Math. Struct. in Comp. Science (2009), *vol.* 19, *pp.* 73–100. © 2009 Cambridge University Press
doi:10.1017/S096012950800738X Printed in the United Kingdom

Complexity of propositional projection temporal logic with star[†]

CONG TIAN and ZHENHUA DUAN[‡]

Institute of Computing Theory and Technology, Xidian University, Xi'an, 710071, P.R.China

Received 19 October 2007

This paper investigates the complexity of Propositional Projection Temporal Logic with Star (PPTL*). To this end, Propositional Projection Temporal Logic (PPTL) is first extended to include *projection star*. Then, by reducing the emptiness problem of star-free expressions to the problem of the satisfiability of PPTL* formulas, the lower bound of the complexity for the satisfiability of PPTL* formulas is proved to be non-elementary. Then, to prove the decidability of PPTL*, the normal form, normal form graph (NFG) and labelled normal form graph (LNFG) for PPTL* are defined. Also, algorithms for transforming a formula to its normal form and LNFG are presented. Finally, a decision algorithm for checking the satisfiability of PPTL* formulas is formalised using LNFGs.

1. Introduction

A Temporal Logic (TL), such as Linear TL (LTL), Computational Tree Logic (CTL), TL of Actions (TLA) and Interval TL (ITL) (Moszkowski 1983) is a useful formalism for specification (Liu and Wang 2007) and verification (Liu and Chen 2008) of concurrent systems. In the past two decades, a number of axiom systems have been proposed within the ITL community for verifying the properties of concurrent systems (Rosner and Pnueli 1986; Zhou *et al.* 1991; Bowman and Thompson 2003; Dutertre 1995; Wang and Xu 2004). However, verifications based on axiom systems require considerable expertise from a human being to guide the proof process. In contrast, model checking is an automatic approach based on model theory. In particular, model checking is closely related to the satisfiability of temporal logics. Therefore, several researchers have concentrated on the decision problems of Propositional ITL (PITL): in particular, Halpern and Moszkowski proved the decidability of Quantifier Propositional ITL (QPITL) over finite time (Moszkowski 1983); Kono presented a tableaux-based decision procedure for QPITL with projection (Kono 1993); Bowman and Thompson gave the first tableaux-based decision procedure for quantifier-free propositional ITL (PITL) with projection over finite intervals (Bowman and Thompson 2003). To the best of our knowledge, all the above decision procedures for ITL are confined to finite intervals. However, in practice, many reactive systems are designed not to terminate. Accordingly, a decision procedure for PITL with infinite models is given in Duan *et al.* (2008b).

[†] This research is supported by the NSFC Grant No. 60433010, and Defence Pre-Research Project of China, No. 51315050105, and NSFC Grant No. 60873018 jointly sponsored by Microsoft Asia Research Academy.
[‡] Corresponding author.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Projection Temporal Logic (PTL) (see Duan (2006), Duan *et al.* (1994), Duan and Koutny (2004) and Duan *et al.* (2008a)) is an extension of ITL. It extends ITL to include infinite models and a new projection construct, $(P_1, \ldots, P_m)\ prj\ Q$. However, in the propositional case (notice that in the first-order case, it is a derived formula), the new construct needs to be extended further to *projection star*,

$$(P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q,$$

so that it can subsume *chop*, *chop star* and the original projection (*proj*) defined in Moszkowski (1983). This extension makes the underlying logic more powerful (Tian and Duan 2008), and without loss of decidability.

Plenty of logic laws have been formalised and proved within PTL (Duan 2006; Duan *et al.* 1994), and a decision procedure for checking the satisfiability of Propositional Projection Temporal Logic (PPTL) formulas with infinite models has been given (Duan *et al.* 2008b). Nevertheless, in order to verify non-terminating systems based on PPTL* using a model checking approach, the decidability and complexity of the logic needs to be investigated also. To this end, in this paper we begin by proving that the lower bound for the complexity of the satisfiability of PPTL* formulas is non-elementary by reducing the emptiness problem of star-free expressions to the problem of the satisfiability of PPTL* formulas. We then define the normal form (NF), normal form graph (NFG) and labelled normal form graph (LNFG) for PPTL* formulas, and present algorithms transforming a formula to its normal form and LNFG. We also formalise a decision procedure for checking the satisfiability of PPTL* formulas. The approach we take is, for a given formula P, to first construct the LNFG of P, and then check whether there exist paths in the LNFG of P: if there are, P is satisfiable; otherwise P is unsatisfiable.

There are two main contributions of this paper:

1. We first extend PPTL to PPTL* by introducing the *projection star* operator. We then generalise the decision procedure for checking the satisfiability of PPTL formulas given in Duan *et al.* (2008b) to handle formulas of PPTL*. This extension is non-trivial since the extended logic can subsume *chop*, *chop star* and *slice* defined in Barringer *et al.* (1984), and the two versions of projection operators defined in Moszkowski (1983) and Bowman and Thompson (2003).
2. We prove that the complexity of PPTL* is non-elementary. In fact, our conclusion also holds for PITL, PITL with *chop star* and PPTL. The expressiveness of PPTL* is more powerful, but the complexity of the logic is the same as PITL and PPTL without star. To the best of our knowledge, this is the first systematic proof of the complexity of interval-based propositional temporal logics.

The paper is organised as follows. We discuss some related work in Section 2 to set our work in context. In Section 3 we give a brief presentation of the syntax and semantics together with some logic laws of PPTL, and then show how PPTL can be extended to PPTL*. In Section 4, we prove that the lower bound for the complexity of the satisfiability of PPTL* formulas is non-elementary. In Section 5, we define the normal form for PPTL* formulas and present algorithms for transforming a PPTL* formula into its normal form. In Section 6, we define the NFGs and LNFGs of PPTL* formulas and propose an algorithm for constructing LNFGs. We then use the LNFGs to give a decision

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

algorithm for checking the satisfiability of PPTL* formulas. Finally, we draw conclusions in Section 7.

2. Related work

The work most closely related to our research on the decision procedure is Bowman and Thompson (2003). They gave a decision procedure for PITL with projection. However, this approach can only be applied to finite models, compared to which the method presented in this paper has a number of advantages. For instance, it can be applied to check the satisfiability of PITL formulas with both finite and infinite models. Furthermore, our approach is based on LNFGs, in contrast with Bowman's approach, which 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. Moreover, our underlying logic, PPTL*, can subsume PITL with a full version including *next*, *chop*, *chop star* and *projection* operators. In addition, Bowman and Thompson did not discuss the complexity of their algorithms, while we show that the complexity of PPTL* is non-elementary – we also extend this result to prove that the complexity of PITL with *chop star* and *projection* operators is non-elementary also.

The literature contains a number of works related to the complexity of Temporal Logic with *chop*: we will mention some of these here, but the list is not meant to be exhaustive. Harel *et al.* (1982) used direct coding of Turing machine computations to prove that the validity problem for local process logic (PL) with the *chop* operator is decidable but not elementary. Later, Chandra *et al.* (1985) stated, but without giving a proof, that temporal logic (TL) with the *chop* operator can also be proved in a similar way to be non-elementary. Moszkowski (2004) reported that D. Kozen (private communication) had proved that the lower bound of the satisfiability of PITL is non-elementary by reducing the emptiness problem of star-free expressions to the satisfiability of PITL, but no detailed proof was given; also the ϵ expression in the star-free expressions is missing. By contrast, in this paper we give a detailed systematic proof of the complexity of PPTL*.

3. Propositional projection temporal logic with star

Our underlying logic is Propositional Projection Temporal Logic with Star, which extends Proposition Projection Temporal Logic (Duan 2006; Duan *et al.* 1994) by including *projection star*, and is also an extension of Propositional Interval Temporal Logic (Moszkowski 1983).

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

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

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

Following the definition of Kripke's structure (Kripke 1963), we define a state s over *Prop* to be a mapping from *Prop* to $B = \{true, false\}$, $s : Prop \longrightarrow B$. We will use $s[p]$ to denote the valuation of p at state s. An interval σ is a non-empty sequence of states, which

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

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 provide 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 $=, <$ and $\leqslant$ to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Moreover, we define $\preceq$ as $\leqslant -\{(\omega, \omega)\}$. To simplify the definitions, we will denote σ by $< s_0, \ldots, s_{|\sigma|} >$, where $s_{|\sigma|}$ is undefined if σ is infinite. With this notation, $\sigma_{(i..j)}$ $(0 \leqslant i \preceq j \leqslant |\sigma|)$ denotes the sub-interval $< s_i, \ldots, s_j >$ and $\sigma^{(k)}$ $(0 \leqslant k \preceq |\sigma|)$ denotes $< s_k, \ldots, s_{|\sigma|} >$. Also, the concatenation $(\cdot)$ of two intervals σ and σ' is defined as follows:

$$\sigma \cdot \sigma' = \begin{cases} \sigma & \text{if } |\sigma| = \omega \\ < s_0, \ldots, s_i, s_{i+1}, \ldots > & \text{if } \sigma =< s_0, \ldots, s_i >,\ \sigma' =< s_{i+1}, \ldots > \quad \text{and } i \in N_0. \end{cases}$$

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

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

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

$$\begin{aligned} \sigma \downarrow () &= \varepsilon \\ \sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) &= \sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) \text{ if } r_h \text{ is not } \omega \\ \sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) &= \sigma \downarrow (r_1, \ldots, r_{h-1}) \text{ if } r_h \text{ is } \omega. \end{aligned}$$

This is convenient for defining 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.

An interpretation is a tuple $\mathscr{I} = (\sigma, k, j)$, where σ is an interval, k is an integer and j is an integer or ω such that $k \preceq j \leqslant |\sigma|$. We use the notation $(\sigma, k, j) \models P$ to denote the fact that formula P is interpreted and satisfied over the subinterval $< s_k, \ldots, s_j >$ of σ with the current state being s_k.

