First publ. in: Automated technology for verification and analysis: 7th International Symposium, ATVA 2009, Macao, China, October 14 - 16, 2009; proceedings / Zhiming Liu ... (eds.). Heidelberg; Berlin: Springer, 2009, pp. 244-254 (=Lecture Notes in Computer Science ; vol. 5799)

Specification Languages for Stutter-Invariant Regular Properties*

Christian Dax[1], Felix Klaedtke[1], and Stefan Leue[2]

[1] ETH Zurich, Switzerland
[2] University of Konstanz, Germany

Abstract. We present specification languages that naturally capture exactly the regular and ω-regular properties that are stutter invariant. Our specification languages are variants of the classical regular expressions and of the core of PSL, a temporal logic, which is widely used in industry and which extends the classical linear-time temporal logic LTL by semi-extended regular expressions.

1 Introduction

Stutter-invariant specifications do not distinguish between system behaviors that differ from each other only by the number of consecutive repetitions of the observed system states. Stutter invariance is crucial for refining specifications and for modular reasoning [13]. Apart from these conceptual reasons for restricting oneself to stutter-invariant specifications, there is also a more practical motivation: stuttering invariance is an essential requirement for using partial-order reduction techniques (see, e.g., [2, 11, 15, 16, 20]) in finite-state model checking.

Unfortunately, checking whether an LTL formula or an automaton describes a stutter-invariant property is PSPACE-complete [18]. To leverage partial-order reduction techniques in finite-state model checking even when it is unknown whether the given property is stutter-invariant, Holzmann and Kupferman [12] suggested to use a stutter-invariant overapproximation of the given property. However, if the given property is not stutter-invariant, we might obtain counterexamples that are false positives. Moreover, the overapproximation of the property blows up the specification and decelerates the model-checking process.

Another approach for avoiding the expensive check whether a given property is stutter-invariant, is to use specification languages that only allow one to specify stutter-invariant properties. For instance, LTL without the next operator X, LTL$_{-\mathsf{X}}$ for short, captures exactly the stutter-invariant star-free properties [10, 17]. An advantage of such a syntactic characterization is that it yields a sufficient and easily checkable condition whether partial-order reduction techniques are applicable. However, LTL$_{-\mathsf{X}}$ is limited in its expressive power.

Independently, Etessami [9] and Rabinovich [19] gave similar syntactic characterizations of the stutter-invariant ω-regular properties. However, these characterizations are not satisfactory from a practical point of view. Both extend

* Partly supported by the Swiss National Science Foundation.

fragments of $\mathrm{LTL}_{-\mathsf{X}}$ by allowing one to existentially quantify over propositions. To preserve stutter invariance the quantification is semantically restricted. Due to this restriction, the meaning of quantifying over propositions becomes unintuitive and expressing properties in the proposed temporal logics becomes difficult. Note that even the extension of LTL with the standard quantification over propositions is considered as difficult to use in practice [21]. Another practical drawback of the temporal logic in [19] is that the finite-state model-checking problem has a non-elementary worst-case complexity. The finite-state model-checking problem with the temporal logic in [9] remains in PSPACE, as for LTL. This upper bound on the complexity of the model-checking problem is achieved by additionally restricting syntactically the use of the non-standard quantification over propositions. The downside of this restriction is that the logic is not syntactically closed under negation anymore, which can make it more difficult or even impossible to express properties naturally and concisely in it. Expressing the complement of a property might lead to an exponential blow-up.

In this paper, we give another syntactic characterization in terms of a temporal logic of the ω-regular properties that are stutter invariant. Our characterization overcomes the limitations of the temporal logics from [9] and [19]. Namely, it is syntactically closed under negation, it is easy to use, and the finite-state model-checking problem with it is solvable in practice. Furthermore, we also present a syntactic characterization of the stutter-invariant regular properties. Our characterizations are given as variants of the classical regular expressions and the linear-time core of the industrial-strength temporal logic PSL [1], which extends LTL with semi-extended regular expressions (SEREs). We name our variants siSEREs and siPSL, respectively. Similar to PSL, siPSL extends $\mathrm{LTL}_{-\mathsf{X}}$ with siSEREs. For siSEREs, the use of the concatenation operator and the Kleene star is syntactically restricted. Moreover, siSEREs make use of a novel iteration operator, which is a variant of the Kleene star.

2 Preliminaries

Words. For an alphabet Σ, we denote the set of finite and infinite words by Σ^* and Σ^ω, respectively. Furthermore, we write $\Sigma^\infty := \Sigma^* \cup \Sigma^\omega$ and $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$, where ε denotes the empty word. The concatenation of words is written as juxtaposition. The *concatenation* of the languages $K \subseteq \Sigma^*$ and $L \subseteq \Sigma^\infty$ is $K ; L := \{uv : u \in K \text{ and } v \in L\}$, and the *fusion* of K and L is $K : L := \{ubv \in \Sigma^* : b \in \Sigma,\ ub \in K,\ \text{and } bv \in L\}$. Furthermore, for $L \subseteq \Sigma^*$, we define $L^* := \bigcup_{n\geq 0} L^n$ and $L^+ := \bigcup_{n\geq 1} L^n$ with $L^0 := \{\varepsilon\}$ and $L^{i+1} := L;L^i$, for $i \in \mathbb{N}$. We write $|w|$ for the length of $w \in \Sigma^\infty$ and we denote the $(i+1)$st letter of w by $w(i)$, where we assume that $i < |w|$. For a word $w \in \Sigma^\omega$ and $i \geq 0$, we define $w^{\geq i} := w(i)w(i+1)\dots$ and $w^{\leq i} := w(0)\dots w(i)$.

