Theoretical Computer Science 412 (2011) 1729–1744

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Expressiveness of propositional projection temporal logic with star ☆

Cong Tian, Zhenhua Duan *

ICTT and ISN Laboratory, Xidian University, Xi'an, 710071, PR China

ARTICLE INFO

Keywords:
Temporal logic
Expressiveness
Automata theory
Regular expressions
Verification

ABSTRACT

This paper investigates the expressiveness of Propositional Projection Temporal Logic with Star (PPTL*). To this end, Büchi automata and ω-regular expressions are first extended as Stutter Büchi Automata (SBA) and Extended Regular Expressions (ERE) to include both finite and infinite strings. Further, by equivalent transformations among PPTL* formulas, SBAs and EREs, PPTL* is proved to represent exactly the full regular language. Moreover, some fragments of PPTL* are characterized, and finally, PPTL* and its fragments are classified into five different language classes.

1. Introduction

Temporal logic is a useful formalism for describing sequences of transitions between states in reactive systems. In the past thirty years, many kinds of temporal logics were 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) [1] and its variations. In the propositional framework, Propositional LTL (PLTL) has been proved to have the expressiveness of star-free regular expressions [14,18]. Considering the expressive limitation of PLTL, extensions such as Quantified Linear time Temporal Logic (QLTL) [15], Extended Temporal Logic (ETL) [11,16] and linear mu-calculus (νTL) [17] etc, were introduced to PLTL for the expressiveness of full regular language. Nevertheless, results [19–22] have shown that temporal logic needs some further extensions in order to support a compositional approach for the specification and verification of concurrent systems. These extensions should enable modular and compositional reasoning about loops and sequential composition as well as concurrent ones. Therefore, kinds of extensions were proposed. Prominently, one of the important extensions is the addition of the *chop* operator. The work in [11] 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.

Interval Temporal Logic (ITL) [4] is an easily understood temporal logic with *next, chop* and projection operator, *proj*. In two characteristic operators, *chop* implements a form of sequential composition while *proj* yields repetitive behaviors. ITL without *projection* has the similar expressiveness as Rosner and Pnueli's choppy logic [3]. Further, addition of the *proj* operator will bring more powerful expressiveness, since repetitive behaviors are allowed. However, no systematic proofs have been given in this aspect. Projection Temporal Logic (PTL) [6] is an extension of ITL. It extends ITL to include infinite models and a new projection construct, $(P_1, \ldots, P_m)$ *prj* Q, which is much more flexible than the original one. Further, in the propositional case,[1] the *projection* construct can be extended to *projection star*, $(P_1, \ldots, (P_i, \ldots, P_j)^{\circledast}, \ldots, P_m)$ *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 loss of decidability [29].

☆ This research is supported by the NSFC Grant No. 61003078, 91018010, 60433010, 60873018 and 60910004, National Program on Key Basic Research Project of China (973 Program) Grant No. 2010CB328102 and SRFDP Grant 200807010012.

* Corresponding author.
E-mail address: zhhduan@mail.xidian.edu.cn (Z. Duan).

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

doi:10.1016/j.tcs.2010.12.047

Within PTL, plenty of logic laws have been formalized and proved [6,7], and a decision procedure for checking the satisfiability of Propositional Projection Temporal Logic (PPTL) formulas is given in [8]. Based on the decision procedure, a model checking approach based on SPIN for this logic is proposed [10]. Further, in [29], *projection star* is introduced to PPTL, and the satisfiability for PPTL with star (PPTL*) formulas is proved to be still decidable. Moreover, the complexity for the satisfiability of PPTL* formulas is proved to be non-elementary by means of reducing the emptiness problem of star-free expressions [28] to the problem of the satisfiability of PPTL* formulas. Intuitively, PPTL* is powerful enough to express the full regular expression. If so, by employing PPTL* formulas as the property specification language, the verification of concurrent systems with the model checker SPIN can be done completely automatically. This will overcome the error-prone hand-writing of a never claim in the original SPIN since some properties cannot be specified by a PLTL formula. Further, since PPTL* can subsume the *chop* construct, compositional approaches for the specification and verification of concurrent systems with SPIN are allowed.

The complexity of model checking PPTL* is non-elementary for the non-elementary decidability of the logics with the *chop* operator. However, we can prove that with the *chop* construct, logics will be non-elementary succincter. That is to express a *chop* formula with the existing operators, such as *next* and *until* in PLTL, a non-elementary longer formula is needed. Thus, actually, model checking PPTL* shares the same time complexity with PLTL. Therefore, we are motivated to systematically investigate the expressiveness of PPTL* and the characterizations of its fragments. To this end, Stutter Büchi Automata (SBA) and Extended Regular Expressions (ERE) are introduced. And the underlying logic is proved to represent exactly the full regular language by three transforming procedures among PPTL*, SBA, and ERE. Subsequently, fragments of PPTL* are defined and characterized, and finally, PPTL* and its fragments are classified into five different language classes. In addition, the expressiveness of Propositional ITL (PITL) is also investigated.

The rest of the paper is organized as follows. The syntax and semantics of PPTL* are briefly introduced in the next section. Sections 3 and 4 presents the definitions of stutter Büchi automata and extended regular expressions respectively. Section 5 is devoted to proving the expressive power of PPTL* by equivalently transformations among PPTL* formulas, SBAs, and EREs. In Section 6, fragments of PPTL* are defined and characterized, and eventually, the full logic and its fragments are classified into five language classes. Finally, conclusions are drawn in Section 7.

2. Propositional projection temporal logic with star

Our underlying logic is Propositional Projection Temporal Logic with Star (PPTL*). It extends PPTL to include *projection star*. It is also an extension of PITL.

2.1. Syntax

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

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

where $p \in Prop$, $P_1, \ldots, P_m$, 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, $true \stackrel{\text{def}}{=} P \vee \neg P$ and $false \stackrel{\text{def}}{=} P \wedge \neg P$. In addition, we have the derived formulas as shown in Table 1. 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; *halt*(P) is true over an interval if and only if P is true at the final state; *fin*(P) is true as long as P is true at the final state; and *keep*(P) is true if P is true at every state ignoring the final one.

2.2. 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 $\langle s_0, \ldots, s_{|\sigma|}\rangle$, where $s_{|\sigma|}$ is undefined if σ is infinite. So, $\langle s_0, \ldots\rangle$ is often used to present an infinite interval. With such a notation, $\sigma_{(i..j)}$ $(0 \leq i \preceq j \leq |\sigma|)$ denotes the sub-interval $\langle s_i, \ldots, s_j\rangle$ and $\sigma^{(k)}$ $(0 \leq k \preceq |\sigma|)$ denotes $\langle s_k, \ldots, s_{|\sigma|}\rangle$. Further, the concatenation $(\cdot)$ of two intervals σ and σ' is defined as follows,

$$\sigma \cdot \sigma' = \begin{cases} \sigma, & \text{if } |\sigma| = \omega \\ \langle s_0, \ldots, s_i, s_{i+1}, \ldots\rangle & \text{if } \sigma = \langle s_0, \ldots, s_i\rangle, \sigma' = \langle s_{i+1}, \ldots\rangle, i \in N_0 \end{cases}$$

Table 1
Derived formulas.

Empty	$empty$	$\stackrel{\text{def}}{=}$	$\neg \bigcirc true$
More	$more$	$\stackrel{\text{def}}{=}$	$\neg empty$
Next-0	$\bigcirc^0 P$	$\stackrel{\text{def}}{=}$	P
Next-n	$\bigcirc^n P$	$\stackrel{\text{def}}{=}$	$\bigcirc(\bigcirc^{n-1}P),\ n \geq 1$
Len-0	$len(0)$	$\stackrel{\text{def}}{=}$	$empty$
Len-n	$len(n)$	$\stackrel{\text{def}}{=}$	$\bigcirc^n\ empty,\ n \geq 1$
Skip	$skip$	$\stackrel{\text{def}}{=}$	$len(1)$
W-Next	$\odot P$	$\stackrel{\text{def}}{=}$	$empty \vee \bigcirc P$
Chop	$P; Q$	$\stackrel{\text{def}}{=}$	$(P, Q)\ prj\ empty$
SomeTimes	$\Diamond P$	$\stackrel{\text{def}}{=}$	$true; P$
Always	$\Box P$	$\stackrel{\text{def}}{=}$	$\neg\Diamond\neg P$
Chop-0	P^0	$\stackrel{\text{def}}{=}$	$empty$
Chop-n	P^n	$\stackrel{\text{def}}{=}$	$P^{n-1}; P,\ n \geq 1$
Keep	$keep(P)$	$\stackrel{\text{def}}{=}$	$\Box(\neg empty \rightarrow P)$
Halt	$halt(P)$	$\stackrel{\text{def}}{=}$	$\Box(empty \leftrightarrow P)$
Fin	$fin(P)$	$\stackrel{\text{def}}{=}$	$\Box(empty \rightarrow P)$
Chop-$*$	P^*	$\stackrel{\text{def}}{=}$	$(P^{\circledast})\ prj\ empty$
Chop-$+$	P^+	$\stackrel{\text{def}}{=}$	$(P^{\oplus})\ prj\ empty$
Prj-0	$(P_1, \ldots, P_{i-1}, (P_i, \ldots, P_j)^{(0)}, P_{j+1}, \ldots, P_m)\ prj\ Q$	$\stackrel{\text{def}}{=}$	$(P_1, \ldots, P_{i-1}, P_{j+1}, \ldots, P_m)\ prj\ Q$
Prj-n	$(P_1, \ldots, P_{i-1}, (P_i, \ldots, P_j)^{(n)}, P_{j+1}, \ldots, P_m)\ prj\ Q$	$\stackrel{\text{def}}{=}$	$(P_1, \ldots, P_{i-1}, P_i, \ldots P_j, (P_i, \ldots, P_j)^{n-1}, P_{j+1}, \ldots, P_m)\ prj\ Q,\ n \geq 1$
Prj-$\oplus$	$(P_1, \ldots, P_{i-1}, (P_i, \ldots, P_j)^{\oplus}, P_{j+1} \ldots, P_m)\ prj\ Q$	$\stackrel{\text{def}}{=}$	$(P_1, \ldots, P_{i-1}, P_i, \ldots P_j, (P_i, \ldots, P_j)^{\circledast}, P_{j+1}, \ldots, P_m)\ prj\ Q$

