Theory of Computing Systems
https://doi.org/10.1007/s00224-024-10169-9

Pumping Lemmas Can be "Harmful"

Jingnan Xie[1] · Harry B. Hunt III[2] · Richard E. Stearns[2]

Accepted: 29 February 2024
© The Author(s) 2024

Abstract

A pumping lemma for a class of languages $\mathcal{C}$ is often used to show particular languages are not in $\mathcal{C}$. In contrast, we show that a pumping lemma for a class of languages $\mathcal{C}$ can be used to study the computational complexity of the predicate "$\in \mathcal{C}$" via highly efficient many-one reductions. In this paper, we use extended regular expressions (EXREGs, introduced in Câmpeanu et al. (Int. J. Foundations Comput. Sci. **14**(6), 1007–1018, 2003)) as an example to illustrate the proof technique and establish the complexity of the predicate "is an EXREG language" for several classes of languages. Due to the efficiency of the reductions, both productiveness (a stronger form of non-recursive enumerability) and complexity results can be obtained simultaneously. For example, we show that the predicate "is an EXREG language" is productive (hence, not recursively enumerable) for context-free grammars, and is Co-NEXPTIME-hard for context-free grammars generating bounded languages. The proof technique is easy to use and requires only a few conditions. This suggests that for any class of languages $\mathcal{C}$ having a pumping lemma, the language class comparison problems (e.g., does a given context-free grammar generate a language in $\mathcal{C}$?) are almost guaranteed to be hard. So, pumping lemmas sometimes could be "harmful" when studying computational complexity results.

Keywords Extended regular expressions · Pumping lemmas · Undecidability · Productiveness · EDT0L · Synchronized regular expressions

✉ Jingnan Xie
jingnan.xie@millersville.edu

Harry B. Hunt III
hunt@cs.albany.edu

Richard E. Stearns
thestearns2@gmail.com

[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

Published online: 05 April 2024

1 Introduction

Extended regular expressions (EXREGs) introduced by Campeanu et al. [1] are standard regular expressions augmented with backreferences (as defined in [2]) to match the same text again. Campeanu et al. showed that EXREGs represent a family of languages that is larger than the family of regular languages and is incomparable with the family of context-free languages. So it is desirable to study the language comparison problem between context-free languages and EXREGs (i.e., does a given context-free grammar generate an EXREG language?).

Similarly, another extension of regular expressions — synchronized regular expressions (SRE) is defined and studied in [3]. 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 with other extensions such as backreferences. Della Penna et al. used SRE to present a formal study of the backreferences extension and of a new extension called the *synchronized exponents* proposed by them. In [4], Carle showed that the family of languages expressed by SRE properly contains the family of languages expressed by EXREGs. So it is also desirable to study the language comparison problem between SRE and EXREGs (i.e., does a given synchronized regular expression generate an EXREG language?)

Lindenmayer systems (L-systems) were introduced by Aristid Lindenmayer in 1968 [5] to model the development of simple multi-cellular organisms in terms of division, growth, and death of individual cells. EDT0L systems (discussed in [6]) are a special type of L-systems with great research value (for example, see [7, 8]). How hard is it to determine whether a given EDT0L system generates an EXREG language?

All these problems can be solved due to the existence of a pumping lemma for EXREGs. In [1], a pumping lemma for the languages expressed by EXREGs is proven. Often, a pumping lemma for a class of languages $\mathcal{C}$ is used to show particular languages are not in $\mathcal{C}$. In contrast, we use this pumping lemma to study the complexity of the predicate "is an EXREG language" via highly efficient many-one reductions. Due to the efficiency of these reductions, both productiveness (a stronger form of non-recursive enumerability) and complexity results are obtained simultaneously. The proof technique we illustrate in this paper is easy to use and requires very little. This suggests that for any class of languages $\mathcal{C}$ having a pumping lemma, the predicate "$\in \mathcal{C}$" is almost guaranteed to be hard. Since developing pumping lemmas is still an important research topic in the theory of formal languages (for example, see [9–11]), the proof technique in this paper has the potential to be used for many classes of languages.

