Expressiveness of propositional projection temporal logic with star

Cong Tian, Zhenhua Duan*
ICT 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 \(\omega\)-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.

© 2010 Elsevier B.V. All rights reserved.

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 (QTLT) [15], Extended Temporal Logic (ETL) [11,16] and linear mu-calculus (\(\nu\)TL) [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 \(\text{chop}\) operator. The work in [11] showed that process logic with both \(\text{chop}\) operator and its reflexive-transitive closure (\(\text{chop star}\)), which is called \(\text{slice}\) 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 \(\text{next}\), \(\text{chop}\) and projection operator, \(\text{proj}\). In two characteristic operators, \(\text{chop}\) implements a form of sequential composition while \(\text{proj}\) yields repetitive behaviors. ITL without \(\text{projection}\) has the similar expressiveness as Rosner and Pnueli’s choppy logic [3]. Further, addition of the \(\text{proj}\) 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, \((P_1, \ldots, P_m) \text{prj} Q\), which is much more flexible than the original one. Further, in the propositional case,\(^1\) the \(\text{projection}\) construct can be extended to \(\text{projection star}\), \((P_1, \ldots, (P_i, \ldots, P_j)^*, \ldots, P_m) \text{prj} Q\), so that it can subsume \(\text{chop}\), \(\text{chop star}\), and the original projection (\(\text{proj}\)) in [4]. This extension makes the underlying logic more powerful without loss of decidability [29].

\(\ast\) 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 SRFP Grant 200807010012.

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

\(^1\) In the first order case, \(\text{projection star}\) is a derived formula.

0304-3975/ - see front matter © 2010 Elsevier B.V. All rights reserved.
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 PTL 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 succint. That is to express a chop formula with the existing operators, such as next and until in PTL, a non-elementary longer formula is needed. Thus, actually, model checking PPTL* shares the same time complexity with PTL. 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 ERPs. 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 PTL.

2.1. Syntax

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

$$P ::= p | P ∪ P | P ∨ Q | (P₁, . . . , Pₘ) prj Q | (P₁, . . . , (P₁, . . . , Pₗ)*) | . . . , Pₘ) prj Q \tag{1}$$

where $p \in$ Prop, $P₁, . . . , Pₘ, P$ and $Q$ are all well-formed PPTL* formulas. $∪$ (next), $prj$ (projection) and $prj^*$ (projection star) are basic temporal operators.

The abbreviations true, false, ∧, → and ↔ are defined as usual. In particular, true $≡ P ∨ ¬P$ and false $≡ P ∧ ¬P$. In addition, we have the derived formulas as shown in Table 1. Where $∪$ (weak next), $□$ (always), $⊙$ (sometimes), $∶$ (chop), $prj^*$ (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 $B = \{true, false\}$. $s : Prop → B$. 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, $|σ|$, 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 $N₀$ of non-negative integers and $ω, N₀ = N₀ ∪ \{ω\}$, and extend the comparison operators, $=, <, ≤, ≥$ to $N₀$ by considering $ω = ω$, and for all $i \in N₀, i < ω$. Moreover, we define $≤$ as $≤ − \{(ω, ω)\}$. To simplify definitions, we will denote $σ$ by $(s₀, . . . , s_{|σ|})$, where $s_{|σ|}$ is undefined if $σ$ is infinite. $s₀, . . . , s_{|σ|}$ is often used to present an infinite interval. With such a notation, $σ_{i,j}(0 ≤ i ≤ j ≤ |σ|)$ denotes the sub-interval $(s_i, . . . , s_j)$ and $σ_{k}(0 ≤ k ≤ |σ|)$ denotes $(s_k, . . . , s_{|σ|})$. Further, the concatenation $⋅$ of two intervals $σ$ and $σ'$ is defined as follows:

$$σ · σ' = \begin{cases} σ, & \text{if } |σ| = ω \\ (s₀, . . . , sᵢ, sᵢ₊₁, . . .) & \text{if } σ = (s₀, . . . , sᵢ), σ' = (sᵢ₊₁, . . .), i \in N₀ \end{cases}$$
And the fusion of two intervals $\sigma$ and $\sigma'$ is also defined as below,

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

Moreover, $\sigma^\omega$ means infinitely many copies of interval $\sigma$ are concatenated, while $\sigma^{\omega\omega}$ denotes that infinitely many copies of $\sigma$ are fused together. In particular, $\sigma \circ \sigma$ requires $s_0 = s_1$ if $\sigma = \langle s_0, \ldots, s_i \rangle$.

Let $\sigma = \langle s_0, s_1, \ldots, s_{|\sigma|} \rangle$ be an interval and $r_1, \ldots, r_n$ be integers ($h \geq 1$) such that $0 \leq r_1 \leq r_2 \leq \ldots \leq r_h \leq |\sigma|$. The projection of $\sigma$ 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,

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

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 $\omega$. 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

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

$$\sigma \downarrow (r_1, \ldots, r_{h-1}, r_h) = \sigma \downarrow (r_1, \ldots, r_{h-1}) \text{ if } r_h = \omega$$

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 $I = (\sigma, k, j)$, where $\sigma$ is an interval, $k$ is an integer, and $j$ an integer or $\omega$ such that $k \leq j \leq |\sigma|$. 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 $\sigma$.
can be illustrated in construct, the repeated occurrence for some part in the l.h.s of \( \text{prj} \) operator. For instance, semantics of \((P_1, P_2^n) \text{ prj } Q\) can be illustrated in Fig. 3.

