Propositional Projection Temporal Logic, B$\ddot{u}$chi Automata and ω-Regular Expressions*

Cong Tian and Zhenhua Duan

Institute of Computing Theory and Technology
Xidian University
{ctian,zhhduan}@mail.xidian.edu.cn

Abstract. This paper investigates the language class defined by Propositional Projection Temporal Logic with star (PPTL with star). To this end, B$\ddot{u}$chi automata are first extended with stutter rule (SBA) to accept finite words. Correspondingly, ω-regular expressions are also extended (ERE) to express finite words. Consequently, by three transformation procedures between PPTL with star, SBA and ERE, PPTL with star is proved to represent exactly the full regular language.

Keywords: Propositional Projection Temporal Logic, B$\ddot{u}$chi automata, ω-regular expression, expressiveness.

1 Introduction

Temporal logic is a useful formalism for describing sequences of transitions between states in a reactive system. In the past thirty years, many kinds of temporal logics are proposed within two categories, linear-time and branching-time logics. In the community of linear-time logics, the most widely used logics are Linear Temporal Logic (LTL) [?] and its variations. In the propositional framework, Propositional LTL (PLTL) has been proved to have the expressiveness of star-free regular expressions [12,16]. Considering the expressive limitation of PLTL, extensions such as Quantified Linear Time Temporal Logic (QLTL) [13], Extended Temporal Logic (ETL) [9,14] and Linear mu-calculus (νTL) [15] etc, were introduced to PLTL to express the full regular language. Nevertheless, results [17,18,19,20] have shown that temporal logic needs some further extensions in order to support a compositional approach to the specification and verification of concurrent systems. These extensions should enable modular and compositional reasoning about loops and sequential composition as well as about concurrent ones. Therefore, kinds of extensions are proposed. Prominently, one of the important extensions is the addition of the *chop* operator. The work in [9] showed that process logic with both *chop* operator and its reflexive-transitive closure (*chop star*), which is called *slice* in process logic, is strictly more expressive. The resulting logic is still decidable and in fact has the expressiveness of full regular expressions.

* This research is supported by the NSFC Grant No.60433010.

Interval Temporal Logic (ITL) [4] is an easily understood temporal logic with *next, chop* and a projection operator *proj*. In the two characteristic operators, *chop* implements a form of sequential composition while *proj* yields repetitive behaviors. ITL without projection has similar expressiveness as Rosner and Pnueli's choppy logic [3]. Further, addition of the *proj* operator will brings more powerful expressiveness, since repetitive behaviors are allowed. However, no systematic proofs have been given in this aspect. Projection Temporal Logic (PTL) [5] is an extension of ITL. It extends ITL to include infinite models and a new *projection* construct, $(P_1, ..., P_m)$ *prj* Q, which is much more flexible than the original one. However, in the propositional case[1], the *projection* construct needs further to be extended to *projection star*, $(P_1, ..., (P_i, ..., P_j)^{\circledast}, ..., P_m)$ *prj* Q, so that it can subsume *chop*, *chop star*, and the original projection (*proj*) in [4]. This extension makes the underlying logic more powerful without the lose of decidability [22].

Within PTL, plenty of logic laws have been formalized and proved [5], and a decision procedure for checking the satisfiability of Propositional Projection Temporal Logic (PPTL) formulas are given in [6,7]. Based on the decision procedure, a model checking approach based on SPIN for PPTL is proposed [8]. Further, in [22], *projection star* is introduced to PPTL, and the satisfiability for PPTL with star formulas is proved to be still decidable. Instinctively, PPTL with star is powerful enough to express the full regular expression. Thus, by employing PPTL with star formulas as the property specification language, the verification of concurrent systems with the model checker SPIN will be completely automatic. This will overcome the error-prone hand-writing of a never claim in the original SPIN since some properties cannot be specified by PLTL formulas. Further, since PPTL with star can subsume *chop* construct, compositional approach for the specification and verification of concurrent systems with SPIN will be allowed. Therefore, we are motivated to give a systematic proof concerning the expressiveness of PPTL with star formulas. To this end, stutter B*ü*chi automata and extended ω-regular expressions are introduced first. Subsequently, by three transformation procedures beteen PPTL with star, SBA, and ERE, PPTL with star is proved to represent exactly the full regular language.

The paper is organized as follows. The syntax and semantics of PPTL with star are briefly introduced in the next section. Section 3 and Section 4 present the definition of stutter B*ü*chi automata and extended regular expressions respectively. Section 5 is devoted to proving the expressiveness of PPTL with star formulas. Precisely, three transformations between PPTL with star, SBA and ERE are given. Finally, conclusions are drawn in Section 6.

2 Propositional Projection Temporal Logic with Star

Our underlying logic is propositional projection temporal logic with star. It extends PPTL to include *projection star*. It is an extension of propositional interval temporal Logic (PITL).

[1] In the first order case, *projection star* is a derived formula.

