1 = |P_1; P_2|$.
(6) $P \equiv (P_1, ..., P_m)\ prj\ Q$: The conclusion has been proved in Lemma 1.

Theorem 3. For any PPTL formula Q, let $|Q| = n$, and Q_p denote the set of atomic propositions appearing in Q, and $|Q_p| = l$. Let the NFG of Q be $G = (CL(Q), EL(Q))$. Then we have $|CL(Q)| \leq (10 + l)^n$.

Proof. By algorithm Nfg, the nodes of the NFG of Q are generated by repeatedly rewriting the new generated succ-formulas into their normal forms. Further, Lemma 2 confirms that when written into the normal form, the length of each succ-formula of Q is not larger than the length of Q. Moreover, each node (formula) in the NFG of Q is composed of basic connectives, $\neg, \wedge, \vee, \bigcirc, ;$, prj, and comma $(,)$[3] brought forth by prj, atomic propositions appearing in Q, as well as $true$ and $false$. Accordingly, there are at most $(9 + l)$ symbols possibly appearing in a formula. In addition, each formula is no longer than n. Hence, by the principle of permutation and combination, at most $(10 + l)^n$ formulas (as nodes) can appear in the NFG of Q, leading to $|CL(Q)| \leq (10 + l)^n$.

In the NFG constructed by algorithm Nfg for formula Q, a finite path, $\Pi = \langle Q, Q_e, Q_1, Q_{1e}, ..., \varepsilon\rangle$, is an alternating sequence of nodes and edges from the root to ε node. Actually, a finite path in the NFG of formula Q corresponds to a finite model of Q. The fact is concluded in [9].

5 Nondeterministic Finite Automata of PPTL

In this section, we show how to build a Nondeterministic Finite Automaton from an NFG. First, let us recall the definition of Nondeterministic Finite Automaton [12].

5.1 Nondeterministic Finite Automata

Definition 5. A Nondeterministic Finite Automaton is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where,
- Q is a finite set of states
- Σ is a finite set of input symbols
- $q_0 \in Q$, is the start state
- $F \subseteq Q$, is the set of final (or accepting) states
- δ, a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$

[3] Here , is used in the prj construct.

For an NFA, the transition function δ is extended to a function $\hat{\delta}$ that takes a state q and a string of input symbols w as its input, and returns the set of states in Q if it starts in state q and successfully processes the string w. The NFA accepts a string w if it is possible to make any sequence of choices of next state, while reading the characters of w, and go from the start state to any accepting state. The fact that other choices using the input symbols of w lead to a non-accepting state, or do not lead to any state at all (i.e., the sequence of states "dies"), doses not prevent w from being accepted by the NFA as a whole. Formally, for an NFA $A = (Q, \Sigma, \delta, q_0, F)$, then

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$$

That is, $L(A)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

5.2 Constructing NFAs from NFGs

For a PPTL formula P, let P_p be the set of atomic propositions appearing in P, and $|P_p| = l$. Further, we define sets A_i $(1 \leq i \leq l)$ as follows,

$$A_i = \{\{\dot{q_{j_1}}, ..., \dot{q_{j_i}}\} \mid q_{j_k} \in P_p, 1 \leq k \leq i\}$$

Thus, the alphabet Σ for the DFA of formula P can be defined as follows,

$$\Sigma = \bigcup_{i=1}^{l} A_i \cup \{true\}$$

It can be proved that $|\Sigma| = 3^l$.

Let q_k be an atomic proposition, we define a function $atom(\bigwedge_{k=1}^{m_0} \dot{q}_k)$ for picking up atomic propositions or their negations appearing in $\bigwedge_{k=1}^{m_0} \dot{q}_k$ as follows,

$$\begin{aligned} atom(true) &= \{true\} \\ atom(\dot{q_1}) &= \begin{cases} \{q_1\}, \text{ if } \dot{q_1} \equiv q_1 \\ \{\neg q_1\}, \text{ otherwise} \end{cases} \\ atom(\textstyle\bigwedge_{k=1}^{m_0} \dot{q}_k) &= atom(\dot{q_1}) \cup atom(\textstyle\bigwedge_{k=2}^{m_0} \dot{q}_k) \end{aligned}$$

Accordingly, algorithm NFG-NFA is given for obtaining an NFA from the NFG, $G = (CL(P), EL(P))$, of PPTL formula P. In the algorithm, each node in the NFG is transformed as a state in the corresponding NFA; each edge (P_i, P_e, P_j) forms a transition in the NFA, $P_j \in \delta(P_i, atom(P_e))$; the root node P and ε node forms the start state q_0 and the accepting state respectively. Alphabet Σ is defined as above. Further, as proved, the number of the nodes in the NFG of P meets $|CL(P)| \leq (10+l)^n$, so does the number of states in NFA, $|Q| \leq (10+l)^n$.