This paper is organized as follows.

In Section 2, we review the definitions of EXREGs, SRE, and EDT0L systems to make the paper more self-contained. The definition and importance of productiveness are discussed. Several preliminary definitions and notations are also explained.

In Section 3, we establish our major results on the predicate "is an EXREG language". We show that the predicate "is an EXREG language" is productive, hence non-recursively enumerable for non-deterministic 1-reversal bounded 1-counter machines, linear context-free grammars, context-free grammars, and SRE. We also show that the predicate "is an EXREG language" is Co-NEXPTIME-hard for context-free grammars generating bounded languages, and is PSPACE-hard for EDT0L systems generating bounded languages. It is worth mentioning that even for a polynomial time recognizable subset whose elements only generate bounded languages, the predicate "is an EXREG language" is already hard. This means that the problem is hard independent of the complexity of testing whether a language is bounded.

2 Definitions and Notations

In this section, we review the definitions of EXREGs, SRE, and EDT0L systems from [1, 3], and [6], respectively. The importance of productiveness and several preliminary definitions and notations in language theory are also explained. The reader is referred to [12] 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. Let **PSPACE** denote the class of sets that can be recognized using polynomial space by a Turing Machine. Let **NEXPTIME** denote the class of sets that can be recognized in exponential time by a non-deterministic Turing Machine. We use **Co-NEXPTIME** to denote the set of the complements of the languages in **NEXPTIME**. If A is many-one reducible to B, we write $A \leqslant_m B$; if this reduction is polynomial-time bounded, we write $A \leqslant_{ptime} B$.

Let $\mathcal{D}$ be a class of language descriptors that describe languages over Σ. In this paper, we only consider finite Σ. Then, $\forall d \in \mathcal{D}$, $\mathcal{L}(d) = \{w \in \Sigma^* \mid w \text{ is described by } d\}$ and $\mathcal{L}(\mathcal{D}) = \{L \subseteq \Sigma^* \mid \exists d \in \mathcal{D} \text{ such that } L = \mathcal{L}(d)\}$. $\forall d \in \mathcal{D}$, let $|d|$ denote the size of d. 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 $|d| = 8$).

A language class comparison problem 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 $\mathcal{L}(a) \in \mathcal{L}(\mathcal{D}_2)$?

Definition 1 An *non-deterministic 1-reversal bounded 1-counter machine* (denoted by N 1-rbd 1-CM) is a pushdown automaton where the cardinality of the stack alphabet is two(including the bottom symbol) and the machine makes at most one single reversal on the stack. Hence, the class of languages accepted by N 1-rbd 1-CMs is a proper subset of linear context-free languages. Throughout the paper, we use **N11CM** to denote the set of N 1-rbd 1-CMs with input alphabet $\{0, 1\}$.

Definition 2 The *synchronized regular expressions* on an alphabet Σ, a set of variables V and a set of exponents X are defined as follows:

$$\emptyset \in SRE \text{ (empty set)}$$

$$\lambda \in SRE \text{ (empty string)}$$

$$\forall a \in \Sigma : a \in SRE \text{ (letters)}$$

$$\forall v \in V : v \in SRE \text{ (variables)}$$

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 3 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 $0^x 1^x$ specifies the language $\{0^n 1^n \mid n \geq 0\}$.

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

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

The syntax of extended regular expressions (EXREGs) is defined in [1]. EXREGs are standard regular expressions augmented with *backreferences*. The backreference $\backslash n$ stands for the string previously matched by the regular expression between the n^{th} left parenthesis and the corresponding right parenthesis. A formal definition of matching a string with an EXREG is given in [4]. Here we give that definition with a slight modification. To present the definition, we need to define the following notation.