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

\[
\begin{align*}
1 - \text{prop} & \quad t \models p \text{ iff } s_k[p] = \text{true}, \text{ for any given proposition } p \\
1 - \text{not} & \quad t \models \neg p \text{ iff } t \not\models p \\
1 - \text{or} & \quad t \models p \lor q \text{ iff } t \models p \text{ or } t \models q \\
1 - \text{next} & \quad t \models \Box p \text{ iff } k < j \text{ and } (\sigma, k + 1, j) \models p \\
1 - \text{prj} & \quad t \models (P_1, \ldots, P_m) \text{ prj } Q \text{ iff there exist integers } k = r_0 \leq r_1 \leq \ldots \leq r_m \\
& \quad \leq j \text{ such that } (\sigma, r_{i-1}, r_i) \models P_i, 1 \leq i \leq m, \text{ and } (\sigma', 0, |\sigma'|) \models Q \\
& \quad \text{for one of the following } \sigma' : \\
& \quad (a) \quad r_m < j \text{ and } \sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_{m+1}, j)} \text{ or} \\
& \quad (b) \quad r_m = j \text{ and } \sigma' = \sigma \downarrow (r_0, \ldots, r_h) \text{ for some } 0 \leq h < m \\
1 - \text{prj}^+ & \quad t \models (P_1, \ldots, (P_i, \ldots, P_j) \cdots, P_m) \text{ prj } Q \text{ iff: } 1 \leq i \leq s \leq m \\
& \quad \text{and } \exists n \in \mathbb{N}, t \models (P_1, \ldots, (P_i, \ldots, P_j)^{(n)}, \ldots, P_m) \text{ prj } Q \\
& \quad \text{or: } s = m \text{ and there exist infinitely many integers } k = r_0 \leq r_1 \\
& \quad \leq \ldots \leq r_h \leq \omega, \text{ such that } (\sigma, r_{i-1}, r_i) \models P_i, 0 < l < i, \\
& \quad (\sigma, r_{i-1}, r_i) \models P_i, l \geq i, t = i + ((l - i) \mod (s - i + 1)), \\
& \quad \text{and } (\sigma', 0, |\sigma'|) \models Q, \text{ where } \sigma' = \sigma \downarrow (r_0, r_1, \ldots). 
\end{align*}
\]

Note that in \((P_1, \ldots, (P_i, \ldots, P_j) \cdots, P_m) \text{ prj } Q\), if \( i = 1 \) and \( s = m \), we have \((P_1, \ldots, P_m) \text{ prj } Q\); if \( i = s \), we obtain \((P_1, \ldots, (P_i) \cdots, P_m) \text{ prj } Q\); and if \( i = s = m = 1 \), we get \((P_1) \text{ prj } Q\). Fig. 2 shows the possible semantics of \((P_1, P_2) \text{ prj } Q\). 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 \( \text{prj} \) construct, the \( \text{prj}^+ \) construct permits the repeated occurrence for some part in the l.h.s of \( \text{prj} \) operator. For instance, semantics of \((P_1, P_2^n) \text{ prj } Q\) can be illustrated in Fig. 3.
2.3. Satisfaction and validity

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

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 $\Sigma$ is an infinite sequence $w = a_0 a_1 \ldots$ of symbols, for each $i, a_i \in \Sigma$. A run of $B$ over an infinite word $w = a_0 a_1 \ldots$ is an infinite sequence $\rho = q_0 a_1 \ldots$ 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 $\rho$, and $\rho$ the run associated with $w$. The run $\rho$ 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 $\rho$ 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 $\Sigma$ with a fixed predefined null-label $\epsilon$, representing a no-op operation that is always executable and has no effect. For a Büchi automaton $B$, the stutter extension of finite run $\rho$ with final state $q_k$ is the $\omega$-run $\rho$ such that $q_0^\omega$ is the suffix of $\rho$ such that $(q_n, \epsilon, q_n) \in \delta$. The final state of the run can be thought to repeat null action $\epsilon$ 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 acceptability. 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,

\[
\text{ERE} \quad R ::= \emptyset | \epsilon | r | R + R | R \cdot R | R^* | R^+ \]

where $r \in \Upsilon$, $\epsilon$ denotes an empty string; $+, \cdot$ and $*$ are union, concatenation and Kleene (star) closure respectively; $x^\omega$ 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, $a_0 a_1 \ldots a_i \ldots$, where each $a_i$ is chosen from the alphabet $\Upsilon$. 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 $\omega$. For two strings $w$ and $w'$, $w \cdot w'$, $w^*$ and $w^\omega$ are defined as follows,

\[
w \cdot w' = \begin{cases}
w, & \text{if } |w| = \omega \\
q_0 a_0 \ldots a_i a_{i+1} \ldots, & \text{if } w \text{ is finite and } w = a_0 \ldots a_i
\end{cases}
\]

\[
w^\omega = \begin{cases}
w, & \text{if } |w| = \omega \\
q_0 a_0 \ldots a_i a_{i+1} \ldots a_i, & \text{if } w \text{ is finite and } w = a_0 \ldots a_i
\end{cases} \text{ $\omega$ times}
\]

\[
w^* = \begin{cases}
w, & \text{if } |w| = \omega \\
(q_0 a_0 \ldots a_i a_{i+1} \ldots a_i | n \in N_0), & \text{if } w \text{ is finite and } w = a_0 \ldots a_i
\end{cases} \text{ $n$ times}
\]

\[
w^+ = \begin{cases}
w, & \text{if } |w| = \omega \\
(q_0 a_0 \ldots a_i a_{i+1} \ldots a_i | n \in N_0 \setminus \{0\}), & \text{if } w \text{ is finite and } w = a_0 \ldots a_i
\end{cases} \text{ $n$ times}
\]
Further, if $W$ and $W'$ are two sets of strings. Then $W \cdot W'$, $W^\omega$ and $W^*$ are defined as follows,

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

$W^\omega = \{w^\omega \mid w \in W\}$

$W^* = \bigcup_{w \in W} w^*$

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

$L(r_1) = \emptyset$

$L(r_2) = \{r\}$

$L(\epsilon) = \{\epsilon\}$

$L(R+R) = L(R) \cup L(R)$

$L(R \cdot R) = L(R) \cdot L(R)$

$L(R^\omega) = L(R)^\omega$

$L(R^n) = L(R)^n$

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 $Q_0$ of atomic propositions appearing in PPTL* formula $Q$, $|Q_0| = l$, alphabets $\Sigma$ and $\Upsilon$ corresponding to SBA and ERE are defined respectively. We first define sets $A_i$, $1 \leq i \leq l$, as follows,

