Acta Informatica (2023) 60:257–278
https://doi.org/10.1007/s00236-023-00439-3

ORIGINAL ARTICLE

On the undecidability and descriptional complexity of synchronized regular expressions

Jingnan Xie[1] · **Harry B. Hunt III**[2]

Received: 13 June 2022 / Accepted: 20 March 2023 / Published online: 10 April 2023

Abstract

In Freydenberger (Theory Comput Syst 53(2):159–193, 2013. https://doi.org/10.1007/s00224-012-9389-0), Freydenberger shows that the set of invalid computations of an extended Turing machine can be recognized by a synchronized regular expression [as defined in Della Penna et al. (Acta Informatica 39(1):31–70, 2003. https://doi.org/10.1007/s00236-002-0099-y)]. Therefore, the widely discussed predicate "$= \{0, 1\}^*$" is not recursively enumerable for synchronized regular expressions (SRE). In this paper, we employ a stronger form of non-recursive enumerability called *productiveness* and show that the set of invalid computations of a deterministic Turing machine on a single input can be recognized by a synchronized regular expression. Hence, for a polynomial-time decidable subset of SRE, where each expression generates either $\{0, 1\}^*$ or $\{0, 1\}^* - \{w\}$ where $w \in \{0, 1\}^*$, the predicate "$= \{0, 1\}^*$" is productive. This result can be easily applied to other classes of language descriptors due to the simplicity of the construction in its proof. This result also implies that many computational problems, especially promise problems, for SRE are productive. These problems include language class comparison problems (e.g., does a given synchronized regular expression generate a context-free language?), and equivalence and containment problems of several types (e.g., does a given synchronized regular expression generate a language equal to a fixed unbounded regular set?). In addition, we study the descriptional complexity of SRE. A generalized method for studying trade-offs between SRE and many classes of language descriptors is established.

1 Introduction

An extension of regular expressions—synchronized regular expressions (SRE)—is defined and studied in [6]. SRE may allow to find if certain subexpressions are repeated the same number of times in a text. This can be useful for integrity checks, especially when mixed

✉ Jingnan Xie
jingnan.xie@millersville.edu

Harry B. Hunt III
hhunt@albany.edu

[1] Computer Science, Millersville University of PA, 40 Dilworth Rd, Millersville, PA 17551, USA

[2] Computer Science, University at Albany, SUNY, 1400 Washington Avenue, Albany, NY 12222, USA

with other extensions such as backreferences (as defined in [1]). Della Penna et al. use SRE to present a formal study of the backreferences extension and of a new extension called the *synchronized exponents* proposed by them. They also study the classification of SRE in the formal language hierarchy and show that SRE are context-sensitive but do not generate all context-free languages. The membership problem for SRE and SRE with certain restrictions is studied in [6] as well. In [5], Carle shows that the language of palindromes cannot be generated by a synchronized regular expression, and the language $\{ww|w \in \{0, 1\}^*\}$ is a synchronized regular language. Hence, the class of synchronized regular languages is incomparable with the class of context-free languages.

In [7], Freydenberger shows that the set of invalid computations of an extended Turing machine can be recognized by an extended regular expression (introduced by Câmpeanu et al [4]), hence, by a synchronized regular expression. Therefore, the widely discussed predicate "$= \{0, 1\}^*$" is not recursively enumerable for SRE. Moreover, Freydenberger shows that the regularity, cofiniteness, and RegEx(k)-ity (defined in [7]) problems are not recursively enumerable for SRE. More research on extended regular expressions (EXREGs) can be found in [2, 9, 26, 27].

In this paper, we employ a stronger form of non-recursive enumerability called *productiveness*. A productive set S is not recursively enumerable. Furthermore, for any effective axiomatic system F, there is an effective procedure to construct an element that is in S, but not provable in F (see Sect. 3.1 for precise definitions). Then, we show that the set of invalid computations of a deterministic Turing machine on a single input can be recognized by a synchronized regular expression. Hence, for a polynomial-time decidable subset of SRE, where each expression generates either $\{0, 1\}^*$ or $\{0, 1\}^* - \{w\}$ where $w \in \{0, 1\}^*$, the predicate "$= \{0, 1\}^*$" is productive. This special type of universality problem is denoted by "$= \{0, 1\}^* |_{|L^c| \leq 1}$". This result can be easily applied to other classes of language descriptors, such as 1-SRE (see Definition 6), one-reversal bounded one-counter machines, and real-time one-way cellular automata (the definition can be found in [20]), due to the simplicity of the construction in its proof. This result also implies the productiveness of many problems for SRE. These problems include:

1. a variety of equivalence and containment problems such as testing equivalence to any fixed unbounded regular languages,
2. *language class comparison problem* which is defined as follows:
   For two classes of language descriptors $\mathcal{D}_1$ and $\mathcal{D}_2$, determine for any $a \in \mathcal{D}_1$, whether $\mathcal{L}(a) \in \mathcal{L}(\mathcal{D}_2)$?

The general containment problem for pattern languages over fixed alphabets is shown to be undecidable in [8] which can be applied to SRE directly since all pattern languages can be represented by SRE (see [6]). We study the problems of testing equivalence and containment to many fixed languages since these results are stronger and have more practical meanings. For example, the result of testing equivalence to a fixed regular set enables us to show there is no approximating minimization algorithm between SRE and DFA accepting this fixed regular set.

Several authors have investigated the existence and applicability of analogues of Rice's Theorem for many classes of languages. For example, in [17, 18], sufficient conditions are given for a language predicate to be as hard as the language predicate $= \{0, 1\}^*$. There are five major differences between the previous results in [17, 18] and the results in this paper.

1. In Theorem 4.5, we show a way to study predicates that are not true for any regular/context-free sets. This is not done in the previous research. Most of the previous results require the language predicates to be true for some regular sets.

2. Due to the properties of the predicate "$= \{0, 1\}^* \ |_{|L^c|\leq 1}$", most of the results in this paper are applicable to promise problems. For example, Theorem 4.3 states that for a polynomial-time decidable subset of SRE, where each expression is guaranteed to generate a regular set, the predicates are productive.
3. The previous results are only for regular expressions, context-free grammars, and in some cases, context-sensitive grammars. But, because of the simple construction in Proposition 3, we can easily apply the results of this paper to any class of language descriptors $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D})$ contains the language $\{x \# y \mid x, y \in (\Sigma - \{\#\})^* \text{ and } |x| = |y|\}$ and $\mathcal{L}(\mathcal{D})$ is closed under union, and concatenation with regular sets. For example, the results of this paper can be applied to SRE, 1-SRE, one-reversal bounded one-counter machines, and real-time one-way cellular automata.
4. The previous results cannot be applied to the language class comparison problems between incomparable classes. But the results in this paper enable us to study such problems. For example, we know that the class of synchronized regular languages is incomparable with the class of context-free languages. In Corollary 4, we show that it is productive to determine, for an arbitrary synchronized regular expression, whether it generates a context-free language.
5. The previous results require the language predicates to be with certain restrictions, such as closed under left or right derivatives. But in Corollary 3, we show that it is productive to determine, for an arbitrary synchronized regular expression, whether it generates a k-pattern language (defined in Definition 3) for any $k \geq 1$. Due to the dichotomization of the reduction in the proof, we do not need any closure property for k-pattern languages.

The second aim of this paper is to study the descriptional complexity of SRE. In the theory of formal languages, questions concerning descriptional complexity are widely discussed. How succinctly can a descriptor generate a language in comparison with some other descriptors generating the same language? It is well-known that for all natural number $n \geq 1$, there exists a regular language accepted by some nondeterministic finite automata (NFA) with n states but every deterministic finite automaton (DFA) accepting the same language has at least 2^n states. In [13], Hartmanis shows that there is no recursive trade-off between pushdown automata (PDA) and deterministic pushdown automata (DPDA). In [7], Freydenberger shows that there is no recursive trade-off between SRE and regular expressions. More related research can be found in [14]. In this paper, we study trade-offs between SRE and many language descriptors including DFA, subclasses of regular expressions, and multi-patterns.

Multi-patterns (MP) and multi-pattern languages (MPL) are defined in [19] to exhibit common patterns for a given sample of words. As Della Penna et al. mentioned in [6],

> backreferences are a generalization of patterns, i.e., expressions that make reference to the string matched by a previous subexpression,

we believe it is interesting to consider the relationship between SRE and MPL. Several results established in this paper are related to MPL. We prove that it is productive whether a given synchronized regular expression generates a multi-pattern language. In addition, we show that there is no recursive trade-off between SRE and MP.

This paper is organized as follows.

In Sect. 2, we review the definitions of SRE and MP. Several preliminary definitions and notations are also explained.

In Sect. 3.1, the definition and importance of productiveness are discussed. In Sect. 3.2, we show the predicate "$= \{0, 1\}^* \ |_{|L^c|\leq 1}$" is productive for SRE.

In Sect. 4, sufficient conditions are given for a language predicate to be as hard as the language predicates "$= \{0, 1\}^* \mid_{|L^c| \leq 1}$" for SRE. These conditions yield a method for proving productiveness results through highly efficient many-one reductions. Using this method, we prove many computational problems are productive for SRE.

In Sect. 5, we study the descriptional complexity of SRE and generalize a method for showing non-recursive trade-offs between SRE and many classes of language descriptors.

2 Definitions and notations

In this section, we review the definitions of SRE and MPL from [6] and [19], respectively. Several preliminary definitions and notations are also explained. The reader is referred to [16] for all unexplained notations and terminologies in language theory.

We use λ to denote the empty string and $\emptyset$ to denote the empty set. We use $\mathbb{N}$ to denote the set of natural numbers. Let **P** denote the class of sets that can be recognized in polynomial time by a deterministic Turing Machine. If A is many-one reducible to B, we write $A \leq_m B$.

Let **REG({0,1})** be the set of $(\cup, \cdot, *)$-regular expressions over language alphabet $\{0, 1\}$. Let **CFG({0,1})** be the set of context-free grammars over terminal alphabet $\{0, 1\}$.

Definition 1 The *synchronized regular expressions* on an alphabet Σ, a set of variables V and a set of exponents X are defined as follows:

$\emptyset \in SRE$ (empty set)
$\lambda \in SRE$ (empty string)
$\forall a \in \Sigma : a \in SRE$ (letters)
$\forall v \in V : v \in SRE$ (variables)

If $e_1, e_2 \in SRE$, then:

1. $e_1^* \in SRE$ (star)
2. $\forall x \in X : e_1^x \in SRE$ (exponentiation)
3. $\forall v \in V : e_1 \% v \in SRE$ (variable binding)
4. $e_1 e_2 \in SRE$ (concatenation)
5. $e_1 + e_2 \in SRE$ (union)

□

Beyond these basic syntactic definitions, a synchronized regular expression must meet the following conditions to be considered *valid*.