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

$$\sigma \circ \sigma' = \begin{cases} \sigma, & \text{if } |\sigma| = \omega \\ \langle s_0, s_1 \ldots, s_i, s_{i+1}, \ldots \rangle, & \text{if } \sigma = \langle s_0, s_1 \ldots, s_i \rangle \text{ and } \sigma' = \langle s_i, s_{i+1} \ldots \rangle \end{cases}$$

Moreover, $\sigma^{\cdot\omega}$ means infinitely many copies of interval σ are concatenated, while $\sigma^{\circ\omega}$ denotes that infinitely many copies of σ are fused together. In particular, $\sigma \circ \sigma$ requires $s_0 = s_i$ if $\sigma = \langle s_0, \ldots, s_i \rangle$.

Let $\sigma = \langle s_0, s_1, \ldots, s_{|\sigma|} \rangle$ 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)

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

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,

$$\langle s_0, s_1, s_2, s_3, s_4 \rangle \downarrow (0, 0, 2, 2, 2, 3) = \langle s_0, s_2, s_3 \rangle$$

This projected interval is shown in Fig. 1 (1). We need also to generalize the notation of $\sigma \downarrow (r_1, \ldots, r_h)$ to allow r_h to be ω. For an interval $\sigma = \langle s_0, s_1, \ldots, s_{|\sigma|} \rangle$ and $0 \leq r_1 \leq r_2 \leq \ldots \leq r_h \leq |\sigma|$, we define

$$\sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) = \sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) \quad \text{if } r_h \text{ is not } \omega$$
$$\sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) = \sigma \downarrow (r_1, \ldots, r_{h-1}) \quad \text{if } r_h \text{ is } \omega$$

For instance,

$$\langle s_0, \ldots \rangle \downarrow (0, 1, 3, 4, \omega) = \langle s_0, s_1, s_3, s_4 \rangle$$

This projected interval is shown in Fig. 1(2).

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 $\langle s_k, \ldots, s_j \rangle$ of σ

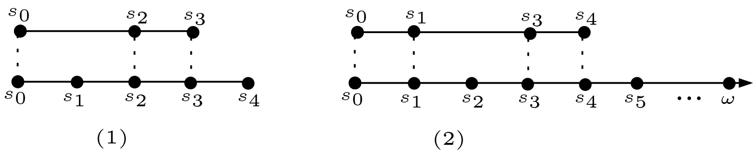

Fig. 1. Projected intervals.

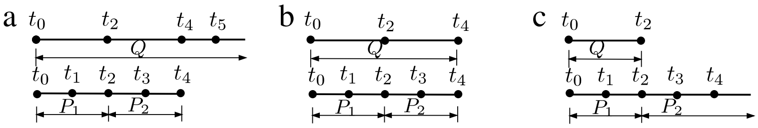

Fig. 2. Semantics of (P_1, P_2) *prj* Q.

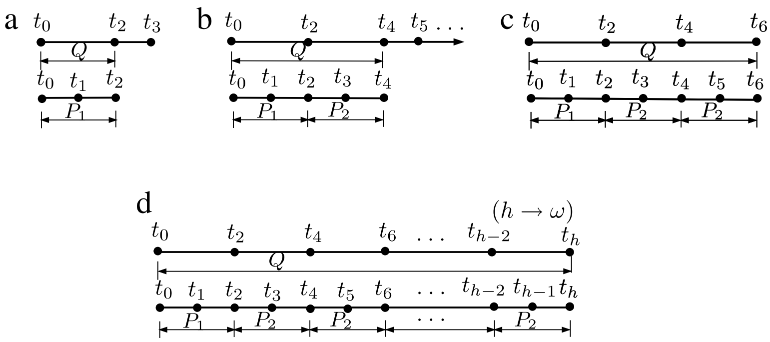

Fig. 3. Semantics of $(P_1, P_2^{\circledast})$ *prj* Q.

with the current state being s_k. The satisfaction relation ($\models$) is inductively defined as follows:

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

I − not $\quad \mathcal{I} \models \neg P$ iff $\mathcal{I} \not\models P$

I − or $\quad \mathcal{I} \models P \vee Q$ iff $\mathcal{I} \models P$ or $\mathcal{I} \models Q$

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

I − prj $\quad \mathcal{I} \models (P_1, \ldots, P_m)$ *prj* Q, if there exist integers $k = r_0 \le r_1 \le \ldots \le r_m \le j$ such that $(\sigma, r_{l-1}, r_l) \models P_l$, $1 \le l \le m$, and $(\sigma', 0, |\sigma'|) \models Q$ for one of the following σ' :
(a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_m+1..j)}$ or
(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_h)$ for some $0 \le h \le m$

I − prj$^{\circledast}$ $\quad \mathcal{I} \models (P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)$ *prj* Q iff : $1 \le i \le s \le m$ and $\exists n \in N_0$, $\mathcal{I} \models (P_1, \ldots, (P_i, \ldots, P_s)^{(n)}, \ldots, P_m)$ *prj* Q
or: $s = m$ and there exist infinitely many integers $k = r_0 \le r_1 \le \ldots \le r_h \le \omega$, such that $(\sigma, r_{l-1}, r_l) \models P_l$, $0 < l < i$, $(\sigma, r_{l-1}, r_l) \models P_t$, $l \ge i$, $t = i + ((l - i)\ mod\ (s - i + 1))$, and $(\sigma', 0, |\sigma'|) \models Q$, where $\sigma' = \sigma \downarrow (r_0, r_1, \ldots)$.

Note that in $(P_1, \ldots, (P_i, \ldots, P_s)^{\circledast}, \ldots, P_m)$ *prj* Q, if $i = 1$ and $s = m$, we have $(P_1, \ldots, P_m)^{\circledast}$ *prj* Q; if $i = s$, we obtain $(P_1, \ldots, (P_i)^{\circledast}, \ldots, P_m)$ *prj* Q; and if $i = s = m = 1$, we get $(P_1)^{\circledast}$ *prj* Q. Fig. 2 shows the possible semantics of (P_1, P_2) *prj* Q. Here Q and P_1 start to be interpreted at state t_0; subsequently, P_1 and P_2 are interpreted sequentially; Q is interpreted in parallel with $(P_1; P_2)$ over the interval consisting of endpoints of subintervals over which P_1 and P_2 are interpreted. The possible three cases are given: (a) P_2 terminates before Q; (b) Q and P_2 terminate at the same state; (c) Q terminates before P_2. Projection construction is useful in the specification of concurrent system. Compared with the *prj* construct, the *prj*$^{\circledast}$ construct permits the repeated occurrence for some part in the l.h.s of *prj* operator. For instance, semantics of $(P_1, P_2^{\circledast})$ *prj* Q can be illustrated in Fig. 3.

2.3. 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. Büchi automata with stutter

Since a PPTL* formula can be satisfied by both finite and infinite models, it is not capable of specifying such a formula with a classical finite state automaton or a Büchi automaton. Thus, a kind of extended Büchi automata is introduced for this purpose. We first recall the definition of Büchi automata.

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

- $Q = \{q_0, q_1, \ldots, q_n\}$ is a finite, non-empty set of locations;
- $\Sigma = \{a_0, a_1, \ldots, a_m\}$ 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 $w = a_0a_1\ldots$ of symbols, for each i, $a_i \in \Sigma$. A run of B over an infinite word $w = a_0a_1\ldots$ is an infinite sequence $\rho = q_0q_1\ldots$ 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 Büchi 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 [12] for modeling finite behaviors of a system with a Büchi 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 Büchi 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 Büchi automaton B, the stutter extension of finite run ρ with final state q_n is the ω-run ρ such that q_n^ω is the suffix of ρ such that $(q_n, \epsilon, q_n) \in \delta$. 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 Büchi 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 Büchi automata with the stutter extension, simply as Stutter-Büchi Automata (SBA for short).

4. Extended regular expression

Corresponding to the stutter-Büchi automata, we define a kind of Extended Regular Expression (ERE) which is capable of defining both finite and infinite strings. Let $\Upsilon = \{r_1, \ldots, r_n\}$ be a finite set of symbols, namely an alphabet. The extended regular expressions are defined as follows,

ERE	$R ::= \emptyset \mid \epsilon \mid r \mid R + R \mid R \bullet R \mid R^\omega \mid R^*$

where $r \in \Upsilon$, ϵ denotes an empty string; $+, \bullet$ and $*$ are union, concatenation and Kleene (star) closure respectively; x^ω means x is concatenated for infinitely many times. 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, $a_0a_1\ldots a_i\ldots$, 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,

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

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

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

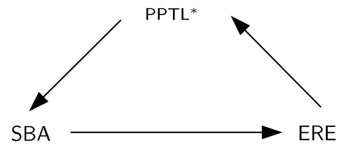

Fig. 4. The relationship between three different notations.