The satisfaction relation $(\models)$ is defined inductively as follows:

(I-prop) $\mathscr{I} \models p$ if and only if $s_k[p] = true$, for any given proposition p

(I-not) $\mathscr{I} \models \neg P$ if and only if $\mathscr{I} \not\models P$

(I-or) $\mathscr{I} \models P \vee Q$ if and only if $\mathscr{I} \models P$ or $\mathscr{I} \models Q$

(I-next) $\mathscr{I} \models \bigcirc P$ if and only if $k < j$ and $(\sigma, k+1, j) \models P$

(I-prj) $\mathscr{I} \models (P_1, \ldots, P_m)\ prj\ Q$ if there exist integers $k = r_0 \leqslant r_1 \leqslant \ldots \leqslant r_m \leqslant j$ such that $(\sigma, r_0, r_1) \models P_1$, $(\sigma, r_{l-1}, r_l) \models P_l, 1 < l \leqslant m$, and $(\sigma', 0, |\sigma'|) \models Q$ for one of the following σ' :

(a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_m+1 \ldots j)}$; or

(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_h)$ for some $0 \leqslant h \leqslant m$.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

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

Table 1. *Precedence rules*

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

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

The abbreviations *true*, $false$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, $true \stackrel{\text{def}}{=} P \vee \neg P$ and $false \stackrel{\text{def}}{=} P \wedge \neg P$ for any formula P. We also have the following derived formulas:

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

where:

— $\odot$ (weak next), $\Box$ (always), $\diamond$ (sometimes) and ; (chop) are derived temporal operators;

— *empty* denotes an interval with zero length and *more* means the current state is not the final one over an interval;

— $halt(P)$ is true over an interval if and only if P is true in the final state, $fin(P)$ is true when P is true in the final state and $keep(P)$ is true if P is true in every state ignoring the final one.

Two formulas P and Q are equivalent, denoted $P \equiv Q$, if $\models \Box(P \leftrightarrow Q)$. Let w be a state formula.

Some useful logic laws are given below:

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

$$
\begin{aligned}
&(L_1) && \Box(P \wedge Q) \equiv \Box P \wedge \Box Q\\
&(L_2) && \Diamond(P \vee Q) \equiv \Diamond P \vee \Diamond Q\\
&(L_3) && \bigcirc(P \vee Q) \equiv \bigcirc P \vee \bigcirc Q\\
&(L_4) && \bigcirc(P \wedge Q) \equiv \bigcirc P \wedge \bigcirc Q\\
&(L_5) && R;(P \vee Q) \equiv (R;P) \vee (R;Q)\\
&(L_6) && (P \vee Q);R \equiv (P;R) \vee (Q;R)\\
&(L_7) && \Diamond P \equiv P \vee \bigcirc \Diamond P\\
&(L_8) && \Box P \equiv P \wedge \odot \Box P\\
&(L_9) && more \wedge \neg \bigcirc P \equiv more \wedge \bigcirc \neg P\\
&(L_{10}) && \neg \odot P \equiv \bigcirc \neg P\\
&(L_{11}) && \bigcirc P;Q \equiv \bigcirc(P\ ;Q)\\
&(L_{12}) && w \wedge (P\ ;Q) \equiv (w \wedge P)\ ;Q\\
&(L_{13}) && Q\ prj\ empty \equiv Q\\
&(L_{14}) && empty\ prj\ Q \equiv Q\\
&(L_{15}) && empty;P \equiv P\\
&(L_{16}) && (P_1,\ldots,P_m)\ prj\ empty \equiv P_1;\ldots;P_m\\
&(L_{17}) && (P_1,\ldots,P_t,w \wedge empty,P_{t+1},\ldots,P_m)\ prj\ Q\\
& && \quad \equiv (P_1,\ldots,P_t,w \wedge P_{t+1},\ldots,P_m)\ prj\ Q\\
&(L_{18}) && (P_1,\ldots,(P_i \vee P_i'),\ldots,P_m)\ prj\ Q \equiv ((P_1,\ldots,P_i,\ldots,P_m)\ prj\ Q)\\
& && \quad \vee((P_1,\ldots,P_i',\ldots,P_m)\ prj\ Q)\\
&(L_{19}) && (P_1,\ldots,P_m)\ prj\ (P \vee Q) \equiv ((P_1,\ldots,P_m)\ prj\ P)((P_1,\ldots,P_m)\ prj\ Q)\\
&(L_{20}) && (P_1,\ldots,P_m)\ prj \bigcirc Q \equiv (P_1 \wedge more;(P_2,\ldots,P_m)\ prj\ Q)\\
& && \quad \vee(P_1 \wedge empty;(P_2,\ldots,P_m)\ prj \bigcirc Q)\\
&(L_{21}) && (\bigcirc P_1,\ldots,P_m)\ prj \bigcirc Q \equiv \bigcirc(P_1\ ;(P_2,\ldots,P_m)\ prj\ Q)\\
&(L_{22}) && (w \wedge P_1,\ldots,P_m)\ prj\ Q \equiv w \wedge ((P_1,\ldots,P_m)\ prj\ Q)\\
&(L_{23}) && (P_1,\ldots,P_m)\ prj\ (w \wedge Q) \equiv w \wedge ((P_1,\ldots,P_m)\ prj\ Q).
\end{aligned}
$$

The proofs of these logic laws can be found in Duan (2006).

Also, with the projection construct $(P_1,\ldots,P_m)\ prj\ Q$ in PPTL, we have the following derived formulas:

$$
\begin{cases}
(P_1,\ldots,P_{i-1},(P_i,\ldots,P_s)^{(0)},P_{s+1},\ldots,P_m)\ prj\ Q \stackrel{\text{def}}{=} (P_1,\ldots,P_{i-1},P_{s+1},P_m)\ prj\ Q\\
(P_1,\ldots,P_{i-1},(P_i,\ldots,P_s)^{(1)},P_{s+1},\ldots,P_m)\ prj\ Q \stackrel{\text{def}}{=} (P_1,\ldots,P_{i-1},P_i,\ldots,P_s,\\
\qquad\qquad P_{s+1},\ldots,P_m)\ prj\ Q\\
(P_1,\ldots,P_{i-1},(P_i,\ldots,P_s)^{(n)},P_{s+1},\ldots,P_m)\ prj\ Q \stackrel{\text{def}}{=} (P_1,\ldots,P_{i-1},(P_i,\ldots,P_s),\\
\qquad\qquad (P_i,\ldots,P_s)^{(n-1)},P_{s+1},\ldots,P_m)\ prj\ Q,\\
\qquad\qquad 1 \leqslant i \leqslant s \leqslant m, n \geqslant 1, n \in N_0
\end{cases}
$$

Using the above derivation, we can introduce *projection star* to PPTL as follows:

$$
(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q \stackrel{\text{def}}{=}
\begin{cases}
(P_1,\ldots,P_{i-1},(P_i,\ldots,P_s)^{(n)},\ldots,P_m)\ prj\ Q,\\
\qquad\qquad \text{if } 1 \leqslant i \leqslant s \leqslant m, n \in N_0\\
(P_1,\ldots,P_{i-1},(P_i,\ldots,P_m)^{(\omega)})\ prj\ Q,\\
\qquad\qquad \text{if } 1 \leqslant i \leqslant m.
\end{cases}
$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

The logic extended in this way is called Propositional Projection Temporal Logic with Star (PPTL*). With this extension, the previous projection is merely a special case of the projection star. So rather than introduce a new operator, we just use the old one prj. The semantics of *projection star* is given as follows:

(I-prj$^{\circledast}$) $\mathscr{I} \models (P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q$ if and only if either:

$1 \leqslant i \leqslant s \leqslant m$ and

$$\exists n \in N_0, \mathscr{I} \models (P_1,\ldots,(P_i,\ldots,P_s)^{(n)},\ldots,P_m)\ prj\ Q$$

or:

$s = m$ and there exist infinitely many integers

$$k = r_0 \leqslant r_1 \leqslant \cdots \leqslant r_k \preceq \omega, \lim_{k\to\infty} r_k = \omega$$

such that

$$(\sigma, r_{l-1}, r_l) \models P_l, 0 < l < i$$
$$(\sigma, r_{l-1}, r_l) \models P_t, l \geqslant i, t = i + ((l-i)\ mod\ (s-i+1))$$
$$(\sigma', 0, |\sigma'|) \models Q,$$

where $\sigma' = \sigma \downarrow (r_0,\ldots,r_h)$, for some $r_h \in N_\omega$.

We also have the following derived formulas involving *projection star*:

$$(P_1,\ldots,P_{i-1},(P_i,\ldots,P_s)^{\oplus},P_{s+1}\ldots,P_m)\ prj\ Q \stackrel{\text{def}}{=} (P_1,\ldots,P_{i-1},P_i,\ldots P_s,$$
$$(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)\ prj\ Q$$
$$P^* \stackrel{\text{def}}{=} (P^{\circledast})\ prj\ empty$$
$$P^+ \stackrel{\text{def}}{=} (P^{\oplus})\ prj\ empty.$$

The new temporal operators $*$, $+$, $prj^{\circledast}$ and $prj^{\oplus}$ have the same precedence as ; and prj.

Moreover, it is easy to prove the following logic laws:

(L_{24}) $(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q \equiv (P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)\ prj\ Q \vee$
$(P_1,\ldots,P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q$

(L_{25}) $P^* \equiv empty \vee P;P^*$

(L_{26}) $(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ empty \equiv P_1;\ldots;(P_i;\ldots;P_s)^*;\ldots;P_m$

(L_{27}) $(P_1,\ldots,(P_i,\ldots,P_t,w \wedge empty,P_{t+1},\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q \equiv$
$(P_1,\ldots,(P_i,\ldots,P_t,w \wedge P_{t+1},\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q$

(L_{28}) $(P_1,\ldots,(P_i,\ldots,(P_t \vee P'_t),\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q \equiv$
$(P_1,\ldots,(P_i,\ldots,P_t,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q$
$\vee(P_1,\ldots,(P_i,\ldots,P'_t,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q$

(L_{29}) $(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ (P \vee Q) \equiv$
$(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ P$
$\vee(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q$

(L_{30}) $(\bigcirc P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj \bigcirc Q \equiv$
$\bigcirc(P_1\ ;(P_2,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q)$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

(L_{31}) $(w \wedge P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q \equiv$
$w \wedge ((P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q)$

(L_{32}) $(P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ (w \wedge Q) \equiv$
$w \wedge ((P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q)$

4. Lower bound of the complexity for satisfiability of PPTL*

Stockmeyer (1974) showed that the emptiness problem for star-free expressions is non-elementary. He also claimed that if this problem is efficiently reducible to a particular decision problem, the decision problem is non-elementary. In the following we will show that the emptiness problem for star-free expression can be reduced to the problem of the satisfiability of PPTL*. To this end, we first introduce the basic concepts of star-free expressions and some notation.

Definition 4.1. A star-free expression is defined by the following grammar:

$$E ::= 0 \mid 1 \mid \epsilon \mid \varnothing \mid E_1 \cup E_2 \mid E_1 \bullet E_2 \mid \sim E$$

where $\cup$, $\bullet$ and $\sim$ are read as union, concatenation and complement, respectively, and ϵ denotes an empty string.