Definition 2 The *SRE validity test* is defined as follows:

1. Each variable occurs in a binding operation no more than once in the expression.
2. Each occurrence of a variable in the expression is preceded by a binding of that variable somewhere to the left of the occurrence in the expression.

Throughout this paper, let **SRE**($\{\mathbf{0}, \mathbf{1}\}$) denote the set of valid synchronized regular expressions over alphabet $\{0, 1\}$. □

Unless otherwise specified, any mention of SRE in this paper refers to valid SRE. The following examples are used in later proofs of this paper and can help the readers better understand SRE.

Example 2.1 The synchronized regular expression 0^x1^x specifies the language $\{0^n1^n \mid n \geq 0\}$.

Example 2.2 The synchronized regular expression $(0+1)^x\#(0+1)^x$ specifies the language $\{a\#b \mid a, b \in \{0,1\}^*, |a| = |b|\}$.

Example 2.3 The synchronized regular expression $(0+1)^*\%X \cdot X$ (X is a variable) specifies the language $\{ww \mid w \in \{0,1\}^*\}$.

Definition 3 Let V be an alphabet of variables such that $V \cap \{0,1\} = \emptyset$. A *pattern* α is a string over $V \cup \{0,1\}$. Let $\mathcal{H}$ be the set of homomorphisms h where $h : (V \cup \{0,1\})^* \mapsto (V \cup \{0,1\})^*$. Then, *the language generated by the pattern* α is defined as
$\mathcal{L}(\alpha) = \{w \in \{0,1\}^* \mid w = h(\alpha)$ for some $h \in \mathcal{H}$ such that $h(0) = 0$ and $h(1) = 1\}$.
A *multi-pattern* π is a finite set of patterns, $\pi = \{\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n\}$ where $\alpha_i \in (V \cup \{0,1\})^*$ $(1 \leq i \leq n)$. *The language generated by the multi-pattern* π is

$$\mathcal{L}(\pi) = \bigcup_{i=1}^{n} \mathcal{L}(\alpha_i).$$

For all integer $k \geq 1$, a *k-pattern* p is a set of patterns of cardinality k. *The language generated by the k-pattern p* is

$$\mathcal{L}(p) = \bigcup_{i=1}^{k} \mathcal{L}(\alpha_i).$$

Throughout this paper, **MP({0,1})**denotes the set of all multi-patterns over terminal alphabet $\{0,1\}$. □

Example 2.4 The language $\{ww \mid w \in \{0,1\}^*\}$ is a pattern language.

Proof Consider the pattern $\alpha = xx$ where x is a variable. Since x can be replaced by any string in $\{0,1\}^*$, it is clear $\mathcal{L}(\alpha) = \{ww \mid w \in \{0,1\}^*\}$. □

Example 2.5 The language $\{0ww \mid w \in \{0,1\}^*\} \cup \{1w \mid w \in \{0,1\}^*\}$ is a multi-pattern language but not a pattern language.

Proof It is not hard to see that no single pattern can specify this language but the multi-pattern $\pi = \{0xx, 1x\}$ specifies this language. □

Example 2.6 The simple regular language $\{0\}^* \cdot \{1\}^*$ is not a multi-pattern language.

Proof The proof can be found in [19]. □

Let $\mathcal{D}$ be a class of language descriptors that describe languages over Σ. In this paper, we only consider finite Σ. Then, $\forall d \in \mathcal{D}$, $\mathcal{L}(d) = \{w \in \Sigma^* \mid w$ is described by $d\}$ and $\mathcal{L}(\mathcal{D}) = \{L \subseteq \Sigma^* \mid \exists d \in \mathcal{D}$ such that $L = \mathcal{L}(d)\}$. $\forall d \in \mathcal{D}$, let $|d|$ denote the size of d and $< d >$ denote a code of d.

Our code of a language descriptor is efficient and described informally below.

1. For a regular expression, synchronized regular expression, or multi-pattern, the code is itself.
2. For a context-free grammar with n nonterminals, nonterminals are denoted by s_u where u is a base 10 numeral without leading 0's representing integers $\{0, 1, \ldots, n-1\}$. Each production $A \rightarrow B$ is denoted by a pair (A, B).

3. For a Turing machine with a set of states Q and tape alphabet T, states are denoted by q_u where u is a base 10 numeral without leading 0's representing integers $\{0, 1, \ldots, |Q|-1\}$. Each machine move $\delta(q, a) = (q', a', d)$ where $q, q' \in Q, a, a' \in T$ and $d \in \{L, R\}$ is denoted by a 5-tuple (q, a, q', a', d). For other types of automata, we have similar rules. We use tuples to denote machine moves and q_u to denote states.

The size of a DFA is the number of states of the DFA. The size of a pattern is the number of symbols of the pattern. The size of a context-free grammar is the number of symbols of all its productions. For example, the following context-free grammar d accepts the language $\{0, 1\}^*$. $d = (\{s_1\}, \{0, 1\}, \{(s_1, 0s_1), (s_1, 1s_1), (s_1, \lambda)\}, s_1)$. The size of d is 8 (denoted by $|d| = 8$).

Comparing two classes of language descriptors $\mathcal{D}_1$ and $\mathcal{D}_2$, we assume that $\mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$ is not finite. We say that a function $f : \mathbb{N} \mapsto \mathbb{N}$ where $f(n) \geq n$ is an *upper bound for the trade-off between* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ when transforming from a minimal descriptor in $\mathcal{D}_1$ for an arbitrary language to an equivalent minimal descriptor in $\mathcal{D}_2$, if for all $L \in \mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$ the following holds:

$$Min\{|d| \mid d \in \mathcal{D}_2, \mathcal{L}(d) = L\} \leq f(Min\{|d| \mid d \in \mathcal{D}_1, \mathcal{L}(d) = L\}).$$

If no recursive function is an upper bound for the trade-off between $\mathcal{D}_1$ and $\mathcal{D}_2$, we say *the trade-off between* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *is non-recursive*.

3 Productiveness and the predicate "= {0, 1}*" for SRE

Section 3 consists of the two Sects. 3.1 and 3.2. Section 3.1 consists of the definition of productiveness, a stronger form of non-recursive enumerability, and two propositions that can be used to prove productiveness results. Section 3.2 consists of Theorem 3.1, which shows for a polynomial-time recognizable subset D' of $\mathbf{SRE}(\{0, 1\})$, such that $\forall d \in D'$, $\mathcal{L}(d) \subseteq \{0, 1\}^*$ and $|\{0, 1\}^* - \mathcal{L}(d)| \leq 1$, the set $\{< d >| d \in D', \mathcal{L}(d) = \{0, 1\}^*\}$ is productive, hence, non-recursively enumerable.

3.1 Productiveness

Productive sets and their properties are a standard topic in mathematical logic/recursion theory textbooks such as [25] and [28]. Productiveness is a recursion-theoretic abstraction of what causes Gödel's first incompleteness theorem to hold. Definition 4 recalls the definition of a productive set on $\mathbb{N}$, as developed in [25].

Definition 4 Let W be an effective Gödel numbering of the recursively enumerable sets. A set A of natural numbers is called *productive* if there exists a total recursive function f so that for all $i \in \mathbb{N}$, if $W_i \subseteq A$, then $f(i) \in A - W_i$. The function f is called the *productive function* for A. □

From this definition, we can see that no productive set is recursively enumerable. It is well-known that the set of all provable sentences in an effective axiomatic system is always a recursively enumerable set. So for any effective axiomatic system F, if a set A of Gödel numbers of true sentences in F is productive, then there is at least one element in A which is true but cannot be proven in F. Moreover, there is an effective procedure to produce such an element.

Let W be an effective Gödel numbering of the recursively enumerable sets. $\mathbf{K}$ denotes the set $\{i \in \mathbb{N} \mid i \in W_i\}$. $\overline{\mathbf{K}}$ denotes the set $\{i \in \mathbb{N} \mid i \notin W_i\}$. Two well-known facts of productive sets (see [25]) that are necessary for the research developed here are as follows:

Proposition 1 1. *$\overline{K}$ is productive.*
2. *For all $A \subseteq \mathbb{N}$, A is productive if and only if $\overline{\mathbf{K}} \leq_m A$.*

□

Let Σ, Δ be two different finite alphabets such that both $A \subseteq \Sigma^*$ and $A \subseteq \Delta^*$. It is easily seen that

There exists a total recursive function $F : \mathbb{N} \to \Sigma^*$ such that $\overline{\mathbf{K}} \leq_m A$ (via F) if and only if there exists a total recursive function $G : \mathbb{N} \to \Delta^*$ such that $\overline{\mathbf{K}} \leq_m A$ (via G).

Hence, language A is productive for some finite alphabet Σ such that $A \subseteq \Sigma^*$ if and only if A is productive for all finite alphabets Δ such that $A \subseteq \Delta^*$. This is the sense in which the concepts of productiveness are independent of particular finite alphabets.

The following proposition is used to prove many productiveness results for SRE. It also shows in which way the productiveness is stronger than non-recursive enumerability, i.e., every productive set A has an infinite recursively enumerable subset, and for any sound proof procedure P, one can effectively construct an element that is in A, but not provable in P.

Proposition 2 *Let $A \subseteq \Sigma^*$, $B \subseteq \Delta^*$, and $A \leq_m B$. Then, the following holds:*

1. *If A is productive, then so is B.*
2. *If A is productive, then there exists a total recursive function $\Psi : \Sigma^* \to \Sigma^*$, called a* productive function for A, *such that for all $x \in \Sigma^*$,*

   $\mathcal{L}(M_x) \subseteq A \Rightarrow \Psi(x) \in A - \mathcal{L}(M_x)$, where $\{M_x \mid x \in \Sigma^*\}$ is some Gödel-numbering of Turing machines over alphabet Σ.