Stutter-Invariant Languages. Let us recall the definition of stutter invariance from [18]. The *stutter-removal operator* $\natural : \Sigma^\infty \to \Sigma^\infty$ maps a word $v \in \Sigma^\infty$ to the word that is obtained from v by replacing every maximal finite substring of identical letters by a single copy of the letter. For instance, $\natural(aabbbccc) = abc$,

$\sharp(aab(bbc)^\omega) = a(bc)^\omega$, and $\sharp(aabbbccc^\omega) = abc^\omega$. A language $L \subseteq \Sigma^\infty$ is *stutter-invariant* if $u \in L \Leftrightarrow v \in L$, for all $u, v \in \Sigma^\infty$ with $\sharp(u) = \sharp(v)$. A word $w \in \Sigma^\infty$ is *stutter free* if $w = \sharp(w)$. For $L \subseteq \Sigma^\infty$, we define $L_\sharp := \{\sharp(w) : w \in L\}$.

Propositional Logic. For a set of propositions P, we denote the set of *Boolean formulas* over P by $\mathcal{B}(P)$, i.e., $\mathcal{B}(P)$ consists of the formulas that are inductively built from the propositions in P and the connectives $\wedge$ and $\neg$. For $M \subseteq P$ and $b \in \mathcal{B}(P)$, we write $M \models b$ iff b evaluates to true when assigning true to the propositions in M and false to the propositions in $P \setminus M$.

Semi-extended Regular Expressions. The syntax of *semi-extended regular expressions* (SEREs) over the proposition set P is defined by the grammar

$$r ::= \varepsilon \mid b \mid r^* \mid r\,;r \mid r : r \mid r \cup r \mid r \cap r\,,$$

where $b \in \mathcal{B}(P)$. We point out that in addition to the concatentation operator ;, SEREs have the operator : for expressing the fusion of two languages. The language of an SERE over P is inductively defined:

$$L(r) := \begin{cases} \{\varepsilon\} & \text{if } r = \varepsilon, \\ \{b \in 2^P : b \models r\} & \text{if } r \in \mathcal{B}(P), \\ L(s) \star L(t) & \text{if } r = s \star t, \\ \big(L(s)\big)^* & \text{if } r = s^*, \end{cases}$$

where $\star \in \{;, :, \cup, \cap\}$. The *size* of an SERE is its syntactic length, i.e., $\|\varepsilon\| := 1$, $\|b\| := 1$, for $b \in \mathcal{B}(P)$, $\|r \star s\| := 1 + \|r\| + \|s\|$, for $\star \in \{\cup, \cap, ;, :\}$, and $\|r^*\| := 1 + \|r\|$.

Propositional Temporal Logic. The core of the linear-time fragment of PSL [1] is as follows. Its syntax over the set P of propositions is given by the grammar

$$\varphi ::= p \mid \mathsf{cl}(r) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid r \diamondsuit\!\!\rightarrow \varphi\,,$$

where $p \in P$ and r is an SERE over P. A PSL formula[1] over P is interpreted over an infinite word $w \in (2^P)^\omega$ as follows:

$w \models p$ iff $p \in w(0)$
$w \models \mathsf{cl}(r)$ iff $\exists k \geq 0 \colon w^{\leq k} \in L(r)$ or $\forall k \geq 0 \colon \exists v \in L(r) \colon w^{\leq k}$ is a prefix of v
$w \models \varphi \wedge \psi$ iff $w \models \varphi$ and $w \models \psi$
$w \models \neg\varphi$ iff $w \not\models \varphi$
$w \models \mathsf{X}\varphi$ iff $w^{\geq 1} \models \varphi$
$w \models \varphi \,\mathsf{U}\, \psi$ iff $\exists k \geq 0 \colon w^{\leq k} \models \psi$ and $\forall j < k \colon w^{\geq j} \models \varphi$
$w \models r \diamondsuit\!\!\rightarrow \varphi$ iff $\exists k \geq 0 \colon w^{\leq k} \in L(r)$ and $w^{\geq k} \models \varphi$

The *language* of a PSL formula φ is $L(\varphi) := \{w \in (2^P)^\omega : w \models \varphi\}$. As for SEREs, we define the *size* of a PSL formula as its syntactic length. That means, $\|p\| := 1$, $\|\mathsf{cl}(r)\| := 1 + \|r\|$, $\|\neg\varphi\| := \|\mathsf{X}\varphi\| := 1 + \|\varphi\|$, $\|\varphi \wedge \psi\| := \|\varphi \,\mathsf{U}\, \psi\| := 1 + \|\varphi\| + \|\psi\|$, and $\|r \diamondsuit\!\!\rightarrow \varphi\| := 1 + \|r\| + \|\varphi\|$.