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

$$W \bullet W' = \{w \bullet w' \mid w \in W \text{ and } w' \in W'\}$$
$$W^\omega = \{w^\omega \mid w \in W\}$$
$$W^* = \bigcup_{w \in W} w^*$$

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

$$\begin{array}{lrcl}
\mathsf{Lr}_1 & L(\emptyset) & = & \emptyset \\
\mathsf{Lr}_2 & L(r) & = & \{r\} \\
\mathsf{Lr}_3 & L(\epsilon) & = & \{\epsilon\} \\
\mathsf{Lr}_4 & L(R{+}R) & = & L(R) \cup L(R) \\
\mathsf{Lr}_5 & L(R \bullet R) & = & L(R) \bullet L(R) \\
\mathsf{Lr}_6 & L(R^\omega) & = & L(R)^\omega \\
\mathsf{Lr}_7 & L(R^*) & = & L(R)^*
\end{array}$$

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

5. Equivalence between PPTL*, ERE and SBA

Even though the extended regular expression, PPTL*, and stutter-Büchi automata describe languages fundamentally in different ways, it turns out that they represent exactly the same class of languages, named the "full regular languages". In order to prove that PPTL*, ERE and SBA define the same class of languages, we will show the following facts: (1) each language defined by a PPTL* formula can be defined by an SBA; (2) each language defined by an SBA can be defined by an extended regular expression; (3) each language defined by an ERE can be defined by a PPTL* formula. The relationship is depicted in Fig. 4, where an arc from language class X to Y means that each language defined by X can also be defined by Y. This convinces us that three notations define the same language class.

For smooth transformations among PPTL* formulas, SBAs and EREs, according to the set Q_p of atomic propositions appearing in PPTL* formula Q, $|Q_p| = l$, alphabets Σ and Υ corresponding to SBA and ERE are defined respectively. We first define sets A_i, $1 \le i \le l$, as follows,

$$A_i = \{\{\dot{q_{j_1}}, \ldots, \dot{q_{j_i}}\} \mid q_{j_k} \in Q_p, \dot{q_{j_k}} \text{ denotes } q_{j_k} \text{ or } \neg q_{j_k}, 1 \le k \le i\}$$

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,

$$\begin{aligned}
\Sigma &= \bigcup_{i=1}^{3} A_i \cup \{true\} \cup \{\epsilon\} \\
&= \{\{\dot{p_1}\}, \{\dot{p_2}\}, \{\dot{p_3}\}, \{\dot{p_1}, \dot{p_2}\}, \{\dot{p_1}, \dot{p_3}\}, \{\dot{p_2}, \dot{p_3}\}, \{\dot{p_1}, \dot{p_2}, \dot{p_3}\}, true, \epsilon\} \\
\Upsilon &= \bigcup_{i=1}^{3} A_i \cup \{true\} \\
&= \{\{\dot{p_1}\}, \{\dot{p_2}\}, \{\dot{p_3}\}, \{\dot{p_1}, \dot{p_2}\}, \{\dot{p_1}, \dot{p_3}\}, \{\dot{p_2}, \dot{p_3}\}, \{\dot{p_1}, \dot{p_2}, \dot{p_3}\}, true\}
\end{aligned}$$

Obviously, for each $r \in \Upsilon$, r is a set of atomic propositions or their negations, denoted by $\{\dot{q}_i, \ldots, \dot{q}_j\}$, where $1 \le i \le j \le l$, or *true*.

5.1. From PPTL* to stutter-Büchi automata

For PPTL* formulas, their normal forms are the same as the ones for PPTL formulas [8,9]. In [29], an algorithm is given for transforming a PPTL* formula to its normal form. Further, based on the normal form, Labeled Normal Form Graphs (LNFGs)

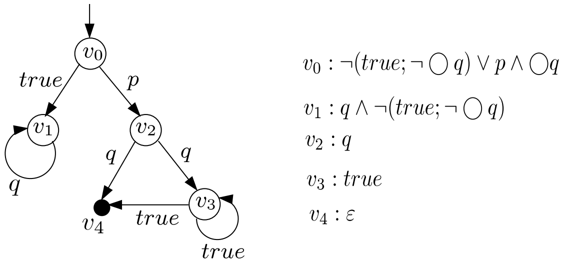

Fig. 5. LNFG of formula $\neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$.

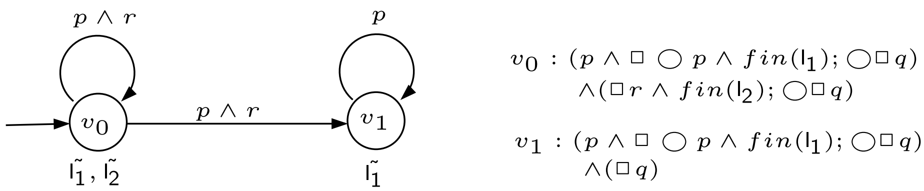

Fig. 6. LNFG of $(p \wedge \Box \bigcirc p; \bigcirc\Box q) \wedge (\Box r; \bigcirc\Box q)$.

for PPTL* formulas are constructed to precisely characterize the models of PPTL* formulas. Also an algorithm is given to construct the LNFG of a PPTL* formula [29]. The details about normal forms and LNFGs can be found in [8,29,9]. Here we focus on how to transform an LNFG to an SBA. For the clear presentation of the transformation, LNFGs are briefly introduced first.

For a PPTL* formula P, its LNFG is a tuple $G = (CL(P), EL(P), V_0, \mathbb{L} = \{\mathbb{L}_1, \dots, \mathbb{L}_m\})$, where $CL(P)$ and $EL(P)$ is the set of nodes and edges respectively, V_0 is the set of initial nodes, each $\mathbb{L}_k \subseteq CL(P)$, $1 \le k \le m$, is the set of nodes with $\tilde{l}_k$ labels. Actually, in an LNFG, a node $v \in CL(P)$ denotes a PPTL* formula 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* 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 are examples of LNFGs.

Example 1. LNFG of formula $P \equiv \neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$.

As shown in Fig. 5, the LNFG of formula $P \equiv \neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$ is $G = \{CL(P), EL(P), V_0, \mathbb{L}\}$, where $CL(P) = \{v_0, v_1, v_2, v_3, v_4\}$; $EL(P) = \{(v_0, true, v_1), (v_0, p, v_2), (v_1, q, v_1), (v_2, q, v_4), (v_2, q, v_3), (v_3, true, v_3), (v_3, true, v_4)\}$, $V_0 = \{v_0\}$; and $\mathbb{L} = \emptyset$. □

Example 2. LNFG of $P \equiv (p \wedge \Box \bigcirc p; \bigcirc\Box q) \wedge (\Box r; \bigcirc\Box q)$.

LNFG $G = (CL(P), EL(P), V_0, \ \mathbb{L} = \{\mathbb{L}_1, \dots, \mathbb{L}_m\})$ of formula $(p \wedge \Box \bigcirc p; \bigcirc\Box q) \wedge (\Box r; \bigcirc\Box q)$ is constructed as depicted in Fig. 6, where $CL(P) = \{v_0, v_1\}$, $EL(P) = \{(v_0, p \wedge r, v_0), (v_0, p \wedge r, v_1), (v_1, p, v_1)\}$, $V_0 = \{v_0\}$, $\mathbb{L} = \{\mathbb{L}_1, \mathbb{L}_2\}$, $\mathbb{L}_1 = \{v_0, v_1\}$ and $\mathbb{L}_2 = \{v_0\}$. □

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 $G = (CL(P), EL(P), V_0, \ \mathbb{L} = \{\mathbb{L}_1, \dots, \mathbb{L}_m\})$ of formula P, an SBA, $B = (Q, \Sigma, I, \delta, F)$, 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, and 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,

$$atom(true) = true$$

$$atom(\dot{q}_k) = \begin{cases} \{q_k\}, & \text{if } \dot{q}_k \equiv q_k \ 1 \le k \le l \\ \{\neg q_k\}, & \text{otherwise} \end{cases}$$

$$atom\left(\bigwedge_{k=1}^{m_0} \dot{q}_k\right) = atom(\dot{q}_1) \cup atom\left(\bigwedge_{k=2}^{m_0} \dot{q}_k\right)$$

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, $\delta(\varepsilon, \epsilon) = \{\varepsilon\}$.

Table 2
Algorithm for obtaining SBAs from an LNFG.

Function LNFG-SBA(G)
/* precondition: $G = (CL(P), EL(P), V_0, \ \mathbb{L} = \{\mathbb{L}_1, \ldots, \mathbb{L}_m\})$ 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 ε, $F = F \cup \{q_i\}$; $\delta(q_i, \epsilon) = \{q_i\}$;
    else if $v_i \in \overline{(\mathbb{L}_1 \cup \cdots \cup \mathbb{L}_m)}$, $F = F \cup \{q_i\}$;
  end for
  if $q_0 \in V_0, I = I \cup \{q_0\}$;
  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

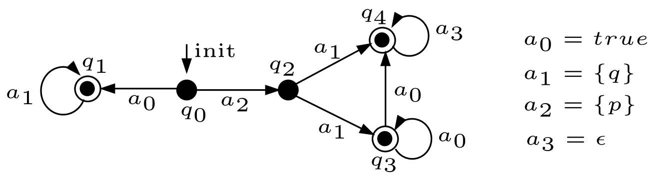

Fig. 7. Stutter-Büchi automaton of formula $P \equiv \neg(true; \neg \bigcirc q) \vee p \wedge \bigcirc q$.