3. *If A is productive, then A is not recursively enumerable (RE). However, A does have an infinite RE subset.*

□

Proof of 1: By Proposition 1, if A is productive, then $\overline{\mathbf{K}} \leq_m A$. Hence by the transitivity of the many-one reducibility, $\overline{\mathbf{K}} \leq_m B$. Hence by Proposition 1, B is also productive.
Proof of 2: Let the natural numbers be represented in unary. Let $\overline{\mathbf{K}} \leq_m A$ (via F). Then, there exists a total recursive function $g : \Sigma^* \to \{1\}^*$ such that, for all $x \in \Sigma^*$,

$$\mathcal{L}(M_{g(x)}) = F^{-1}(\mathcal{L}(M_x)).$$

The proof of the existence of function g can be seen in Theorem V(a) [25] page 84. Let the function $\Psi : \Sigma^* \to \Sigma^*$ be defined by, for all $x \in \Sigma^*$, $\Psi(x) = F(g(x))$. The function Ψ is a total recursive function since it is the composition of two total recursive functions with appropriate domains and ranges. The function Ψ is actually a productive function for A. This is seen as follows. Let $x \in \Sigma^*$; and suppose that $\mathcal{L}(M_x) \subseteq A$. Then, $\mathcal{L}(M_{g(x)}) \subseteq F^{-1}(A) \subseteq \overline{\mathbf{K}}$. By the productive property of $\overline{\mathbf{K}}$ using productive function $\mathbf{I}_{\{1\}^*}$, $g(x) \in \overline{\mathbf{K}} - \mathcal{L}(M_{g(x)})$. Hence, $\Psi(x) = F(g(x)) \in A$. But $\Psi(x) \notin \mathcal{L}(M_x)$, since otherwise,

$$\Psi(x) = F(g(x)) \Rightarrow g(x) \in F^{-1}(\mathcal{L}(M_x)) = \mathcal{L}(M_{g(x)}),$$

contradicting, $g(x) \in \overline{\mathbf{K}} - \mathcal{L}(M_{g(x)})$. Hence, as was to be verified,

$\Psi(x) \in A - \mathcal{L}(M_x)$.

Proof of 3: Since $\overline{\mathbf{K}}$ is not RE and $\overline{\mathbf{K}} \leq_m A$ by assumption, A is also not RE. The remainder of the proof is essentially the same as that of Theorem X [25] pages 90–91. It is given here, for the convenience of the reader.

Let $\Psi : \Sigma^* \to \Sigma^*$ be a productive total recursive function for A. A total recursive function $g : \mathbb{N} \to \Sigma^*$ can be computed inductively as follows. Let x_0 be some Gödel index for Φ. Then, $\Phi = \mathcal{L}(M_{x_0}) \in A$; and hence, $\Psi(x_0) \in A - \mathcal{L}(M_{x_0})$. Let $g(0) = \Psi(x_0)$. To compute $g(n+1)$, do the following. Let x_{n+1} be a Gödel index, for the finite set $\{g(0), \ldots, g(n)\} \subseteq A$. Let $g(n+1) = \Psi(x_{n+1})$. Then $\mathcal{L}(M_{x_{n+1}}) \subseteq A$; and hence, $g(n+1) = \Psi(x_{n+1}) \in A - \{g(0), \ldots, g(n)\}$. Since the function g as defined is one-to-one, the set $\{g(n) \mid n \geq 0\}$ is an infinite RE subset of A. □

3.2 The predicate "= {0, 1}*" for SRE

To make our results stronger and more applicable, we first study the sets of valid and invalid computations of Turing machines. Unlike the definition stated in [12] and [16], we define the sets of valid and invalid computations of Turing machines on given inputs. This refined definition enables us to investigate the complexity/undecidability of the restricted language predicate:

> testing equivalence to $\{0, 1\}^*$ for languages whose complements' cardinalities are less than or equal to one (denoted by "$= \{0, 1\}^* \mid_{|L^c| \leq 1}$").

The instances of this restricted predicate have very important semantic properties: they are the simplest regular sets. These restrictions make the predicate more widely applicable: for example, they directly apply to promise problems, predicates on regular sets, and descriptional complexity of language descriptors.

Throughout this section, $M = (Q, \Sigma, T, \delta, q_0, B, F)$ is a single tape deterministic Turing machine where:

1. Q is M's nonempty finite set of states;
2. $q_0 \in Q$ is M's unique start state;
3. $F \subseteq Q$ is M's set of accepting states. Each one in F is final;
4. M's input alphabet is Σ and T is M's tape alphabet where $\Sigma \subseteq T$;
5. $B \in T$ is the blank symbol;
6. $\delta : ((Q - F) \times T) \mapsto (Q \times T \times \{L, R\})$ is the transition function where L is the left shift and R is the right shift; and
7. $\Delta_M = T \cup (Q \times T) \cup \{\#\}$ where the sets T, $(Q \times T)$ and $\{\#\}$ are pairwise disjoint. $\Delta'_M = \Delta_M - \{\#\}$

Definition 5 Let M be any fixed deterministic Turing machine. For all $w \in \Sigma^+$, letting $w = w_1 w_2 w_3 \ldots w_k$ where $w_j \in \Sigma$ $(1 \leq j \leq k)$, *the set of valid computations of M on w* denoted by $VALCM(w)$, is the set of strings of the form $\#id_0\#id_1\#id_2 \cdots \#id_n\#$ such that

1. each id_i $(1 \leq i \leq n)$ is an ID[1] of M
2. $id_0 = (q_0, w_1)w_2 w_3 \ldots w_k$ is the initial ID of M on w
3. id_n is a final ID
4. $id_i \vdash_M id_{i+1}$[2] for $0 \leq i < n$

[1] The definition of an *ID* can be seen in [16].

[2] $\vdash_M$ represents a move of M. The definition of $id_i \vdash_M id_{i+1}$ can be seen in [16].

The set of invalid computations of M on w denoted by $INVALCM(w)$, is the complement of $VALCM(w)$ with respect to Δ_M^*.
We write $a \vdash_M bc$ where $a, b, c \in \Delta'_M$ if and only if $a \in (Q \times T)$ is the rightmost letter of an ID, $\delta(a) = (q_i, b, R)$ and $c = (q_i, B)$.
We write $ab \vdash_M c$ where $a, b, c \in \Delta'_M$ if and only if a is the leftmost letter of an ID and

1. if $a, b \notin (Q \times T)$, then $c = a$;
2. if $a \in (Q \times T)$ and $\delta(a) = (q_i, t, L)$, then $c = (q_i, B)$;
3. if $a \in (Q \times T)$ and $\delta(a) = (q_i, t, R)$, then $c = t$;
4. if $b \in (Q \times T)$ and $\delta(b) = (q_i, t, L)$, then $c = (q_i, a)$;
5. if $b \in (Q \times T)$ and $\delta(b) = (q_i, t, R)$, then $c = a$;

Or, b is the rightmost letter of an ID and

1. if $a, b \notin (Q \times T)$, then $c = b$;
2. if $a \in (Q \times T)$ and $\delta(a) = (q_i, t, L)$, then $c = b$;
3. if $a \in (Q \times T)$ and $\delta(a) = (q_i, t, R)$, then $c = (q_i, b)$;
4. if $b \in (Q \times T)$ and $\delta(b) = (q_i, t, L)$, then $c = t$.

We write $abc \vdash_M d$ where $a, b, c, d \in \Delta'_M$ if and only if abc is an infix of an ID and

1. if $a, b, c \notin (Q \times T)$, then $d = b$;
2. if $a \in (Q \times T)$ and $\delta(a) = (q_i, t, R)$, then $d = (q_i, b)$;
3. if $a \in (Q \times T)$ and $\delta(a) = (q_i, t, L)$, then $d = b$;
4. if $c \in (Q \times T)$ and $\delta(a) = (q_i, t, L)$, then $d = (q_i, b)$;
5. if $c \in (Q \times T)$ and $\delta(c) = (q_i, t, R)$, then $d = b$;
6. if $b \in (Q \times T)$ and $\delta(b) = (q_i, t, L)$ or $\delta(b) = (q_i, t, R)$, then $d = t$.

□

Intuitively, the notation $abc \vdash_M d$ means three consecutive letters of an ID determine one letter of the next ID. By checking every three consecutive letters of id_i, if the corresponding letter of id_{i+1} is always the correct one (i.e., $abc \vdash_M d$ is true for every three consecutive letters of id_i), we know $id_i \vdash_M id_{i+1}$. The notations $a \vdash_M bc$ and $ab \vdash_M c$ are used to handle the boundary cases. For example, $a \vdash_M bc$ means the head of Turing machine M is scanning the rightmost letter of the input, rewriting it to b, and moving to the right. So, $c = (q, B)$ where q is a state of M and B is the blank symbol. Since M is a deterministic Turing machine, *VALCM(w)* only contains one single string when M accepts w; otherwise *VALCM(w)* is the empty set. Hence, *INVALCM(w)* is either Δ_M^* or $\Delta_M^* - \{t\}$ where $t \in \Delta_M^*$. The following proposition shows that the class of synchronized regular languages contains *INVALCM(w)* very efficiently. The intuitive explanation of the languages L_1 through L_5 in the following proposition can be found later in the proof.

Proposition 3 1. $INVALCM(w) = L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$, *where the language* L_j $(1 \leq j \leq 5)$ *is defined as follows:*

$$
\begin{aligned}
L_1 &= \Delta_M^* - \{\#\} \cdot (T^* \cdot (Q \times T) \cdot T^* \{\#\})^+ \\
L_2 &= \Delta_M^* - \Delta_M^* \cdot \{\#\} \cdot T^* \cdot (F \times T) \cdot T^* \cdot \{\#\} \\
L_3 &= \{\lambda\} \cup ((\Delta_M - \{\#\}) \cup \{\#\} \cdot ((\Delta_M - \{(q_0, w_1)\}) \cup \{(q_0, w_1)\} \\
&\quad \cdot ((\Delta_M - \{w_2\}) \cup \cdots \cup \{w_{k-1}\} \\
&\quad \cdot ((\Delta_M - \{w_k\}) \cup \{w_k\} \cdot \Delta'_M) \cdot \cdots))) \cdot \Delta_M^* \\
L_4 &= \Delta_M^* \cdot \Delta'_M \cdot \{x\#y \mid x, y \in \Delta_M'^* \text{ and } |x| = |y|\} \cdot \{\#\} \cdot \Delta_M^*
\end{aligned}
$$

$$\cup$$

$$\Delta_M^* \cdot \{\#\} \cdot \{x\#y \mid x, y \in \Delta_M'^* \text{ and } |x| = |y|\} \cdot \Delta_M' \cdot \Delta_M' \cdot \Delta_M^*$$

$$L_5 = L_{5.1} \cup L_{5.2} \cup L_{5.3} \cup L_{5.4} \text{ where}$$

$$L_{5.1} = \bigcup_{\substack{a,b,c \in \Delta_M' \\ a \notin (Q \times T) \\ or\ a \nvdash_M bc}} \Delta_M^* \cdot \{\#\} \cdot \{ua\#vbc \mid u, v \in \Delta_M'^* \text{ and } |u| = |v|\} \cdot \{\#\} \cdot \Delta_M^*$$

$$L_{5.2} = \bigcup_{\substack{a,b,c \in \Delta_M' \\ ab \nvdash_M c}} \Delta_M^* \cdot \{\#ab\} \cdot \Delta_M'^* \cdot \{\#c\} \cdot \Delta_M^*$$

$$L_{5.3} = \bigcup_{\substack{a,b,c,d,e \in \Delta_M' \\ ab \nvdash_M c\ or \\ b \vdash_M de}} \Delta_M^* \cdot \{\#\} \cdot \{uab\#vc \mid u, v \in \Delta_M'^* \text{ and } |u| = |v| - 1\} \cdot \{\#\} \cdot \Delta_M^*$$

$$L_{5.4} = \bigcup_{\substack{a,b,c,d \in \Delta_M' \\ abc \nvdash_M d}} \Delta_M^* \cdot \{\#uabcw\#vd \mid u, v, w \in \Delta_M'^* \text{ and } |u| = |v| - 1\} \cdot \Delta_M^*$$