[1] For the ease of exposition, we identify PSL with its linear-time core.

Syntactic Sugar. We use the standard conventions to omit parenthesis, e.g., temporal operators bind stronger than Boolean connectives and the binary operators of the SEREs are left associative. We also use standard syntactic sugar for the Boolean values, the Boolean connectives, and the linear-time temporal operators: $\mathsf{ff} := p \wedge \neg p$, for some proposition $p \in P$, $\mathsf{tt} := \neg\mathsf{ff}$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi := \neg\varphi \vee \psi$, $\mathsf{F}\varphi := \mathsf{tt}\,\mathsf{U}\,\varphi$, $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$, and $\varphi\,\mathsf{W}\,\psi := (\varphi\,\mathsf{U}\,\psi) \vee \mathsf{G}\varphi$, where φ and ψ are formulas. Moreover, $r \mathrel{\Box\!\!\rightarrow} \varphi$ abbreviates $\neg(r \mathrel{\Diamond\!\!\rightarrow} \neg\varphi)$.

3 Stutter-Invariant Regular Properties

In this section, we present syntactic characterizations for stutter-invariant regular and ω-regular languages. In Section 3.1, we define a variant of SEREs that can describe only stutter-invariant languages. Furthermore, we show that this variant of SEREs is complete in the sense that any stutter-invariant regular language can be described by such an expression. Similarly, in Section 3.2, we present a variant of PSL for expressing stutter-invariant ω-regular languages. In Section 3.3, we give examples that illustrate the use of our stutter-invariant variant of PSL.

3.1 Stutter-Invariant SEREs

It is straightforward to see that stutter-invariant languages are not closed under the concatenation and the Kleene star. A perhaps surprising example is the SERE $p^+ \,;\, q^+$ over the proposition set $\{p, q\}$, which does not describe a stutter-invariant language, although $L(p^+)$ and $L(q^+)$ are stutter-invariant languages.[2] In our variant of SEREs, we restrict the use of concatenation and replace the Kleene star by an iteration operator, which uses the fusion instead of the concatenation for gluing words together. Namely, for a language L of finite words, we define $L^{\oplus} := \bigcup_{n\in\mathbb{N}} L_n$, where $L_0 := L$ and $L_{i+1} := L_i : L$, for $i \in \mathbb{N}$.

The following lemma summarizes some closure properties of the class of stutter-invariant languages.

Lemma 1. *Let $K \subseteq \Sigma^*$ and $L, L' \subseteq \Sigma^\infty$ be stutter-invariant languages. The languages $L \cap L'$, $L \cup L'$, $K : L$, and $K^{\oplus}$ are stutter-invariant. Furthermore, $\Sigma^* \setminus K$, $\Sigma^\omega \setminus L$, and $\Sigma^\infty \setminus L$ are stutter-invariant.*

Proof. We only show that the language $K : L$ is stutter-invariant. The other closure properties are similarly proved. Assume that $u \in K : L$ and $\sharp(u) = \sharp(v)$ for $u, v \in \Sigma^\infty$. Let $u = u'bu''$, for some $u' \in \Sigma^*$, $u'' \in \Sigma^\infty$, and $b \in \Sigma$ with $u'b \in K$ and $bu'' \in L$. Since K is stutter-invariant, we can assume without loss of generality that if u' is nonempty then $u'(|u'|-1) \neq b$. Since $\sharp(u) = \sharp(v)$, there are $v' \in \Sigma^*$ and $v'' \in \Sigma^\infty$ such that $v = v'bv''$, $\sharp(v') = \sharp(u')$, and $\sharp(bv'') = \sharp(bu'')$. From the stutter invariance of K and L, it follows that $v \in K : L$. □

Our variant of SEREs is defined as follows.

[2] Note that the word $\{p, q\}\,\{p, q\}$ belongs to $L(p^+ \,;\, q^+)$ but the word $\{p, q\}$ does not.

Definition 1. *The syntax of* siSEREs *over the proposition set P is given by the grammar*

$$r ::= \varepsilon \;\big|\; b^+ \;\big|\; b^* \,;\, r \;\big|\; r \,;\, b^* \;\big|\; r : r \;\big|\; r \cup r \;\big|\; r \cap r \;\big|\; r^{\oplus}\,,$$

where b ranges over the Boolean formulas in $\mathcal{B}(P)$. The language $L(r)$ of an siSERE r is defined as expected.

By an induction over the structure of siSEREs, which uses the closure properties from Lemma 1, we easily obtain the following theorem.