- Accepting locations F: It has been proved that infinite paths with $\mathsf{Inf}(\pi) \not\subseteq \mathbb{L}_i$ for all $1 \le i \le m$ precisely characterize infinite models of P. This can be equivalently expressed by "infinite paths with $\mathsf{Inf}(\pi) \cap \overline{\mathbb{L}_i} \neq \emptyset$ for all $1 \le i \le m$ precisely characterize infinite models of P", where $\overline{\mathbb{L}_i}$ denotes $CL(P) \setminus \mathbb{L}_i$. That is,

$$\mathsf{Inf}(\pi) \cap (\overline{\mathbb{L}_1} \cap \cdots \cap \overline{\mathbb{L}_m}) \neq \emptyset \Leftrightarrow \mathsf{Inf}(\pi) \cap \overline{(\mathbb{L}_1 \cup \cdots \cup \mathbb{L}_m)} \neq \emptyset$$

  This precisely coincides with defining $F = \overline{(\mathbb{L}_1 \cup \cdots \cup \mathbb{L}_m)}$ to be Büchi acceptance. In addition, by employing the stutter extension rule, ε node is also an accepting location.

Formally, algorithm LNFG-SBA shown in Table 2 is used for transforming an LNFG to an SBA. Also Example 3 is given to show how the algorithm works.

Example 3. Constructing the SBA, $B = (Q, \Sigma, I, \delta, F)$, from the LNFG in Example 1.

As depicted in Fig. 7, 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. □

For the LNFG in Example 2, according to algorithm LNFG-SBA, when transformed as an SBA, the accepting set $F = \emptyset$.

5.2. From stutter-Büchi automata to extended regular expression

For the proof of the language $L(A)$ of any finite state automaton A is regular [27], Arden's rule [13] plays important roles.

Theorem 1. *(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. 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 $Q = \{q_0, \ldots, q_n\}$ and the starting location q_0. For $1 \le i \le n$, 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 3, we have,

(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) $X_4 = a_3X_4 + a_3^\omega$

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),

$$\begin{aligned} X_3 &= a_0X_3 + a_0X_4 + a_0^\omega = a_0X_3 + a_0 + a_0^\omega = a_0^*a_0 + a_0^*a_0^\omega = a_0^*a_0 \\ &= true^*true \end{aligned}$$

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,

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

5.3. From extended regular expressions to PPTL* formulas

Let Γ be the set of all models of PPTL*. For any extended regular expression $R \in$ ERE, we can construct a PPTL* 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 $\Omega : \Gamma \to \Upsilon^*$ from models of PPTL* formulas to words of extended regular expression is defined as follows,

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

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.

$$F_\emptyset \stackrel{\text{def}}{=} false$$

$$F_\epsilon \stackrel{\text{def}}{=} empty$$

$$F_r \stackrel{\text{def}}{=} \begin{cases} \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge skip, & \text{if } r = \{\dot{p}_i, \dots, \dot{p}_j\},\ 1 \leq i \leq j \leq l \\ true \wedge skip, & \text{if } r = true \end{cases}$$

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

$$F_{R_1+R_2} \stackrel{\text{def}}{=} F_{R_1} \vee F_{R_2}$$

$$F_{R_1 \bullet R_2} \stackrel{\text{def}}{=} F_{R_1}; F_{R_2}$$

$$F_{R^\omega} \stackrel{\text{def}}{=} F_R^* \wedge \Box more$$

$$F_{R_1^*} \stackrel{\text{def}}{=} F_{R_1}^*$$

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 $\sigma \models F_R$.

Theorem 2. *For an arbitrary extended regular expression $R \in ERE$ and any $\sigma \in \Gamma$, if $\sigma \models F_R$ then $\Omega(\sigma) \in L(R)$; conversely, for any $w \in L(R)$, there exists $\sigma \in \Gamma$, $\sigma \models F_R$, and $\Omega(\sigma) = w$.*

Proof. $\Rightarrow$: For any $\sigma \in \Gamma$, if $\sigma \models F_R$, then $\Omega(\sigma) \in L(R)$. The proof proceeds by induction on the structure of R.
Base Case:

1. For $r \in \Upsilon$, if $r = \{\dot{p}_i, \dots, \dot{p}_j\}$, $1 \leq i \leq j \leq l$, the unique model satisfying $F_r \equiv \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge skip$ is $\sigma = \langle \{\dot{p}_i, \dots, \dot{p}_j\}, true \rangle$ (this means $A(s_0) = \{\dot{p}_i, \dots, \dot{p}_j\}$ and $A(s_1) = true$). Clearly, $\Omega(\sigma) = \{\dot{p}_i, \dots, \dot{p}_j\} = r \in L(r)$; otherwise, if $r = true$, the unique model satisfying $F_r \equiv true \wedge skip$ is $\sigma = \langle true, true \rangle$. Obviously, $\Omega(\sigma) = true = r \in L(r)$.

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

2. For ϵ, the unique model satisfying $F_\epsilon \equiv \mathit{empty}$ is $\sigma = \langle \mathit{true} \rangle$. Since $|\sigma| = 0$, $\Omega(\sigma) = \epsilon \in L(\epsilon)$.
3. For $\emptyset$, no models can satisfy $F_\emptyset \equiv \mathit{false}$.

Inductive Step: Suppose for regular expression R_1 and R_2, and for any $\sigma_1 \in \Gamma$, if $\sigma_1 \models F_{R_1}$, then $\Omega(\sigma_1) \in L(R_1)$; and for any $\sigma_2 \in \Gamma$, if $\sigma_2 \models F_{R_2}$, then $\Omega(\sigma_2) \in L(R_2)$. Thus,

1. For $R_1 + R_2$, $F_{R_1+R_2} \equiv F_{R_1} \vee F_{R_2}$. So, for any $\sigma \models F_{R_1+R_2}$, we have $\sigma \models F_{R_1}$ or $\sigma \models F_{R_2}$. By hypothesis, $\Omega(\sigma) \in L(R_1)$ or $\Omega(\sigma) \in L(R_2)$. So, by Lr_4, $\Omega(\sigma) \in L_{R_1+R_2}$.
2. For $R_1 \bullet R_2$, $F_{R_1 \bullet R_2} \equiv F_{R_1}; F_{R_2}$. So for any $\sigma_1, \sigma_2 \in \Gamma$, if $\sigma_1 \models F_{R_1}$ and $\sigma_2 \models F_{R_2}$, then $\sigma_1 \circ \sigma_2 \models F_{R_1}; F_{R_2}$. By hypothesis, $\Omega(\sigma_1) \in L(R_1)$ and $\Omega(\sigma_2) \in L(R_2)$, so $\Omega(\sigma_1) \bullet \Omega(\sigma_2) \in L(R_1) \bullet L(R_2)$. Thus, by Lr_5, $\Omega(\sigma_1) \bullet \Omega(\sigma_2) \in L(R_1 \bullet R_2)$.
3. For R_1^ω, $F_{R_1^\omega} \equiv F_{R_1^*} \wedge \Box \mathit{more}$. So for any $\sigma_1 \in \Gamma$, if $\sigma_1 \models F_{R_1}$, then $\sigma_1^{\circ\omega} \models F_{R_1} \wedge \Box \mathit{more}$. By hypothesis, $\Omega(\sigma) \in L(R_1)$, so $\Omega(\sigma^{\circ\omega}) = \Omega(\sigma)^\omega \in L(R_1)^\omega$. Thus, by Lr_6, $\Omega(\sigma^{\circ\omega}) \in L(R_1^\omega)$.
4. For R_1^*, $F_{R_1^*} \equiv F_{R_1}^*$, so for any $\sigma \in \Gamma$, $\sigma \models F_{R_1}$, we have $\sigma^{\circ *} \models F_{R_1}^*$. By hypothesis, $\Omega(\sigma) \in L(R_1)$, so $\Omega(\sigma^{\circ *}) = \Omega(\sigma)^* \in L(R_1)^*$. By Lr_7, $\Omega(\sigma^{\circ *}) \in L(R_1^*)$.

$\Leftarrow$: For any $w \in L(R)$, there exists $\sigma \models F_R$, and $\Omega(\sigma) = w$. The proof also proceeds by induction on the structure of R.
Base Case:

1. For $r \in \Upsilon$, $L(r) = \{r\}$. If $r = \{\dot{p}_i, \ldots, \dot{p}_j\}$, $1 \le i \le j \le l$, $F_r \equiv \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge \mathit{skip}$, we have $\sigma = \langle \{\dot{p}_i, \ldots, \dot{p}_j\}, \mathit{true} \rangle \models F_r$ and $\Omega(\sigma) = \{\dot{p}_i, \ldots, \dot{p}_j\} = r \in L(r)$; otherwise if $r = \mathit{true}$, $F_r \equiv \mathit{true} \wedge \mathit{skip}$, we have $\sigma = \langle \mathit{true}, \mathit{true} \rangle \models F_r$ and $\Omega(\sigma) = \mathit{true} = r \in L(r)$;
2. For ϵ, $L(\epsilon) = \{\epsilon\}$. Since $F_\epsilon \equiv \mathit{empty}$, we have $\sigma = \langle \mathit{true} \rangle \models F_\epsilon$, and $\Omega(\sigma) = \epsilon \in L(\epsilon)$.
3. For $\emptyset$, $L(\emptyset) = \emptyset$. Since $F_\emptyset \equiv \mathit{false}$, there exist no models that satisfy $F_\emptyset$.