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

$$P ::= p \mid \bigcirc P | \neg P | P \vee Q | (P_1, ..., P_m)\ prj\ Q | (P_1, ..., (P_i, ..., P_j)^{\circledast}, ..., P_m)\ prj\ Q$$

where $p \in Prop$, $P_1, ..., P_m$, P and Q are all well-formed PPTL formulas. $\bigcirc$ (*next*), *prj* (*projection*) and $prj^{\circledast}$ (*projection star*) 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 $false \overset{\text{def}}{=} P \wedge \neg P$. In addition, we have the following derived formulas.

$$\begin{array}{rclrcl}
empty & \overset{\text{def}}{=} & \neg \bigcirc true & more & \overset{\text{def}}{=} & \neg empty \\
\bigcirc^0 P & \overset{\text{def}}{=} & P & \bigcirc^n P & \overset{\text{def}}{=} & \bigcirc(\bigcirc^{n-1} P), n \geq 1 \\
len(0) & \overset{\text{def}}{=} & empty & len(n) & \overset{\text{def}}{=} & \bigcirc^n\ empty, n \geq 1 \\
skip & \overset{\text{def}}{=} & len(1) & \odot P & \overset{\text{def}}{=} & empty \vee \bigcirc P \\
P; Q & \overset{\text{def}}{=} & (P, Q)\ prj\ empty & \Diamond P & \overset{\text{def}}{=} & true; P \\
\Box P & \overset{\text{def}}{=} & \neg \Diamond \neg P & P^0 & \overset{\text{def}}{=} & empty \\
P^1 & \overset{\text{def}}{=} & P & P^n & \overset{\text{def}}{=} & (P^{(n)})\ prj\ empty, n \geq 1 \\
P^* & \overset{\text{def}}{=} & (P^{\circledast})\ prj\ empty & P^+ & \overset{\text{def}}{=} & (P^{\oplus})\ prj\ empty
\end{array}$$

$$\begin{array}{cl}
 & (P_1, ..., P_{i-1}, (P_i, ..., P_j)^{(0)}, P_{j+1}, ..., P_m)\ prj\ Q \\
\overset{\text{def}}{=} & (P_1, ..., P_{i-1}, P_{j+1}, ..., P_m)\ prj\ Q \\
 & (P_1, ..., P_{i-1}, (P_i, ..., P_j)^{(1)}, P_{j+1}, ..., P_m)\ prj\ Q \\
\overset{\text{def}}{=} & (P_1, ..., P_{i-1}, P_i, ...P_j, P_{j+1}, ..., P_m)\ prj\ Q \\
 & (P_1, ..., P_{i-1}, (P_i, ..., P_j)^{(n)}, P_{j+1}, ..., P_m)\ prj\ Q \\
\overset{\text{def}}{=} & (P_1, ..., P_{i-1}, P_i, ...P_j, (P_i, ..., P_j)^{n-1}, P_{j+1}, ..., P_m)\ prj\ Q, n \geq 1 \\
 & (P_1, ..., P_{i-1}, (P_i, ..., P_j)^{\oplus}, P_{j+1} ..., P_m)\ prj\ Q \\
\overset{\text{def}}{=} & (P_1, ..., P_{i-1}, P_i, ...P_j, (P_i, ..., P_j)^{\circledast}, P_{j+1}, ..., P_m)\ prj\ Q
\end{array}$$

where $\odot$ (weak next), $\Box$ (always), $\Diamond$ (sometimes), ; (chop), $prj^{\oplus}$ (projection plus), $*$ (chop star) and $+$ (chop plus) 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.

Semantics: Following the definition of Kripke's structure [2], 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 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 by $< s_0, ..., s_{|\sigma|} >$, 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 $< s_k, ..., s_{|\sigma|} >$. Further, the concatenation $(\cdot)$ of two intervals σ and σ' is defined as follows,

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

And the fusion of two intervals σ and σ' is also defined as below,

$$\sigma \circ \sigma' = \begin{cases} \sigma, \text{ if } |\sigma| = \omega \\ < s_0, ..., s_i, ... > \text{ if } \sigma =< s_0, ..., s_i >, \sigma' =< s_i, ... >, i \in N_0 \end{cases}$$

Moreover, $\sigma^{\cdot\omega}$ means infinitely many interval σ are concatenated, while $\sigma^{\circ\omega}$ denotes infinitely many σ are fused together.