A string is a finite or infinite sequence of symbols chosen form $\{0,1\}$. So $\Gamma = \{0,1\}^*$ denotes the set of all strings. The length of a finite string w, denoted $|w|$, is the number of symbols in w, and the length of an infinite string is ω. For two strings w and w', $w \bullet w'$ is defined as follows:

$$w \bullet w' = \begin{cases} w', & \text{if } w \text{ is } \epsilon \\ w, & \text{if } w' \text{ is } \epsilon \text{ or } |w| = \omega \\ a_0 \ldots a_i a_{i+1} \ldots, & \text{if } w = a_0 \ldots a_i, \text{ and } w' = a_{i+1} \ldots \end{cases}$$

Also, if W and W' are two sets of strings, $W \bullet W'$ is defined as follows:

$$W \bullet W' = \{w \bullet w' \mid w \in W \text{ and } w' \in W'\}.$$

In particular, $\varnothing \bullet W = \varnothing$ and $W \bullet \varnothing = \varnothing$. Accordingly, the language $L(E)$ expressed by star-free expression E is given by

$$\begin{aligned} L(\varnothing) &= \varnothing \\ L(\epsilon) &= \{\epsilon\} \\ L(0) &= \{0\} \\ L(1) &= \{1\} \\ L(E_1 \cup E_2) &= L(E_1) \cup L(E_2) \\ L(E_1 \bullet E_2) &= L(E_1) \bullet L(E_2) \\ L(\sim E) &= \Gamma \setminus L(E). \end{aligned}$$

For a string w, if $w \in L(E)$, w is called a sentence of star-free expression E.

We say E is an empty star-free expression if $L(E) = \varnothing$, and a non-empty star-free expression otherwise. Then the set $NE_{star\text{-}free}$ of all non-empty star-free expressions is

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Precedence	Operators
1	$\sim$
2	$\bullet$
3	$\cup$

Table 2. *Precedence rules*

defined as follows:

$$NE_{star\text{-}free} = \{E \mid E \text{ is a star-free expression and } L(E) \neq \varnothing\}.$$

To avoid an excessive number of parentheses, we use precedence rules a shown in Table 2, where 1 = highest and 3 = lowest.

Let SAT_l denote the set of formulas in logic L that are satisfiable. We have from Stockmeyer (1974) that

If $NE_{star\text{-}free} \leqslant_{pl} SAT_l$, *the satisfiability for logic* L *is non-elementary.*

Here, $NE_{star\text{-}free} \leqslant_{pl} SAT_l$ means that there exists an efficiently computable function $f : NE_{star\text{-}free} \to SAT_l$ such that answering '$E \in NE_{star\text{-}free}$?' if and only if answering '$f(E) \in SAT_l$?', and any expression E in $NE_{star\text{-}free}$ can be reduced to a formula $f(E)$ in SAT_l within polynomial time and linear space, with f a linear bounded function (see Stockmeyer (1974) for the details). So, if $NE_{star\text{-}free} \leqslant_{pl} SAT_{pptl^*}$ holds, the lower bound of the complexity for the satisfiability of PPTL* is non-elementary. Now we come to prove the following theorem.

Theorem 4.1. $NE_{star\text{-}free} \leqslant_{pl} SAT_{pptl^*}$.

Proof. Let F be a function mapping $E \in E_{star\text{-}free}$, the set of all star-free expressions, to $F_E(q) \in L_{pptl^*}$. Given a star-free expression E, we can construct a formula $F_E(q) \in L_{pptl^*}$ containing a single proposition q such that, for $B = \{0, 1\}$, we have $s_i : \{q\} \to B$, and

$$s_i[q] = \begin{cases} 1, & \text{if } q \text{ holds at state } s_i \\ 0, & \text{if } \neg q \text{ holds at state } s_i. \end{cases}$$

Thus, $F_E(q)$ holds under the interpretation (σ, k, j) if and only if:

1. If $k < j$, then $s_k[q]s_{k+1}[q] \ldots s_{j-1}[q] \in L(E)$.
2. If $k = j$, then $\epsilon \in L(E)$.

$F_E(q)$ is constructed inductively on the structure of E:

$$\begin{aligned} F_0(q) &\stackrel{\text{def}}{=} \neg q \wedge skip \\ F_1(q) &\stackrel{\text{def}}{=} q \wedge skip \\ F_\epsilon(q) &\stackrel{\text{def}}{=} empty \\ F_\varnothing(q) &\stackrel{\text{def}}{=} false \\ F_\Gamma(q) &\stackrel{\text{def}}{=} true. \end{aligned}$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Inductively, if E_1 and E_2 are star-free expressions, then

$$F_{E_1 \cup E_2}(q) \stackrel{\text{def}}{=} F_{E_1}(q) \vee F_{E_2}(q)$$
$$F_{E_1 \bullet E_2}(q) \stackrel{\text{def}}{=} F_{E_1}(q); F_{E_2}(q)$$
$$F_{\sim E_1}(q) \stackrel{\text{def}}{=} \neg F_{E_1}(q).$$

Now we need to prove that $E \in NE_{star\text{-}free}$ if and only if $F_E(q) \in SAT_{pptl^*}$:

($\Rightarrow$) If $E \in NE_{star\text{-}free}$, then $F_E(q) \in SAT_{pptl^*}$. The proof proceeds by induction on the structure of E:

Base case: For expression 0, the corresponding formula is $F_0(q) \stackrel{\text{def}}{=} \neg q \wedge skip$. We can construct a model $\sigma =< \{\neg p\}, \{true\} >$, so formula $F_0(q)$ is satisfiable. In a similar way, models can also be constructed for expressions 1, $\epsilon \in NE_{star\text{-}free}$ so that the corresponding formulas $F_1(q) \stackrel{\text{def}}{=} q \wedge skip$ and $F_\epsilon(q) \stackrel{\text{def}}{=} empty$ are satisfiable.

Induction step: Suppose, for $E_1, E_2 \in NE_{star\text{-}free}$, there exist models σ_1 and σ_2 that can satisfy $F_{E_1}(q)$ and $F_{E_2}(q)$, respectively. Then:

(1) If $E_1 \cup E_2 \in NE_{star\text{-}free}$, we have $L(E_1 \cup E_2) \neq \varnothing$. By Pr_3, we have $L(E_1) \neq \varnothing$ or $L(E_2) \neq \varnothing$. Furthermore, by hypothesis, $F_{E_1}(q)$ is satisfied by σ_1 or $F_{E_2}(q)$ is satisfied by σ_2. Thus, $F_{E_1}(q) \vee F_{E_2}(q)$ is satisfied by σ_1 or σ_2.

(2) If $E_1 \bullet E_2 \in NE_{star\text{-}free}$, by Pr_4 and Pr_7, we have E_1 and $E_2 \in NE_{star\text{-}free}$. Furthermore, by hypothesis, $F_{E_1}(q)$ and $F_{E_2}(q)$ are satisfied by σ_1 and σ_2, respectively. Thus, $F_{E_1}(q); F_{E_2}(q)$ is satisfied by $\sigma_1 \cdot \sigma_2$.

(3) If $\sim E_1 \in NE_{star\text{-}free}$, by Pr_5, Pr_2, we have $L(E_1) \neq \Gamma$. So $F_{E_1}(q) \not\equiv true$. Thus, $\neg F_{E_1}(q) \not\equiv false$, and can be satisfied.

($\Leftarrow$) If $F_E(q) \in SAT_{pptl^*}$, then $E \in NE_{star\text{-}free}$. The proof proceeds by induction on the structure of $F_E(q)$:

Base case: $F_0(q) \equiv \neg q \wedge skip \in SAT_{pptl^*}$ is satisfiable under the interpretation of $(\sigma, 0, 1)$ with $\sigma =< \{\neg q\}, \{true\} >$. By the generating rules of sentences, since $1 > 0$, the sentence $s_0[q] = 0 \in L(0)$, so the corresponding expression $0 \in NE_{star\text{-}free}$. A similar argument can be given for $F_1(q)$ and $F_\epsilon(q)$.

Induction step: Suppose that if $F_{E_1}(q)$ and $F_{E_2}(q)$ are satisfiable under the interpretation of $(\sigma, 0, i)$ and $(\sigma, 0, j)$ with $\sigma_1 =< s_0, \ldots, s_i >$ and $\sigma_2 =< s'_0, \ldots, s'_j >$, respectively, then there exist sentences $s_0[q] \ldots s_{i-1}[q] \in L(E_1)$ and $s'_0[q] \ldots s'_{j-1}[q] \in L(E_2)$.

(1) If $F_{E_1}(q) \vee F_{E_2}(q)$ is satisfiable under the interpretation of $(\sigma, 0, i)$ or $(\sigma, 0, j)$ with $\sigma_1 =< s_0, \ldots, s_i >$ or $\sigma_2 =< s'_0, \ldots, s'_j >$, we have $F_{E_1}(q)$ is satisfiable under the interpretation of $(\sigma, 0, i)$ with $\sigma_1 =< s_0, \ldots, s_i >$ or $F_{E_2}(q)$ is satisfiable under the interpretation of $(\sigma, 0, j)$ with $\sigma_2 =< s'_0, \ldots, s'_j >$. By hypothesis, there exists sentence $s_0[q] \ldots s_{i-1}[q] \in L(E_1)$ or $s'_0[q] \ldots s'_{j-1}[q] \in L(E_2)$. So, by Pr_3, we have $s_0[q] \ldots s_{i-1}[q]$ or $s_0[q] \ldots s_{j-1}[q] \in L(E_1 \cup E_2)$.

(2) If $F_{E_1}(q); F_{E_2}(q)$ is satisfiable under the interpretation of $(\sigma, 0, j)$ with $\sigma =< s_0, \ldots, s_j >$, there exists i, $0 \leqslant i \leqslant j$, such that $F_{E_1}(q)$ and $F_{E_2}(q)$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

are satisfiable under the interpretation of $(\sigma, 0, i)$ and (σ, i, j), respectively. Let $\sigma_1 =< s_0, \ldots, s_i >$ and $\sigma_2 =< s_i, \ldots, s_j >$. By hypothesis, there exist sentences $s_0[q] \ldots s_{i-1}[q] \in L(E_1)$ and $s_i[q] \ldots s_{j-1}[q] \in L(E_2)$. So, by Pr_4, the sentence $s_0[q] \ldots s_{i-1}[q] s_i[q] \ldots s_{j-1}[q] \in L(E_1 \bullet E_2)$.