2. *The languages L_1, L_2 and L_3 are regular sets. The language L_1 and L_2 depend only on M. There exists a regular expression $N_{M,w}$ such that $\mathcal{L}(N_{M,w}) = L_3$ and $N_{M,w}$ is constructible from w deterministically in time $O(|w|log|w|)$.*
3. *L_4 and L_5 depend only on M and can be generated by synchronized regular expressions.*
4. *A synchronized regular expression e is constructible deterministically from w in time $O(|w|log|w|)$ such that $\mathcal{L}(e) = INVALCM(w)$.*

Proof of 1: The proof of $L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5 \subseteq INVALCM(w)$ is straightforward by the definition of *INVALCM(w)*.

All strings in L_1 are not of the form $\#id_0\#id_1\#id_2 \cdots \#id_n\#$ where $id_i (1 \leq i \leq n)$ is an ID of M.
All strings in L_2 do not end with $id_n\#$ where id_n is a final ID of M.
All strings in L_3 do not start with $\#id_0$ where id_0 is the initial ID of M on w
Every string in L_4 has an infix $\#x\#y\#$ where $x, y \in \Delta_M'^*$ such that $|x| > |y|$ or $|y| - |x| > 1$.
Every string in L_5 has an infix $\#x\#y\#$ where $x, y \in \Delta_M'^*$ such that $x \nvdash_M y$.
$L_{5.1}$ covers the case when $|x| = |y| - 1$, an error causing $x \nvdash_M y$ shows at the rightmost part of x and y.
$L_{5.2}$ covers the case that an error causing $x \nvdash_M y$ shows at the leftmost part of x and y.
$L_{5.3}$ covers the case when $|x| = |y|$, an error causing $x \nvdash_M y$ shows at the rightmost part of x and y.
$L_{5.4}$ covers the case that an error causing $x \nvdash_M y$ shows at the middle of x and y.

The proof of $INVALCM(w) \subseteq L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$:
$\forall t \in INVALCM(w)$, there are only two possibilities:

1. t is not of the form $\#id_{k_1}\#id_{k_2}\#id_{K_3} \cdots \#id_{k_n}\#$ where $id_{k_i} (1 \leq i \leq n)$ is an ID of M. Then $t \in L_1$.
2. t is of the form above. There are only two possibilities:
   (a) t does not end with $id_n\#$ where id_n is a final ID of M. Then, $t \in L_2$.
   (b) t ends with $id_n\#$. Then, there are only two cases:
      (i) t does not start with $\#id_0$ where id_0 is the initial ID of M on w. Then, $t \in L_3$.

(ii) t starts with $\#id_0$. Then, let $t = \#id_0\#id_1\#id_2 \cdots \#id_n\#$.
$t \in INVALCM(w) \Rightarrow \exists$ a leftmost $i (0 \leq i < n)$ such that $id_i \nvdash_M id_{i+1}$.
If $|id_i| > |id_{i+1}|$ or $|id_{i+1}| - |id_i| > 1$, then $t \in L_4$.
Otherwise, $\exists x \in \Delta'_M$ which is the leftmost error in id_{i+1} so that $id_i \nvdash_M id_{i+1}$.
If $|id_i| < |id_{i+1}|$
(A) x is the first letter of id_{i+1}, then $t \in L_{5.2}$
(B) x is one of the last two letters of id_{i+1}, then $t \in L_{5.1}$
(C) otherwise, $t \in L_{5.4}$
If $|id_i| = |id_{i+1}|$
(A) x is the first letter of id_{i+1}, then $t \in L_{5.2}$
(B) x is the last letter of id_{i+1}, then $t \in L_{5.3}$
(C) otherwise, $t \in L_{5.4}$

Proof of 2, 3, and 4: From the definition of L_3, we can see that the regular expression $N_{M,w}$ accepting L_3 contains $O(|w|)$ parentheses. Hence, we need $O(|w|log|w|)$ time to encode and count these parentheses. From Definition 1, SRE languages are closed under union and concatenation efficiently. It is obvious that every regular language is an SRE language. From Example 2.2, it is easy to see that we can efficiently construct an SRE to specify the language $L_1 \cup L_2 \cup L_3 \cup L_4$. Now, we give a synchronized regular expression to specify a language that is central to the specification of L_5. For simplicity, assume the input alphabet of the Turing machine $\Sigma = \{0, 1\}$. For any $a, b, c, d \in \{0, 1\}$, the SRE $(0+1)^x abc(0+1)^*\#(0+1)^x(0+1)d$ specifies the language $\{uabcw\#vd \mid u, w, v \in \{0, 1\}^*, |u| = |v|-1\}$ over language alphabet $\{0, 1, \#\}$. From this synchronized regular expression, we can construct a synchronized regular expression to specify L_5 by concatenation with regular sets and union with SRE languages. Hence, we can efficiently construct a synchronized regular expression to specify $L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$. □

By this proposition and the following theorem, we show that even for a polynomial-time recognizable subset D' of $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$ where each element in D' generates either $\{0, 1\}^*$ or $\{0, 1\}^* - \{w\}$ $(w \in \{0, 1\}^*)$, the predicate "$= \{0, 1\}^*$" is already productive. This means the predicate "$= \{0, 1\}^* \mid_{|L^c| \leq 1}$" is not recursively enumerable for $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, independent of the complexity of testing whether an instance is in D'. Results of this type occur throughout this paper and have many applications, especially for promise problems. Moreover, since synchronized regular expressions are recursive language descriptors, the predicate "$\neq \{0, 1\}^*$" is recursively enumerable.
It is worth noticing that the languages L_1 through L_5 in Proposition 3 are very simple languages. So we can easily apply the results of this paper to any class of language descriptors $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D})$ contains the language $\{x \# y \mid x, y \in (\Sigma - \{\#\})^*$ and $|x| = |y|\}$ and $\mathcal{L}(\mathcal{D})$ is closed under union, and concatenation with regular sets. For example, the results of this paper can be applied to one-reversal bounded one-counter machines, and real-time one-way cellular automata (defined in [20]).

Theorem 3.1 *There exists a subset D' of* $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$ *such that*

1. $D' \in \boldsymbol{P}$;
2. $\forall d \in D'$, $\mathcal{L}(d) \subseteq \{0, 1\}^*$ *and* $|\{0, 1\}^* - \mathcal{L}(d)| \leq 1$; *and*
3. $\overline{\boldsymbol{K}} \leq_m \{< d > \mid d \in D', \mathcal{L}(d) = \{0, 1\}^*\}$

□

Proof of 2, 3: It is not hard to see we can efficiently code *INVALCM(w)* into alphabet $\{0, 1\}$. According to Proposition 3, a synchronized regular expression e is constructible deterministically in time $O(|w|log|w|)$ to accept the coded $INVALCM(w)$. Let D' be the set of

all possible e. Since M is a deterministic Turing machine, we know $|\mathcal{L}(e)^c| \leq 1$ and $\mathcal{L}(e) = \{0, 1\}^*$ if and only if M does not accept w.

Proof of 1: Let e be constructed in a certain way to accept $L_1, \ldots, L_5$ so that e has a special format. For example, e must contain 5 easily separable sub-expressions such that the first sub-expression accepts L_3, and the remaining sub-expressions accept L_1, L_2, L_4, L_5 in this exact order. Since L_1, L_2, L_4, and L_5 only depend on the fixed Turing machine M, one can determine the input w of M from e in polynomial time in $|e|$ by reading the first sub-expression of e. So for any synchronized regular expression e_d, if one cannot determine an input w from e_d, then $e_d \notin D'$. Otherwise, one can determine w according to the format of e_d and construct a synchronized regular expression e_w from w and M (M is fixed) in polynomial time so it accepts the coded *INVALCM(w)*. Make sure that e_w must contain 5 easily separable sub-expressions such that the first sub-expression accepts L_3, and the remaining sub-expressions accept L_1, L_2, L_4, L_5 in this exact order. $e_d \in D'$ if and only if $e_d = e_w$. This shows that $D' \in \mathbf{P}$. □

Della Penna et al. also introduced a proper subclass of SRE, namely the 1-level or "flat" SRE in [6]. 1-SRE are a yet useful but much less complex subclass of SRE. Definition 6 review the definition of a 1-level synchronized regular expression. From the proof of Proposition 3 and Theorem 3.1, it is not hard to see corollary 1 holds.

Definition 6 [6] *1-level synchronized regular expressions(1-SRE)* are SRE where variables and exponents cannot be nested (i.e., variables and exponents cannot appear inside an exponentiated expression or in the expression that is bound to a variable). □

Corollary 1 *The predicate "$= \{0, 1\}^* \; |_{|L^c| \leq 1}$" is productive for 1-SRE.* □

Proof In Example 2.2 and the proof of Proposition 3, the synchronized regular expressions we present are 1-SRE. A 1-SRE language concatenation with a regular set or union with a regular set is still a 1-SRE language. A 1-SRE language union with a 1-SRE language is still a 1-SRE language. So we can construct a 1-level synchronized regular expression to accept $L_1 \cup L_2 \cup L_3 \cup L_4 \cup L_5$ defined in Proposition 3. □

Therefore, all the results for SRE in this paper hold for 1-SRE.