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

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

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

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

We need also to generalize the notation of $\sigma \downarrow (r_1, ..., r_m)$ to allow r_i to be ω. For an interval $\sigma =< s_0, s_1, ..., s_{|\sigma|} >$ and $0 \leq r_1 \leq r_2 \leq ... \leq r_h \leq |\sigma|$ $(r_i \in N_\omega)$, we define $\sigma \downarrow () = \varepsilon$, $\sigma \downarrow (r_1, ..., r_h, \omega) = \sigma \downarrow (r_1, ..., r_h)$. This is convenient to define 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 $\mathcal{I} = (\sigma, k, j)$, where σ is an interval, k is an integer, and j an integer or ω such that $k \preceq j \leq |\sigma|$. We use the notation $(\sigma, k, j) \models P$ to denote that formula P is interpreted and satisfied over the subinterval $< s_k, ..., s_j >$ of σ with the current state being s_k. The satisfaction relation $(\models)$ is inductively defined as follows:

$\mathsf{I-prop}$ $\mathcal{I} \models p$ iff $s_k[p] = true$, for any given proposition p

$\mathsf{I-not}$ $\mathcal{I} \models \neg P$ iff $\mathcal{I} \not\models P$

$\mathsf{I-or}$ $\mathcal{I} \models P \vee Q$ iff $\mathcal{I} \models P$ or $\mathcal{I} \models Q$

$\mathsf{I-next}$ $\mathcal{I} \models \bigcirc P$ iff $k < j$ and $(\sigma, k+1, j) \models P$

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

$\mathsf{I-prj}^{\circledast}$ $\mathcal{I} \models (P_1, ..., (P_i, ..., P_j)^{\circledast}, ..., P_m)\ prj\ Q$ iff $\exists n \in N_0, \mathcal{I} \models (P_1, ..., (P_i, ..., P_j)^n, ..., P_m)\ prj\ Q$ or there exist integers $r_0 \leq r_1 \leq ... \leq r_i \leq ... \leq r_j \leq r_{2i} \leq ... \leq r_{2j} \leq r_{3i} ... \leq r_k \preceq \omega, lim_{k \to \omega} r_k = \omega$, such that $(\sigma, r_{l-1}, r_l) \models P_l, 0 < l < i, (\sigma, r_{l-1}, r_l) \models P_t, l \geq i, t = i + (l\ mod\ (j - i + 1))$, and $(\sigma', 0, |\sigma'|) \models Q$, where $\sigma' = \sigma \downarrow (r_0, ..., r_k, \omega)$, $l\ mod\ 1 = 0$.

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

3 B*ü*chi Automata with Stutter

Definition 1. A B*ü*chi automaton is a tuple $B = (Q, \Sigma, I, \delta, F)$, where,

- $Q = \{q_0, q_1, ..., q_n\}$ is a finite, non-empty set of locations;
- $\Sigma = \{a_0, a_1, ..., a_m\}$ is a finite, non-empty set of symbols, namely alphabet;
- $I \subseteq Q$ is a non-empty set of initial locations;
- $\delta \subseteq Q \times \Sigma \times Q$ is a transition function;
- $F \subseteq Q$ is a set of accepting locations.

An infinite word w over Σ is an infinite sequence $w = a_0a_1...$ of symbols, $a_i \in \Sigma$. A run of B over an infinite word $w = a_0a_1...$ is an infinite sequence $\rho = q_0q_1...$ of locations $q_i \in Q$ such that $q_0 \in I$ and $(q_i, a_i, q_{i+1}) \in \delta$ holds for all $i \in N_0$. In this case, we call w the word associated with ρ, and ρ the run associated with w. The run ρ is an accepting run iff there exists some $q \in F$ such that $q_i = q$ holds for infinitely many $i \in N_0$. The language $L(B)$ accepted by a B*ü*chi automaton B is the set of infinite words for which there exists some accepting run ρ of B.

Similar to the approach adopted in SPIN [10] for modeling finite behaviors of a system with a B*ü*chi automaton, the stuttering rule is adopted so that the classic notion of acceptance for finite runs (thus words) would be included as a special case in B*ü*chi automata. To apply the rule, we extend the alphabet Σ with a fixed predefined null-label ϵ, representing a no-op operation that is always executable and has no effect. For a B*ü*chi automaton B, the stutter extension of finite run ρ with final state q_n is the ω-run ρ such that $(q_n, \epsilon, q_n)^\omega$ is the suffix of ρ. The final state of the run can be thought to repeat null action ϵ infinitely. It follows that such a run would satisfy the rules for B*ü*chi acceptance if and only if the original final location q_n is in the set F of accepting locations. This means that it indeed generalizes the classical definition of the finite acceptance. In what follows, we denote B*ü*chi automata with the stutter extension, simply as stutter-B*ü*chi automata (SBA for short).

4 Extended Regular Expression

Corresponding to the stutter-B*ü*chi automata, we define a kind of extended ω-regular expression (ERE) which is capable of defining both finite and infinite strings. Let $\Upsilon = \{r_1, ..., r_n\}$ be a finite set of symbols, namely alphabet. The extended ω-regular expressions are defined as follows,

$$\text{ERE} \qquad R ::= \emptyset \mid \epsilon \mid r \mid R + R \mid R \bullet R \mid R^\omega \mid R^*$$

where $r \in \Upsilon$, ϵ denotes an empty string; $+$, $\bullet$ and $*$ are union, concatenation and Kleene (star) closure respectively; R^ω means infinitely many R are concatenated. In what follows, we use ERE to denote the set of extended regular expressions.