$A_1 = \{\neg q_1, \ldots, \neg q_l\}$

Then, $\Sigma = \bigcup_{i=1}^3 A_i \cup \{true\} \cup \{\epsilon\}$, and $\Upsilon = \bigcup_{j=1}^3 A_j \cup \{true\}$. For instance, if $Q_0 = \{p_1, p_2, p_3\}$, it is obtained that,

$A_1 = \{\neg p_1, \neg p_2, \neg p_3\}$

$A_2 = \{\neg p_1, \neg p_2\}$

$A_3 = \{\neg p_1, \neg p_2, \neg p_3\}$

So,

$\Sigma = \bigcup_{i=1}^3 A_i \cup \{true\} \cup \{\epsilon\}$

$\Upsilon = \bigcup_{j=1}^3 A_j \cup \{true\}$

Obviously, for each $r \in \Upsilon$, $r$ is a set of atomic propositions or their negations, denoted by $\{\neg q_i, \ldots, \neg q_j\}$, where $1 \leq i \leq j \leq l$, 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 \( \neg (\text{true} \land \Box q) \lor p \land \Diamond q \).

![LNFG diagram](image-url)

Fig. 6. LNFG of \((p \land \Box q; \Box \Diamond q) \land (\Box r; \Diamond q)\).

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 \( G = (\text{CL}(P), \text{EL}(P), V_0, L = \{L_1, \ldots, L_m\}) \), where CL(P) and EL(P) is the set of nodes and edges respectively, \( V_0 \) is the set of initial nodes, each \( L_k \subseteq \text{CL}(P), 1 \leq k \leq m \), is the set of nodes with \( L_k \) labels. Actually, in an LNFG, a node \( v \in \text{CL}(P) \) denotes a PPTL* formula while an edge from node \( v_1 \) to \( v_2 \) is a tuple \((v_1, Q, v_2)\) where \( v_1 \) and \( v_2 \) are PPTL* formulas and \( Q \equiv \bigwedge_i \hat{q}_i \), \( \hat{q}_i \) is an atomic proposition, \( \neg \hat{q}_i \) denotes \( q_i \) or \( \neg q_i \). The following are examples of LNFGs.

**Example 1.** LNFG of formula \( P \equiv \neg (\text{true} \land \Box q) \lor p \land \Diamond q \).

As shown in Fig. 5, the LNFG of formula \( P \equiv \neg (\text{true} \land \Box q) \lor p \land \Diamond q \) is \( G = \{\text{CL}(P), \text{EL}(P), V_0, L\} \), where \( \text{CL}(P) = \{v_0, v_1, v_2, v_3, v_4\}; \text{EL}(P) = \{(v_0, \text{true}, v_1), (v_0, p, v_2), (v_1, q, v_1), (v_2, q, v_3), (v_2, q, v_4), (v_2, \text{true}, v_3), (v_3, \text{true}, v_4)\} \), \( V_0 = \{v_0\} \); and \( L = \emptyset \). □

**Example 2.** LNFG of \( P \equiv (p \land \Box q; \Box \Diamond q) \land (\Box r; \Diamond q) \).

LNFG \( G = (\text{CL}(P), \text{EL}(P), V_0, L = \{L_1, \ldots, L_m\}) \) of formula \( P \equiv (p \land \Box q; \Box \Diamond q) \land (\Box r; \Diamond q) \) is constructed as depicted in Fig. 6, where \( \text{CL}(P) = \{v_0, v_1\}, \text{EL}(P) = \{(v_0, p \land r, v_0), (v_0, p \land r, v_1), (v_1, p, v_1)\} \), \( V_0 = \{v_0\} \), \( L = \{L_1, L_2\}, L_1 = \{v_0, v_1\} \) and \( L_2 = \{v_0\} \). □

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 \((v_1, Q, v_2)\) forms a transition; there exists only one initial location, the root node; the set of accepting locations consists of \( \epsilon \) node and the nodes which can appear in infinite paths for infinitely many times. Given an LNFG \( G = (\text{CL}(P), \text{EL}(P), V_0, L = \{L_1, \ldots, L_m\}) \) of formula \( P \), an SBA, \( B = (Q, \Sigma, l, \delta, F) \), over an alphabet \( \Sigma \) can be constructed as follows.

- Sets of the locations \( Q \) and the initial locations \( l: Q = V, \) and \( l = \{v_0\} \).
- Transition \( \delta \): Let \( \hat{q}_k \) be an atomic proposition or its negation, and we define a function \( \text{atom}(\bigwedge_{k=1}^{\epsilon} \hat{q}_k) \) for picking up atomic propositions or their negations appearing in \( \bigwedge_{k=1}^{\epsilon} \hat{q}_k \) as follows,

\[
\text{atom} \text{(true) = true}
\]

\[
\text{atom}(\hat{q}_k) = \begin{cases} 
\{\hat{q}_k\}, & \text{if } \hat{q}_k \equiv q_k \ 1 \leq k \leq l \\
\{-\hat{q}_k\}, & \text{otherwise}
\end{cases}
\]

\[
\text{atom} \left( \bigwedge_{k=1}^{\epsilon} \hat{q}_k \right) = \text{atom}(\hat{q}_1) \cup \text{atom} \left( \bigwedge_{k=2}^{\epsilon} \hat{q}_k \right)
\]

For each \( e_i = (v_i, Q, v_{i+1}) \in E \), there exists \( v_{i+1} \in \delta(v_i, \text{atom}(Q_e)) \). For node \( \epsilon, \delta(\epsilon, \epsilon) = \{\epsilon\} \).
Table 2
Algorithm for obtaining SBAs from an LNFG.

```plaintext
Function LNFG-SBA(G)
/* precondition: G = (CL(P), EL(P), V₀, L = {L₁, . . . , Lₘ}) is the LNFG of
 PPTL* formula P */
/* postcondition: LNFG-SBA(G) computes an SBA B = (Q, Σ, I, δ, F) from G */
begin function
  Q = ∅; F = ∅; I = ∅;
  for each node v₁ ∈ V, add a state q₁ to Q, Q = Q ∪ {q₁};
  if v₁ is ε, F = F ∪ {q₁}; δ(q₁, ε) = {q₁};
  else if v₁ ∈ (L₁ ∪ · · · ∪ Lₘ), F = F ∪ {q₁};
  end for
  if q₀ ∈ V₀, I = I ∪ {q₀};
  for each edge e = (vᵢ, Pᵢ, vⱼ) ∈ E, qᵢ ∈ δ(qᵢ, atom(Pᵢ));
  end for
  return B = (Q, Σ, I, δ, F)
end function
```