Inductive Step: Suppose for regular expressions R_1 and R_2, for any $w_1 \in L(R_1)$, there exists $\sigma_1 \in \Gamma$, $\sigma_1 \models F_{R_1}$ and $\Omega(\sigma_1) = w_1$, and for any $w_2 \in L(R_2)$, there exists $\sigma_2 \in \Gamma$, $\sigma_2 \models F_{R_2}$ and $\Omega(\sigma_2) = w_2$. Thus,

1. For $R_1 + R_2$, $L(R_1 + R_2) = L(R_1) \cup L(R_2)$. For any $w \in L(R_1) \cup L(R_2)$, we have $w \in L(R_1)$ or $w \in L(R_2)$. By hypothesis, there exists $\sigma \in \Gamma$, $\sigma \models F_{R_1}$ and $\Omega(\sigma) = w$, or $\sigma \models F_{R_2}$ and $\Omega(\sigma) = w$. Thus, we have for any $w \in L(R_1) \cup L(R_2)$, there exists $\sigma \models F_{R_1} \vee F_{R_2}$, and $\Omega(\sigma) = w$.
2. For $R_1 \bullet R_2$, $L(R_1 \bullet R_2) = L(R_1) \bullet L(R_2)$. For any $w \in L(R_1) \bullet L(R_2)$, there exist w_1, w_2, $w = w_1 \bullet w_2$, $w_1 \in L(R_1)$ and $w_2 \in L(R_2)$. By hypothesis, there exists $\sigma_1 \in \Gamma$, $\sigma_1 \models F_{R_1}$ and $\Omega(\sigma_1) = w_1$, and there exists $\sigma_2 \in \Gamma$, $\sigma_2 \models F_{R_2}$ and $\Omega(\sigma_2) = w_2$. Thus, we have $\sigma = \sigma_1 \circ \sigma_2 \models F_{R_1}; F_{R_2} = F_{R_1 \bullet R_2}$, and $\Omega(\sigma) = w_1 \bullet w_2 = w$.
3. For R_1^ω, $L(R_1^\omega) = L(R_1)^\omega$. For any $w \in L(R_1)^\omega$, there exists $w_1 \in L(R_1)$, and $w_1^\omega = w$. By hypothesis, there exists $\sigma_1 \in \Gamma$, $\sigma_1 \models F_{R_1}$ and $\Omega(\sigma_1) = w_1$. Thus we have $\sigma = \sigma_1^\omega \models F_{R_1} \wedge \Box \mathit{more} \equiv F_{R_1}^\omega$, $\Omega(\sigma) = \Omega(\sigma_1^{\circ\omega}) = \Omega(\sigma_1)^\omega = w_1^\omega = w$.
4. For R_1^*, $L(R_1^*) = L(R_1)^*$. For any $w \in L(R_1)^*$, there exists $w_1 \in L(R_1)$ and $w_1^* = w$. By hypothesis, there exists $\sigma_1 \in \Gamma$, $\sigma_1 \models F_{R_1}$ and $\Omega(\sigma_1) = w_1$. Thus, we have $\sigma = \sigma_1^{\circ *} \models F_{R_1}^* \equiv F_{R_1^*}$ and $\Omega(\sigma) = \Omega(\sigma_1^{\circ *}) = \Omega(\sigma_1)^* = w_1^* = w$. $\Box$

Example 4. Constructing PPTL* formula from the extended regular expression, $\mathit{true}\{q\}^\omega + \{p\}\{q\} + \{p\}\{q\}\mathit{true}^*\mathit{true}$ obtained in Example 3.

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

6. Characterizing fragments of PPTL*

The basic temporal operators in PPTL* are *next*, *chop*, *chop**, *chop*$^+$, *prj*, *prj*$^\circledast$ and *prj*$^\oplus$. So if we determine fragments of PPTL* by disallowing the use of some of these operators, different fragments of PPTL* are obtained. 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 temporal operators *next*, *chop* and the basic connections in typical propositional logic. In the rest of the section, we mainly investigate the characters of $\mathsf{L}(\mathsf{next}, \mathsf{chop})$, $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*)$, $\mathsf{L}(\mathsf{chop})$, $\mathsf{L}(\mathsf{chop}, \mathsf{chop}^+)$ and $\mathsf{L}(\mathsf{next}, \mathsf{prj})$, for some others obviously share the same characters with the five fragments.

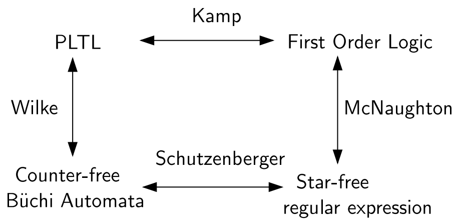

Fig. 8. Transformations among PLTL, first order logic, counter-free Büchi automata and star-free regular expressions.

6.1. Characterization of L(next, chop)

Researching results [24–26] show that PLTL [1] with basic temporal operators *next* and *until*, denoted by L(next, until) is less expressive than Büchi automata, and has the same expressiveness as first order logic, counter-free Büchi automata and star-free regular regular expressions, and the corresponding transformations as shown in Fig. 8 were given.

Actually, L(next, chop) has the same expressiveness as L(next, until), and equals to star-free regular expressions. The conclusion is formalized and proved in Theorem 3. We first recall the definition of star-free regular expressions. Notice that, in the following, the notations Γ, Υ and Ω are the same as before, so we use them without declaration.

Definition 2. The general regular expressions which are those built up from the constants $\emptyset$, ϵ and the alphabet symbols $r \in \Upsilon$, and $\bullet, +, *$ and $\sim$ (complementation). The star-free regular expressions are general regular expressions without occurrences of Kleene closure [23].

Let Υ be the alphabet. Star-free Regular Expression (star-free RE) is defined as follows,

star-free RE	$\mathcal{R} ::= \emptyset \mid \epsilon \mid r \mid \mathcal{R} + \mathcal{R} \mid \mathcal{R} \bullet \mathcal{R} \mid \ \sim \mathcal{R}$

The language $L(\mathcal{R})$ expressed by the extended star-free regular expression $\mathcal{R}$ is given by,

$$\begin{array}{llcl}
\mathsf{L}_{\mathcal{R}_1} & L(\emptyset) & = & \emptyset \\
\mathsf{L}_{\mathcal{R}_2} & L(r) & = & \{r\} \\
\mathsf{L}_{\mathcal{R}_3} & L(\epsilon) & = & \{\epsilon\} \\
\mathsf{L}_{\mathcal{R}_4} & L(\mathcal{R}+\mathcal{R}) & = & L(\mathcal{R}) \cup L(\mathcal{R}) \\
\mathsf{L}_{\mathcal{R}_5} & L(\mathcal{R} \bullet \mathcal{R}) & = & L(\mathcal{R}) \bullet L(\mathcal{R}) \\
\mathsf{L}_{\mathcal{R}_6} & L(\sim \mathcal{R}) & = & \Upsilon^* \setminus L(\mathcal{R})
\end{array}$$

For a string w, if $w \in L(\mathcal{R})$, w is called a word of the star-free regular expression $\mathcal{R}$.

Theorem 3. L(next, chop) *has the same expressiveness as star-free regular expressions.*

Proof. The proof consists of a pair of transformations between the counter-parts of the operators in L(next, chop) and star-free regular expressions.

(i) For any star-free expression $\mathcal{R}$, a formula $F_{\mathcal{R}}$ in L(next, chop) can be constructed such that for any word $w \in L(\mathcal{R})$, there exists $\sigma \in \Gamma$, $\sigma \models F_{\mathcal{R}}$, and $\Omega(\sigma) = w$; and for any $\sigma \in \Gamma$, $\sigma \models F_{\mathcal{R}}$, $\Omega(\sigma) \in L(\mathcal{R})$. $F_{\mathcal{R}}$ is constructed inductively on the structure of $\mathcal{R}$ as follows.

$$F_{\emptyset} \stackrel{\text{def}}{=} \mathit{false}$$

$$F_{\epsilon} \stackrel{\text{def}}{=} \mathit{empty}$$

$$F_r \stackrel{\text{def}}{=} \begin{cases} \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge \mathit{skip}, & \text{if } r = \{\dot{p}_i, \ldots, \dot{p}_j\}, \quad 1 \le i \le j \le l \\ \mathit{true} \wedge \mathit{skip}, & \text{if } r = \mathit{true} \end{cases}$$

where $r \in \Upsilon$. Inductively, if $\mathcal{R}_1$ and $\mathcal{R}_2$ are star-free regular expressions, then

$$F_{\mathcal{R}_1+\mathcal{R}_2} \stackrel{\text{def}}{=} F_{\mathcal{R}_1} \vee F_{\mathcal{R}_2}$$

$$F_{\mathcal{R}_1 \bullet \mathcal{R}_2} \stackrel{\text{def}}{=} F_{\mathcal{R}_1}; F_{\mathcal{R}_2}$$

$$F_{\sim \mathcal{R}_1} \stackrel{\text{def}}{=} \neg F_{\mathcal{R}_1}$$

The proof for the correctness of the transition for $\emptyset, \epsilon, r$, concatenation and union can be found in Theorem 2. In the following, we prove that, for any $w \in L(\sim \mathcal{R})$, there exists $\sigma \in \Gamma$, $\sigma \models F_{\sim \mathcal{R}}$, and $\Omega(\sigma) = w$; and for any $\sigma \in \Gamma$, $\sigma \models F_{\sim \mathcal{R}}$, $\Omega(\sigma) \in L(\sim \mathcal{R})$.