(3) If $\neg F_{E_1}(q)$ is satisfiable, $\neg F_{E_1}(q) \not\equiv false$, so Thus, $F_{E_1}(q) \not\equiv true$. So $L(E_1) \neq \Gamma$. By Pr_2, we have $L(\sim E_1) = \Gamma - L(E_1) \neq \phi$, and thus $\sim E_1 \in NE_{star\text{-}free}$. □

Clearly, F can be computed within polynomial time and linear space, and F is linear bounded. Therefore, checking the satisfiability of PPTL* formulas is at least non-elementary.

Therefore, the lower bound for the complexity of the satisfiability of PPTL* formulas is non-elementary. In order to prove that checking the satisfiability of PPTL* is decidable, we need a decision algorithm. In any case, the most efficient decision algorithm will be non-elementary. Furthermore, the above proof tells us that the lower bounds for the complexity of the satisfiability of PPTL and PITL are also non-elementary since only the temporal operators *chop* and *next* are involved in the proof.

5. Normal form of PPTL*

In order to prove the decidability of satisfiability of PPTL* formulas, we first rewrite a formula to its normal form. The normal form is useful for constructing a graph structure that precisely characterises the models of the corresponding formula.

Let L_{pptl^*} denote the set of all PPTL* formulas. For any $Q \in L_{pptl^*}$, we have:

1. Q is called a state formula if it contains no temporal operators.
2. Q is called a terminating formula if $Q \equiv Q \wedge empty$.
3. Q is called a future formula if $Q \equiv Q \wedge more$.

In some circumstances we will call a state or terminating formula a local formula. In the following, we briefly introduce the definitions of normal form and complete normal form that were given in Duan *et al.* (2008b) since we will use the notation to handle PPTL* formulas.

Definition 5.1. Let Q_p be the set of atomic propositions appearing in PPTL* formula Q. The normal form of Q can be defined as follows:

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

where $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$, $q_{jk}, q_{ih} \in Q_p$, for any $r \in Q_p$, $\dot{r}$ denotes r or $\neg r$; Q'_i is any PPTL* formula that does not have '$\vee$' as its main operator.

According to the definition, the normal form of $p \wedge q \wedge \bigcirc(\Box p \vee q)$ must be written as $p \wedge q \wedge \bigcirc(\Box p) \vee p \wedge q \wedge \bigcirc q$ since '$\vee$' is the main operator of $\Box p \vee p$. For convenience, we will sometimes write $Q_e \wedge empty$ instead of $\bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty)$ and $\bigvee_{i=0}^{r}(Q_i \wedge \bigcirc Q'_i)$ instead of $\bigvee_{i=0}^{n}(Q_{ci} \wedge \bigcirc Q'_i)$. Thus we have

$$Q \equiv (Q_e \wedge empty) \vee \bigvee_{i=0}^{r} (Q_i \wedge \bigcirc Q'_i)$$

where Q_e and Q_i are state formulas.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

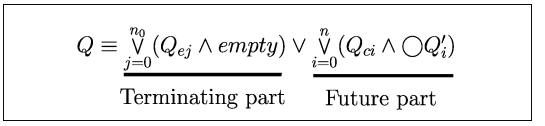


Fig. 1. Normal form of PPTL* formulas.

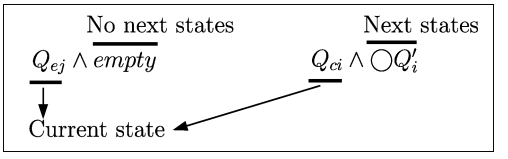


Fig. 2. Basic components of normal form.

Definition 5.2. If $\bigvee_i Q_i \equiv true$ and $\bigvee_{i \neq j}(Q_i \wedge Q_j) \equiv false$ in a normal form, the normal form is said to be a complete normal form.

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

$$P_e \wedge empty \vee \bigvee_{i=0}^{r}(m_i \wedge \bigcirc M'_i),$$

we have

$$\neg P \equiv \neg P_e \wedge empty \vee \bigvee_{i=0}^{r}(m_i \wedge \bigcirc \neg M'_i).$$

The normal form of a formula divides the formula into two parts: the terminating part and the future part, as shown in Figure 1. The terminating part is a terminating formula, since $\bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty) \equiv (\bigvee_{j=0}^{n_0}(Q_{ej} \wedge empty)) \wedge empty$, while the future part is a future formula because of $\bigvee_{i=0}^{n}(Q_{ci} \wedge \bigcirc Q'_i) \equiv \bigvee_{i=0}^{n}(Q_{ci} \wedge \bigcirc Q'_i) \wedge more$. In addition, $Q_{ej} \wedge empty$ or $Q_{ci} \wedge \bigcirc Q'_i$ in a normal form is said to be a basic component of the normal form. As shown in Figure 2, for each basic component in the terminating part, $Q_{ej} \equiv \bigwedge_{k=1}^{m_0} \dot{q}_{jk}$ means that each proposition or its negation in $\{\dot{q_{j1}}, \ldots, \dot{q_{jk}}\}$ holds in the current state, while *empty* means that the current interval ending point has been reached; similarly, for each basic component in the future part, $Q_{ci} \equiv \bigwedge_{h=1}^{m} \dot{q}_{ih}$ indicates that each proposition or its negation in $\{\dot{q_{i1}}, \ldots, \dot{q_{ih}}\}$ is true in the current state and $\bigcirc Q'_i$ means that Q'_i holds over the interval starting from the next state.

To further the proof that any PPTL* formula can be rewritten to its normal form, we need the following fixed-point theorem due to Knaster and Tarski (Tarski 1955).

Theorem 5.1. Every monotonic function F over a complete lattice has a complete lattice of fixed points (and hence unique greatest and least fixed points). The least fixed point of monotonic function F on the complete lattice $< A, \sqsubseteq >$ can be computed by $\sqcup_i F^i(\bot)$, that is the lowest upper bound of the series $\bot$, $F(\bot)$, $F(F(\bot))$, ..., while the greatest fixed

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

point can be computed by $\sqcap_i F^i(\top)$, that is the greatest lower bound of the series $\top$, $F(\top)$, $F(F(\top))$, ...

We define the binary relationship $\sqsubseteq$ over L_{pptl^*} as follows:

$$\text{For any formulas } P_1, P_2 \in L_{pptl^*}, P_1 \sqsubseteq P_2 \text{ if and only if } P_1 \to P_2\,.$$

It is easy to prove that the binary relationship $\sqsubseteq$ on set L_{pptl^*} is a partial order. Thus, the pair $< L_{pptl^*}, \sqsubseteq >$ is a partially ordered set, or *poset* for short. Furthermore, $< L_{pptl^*}, \sqsubseteq >$ is a complete lattice since for each set $A \subseteq L_{pptl^*}$, the least upper bound $\sqcup A = \bigvee_i A_i$, and the greatest lower bound $\sqcap A = \bigwedge_i A_i$, $A_i \in A$. In particular, if A is L_{pptl^*}, we have $\sqcup A = true$ and $\sqcap A = false$. Thus, by Theorem 5.1, a monotonic function F over $< L_{pptl^*}, \sqsubseteq >$ has a complete lattice of fixed points (and hence a unique greatest fixed point, F_{gfp}, and a unique least fixed point, F_{lfp}). Now we prove the following lemmas regarding $P;Q$ and P^*.

Lemma 5.1. Let $P;Q$ be a PPTL* formula. If P and Q have been rewritten to their normal form, $P;Q$ can be rewritten to its normal form.

Proof. The lemma can be proved in a similar way to the corresponding proof in Duan *et al.* (2008b). □

Lemma 5.2. For any PPTL* formula P, if $P \equiv P_e \wedge empty \vee \bigvee_{i=0}^{r} P_i \wedge \bigcirc P_i'$, then P^* can be rewritten to its normal form $P^* \equiv empty \vee \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*)$.

Proof.

$$\begin{aligned}
P^* &\equiv empty \vee (P; P^*) && (L_{25})\\
&\equiv empty \vee ((P_e \wedge empty \vee \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc P_i'); P^*) && (\text{Precondition})\\
&\equiv empty \vee (P_e \wedge empty; P^*) \vee \textstyle\bigvee_{i=0}^{r} (P_i \wedge \bigcirc P_i'; P^*) && (L_6)\\
&\equiv empty \vee P_e \wedge P^* \vee \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*)\,. && (L_{11}, L_{12}, L_{15})
\end{aligned}$$

We define function F over $< L_{pptl^*}, \sqsubseteq >$ as follows:

$$F(X) = empty \vee P_e \wedge X \vee \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*)\,.$$

It is not hard to prove that F is monotonic. Thus, by Theorem 5.1, F has a unique least fixed point F_{lfp} and unique greatest fixed point F_{gfp}. That is,

$$F_{lfp} = F(false) \vee F(F(false)) \vee \ldots = empty \vee \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*)\,.$$

Thus, we claim $F_{lfp} \equiv P^*$. Let σ be any interval. We have:

$$\begin{aligned}
\sigma \models P^* &\Leftrightarrow \sigma \models empty \vee P_e \wedge P^* \vee \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*)\\
&\Leftrightarrow \sigma \models P_e \wedge P^* \text{ or } \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty\\
&\Leftrightarrow \exists n \in N_\omega, \sigma \models P_e \wedge P^n \text{ or } \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty.
\end{aligned}$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

We will now prove the following result by induction on n:

$$\sigma \models P_e \wedge P^n \text{ or } \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \Leftrightarrow \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty\,.$$

If $n = 0$,

$$\begin{aligned}\sigma \models P_e \wedge P^0 \text{ or } \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ \Leftrightarrow \sigma \models P_e \wedge empty \text{ or } \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ \Leftrightarrow \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty\,.\end{aligned}$$

Assume for $n = i$,

$$\sigma \models P_e \wedge P^i \text{ or } \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \Leftrightarrow \sigma \models \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty\,.$$

When $n = i + 1$,

$$\begin{aligned}&\sigma \models P_e \wedge P^{i+1} \\ &\text{or} \\ &\sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ &\qquad \Leftrightarrow \sigma \models P_e \wedge P; P^i \\ &\qquad\quad \text{or} \\ &\qquad\quad \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ &\qquad \Leftrightarrow \sigma \models P_e \wedge (P_e \wedge empty \vee \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc P_i'); P^i \\ &\qquad\quad \text{or} \qquad\qquad \text{(Precondition)} \\ &\qquad\quad \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ &\qquad \Leftrightarrow \sigma \models P_e \wedge P^i \vee \textstyle\bigvee_{i=0}^{r} P_e \wedge P_i \wedge \bigcirc(P_i'; P^i) \\ &\qquad\quad \text{or} \qquad\qquad (L_6, L_{11}, L_{12}, L_{16}) \\ &\qquad\quad \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ &\qquad \Leftrightarrow \sigma \models P_e \wedge P^i \\ &\qquad\quad \text{or} \\ &\qquad\quad \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ &\qquad \Leftrightarrow \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty\,. \qquad \text{(By hypothesis)}\end{aligned}$$

So, for any n,

$$\begin{aligned}\sigma \models P_e \wedge P^n \text{ or } \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty \\ \Leftrightarrow \sigma \models \textstyle\bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*) \vee empty\,.\end{aligned}$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Thus, $P^* \equiv F_{lfp}$. So, P^* can be rewritten to its normal form

$$P^* \equiv empty \vee \bigvee_{i=0}^{r} P_i \wedge \bigcirc(P_i'; P^*).$$

□

Lemmas 5.3 and 5.4, respectively, are concerned with rewriting formulas in prj and $prj^{\circledast}$ constructs into their normal forms.