Definition 4 We use $\underset{i}{(}$ to denote the i^{th} left parenthesis and $\underset{i}{)}$ denote its corresponding right parenthesis. For an EXREG $e = \alpha(r)\beta$ where (is the i^{th} left parenthesis of e and) is its corresponding right parenthesis, we use $\underset{i}{(}r\underset{i}{)}$ to denote (r).[1]

As in [1] we assume that any occurrence of a backreference $\backslash m$ in an EXREG is preceded by $\underset{m}{)}$.

Definition 5 *Matching a string with an extended regular expression* is often defined as follows:

1. If t is a symbol in the alphabet, then t matches t;

[1] If the number of the parenthesis is easily attainable, we may omit the index of the parenthesis.

2. if r matches a string w, then $\underset{i}{(}r\underset{i}{)}$ matches w. Once $\underset{i}{(}r\underset{i}{)}$ matches a string w, the string w is assigned to $\backslash i$ and any occurrence of $\backslash i$ matches w;
3. if r_1 and r_2 are EXREGs, then $r_1 + r_2$ matches any string matched by either r_1 or r_2;
4. if r_1 and r_2 are EXREGs, then $r_1 r_2$ matches any string of the form xy where r_1 matches x and r_2 matches y; and
5. if r is an EXREG, then r^* matches any string of the form $x_1 x_2 ... x_n$ for any $n \geq 0$, where r matches each x_i $(1 \leq i \leq n)$.

Example 2.4 The EXREG $(0^+)(1^+)\backslash 1\backslash 2$ specifies the language $\{0^i 1^j 0^i 1^j \mid i, j > 0\}$.

Example 2.5 The EXREG $((0 + 1)^*) \backslash 1$ specifies the language $\{ww \mid w \in \{0, 1\}^*\}$.

Definition 6 A *finite substitution* σ over alphabet Σ is a mapping of Σ^* into the set of all finite nonempty languages (possibly over another alphabet Δ) defined as follows. For each letter $a \in \Sigma$, $\sigma(a)$ is a finite nonempty language, $\sigma(\lambda) = \{\lambda\}$ and for all $w_1, w_2 \in \Sigma^*$,

$$\sigma(w_1 w_2) = \sigma(w_1)\sigma(w_2).$$

For any language L over Σ, $\sigma(L) = \bigcup_{w \in L} \sigma(w)$.

If $\forall a \in \Sigma, \lambda \notin \sigma(a)$, the substitution σ is referred to as λ*-free or non-erasing*. If each $\sigma(a)$ contains a single string, σ is called a *morphism*.

In this paper, we only consider L-systems over the terminal alphabet $\{0, 1\}$. This restriction has been taken into account in the following definitions.

Definition 7 A *0L system* is a triple $G = (\{0, 1\}, \sigma, s)$ where σ is a finite substitution over $\{0, 1\}$ and $s \in \{0, 1\}^*$ is the axiom. The 0L system G generates the language

$$\mathcal{L}(G) = \{s\} \cup \sigma(s) \cup \sigma(\sigma(s)) \cup ... = \bigcup_{i \geq 0} \sigma^i(s).$$

A 0L system is *deterministic* or a *D0L system* if and only if σ is a morphism.

The letter E ("extended") in the name of an L system means that the use of nonterminals is allowed. Thus, an E0L system is a 0L system augmented with nonterminals.

Definition 8 An *E0L system* is a 4-tuple $G = (\{0, 1\}, V, \sigma, s)$ where V is the set of nonterminals (disjoint with $\{0, 1\}$), σ is a finite substitution over $V \cup \{0, 1\}$ and $s \in (V \cup \{0, 1\})^*$ is the axiom. The E0L system G generates the language

$$\mathcal{L}(G) = \bigcup_{i \geq 0} \sigma^i(s) \cap \{0, 1\}^*.$$

An E0L system is *deterministic* or a *ED0L system* if and only if σ is a morphism.

The letter T ("table") in the name of an L system means instead of having one finite substitution, the system has a finite number of finite substitutions.