![Fig. 7. Stutter-Büchi automaton of formula P ≡ ¬(true; ¬ ∪ q) ∨ p ∧ ∅q.](image)

- Accepting locations F: It has been proved that infinite paths with Inf(π) ⊈ Lᵢ for all 1 ≤ i ≤ m precisely characterize infinite models of P. This can be equivalently expressed by “infinite paths with Inf(π) ∩ Lᵢ ≠ ∅ for all 1 ≤ i ≤ m precisely characterize infinite models of P”, where Lᵢ denotes CL(P) \ Lᵢ. That is,

  \[ \text{Inf}(π) \cap (L₁ \cap · · · \cap Lₘ) \neq ∅ ⇔ \text{Inf}(π) \cap (L₁ \cup · · · \∪ Lₘ) \neq ∅ \]

  This precisely coincides with defining \( F = (L₁ \cup · · · \∪ Lₘ) \) 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, \( B = (Q, Σ, I, δ, F) \), from the LNFG in Example 1.

As depicted in Fig. 7, the set of locations, \( Q = \{q₀, q₁, q₂, q₃, q₄\} \), comes from V directly. The set of initial locations \( I = \{q₀\} \) is root node \( ν₀ \) in G. The set of the accepting locations \( F = \{q₁, q₃, q₄\} \) consists of nodes \( v₁, v₃ \) appearing in loops and ε node in V. The transitions, \( δ(q₀, a₀) = \{q₁\}, δ(q₀, a₂) = \{q₂\}, δ(q₁, a₁) = \{q₁\}, δ(q₂, a₁) = \{q₃, q₄\}, δ(q₃, a₀) = \{q₃, q₄\}, δ(q₄, a₂) = \{q₄\} \) 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 \( F = ∅ \).

### 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 \( ε \notin S \). □

From now on we shall often drop the concatenation symbol \( \bullet \), writing SX for \( S \cdot X \) 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 \( Q = \{q₀, . . . , qₙ\} \) and the starting location \( q₀ \). For \( 1 ≤ i ≤ n \), 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 \( qᵢ \); thus \( L(B) = L(X₀) \). We can write an equation for each \( Xᵢ \) in terms of the languages defined by its successor locations. For example, for the stutter-Büchi automaton \( B \) in Example 3, we have,

\[
\begin{align*}
(0) \quad X₀ &= a₀X₁ + a₂X₂ \\
(1) \quad X₁ &= a₁X₁ + a¹ᵠ \\
(2) \quad X₂ &= a₁X₄ + a₁X₃ \\
(3) \quad X₃ &= a₀X₃ + a₀X₄ + a₀⁺ \\
(4) \quad X₄ &= a₃X₄ + a₃⁺
\end{align*}
\]
Note that $X_1$, $X_3$, and $X_4$ contains $a_0^*$, $a_0^q$, and $a_0^q$ 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 $\varepsilon$,
\[
X_4 = a_3X_4 + a_0^a a_0^a a_0^a = \varepsilon
\]
Replacing $X_4$ in (3),
\[
X_3 = a_0 X_3 + a_0 X_4 + a_0^a = a_0 X_3 + a_0 + a_0^a = a_0^a a_0 + a_0^a a_0^a = a_0^a a_0
\]
Replacing $X_3$ and $X_4$ in (2),
\[
X_2 = a_1X_4 + a_1X_3 = [q] + [q]false \wedge true
\]
Finally, replacing $X_1$ and $X_2$ in (0), we have,
\[
X_0 = a_0 X_1 + a_0 X_2 = a_0[q]^\omega + a_2([q] + [q]false \wedge true)
\]
\[
\begin{align*}
\Omega(\sigma) = \begin{cases} 
\varepsilon & \text{if } |\sigma| = 0 \\
A(s_0) \ldots A(s_{j-1}) & \text{if } \sigma \text{ is finite and } \sigma = \langle s_0, \ldots, s_j \rangle, j \geq 1 \\
A(s_0) \ldots A(s_j) \ldots & \text{if } \sigma \text{ is infinite and } \sigma = \langle s_0, \ldots, s_j \ldots \rangle
\end{cases}
\end{align*}
\]

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 $\Omega(\sigma_1 \circ \sigma_2) = \Omega(\sigma_1) \bullet \Omega(\sigma_2)$, $\Omega(\sigma_1^\omega) = \Omega(\sigma_1)^\omega$ and $\Omega(\sigma^{\omega*}) = \Omega(\sigma)^{\omega*}$. $F_R$ is constructed inductively on the structure of $R$.

\[
\begin{align*}
F_0 & \text{ def } \text{false} \\
F_{\varepsilon} & \text{ def } \text{empty} \\
F_r & \text{ def } \begin{cases} 
\hat{p}_1 \wedge \cdots \wedge \hat{p}_j \wedge \text{skip} & \text{if } r = \langle \hat{p}_1, \ldots, \hat{p}_j \rangle, 1 \leq i \leq j \leq l \\
\text{true} \wedge \text{skip} & \text{if } r = \text{true}
\end{cases}
\end{align*}
\]

where $r \in \mathcal{Y}$. Inductively, if $R_1$ and $R_2$ are extended regular expressions, then
\[
\begin{align*}
F_{R_1 + R_2} & \text{ def } F_{R_1} \lor F_{R_2} \\
F_{R_1 \bullet R_2} & \text{ def } F_{R_1} ; F_{R_2} \\
F_{R^{\omega*}} & \text{ def } F_{R} \sqcup \square \text{true} \\
F_{R^*} & \text{ def } F_{R}^+
\end{align*}
\]

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

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

**Proof.** $\Rightarrow$: For any $\sigma \in \mathcal{Y}$, 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 $r \in \mathcal{Y}$, if $r = \langle \hat{p}_1, \ldots, \hat{p}_j \rangle$, $1 \leq i \leq j \leq l$, the unique model satisfying $F_r \equiv \hat{p}_1 \wedge \cdots \wedge \hat{p}_j \wedge \text{skip}$ is $\sigma = \langle \{\hat{p}_1, \ldots, \hat{p}_j\}, \text{true} \rangle$ (this means $A(s_0) = \{\hat{p}_1, \ldots, \hat{p}_j\}$ and $A(s_1) = \text{true}$). Clearly, $\Omega(\sigma) = \{\hat{p}_1, \ldots, \hat{p}_j\} = r \in L(r)$; otherwise, if $r = \text{true}$, the unique model satisfying $F_r \equiv \text{true} \wedge \text{skip}$ is $\sigma = \langle \text{true}, \text{true} \rangle$. Obviously, $\Omega(\sigma) = \text{true} = r \in L(r)$.