$\Rightarrow$: For any $w \in L(\sim \mathcal{R})$, by $\mathsf{L}_{\mathcal{R}_6}$, $w \in \Upsilon^* \setminus L(\mathcal{R})$. So $w \notin L(\mathcal{R})$. By Theorem 2, there exists $\sigma \in \Gamma$, $\sigma \not\models F_{\mathcal{R}}$, and $\Omega(\sigma) = w$. Thus, we have $\sigma \models \neg F_{\mathcal{R}}$, and $\Omega(\sigma) = w$.

$\Leftarrow$: For any $\sigma \in \Gamma$, if $\sigma \models \neg F_{\mathcal{R}}$, we have $\sigma \not\models F_{\mathcal{R}}$. By Theorem 2, $\Omega(\sigma) \notin L(\mathcal{R})$. So $\Omega(\sigma) \in \Upsilon^* \setminus L(\mathcal{R})$. By $\mathsf{L}_{\mathcal{R}6}$, $\Omega(\sigma) \in L(\sim \mathcal{R})$.

(ii) For any formula P in L(next, chop), a star-free regular expression R_P can be constructed such that for any $\sigma \in \Gamma$, $\sigma \models P$, $\Theta(\sigma) \in L(R_P)$; and for any $w \in L(R_P)$, there exists $\sigma \in \Gamma$, $\sigma \models P$, and $\Theta(\sigma) = w$. The mapping function $\Theta : \Gamma \to \Upsilon^*$ from models of PPTL* formulas to words of regular expression is defined as follows,

$p \in Prop$:	for any $\sigma \in \Gamma, \sigma \models p, \Theta(\sigma) = \{p\}, \{p\} \in \Upsilon$;
$\neg P$:	for any $\sigma \in \Gamma, \sigma \models \neg P, \Theta(\sigma) = \sim \Theta(\sigma')$, where $\sigma' \models P$;
$P_1 \vee P_2$:	for any $\sigma \in \Gamma, \sigma \models P_1 \vee P_2, \Theta(\sigma) = \Theta(\sigma_1) + \Theta(\sigma_2), \sigma_1 \models P_1, \sigma_2 \models P_2$;
$\bigcirc P$:	for any $\sigma \in \Gamma, \sigma \models \bigcirc P, \Theta(\sigma) = true \bullet \Theta(\sigma')$, where $\sigma' \models P$;
$P_1; P_2$:	for any $\sigma \in \Gamma, \sigma \models P_1; P_2, \Theta(\sigma) = \Theta(\sigma_1)' \bullet (r_1 \cup r_2) \bullet \Theta(\sigma_2)'$, where $\Theta(\sigma_1)' \bullet r_1 = \Theta(\sigma_1), r_2 \bullet \Theta(\sigma_2)' = \Theta(\sigma_2)$, and $\sigma_1 \models P_1, \sigma_2 \models P_2, \sigma = \sigma_1 \circ \sigma_2$.

R_P is constructed inductively on the structure of formulas in L(next, chop) as follows,

Proposition	R_p	$\stackrel{\text{def}}{=}$	$\{p\}, \{p\} \in \Upsilon$
Or	$R_{P_1 \vee P_2}$	$\stackrel{\text{def}}{=}$	$R_{P_1} + R_{P_2}$
Negation	$R_{\neg P}$	$\stackrel{\text{def}}{=}$	$\sim R_P$
Next	$R_{\bigcirc P}$	$\stackrel{\text{def}}{=}$	$true \bullet R_P$
Chop	$R_{P_1;P_2}$	$\stackrel{\text{def}}{=}$	$R'_{P_1} \bullet (r_1 \cup r_2) \bullet R'_{P_2}$, where $R'_{P_1} \bullet r_1 = R_{P_1}, r_2 \bullet R'_{P_2} = R_{P_2}$

The proof for the correctness of the transformation is straightforward by induction on the structure of formulas in L(next, chop). □

Theorem 4. *To express* chop *construct in* L(next, until)*, a non-elementary longer formula is needed.*

Proof. Following the conclusions in [8], it can be obtained that any *chop* construct $P; Q$ can be rewritten to its normal form,

$$P; Q \equiv R_{e1} \wedge empty \vee \bigvee_i (R_{e2} \wedge \bigcirc R)$$

where R_{e1} and R_{e2} are state formulas without temporal operators and R is an arbitrary formula with possibly *chop* operators. More precisely, if P and Q are rewritten into their normal forms,

$$P \equiv P_e \wedge empty \vee \bigvee_i (P_i \wedge \bigcirc P'_i)$$

$$Q \equiv Q_e \wedge empty \vee \bigvee_j (Q_j \wedge \bigcirc Q'_j)$$

It turns out that,

$$\begin{aligned} P; Q &\equiv (P_e \wedge empty \vee \bigvee_i (P_i \wedge \bigcirc P'_i)); Q \\ &\equiv P_e \wedge Q \vee \bigvee_i (P_i \wedge \bigcirc P'_i); Q \\ &\equiv P_e \wedge Q \vee \bigvee_i P_i \wedge \bigcirc (P'_i; Q) \end{aligned}$$

Notice that if P has only infinite models, $P; Q$ is obviously false. If P has finite models, as shown above, it can be expressed in terms of $\bigcirc(P'_i; Q)$. Thus, to eliminate the original *chop* operator, $P'_i; Q$ should again be recursively rewritten into its normal form. Eventually, $empty; Q \equiv Q$ will be produced since P has finite models and a sub-formula with an infinite model such as $P_{inf}; Q$ with P_{inf} being of infinite models is false. So, in the worst case, the original *chop* operator can be eliminated in terms of finitely many embedded next operators. In fact, The above rewriting procedure is the LNFG constructing procedure in [8] with non-elementary complexity in the terms of the length of formula P. Further, P and Q can contain only finitely many chop operators. So, to eliminate the chop operators contained in P and Q, similar procedures as above are needed. Thus, obviously, to eliminate *chop* operators in formula $P; Q$, a non-elementary longer formula is needed. □

Therefore, we can conclude that with the help of the *chop* operator, L(next, until) will be a non-elementary succincter. Thus, even though the complexity for model checking of logic with the *chop* operator is non-elementary to the length of the formula, it in fact shares the same complexity with model checking PLTL.

6.2. *Characterization of* L(next, chop, chop*)

Similar to L(next, chop), Theorem 5 is proved to show that L(next, chop, chop*) is as powerful as the expressiveness of extended full regular expressions.

Theorem 5. L(next, chop, chop*) *has the same expressiveness as extended regular expressions.*

Proof. The proof also consists of a pair of transformations between the counter-parts of the operators in L(next, chop, chop*) and extended regular expressions.

(i) For any regular expression R, a formula F_R in L(next, chop, chop*) can be constructed such that for any word $w \in L(R)$, there exists $\sigma \in \Gamma, \sigma \models F_R, \Omega(\sigma) = w$; and for any $\sigma \in \Gamma, \sigma \models F_R, \Omega(\sigma) \in L(R)$. F_R is inductively constructed as follows,

$$F_\emptyset \stackrel{\text{def}}{=} \mathit{false}$$

$$F_\epsilon \stackrel{\text{def}}{=} \mathit{empty}$$

$$F_r \stackrel{\text{def}}{=} \begin{cases} \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge \mathit{skip}, & \text{if } r = \{\dot{p}_i, \ldots, \dot{p}_j\},\ 1 \le i \le j \le l \\ \mathit{true} \wedge \mathit{skip}, & \text{if } r = \mathit{true} \end{cases}$$

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

$$F_{R_1+R_2} \stackrel{\text{def}}{=} F_{R_1} \vee F_{R_2}$$

$$F_{R_1 \bullet R_2} \stackrel{\text{def}}{=} F_{R_1} ; F_{R_2}$$

$$F_{R^\omega} \stackrel{\text{def}}{=} F_R^* \wedge \Box \mathit{more}$$

$$F_{R_1^*} \stackrel{\text{def}}{=} F_{R_1}^*$$

The proofs for the correctness of the transitions can be found in Theorem 2.

(ii) For any formula P in L(next, chop, chop*), a extended regular expression R_P can be constructed such that for any $\sigma \in \Gamma, \sigma \models P, \Theta(\sigma) \in L(R_P)$; and for any $w \in L(R_P)$, there exists $\sigma \in \Gamma, \sigma \models P$, and $\Theta(\sigma) = w$. Note that, for *chop star* construct, Θ is defined below,

For any $\sigma \in \Gamma, \sigma \models P^*, \Theta(\sigma) = \epsilon + r_1 \bullet (\Theta(\sigma)' \bullet (r_2 \cup r_1))^* \bullet \Theta(\sigma)' \bullet r_2$, where $r_1 \bullet \Theta(\sigma')' \bullet r_2 = \Theta(\sigma')$, and $\sigma' \models P$.