Before defining the language expressed by the extended regular expressions, we first introduce strings and operations on strings. A string is a finite or infinite sequence of symbols, $a_0a_1...a_i...$, where each a_i is chosen from the alphabet Υ. The length of a finite string w, denoted by $|w|$, is the number of the symbols in

w while the length of an infinite string is ω. For two strings w and w', $w \bullet w'$, w^* and w^ω are defined as follows,

$$w \bullet w' = \begin{cases} w, \text{ if } |w| = \omega \\ a_0 ... a_i a_{i+1} ..., \text{ if } w = a_0 ... a_i, \text{ and } w' = a_{i+1} ... \end{cases}$$

$$w^\omega = \begin{cases} w, \text{ if } |w| = \omega \\ \underbrace{a_0 ... a_i ... a_0 ... a_i ...}_{\omega \text{ times}}, \text{ if } w \text{ is finite and } w = a_0 ... a_i \end{cases}$$

$$w^* = \begin{cases} w, \text{ if } |w| = \omega \\ \exists n \in N_\omega, \underbrace{a_0 ... a_i ... a_0 ... a_i}_{n \text{ times}}, \text{ if } w \text{ is finite and } w = a_0 ... a_i \end{cases}$$

Further, if W and W' are two sets of strings. Then $W \bullet W'$, W^ω and W^* are defined as follows,

$$\begin{aligned} W \bullet W' &= \{w \bullet w' \mid w \in W \text{ and } w' \in W'\} \\ W^\omega &= \{w^\omega \mid w \in W\} \\ W^* &= \{w^* \mid w \in W\} \end{aligned}$$

Accordingly, the language $L(R)$ expressed by extended regular expression R is given by,

Lr_1	$L(\emptyset) = \emptyset$	Lr_2	$L(r) = \{r\}$
Lr_3	$L(\epsilon) = \{\epsilon\}$	Lr_4	$L(R{+}R) = L(R) \cup L(R)$
Lr_5	$L(R \bullet R) = L(R) \bullet L(R)$	Lr_6	$L(R^\omega) = L(R)^\omega$
Lr_7	$L(R^*) = L(R)^*$		

For a string w, if $w \in L(R)$, w is called a word of expression R.

For convenience, we use PPTL* to denote the set of all PPTL with star formulas, SBA the set of all Stutter B*ü*chi Automata, and ERE the set of all Extended ω-Regular Expressions. Further, the language classes determined by PPTL*, SBA and ERE are represented by L(PPTL*), L(SBA) and L(ERE) respectively. That is,

$$\begin{aligned} L(PPTL^*) &= \{L(P) | P \in PPTL^*\} \\ L(SBA) &= \{L(B) | B \in SBA\} \\ L(ERE) &= \{L(R) | R \in ERE\} \end{aligned}$$

5 Relationship between PPTL*, ERE and SBA

Even though the extended ω-regular expressions, PPTL with star formulas, and stutter-Büchi automata describe languages fundamentally in different ways, however, it turns out that they represent exactly the same class of languages, named the "full regular languages" as concluded in Theorem 1.

Theorem 1. PPTL with star formulas, extended regular expressions and stutter B*ü*chi automata have the same expressiveness. □

$$\begin{tikzcd} & L(PPTL^*) \arrow[dl] & \\ L(SBA) \arrow[rr] & & L(ERE) \arrow[ul] \end{tikzcd}$$

Fig. 1. The relationship between three language classes

In order to prove this theorem, we will show the following facts: (1) For any P, there is an SBA B such that $L(B) = L(P)$; (2) For any SBA B, there is an ERE R such that $L(R) = L(B)$; (3) For any ERE R, there is a PPTL with star formula P such that $L(P) = L(R)$. The relationship is depicted in Fig.1, where an arc from language class X to Y means that each language in X can also be defined in Y. This convinces us that three language classes are equivalence.

For the purpose of transformations between PPTL*, SBA and ERE, we define the set Q_p of atomic propositions appearing in PPTL with star formula Q, $|Q_p| = l$, and further need define alphabets Σ and Υ for SBA and ERE. To do so, We first define sets A_i, $1 \leq i \leq l$, as follows,

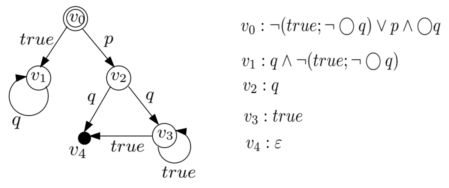
$$A_i = \{\{\dot{q}_{j_1}, ..., \dot{q}_{j_i}\} \mid q_{j_k} \in Q_p, \dot{q}_{j_k} \text{ denotes } q_{j_k} \text{ or } \neg q_{j_k}, 1 \leq k \leq i\}$$