Theorem 1. *The language of every siSERE is stutter-invariant.*

In the remainder of this subsection, we show that any regular language that is stutter-invariant can be described by an siSERE. We prove this result by defining a function κ that maps SEREs to siSEREs. We show that it preserves the language if the given SERE describes a stutter-invariant language. The function κ is defined recursively over the structure of SEREs:

$$\begin{aligned}
\kappa(\varepsilon) &:= \varepsilon\\
\kappa(b) &:= b^+\\
\kappa(s \cup t) &:= \kappa(s) \cup \kappa(t)\\
\kappa(s \cap t) &:= \kappa(s) \cap \kappa(t)\\
\kappa(s : t) &:= \kappa(s) : \kappa(t)\\
\kappa(s \,;\, t) &:= \Big(\kappa(s) : \bigcup_{a \in 2^P} \big(\hat{a}^+ : \big(\hat{a}^* \,;\, \kappa(t)\big)\big)\Big) \cup \begin{cases} \kappa(t) & \text{if } \varepsilon \in L(s)\\ \mathsf{ff} & \text{otherwise}\end{cases}\\
\kappa(s^*) &:= \varepsilon \cup \kappa(s) \cup \Big(\kappa(s) : \Big(\bigcup_{a \in 2^P} \big(\hat{a}^+ : (\hat{a}^* \,;\, \kappa(s))\big)\Big)^{\oplus}\Big)\,,
\end{aligned}$$

where $b \in \mathcal{B}(P)$, s, t are SEREs, and $\hat{a} := \bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \neg p$, for $a \in 2^P$.

Lemma 2. *For every SERE r, the equality $L_\sharp(r) = L_\sharp(\kappa(r))$ holds.*

Proof. We show the lemma by induction over the structure of the SERE r. The base cases where r is ε or b with $b \in \mathcal{B}(P)$ are obvious. The step cases where r is of one of the forms $s \cup t$, $s \cap t$, or $s : t$ follow straightforwardly from the induction hypothesis.

Next, we prove the step case where r is of the form $s \,;\, t$. For showing $L_\sharp(r) \subseteq L_\sharp(\kappa(r))$, assume that $u \in L_\sharp(r)$. There are words $x \in L(s)$ and $y \in L(t)$ such that $u = \sharp(xy)$. By induction hypothesis, we have that $\sharp(x) \in L_\sharp(\kappa(s))$ and $\sharp(y) \in L_\sharp(\kappa(t))$. The case where x the empty word is obvious. Assume that $x \neq \varepsilon$ and $a \in 2^P$ is the last letter of x. We have that $\sharp(xy) \in L_\sharp\big((\kappa(s) : \hat{a}) \,;\, \kappa(t)\big)$ and

$$\begin{aligned}
L_\sharp\big((\kappa(s) : \hat{a}) \,;\, \kappa(t)\big) \subseteq L_\sharp\big((\kappa(s) : (\hat{a} \,;\, \kappa(t))\big) &\subseteq L_\sharp\big(\kappa(s) : ((\hat{a} : \hat{a}) \,;\, \kappa(t))\big)\\
&\subseteq L_\sharp\big(\kappa(s) : (\hat{a}^+ : (\hat{a}^* \,;\, \kappa(t)))\big)\,.
\end{aligned}$$

For showing $L_\sharp(r) \supseteq L_\sharp(\kappa(r))$, assume that $u \in L_\sharp(\kappa(r))$. We make a case split.

1. If $\varepsilon \in L(s)$ and $u \in L_\sharp(\kappa(t))$ then $u \in L_\sharp(t)$ by induction hypothesis. We conclude that $u \in L_\sharp(\varepsilon\,;t) \subseteq L_\sharp(s\,;t) = L_\sharp(r)$.
2. Assume that $u \in L_\sharp(\kappa(s):\bigcup_{a\in 2^P}\big(\hat{a}^+:(\hat{a}^*;\kappa(t))\big))$. There is a letter $a \in 2^P$ such that $u \in L_\sharp(\kappa(s):(\hat{a}^+:(\hat{a}^*\,;\kappa(t)))) = L_\sharp(\kappa(s):(\hat{a}\,;\kappa(t)))$. It follows that there are words x and y such that $u = xay$, $xa \in L_\sharp(\kappa(s))$, and $ay \in L_\sharp(\hat{a}\,;\kappa(t))$. We have that either $ay \in L_\sharp(\kappa(t))$ or $y \in L_\sharp(\kappa(t))$. By induction hypothesis, we have that $xa \in L_\sharp(s)$ and either $ay \in L_\sharp(t)$ or $y \in L_\sharp(t)$. It follows that $u \in L_\sharp(r)$.