Example 2. (Continue Example 1) Construct NFA of formula $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$.

```
Function NFG-NFA($G$)
/* precondition: $G = (CL(P), EL(P))$ is an NFG of PPTL formula $P$*/
/* postcondition: NFG-NFA($G$) computes an NFA $A = (Q, \Sigma, \delta, q_0, F)$ from G*/
begin function
    $Q = \phi$; $F = \phi$; $q_0 = \{P\}$; $\delta = \phi$;
    for each node $P_i \in CL(P)$,
        add a state $P_i$ to $Q$, $Q = Q \cup \{P_i\}$;
        if $P_i$ is $\varepsilon$, $F = F \cup \{P_i\}$;
    end for
    for each edge $(P_i, P_e, P_j) \in EL(P)$,
        $P_j \in \delta(P_i, atom(P_e))$;
    end for
    return $A = (Q, \Sigma, \delta, q_0, F)$
end function
```

By algorithm NFG-NFA, the NFA $A = (Q, \Sigma, \delta, q_0, F)$ for formula $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$ can be obtained from the NFG G built with Example 1 as follows,

- Q=$\{q_0, q_1, q_2, q_3, q_4\}$ is obtained from the set of nodes in G, $\{v_0, v_1, v_2, v_3, \varepsilon\}$
- q_0 is v_0 in G
- F=$\{q_4\}$, since q_4 is the ε node in G
- a_0=$atom(ture)$=$\{true\}$, a_1=$atom(q)$=$\{q\}$, a_2=$atom(p)$=$\{p\}$; $\delta(q_0, a_0)$=$\{q_1\}$, $\delta(q_0, a_2)$=$\{q_2\}$, $\delta(q_1, a_1)$=$\{q_1\}$, $\delta(q_2, a_1)$=$\{q_3\}$, $\delta(q_2, a_0)$=$\{q_4\}$, $\delta(q_3, a_0)$=$\{q_3, q_4\}$

Thus, A is depicted in Fig.2.

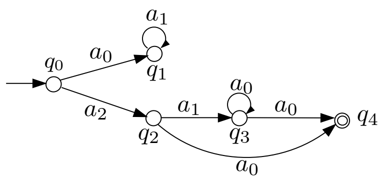


Fig. 2. NFA for $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$

Given a PPTL formula P, let $M(P)$ denote the set of finite models of P, $G(P)$ the NFG of P, and $A(P)$ the DFA of P. According to algorithm NFG-NFA, for each finite path $\Pi = \langle P, P_e, P_1, P_{1e}, ..., P_{ie}, \varepsilon\rangle$ in $G(P)$, there exists a word $w = atom(P_e)atom(P_{1e})...\ atom(P_{ie})$ accepted by $A(P)$. Further, for an arbitrary word $w = a_0a_1...a_i$ accepted by $A(P)$, there exists a finite model $\sigma =< s_0, s_1, ..., s_i >$ in $M(P)$, where if atomic proposition $q \in a_0$, $s_i[q] = true$, otherwise if $\neg q \in a_0$, $s_i[q] = false$. Moreover, in [9], we have proved that for any finite model in $M(P)$, there exists a finite path in $G(P)$, and also for any finite path in $G(P)$, there exists a corresponding model in $M(P)$. So the relationship among $M(P)$, $G(P)$ and $A(P)$ is shown in Fig.3. Thus, we can conclude that

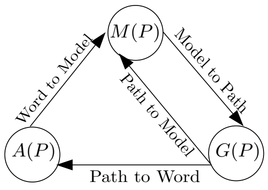


Fig. 3. Relationship among models, NFGs and DFAs

the DFA of formula P precisely characterizes the finite models of P. In a sense, the $M(P)$, $G(P)$ and $A(P)$ are equivalent.