Then, $\Sigma = \bigcup_{i=1}^{l} A_i \cup \{true\} \cup \{\epsilon\}$, and $\Upsilon = \bigcup_{i=1}^{l} A_i \cup \{true\}$. For instance, if $Q_p = \{p_1, p_2, p_3\}$, it is obtained that, $A_1 = \{\{\dot{p_1}\}, \{\dot{p_2}\}, \{\dot{p_3}\}\}$, $A_2 = \{\{\dot{p_1}, \dot{p_2}\}, \{\dot{p_1}, \dot{p_3}\}, \{\dot{p_2}, \dot{p_3}\}\}$, $A_3 = \{\{\dot{p_1}, \dot{p_2}, \dot{p_3}\}\}$. So, we have, $\Sigma = \bigcup_{i=1}^{3} A_i \cup \{true\} \cup \{\epsilon\} = \{\{\dot{p_1}\}, \{\dot{p_2}\}, \{\dot{p_3}\}, \{\dot{p_1}, \dot{p_2}\}, \{\dot{p_1}, \dot{p_3}\}, \{\dot{p_2}, \dot{p_3}\}, \{\dot{p_1}, \dot{p_2}, \dot{p_3}\}, true, \epsilon\}$, $\Upsilon = \bigcup_{i=1}^{3} A_i \cup \{true\} = \{\{\dot{p_1}\}, \{\dot{p_2}\}, \{\dot{p_3}\}, \{\dot{p_1}, \dot{p_2}\}, \{\dot{p_1}, \dot{p_3}\}, \{\dot{p_2}, \dot{p_3}\}, \{\dot{p_1}, \dot{p_2}, \dot{p_3}\}, true\}$. Obviously, for each $r \in \Upsilon$, r is a set of atomic propositions or their negations, denoted by $true$ or $\{\dot{q}_i, ..., \dot{q}_j\}$, where $1 \leq i \leq j \leq l$.

5.1 From PPTL with Star Formulas to SBAs

For PPTL with star formulas, their normal forms are the same as for PPTL formulas [6,7]. In [22], an algorithm was given for transforming a PPTL with star formula to its normal form. Further, based on the normal form, labeled normal form graph (LNFG) for PPTL with star formulas are constructed to precisely characterize the models of PPTL with star formulas. Also an algorithm was given to construct the LNFG of a PPTL with star formula [22]. The details about normal forms and LNFGs can be found in [7,22]. Here we focus on how to transform an LNFG G to an SBA B (corresponding SBA of G). For the clear presentation of the transformation, LNFGs are briefly introduced first.

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 inductively be constructed by algorithm *LNFG* composed in [7]. Actually, in an LNFG, a node $v \in V$ denotes a PPTL with star formula and initial node is P itself while an edge from node v_i to v_j is a tuple (v_i, Q_e, v_j) where v_i and v_j are PPTL with star formulas and $Q_e \equiv \bigwedge_i \dot{q}_i$, q_i is an atomic proposition, $\dot{q}_i$ denotes q_i or $\neg q_i$. The following is an example of LNFG.



v_0
true
p
v_1
v_2
q
q
q
v_3
v_4
true
true
$v_0 : \neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$
$v_1 : q \wedge \neg(true; \neg \bigcirc q)$
$v_2 : q$
$v_3 : true$
$v_4 : \varepsilon$

Fig. 2. LNFG of formula $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$

Example 1. The LNFG of formula $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$.
As shown in Fig.2, the LNFG of formula $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$ is $G=\{V, E, v_0, V_f\}$, where $V = \{v_0, v_1, v_2, v_3, v_4\}$; $E = \{(v_0, true, v_1), (v_0, p, v_2), (v_1, q, v_1), (v_2, q, v_4), (v_2, q, v_3), (v_3, true, v_3), (v_3, true, v_4)\}$; the root node is v_0; and $V_f = \emptyset$. □

Factually, an LNFG contains all the information of the corresponding SBA. The set of nodes is in fact the set of locations in the corresponding SBA; each edge (v_i, Q_e, v_j) forms a transition; there exists only one initial location, the root node; the set of accepting locations consists of ε node and the nodes which can appear in infinite paths for infinitely many times. Given an LNFG $G = (V, E, v_0, V_f)$ of formula P, an SBA of formula P, $B = (Q, \Sigma, I, \delta, F)$, over an alphabet Σ can be constructed as follows.

• Sets of the locations Q and the initial locations I: $Q = V$, and $I = \{v_0\}$.
• Transition δ: Let $\dot{q_k}$ be an atomic proposition or its negation. 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,

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

For each $e_i = (v_i, Q_e, v_{i+1}) \in E$, there exists $v_{i+1} \in \delta(v_i, atom(Q_e))$. For node ε, $\delta(\varepsilon, \epsilon) = \{\varepsilon\}$.
• Accepting locations F: We have proved in [6,7] that infinite paths in an LNFG precisely characterize the infinite models of the corresponding formula. In fact, there exists an infinite path if and only if there are some nodes appearing in the path for infinitely many times. Therefore, the nodes appearing in infinite paths for infinitely many times are defined as the accepting locations in the SBA. In addition, by employing the stutter extension rule, ε node is also an accepting location.