\(^2\) For finite state automata, $X_1$ contains $\varepsilon$ if $q_1$ is accepted.
2. For $\varepsilon$, the unique model satisfying $F_\varepsilon \equiv \text{empty is } \sigma = \langle \text{true} \rangle$. Since $|\sigma| = 0$, $\Omega(\sigma) = \varepsilon \in L(\varepsilon)$.
3. For $\emptyset$, no models can satisfy $F_\emptyset \equiv \text{false}$.

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

1. For $R_1 + R_2$, $F_{R_1 + R_2} \equiv F_{R_1} \lor F_{R_2}$. 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 $L_{R_1}$, $\Omega(\sigma) \in L_{R_1 + R_2}$.

2. For $R_1 \cdot R_2$, $F_{R_1 \cdot R_2} \equiv F_{R_1} \cdot F_{R_2}$. So, for any $\sigma_1, \sigma_2 \in \Gamma$, if $\sigma_1 \models F_{R_1}$ and $\sigma_2 \models F_{R_2}$, then $\sigma_1 \cdot \sigma_2 \models F_{R_1 \cdot R_2}$. By hypothesis, $\Omega(\sigma_1) \in L(R_1)$ and $\Omega(\sigma_2) \in L(R_2)$, so $\Omega(\sigma_1) \cdot \Omega(\sigma_2) \in L(R_1) \cdot L(R_2)$. Thus, by $L_{R_1 \cdot R_2}$, $\Omega(\sigma_1 \cdot \sigma_2) \in L_{R_1 \cdot R_2}$.

3. For $\neg R_1$, $F_{\neg R_1} \equiv F_{R_1} \land \square \text{more}$. So for any $\sigma_1 \in \Gamma$, if $\sigma_1 \models F_{R_1}$, then $\sigma_1^{\text{more}} \models F_{\neg R_1} \land \square \text{more}$. By hypothesis, $\Omega(\sigma) \in L(R_1)$, so $\Omega(\sigma^{\text{more}}) = \Omega(\sigma) \ominus \in L(R_1)^{\text{more}}$. Thus, by $L_{\neg R_1}$, $\Omega(\sigma^{\text{more}}) \in L_{R_1}^\ominus$.

4. For $R_1^*$, $F_{R_1}^* \equiv F_{R_1}^*$. So for any $\sigma \in \Gamma$, $\sigma \models F_{R_1}$, we have $\sigma^{\ast} \models F_{R_1}^*$. By hypothesis, $\Omega(\sigma) \in L(R_1)$, so $\Omega(\sigma^{\ast}) = \Omega(\sigma)^* \in L(R_1)^*$. By $L_{R_1^*}$, $\Omega(\sigma^{\ast}) \in L_{R_1}^*$.

$\iff$ For any $w \in L(R)$, there exists $\sigma \models F_{R}$, and $\Omega(\sigma) = \omega$. The proof also proceeds by induction on the structure of $R$.

**Base Case:**
1. For $r \in \Upsilon$, $L(r) = \{r\}$. If $r = \{p_1, \ldots, p_k\}$, $1 \leq i \leq j \leq l$, $F_i \equiv \hat{p}_i \ldots \hat{p}_j \land \text{skip}$, we have $\sigma = \langle \hat{p}_1, \ldots, \hat{p}_k \rangle$, true $\models F_r$ and $\Omega(\sigma) = \{\hat{p}_1, \ldots, \hat{p}_k\} = r \in L(r)$; otherwise if $r = \text{true}$, $F_i \equiv \text{true} \land \text{skip}$, we have $\sigma = \langle \text{true}, \text{true} \rangle = F_r$ and $\Omega(\sigma) = \text{true} = r \in L(r)$.
2. For $\varepsilon$, $L(\varepsilon) = \{\varepsilon\}$. Since $F_\text{true} \equiv \text{true}$, we have $\sigma = \langle \text{true} \rangle \models F_r$, and $\Omega(\sigma) = \varepsilon \in L(\varepsilon)$.
3. For $\emptyset$, $L(\emptyset) = \emptyset$. Since $F_\emptyset \equiv \text{false}$, 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 $\Omega(\sigma_1) = \omega$, and for any $w_2 \in L(R_2)$, there exists $\sigma_2 \in \Gamma$, $\sigma_2 \models F_{R_2}$ and $\Omega(\sigma_2) = \omega$. 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 $\Omega(\sigma) = \omega$, or $\sigma \models F_{R_2}$ and $\Omega(\sigma) = \omega$. Thus, we have for any $w \in L(R_1) \cup L(R_2)$, there exists $\sigma \models F_{R_1} \lor F_{R_2}$ and $\Omega(\sigma) = \omega$.

2. For $R_1 \cdot R_2$, $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$. For any $w \in L(R_1) \cdot L(R_2)$, there exist $w_1, w_2, w = w_1 \cdot w_2$, $w_1 \in L(R_1)$ and $w_2 \in L(R_2)$. By hypothesis, there exists $\sigma \in \Gamma$, $\sigma_1 \models F_{R_1}$ and $\Omega(\sigma_1) = \omega$, and there exists $\sigma_2 \in \Gamma$, $\sigma_2 \models F_{R_2}$ and $\Omega(\sigma_2) = \omega$. Thus, we have $\sigma = \sigma_1 \cdot \sigma_2 \models F_{R_1} \cdot F_{R_2} = F_{R_1 \cdot R_2}$, and $\Omega(\sigma) = \omega$. Thus, for any $w \in L(R_1)$, $w_1 \models F_{R_1}$, $\Omega(w_1) = \omega$.