6 Model Checking PPTL Based on SPIN

6.1 Model Checking PPTL Based on SPIN

Similar to the traditionally automata based model checking algorithm for PLTL [13], with our approach, the system M is modeled as a DFA, while the property is specified by a PPTL formula P. To check whether M satisfies P or not, $\neg P$ is transformed into an NFG, and further an NFA. The NFA can be equivalently represented as a DFA. Thus, the model checking procedure can be done by computing the product of the two automata and then deciding whether the words accepted by the product automaton is empty or not as shown in Fig. 4. If

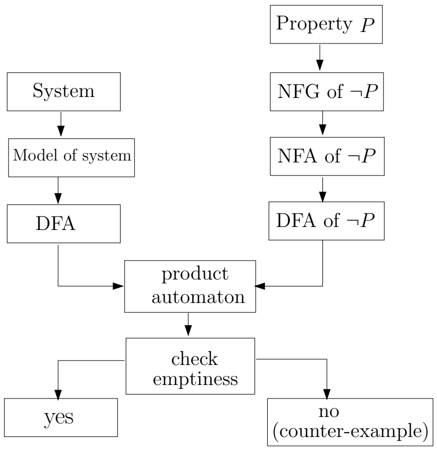


Fig. 4. Model checking PPTL

the words accepted by the product automaton is empty, the system can satisfy the property, otherwise the system cannot satisfy the property, and counter-examples are found.

To implement our method with Spin, the model of the system is still specified in ProMeLa. The property of the system is expressed by a PPTL formula. The negation of the formula is transformed into an NFG and then an NFA. Further, we transform the NFA to a DFA. By describing the DFA in terms of Never Claim, Spin can be employed to complete the model checking procedure.

6.2 Case Studies

Example 3. The property "p is true at every odd position" is a typical example for showing the limitation of the expressive power of PLTL. Here, we present a simple system which has this property first; then specify the property by PPTL; finally illustrate how the system can be checked with our method. In Fig.5, a

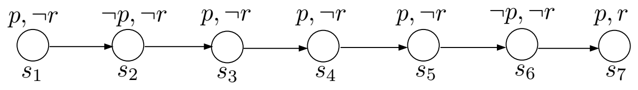
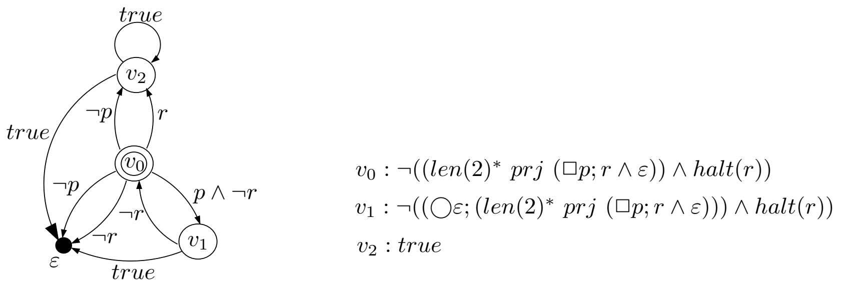


Fig. 5. Model of the system

system is shown as a Kripke structure, where p holds at states s_1, s_3, s_4, s_5 and s_7; $\neg p$ holds at states s_2 and s_6; $\neg r$ holds over the sequence except for the final state s_7. The system has a property that p holds at every odd state. The property can be specified by the following PPTL formula,

$$(len(2)^*\ prj\ (\Box p; r \wedge \varepsilon)) \wedge halt(r)$$

Accordingly, the NFG of formula $\neg((len(2)^*\ prj\ (\Box p; r \wedge \varepsilon)) \wedge halt(r))$ can be constructed as shown in Fig.6. And the corresponding NFA and DFA are shown



true
v_2
$\neg p$
r
true
v_0
$\neg p$
$p \wedge \neg r$
$\neg r$
$\neg r$
v_1
ε
true
$v_0 : \neg((len(2)^*\ prj\ (\Box p; r \wedge \varepsilon)) \wedge halt(r))$
$v_1 : \neg((\bigcirc\varepsilon; (len(2)^*\ prj\ (\Box p; r \wedge \varepsilon))) \wedge halt(r))$
$v_2 : true$

Fig. 6. NFG of $\neg((len(2)^*\ prj\ (\Box p; r \wedge \varepsilon)) \wedge halt(r))$

in Fig.7 (a) and (b) respectively.