Definition 9 A *T0L system* is a triple $G = (\{0, 1\}, P, s)$ where P is a finite set of finite substitutions such that for each $\sigma \in P$, $(\{0, 1\}, \sigma, s)$ is a 0L system. For a T0L system $G = (\{0, 1\}, P, s)$,

1. let $X = x_1 x_2 ... x_k$ $(k \geq 1)$ where x_i $(1 \leq i \leq k) \in \{0, 1\}$. Let σ be a finite substitution in P and let $Y \in \{0, 1\}^*$. We write $X \rightarrow_\sigma Y$ if there exist $y_1, y_2, ..., y_k \in \{0, 1\}^*$ such that $y_i \in \sigma(x_i)$ $(1 \leq i \leq k)$ and $Y = y_1 y_2 ... y_k$. We write $X \rightarrow_P Y$ if there exists $\sigma \in P$ such that $X \rightarrow_\sigma Y$;
2. $\rightarrow_P^*$ denotes the transitive and reflexive closure of the binary relation $\rightarrow_P$; and
3. $\mathcal{L}(G) = \{w \in \{0, 1\}^* \mid s \rightarrow_P^* w\}$.

An *ET0L system* is a 4-tuple $G = (\{0, 1\}, V, P, s)$ where V is the set of nonterminals (disjoint with $\{0, 1\}$), P is a finite set of finite substitutions over $V \cup \{0, 1\}$ and $s \in (V \cup \{0, 1\})^*$ is the axiom. For an ET0L $G = (\{0, 1\}, V, P, s)$,

1. let $X = x_1 x_2 ... x_k$ $(k \geq 1)$ where x_i $(1 \leq i \leq k) \in (V \cup \{0, 1\})$. Let σ be a finite substitution in P and let $Y \in (V \cup \{0, 1\})^*$. We write $X \rightarrow_\sigma Y$ if there exist $y_1, y_2, ..., y_k \in (V \cup \{0, 1\})^*$ such that $y_i \in \sigma(x_i)$ $(1 \leq i \leq k)$ and $Y = y_1 y_2 ... y_k$. We write $X \rightarrow_P Y$ if there exists $\sigma \in P$ such that $X \rightarrow_\sigma Y$;
2. $\rightarrow_P^*$ denotes the transitive and reflexive closure of the binary relation $\rightarrow_P$; and
3. $\mathcal{L}(G) = \{w \in \{0, 1\}^* \mid s \rightarrow_P^* w\}$.

An ET0L system is *deterministic* or an *EDTOL system* if every finite substitution in P is a morphism. Throughout this paper, let **EDT0L** denote the set of EDT0L systems over terminal alphabet $\{0, 1\}$.

For a better understanding of these definitions, we give several examples here.

Example 2.6 Let the D0L system $G = (\{0, 1\}, h, 01)$ with $h(0) = \{0\}$ and $h(1) = \{01\}$.
Hence, $h(01) = \{001\}$, $h(h(01)) = \{0001\}$, $h(h(h(01))) = \{00001\}$, ...
Then, $\mathcal{L}(G) = \{0^n 1 \mid n \geq 1\}$.

Example 2.7 Let the 0L system $G = (\{0, 1\}, h, 0)$ with $h(0) = \{\lambda, 1, 0, 00, 01\}$ and $h(1) = \{1, 10, 11\}$. Then $\mathcal{L}(G) = \{0, 1\}^*$.

Example 2.8 Let the EDT0L system $G = (\{0, 1\}, \{A, B, C, D\}, P, CD)$ where $P = \{h_1, h_2, h_3\}$ and
$h_1(0) = \{0\}$, $h_1(1) = \{1\}$, $h_1(A) = \{A\}$, $h_1(B) = \{B\}$, $h_1(C) = \{ACB\}$, $h_1(D) = \{DA\}$;
$h_2(0) = \{0\}$, $h_2(1) = \{1\}$, $h_2(A) = \{A\}$, $h_2(B) = \{B\}$, $h_2(C) = \{CB\}$, $h_2(D) = \{D\}$;
$h_3(0) = \{0\}$, $h_3(1) = \{1\}$, $h_3(A) = \{0\}$, $h_3(B) = \{1\}$, $h_3(C) = \{\lambda\}$, $h_3(D) = \{\lambda\}$.
Then $\mathcal{L}(G) = \{0^n 1^m 0^n \mid n \geq 0, m \geq n\}$.