4 Language predicates for SRE

In this section, we show that many important language predicates are as hard as the predicate "$= \{0, 1\}^* \; |_{|L^c| \leq 1}$" for SRE. This section consists of three major theorems. Theorem 4.1 shows the productiveness of testing equivalence and containment to any fixed unbounded regular set for SRE. Theorem 4.3 gives widely applicable sufficient conditions for proving productiveness results for SRE. One condition of Theorem 4.3 is that the language predicates need to be true for only one regular set $\{0, 1\}^*$. Theorem 4.5 shows how to prove productiveness results for predicates that are not true for any regular/context-free languages by giving two interesting examples related to multi-pattern languages.

The following definition from [15] is necessary for Theorem 4.1.

Definition 7 A regular set $R_0 \subseteq \{0, 1\}^*$ is unbounded if and only if there exist strings $r, s, x, y \in \{0, 1\}^*$ such that $R_0 \supseteq \{r\} \cdot \{0x, 1y\}^* \cdot \{s\}$. □

Theorem 4.1 *Let R_0 be any fixed unbounded regular set over $\{0, 1\}$. There exists a subset S' of $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$ such that*

1. $S' \in \boldsymbol{P}$;
2. $\forall d \in S', |R_0 - \mathcal{L}(d)| \leq 1$;
3. $\overline{\boldsymbol{K}} \leq_m \{< d > | \ d \in S', \mathcal{L}(d) = R_0\}$; *and*
4. $\overline{\boldsymbol{K}} \leq_m \{< d > | \ d \in S', \mathcal{L}(d) \supseteq R_0\}$.

□

Proof The proof is similar to that used in [17] to show that the predicate "$= L_0$" is undecidable for context-free grammars where L_0 is any fixed context-free language with unbounded regular subset. Since R_0 is unbounded, from Lemma 7, there exist $r, s, x, y \in \{0, 1\}^*$ such that $\{r\} \cdot \{0x, 1y\}^* \cdot \{s\} \subseteq R_0$. $\forall e_1 \in D'$ where D' is defined in Theorem 3.1, we can efficiently construct a synchronized regular expression e_2 such that

$$\mathcal{L}(e_2) = \{r\} \cdot h(\mathcal{L}(e_1)) \cdot \{s\} \cup R_0 \cap \overline{\{r\} \cdot \{0x, 1y\}^* \cdot \{s\}}$$

where $h : \{0, 1\}^* \mapsto \{0, 1\}^*$ is the homomorphism defined by $h(0) = 0x$ and $h(1) = 1y$. For any $e_1 \in D'$, we can construct e_2 in polynomial time in $|e_1|$ since R_0, x, y, s and r are fixed constants. Let S' be the set of e_2. $D' \in \mathbf{P} \Rightarrow S' \in \mathbf{P}$. If $\mathcal{L}(e_1) = \{0, 1\}^*$, then $\mathcal{L}(e_2) = R_0$; otherwise, $\mathcal{L}(e_1) = \{0, 1\}^* - \{w\}$. Hence, $\mathcal{L}(e_2) = R_0 - \{rh(w)s\}$. □

Theorem 4.1 shows that for any fixed unbounded regular set R_0, the predicates "$= R_0$" and "$\supseteq R_0$" are productive even for a polynomial-time recognizable subset of $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$ where each element generates either R_0 or $R_0 - \{w\}$ ($w \in \{0, 1\}^*$). We believe this result has significant practical meanings since in reality, finding an approximation that differs from R_0 by a finite set is often done and very interesting.

The proof of Theorem 4.1 can be easily applied to any fixed language L_0 with an unbounded regular subset as long as the language $L_0 \cap \overline{\{r\} \cdot \{0x, 1y\}^* \cdot \{s\}}$ can be generated by a synchronized regular expression. Here, we give an example to show that Theorem 4.1 works for many non-regular languages. Extended regular expressions (EXREGs) are introduced by Câmpeanu et al [4] and are closed under intersection with regular sets [3]. It is easy to see that SRE contain EXREGs effectively since variable bindings can function as backreferences (defined in [4]). Hence, for any language L_0 generated by an extended regular expression, the language $L_0 \cap \overline{\{r\} \cdot \{0x, 1y\}^* \cdot \{s\}}$ can be generated by a synchronized regular expression. Hence, we can get the following corollary.

Corollary 2 *Let L_0 be any extended regular language (defined in [4]) over {0, 1} with an unbounded regular subset. The predicates "$=L_0$" and "$\supseteq L_0$" are productive for* $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$. □

It may be practically more relevant to ask for an approximating minimization algorithm between SRE and DFA/CFG/EXREGs, i.e., given a synchronized regular expression e, finding a DFA/CFG/EXREG accepting $\mathcal{L}(e)$ whose size is bounded by $f(M)$ where $f : \mathbb{N} \to \mathbb{N}$ is a recursive function and M is the size of a minimal DFA/CFG/EXREG accepting $\mathcal{L}(e)$. The results of testing equivalence and containment to some fixed language L_0 also enable us to show there is no approximating minimization algorithm between SRE and DFA/CFG/EXREGs accepting L_0. For simplicity, we only show the following theorem for the case $L_0 = \{0, 1\}^*$.

Theorem 4.2 *Let $f : \mathbb{N} \to \mathbb{N}$ be a recursive function. Then, there is no algorithm for solving the f-bounded approximating minimization problem between SRE and DFA/CFG/EXREGs.* □

Proof We only prove there is no algorithm for solving the approximating minimization problem between SRE and CFG. For DFA, the proof is easier since the universality problem is decidable for DFA. For EXREGs, the proof is similar. Assume there is an algorithm for solving the approximating minimization problem between SRE and CFG. Then, for any synchronized regular expression generating a context-free language L_0, one can find an equivalent CFG of size K accepting L_0, such that $K \leq f(M)$ where M is the size of a minimal CFG accepting L_0. In this case, let $L_0 = \{0, 1\}^*$. We know the size of a minimal CFG accepting $\{0, 1\}^*$ is 8 (see the example in Sect. 2). Hence, for any synchronized regular expression e, we can run this algorithm and find an equivalent CFG d. For any context-free grammar with n nonterminals, nonterminals are denoted by s_u where u is a base 10 numeral without leading 0's representing integers $\{0, 1, \ldots, n-1\}$. If $|d| > f(8)$, then $\mathcal{L}(e) \neq \{0, 1\}^*$. Otherwise, $|d| \leq f(8)$. There exists a finite set T such that for any context-free grammar p where $|p| \leq f(8)$, $p \in T$. Hence, there exists a finite table telling if $\mathcal{L}(p) = \{0, 1\}^*$, for all $p \in T$. Since the table is finite, it is decidable to check which $p = d$ and if $\mathcal{L}(p) = \{0, 1\}^*$. if $\mathcal{L}(d) = \{0, 1\}^*$, then $\mathcal{L}(e) = \{0, 1\}^*$; otherwise, $\mathcal{L}(e) \neq \{0, 1\}^*$. This shows the universality problem is decidable for SRE, which is a contradiction. □

To better describe Theorem 4.3, we introduce the following notations. For any predicate Π on a class of languages over Σ, let Π_{left} denote the set $\{L \subseteq \Sigma^* \mid \exists L'$ where $\Pi(L') = true$ and $\exists a \in \Sigma^*$, such that $L = a\backslash L'\}$. Let Π_{right} denote the set $\{L \subseteq \Sigma^* \mid \exists L'$ where $\Pi(L') = true$ and $\exists a \in \Sigma^*$, such that $L = L'/a\}$. The notations $a\backslash L$ and L/a denote left and right quotients with a single letter, respectively.

Theorem 4.3 *Let Π be any non-trivial predicate on the regular sets, such that*

1. $\Pi(\{0, 1\}^*) = true$ *and*
2. $\mathcal{L}(\mathbf{REG}(\{\mathbf{0}, \mathbf{1}\})) - \Pi_{left} \neq \emptyset$ *or* $\mathcal{L}(\mathbf{REG}(\{\mathbf{0}, \mathbf{1}\})) - \Pi_{right} \neq \emptyset$

Then, there exists a subset S' of $\mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$ *such that*

1. $S' \in \boldsymbol{P}$;
2. $\forall d \in S'$, $\mathcal{L}(d)$ *is regular; and*
3. $\overline{\boldsymbol{K}} \leq_m \{< d > \mid d \in S', \Pi(\mathcal{L}(d)) = true\}$, *hence, testing if the predicate Π is true for SRE is productive.*

□

Proof The proof is similar to that used in [18] which shows the undecidability of many predicates on context-free languages which are true for $\{0, 1\}^*$. We only prove when $\mathcal{L}(\mathbf{REG}(\{\mathbf{0}, \mathbf{1}\})) - \Pi_{left} \neq \emptyset$, the theorem holds. The other part of the proof is very similar. Since $\mathcal{L}(\mathbf{REG}(\{\mathbf{0}, \mathbf{1}\})) - \Pi_{left} \neq \emptyset$, there exists a regular language L_f such that $L_f \notin \Pi_{left}$. Then, $\forall e_1 \in D'$ where D' is defined in Theorem 3.1, we can efficiently construct a synchronized regular expression e_2 such that

$$\begin{gathered}\mathcal{L}(e_2) = h(\mathcal{L}(e_1)) \cdot \{11\} \cdot \{0, 1\}^* \\ \cup \\ \{00, 01\}^* \cdot \{11\} \cdot L_f \\ \cup \\ \overline{\{00, 01\}^* \cdot \{11\} \cdot \{0, 1\}^*}\end{gathered}$$

where $h : \{0, 1\}^* \mapsto \{0, 1\}^*$ is the homomorphism defined by $h(0) = 00$ and $h(1) = 01$. Let S' be the set of e_2. Since L_f is a fixed regular set, we can determine e_1 from e_2 in polynomial

time in $|e_2|$. $D' \in \mathbf{P} \Rightarrow S' \in \mathbf{P}$. If $\mathcal{L}(e_1) = \{0, 1\}^*$, then $\mathcal{L}(e_2) = \{0, 1\}^*$. Hence, $\Pi(\mathcal{L}(e_2)) = true$; otherwise, $\mathcal{L}(e_1) = \{0, 1\}^* - \{w\}$. Hence, $\mathcal{L}(e_2)$ is regular and $h(w)11 \setminus \mathcal{L}(e_2) = L_f$. According to the definition of Π_{left}, $L_f \notin \Pi_{left} \Rightarrow \Pi(\mathcal{L}(e_2)) = false$. □

Theorem 4.3 is extremely useful for proving productiveness results of language class comparison problems for SRE with a promise that each synchronized regular expression is guaranteed to generate a regular language. We illustrate the power and applicability of Theorem 4.3.