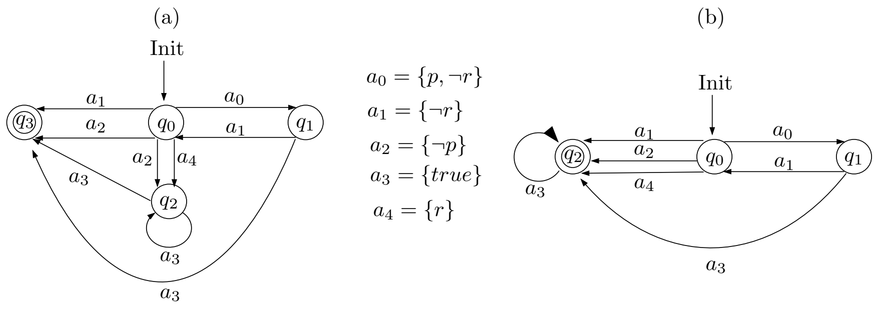


Fig. 7. NFA and DFA of $\neg((len(2)^* \ prj \ (\Box p; r \wedge \varepsilon)) \wedge halt(r))$

Further, the DFA can be expressed in Never Claim as shown in Fig.8. When implemented in SPIN, it outputs that the system satisfies the property.

```
Never{/*¬((len(2)* prj (□p; r)) ∧ halt(r))*/
        T0_init:
             if
             ::((!r)||(!p)||(r)) → goto accept_all
             ::((p) && (!r))→ goto T0_S2
             fi;
        T0_S2:
             if
             ::(!r)→ goto T0_init
             ::(1)→ goto accept_all
             fi
        accept_all:
             if
             ::skip
             ::(1)→ goto accept-all
             fi
        }
```

Fig. 8. Never Claim of $\neg((len(2)^* \ prj \ (\Box p; r \wedge \varepsilon)) \wedge halt(r))$

Example 4. This example shows how Needham-Schroeder protocol [18] can be verified by our method. In the protocol, two agents A(lice) and B(ob) try to establish a common secret over an insecure channel in such a way that both are convinced of each other's presence and no intruder can get hold of the secret without breaking the underlying encryption algorithm. The protocol is pictorially represented in Fig.9. It requires the exchanges of three messages between the participating agents. Notation such as $\langle M \rangle_C$ denotes the message M is encrypted using C's public key. Throughout, we assume the underlying encryption algorithm to be secure and the private keys of the honest agents to be uncompromised. Therefore, only agent C can decrypt $\langle M \rangle_C$ to learn M.

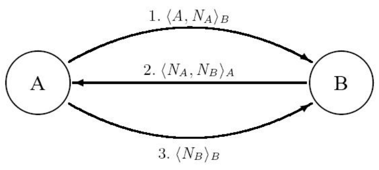

1. $\langle A, N_A \rangle_B$
2. $\langle N_A, N_B \rangle_A$
3. $\langle N_B \rangle_B$
A
B

Fig. 9. Needham-Schroeder public-key protocol

1. Alice initiates the protocol by generating a random number N_A and sending the message $\langle A, N_A \rangle_B$ to Bob (numbers such as N_A are called *nonces* in cryptographic jargon, indicating that they should be used only once by any honest agent). The first component of the message informs Bob of the identity of the initiator. The second component represents one half of the secret.
2. Bob similarly generates a nonce N_B and responds with the message $\langle N_A, N_B \rangle_A$. The presence of the nonce N_A generated in the first step, which only Bob could have decrypted, convinces Alice of the authenticity of the message. She therefore accepts the pair $\langle N_A, N_B \rangle$ as the common secret.
3. Finally, Alice responds with the message $\langle N_B \rangle_B$. By the same argument as above, Bob concludes that this message must originate with Alice, and therefore also accepts $\langle N_A, N_B \rangle$ as the common secret.

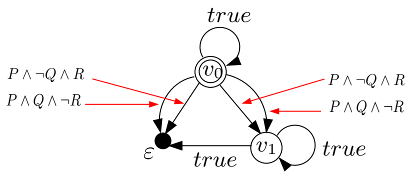
We assume all messages to be sent over an insecure medium. Attackers may intercept messages, store them, and perhaps replay them later. They may also participate in ordinary runs of the protocol, initiate runs or respond to runs initiated by honest agents, who need not be aware of their partners true identity. However, even an attacker can only decrypt messages that were encrypted with his own public key.