Lemma 5.3. If PPTL* formulas $P_1, \ldots, P_m$ and Q can be rewritten into their normal forms, then $(P_1, \ldots, P_m)\ prj\ Q$ can be rewritten into its normal form.

Proof. The lemma can be proved in a similar way to the corresponding proof in Duan *et al.* (2008b). □

Lemma 5.4. If PPTL* formulas P_k, $1 \leqslant k \leqslant m$, and Q can be rewritten into their normal forms, then $(P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q$ can be rewritten into its normal form.

Proof. According to Lemma 5.3, $(P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q$ can be written as

$$((P_e \wedge empty \vee \bigvee_{l=0}^{r} P_l \wedge \bigcirc P_l'), (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ (R_e \wedge empty \vee \bigvee_{l=0}^{r} R_l \wedge \bigcirc R_l').$$

Thus, by L_{24} we have,

$$\begin{gathered}((P_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} P_l \wedge \bigcirc P_l'), (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj \\ (R_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} R_l \wedge \bigcirc R_l') \\ \equiv \\ ((P_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} P_l \wedge \bigcirc P_l'), P_{s+1}, \ldots, P_m)\ prj\ (R_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} R_l \wedge \bigcirc R_l') \vee \\ ((P_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} P_l \wedge \bigcirc P_l'), P_i, \ldots, P_s, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ (R_e \wedge empty \vee \\ \textstyle\bigvee_{l=0}^{r} R_l \wedge \bigcirc R_l')\end{gathered}$$

By Lemma 5.3,

$$((P_e \wedge empty \vee \bigvee_{l=0}^{r} P_l \wedge \bigcirc P_l'), P_{s+1}, \ldots, P_m)\ prj\ (R_e \wedge empty \vee \bigvee_{l=0}^{r} R_l \wedge R_l')$$

can be rewritten into its normal form. And

$$\begin{gathered}((P_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} P_l \wedge \bigcirc P_l'), P_i, \ldots, P_s, (P_i, \ldots, P_s^{\circledast}, \ldots, P_m)\ prj \\ (R_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} R_l \wedge R_l')\end{gathered}$$

can be further rewritten using L_{11} and L_{26}–L_{32} as

$$\begin{gathered}((Q_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} Q_l \wedge \bigcirc Q_l'), (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ (T_e \wedge empty \vee \textstyle\bigvee_{l=0}^{r} T_l \wedge \bigcirc T_l') \\ \equiv \\ Q_e \wedge T_e \wedge ((P_i; \ldots; P_s)^*; \ldots; P_m) \\ \vee \textstyle\bigvee_{l=0}^{r} Q_l \wedge T_e \wedge \bigcirc(Q_l'; (P_i; \ldots; P_s)^*; \ldots; P_m) \\ \vee \textstyle\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' \\ \vee \textstyle\bigvee_{l=0}^{r} Q_l \wedge T_l \wedge \bigcirc(Q_l; ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj\ T_l')\end{gathered}$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

By Lemmas 5.1 and 5.2, $Q_e \wedge T_e \wedge ((P_i; \ldots; P_s)^*; \ldots; P_m)$ can be rewritten into its normal form. Also, $\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l'$ can be rewritten using L_{24} as,

$$\begin{gathered}\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' \\ \equiv \\ \bigvee_{l=0}^{r} Q_e \wedge T_l \wedge (P_{s+1}, \ldots, P_m)\ prj \bigcirc T_l' \\ \vee \bigvee_{l=0}^{r} Q_e \wedge T_l \wedge (P_i, \ldots, P_s, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' .\end{gathered}$$

By Lemma 5.3,

$$\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge (P_{s+1}, \ldots, P_m)\ prj \bigcirc T_l'$$

can be rewritten into its normal form. And

$$\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge (P_i, \ldots, P_s, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l'$$

can be further rewritten as

$$\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((Q_e \wedge empty \vee Q_l \wedge \bigcirc Q_l'), (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' .$$

Thus, using L_{27}, L_{28}, L_{30} and L_{31}, we have

$$\begin{gathered}\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((Q_e \wedge empty \vee Q_l \wedge \bigcirc Q_l'), (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' \\ \equiv \\ \bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' \\ \vee \bigvee_{l=0}^{r} Q_e \wedge T_l \wedge Q_l \wedge \bigcirc ((Q_l'; (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l')\end{gathered}$$

Thus, we have

$$\begin{gathered}\bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' \\ \equiv \\ S \vee \bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l'\end{gathered}$$

where S is in normal form. We define function F over $< L_{pptl^*}, \sqsubseteq >$ as follows:

$$F(X) = X \vee S .$$

It is not hard to prove that F is monotonic. Thus, by Theorem 5.1, F has a unique least fixed point F_{lfp} and unique greatest fixed point F_{gfp}. That is,

$$F_{lfp} = F(false) \vee F(F(false)) \vee \ldots = S .$$

Using similar reasoning to that in the proof of Lemma 5.3, it can be proved that

$$F_{lfp} \equiv \bigvee_{l=0}^{r} Q_e \wedge T_l \wedge ((P_i, \ldots, P_s)^{\circledast}, \ldots, P_m) prj \bigcirc T_l' .$$

Thus,

$$(P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q$$

can be rewritten into its normal form. □

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Lemma 5.5 is useful for rewriting a formula in normal form into its complete normal form.

Lemma 5.5. Any PPTL* formula in its normal form can be further rewritten into its complete normal form.

Proof. The lemma can be proved in a similar way to the corresponding proof in Duan *et al.* (2008b). □

We now come to the main conclusion.

Theorem 5.2. Any PPTL* formula R can be rewritten to its normal form.

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

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

$$R \equiv p \wedge (empty \vee more) \equiv p \wedge empty \vee p \wedge \bigcirc\ true\,.$$

Induction step: Suppose P_i, $1 \leqslant i \leqslant m$, and Q can be rewritten to their normal forms as follows:

$$\begin{aligned} P_i &\equiv P_{1e} \wedge empty \vee \bigvee_{j=0}^{r}(P_{ij} \wedge \bigcirc P'_{ij}) \\ Q &\equiv Q_e \wedge empty \vee \bigvee_{j=0}^{r}(Q_j \wedge \bigcirc Q'_j)\,. \end{aligned}$$

Then:

1. If $R \equiv P_1 \vee P_2$, the proof is straightforward.
2. If $R \equiv \bigcirc Q$,
$$\begin{aligned} R &\equiv \bigcirc(Q_e \wedge empty \vee \bigvee_{j=0}^{r}(Q_j \wedge \bigcirc Q'_j)) \\ &\equiv \bigcirc(Q_e \wedge empty) \vee \bigvee_{j=0}^{r} \bigcirc(Q_i \wedge \bigcirc Q'_i)\,. \end{aligned}$$
3. If the formula R is in the form $\neg Q$, by Lemma 5.5, Q can be rewritten to its complete normal form:
$$Q \equiv Q_e \wedge empty \vee \bigvee_{i=0}^{2^n-1}(m_i \wedge \bigcirc M'_i)\,.$$
Thus, we have
$$\neg Q \equiv \neg Q_e \wedge empty \vee \bigvee_{i=0}^{2^n-1}(m_i \wedge \bigcirc \neg M'_i)\,.$$
4. If $R \equiv (P_1, \ldots, P_m)\ prj\ Q$, the conclusion holds by Lemma 5.3.
5. If $R \equiv (P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)\ prj\ Q$, the conclusion holds by Lemma 5.4.

Therefore, any PPTL* formula can be rewritten into its normal form. □

We can now write pseudo codes for algorithm NF, as shown in Table 3. The basic constructs such as atomic propositions, the *next*, *disjunction*, *negation*, *projection* and *projection star* constructs need to be considered. However, to improve the efficiency, *chop* and *chop star* constructs are also taken into account.

In algorithm NF, the sub-algorithm CNF is used to transform a normal form into its complete normal form, while algorithm NEG is used to negate a complete normal form obtained from algorithm CNF. Algorithms PRJ and CHOP, respectively, are used to

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Function NF(R)
/* precondition: R is a PPTL* formula */
/* postcondition: NF(R) computes an equivalent normal form for formula R*/
begin function
case
R is an atomic proposition p: **return** $p \wedge empty \vee p \wedge \bigcirc true$;
R is $\neg P$: **return** NEG(CNF(NF(P)));
R is $\bigcirc P$: **If** $P \equiv \bigvee_{i=2}^{i \in N_0} P_i$ **return** $\bigvee_{i=2}^{i \in N_0} \bigcirc P_i$; **else return** $\bigcirc P$;
R is $P \vee Q$: **return** NF(P)$\vee$NF(Q);
R is $P;Q$: **return** CHOP(R);
R is $empty \vee (Q;Q^*)$: **return** $empty \vee$ CHOP($Q;Q^*$);
R is $(P_1,\ldots,P_m)$ prj Q: **return** PRJ(R);
R is $((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)$ prj $Q) \vee ((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,$ $P_m)$ prj $Q)$ **return**: PRJ$((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)$ prj $Q) \vee$PRJ$((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,$ $(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)$ prj $Q)$;
end case
end function

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

transform the formulas in *projection* and *chop* constructs to their normal forms. These were given in Duan *et al.* (2008b). Note that for *chop star* and *projection star* constructs, for the convenience of the proofs given in the Appendix, we deal with formula $empty \vee (Q;Q^*)$ instead of Q^* since $Q^* \equiv empty \vee (Q;Q^*)$, and

$$((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)\ prj\ Q) \vee((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)\ prj\ Q))$$

instead of $(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)$ prj $Q)$ since

$$(P_1,\ldots,(P_i,\ldots,P_s)^{\circledast},\ldots,P_m)\ prj\ Q \equiv ((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)\ prj\ Q) \vee((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)\ prj\ Q).$$

6. A decision procedure for checking the satisfiability of PPTL* formulas

In this section we present a decision procedure for checking the satisfiability of PPTL* formulas over infinite models. The decision procedure is based mainly on a so-called Labelled Normal Form Graph (LNFG) (Duan *et al.* 2008b). It is generated from the Normal Form Graph (NFG), which is constructed according to the normal form.