Lemma 1. For any PPTL with star formula P, there is an SBA B such that $L(B) = L(P)$. □

Formally, algorithm Lnfg-Sba is useful for transforming an LNFG to an SBA. Also Example 2 is given to show how the algorithm works.

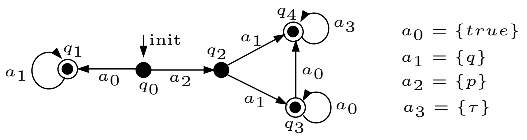


Fig. 3. Stutter-B*ü*chi automaton of formula $P \equiv \neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$

Example 2. Constructing the SBA, B=$(Q, \Sigma, I, \delta, F)$, from the LNFG in Example 1.

As depicted in Fig.3, the set of locations, Q=$\{q_0, q_1, q_2, q_3, q_4\}$, comes from V directly. The set of initial locations I=$\{q_0\}$ is root node v_0 in G. The set of the accepting locations F=$\{q_1, q_3, q_4\}$ consists of nodes v_1, v_3 appearing in loops and ε node in V. The transitions, $\delta(q_0, a_0)$=$\{q_1\}$, $\delta(q_0, a_2)$=$\{q_2\}$, $\delta(q_1, a_1)$=$\{q_1\}$, $\delta(q_2, a_1)$=$\{q_3, q_4\}$, $\delta(q_3, a_0)$=$\{q_3, q_4\}$, $\delta(q_4, a_3)$=$\{q_4\}$ are formalized according to the edges in E. □

```
Function LNFG-SBA($G$)
/* precondition: $G = (V, E, v_0, V_f)$ is the LNFG of PPTL formula $P$*/
/* postcondition: LNFG-SBA($G$) computes an SBA $B = (Q, \Sigma, I, \delta, F)$ from G*/
begin function
    $Q = \emptyset$; $F = \phi$; $I = \emptyset$;
    for each node $v_i \in V$,
        add a state $q_i$ to $Q$, $Q = Q \cup \{q_i\}$;
        if $v_i$ is $\varepsilon$, $F = F \cup \{q_i\}$; $\delta(q_i, \epsilon) = \{q_i\}$;
        else if $v_i$ appears in loops and $v_i \notin V_f$ , $F = F \cup \{q_i\}$;
    end for
    if $q_0 \in Q$, $I = I \cup \{q_0\}$;
    for each edge $e = (v_i, P_e, v_j) \in E$,
        $q_j \in \delta(q_i, atom(P_e))$;
    end for
    return $B = (Q, \Sigma, I, \delta, F)$
end function
```

5.2 From SBAs to EREs

For the proof of the language $L(A)$ of any finite state automaton A being regular [24], Arden's rule [11] plays an important role.

Lemma 2. (Arden's Rule) For any sets of strings S and T, the equation X=$S \bullet X + T$ has $X = S^* \bullet T$ as a solution. Moreover, this solution is unique if $\epsilon \notin S$. □

From now on we shall often drop the concatenation symbol $\bullet$, writing SX for $S \bullet X$ etc. In the following, we show how Arden's rule is used to equivalently transform an SBA to an ERE.

Given a stutter-Büchi automaton B with $Q = \{q_0, ..., q_n\}$ and the starting location q_0. For $1 \leq i \leq n$, let X_i denote the ERE where $L(X_i)$ equals to the set of strings accepted by the sub-automaton of B starting at location q_i; thus $L(B) = L(X_0)$. We can write an equation for each X_i in terms of the languages defined by its successor locations. For example, for the stutter-Büchi automaton B in Example 2, we have,

$$\begin{array}{llll} (0) & X_0 = a_0X_1 + a_2X_2 & (1) & X_1 = a_1X_1 + a_1^\omega \\ (2) & X_2 = a_1X_4 + a_1X_3 & (3) & X_3 = a_0X_3 + a_0X_4 + a_0^\omega \\ (4) & X_4 = a_3X_4 + a_3^\omega & & \end{array}$$

Note that X_1, X_3 and X_4 contains a_1^ω, a_0^ω and a_3^ω respectively because q_1, q_3 and q_4 are accepting states with self-loops[2]. Now we use Arden's rule to solve the equations. First, for (4), since a_3 is ϵ,

$$X_4 = a_3X_4 + a_3^\omega = a_3^*a_3^\omega = a_3^\omega = \epsilon$$

Replacing X_4 in (3),

$$X_3 = a_0X_3 + a_0X_4 + a_0^\omega = a_0X_3 + a_0 + a_0^\omega = a_0^*a_0 + a_0^*a_0^\omega = a_0^*a_0 = true^*true$$