Finally, we prove the step case where r is of the form s^*. We first show $L_\sharp(r) \subseteq L_\sharp(\kappa(r))$. Assume that $u \in L_\sharp(s^*)$. If u is the empty word or $u \in L_\sharp(s)$ then there is nothing to prove. Assume that u is of the form $u_1u_2\dots u_n$ with $u_i \in L_\sharp(s)$ and $u_i \neq \varepsilon$, for all $1 \leq i \leq n$. By induction hypothesis, we have that $u_i \in L_\sharp(\kappa(s))$. Let a_i be the last letter of u_i, for each $1 \leq i < n$, respectively. We have that $\sharp(a_{i-1}u_i) \in L_\sharp(\hat{a}^+_{i-1} : (\hat{a}^*_{i-1}\,; \kappa(s)))$, for all $1 < i \leq n$. It follows that $\sharp(u_1a_1u_2\dots a_{n-1}a_n) \in L(\kappa(s)) : L_\sharp(\hat{a}^+_1 : (\hat{a}^*_2\,;\kappa(s))) : \ldots : L_\sharp(\hat{a}^+_{n-1} : (\hat{a}^*_n\,;\kappa(s)))$. Since $\sharp(u) = \sharp(u_1a_1u_2\dots a_{n-1}a_n)$, we conclude that $\sharp(u) \in L_\sharp(\kappa(r))$.

For showing $L_\sharp(r) \supseteq L_\sharp(\kappa(r))$, we assume that $u \in L_\sharp(\kappa(r))$. The cases $u = \varepsilon$ and $u \in L_\sharp(\kappa(s))$ are obvious. So, we assume that $u \in L_\sharp\big(\kappa(s) : \big(\bigcup_{a\in 2^P}(\hat{a}^+ : (\hat{a}^*\,;\kappa(s)))\big)^{\oplus}\big) = L_\sharp\big(\kappa(s): \big(\bigcup_{a\in 2^P}(\hat{a}\,;\kappa(s))\big)^{\oplus}\big) = L_\sharp\big(s: \big(\bigcup_{a\in 2^P}(\hat{a}\,;s)\big)^{\oplus}\big)$, where the last equality holds by induction hypothesis. There is an integer $n \geq 2$ and words $u_1, u_2, \dots, u_n \in L(s)$ and letters $a_1, a_2, \dots, a_{n-1} \in 2^P$ such that $u = \sharp(u_1a_1u_2\dots a_{n-1}u_n)$ and $\sharp(u_i) = \sharp(u_ia_i)$, for all $1 \leq i < n$. It follows that $u = \sharp(u_1u_2\dots u_n) \in L_\sharp(s^*)$. □

A consequence of Lemma 2 is that the translated siSERE describes the minimal stutter-invariant language that overapproximates the language of the given SERE.

Lemma 3. *For every SERE r, $L(r) \subseteq L(\kappa(r))$ and if K is a stutter-invariant language with $L(r) \subseteq K$ then $L(\kappa(r)) \subseteq K$.*

Proof. Let K be a stutter-invariant language with $L(r) \subseteq K$ and let $w \in L(\kappa(r))$. We have to show that $w \in K$. Since $L(\kappa(r))$ is stutter-invariant, we have that $\sharp(w) \in L(\kappa(r))$. With Lemma 2, we conclude that $\sharp(w) \in L_\sharp(r)$. It follows that there is a word $u \in L(r)$ with $\sharp(u) = \sharp(w)$. Since $K \supseteq L(r)$, we have that $\sharp(w) \in K$ and thus, $w \in K$ since K is stutter-invariant.

It remains to be proven that $L(r) \subseteq L(\kappa(r))$. For $w \in L(r)$, we have that $\sharp(w) \in L_\sharp(r)$. By Lemma 2, we have that $\sharp(w) \in L_\sharp(\kappa(r))$. Since $L(\kappa(r))$ is stutter-invariant, we conclude that $w \in L(\kappa(r))$. □

From Lemma 3 we immediately obtain the following theorem.

Theorem 2. *For every stutter-invariant regular language L, there is an siSERE r such that $L(r) = L$.*

Note that the intersection and the fusion operation is not needed for SEREs to describe the class of regular languages. However, they are convenient for expressing regular languages naturally and concisely. It follows immediately from the definition of the function κ that siSEREs even without the intersection operation exactly capture the class of stutter-invariant regular languages. However, in contrast to the intersection operator, the fusion operator is essential for describing this class of languages with siSEREs.

Finally, we remark that when translating an SERE of the form $r\,;s$ or s^*, we obtain an siSERE that contains a disjunction of all the letters in 2^P that contains $2^{|P|}$ copies of $\kappa(s)$. We conclude that in the worst case, the size of the siSERE $\kappa(r)$ for a given SERE r is exponential in $\|r\|$. It remains open whether for every SERE that describes a stutter-invariant regular language, there is a language-equivalent siSERE of polynomial size.

3.2 Stutter-Invariant PSL

Similar to the previous subsection, we define a variant of the core of PSL and show that this temporal logic describes exactly the class of stutter-invariant ω-regular languages.

Definition 2. *The syntax of* siPSL *formulas is similar to that of PSL formulas except that the formulas do not contain the temporal operator* X *and instead of SEREs they contain siSEREs. The semantics is defined as expected.*

By a straightforward induction over the structure of siPSL formulas and by using the closure properties from Lemma 1, we obtain the following theorem. Note that $L(r \diamond\!\!\rightarrow \varphi) = L(r) : L(\varphi)$. Furthermore, it is easy to see that the language $L(\mathsf{cl}(r))$ is stutter-invariant if r is an SERE or siSERE that describes a stutter-invariant language.