At last, we discuss the definition and importance of productiveness. Productive sets and their properties are a standard topic in mathematical logic/recursion theory textbooks such as [13] and [14]. Productiveness is a recursion-theoretic abstraction of what causes Gödel's first incompleteness theorem to hold. Definition 10 recalls the definition of a productive set on $\mathbb{N}$, as developed in [13].

Definition 10 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 [13]) that are necessary for the research developed here are as follows:

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

The following proposition is proven in [15] and is used to prove productiveness results. 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 hold:*

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.*

3 On the Predicate "is an EXREG Language"

In this section, a meta theorem is developed to show the predicate "is an EXREG language" is as hard as the universality problem ("$= \{0, 1\}^*$") for many classes of languages under certain conditions. Several authors have investigated the existence and applicability of analogues of Rice's Theorem for different classes of languages. For example, in [15, 16], sufficient conditions are given for a language predicate to be as hard as the language predicate "$= \{0, 1\}^*$" such as requiring the language predicate to be closed under left or right derivatives. Here, we take a different approach and show that having a pumping lemma for a class of languages $\mathcal{C}$ could cause the predicate "$\in \mathcal{C}$" to be as hard as "$= \{0, 1\}^*$". Besides the predicate "$= \{0, 1\}^*$", the proof technique can also be applied to reductions of other sources. Since the proof technique requires very little to use, we believe it has the potential to have a wide range of applications.

The following lemma is necessary to prove Theorem 3.1. Both productiveness and complexity results can be derived from Theorem 3.1 due to the high efficiency of the reduction in its proof.

Lemma 3.1 [17] *EXREG languages are closed under intersection with regular sets.*

Theorem 3.1 *Let $\mathcal{D}$ be any class of language descriptors over alphabet $\{0,1\}$ such that*

1. *$\mathcal{L}(\mathcal{D})$ is efficiently closed under union, concatenation with regular sets and a 1-1 homomorphism $h : \{0,1\}^* \mapsto \{0,1\}^*$ defined by $h(0) = 00$ and $h(1) = 01$; and*
2. *there exists a language $L_f \in \mathcal{L}(\mathcal{D})$ such that $\forall w \in \{0,1\}^*$, the language $L_f \cdot \{w\}$ (or $\{w\} \cdot L_f$) is not an EXREG language.*

Then $\{d \mid d \in \mathcal{D}, \mathcal{L}(d) = \{0,1\}^*\} \leqslant_{ptime} \{d \mid d \in \mathcal{D}, \mathcal{L}(d)$ is an EXREG language$\}$.

Proof For any $G \in \mathcal{D}$, we can efficiently construct a $H \in \mathcal{D}$ such that

$$\begin{aligned}\mathcal{L}(H) \ &= \{0,1\}^* \cdot \{11\} \cdot h(\mathcal{L}(G)) \\ &\qquad\qquad \cup \\ &\quad L_f \cdot \{11\} \cdot \{00,01\}^* \\ &\qquad\qquad \cup \\ &\quad \overline{\{0,1\}^* \cdot \{11\} \cdot \{00,01\}^*}\end{aligned}$$

where $h : \{0,1\}^* \mapsto \{0,1\}^*$ is the homomorphism defined by $h(0) = 00$ and $h(1) = 01$.