The purpose of the protocol is to ensure mutual authentication (of honest agents) while maintaining secrecy. In other words, whenever both A and B have successfully completed a run of the protocol, then A should believe her partner to be B if and only if B believes to talk to A. Moreover, if A successfully completes a run with B then the intruder should not have learnt A's nonce, and similarly for B. These properties can be expressed in PPTL as follows:

$$\Box((statusA = ok \wedge statusB = ok) \rightarrow ((partnerA = B) \leftrightarrow (partnerB = A)))$$
$$\Box(statusA = ok \wedge partnerA = B \rightarrow \neg knows - nonceA)$$
$$\Box(statusB = ok \wedge partnerB = A \rightarrow \neg knows - nonceB)$$

We focus on the first formula. To present it in a standard way, P, Q and R are employed to denote $statusA{=}ok \wedge statusB{=}ok$, $partnerA{=}B$ and $partnerB{=}A$ respectively. Thus, we have

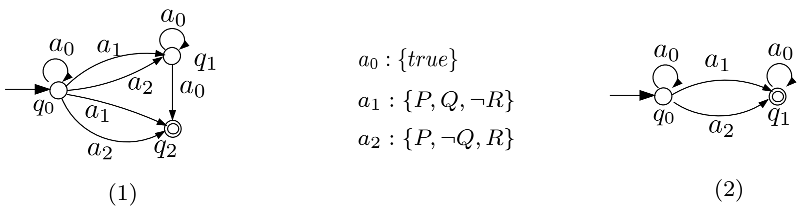
$$\Box(P \rightarrow (Q \leftrightarrow R))$$


true
v_0
$P \wedge \neg Q \wedge R$
$P \wedge Q \wedge \neg R$
$P \wedge \neg Q \wedge R$
$P \wedge Q \wedge \neg R$
ε
true
v_1
true

v_0 : *true*; $(P \wedge Q \wedge \neg R \vee P \wedge \neg Q \wedge R)$
v_1 : *true*

Fig. 10. NFG of formula $\neg\Box(P \rightarrow (Q \leftrightarrow R))$

Accordingly, $\neg\Box(P \rightarrow (Q \leftrightarrow R))$ is transformed to NFG (see Fig.10), NFA and then DFA subsequently (the NFA and DFA are depicted in Fig.11 (1) and (2) respectively). Note that, to transform the NFG of $\neg\Box(P \rightarrow (Q \leftrightarrow R))$ by Algorithm NFG, the formula is equivalently rewritten as *true*; $(P \wedge Q \wedge \neg R \vee P \wedge \neg Q \wedge R)$. Further, the DFA can be expressed in Never Claim as shown in Fig.12



a_0 : $\{true\}$
a_1 : $\{P, Q, \neg R\}$
a_2 : $\{P, \neg Q, R\}$

Fig. 11. NFA and DFA of formula $\neg\Box(P \rightarrow (Q \leftrightarrow R))$

```
Never{/*□(P → (Q ↔ R))*/
    T0-init:
        if
        ::((P&&Q&&!R)||(P&&!Q&&R)) → goto accept-all
        ::(1)→ goto T0-init
        fi
    accept-all:
        if
        ::skip
        ::(1)→ goto accept-all
        fi
}
```

Fig. 12. Never Claim of formula $\neg\Box(P \rightarrow (Q \leftrightarrow R))$

Providing the PROMELA specification of the protocol and the Never Claim of $\neg\Box(P \rightarrow (Q \leftrightarrow R))$, SPIN declares the property violated and outputs a run that contains the attack. The run is visualized as a message sequence of chart, shown in Fig.13. Alice initiates a protocol run with Intruder who in turn (but masquerading as A) starts a run with Bob, using the nonce received in the first message. Bob replies with a message of type 2 that contains both A's and B's nonces, encrypted for A. Although agent I cannot decrypt that message itself, it forwards it to A. Unsuspecting, Alice finds her nonce, returns the second nonce to her partner I, and declares success. This time, agent I can decrypt the message, extracts B's nonce and sends it to B who is also satisfied. As a result, we have reached a state where A correctly believes to have completed a run with I, but B is fooled into believing to talk to A.