Theorem 3. *The language of every siPSL formula is stutter-invariant.*

In the following, we show that every stutter-invariant ω-regular language can be described by an siPSL formula. We do this by extending the translations in [17] for eliminating the temporal operator X in LTL formulas to PSL formulas. We define the function τ that translates PSL formulas into siPSL formulas as follows. It is defined recursively over the formula structure and it uses the function κ from Section 3.1 for translating SEREs into siSEREs.

$$\begin{aligned}
\tau(p) &:= p \\
\tau(\mathsf{cl}(r)) &:= \mathsf{cl}(\kappa(r)) \\
\tau(\neg\varphi) &:= \neg\tau(\varphi) \\
\tau(\varphi \wedge \psi) &:= \tau(\varphi) \wedge \tau(\psi) \\
\tau(\varphi \mathbin{\mathsf{U}} \psi) &:= \tau(\varphi) \mathbin{\mathsf{U}} \tau(\psi) \\
\tau(r \diamond\!\!\rightarrow \varphi) &:= \kappa(r) \diamond\!\!\rightarrow \tau(\varphi) \\
\tau(\mathsf{X}\varphi) &:= \bigvee_{a \in 2^P} \Big(\big(\mathsf{G}\hat{a} \wedge \tau(\varphi)\big) \vee \bigvee_{b \in 2^P \setminus \{a\}} \big(\hat{a} \mathbin{\mathsf{U}} \big(\hat{b} \wedge \tau(\varphi)\big)\big)\Big)
\end{aligned}$$

The intuition of the elimination of the outermost operator X in a formula $\mathsf{X}\varphi$ is as follows: “the first time after now that some new event happens, φ must hold, or else, if nothing new ever happens, φ must hold right now.”

Note that the size of the resulting siPSL formula is in the worst case exponential in the size of the given PSL formula. The sources of the blow-up are (1) the translation of the SEREs in the given PSL formula into siSEREs and (2) the elimination of the temporal operator X. We can improve the translation τ with respect to the size of the resulting formula by using the translation defined in [10] for eliminating the operator X in LTL formulas that describe stutter-invariant languages. The translation in [10] avoids the conjunctions over the letters in 2^P. Instead the conjunctions only range over the propositions in P. The elimination of an operator X is not exponential in $|P|$ anymore. However, the resulting translation for PSL into siPSL is still exponential in the worst case because of (1). The question whether the exponential blow-up can be avoided remains open.

The following lemma for τ is the analog of Lemma 2 for the function κ.

Lemma 4. *For every PSL formula φ, the equality $L_\sharp(\varphi) = L_\sharp(\tau(\varphi))$ holds.*

Similar to Lemma 3 for SEREs, we obtain that the function τ translates PSL formulas into siPSL formulas that minimally overapproximate the described languages with respect to stutter invariance.

Lemma 5. *For every PSL formula φ, $L(\varphi) \subseteq L(\tau(\varphi))$ and if L is a stutter-invariant language with $L(\varphi) \subseteq L$ then $L(\tau(\varphi)) \subseteq L$.*

From Lemma 5 we immediately obtain the following theorem.

Theorem 4. *For every stutter-invariant ω-regular language L, there is an siPSL formula φ such that $L(\varphi) = L$.*

We remark that the finite-state model-checking problem for PSL and siPSL fall into the same complexity classes. Namely, the finite-state model-checking problem for siPSL is EXPSPACE-complete and the problem becomes PSPACE-complete when the number of intersection operators in the given siPSL formulas is bounded. These complexity bounds can be easily established from the existing bounds on PSL, see [4] and [5,14]. Note that the automata-theoretic realization of the iteration operator $\oplus$ is similar to the one that handles the Kleene-star.

Recently, we proposed an extension of PSL with past operators [7]. As for $\mathrm{LTL}_{-\mathsf{X}}$ [17], we remark that our result on the stutter invariance of siPSL straightforwardly carries over to an extension of siPSL with past operators.

3.3 siPSL Examples

In the following, we illustrate that stutter-invariant ω-regular properties can be naturally expressed in siPSL. For comparison, we describe these properties in siPSL and other temporal logics that express stutter-invariant properties.

Star-Free Properties. Consider the following commonly used specification patterns taken from [8]:

(P1) *Absence:* p is false after q until r.
(P2) *Existence:* p becomes true between q and r.