6.1. *Labelled 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 $CL(P)$, each node is specified by a formula in PPTL*, while in $EL(P)$, each edge is a directed arc labelled with a state formula Q_e from node Q to node R and identified by a triple, (Q, Q_e, R). $CL(P)$ and $EL(P)$ of G can be defined inductively as in Definition 6.1.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Function LNFG(P)
/* precondition: P is a PPTL* formula in pre-prepared form*/
/* postcondition: LNFG(P) computes LNFG of P, $G = (CL(P), EL(P), V_0, V_f)$*/
begin function
$CL(P) = \{P\}$; $EL(P) = \varnothing$; $V_0 = \{P\}$; $V_f = \varnothing$; $mark[P] = 0$; AddE = AddN = 0;
while there exists $R \in CL(P) \setminus \{\varepsilon, false\}$, and $mark[R]$ ==0
do $mark[R]$ =1; /*marking R is decomposed*/
if R *is* Q_1 ; Q_2 /*computing LNFG of Q_1*/
then $V_f = V_f \cup \{R\}$; $G' = (CL(Q_1), EL(Q_1), V_0', V_f')$ =LNFG(Q_1);
if $\varepsilon \notin CL(Q_1)$ **then continue**; /*no need to decompose R, jump to while*/
Q =NF(R);
case
Q is $\bigvee_{j=0}^{h} Q_{ej} \wedge empty$: AddE=1; /*first part of NF needs to be added*/
Q is $\bigvee_{i=0}^{k} Q_i \wedge \bigcirc Q_i'$: AddN=1; /*second part of NF needs to be added*/
Q is $\bigvee_{j=0}^{h} Q_{ej} \wedge empty \vee \bigvee_{i=0}^{k} Q_i \wedge \bigcirc Q_i'$: AddE=AddN=1;
/*both parts of NF need to be added*/
end case
if AddE == 1 **then** /*add first part of NF*/
$CL(P) = CL(P) \cup \{\varepsilon\}$;
$EL(P) = EL(P) \cup \bigcup_{j=0}^{h} \{(R, Q_{ej}, \varepsilon)\}$;
AddE=0;
if AddN == 1 **then** /*add second part of NF*/
for $i = 0$ **to** k **do if** Q_i' is $false$
then $mark[Q_i']$=1; /*no need to decompose Q_i'*/
else if $Q_i' \notin CL(P)$
then $mark[Q_i']$=0; /*Q_i' needs to be decomposed*/
$CL(P) = CL(P) \cup \bigcup_{i=0}^{k} \{Q_i'\}$;
$EL(P) = EL(P) \cup \bigcup_{i=0}^{k} \{(R, Q_i, Q_i')\}$;
AddN=0;
end while
return G;
end function

Table 4. *Algorithm for constructing the LNFG of a PPTL* formula*

Definition 6.1. For a PPTL* formula P, the set $CL(P)$ of nodes and the set $EL(P)$ of edges connecting nodes in $CL(P)$ are defined inductively as follows:

1. $P \in CL(P)$.
2. For all $Q \in CL(P) \setminus \{\varepsilon, false\}$, if

$$Q \equiv \bigvee_{j=0}^{h} (Q_{ej} \wedge empty) \vee \bigvee_{i=0}^{k} (Q_{ci} \wedge \bigcirc Q_i'),$$

then

$$\varepsilon \in CL(P), \quad (Q, Q_{ej}, \varepsilon) \in EL(P) \text{ for each } j, \quad 0 \leqslant j \leqslant h$$

and

$$Q_i' \in CL(P), \quad (Q, Q_{ci}, Q_i') \in EL(P) \text{ for all } i, \ 0 \leqslant i \leqslant k.$$

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

As shown by the analysis in Duan *et al.* (2008b), the NFG of formula P generated by Definition 6.1 might wrongly describe models of P with the chop construct. To solve the problem, we have formalised Algorithms LNFG and SIMPLIFY, as shown in Tables 4 and 5,

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Function SIMPLIFY(G)
/* precondition: $G = (CL(P), EL(P), V_0, V_f)$ is an LNFG of PPTL* formula P*/
/* postcondition: SIMPLIFY(G) computes an LNFG of P, $G' = (CL'(P), EL'(P), V'_0, V'_f)$, that contains no redundant nodes*/
begin function
$CL'(P) = CL(P)$; $EL'(P) = EL(P)$; $V'_0 = V_0$; $V'_f = V_f$;
while $\exists\ R \in CL'(P)$ and R is not ε and has no edges departing from
do $CL'(P) = CL'(P) \setminus \{R\}$; $EL'(P) = EL'(P) \setminus \bigcup_i (R_i, R_e, R)$;
/* $\bigcup_i (R_i, R_e, R)$ denotes the set of edges connecting to node R*/
if $R \in V'_f$, $V_f = V'_f \setminus \{R\}$;
if $R \in V'_0$, $V_0 = V'_0 \setminus \{R\}$;
end while
return G';
end function

Table 5. *Algorithm for simplifying the LNFG of a PPTL* formula*

to generate a Labelled NFG (which is a subgraph of the NFG) for a given formula so that models of the formula can be generated correctly.

In Algorithm LNFG, whenever the *chop* operator is the main operator of a formula, F is placed in the node as a mark. The algorithm uses *mark*[] to indicate whether a formula needs to be decomposed. If *mark*$[P] = 0$ (unmarked), P needs to be decomposed further, otherwise *mark*$[P] = 1$ (marked) means P has been decomposed already or does not need to be decomposed. Also, two global boolean variables AddE and AddN, respectively, are used to indicate whether the terminating and future parts in the normal form have been encountered. The termination of Algorithm LNFG is shown in the Appendix.

Algorithm SIMPLIFY is used to remove the redundant nodes from the LNFG constructed by Algorithm LNFG. Note that when constructing the LNFG of formula P, we first rewrite P into a pre-prepared form containing only the operators considered in Algorithm NF.

The LNFG of formula P can be expressed as a graph $G = (V, E, v_0, V_f)$, where V denotes the set of nodes in the LNFG, E is the set of directed edges among V, $v_0 \in V$ is the initial (or root) node and $V_f \subseteq V$ denotes the set of nodes with finite label F. V and E in G can be constructed inductively by Algorithm LNFG.

6.2. *Decision procedure*

In the LNFG of formula P, a finite path, $\Pi = \langle P, P_e, P_1, P_{1e}, \ldots, \varepsilon \rangle$, is an alternating sequence of nodes and edges from the root to the ε node, while an infinite path, $\Pi = \langle P, P_e, P_1, P_{1e}, \ldots \rangle$, is an infinite alternating sequence of nodes and edges emanating from the root. In fact, a finite path in the LNFG of formula P corresponds to a finite model and an infinite path, where F occurs only finitely often, in the LNFG of P corresponds to an infinite model of P. This can be proved in the same way as in Duan *et al.* (2008b) and is stated formally in Theorems 6.1 and 6.2. Obviously, P is satisfiable if and only if there exist finite or infinite paths, where F occurs only finitely often, in the LNFG of P. Consequently, a decision algorithm for checking the satisfiability of a PPTL* formula P can be constructed based on the LNFG of P. A pseudo code skeleton of the algorithm, function CHECK, is given in Table 6.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

```
Function CHECK(P)
/* precondition: P is a PPTL* formula*/
/* postcondition: CHECK(P) checks whether formula P is satisfiable or not*/
begin function
    G =LNFG(P);
    G' =SIMPLIFY(G);
    if there exist no paths in G', return unsatisfiable;
    else return satisfiable;
end function
```

Table 6. *Algorithm for checking the satisfiability of PPTL* formulas*

Theorem 6.1. A formula Q of PPTL* can be satisfied by finite models if and only if there exist finite paths in the LNFG of Q.

Theorem 6.2. A formula Q of PPTL* can be satisfied by infinite models if and only if there exist infinite paths, where F occurs only finitely often, in the LNFG of Q.

From Theorem 4.1 and the decision procedure given above, we can obtain the following conclusion.

Corollary 6.1. The complexity of PPTL* is non-elementary.

In fact, Theorem 4.1 holds for both PPTL and PITL since the proof only involves the temporal operators *chop* and *next*. In addition, we have given decision procedures for PPTL and PITL in Duan *et al.* (2008b), so we also have the following results.

Corollary 6.2.

1. The complexity of PPTL is non-elementary.
2. The complexity of PITL is non-elementary.

7. Conclusions

In this paper we have proved that the complexity of PPTL* is non-elementary. We have also presented a decision procedure for checking the satisfiability of the logic. This enables us to verify automatically properties of concurrent systems with PPTL* by means of model checking. The specification language PLTL is not powerful enough to describe all the ω-regular properties (Holzmann 1997) that can be verified in the well-known verification system SPIN. However, these properties can be specified by the more powerful logic with *chop* operator (Wolper 1983). Our experience has been that 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++ (Tian and Duan 2007). With this model checker, the negation of a property is first described by a PPTL* formula, and then an LNFG of the formula is constructed using the algorithm given in this paper. The LNFG has also been transformed to an equivalent Büchi automaton, which was then described in Never Claim[†]. The model part is still described by PROMELA in the same way as in SPIN.

[†] A Büchi automaton in the syntax of PROMELA

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

In our future research we intend to focus on:

1. verifying several big examples as case studies to evaluate our approach;
2. making improvements to the model checker prototype to make it practical software, and developing a practical verification environment for PPTL*;
3. formalising axiom systems for PPTL* in both propositional and first-order cases.

Appendix A. Finiteness of LNFGs

We prove the finiteness of LNFGs (and the termination of Algorithm LNFG) in this appendix by computing the maximum number of nodes in the LNFG of a PPTL* formula P, which we show is $(12 + l)^n$, where l is the number of atomic propositions appearing in P and n is the length of formula P. For convenience, we will first define some notation.