Theorem 4.4 *The following predicates on the regular sets over {0, 1} satisfy the conditions of Theorem 4.3, i.e., for each of the following predicates, testing if the predicate is true for SRE is productive.*

1. *L is a star event, i.e.,* $L = (L)^*$;
2. *L is a code event, i.e., there exist strings* $w_1, \ldots, w_k \in \{0, 1\}^*$ *such that* $L = \{w_1, \ldots, w_k\}^*$;
3. *For all* $k \geq 1$, *L is a k-parsable event; and L is a locally parsable event;*
4. *L is an ultimate definite event, reverse ultimate definite event, or generalized ultimate definite event;*
5. *L is a comet event, reverse comet event, or generalized comet event;*
6. $L = \gamma(L)$, *where* $\gamma(L) = \{y \mid$ *there exists x in L such that* $\mid y \mid = \mid x \mid\}$;
7. *L is prefix closed, i.e.,* $L = \{x \mid$ *there exists y in* $\{0, 1\}^*$ *and* $x \cdot y \in L\}$;
8. *L is suffix closed, i.e.,* $L = \{y \mid$ *there exists x in* $\{0, 1\}^*$ *and* $x \cdot y \in L\}$;
9. *L is infix closed, i.e.,* $L = \{y \mid$ *there exists x, z in* $\{0, 1\}^*$ *and* $x \cdot y \cdot z \in L\}$;
10. *L is co-finite;*
11. *For all* $k \geq 1$, *L is a k-definite event, k-reverse definite event, or k-generalized definite event;*
12. *L is definite, reverse definite, or generalized definite event;*
13. *For all* $k \geq 1$, *L is a k-testable event;*
14. *For all* $k \geq 1$, *L is k-testable in the strict sense;*
15. *L is locally testable in the strict sense;*
16. *L is locally testable;*
17. *L is a star-free, non-counting, group-free, permutation-free, or LTO event;*
18. *For all* $k > 2$, *L is a CMk event;*
19. *L is accepted by some strongly connected deterministic finite automaton;*
20. *L is accepted by some permutation automaton;*
21. *L is a pure group event;*
22. $L = L^{rev}$; *and*
23. *L is dot-free, i.e., L is denoted by some* $(\cup, \cdot, *, -)$ *regular expression over {0, 1} with no occurrence of "·";*

□

Proof The definitions of the classes of regular sets of 2, 3, and 11 through 18 can be found in [22]. The definition of 4 can be found in [23], 5 in [24], 19 in [11], 20 in [29], and 21 in [21]. The proof for each of the above predicates consists of two parts. The first part consists of observing that the predicates are true for $\{0, 1\}^*$. The second part of the proof consists of showing that Π_{left} or Π_{right} is a proper subset of the regular sets which can be found in [18]. □

Since Theorem 4.3 only requires the predicates to be true for one regular set $\{0, 1\}^*$, we can use its proof to study the complexity/undecidability of predicates on many classes of

languages other than the regular sets. Two interesting examples are the following corollaries of the proof of Theorem 4.3. Due to the power and applicability of this proof and the dichotomization of its reduction, we can show Corollary 3 despite that MPL is an anti-AFL (anti-abstract family of languages, see [10] for definition) and k-pattern languages may not be closed under left or right derivatives ([19]), and any class of languages $\Gamma \subseteq \mathcal{L}(\mathbf{CFG}(\{\mathbf{0}, \mathbf{1}\}))$ that contains $\{0, 1\}^*$ satisfies Corollary 4 regardless of the closure properties of Γ.

Corollary 3 *$\{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d)$ is a k-pattern language for any $k \geq 1\}$ is productive, and $\{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d)$ is a multi-pattern language $\}$ is productive.* □

Proof Let $L_f = \{0^n 1^n \mid n \geq 0\}$. We know that L_f is an SRE language by Example 2.1. We also know that L_f is not a multi-pattern language and MPL is closed under left and right derivatives [19]. Hence, with the same construction in the proof of Theorem 4.3, if $\mathcal{L}(e_1) = \{0, 1\}^*$, then $\mathcal{L}(e_2) = \{0, 1\}^*$ which is a 1-pattern language. Otherwise, there exists a string $w \in \{0, 1\}^+$ such that $w \backslash \mathcal{L}(e_2) = L_f$. Since L_f is not a multi-pattern language and MPL languages are closed under left derivatives, $\mathcal{L}(e_2)$ is not a multi-pattern language. Hence, $\mathcal{L}(e_2)$ is not a k-pattern language for any $k \geq 1$. □

Corollary 4 *For any fixed set Γ where $\Gamma \subseteq \mathcal{L}(\mathbf{CFG}(\{\mathbf{0}, \mathbf{1}\}))$ and $\{0, 1\}^* \in \Gamma$,*
$\{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d) \in \Gamma\}$ is productive.
Thus, in particular,
$\{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d)$ is context-free $\}$, and
$\{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d)$ is regular $\}$ are productive. □

Proof Let $L_f = \{ww \mid w \in \{0, 1\}^*\}$. The SRE $(0 + 1)^*\%X \cdot X$ (X is a variable) specifies L_f. So with the same construction in the proof of Theorem 4.3, if $\mathcal{L}(e_1) = \{0, 1\}^*$, then $\mathcal{L}(e_2) = \{0, 1\}^*$. Hence, $\mathcal{L}(e_2) \in \Gamma$. Otherwise, there exists a string $w \in \{0, 1\}^+$ such that $w \setminus \mathcal{L}(e_2) = L_f$. Since L_f is not context-free and context-free languages are closed under left derivatives, $\mathcal{L}(e_2)$ is not context-free. Hence, $\mathcal{L}(e_2) \notin \Gamma$. □

So far, all language class comparison problems we have studied need to be true for only one regular set $\{0, 1\}^*$. The following theorem illustrates how we can investigate the complexity/undecidability of predicates that are not true for any regular set, or any context-free languages.

Theorem 4.5 *$\overline{K} \leq_m \{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d)$ is not regular but in $\mathcal{L}(\boldsymbol{MP(\{0,1\})})\}$, and $\overline{K} \leq_m \{< d >\mid d \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, $\mathcal{L}(d)$ is not context-free but in $\mathcal{L}(\boldsymbol{MP(\{0,1\})})\}$.* □

Proof Let $L_t = \{1\} \cdot \{ww \mid w \in \{0, 1\}^*\} \cup \{0\} \cdot \{0, 1\}^*$ and $L_f = \{0\}^* \cdot \{1\}^*$. Consider the multi-pattern $\pi = \{1xx, 0x\}$ where x is a variable. Then, $\mathcal{L}(\pi) = L_t$. Hence, L_t is a multi-pattern language. It is easy to see that L_t is not a context-free language. $\forall e_1 \in \mathbf{SRE}(\{\mathbf{0}, \mathbf{1}\})$, we can effectively construct a synchronized regular expression e_2 such that

$$\begin{aligned} \mathcal{L}(e_2) = & \{0\} \cdot h(\mathcal{L}(e_1)) \cdot \{11\} \cdot \{0, 1\}^* \\ & \cup \\ & \{0\} \cdot \{00, 01\}^* \cdot \{11\} \cdot L_f \\ & \cup \\ & L_t \cap \overline{\{0\} \cdot \{00, 01\}^* \cdot \{11\} \cdot \{0, 1\}^*} \end{aligned}$$

where $h : \{0,1\}^* \mapsto \{0,1\}^*$ is the homomorphism defined by $h(0) = 00$ and $h(1) = 01$. Extended regular expressions (EXREGs) are introduced by Câmpeanu et al [4] and are closed under intersection with regular sets [3]. The language $\{ww \mid w \in \{0,1\}^*\}$ can be specified by an extended regular expression $((0+1)^*) \setminus 1$ ($\setminus 1$ is a backreference used to match the same content as a previously matched subexpression). Hence, L_t can be specified by an EXREG. Hence, $L_t \cap \overline{\{0\} \cdot \{00,01\}^* \cdot \{11\} \cdot \{0,1\}^*}$ can be specified by an EXREG. It is easy to see that SRE contain EXREGs effectively since variable bindings can function as backreferences. Hence, $L_t \cap \overline{\{0\} \cdot \{00,01\}^* \cdot \{11\} \cdot \{0,1\}^*}$ can be specified by a synchronized regular expression. Since L_t, L_f and $\overline{\{0\} \cdot \{00,01\}^* \cdot \{11\} \cdot \{0,1\}^*}$ are fixed languages, the construction of e_2 can be done in polynomial time in $|e_1|$. If $\mathcal{L}(e_1) = \{0,1\}^*$, since $\{0\} \cdot \{0,1\}^* \subseteq L_t$, $\mathcal{L}(e_2) = L_t$. Hence, $\mathcal{L}(e_2)$ is a multi-pattern language but not context-free. Otherwise, $\exists w \in \{0,1\}^*$ such that $0\,h(w)11 \backslash \mathcal{L}(e_2) = L_f$. MPL is closed under left and right derivatives [19] $\Rightarrow$ $\mathcal{L}(e_2)$ is not a multi-pattern language. □

5 Descriptional complexity of SRE

In this section, we study the descriptional complexity of SRE using the special properties of the predicate "$= \{0,1\}^* \,|_{|L^c|\leq 1}$". Regular languages are the most commonly used formal languages. In [7], Freydenberger shows that there is no recursive trade-off between SRE and regular expressions. But it may be practically more relevant to ask the trade-off between SRE and a class of language descriptors accepting a particular subset of regular languages. The special properties of the predicate "$= \{0,1\}^* \,|_{|L^c|\leq 1}$" enable us to study such trade-offs since the language $\{0,1\}^*$ and $\{0,1\}^* - \{w\}$ are both co-finite and are the simplest regular languages. In Theorem 5.1, we show that there is no recursive trade-off between SRE and DFA. Then, Theorem 5.2 generalizes the proof of Theorem 5.1 and gives sufficient conditions for establishing non-recursive trade-offs between SRE and many classes of language descriptors. To illustrate the power and applicability of Theorem 5.2, we show that any class of language descriptors accepting languages satisfying any predicate listed in Theorem 4.4 satisfies the conditions of Theorem 5.2.