Table 1. siPSL formulas and LTL$_{-\mathsf{X}}$ formulas of the specification patterns

pattern	siPSL formula	LTL$_{-\mathsf{X}}$ formula
P1	$\mathsf{G}(q^+ : \neg r^+ \mathrel{\Box\!\!\rightarrow} \neg p)$	$\mathsf{G}(q \wedge \neg r \rightarrow (\neg p) \mathbin{\mathsf{W}} r)$
P2	$\mathsf{G}((q \wedge \neg r)^+ : (\neg p^* \,; r^+) \mathrel{\Box\!\!\rightarrow} \mathsf{ff})$	$\mathsf{G}(q \wedge \neg r \rightarrow (\neg r) \mathbin{\mathsf{W}} (p \wedge \neg r))$
P3	$\mathsf{G}(q^+ : \neg r^+ : \neg p : (\neg r^*; r^+) \mathrel{\Box\!\!\rightarrow} \mathsf{ff})$	$\mathsf{G}(q \wedge \neg r \wedge \mathsf{F} r \rightarrow p \mathbin{\mathsf{U}} r)$
P4	$\mathsf{G}(q^+ : (\neg r \wedge \neg s)^+ \mathrel{\Box\!\!\rightarrow} \neg p)$	$\mathsf{G}(q \wedge \neg r \rightarrow (\neg p) \mathbin{\mathsf{W}} (s \vee r))$
P5	$\mathsf{G}(q^+ : \neg r^+ : p \mathrel{\Box\!\!\rightarrow} (\neg r^+ : s^+ \mathrel{\Diamond\!\!\rightarrow} \mathsf{tt}))$	$\mathsf{G}(q \wedge \neg r \rightarrow (p \rightarrow (\neg r) \mathbin{\mathsf{U}} (s \wedge \neg r)) \mathbin{\mathsf{W}} r)$

(P3) *Universality:* p is true between q and r.
(P4) *Precedence:* s precedes p, after q until r.
(P5) *Response:* s responds to p, after q until r.

Table 1 contains the formalization of these specification patterns in siPSL and LTL$_{-\mathsf{X}}$. Note that any LTL$_{-\mathsf{X}}$ is also an siPSL formula. However, since practitioners often find it easier to use (semi-extended) regular expressions than the temporal operators in LTL, we have used siSEREs in the siPSL formulas to formalize the patterns in siPSL. An advantage of siPSL over LTL$_{-\mathsf{X}}$ is that one can choose between the two specifications styles and mix them.

Omega-regular Properties. We consider the stutter-invariant ω-regular language

$$L_n := \{w \in (2^{\{p\}})^\omega \;:\; \text{the number of occurrences of the subword } \{p\}\emptyset \text{ in } w \text{ is divisible by } n\}\,,$$

for $n \geq 2$. The following siPSL formula describes the language L_n:

$$\mathsf{neverswitch} \vee \Big(\big(\underbrace{(\neg p^* \,; \mathsf{switch}) : \ldots : (\neg p^* \,; \mathsf{switch})}_{n \text{ times}}\big)^{\oplus} \mathrel{\Diamond\!\!\rightarrow} \mathsf{neverswitch}\Big),$$

where $\mathsf{switch} := p^+ : (p^* \,; \neg p^+)$ and $\mathsf{neverswitch} := (\neg p) \mathbin{\mathsf{W}} \mathsf{G}p$.

Note that the language L_n is not star-free and thus, it cannot be described in LTL$_{-\mathsf{X}}$. In the following, we compare our siPSL formalization of L_n with a formalization in the temporal logic SI-EQLTL from [9], which has the same expressive power as siPSL. We briefly recall the syntax and semantics of SI-EQLTL. The formulas in SI-EQLTL are of the form $\exists^h q_1 \ldots \exists^h q_n \varphi$, where φ is an LTL$_{-\mathsf{X}}$ formula over a proposition set that contains the propositions $q_1, \ldots, q_n$. The semantics of the quantifier $\exists^h$ is as follows. Let P be a proposition set with $q \notin P$. The word $w \in (2^{P \cup \{q\}})^\omega$ is a *harmonious extension* of $v \in (2^P)^\omega$ if for all $i \in \mathbb{N}$, it holds that $v(i) = w(i) \cap P$ and if $v(i) = v(i+1)$ then $w(i) = w(i+1)$. For $v \in (2^P)^\omega$, we define $v \models \exists^h q\, \varphi$ iff $w \models \varphi$, for some harmonious extension $w \in (2^{P \cup \{q\}})^\omega$ of v.

For readability, we only state an SI-EQLTL formula that describes the language L_2 (the formula can be straightforwardly generalized for describing the language L_n with $n \geq 2$):

$$\exists^h q\big(q \wedge \mathsf{G}(q \rightarrow \mathsf{neverswitch} \vee \mathsf{switch}_2) \wedge \mathsf{F}\,\mathsf{neverswitch}\big),$$

where

$$\mathsf{switch}_2 := (\neg p \wedge q)\ \mathsf{U}\ \Bigg((p \wedge q)\ \mathsf{U}\ \Big((\neg p \wedge \neg q)\ \mathsf{U}\ \big((p \wedge \neg q)\ \mathsf{U}\ (\neg p \wedge q)\big)\Big)\Bigg).$$

Intuitively, the subformula switch_2 matches subwords that contain two occurrences of $\{p\}\emptyset$. Furthermore, the harmoniously existentially quantified proposition q marks every position k of a word in L_2, where the number of occurrences of $\{p\}\emptyset$ in $w^{\leq k}$ is even.