For any PPTL* formula Q, if Q is rewritten into its normal form

$$Q \equiv Q_e \wedge empty \vee \bigvee_{i=0}^{r} (Q_i \wedge \bigcirc Q_i').$$

then *empty* and each Q_i' are called succ-formulas of Q. We use $succ(Q)$ to denote the set of all succ-formulas of Q.

The length of a formula Q, denoted $length(Q)$ (or $|Q|$), is defined inductively in Definition A.1. Note that we need only be concerned with the constructs considered in Algorithm NF.

Definition A.1. If θ is an atomic proposition, derived formula, *true*, *false* or *empty*, then $length(\theta)$=1.

If P_i, $1 \leqslant i \leqslant m$ and Q are PPTL* formulas with $length(P_i) = n_i$ and $length(Q) = n$, then:

— For unary operators $\neg$ or $\bigcirc$, denoted Θ_1, we have $length(\Theta_1 Q) = n + 1$.
— For binary operators ; or $\vee$, denoted Θ_2, we have $length(P_1 \Theta_2 P_2) = n_1 + n_2 + 1$.
— For operator *prj*, we have

$$length((P_1, \ldots, P_m)\ prj\ Q) = n_1 + \cdots + n_m + m + n.$$

— For $empty \vee Q; Q^*$, we have

$$\begin{aligned} length(empty \vee Q; Q^*) &= length(empty) + 1 + lengh(Q) + 1 + length(Q) + 1 \\ &= 2 \times n + 4. \end{aligned}$$

— For

$$\begin{aligned} &((P_1, \ldots, P_{i-1}, P_{s+1}, \ldots, P_m) prj\ Q) \\ &\qquad \vee ((P_1, \ldots, P_{i-1}, P_i, \ldots, P_s, (P_i, \ldots, P_s)^{\circledast}, P_{s+1}, \ldots, P_m) prj\ Q). \end{aligned}$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

we have

$$\begin{aligned} &length(((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)prj\ Q)\\ &\quad\vee((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)prj\ Q))\\ &\qquad= length(P_1)+\cdots+length(P_{i-1})+length(P_{s+1})+\cdots\\ &\qquad\quad+length(P_m)+i+(m-s+1)+length(Q)+1+length(P_1)+\cdots\\ &\qquad\quad+length(P_m)+length(P_i)+\cdots\\ &\qquad\quad+length(P_s)+m+s-i+2+length(Q)\\ &\qquad= 2n_1+..+2n_m+2m+2n+4. \end{aligned}$$

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

Lemma A.1. Suppose for each formula P_i and Q, $1 \leqslant i \leqslant m$, the length of each succ-formula of P_i (or Q) is no larger than the length of P_i (or Q). Then the length of each succ-formula of $(P_1,\ldots,P_m)\ prj\ Q$ is no larger than the length of $(P_1,\ldots,P_m)\ prj\ Q$.

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

$$\begin{aligned} P_1 &\equiv P_{1e}\wedge empty\vee\bigvee_{i=0}^{n}(P_{1i}\wedge\bigcirc P'_{1i})\\ Q &\equiv Q_e\wedge empty\vee\bigvee_{k=0}^{n'}(Q_k\wedge\bigcirc Q'_k). \end{aligned}$$

By hypothesis, $|empty|\leqslant|P_1|$, $|empty|\leqslant|Q|$, for each i, $0\leqslant i\leqslant n$, $|P'_{1i}|\leqslant|P_1|$, and for each k, $0\leqslant k\leqslant n'$, $|Q'_k|\leqslant|Q|$. We have

$$\begin{aligned} P_1\ prj\ Q \equiv\ & P_{1e}\wedge Q_e\wedge empty\vee\bigvee_{i=0}^{n}(P_{1i}\wedge Q_e\wedge\bigcirc P'_{1i})\\ &\vee\bigvee_{i=0}^{n}\bigvee_{k=1}^{n'}(P_{1i}\wedge Q_k\wedge\bigcirc(P'_{1i};Q'_k))\\ &\vee\bigvee_{k=0}^{n'}(P_{1e}\wedge Q_k\wedge\bigcirc Q'_k). \qquad (L_{17-19},L_{21-23}) \end{aligned}$$

So,

$$succ(P_1\ prj\ Q)=\{empty\}\cup\bigcup_{i=0}^{n}(P'_{1i})\cup\bigcup_{i=0}^{n}\bigcup_{k=0}^{n'}(P'_{1i};Q'_k)\cup\bigcup_{k=0}^{n'}(Q'_k).$$

Obviously:

— $|empty|\leqslant|P_1\ prj\ Q|$;
— for each P'_{1i} and Q'_k, we have $|P'_{1i}|\leqslant|P_1|\leqslant|P_1\ prj\ Q|$ and $|Q'_k|\leqslant|Q|\leqslant|P_1\ prj\ Q|$;
— for each $P'_{1i};Q'_k$, we have $|P'_{1i};Q'_k|=|P'_{1i}\ prj\ Q'_k|\leqslant|P_1\ prj\ Q|$.

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

$$(P_2,\ldots,P_m)\ prj\ Q\equiv R_e\wedge empty\vee\bigvee_{j=0}^{t}(R_j\wedge\bigcirc R'_j).$$

For *empty* and each R'_j, we have

$$|empty|\leqslant|(P_2,\ldots,P_m)\ prj\ Q|,\quad |R'_j|\leqslant|(P_2,\ldots,P_m)\ prj\ Q|.$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

By L_{16-19} and L_{21-23}, we have:

$$\begin{aligned}(P_1,\ldots,P_m)\ prj\ Q \equiv\ & P_{1e}\wedge R_e\wedge empty\vee\bigvee_{j=0}^{t}(P_{1e}\wedge R_j\wedge\bigcirc R'_j)\\ & \vee\bigvee_{i=0}^{n}\bigvee_{k=0}^{n'}(P_{1i}\wedge Q_k\wedge\bigcirc(P'_{1i};((P_2,\ldots,P_m)\ prj\ Q'_k)))\\ & \vee\bigvee_{i=0}^{n}(P_{1i}\wedge Q_e\wedge\bigcirc(Q'_{1i};P_2;\ldots;P_m))\end{aligned}$$

So,

$$succ((P_1,\ldots,P_m)\ prj\ Q)$$
$$=$$
$$\{empty\}\cup\bigcup_{j=0}^{t}(R'_j)\cup\bigcup_{i=0}^{n}\bigcup_{k=0}^{n'}(P'_{1i};((P_2,\ldots,P_m)\ prj\ Q'_k))\cup\bigcup_{i=0}^{n}(Q'_{1i};P_2;\ldots;P_m).$$

Obviously:

— $|empty|\leqslant|(P_1,\ldots,P_m)\ prj\ Q|$;
— for each R'_j, we have $|R'_j|\leqslant|(P_2,\ldots,P_m)\ prj\ Q|\leqslant|(P_1,\ldots,P_m)\ prj\ Q|$;
— for each $P'_{1i};((P_2,\ldots,P_m)\ prj\ Q'_k)$, we have

$$|P'_{1i};((P_2,\ldots,P_m)\ prj\ Q'_k)|\leqslant|(P_1,\ldots,P_m)\ prj\ Q|\,;$$

— for each $Q'_{1i};P_2;\ldots;P_m$, we have

$$|Q'_{1i};P_2;\ldots;P_m|\leqslant|(P_2,\ldots,P_m)\ prj\ Q|\leqslant|(P_1,\ldots,P_m)\ prj\ Q|.$$

Thus, the lemma holds. □

Lemma A.1 confirms that when projection construct P is rewritten, the length of any formulas in $succ(P)$ cannot be increased.

Lemma A.2. Suppose the length of each succ-formula of P is no larger than the length of P. Then the length of each succ-formula of $empty\vee(P;P^*)$ is no larger than the length of $empty\vee(P;P^*)$.

Proof. Suppose P has been rewritten into its normal form:

$$P\equiv P_e\wedge empty\vee\bigvee_{i=0}^{r}(P_i\wedge\bigcirc P'_i).$$

By hypothesis, $|empty|\leqslant|P|$, and for each i, $0\leqslant i\leqslant r$, $|P'_i|\leqslant|P|$. By lemma 5.2,

$$empty\vee(P;P^*)\equiv empty\vee\bigvee_{i=0}^{r}P_i\wedge\bigcirc(P'_i;P^*).$$

So $Succ(P^*)=\{empty\}\cup\bigcup_{i=0}^{r}(P_i;P^*)$, and, clearly, $|empty|\leqslant|empty\vee(P;P^*)|$ and $|P_i;P^*|\leqslant|P;P^*|\leqslant|empty\vee(P;P^*)|$ for each i. □

Lemma A.2 states that when a *chop star* construct P^* is rewritten, the length of any formulas in $succ(P^*)$ cannot be increased.

Lemma A.3. Suppose for each formula P_i and Q, $1\leqslant i\leqslant m$, the length of each succ-formula of P_i (or Q) is no larger than the length of P_i (or Q). Then the length of each

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

succ-formula of

$$((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)prj\ Q) \vee((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)prj\ Q)$$

is no larger than the length of

$$((P_1,\ldots,P_{i-1},P_{s+1},\ldots,P_m)prj\ Q) \vee((P_1,\ldots,P_{i-1},P_i,\ldots,P_s,(P_i,\ldots,P_s)^{\circledast},P_{s+1},\ldots,P_m)prj\ Q).$$

Proof. This lemma can be proved in a similar way to Lemma A.1 □

Lemma A.3 tells us that when a *projection star* construct P is rewritten, the length of any formulas in $succ(P)$ cannot be increased.

Lemma A.4. When any PPTL* formula P is rewritten into its normal form, the length of each succ-formula of P is no larger than the length of P.

Proof. The proof is by induction on the structure of PPTL* formulas composed of the constructs considered in Algorithm NF.

Base case: P is an atomic proposition p. Rewrite p to its normal form:

$$p \equiv p \wedge empty \vee p \wedge \bigcirc true.$$

For the succ-formulas *empty* and *true*, we have $|empty| \leqslant |p|$ and $|true| \leqslant |p|$.