Replacing X_3 and X_4 in (2),

$$X_2 = a_1X_4 + a_1X_3 = \{q\} + \{q\}true^*true$$

For (1),

$$X_1 = a_1X_1 + a_1^\omega = a_1^*a_1^\omega = a_1^\omega = \{q\}^\omega$$

Finally, replacing X_1 and X_2 in (0), we have,

$$\begin{aligned} X_0 &= a_0X_1 + a_2X_2 = a_0\{q\}^\omega + a_2(\{q\} + \{q\}true^*true) \\ &= true\{q\}^\omega + \{p\}\{q\} + \{p\}\{q\}true^*true \end{aligned}$$

Lemma 3. For any SBA B, there is an ERE R such that $L(R) = L(B)$. □

5.3 From EREs to PPTL with Star Formulas

Let Γ be the set of all models of PPTL with star. For any extended regular expression $R \in$ ERE, we can construct a PPTL with star formula F_R such that, (1) for any model $\sigma \in \Gamma$, if $\sigma \models F_R$, then $\Omega(\sigma) \in L(R)$; and (2) for any word $w \in L(R)$, there exists $\sigma \in \Gamma$, $\sigma \models F_R$ and $\Omega(\sigma) = w$. The mapping function $\Omega : \Gamma \rightarrow \Upsilon^*$ from models of PPTL formulas to words of extended regular expression is defined as follows,

$$\Omega(\sigma) = \begin{cases} \epsilon, & \text{if } |\sigma| = 0 \\ A(s_0)...A(s_{j-1}) & \text{if } \sigma \text{ is finite and } \sigma =< s_0, ..., s_j >, j \geq 1 \\ A(s_0)...A(s_j)... & \text{if } \sigma \text{ is infinite and } \sigma =< s_0, ..., s_j, ... > \end{cases}$$

where $A(s_i)$ denotes *true*, or the set of propositions and their negations holding at state s_i. It is not difficult to prove that $\Omega(\sigma_1 \circ \sigma_2) = \Omega(\sigma_1) \bullet \Omega(\sigma_2)$, $\Omega(\sigma^{\circ\omega}) = \Omega(\sigma)^\omega$ and $\Omega(\sigma^{\circ *}) = \Omega(\sigma)^*$. F_R is constructed inductively on the structure of R.

$$\begin{aligned} F_\emptyset &\overset{\text{def}}{=} false \\ F_\epsilon &\overset{\text{def}}{=} empty \\ F_r &\overset{\text{def}}{=} \begin{cases} \dot{p_i} \wedge ... \wedge \dot{p_j} \wedge skip, & \text{if } r = \{\dot{p_i}, ..., \dot{p_j}\}, 1 \leq i \leq j \leq l \\ true \wedge skip, & \text{if } r = true \end{cases} \end{aligned}$$

where $r \in \Upsilon$. Inductively, if R_1 and R_2 are extended regular expressions, then

$$\begin{aligned} F_{R_1+R_2} &\overset{\text{def}}{=} F_{R_1} \vee F_{R_2} \\ F_{R_1 \bullet R_2} &\overset{\text{def}}{=} F_{R_1}; F_{R_2} \\ F_{R^\omega} &\overset{\text{def}}{=} F_R^* \wedge \Box more \\ F_{R_1^*} &\overset{\text{def}}{=} F_{R_1}^* \end{aligned}$$

[2] For finite state automata, X_i contains ϵ if q_i is accepted.

Now we need to prove that, for any $R \in$ ERE and $\sigma \in \Gamma$, if $\sigma \models F_R$, then $\Omega(\sigma) \in L(R)$; for any w, if $w \in L(R)$, then there exists $\sigma \in \Gamma$ such that $\Omega(\sigma) = w$ and $\sigma \models F_R$.

Lemma 4. For any ERE R, there is a PPTL with star formula P such that $L(P) = L(R)$. $\Box$

Example 3. Constructing PPTL with star formula from the extended regular expression, $true\{q\}^\omega + \{p\}\{q\} + \{p\}\{q\}true^*true$ obtained in Example 2.

$$\begin{aligned}
& F_{true\{q\}^\omega+\{p\}\{q\}+\{p\}\{q\}true^*true} \\
\equiv\ & F_{true\{q\}^\omega} \vee F_{\{p\}\{q\}} \vee F_{\{p\}\{q\}true^*true} \\
\equiv\ & F_{true}; F_{\{q\}^\omega} \vee F_{\{p\}}; F_{\{q\}} \vee F_{\{p\}}; F_{\{q\}}; F_{true^*}; F_{true} \\
\equiv\ & true \wedge skip; F_{\{q\}^*} \wedge \Box more \vee p \wedge skip; q \wedge skip \vee p \wedge skip; q \wedge skip; \\
& (true \wedge skip)^*; true \wedge skip \\
\equiv\ & skip; (q \wedge skip)^* \wedge \Box more \vee p \wedge skip; q \wedge skip \vee p \wedge skip; q \wedge skip; \\
& skip^*; skip
\end{aligned}$$
$\Box$