R_P is constructed inductively as follows,

Proposition	$R_p \stackrel{\text{def}}{=} \{p\}, \{p\} \in \Upsilon$
Or	$R_{P_1 \vee P_2} \stackrel{\text{def}}{=} R_{P_1} + R_{P_2}$
Negation	$R_{\neg P} \stackrel{\text{def}}{=} \sim R_P$
Next	$R_{\bigcirc P} \stackrel{\text{def}}{=} \mathit{true} \bullet R_P$
Chop	$R_{P_1;P_2} \stackrel{\text{def}}{=} R'_{P_1} \bullet (r_1 \cup r_2) \bullet R'_{P_2}$, where $R'_{P_1} \bullet r_1 = R_{P_1}, r_2 \bullet R'_{P_2} = R_{P_2}$
Chop star	$R_{P^*} \stackrel{\text{def}}{=} \epsilon + r_1 \bullet (R'_P \bullet (r_2 \cup r_1))^* \bullet R'_P \bullet r_2$ where $R_P = r_1 \bullet R'_P \bullet r_2$

The proof for the correctness of the transformation is straightforward by induction on the structure of formulas in L(next, chop, chop*). $\square$

6.3. *Characterizations of* L(chop) *and* L(chop, chop$^+$)

In what follows, we will prove that L(chop) has the same expressiveness as star-free regular expressions without ϵ.

Theorem 6. L(chop) *has the same expressiveness as star-free regular expressions without* ϵ*.*

Proof. The proof consists of a pair of transformations between the counter-parts of the operators in L(chop) and star-free regular expressions without ϵ.

(i) For any star-free expression $\mathcal{R}$ without ϵ, a formula $F_{\mathcal{R}}$ in L(chop) can be constructed such that for any word $w \in L(\mathcal{R})$, there exists $\sigma \in \Gamma, \sigma \models F_{\mathcal{R}}$, and $\Omega(\sigma) = w$; and for any $\sigma \in \Gamma, \sigma \models F_{\mathcal{R}}, \Omega(\sigma) \in L(\mathcal{R})$. $F_{\mathcal{R}}$ can be inductively constructed as follows,

$\emptyset$	$F_\emptyset \stackrel{\text{def}}{=} \mathit{false}$
$r \in \Upsilon$	$F_r \stackrel{\text{def}}{=} \begin{cases} \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge \mathit{skip}, & \text{if } r = \{\dot{p}_i, \ldots, \dot{p}_j\}, \text{ and } 1 \le i \le j \le l \\ \mathit{true} \wedge \mathit{skip}, & \text{if } r = \mathit{true} \end{cases}$
$\sim \mathcal{R}$	$F_{\sim\mathcal{R}} \stackrel{\text{def}}{=} \neg F_{\mathcal{R}}$
$\mathcal{R}_1 + \mathcal{R}_2$	$F_{\mathcal{R}_1+\mathcal{R}_2} \stackrel{\text{def}}{=} F_{\mathcal{R}_1} \vee F_{\mathcal{R}_2}$
$\mathcal{R}_1 \bullet \mathcal{R}_2$	$F_{\mathcal{R}_1\bullet\mathcal{R}_2} \stackrel{\text{def}}{=} F_{\mathcal{R}_1}; F_{\mathcal{R}_2}$

(ii) For any formula P in L(chop), a star-free regular expression without ϵ, E_P, can be constructed such that for any $\sigma \in \Gamma$, $\sigma \models P$, and $\Theta(\sigma) \in L(E_P)$; and for any $w \in L(E_P)$, there exists $\sigma \in \Gamma$, $\sigma \models P$, and $\Theta(\sigma) = w$. E_P can inductively be constructed as follows,

Proposition	R_p	$\stackrel{\text{def}}{=}$	$\{p\}, \{p\} \in \Upsilon$
Or	$R_{P_1 \vee P_2}$	$\stackrel{\text{def}}{=}$	$R_{P_1} + R_{P_2}$
Negation	$R_{\neg P}$	$\stackrel{\text{def}}{=}$	$\sim R_P$
Chop	$R_{P_1;P_2}$	$\stackrel{\text{def}}{=}$	$R'_{P_1} \bullet (r_1 \cup r_2) \bullet R'_{P_2}$, where $R'_{P_1} \bullet r_1 = R_{P_1}$, $r_2 \bullet R'_{P_2} = R_{P_2}$

The proofs for the correctness of the transformations are straightforward by induction on the structure of formulas in L(chop) and expressions of star-free expressions without ϵ. □

Similarly, Theorem 7 shows that L(chop, chop$^+$) has the expressiveness of regular expressions without ϵ.

Theorem 7. L(chop, chop$^+$) *has the same expressiveness as regular expressions without* ϵ.

Proof. The proof also consists of a pair of transformations between the counter-parts of the operators in L(chop, chop$^+$) and regular expressions without ϵ.

(i) For any regular expression R without ϵ, a formula F_R in L(chop, chop$^+$) can be constructed such that for any word $w \in L(R)$, there exists $\sigma \in \Gamma$, $\sigma \models F_R$, and $\Omega(\sigma) = w$; and for any $\sigma \in \Gamma$, $\sigma \models F_R$, $\Omega(\sigma) \in L(R)$. F_R is inductively constructed as below,

$\emptyset$	$F_\emptyset \stackrel{\text{def}}{=} \mathit{false}$
$r \in \Upsilon$	$F_r \stackrel{\text{def}}{=} \begin{cases} \dot{p}_i \wedge \cdots \wedge \dot{p}_j \wedge \mathit{skip}, & \text{if } r = \{\dot{p}_i, \ldots, \dot{p}_j\}, \text{ and } 1 \le i \le j \le l \\ \mathit{true} \wedge \mathit{skip}, & \text{if } r = \mathit{true} \end{cases}$
$\sim R$	$F_{\sim R} \stackrel{\text{def}}{=} \neg F_R$
$R_1 + R_2$	$F_{R_1+R_2} \stackrel{\text{def}}{=} F_{R_1} \vee F_{R_2}$
$R_1 \bullet R_2$	$F_{R_1\bullet R_2} \stackrel{\text{def}}{=} F_{R_1}; F_{R_2}$
R^+	$F_{R^+} \stackrel{\text{def}}{=} (F_R)^+$

(ii) For any formula P in L(chop, chop$^+$), a regular expression without ϵ, R_P, can be constructed such that for any $\sigma \in \Gamma$, $\sigma \models P$, and $\Theta(\sigma) \in L(R_P)$; and for any $w \in L(R_P)$, there exists $\sigma \in \Gamma$, $\sigma \models P$, and $\Theta(\sigma) = w$. R_P is inductively constructed as follows,

Proposition	R_p	$\stackrel{\text{def}}{=}$	$\{p\}, \{p\} \in \Upsilon$
Or	$R_{P_1 \vee P_2}$	$\stackrel{\text{def}}{=}$	$R_{P_1} + R_{P_2}$
Negation	$R_{\neg P}$	$\stackrel{\text{def}}{=}$	$\sim R_P$
Chop	$R_{P_1;P_2}$	$\stackrel{\text{def}}{=}$	$R'_{P_1} \bullet (r_1 \cup r_2) \bullet R'_{P_2}$, where $R'_{P_1} \bullet r_1 = R_{P_1}$, $r_2 \bullet R'_{P_2} = R_{P_2}$
Chop plus	R_{P^+}	$\stackrel{\text{def}}{=}$	$r_1 \bullet (R'_P \bullet (r_2 \cup r_1))^* \bullet R'_P \bullet r_2$ where $R_P = r_1 \bullet R'_P \bullet r_2$

The proofs for the correctness of the transformations are straightforward by induction on the structure of formulas in L(chop, chop$^+$) and expressions of regular expressions without ϵ. □

6.4. Characterizations of other fragments

Since *chop* and *chop star* can be subsumed by *projection* and *projection star* constructs respectively. Thus, obviously, L(next, chop, chop*), L(next, chop*, prj), L(next, chop, prj$^\circledast$), L(next, chop, chop*, prj), L(next, chop, chop*, prj$^\circledast$), and the full logic L(Next, Chop, chop*, prj, prj$^\circledast$) have the same expressiveness as full regular expressions.

$$
\begin{array}{ccccc}
\mathsf{L}(\mathsf{chop}) & \subset & \mathsf{L}(\mathsf{next}, \mathsf{chop}) & \subset & \mathsf{L}(\mathsf{next}, \mathsf{prj}) \\
 & & \cap & & \cap \\
 & & \mathsf{L}(\mathsf{chop}, \mathsf{chop}^*) & \subset & \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*) \\
 & & & & \| \\
 & & \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{prj}^{\circledast}) & = & \mathsf{L}(\mathsf{next}, \mathsf{prj}, \mathsf{chop}^*) \quad = \quad \mathsf{L}(\mathsf{next}, \mathsf{prj}, \mathsf{prj}^{\circledast}) \\
 & & \| & & \\
 & & \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*, \mathsf{prj}) & = & \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*, \mathsf{prj}^{\circledast}) \\
 & & \| & & \\
 & & \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*, \mathsf{prj}, \mathsf{prj}^{\circledast}) & &
\end{array}
$$

Fig. 9. Expressiveness relationship among the fragments of PPTL*.

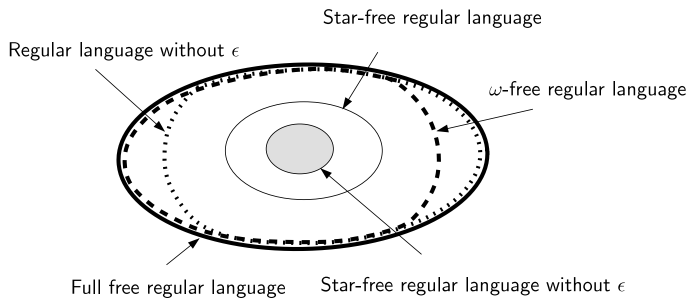

Fig. 10. Language classes for PPTL* and its fragments.