To show that our results are even more widely applicable, we tune Theorem 5.2 with slight changes to study the trade-off between SRE and multi-patterns. Here, we redefine $INVALCM(w)$ developed in Definition 5 to tune the conditions in Theorem 5.2 so they can fit the properties of multi-patterns (Lemmas 5.2 and 5.3). Intuitively, Definition 5 defines $INVALCM(w)$ as either $\{0,1\}^*$ or $\{0,1\}^* - \{w\}$. The redefined $INVALCM(w)$ in Definition 8 defines $INVALCM(w)$ as either $\{0,1\}^*$ or $\{0,1\}^* - \{w\} \cdot \{0,1\}^*$. Lemma 5.2 shows that $\{0,1\}^* - \{w\} \cdot \{0,1\}^*$ is a multi-pattern language. Lemma 5.3 shows that for any multi-pattern π generating $\{0,1\}^* - \{w\} \cdot \{0,1\}^*$, $|\pi| \geq |w| - 1$. With the tuned conditions, we establish Theorem 5.3 to show there is no recursive trade-off between SRE and multi-patterns. This is another example to show Proposition 3 is easily applicable.

The following lemma for DFA is well-known and needed for proving Theorem 5.1.

Lemma 5.1 *$\forall w \in \{0,1\}^*$, let $L_w = \{0,1\}^* - \{w\}$. For any DFA M that accepts L_w, $|M| \geq |w|$.* □

Theorem 5.1 *There exists a subset D' of **SRE({0,1})** such that*

1. *$D' \in \boldsymbol{P}$;*
2. *$\forall d \in D'$, $\mathcal{L}(d)$ is regular;*
3. *There exists no recursive function $f : \mathbb{N} \mapsto \mathbb{N}$ such that $\forall d \in D'$, for any minimal DFA M accepting $\mathcal{L}(d)$, $|M| \leq f(|d|)$; and*

4. *There exists a fixed constant* $C > 0$ *such that* $\{< d >| \ d \in D', \exists$ *a DFA* M *such that* $\mathcal{L}(M) = \mathcal{L}(d)$ *and* $|M| < C\}$ *is productive, hence, not recursively enumerable.*

□

Proof of 1, 2, and 3: Let D' be the same D' defined in Theorem 3.1. We know that every language in $\mathcal{L}(D')$ is either $\{0, 1\}^*$, or $\{0, 1\}^* - \{t\}$ where $t \in \{0, 1\}^*$ is the coded valid computation of a Turing machine N on an input w. Assume such a recursive function f stated in 3 exists. From Lemma 5.1, we know that $f(|d|) \geq |M| \geq |t|$. If the Turing machine N on w halts, let x denote the number of steps N takes to halt. Then, $|t| \geq x \cdot |w|$. Hence, $f(|d|) > x$. Hence, the halting problem is recursive, which is a contradiction.

Proof of 4: Let $C = 2$. For any $d \in D'$, if $\mathcal{L}(d) = \{0, 1\}^*$, then there exists a DFA M with a single state such that $\mathcal{L}(M) = \mathcal{L}(d)$. Hence, $|M| < C$; otherwise, $\mathcal{L}(d) = \{0, 1\}^* - \{t\}$ where $t \in \{0, 1\}^+$, then from Lemma 5.1, for any DFA M specifying $\mathcal{L}(d)$, $|M| \geq |t|$. Therefore, it is clear $|M| \geq C$. □

We establish the following theorem to generalize Theorem 5.1. Many non-recursive trade-offs between SRE and other classes of language descriptors can be proved using it.

Theorem 5.2 *Let* $\mathcal{D}$ *be any class of language descriptors over alphabet* $\{0, 1\}$ *such that*

1. *For any* $w \in \{0, 1\}^*$, $L_w = \{0, 1\}^* - \{w\} \in \mathcal{L}(\mathcal{D})$*; and*
2. *There exists a strictly increasing recursive function* $f : \mathbb{N} \mapsto \mathbb{N}$ *such that for any* $d \in \mathcal{D}$ *specifying* L_w, $f(|d|) > |w|$.

Then, there is no recursive trade-off between SRE and $\mathcal{D}$. □

Proof Assume there exists a recursive function $g : \mathbb{N} \mapsto \mathbb{N}$ such that for any synchronized regular expression e specifying the coded $INVALCM(w)$ (notice that we can efficiently code $INVALCM(w)$ into alphabet $\{0, 1\}$ and $|INVALCM(w)^c| \leq 1$), for any $d \in \mathcal{D}$ specifying $\mathcal{L}(e)$, $g(|e|) > |d|$. Since f is strictly increasing, we know $f(g(|e|)) > f(|d|) > |t|$ where $t \in VALCM(w)$. Clearly, the function $f \circ g$ remains a recursive function. $|t| > x$ where x is the number of steps that the Turing machine N takes to halt on w. Hence, $f(g(|e|)) > x$. Hence, the halting problem is recursive, which is a contradiction. □

The following corollary illustrates the power and applicability of Theorem 5.2.

Corollary 5 *Any class of language descriptors* $\mathcal{D}$ *that* $\mathcal{L}(\mathcal{D})$ *satisfies any predicate listed in Theorem 4.4 satisfies the conditions of Theorem 5.2, i.e., there is no recursive trade-off between SRE and* $\mathcal{D}$.

It is also interesting for us to investigate the trade-off between SRE and multi-patterns. To illustrate that our results are tunable and widely applicable, we slightly change the definition of *INVALCM(w)* as follows to investigate this problem. Intuitively, the redefined *INVALCM(w)* is either $\{0, 1\}^*$ or $\{0, 1\}^* - \{w\} \cdot \{0, 1\}^*$.

Definition 8 Recall the deterministic Turing machine $M = (Q, \Sigma, T, \delta, q_0, B, F)$ we mentioned in Sect. 3. For all $w \in \Sigma^+$, letting $w = w_1 w_2 w_3 \ldots w_k$ where $w_j \in \Sigma$ $(1 \leq j \leq k)$, the set of valid computations of M on w denoted by $VALCM'(w)$, is the set of strings of the form $\#id_0\#id_1\#id_2 \cdots \#id_n\# \cdot t$ such that

1. $t \in \Delta_M^*$

2. each id_i $(1 \leq i \leq n)$ is an ID of M
3. $id_0 = (q_0, w_1)w_2w_3 \ldots w_k$ is the initial ID of M on w
4. id_n is a final ID
5. $id_i \vdash_M id_{i+1}$ for $0 \leq i < n$

The set of invalid computations of M on w denoted by $INVALCM'(w)$ is the complement of $VALCM'(w)$ with respect to Δ_M^*. □

With this refined definition and the following two lemmas, we can study the trade-off between SRE and multi-patterns using a method similar to Theorem 5.2.

Lemma 5.2 *$\forall w \in \{0,1\}^+$, the language $L_w = \{0,1\}^* - \{w\} \cdot \{0,1\}^*$ is a multi-pattern language.* □

Proof Let $w = w_0w_1w_2 \cdots w_k$ where $w_i \in \{0,1\}(0 \leq i \leq k)$. Let $\bar{0} = 1$ and $\bar{1} = 0$. Consider the multi-pattern $\pi = \{\lambda, w_0, w_0w_1, \ldots, w_0w_1 \ldots w_{k-1}, \bar{w_0}x, w_0\bar{w_1}x, w_0w_1\bar{w_2}x, \ldots, w_0w_1w_2 \ldots w_{k-1}\bar{w_k}x\}$ where x is a variable. It is clear that $\mathcal{L}(\pi) = L_w$. □

Lemma 5.3 *For any multi-pattern π that generates the language $L_w = \{0,1\}^* - \{w\} \cdot \{0,1\}^*$ where $w \in \{0,1\}^+$, $|\pi| \geq |w| - 1$.* □

Proof Assume $|\pi| < |w| - 1$. Let w_L be the longest proper prefix of w. Then, we know $|w_L| = |w| - 1$ and $w_L \in L_w$. Hence, there is a pattern $\alpha \in \pi$ such that $w_L \in \mathcal{L}(\alpha)$. According to the assumption, $|\alpha| < |w_L|$ since otherwise $|\pi| \geq |\alpha| \geq |w| - 1$ which is a contradiction. Hence, there must be at least one variable occurring in α.

Let x be the leftmost variable occurred in α and V be the set of variables for π. Let $\alpha = P_1xP_2$ where $P_1 \in \{0,1\}^*$ and $P_2 \in (\{0,1\} \cup V)^*$. Since $w_L \in \mathcal{L}(\alpha)$, we know P_1 is a proper prefix of w. Hence, $\exists t \in \{0,1\}^*$ such that $P_1 \cdot t = w$.

Let x be substituted by t, then there exists a string in $\{w\} \cdot \{0,1\}^*$ that matches α, which is a contradiction. □

Theorem 5.3 *There exists a subset S' of **SRE({0,1})** such that*

1. *$S' \in \boldsymbol{P}$;*
2. *$\forall d \in S'$, $\mathcal{L}(d)$ is a multi-pattern language;*
3. *There exists no recursive function $f : \mathbb{N} \mapsto \mathbb{N}$ such that $\forall d \in S'$, for any minimal multi-pattern π specifying $\mathcal{L}(d)$, $|\pi| \leq f(|d|)$; and*
4. *There exists a fixed constant $C > 0$ such that $\{< d > | \; d \in S', \exists$ a multi-pattern π such that $\mathcal{L}(\pi) = \mathcal{L}(d)$ and $|\pi| < C\}$ is productive, hence, not recursively enumerable.*