6 Conclusions

Further, it is readily to prove the following useful conclusions concerning characters of fragments of PPTL with star. To avoid abuse of notations, we use an expression like $\mathsf{L}(\mathsf{next}, \mathsf{chop})$ to refer to the specific fragment of PPTL with star with temporal operators *next*, *chop* and the basic connections in the typical propositional logic.
1 $\mathsf{L}(\mathsf{chop})$ has the same expressiveness as star-free regular expressions without ϵ.
2 $\mathsf{L}(\mathsf{next}, \mathsf{chop})$ has the same expressiveness as star-free regular expressions.
3 $\mathsf{L}(\mathsf{next}, \mathsf{prj})$ has the same expressiveness as regular expressions without ω.
4 $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*)$ has the same expressiveness as full regular expressions.

In this paper, we have proved that the expressiveness of PPTL with star is the same as the full regular expressions. Also, the proof itself provides approaches to translate a PPTL with star formula to an equivalent Buchi automaton, a Buchi automaton to an equivalent extended ω-regular expression, and an extended ω-regular expression to a PPTL with star formula. This enables us to specify and verify concurrent systems by compositional approach with PPTL with star. Further, we have developed a model checker based on SPIN for PPTL with star. Therefore, any systems with regular properties can be automatically verified within SPIN using PPTL with star.

References

1. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–67.2 IEEE, New York (1977)
2. Kripke, S.A.: Semantical analysis of modal logic I: normal propositional calculi. Z. Math. Logik Grund. Math. 9, 67–96 (1963)
3. Rosner, R., Pnueli, A.: A choppy logic. In: First Annual IEEE Symposium on Logic In Computer Science. LICS, pp. 306–314 (1986)

4. Moszkowski, B.: Reasoning about digital circuits. Ph.D Thesis, Department of Computer Science, Stanford University. TRSTAN-CS-83-970 (1983)
5. Duan, Z.: An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming. PhD thesis, University of Newcastle Upon Tyne (May 1996)
6. Duan, Z., Tian, C.: Decidability of Propositional Projection Temporal Logic with Infinite Models. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 521–532. Springer, Heidelberg (2007)
7. Duan, Z., Tian, C., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. Acta Informatica 45, 43–78 (2008)
8. Tian, C., Duan, Z.: Model Checking Propositional Projection Temporal Logic Based on SPIN. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 246–265. Springer, Heidelberg (2007)
9. Wolper, P.L.: Temporal logic can be more expressive. Information and Control 56, 72–99 (1983)
10. Holzmann, G.J.: The Model Checker Spin. IEEE Trans. on Software Engineering 23(5), 279–295 (1997)
11. Arden, D.: Delayed-logic and finite-state machines. In: Theory of Computing Machine Design, Univ. of Michigan Press, pp. 1–35 (1960)
12. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. In: POPL 1980: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 163–173. ACM Press, New York (1980)
13. Sistla, A.P.: Theoretical issues in the design and verification of distributed systems. PhD thesis, Harvard University (1983)
14. Vardi, M.Y., Wolper, P.: Yet another process logic. In: Clarke, E., Kozen, D. (eds.) Logic of Programs 1983. LNCS, vol. 164, pp. 501–512. Springer, Heidelberg (1984)
15. Vardi, M.Y.: A temporal fixpoint calculus. In: POPL 1988, pp. 250–259 (1988)
16. McNaughton, R., Papert, S.A.: Counter-Free Automata (M.I.T research monograph no.65). The MIT Press, Cambridge (1971)
17. Barringer, H., Kuiper, R., Pnueli, A.: Now You May Compose Temporal Logic Specifications. In: Proc. 16th STOC, pp. 51–63 (1984)
18. Barringer, H., Kuiper, R., Pnueli, A.: The Compositional Temporal Approach to CSP-like Language. In: Proc.IFIP Conference, The Role of Abstract Models in Information Processing (January 1985)
19. Nguyen, V., Demers, A., Gries, D., Owicki, S.: Logic of Programs 1985. LNCS, vol. 193, pp. 237–254. Springer, Heidelberg (1985)
20. Nguyen, V., Gries, D., Owicki, S.: A Model and Temporal Proof System for Networks of Processes. In: Proc. 12th POPL, pp. 121–131 (1985)
21. Harel, D., Peleg, D.: Process Logic with Regular Formulas. Theoretical Computer Science 38, 307–322 (1985)
22. Tian, C., Duan, Z.: Complexity of Propositional Projection Temporal Logic with Star. Technical Report No.25, Institute of computing Theory and Technology, Xidian University, Xian P.R.China (2007)
23. Arden, D.: Delayed-logic and finite-state machines. In: Theory of Computing Machine Design, pp. 1–35. Univ. of Michigan Press (1960)
24. Milner, R.: Communicating and Mobile System: The -Calculus. Cambridge University Press, Cambridge (1999)