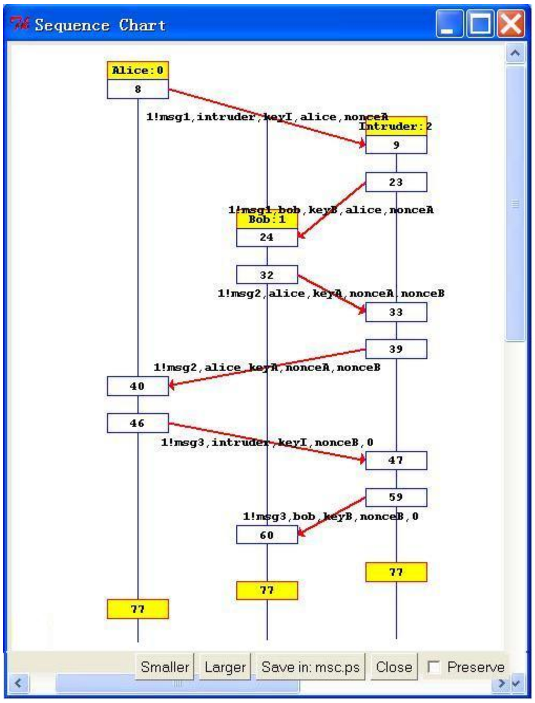

Sequence Chart
Alice:0
8
1!msg1,intruder,keyI,alice,nonceA
Intruder:2
9
23
1!msg1,bob,keyB,alice,nonceA
Bob:1
24
32
1!msg2,alice,keyA,nonceA,nonceB
33
39
1!msg2,alice,keyA,nonceA,nonceB
40
46
1!msg3,intruder,keyI,nonceB,0
47
59
1!msg3,bob,keyB,nonceB,0
60
77
77
77
Smaller
Larger
Save in: msc.ps
Close
Preserve

Fig. 13. Checking result of Needham-Schroeder protocol

7 Conclusions

In this paper, we have presented a model checking approach for PPTL with finite models. This enables us to verify properties of concurrent systems with PPTL by means of SPIN. To support our approach, we have developed a tool, a translator from PPTL formulas to Never Claim in C++, in which all of the algorithms presented in the paper have been implemented. The tool works well with SPIN. However, we are only concerned with finite models in this paper. In the near future, we will further investigate both finite and infinite models with our approach. Furthermore, we are motivated to develop a practical verification environment for the verification of Web services and hardware systems with a set of supporting tools based on the model checker for PPTL.

References

1. Kripke, S.A.: Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9, 67–96 (1963)
2. Rosner, R., Pnueli, A.: A choppy logic. In: LICS, pp. 306–314. IEEE Computer Society Press, Los Alamitos (1986)
3. Moszkowski, B.: Reasoning about digital circuits. Ph.D Thesis, Department of Computer Science, Stanford University. TRSTAN-CS-83-970 (1983)
4. Moszkowski, B.: An Automata-Theoretic Completeness Proof for Interval Temporal Logic. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 223–234. Springer, Heidelberg (2000)
5. Zhou, C., Hoare, C.A.R., Ravn, A.P.: A calculus of duration. Information Processing Letters 40(5), 269–275 (1991)
6. Duan, Z.: An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming. PhD thesis, University of Newcastle Upon Tyne (May 1996)
7. Duan, Z.: Temporal Logic and Temporal Logic Programming. Science Press, China (2006)
8. Duan, Z., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic. Technical Report No.1, Institute of computing Theory and Technology, Xidian University, Xi'an P.R.China (2005), `http://www.paper.edu.cn/process/download.jsp?file=200611-427`
9. Duan, Z., Tian, C.: Decidability of Propositional Projection Temporal Logic with Infinite Models. In: TAMC 2007. LNCS, vol. 4484, pp. 521–532. Springer, Heidelberg (2007)
10. Duan, Z., Yang, X., Koutny, M.: Semantics of Framed Temporal Logic Programs. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 256–270. Springer, Heidelberg (2005)
11. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems. Springer, Heidelberg (1992)
12. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages and Computation, 2nd edn. Addison-Wesley, Reading (2001)
13. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS 1986, pp. 332–344. IEEE CS Press, Los Alamitos (1986)
14. Dutertre, B.: Complete proof systems for first order interval temporal logic. In: Proc. 10th LICS, pp. 36–43. IEEE Computer Soc. Press, Los Alamitos (1995)
15. Holzmann, G.J.: The Model Checker Spin. IEEE Trans. on Software Engineering 23(5), 279–295 (1997)
16. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
17. Lowe, G.: Breaking and fixing the Needham-Schroeder public key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
18. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM 21(12), 993–999 (1978)
19. Merz, S.: Model Checking: A Tutorial Overview. In: Cassez, F., Jard, C., Rozoy, B., Dermot, M. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 3–38. Springer, Heidelberg (2001)
20. Wolper, P.L.: Temporal logic can be more expressive. Information and Control 56, 72–99 (1983)