□

Proof of 1, 2, and 3: Corresponding to Definition 8, L_1 through L_5 in Proposition 3 need to be changed slightly. Let $\Delta_1 = \Delta_M - (F \times T) - \{\#\}$ and $\Delta_2 = \Delta_M - (F \times T)$. Let

$$L'_1 = \Delta_M^* - \{\#\} \cdot (T^* \cdot (Q \times T) \cdot T^*\{\#\})^+ \cdot \Delta_M^*$$
$$L'_2 = \Delta_M^* - \Delta_M^* \cdot \{\#\} \cdot T^* \cdot (F \times T) \cdot T^* \cdot \{\#\} \cdot \Delta_M^*$$
$$L'_3 = \{\lambda\} \cup ((\Delta_M - \{\#\}) \cup \{\#\} \cdot ((\Delta_M - \{(q_0, w_1)\}) \cup \{(q_0, w_1)\}$$
$$\cdot ((\Delta_M - \{w_2\}) \cup \cdots \cup \{w_{k-1}\} \cdot ((\Delta_M - \{w_k\}) \cup \{w_k\} \cdot \Delta'_M) \cdot \cdot))) \cdot \Delta_M^*$$
$$L'_4 = \Delta_2^* \cdot \Delta'_1 \cdot \{x\#y \mid x \in \Delta_1^* \text{ and } y \in \Delta_M'^*, |x| = |y|\} \cdot \{\#\} \cdot \Delta_M^*$$

$$\cup$$

$$\cdot \Delta_2^* \cdot \{\#\} \cdot \{x\#y \mid x \in \Delta_1^* \text{ and } y \in \Delta_M'^*, |x| = |y|\} \cdot \Delta_M' \cdot \Delta_M' \cdot \Delta_M^*$$

$$L_5' = L_{5.1}' \cup L_{5.2}' \cup L_{5.3}' \cup L_{5.4}' \text{ where}$$

$$L_{5.1}' = \bigcup_{\substack{a,b,c \in \Delta_M' \\ a \notin (Q \times T) \\ \text{or } a \not\vdash_M bc}} \Delta_2^* \cdot \{\#\} \cdot \{ua\#vbc \mid u \in \Delta_1^*, v \in \Delta_M'^* \text{ and } |u| = |v|\} \cdot \{\#\} \cdot \Delta_M^*$$

$$L_{5.2}' = \bigcup_{\substack{a,b,c \in \Delta_M' \\ ab \not\vdash_M c}} \Delta_2^* \cdot \{\#ab\} \cdot \Delta_1^* \cdot \{\#c\} \cdot \Delta_M^*$$

$$L_{5.3}' = \bigcup_{\substack{a,b,c,d,e \in \Delta_M' \\ ab \not\vdash_M c \text{ or} \\ b \vdash_M de}} \Delta_2^* \cdot \{\#\} \cdot \{uab\#vc \mid u \in \Delta_1^*, v \in \Delta_M'^* \text{ and } |u| = |v| - 1\} \cdot \{\#\} \cdot \Delta_M^*$$

$$L_{5.4}' = \bigcup_{\substack{a,b,c,d \in \Delta_M' \\ abc \not\vdash_M d}} \Delta_2^* \cdot \{\#uabcw\#vd \mid u, w \in \Delta_1^*, v \in \Delta_M'^* \text{ and } |u| = |v| - 1\} \cdot \Delta_M^*$$

Since all states in F are final, for any string $s = \#id_0\#id_1\#id_2 \cdots \#id_n\#\cdot t \in VALCM'(w)$, id_n is the first ID that contains a letter in $(F \times T)$. So we can use this property to distinguish which part of s belongs to $\#id_0\#id_1\#id_2 \cdots \#id_n\#$ and which part of s belongs to t. Thus, it is not hard to see that $INVALCM'(w) = L_1' \cup L_2' \cup L_3' \cup L_4' \cup L_5'$ and $INVALCM'(w)$ can be expressed by a synchronized regular expression. Recall the set D' defined in Theorem 3.1. We can slightly change the set D' to S' so L(S') is the set of coded INVALCM'(w) over $\{0, 1\}$. The proof is very similar to the proof of Theorem 3.1. From Lemma 5.2, we know that every language in $\mathcal{L}(S')$ is a multi-pattern language. Assume such a recursive function f stated in 3 exists. From Lemma 5.3, we know that $f(|d|) \geq |\pi| \geq |t| - 1$ where $t \in VALCM(w)$. If the Turing machine M on w halts, let x denote the number of steps M takes. Hence, $|t| \geq x \cdot |w|$. Hence, $f(|d|) > x$. Therefore, the halting problem is recursive, which is a contradiction.

Proof of 4: Let $C = 2$. For any $d \in S'$, if $\mathcal{L}(d) = \{0, 1\}^*$, then there exists a multi-pattern $\{x\}$ where x is a variable and $\mathcal{L}(\{x\}) = \mathcal{L}(d)$; otherwise $\mathcal{L}(d) = \{0, 1\}^* - \{u\} \cdot \{0, 1\}^*$ where $u \in \{0, 1\}^+$, then from Lemma 5.3, for any multi-pattern π specifying $\mathcal{L}(d)$, $|\pi| \geq |u| - 1$. It is clear $|\pi| \geq 2$. □

6 Conclusion

In this paper, productiveness is employed, which is a stronger form of non-recursive enumerability. If a language predicate is productive, then the predicate is not recursively enumerable. Moreover, for the given language predicate, there is an effective procedure which, given as input a program enumerating an effective axiomatic system which proves only true values of the language predicate, produces a statement which cannot be proven by the given axiomatic system but which is still true with respect to the language predicate. We have revised the definition of the sets of invalid computations of Turing machines and used this definition to show that the predicate "$= \{0, 1\}^*\ |_{|L^c| \leq 1}$" is productive for SRE. This result enables us to establish the productiveness of many problems for SRE, especially promise problems. Using

the special properties of the predicate "$= \{0, 1\}^* \mid_{|L^c| \leq 1}$", non-recursive trade-offs between SRE and many language descriptors are also proved.

Funding We (Jingnan Xie, Harry B. Hunt, III) certify that we have no affiliations with or involvement in any organization or entity with any financial interest, or non-financial interest in the subject matter or materials discussed in this manuscript.

References

1. Aho, A.V.: Algorithms for finding patterns in strings. In: Van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science Vol A: Algorithms and Complexity, pp. 255–300. Elsevier, Amsterdam (1990). https://doi.org/10.1016/B978-0-444-88071-0.50010-2
2. Berglund, M., Van der Merwe, B.: Re-examining regular expressions with backreferences. Theoret. Comput. Sci. **940**, 66–80 (2023). https://doi.org/10.1016/j.tcs.2022.10.041
3. Câmpeanu, C., Santean, N.: On the intersection of regex languages with regular languages. Theor. Comput. Sci. **410**(24), 2336–2344 (2009). https://doi.org/10.1016/j.tcs.2009.02.022
4. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Found. Comput. Sci. **14**(6), 1007–1018 (2003). https://doi.org/10.1142/S012905410300214X
5. Carle, B., Narendran, P.: On extended regular expressions. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) Language and Automata Theory and Applications. LATA 2009. Lecture Notes in Computer Science, vol. 5457, pp. 279–289. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-00982-2_24
6. Della Penna, G., Intrigila, B., Tronci, E., Venturini Zilli, M.: Synchronized regular expressions. Acta Informatica **39**(1), 31–70 (2003). https://doi.org/10.1007/s00236-002-0099-y
7. Freydenberger, D.D.: Extended regular expressions: succinctness and decidability. Theory Comput. Syst. **53**(2), 159–193 (2013). https://doi.org/10.1007/s00224-012-9389-0
8. Freydenberger, D.D., Reidenbach, D.: Bad news on decision problems for patterns. Inf. Comput. **208**(1), 83–96 (2010). https://doi.org/10.1016/j.ic.2009.04.002
9. Freydenberger, D.D., Schmid, M.L.: Deterministic regular expressions with back-references. J. Comput. Syst. Sci. **105**, 1–39 (2019). https://doi.org/10.1016/j.jcss.2019.04.001
10. Ginsburg, S., Greibach, S.: Abstract families of languages. In: Studies in Abstract Families of Languages, pp. 1–32. Memoirs of the American Mathematical Society, Providence (1969). https://doi.org/10.1090/memo/0087
11. Hartmanis, J., Stearns, R.E.: Algebraic Structure Theory of Sequential Machines (Prentice-Hall International Series in Applied Mathematics). Prentice-Hall Inc (1966)
12. Hartmanis, J.: Context-free languages and Turing machine computations. J. Symb. Logic **37**(4), 759 (1972). https://doi.org/10.2307/2272443
13. Hartmanis, J.: On the succinctness of different representations of languages. In: Maurer, H.A. (ed.) Automata, Languages and Programming: Sixth Colloquium. ICALP 1979. Lecture Notes in Computer Science, vol. 71, pp. 282–288. Springer, Berlin, Heidelberg (1979). https://doi.org/10.1007/3-540-09510-1_22
14. Holzer, M., Kutrib, M.: Descriptional complexity—an introductory survey. In: Martín-Vide, C. (ed.) Scientific Applications of Language Methods. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory, vol. 2, pp. 1–58. World Scientific, Singapore (2010). https://doi.org/10.1142/9781848165458_0001
15. Hopcroft, J.E., Ullman, J.D.: Formal Languages and Their Relation to Automata. Addison-Wesley Longman Publishing Co. Inc., Boston (1969)
16. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)

17. Hunt, H.B., III., Rosenkrantz, D.J.: On equivalence and containment problems for formal languages. J. ACM **24**(3), 387–396 (1977). https://doi.org/10.1145/322017.322020
18. Hunt, H.B., III., Rosenkrantz, D.J.: Computational parallels between the regular and context-free languages. SIAM J. Comput. **7**(1), 99–114 (1978). https://doi.org/10.1137/0207007
19. Kari, L., Mateescu, A., Păun, G., Salomaa, A.: Multi-pattern languages. Theoret. Comput. Sci. **141**(1), 253–268 (1995). https://doi.org/10.1016/0304-3975(94)00087-Y
20. Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom. Lang. Comb. **7**(4), 549–560 (2002). https://doi.org/10.25596/jalc-2002-549
21. McNaughton, R.: The loop complexity of pure-group events. Inf. Control **11**(1), 167–176 (1967). https://doi.org/10.1016/S0019-9958(67)90481-0
22. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge (1971)
23. Paz, A., Peleg, B.: Ultimate-definite and symmetric-definite events and automata. J. ACM **12**(3), 399–410 (1965). https://doi.org/10.1145/321281.321292
24. Paz, A., Peleg, B.: On concatenative decompositions of regular events. IEEE Trans. Comput. **C–17**(3), 229–237 (1968). https://doi.org/10.1109/TC.1968.229096
25. Rogers, H., Jr.: Theory of Recursive Functions and Effective Computability. MIT Press, Cambridge (1987)
26. Schmid, M.L.: Inside the class of regex languages. Int. J. Found. Comput. Sci. **24**(07), 1117–1134 (2013). https://doi.org/10.1142/S0129054113400340
27. Schmid, M.L.: Characterising regex languages by regular languages equipped with factor-referencing. Inf. Comput. **249**, 1–17 (2016). https://doi.org/10.1016/j.ic.2016.02.003
28. Soare, R.I.: Recursively Enumerable Sets and Degrees. Springer, Berlin (1987)
29. Thierrin, G.: Permutation automata. Math. Syst. Theory **2**, 83–90 (1968). https://doi.org/10.1007/BF01691347