3. For $\neg R_1$, $L(\neg R_1) = L(R_1)^{\text{more}}$. For any $w \in L(R_1)^{\text{more}}$, there exists $w_1 \in L(R_1)$, and $w_1^{\text{more}} = w$. By hypothesis, there exists $\sigma_1 \in \Gamma$, $\sigma_1 \models F_{R_1}$, and $\Omega(\sigma_1) = \omega$. Thus we have $\sigma = \sigma_1 \models F_{R_1}$, $\Omega(\sigma) = \omega$. So in the case of $\neg R_1$, $\Omega(\sigma^{\text{more}}) = \Omega(\sigma) = \omega$.

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 $\Omega(\sigma_1) = \omega$. Thus, we have $\sigma = \sigma_1^* \models F_{R_1}^*$, and $\Omega(\sigma) = \Omega(\sigma^{\ast}) = \Omega(\sigma_1^*) = w_1^* = w$. □

**Example 4.** Constructing PPTL* formula from the extended regular expression, true[q]^ω + [p][q] + [p][q]true^true obtained in Example 3.

\[
F_{\text{true}[q]^\omega + [p][q] + [p][q]\text{true}^\true} \equiv F_{\text{true}[q]^\omega} \lor F_{[p][q]} \lor F_{[p][q]\text{true}^\true}
\]
\[
\equiv F_{\text{true}}; F_{[p][q]} \lor F_{[p][q]} \lor F_{[p][q]}; F_{\text{true}^\true}; F_{\text{true}}
\]
\[
= \text{true} \land \text{skip}; F_{[p]}; F_{\text{true}^\true} \land \square \text{more} \lor p \land \text{skip}; q \land \text{skip} \lor p \land \text{skip}; q \land \text{skip};
\]
\[
\land \text{true} \land \text{skip}\;
\]
\[
= \text{true} \land \text{skip}; (q \land \text{skip})^\true \land \square \text{more} \lor p \land \text{skip}; q \land \text{skip} \lor p \land \text{skip}; q \land \text{skip};
\]
\[
\land \text{true} \land \text{skip}\;
\]

6. Characterizing fragments of PPTL*

The basic temporal operators in PPTL* are next, chop, chop*, chop^+, prj, prj^+, prj^#). 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 $L(\text{next}, \text{chop})$ 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 $L(\text{next}, \text{chop}, \text{chop}^*)$, $L(\text{chop}, \text{chop}^*)$ and $L(\text{next}, \text{prj})$, for some others obviously share the same characters with the five fragments.
6.1. Characterization of $L(\text{next}, \text{chop})$

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

Actually, $L(\text{next}, \text{chop})$ has the same expressiveness as $L(\text{next}, \text{until})$, 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 $\Gamma$, $\Upsilon$ and $\Omega$ 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 $\emptyset$, $\epsilon$ and the alphabet symbols $r \in \Upsilon$, and $\bullet$, $+$, $\ast$ and $\sim$ (complementation). The star-free regular expressions are general regular expressions without occurrences of Kleene closure [23].

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

<table>
<thead>
<tr>
<th>star-free RE</th>
<th>$\mathcal{R}$ ::= $\emptyset$</th>
<th>$r$</th>
<th>$\mathcal{R} \cup \mathcal{R}$</th>
<th>$\mathcal{R} \bullet \mathcal{R}$</th>
<th>$\sim \mathcal{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\mathcal{R}_1}$</td>
<td>$L(\emptyset)$</td>
<td>$\emptyset$</td>
<td>$L(r)$</td>
<td>${r}$</td>
<td>$L(\mathcal{R}) \cup L(\mathcal{R})$</td>
</tr>
<tr>
<td>$L_{\mathcal{R}_2}$</td>
<td>$L(\epsilon)$</td>
<td>${\epsilon}$</td>
<td>$L(\mathcal{R} \bullet \mathcal{R})$</td>
<td>$L(\mathcal{R}) \bullet L(\mathcal{R})$</td>
<td>$\Upsilon^* \setminus L(\mathcal{R})$</td>
</tr>
</tbody>
</table>

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.** $L(\text{next}, \text{chop})$ 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 $L(\text{next}, \text{chop})$ and star-free regular expressions.

(i) For any star-free expression $\mathcal{R}$, a formula $F_\mathcal{R}$ in $L(\text{next}, \text{chop})$ 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 $\sigma \in \Gamma$, $\sigma \not\models F_\mathcal{R}$, $\Omega(\sigma) \notin L(\mathcal{R})$. $F_\mathcal{R}$ is constructed inductively on the structure of $\mathcal{R}$ as follows.

- $F_{\emptyset} \overset{\text{def}}{=} \text{false}$
- $F_\epsilon \overset{\text{def}}{=} \text{empty}$
- $F_r \overset{\text{def}}{=} \{ \hat{p}_1 \land \cdots \land \hat{p}_j \land \text{skip}, \text{ if } r = \{\hat{p}_i \}, \quad 1 \leq i \leq j \leq l \}
- \quad \land \text{true} \land \text{skip}, \text{ if } r = \text{true}$

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

- $F_{\mathcal{R}_1 \cup \mathcal{R}_2} \overset{\text{def}}{=} F_{\mathcal{R}_1} \lor F_{\mathcal{R}_2}$
- $F_{\mathcal{R}_1 \bullet \mathcal{R}_2} \overset{\text{def}}{=} F_{\mathcal{R}_1} ; F_{\mathcal{R}_2}$
- $F_{\sim \mathcal{R}_1} \overset{\text{def}}{=} \neg F_{\mathcal{R}_1}$

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 $\sigma \in \Gamma$, $\sigma \not\models F_{\sim \mathcal{R}}$, $\Omega(\sigma) \notin L(\sim \mathcal{R})$.