Induction step: Suppose that when each formula P_i (or Q), $1 \leqslant i \leqslant m$, is rewritten into its normal form, the length of each succ-formula of P_i (or Q) will be no larger than the length of P_i (or Q). Then:

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

— $P \equiv \neg P_1$:
Suppose the complete normal form of P_1 is as follows:

$$P_1 \equiv (P_{1e} \wedge empty) \vee \bigvee_{i=0}^{r}(P_{1i} \wedge \bigcirc P'_{1i}).$$

So,

$$\neg P_1 \equiv (\neg P_{1e} \wedge empty) \vee \bigvee_{i=0}^{r}(P_{1i} \wedge \bigcirc \neg P'_{1i}).$$

By hypothesis, $|empty| \leqslant |P_1|$ and $|P'_{1i}| \leqslant |P_1|$, $0 \leqslant i \leqslant r$, so we have

$$|empty| \leqslant |P_1| < 1 + |P_1| = |\neg P_1|$$

and

$$|\neg P'_{1i}| = 1 + |P'_{1i}| \leqslant 1 + |P_1| = |\neg P_1|, \quad 0 \leqslant i \leqslant r.$$

— $P \equiv P_1 \vee P_2$:
Let

$$P_1 \equiv (P_{1e} \wedge empty) \vee \bigvee_{i=0}^{r}(P_{1i} \wedge \bigcirc P'_{1i})$$
$$P_2 \equiv (P_{2e} \wedge empty) \vee \bigvee_{j=0}^{k}(P_{2j} \wedge \bigcirc P'_{2j}).$$

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

So,

$$P_1 \vee P_2 \equiv (P_{1e} \vee P_{2e}) \wedge empty \vee \bigvee_{i=0}^{r}(P_{1i} \wedge \bigcirc P'_{1i}) \vee \bigvee_{j=0}^{k}(P_{2j} \wedge \bigcirc P'_{2j}).$$

By hypothesis, $|empty| \leqslant |P_1|$, $|empty| \leqslant |P_2|$, $|P'_{1i}| \leqslant |P_1|$, $0 \leqslant i \leqslant r$ and $|P'_{2j}| \leqslant |P_2|$, $0 \leqslant j \leqslant k$. So we have

$$|empty| \leqslant |P_1| < |P_1| + |P_2| + 1 = |P_1 \vee P_2|$$

$$|P'_{1i}| \leqslant |P_1| < |P_1| + |P_2| + 1 = |P_1 \vee P_2|, \quad 0 \leqslant i \leqslant r$$

and

$$|P'_{2i}| \leqslant |P_2| < |P_1| + |P_2| + 1 = |P_1 \vee P_2|.$$

— $P \equiv P_1; P_2$:
Let

$$P_1 \equiv (P_{1e} \wedge empty) \vee \bigvee_{i=0}^{r}(P_{1i} \wedge \bigcirc P'_{1i})$$
$$P_2 \equiv (P_{2e} \wedge empty) \vee \bigvee_{j=0}^{k}(P_{2j} \wedge \bigcirc P'_{2j}).$$

So,

$$P_1; P_2 \equiv P_{1e} \wedge P_{2e} \wedge empty \vee P_{1e} \wedge \bigvee_{j=0}^{k}(P_{2j} \wedge \bigcirc P'_{2j}) \vee \bigvee_{i=0}^{r}(P_{1i} \wedge \bigcirc(P'_{1i}; P_2)).$$

By hypothesis, $|empty| \leqslant |P_1|$, $|empty| \leqslant |P_2|$, $|P'_{1i}| \leqslant |P_1|$, $0 \leqslant i \leqslant r$ and $|P'_{2j}| \leqslant |P_2|$, $0 \leqslant j \leqslant k$, so we have

$$|empty| \leqslant |P_1| < |P_1| + |P_2| + 1 = |P_1; P_2|$$

$$|P'_{2j}| \leqslant |P_2| < |P_1| + |P_2| + 1 = |P_1; P_2|, \quad 0 \leqslant i \leqslant r$$

and

$$|P'_{1i}; P_2| = |P'_{1i}| + |P_2| + 1 \leqslant |P_1| + |P_2| + 1 = |P_1; P_2|.$$

— $P \equiv (P_1, \ldots, P_m)\ prj\ Q$:
This case was proved in in Lemma A.1.

— $P \equiv empty \vee (P_1; P_1^*)$:
This case was proved in Lemma A.2.

— $P \equiv ((P_1, \ldots, P_{i-1}, P_{s+1}, \ldots, P_m) prj\ Q)$
$\vee((P_1, \ldots, P_{i-1}, P_i, \ldots, P_s, (P_i, \ldots, P_s)^{\circledast}, P_{s+1}, \ldots, P_m) prj\ Q)$:
This case was proved in Lemma A.3.

This completes the proof of the lemma. □

Lemma A.4 guarantees that when any PPTL* formulas P are rewritten, it is impossible that the length of any formulas in $succ(P)$ is increased. We will now prove the following theorem on the maximum number of nodes in the LNFG of any PPTL* formula.

Theorem A.1. For any PPTL* formula Q, let $|Q| = n$, and let Q_p denote the set of atomic propositions appearing in Q, and $|Q_p| = l$. Let the LNFG of Q be $G = (V, E, v_0, V_f)$. Then we have $|V| \leqslant (12 + l)^n$.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Proof. By Algorithm LNFG, the root node of the LNFG is Q itself. The other nodes of the LNFG of Q are generated by repeatedly rewriting the newly generated succ-formulas into their normal forms. Also, Lemma A.4 confirms that when written into the normal form, the length of each succ-formula of Q is no larger than the length of Q. Moreover, each node (formula) in the LNFG of Q is composed of basic connectives, $\neg, \wedge, \vee, \bigcirc, ;, *, prj, \circledast$, and comma $(,)^{\dagger}$ brought forth by prj, atomic propositions appearing in Q, as well as *true* and *false*. Accordingly, there are at most $(11 + l)$ symbols appearing in a formula. In addition, each formula is no longer than n. Hence, by the principle of permutation and combination, at most $(12+l)^n$ formulas (as nodes) can appear in the LNFG of Q, leading to $|V| \leqslant (12 + l)^n$. □

References

Barringer, H., Kuiper, R. and Pnueli, A. (1984) Now You May Compose Temporal Logic Specification. In: *Proc. 16th STOC* 51–63.

Bowman, H. and Thompson, S. (2003) A decision procedure and complete axiomatization of interval temporal logic with projection. *Journal of logic and Computation* **13** (2) 195–239.

Chandra, A., Halpern, H., Meyer, A. and Parikh, R. (1985) Equations between Regular Terms and an Application to Process Logic. *SIAM J. Computing* **14** 932–942.

Duan, Z. (2006) *Temporal Logic and Temporal Logic Programming*, Science Press of China.

Duan, Z. and Koutny, M. (2004) A Framed Temporal Logic Programming Language. *Journal of Computer Science and Technology* **19** (3) 341–351.

Duan, Z. and Tian, C. (2007) Decidability of Propositional Projection Temporal Logic with Infinite Models. In: Proceedings, Theory and Applications of Models of Computation 2007. *Springer-Verlag Lecture Notes in Computer Science* **4484** 521–532.

Duan, Z., Koutny, M. and Holt, C. (1994) Projection in temporal logic programming. In: Logic Programming and Automatic Reasoning. *Springer-Verlag Lecture Notes in Artificial Intelligence* **822** 333–344.

Duan, Z., Yang, X. and Koutny, M. (2008a) Framed Temporal Logic Programming. *Science of Computer Programming* **70** 31–61.

Duan, Z., Tian, C. and Zhang, L. (2008b) A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. *Acta Informatica* **45** (1) 43–78.

Dutertre, B. (1995) Complete proof systems for first order interval temporal logic. In: *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society 36–43.

Harel, D., Kozen, D. and Parikh, R. (1982) Process logic: Expressiveness, decidability, completeness. *J. Comput. System Sci.* **25** 144–170.

Holzmann, J. (2003) The Model Checker Spin. *IEEE Trans. on Software Engineering* **23** (5) 279–295.

Kono, S. (1993) A combination of clausal and non-clausal temporal logic programs. *Springer-Verlag Lecture Notes in Artificial Intelligence* **897** 40–57.

Kripke, S. (1963) Semantical analysis of modal logic I: normal propositional calculi. *Z. Math. Logik Grund. Math* **9** 67–96.

Liu, S. and Wang, H. (2007) An automated approach to specification animation for validation. *The Journal of Systems and Software* **80** (8) 1271–1285.

† Here ',' is used in prj and $prj^{\circledast}$ constructs.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press

Liu, S. and Chen, Y. (2008) A relation-based method combining functional and structural testing for test case generation. *The Journal of Systems and Software* **81** (2) 234–248.

Manna, Z. and Pnueli, A. (1992) *The temporal logic of reactive and concurrent systems*, Springer-Verlag.

Moszkowski, B. (1983) *Reasoning about digital circuits*, Ph. D. Thesis, Department of Computer Science, Stanford University.

Moszkowski, B. (1995) Compositional reasoning about projected and infinite time. *Engineering of Complex Computer Systems*, IEEE Computer Society Press 238–245.

Moszkowski, B. (2004) A hierarchical completeness proof for propositional temporal logic. In: Dershowitz, N. (ed.) Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of his 64th Birthday. *Springer-Verlag Lecture Notes in Computer Science* **2772** 480–523.

Rosner, R. and Pnueli, A. (1986) A choppy logic. In: *Proceedings, Symposium on Logic in Computer Science*, IEEE Computer Society 306–314.

Stockmeyer, L. (1974) *The Complexity of Decision Problems in Automata Theory and Logic*, Ph. D. thesis, MIT (available as Project MAC Technical Report 133).

Tarski, A. (1955) A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific J. Math* **5** 285–309.

Tian, C. and Duan, Z. (2007) Model Checking Propositional Projection Temporal Logic Based on SPIN. In: ICFEM 2007. *Springer-Verlag Lecture Notes in Computer Science* **4789** 246–265.

Tian, C. and Duan, Z. (2008) Propositional Projection Temporal Logic, Büchi Automata and ω-Regular Expressions. In: Proceedings, Theory and Applications of Models of Computation 2008. **4978** 47–58.

Wang, H. and Xu, Q. (2004) Completeness of Temporal Logics over Infinite Models. *Discrete Applied Mathematics* **136** 87–103.

Wolper, P. (1983) Temporal logic can be more expressive. *Information and Control* **56** 72–99.

Zhou, C., Hoare, C. A. R. and Ravn, A. P. (1991) A calculus of duration. *Information Processing Letters* **40** (5) 269–275.

https://doi.org/10.1017/S096012950800738X Published online by Cambridge University Press