We remark that we did not manage to come up with a simpler SI-EQLTL formula for describing the language L_n.[3] Nevertheless, we consider the SI-EQLTL formula for L_n still hard to read because of the harmonious quantified variable q and the nesting of the temporal operators, which is linear in n. Furthermore, note that the advantage of siPSL over LTL$_{-\mathsf{X}}$, namely, to mix different specification styles, is also an advantage of siPSL over SI-EQLTL.

4 Concluding Remarks

We have presented the specification languages siSEREs and siPSL, which capture exactly the classes of stutter-invariant regular and ω-regular languages, respectively. siSEREs are a variants of SEREs and siPSL is a variant of the temporal logic PSL [1], which is nowadays widely used in industry. siPSL inherits the following pleasant features from PSL. First, siPSL is easy to use. Second, the computational complexities for solving the finite-state model-checking problem with siPSL and fragments thereof are similar to the corresponding problems for PSL. Third, with only minor modifications we can use the existing tool support for PSL (like the model checker RuleBase [3], the formula translator into nondeterministic Büchi automata rtl2ba [7], or the translator used in [6] with all its optimizations) for siPSL. We only need to provide additional support for the new Kleene-star-like iteration operator $\oplus$ of the siSEREs.

References

1. IEEE standard for property specification language (PSL). IEEE Std 1850TM (October 2005)
2. Alur, R., Brayton, R.K., Henzinger, T.A., Qadeer, S., Rajamani, S.K.: Partial-order reduction in symbolic state-space exploration. Form. Method. Syst. Des. 18(2), 97–116 (2001)
3. Beer, I., Ben-David, S., Eisner, C., Geist, D., Gluhovsky, L., Heyman, T., Landver, A., Paanah, P., Rodeh, Y., Ronin, G., Wolfsthal, Y.: RuleBase: Model checking at IBM. In: Marie, R., Plateau, B., Calzarossa, M.C., Rubino, G.J. (eds.) TOOLS 1997. LNCS, vol. 1245, pp. 480–483. Springer, Heidelberg (1997)

[3] We encourage the reader to find a simpler SI-EQLTL formula that describes L_n.

4. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata construction algorithms optimized for PSL. Technical report, The Prosyd Project (2005), http://www.prosyd.org
5. Bustan, D., Havlicek, J.: Some complexity results for SystemVerilog assertions. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 205–218. Springer, Heidelberg (2006)
6. Cimatti, A., Roveri, M., Tonetta, S.: Symbolic compilation of PSL. IEEE Trans. on CAD of Integrated Circuits and Systems 27(10), 1737–1750 (2008)
7. Dax, C., Klaedtke, F., Lange, M.: On regular temporal logics with past. In: Proceedings of the 36th International Colloquium on Automata, Languages, and Programming, ICALP (to appear, 2009)
8. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering (ICSE), pp. 411–420 (1999), http://patterns.projects.cis.ksu.edu/
9. Etessami, K.: Stutter-invariant languages, ω-automata, and temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 236–248. Springer, Heidelberg (1999)
10. Etessami, K.: A note on a question of Peled and Wilke regarding stutter-invariant LTL. Inform. Process. Lett. 75(6), 261–263 (2000)
11. Godefroid, P., Wolper, P.: A partial approach to model checking. Inf. Comput. 110(2), 305–326 (1994)
12. Holzmann, G., Kupferman, O.: Not checking for closure under stuttering. In: Proceedings of the 2nd International Workshop on the SPIN Verification System. Series in Discrete Mathematics and Theoretical Computer Science, vol. 32, pp. 163–169 (1996)
13. Lamport, L.: What good is temporal logic? In: Proceedings of the 9th IFIP World Computer Congress. Information Processing, vol. 83, pp. 657–668 (1983)
14. Lange, M.: Linear time logics around PSL: Complexity, expressiveness, and a little bit of succinctness. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 90–104. Springer, Heidelberg (2007)
15. Peled, D.: Combining partial order reductions with on-the-fly model-checking. Form. Method. Syst. Des. 8(1), 39–64 (1996)
16. Peled, D.: Ten years of partial order reduction. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 17–28. Springer, Heidelberg (1998)
17. Peled, D., Wilke, T.: Stutter-invariant temporal properties are expressible without the next operator. Inform. Process. Lett. 63(5), 243–246 (1997)
18. Peled, D., Wilke, T., Wolper, P.: An algorithmic approach for checking closure properties of temporal logic specifications and ω-regular languages. Theoret. Comput. Sci. 195(2), 183–203 (1998)
19. Rabinovich, A.M.: Expressive completeness of temporal logic of action. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, pp. 229–238. Springer, Heidelberg (1998)
20. Valmari, A.: A stubborn attack on state explosion. Form. Method. Syst. Des. 1(4), 297–322 (1992)
21. Vardi, M.Y.: From philosophical to industrial logics. In: Ramanujam, R., Sarukkai, S. (eds.) Logic and Its Applications. LNCS, vol. 5378, pp. 89–115. Springer, Heidelberg (2009)