$\Rightarrow$: For any $w \in L(\sim \mathcal{R})$, by $L_{\mathcal{R}_4}$, $w \in \Upsilon^* \setminus L(\mathcal{R})$. 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 $\Omega(\sigma) = w$. 

Fig. 8. Transformations among PLTL, first order logic, counter-free Büchi automata and star-free regular expressions.
For any $\sigma \in \Gamma$, if $\sigma \models -F_R$, we have $\sigma \not\models F_R$. By Theorem 2, $\Omega(\sigma) \not\in L(R)$. So $\Omega(\sigma) \in \Upsilon^* \setminus L(R)$. By $L_{\mathbb{R}}$, $\Omega(\sigma) \in L(\sim R)$.

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

\[
P \in \text{Prop} : \quad \text{for any } \sigma \models \Gamma, \sigma \models p, \Theta(\sigma) = \{p\}, \{p\} \in \Upsilon;
\]

\[
\neg P : \quad \text{for any } \sigma \models \Gamma, \sigma \models \neg P, \Theta(\sigma) = \neg \Theta(\sigma'), \text{where } \sigma' \models P;
\]

\[
P_1 \lor P_2 : \quad \text{for any } \sigma \models \Gamma, \sigma \models P_1 \lor P_2, \Theta(\sigma) = \Theta(\sigma_1) + \Theta(\sigma_2), \sigma_1 \models P_1, \sigma_2 \models P_2;
\]

\[
\bigcirc P : \quad \text{for any } \sigma \models \Gamma, \sigma \models \bigcirc P, \Theta(\sigma) = \text{true} \bullet \Theta(\sigma'), \text{where } \sigma' \models P;
\]

\[
P_1; P_2 : \quad \text{for any } \sigma \models \Gamma, \sigma \models P_1; P_2, \Theta(\sigma) = \Theta(\sigma_1)' \bullet (r_1 \cup r_2) \bullet \Theta(\sigma_2)', \text{where } \Theta(\sigma_1)' \bullet r_1 = \Theta(\sigma_1), r_2 \bullet \Theta(\sigma_2)' = \Theta(\sigma_2), \text{and } \sigma_1 \models P_1, \sigma_2 \models P_2, \sigma = \sigma_1 \circ \sigma_2.
\]

$R_P$ is constructed inductively on the structure of formulas in $L(\text{next}, \text{chop})$ as follows.

\[
\begin{align*}
\text{Proposition} & \quad R_P \overset{\text{def}}{=} \{p\}, \{p\} \in \Upsilon; \\
\text{Or} & \quad R_{P_1 \lor P_2} \overset{\text{def}}{=} R_{P_1} + R_{P_2} \\
\text{Negation} & \quad R_{\neg P} \overset{\text{def}}{=} \sim R_P \\
\text{Next} & \quad R_{\bigcirc P} \overset{\text{def}}{=} \text{true} \bullet R_P \\
\text{Chop} & \quad R_{P_1; P_2} \overset{\text{def}}{=} R'_{P_1} \bullet (r_1 \cup r_2) \bullet R'_{P_2}, \text{where } R'_{P_1} \bullet r_1 = R_{P_1}, r_2 \bullet R'_{P_2} = R_{P_2}.
\end{align*}
\]

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

Theorem 4. To express chop construct in $L(\text{next}, \text{until})$, 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,

\[
P; Q \equiv R_{e_1} \wedge \text{empty} \lor \bigvee_i (R_{e_2} \wedge \bigcirc R)
\]

where $R_{e_1}$ and $R_{e_2}$ 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,

\[
P \equiv P_e \wedge \text{empty} \lor \bigvee_i (P_i \wedge \bigcirc P_i')
\]

\[
Q \equiv Q_e \wedge \text{empty} \lor \bigvee_j (Q_j \wedge \bigcirc Q_j')
\]

It turns out that,

\[
P; Q \equiv (P_e \wedge \text{empty} \lor \bigvee_i (P_i \wedge \bigcirc P_i')); Q
\]

\[
\equiv P_e \wedge Q \lor \bigvee_i (P_i \wedge \bigcirc P_i'); Q
\]

\[
\equiv P_e \wedge Q \lor \bigvee_i P_i \wedge \bigcirc (P_i'; Q)
\]

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. $\Box$
Therefore, we can conclude that with the help of the \textit{chop} operator, \(L(next, until)\) will be a non-elementary succincter. Thus, even though the complexity for model checking of logic with the \textit{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 \(L(next, \text{chop, chop}^*)\)

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

\textbf{Theorem 5.} \(L(next, \text{chop, chop}^*)\) has the same expressiveness as extended regular expressions.

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

(i) For any regular expression \(R\), a formula \(F_R\) in \(L(next, \text{chop, chop}^*)\) 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,

\[
F_n \overset{\text{def}}{=} \text{false} \\
F_e \overset{\text{def}}{=} \text{empty} \\
F_r \overset{\text{def}}{=} \begin{cases} \tilde{p}_1 \land \cdots \land \tilde{p}_j \land \text{skip}, & \text{if } r = [\tilde{p}_1, \ldots, \tilde{p}_j], 1 \leq i \leq j \leq l \\ \text{true} \land \text{skip}, & \text{if } r = \text{true} \end{cases}
\]

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

\[
F_{R_1+R_2} \overset{\text{def}}{=} F_{R_1} \lor F_{R_2} \\
F_{R_1 \cdot R_2} \overset{\text{def}}{=} F_{R_1} \land F_{R_2} \\
F_{R^\omega} \overset{\text{def}}{=} F^*_R \land \text{more} \\
F_{R_1^\ast} \overset{\text{def}}{=} F^*_R
\]

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

(ii) For any formula \(P\) in \(L(next, \text{chop, chop}^*)\), an 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 \(\sigma \in \Gamma, \sigma \models P, \Theta(\sigma) = w\). Note that, for \textit{chop star} construct, \(\Theta\) is defined below.