Further, the expressiveness of $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{prj})$ and $\mathsf{L}(\mathsf{next}, \mathsf{prj})$ are equal, and will have the following unprecise expressive power.

$$\mathsf{L}(\mathsf{next}, \mathsf{chop}) \subseteq \mathsf{L}(\mathsf{next}, \mathsf{prj}) \subseteq \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*) \tag{6.1}$$

For *projection* construct, $(P_1, \ldots, P_m)\ prj\ Q$, if $Q \equiv empty$, and $P_1 \equiv P_2 \equiv \ldots \equiv P_m \equiv P$, it is obtained,

$$(P_1, \ldots, P_m)\ prj\ Q \equiv P^m, \quad m \in N_0 \tag{6.2}$$

Clearly, P^m cannot be expressed using *chop* constructs since m is an arbitrary positive integer. Also since $m \in N_0$, thus P^ω cannot be specified by P^m. Therefore, we can conclude that,

$$\mathsf{L}(\mathsf{next}, \mathsf{chop}) \subset \mathsf{L}(\mathsf{next}, \mathsf{prj}) \subset \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*) \tag{6.3}$$

All in all, Fig. 9 shows the expressiveness relationship among the fragments of PPTL*.

Up to now, five language classes, star-free regular language without ϵ, star-free regular language, regular language without ϵ, ω-free regular language and full regular language for PPTL* and its fragments are obtained as shown in Fig. 10. And $\mathsf{L}(\mathsf{chop})$ has the expressiveness of star-free regular language without ϵ, $\mathsf{L}(\mathsf{next}, \mathsf{chop})$ is in the class of star-free regular language, $\mathsf{L}(\mathsf{chop}, \mathsf{chop}^*)$ is in the same expressiveness with regular language without ϵ, $\mathsf{L}(\mathsf{next}, \mathsf{prj})$ has the same express power as star-free regular language, and $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*)$, $\mathsf{L}(\mathsf{next}, \mathsf{chop}^*, \mathsf{prj})$, $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{prj}^{\circledast})$, $\mathsf{L}(\mathsf{next}, \mathsf{prj}, \mathsf{prj}^{\circledast})$, $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*, \mathsf{prj})$, $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*, \mathsf{prj}^{\circledast})$, as well as the full logic $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^*, \mathsf{prj}, \mathsf{prj}^{\circledast})$ have the same expressiveness as full regular language.

6.5. Characterization of propositional interval temporal logic

For interval temporal logic, $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{proj})$, $chop^{\circledast}$ operator can be derived from *proj*, and has the same expressiveness as *proj* [5]. Thus, it is easily obtained that $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{proj}) = \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^{\circledast})$. Note that, $chop^{\circledast}$ rather than $chop^*$ is used here since they indeed have different meanings. The following theorem shows that $\mathsf{L}(\mathsf{next}, \mathsf{prj})$ has the same expressiveness as $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{proj})$.

Theorem 8. $\mathsf{L}(\mathsf{next}, \mathsf{prj})$ *has the same expressiveness as* $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{proj})$.

Proof. We need only show $\mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^{\circledast})$ has the same expressiveness as $\mathsf{L}(\mathsf{next}, \mathsf{prj})$. In ITL, *proj* is only defined over finite intervals. Since the $chop^{\circledast}$ operator is derived from *proj*, the chop operator within $chop^{\circledast}$ can iterate only finitely many times; that is, $chop^\omega$ is disallowed in $chop^{\circledast}$. Thus, by the previous analysis for the expressiveness of $\mathsf{L}(\mathsf{next}, \mathsf{prj})$ (see (6.3)), we have $\mathsf{L}(\mathsf{next}, \mathsf{prj}) = \mathsf{L}(\mathsf{next}, \mathsf{chop}, \mathsf{chop}^{\circledast})$. $\square$

7. Conclusions

In this paper, we have proved that the expressiveness of PPTL* is the same as the full regular expressions. Also, the proof itself provides approaches to translate a PPTL* formula to an equivalent Büchi automaton, a Büchi automaton to an equivalent extended ω-regular expression, and an extended ω-regular expression to a PPTL* formula. Further, we have investigated the expressiveness of some fragments of PPTL*, and classified them into five language classes. These results are useful in practice. Moreover, we have also shown the expressiveness of PITL. In addition, to verify concurrent systems using PPTL*, we have developed a prototype model checker based on SPIN. Therefore, any systems with regular properties can be automatically verified within SPIN using PPTL*. Since this logic is of *chop*, *projection* and *projection star* operators, thus, the compositional specification and verification of concurrent systems can be done in SPIN.

As applications of PPTL*, we will use this logic to describe properties of composite web-services, in particular, composite processes of BPEL4WS, and employ PROMELA language to model the behavior of the composite process, then verify the properties based on SPIN. To do so, we need further improve our model checker into a practical system in the near future. Also, we are further motivated to formalize an axiomatic system for PPTL* and investigate the techniques for the verification of the concurrent systems based on PVS in the future.

References

[1] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, IEEE, New York, 1977, pp. 46-67.2.
[2] S.A. Kripke, Semantical analysis of modal logic I: normal propositional calculi, Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik 9 (1963) 67–96.
[3] R. Rosner, A. Pnueli, A choppy logic, in: First Annual IEEE Symposium on Logic in Computer Science, LICS, 1986, pp. 306–314.
[4] B. Moszkowski, Reasoning about digital circuits. Ph.D. Thesis, Department of Computer Science, Stanford University. TRSTAN-CS-83-970, 1983.
[5] B. Moszkowski, Compositional reasoning about projected and infinite time, iceccs, in: First IEEE International Conference on Engineering of Complex Computer Systems, ICECCS'95, 1995, p. 238.
[6] Z. Duan, An extended interval temporal logic and a framing technique for temporal logic programming, Ph.D. Thesis, University of Newcastle upon Tyne, May 1996.
[7] Z. Duan, X. Yang, M. Kounty, Framed temporal logic programming, Science of Computer Programming 70 (1) (2008) 31–61.
[8] Z. Duan, C. Tian, L. Zhang, A decision procedure for propositional projection temporal logic with infinite models, Acta Informatica 45 (1) (2008) 43–78.
[9] Cong Tian, Zhenhua Duan, Alternating interval based temporal logics, in: The proceedings of ICFEM 2010, in: LNCS, vol. 6447, 2010, pp. 694–709.
[10] C. Tian, Z. Duan, Model checking propositional projection temporal logic based on SPIN, in: ICFEM 2007, in: LNCS, vol. 4789, Springer-Verlag, 2007.
[11] P.L. Wolper, Temporal logic can be more expressive, Information and Control 56 (1983) 72–99.
[12] Gerard J. Holzmann, The model checker spin, IEEE Transactions on Software Engineering 23 (5) (1997) 279–295.
[13] D. Arden, Delayed-logic and finite-state machines, in: In Theory of Computing Machine Design, Univ. of Michigan Press, 1960, pp. 1–35.
[14] Dov Gabbay, Amir Pnueli, Saharon Shelah, Jonathan Stavi, On the temporal analysis of fairness, in: POPL'80: Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, 1980, pp. 163–173.
[15] Aravinda Prasad Sistla, Theoretical issues in the design and verification of distributed systems, Ph.D. Thesis, Harvard University, 1983.
[16] M.Y. Vardi, P. Wolper, Yet another process logic, in: Logic of Programs, in: Lecture Notes in Computer Science, vol. 164, Springer, 1984, pp. 501–512.
[17] Moshe Y. Vardi, A temporal fixpoint calculus, in: POPL'88, 1988, pp. 250–259.
[18] Robert McNaughton, Seymour A. Papert, Counter-Free Automata, in: M. I. T Research Monograph, vol. 65, The MIT Press, 1971.
[19] H. Barringer, R. Kuiper, A. Pnueli, Now you may compose temporal logic specifications, in: Proc. 16th STOC, 1984, pp. 51–63.
[20] H. Barringer, R. Kuiper, A. Pnueli, The compositional temporal approach to CSP-like language, in: Proc. IFIP Conference, The Role of Abstract Models in Information Processing, January 1985.
[21] V. Nguyen, A. Demers, D. Gries, S. Owicki, Behavior: A temporal approach to process modeling, in: Logics of Programs, in: LNCS, vol. 193, Springer-Verlag, 1985, pp. 237–254.
[22] V. Nguyen, D. Gries, S. Owicki, A model and temporal proof system for networks of processes, in: Proc. 12th POPL, 1985, pp. 121–131.
[23] E.A. Emerson, Temporal and modal logic, in: Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, MIT Press, 1990, pp. 997–1072 (Chapter 16).
[24] W. Thomas, Automata on infinite objects, in: Handbook of Theoretical Computer Science, vol. B, 1990, pp. 133–191.
[25] H. Kamp, Tense logic and the theory of linear order. Ph.D. Thesis, UCLA, 1968.
[26] R. McNaughton, S. Papert, Counter-Free Automata, The MIT Press, 1971.
[27] Robin Milner, Communicating and Mobile System: The-Calculus, Cambridge University Press, Cambridge, 1999.
[28] L. Stockmeyer, The complexity of decision problems in automata theory and logic, Ph.D. Thesis, MIT, available as Project MAC Technical Report 133, 1974.
[29] Cong Tian, Zhenhua Duan, Complexity of propositional projection temporal logic with star, in: Mathematical Structure in Computer Science, vol. 19, Cambridge University Press, 2009, pp. 73–100.