For any \(\sigma \in \Gamma, \sigma \models P^\ast, \Theta(\sigma) = \varepsilon + r_1 \bullet (\Theta(\sigma') \bullet (r_2 \cup r_1))^\ast \bullet \Theta(\sigma') \bullet r_2,\) where \(r_1 \bullet \Theta(\sigma') \bullet r_2 = \Theta(\sigma'),\) and \(\sigma' \models P\). \(R_P\) is constructed inductively as follows,

\[
\begin{align*}
\text{Proposition} & \quad R_p \overset{\text{def}}{=} \{p\}, \{p\} \in \Upsilon \\
\text{Or} & \quad R_{p_1 \lor p_2} \overset{\text{def}}{=} R_{p_1} \lor R_{p_2} \\
\text{Negation} & \quad R_{\lnot p} \overset{\text{def}}{=} \sim R_p \\
\text{Next} & \quad R_{\circ P} \overset{\text{def}}{=} \text{true} \bullet R_P \\
\text{Chop} & \quad R_{p_1:p_2} \overset{\text{def}}{=} R_{p_1'} \bullet (r_1 \cup r_2) \bullet R_{p_2}', \text{where } R_{p_1'} \bullet r_1 = R_{p_1}, r_2 \bullet R_{p_2}' = R_{p_2} \\
\text{Chop star} & \quad R_{p^\ast} \overset{\text{def}}{=} \varepsilon + r_1 \bullet (R_{p^\ast} \bullet (r_2 \cup r_1))^\ast \bullet R_{p^\ast} \bullet r_2, \text{where } R_P = r_1 \bullet R_{p^\ast} \bullet r_2
\end{align*}
\]

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

6.3. Characterizations of \(L(\text{chop})\) and \(L(\text{chop, chop}^+)\)

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

\textbf{Theorem 6.} \(L(\text{chop})\) has the same expressiveness as star-free regular expressions without \(\varepsilon\).

\textbf{Proof.} The proof consists of a pair of transformations between the counter-parts of the operators in \(L(\text{chop})\) and star-free regular expressions without \(\varepsilon\).

(i) For any star-free expression \(R\) without \(\varepsilon\), a formula \(F_R\) in \(L(\text{chop})\) 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 \(\sigma \in \Gamma, \sigma \models F_R, \Omega(\sigma) \in L(R)\), \(F_R\) can be inductively constructed as follows,
The proof also consists of a pair of transformations between the counter-parts of the operators in $L(\text{chop})$ and regular expressions without $\varepsilon$.

**Theorem 7.** $L(\text{chop}, \text{chop}^+)$ has the same expressiveness as regular expressions without $\varepsilon$. $\Box$

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

(i) For any regular expression $R$ without $\varepsilon$, a formula $F_R$ in $L(\text{chop}, \text{chop}^+)$ can be constructed such that for any word $w \in L(R)$, there exists $\sigma \in \Gamma, \sigma \models P$, and $\Theta(\sigma) = w$. $F_R$ can inductively be constructed as follows, where $F_R = F_{R_1 \cdot R_2}$.

(ii) For any formula $P$ in $L(\text{chop}, \text{chop}^+)$, a regular expression without $\varepsilon$, $R_P$, can be constructed such that for any $\sigma \in \Gamma, \sigma \models P$, and $\Theta(\sigma) = w$. $R_P$ can inductively be constructed as follows, where $R_P = R_{P_1 \cdot P_2}$.

The proofs for the correctness of the transformations are straightforward by induction on the structure of formulas in $L(\text{chop})$ and expressions of star-free expressions without $\varepsilon$. $\Box$

6.4. Characterizations of other fragments

Since chop and chop star can be subsumed by projection and projection star constructs respectively. Thus, obviously, $L(\text{next}, \text{chop}, \text{chop}^+), L(\text{next}, \text{chop}^+, \text{prj}), L(\text{next}, \text{chop}, \text{prj}^+), L(\text{next}, \text{chop}^+, \text{prj}), L(\text{next}, \text{chop}, \text{chop}^+, \text{prj}^+)$, and the full logic $L(\text{Next}, \text{Chop}, \text{chop}^+, \text{prj}, \text{prj}^+)$ have the same expressiveness as full regular expressions.
Further, the expressiveness of \( L(\text{next}, \text{chop}, \text{prj}) \) and \( L(\text{next}, \text{prj}) \) are equal, and will have the following unprecise expressive power.

\[
L(\text{next, chop}) \subseteq L(\text{next, prj}) \subseteq L(\text{next, chop}, \text{chop}^*)
\]  

(6.1)

For projection construct, \((P_1, \ldots, P_m) \text{proj} Q\), if \(Q \equiv \text{empty}\), and \(P_1 \equiv P_2 \equiv \ldots \equiv P_m \equiv P\), it is obtained,

\[
(P_1, \ldots, P_m) \text{proj} Q \equiv P^m, \quad m \in N_0
\]  

(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 \(P^\omega\) cannot be specified by \(P^m\). Therefore, we can conclude that,

\[
L(\text{next, chop}) \subseteq L(\text{next, prj}) \subseteq L(\text{next, chop}, \text{chop}^*)
\]  

(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 \(\epsilon\), star-free regular language, regular language without \(\epsilon\), \(\omega\)-free regular language and full regular language for PPTL* and its fragments are obtained as shown in Fig. 10. And \(L(\text{chop})\) has the expressiveness of star-free regular language without \(\epsilon\), \(L(\text{next, chop})\) is in the class of star-free regular language, \(L(\text{chop}, \text{chop}^*)\) is in the same expressiveness with regular language without \(\epsilon\), \(L(\text{next, prj})\) has the same express power as star-free regular language, and \(L(\text{next, chop}, \text{chop}^*), L(\text{next, chop}^*, \text{prj}), L(\text{next, chop, prj}^*), L(\text{next, prj, prj}^*), L(\text{next, chop, chop}^*, \text{prj}), L(\text{next, chop, chop}^*, \text{prj}^*), L(\text{next, chop, chop}^*, \text{prj}^*), L(\text{next, chop, chop}^*, \text{prj}^*), L(\text{next, chop, chop}^*, \text{prj}^*), L(\text{next, chop, chop}^*, \text{prj}^*)\), as well as the full logic \(L(\text{next, chop, chop}^*, \text{prj}, \text{prj}^*)\) have the same expressiveness as full regular language.

6.5. Characterization of propositional interval temporal logic

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

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

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