If $\mathcal{L}(G) = \{0,1\}^*$, it is clear that $\mathcal{L}(H) = \{0,1\}^*$ which is an EXREG language. Otherwise, we want to show the language $\mathcal{L}(H)$ is not an EXREG language. Assume $\mathcal{L}(H)$ is an EXREG language. Since $\mathcal{L}(G) \neq \{0,1\}^*$, there exists a string $w \notin \mathcal{L}(G)$ such that $\mathcal{L}(H) \cap \{0,1\}^* \cdot \{11h(w)\} = L_f \cdot \{11h(w)\}$. Since EXREG languages are closed under intersection with regular sets, $L_f \cdot \{11h(w)\}$ is an EXREG language. This is a contradiction. So $\mathcal{L}(H)$ is not an EXREG language. Since L_f is a fixed language, the construction of H only depends on G in polynomial time in $|G|$. □

Generally, a pumping lemma states that for a language to be in a class of language, any sufficiently long string in the language must contain a section that can be removed or repeated any number of times with the resulting string remaining in the language. So, if we can use a pumping lemma to prove that a language L_f is not in a class of languages, the same proof works for showing that the language $L_f \cdot \{w\}$ or $\{w\} \cdot L_f$, for all $w \in \{0,1\}^*$, is not in that class of languages. Hence, to satisfy condition 2 of Theorem 3.1, the existence of a pumping lemma for EXREGs is sufficient. We use two examples to illustrate the broad applicability of Theorem 3.1 and its ease of use. In [15], the universality problem is shown to be productive for SRE and N 1-rbd 1-CMs, and the reductions in the proofs are highly efficient. Hence, $\overline{\mathbf{K}} \leqslant_{ptime} \{d \mid d \in \mathbf{N11CM}, \mathcal{L}(d) = \{0,1\}^*\}$, and $\overline{\mathbf{K}} \leqslant_{ptime} \{d \mid d \in \mathbf{SRE\{0,1\}}, \mathcal{L}(d) = \{0,1\}^*\}$. With the following pumping lemma for EXREGs, we can get two important productiveness results.

Lemma 3.2 [1] *Let α be an extended regular expression. Then there is a constant $N > 0$ such that if $w \in \mathcal{L}(\alpha)$ and $|w| > N$, then there is a decomposition $w = x_0 y x_1 y \cdots y x_m$ for some $m \geq 1$, such that*

1. $|x_0 y| < N$,
2. $|y| \geq 1$, *and*
3. $x_0 y^i x_1 y^i \cdots y^i x_m \in \mathcal{L}(\alpha)$ *for all* $i > 0$.

Using Lemma 3.2, we get the following results.

Lemma 3.3 $\forall w \in \{0, 1\}^*$, *the language $L_w = \{0^n 1^n \mid n > 0\} \cdot \{w\}$ is not an EXREG language.*

Proof Assume L_w is an EXREG language. Consider the string $t = 0^N 1^N w$ for some constant $N > 0$. It is easy to see $t \in L_w$. Hence, $t = x_0 y x_1 y \cdots y x_m$ where $|x_0 y| < N$ and $|y| \geq 1$. Hence, $y \in \{0\}^+$ which implies $x_0 y^2 x_1 y^2 \cdots y^2 x_m \notin L_w$, which is a contradiction. □

Lemma 3.4 $\forall w \in \{0, 1\}^*$, *the language $L = \{w_1 \# w_2 \# \cdots w_n \# \mid w_1, ..., w_n \in \{0, 1\}^*, |w_1| = |w_2| = ... = |w_n|, n \geq 0\} \cdot \{w\}$ is not an EXREG language, but can be expressed by a synchronized regular expression.*

Proof The poof can be seen in [4]. □

Corollary 1 $\overline{\boldsymbol{K}} \leqslant_{ptime} \{d \mid d \in \mathbf{N11CM}, \mathcal{L}(d)$ *is an EXREG language}. Hence, the predicate "is an EXREG language" is productive (not recursively enumerable) for N 1-rbd 1-CMs, linear context-free grammars, and context-free grammars.*

Proof Let $L_f = \{0^n 1^n \mid n > 0\}$. From Lemma 3.3, we know that $\forall w \in \{0, 1\}^*$, the language $L_f \cdot \{w\}$ is not an EXREG language. This satisfies condition 2 of Theorem 3.1. □

Corollary 2 $\overline{\boldsymbol{K}} \leqslant_{ptime} \{d \mid d \in \mathbf{SRE\{0, 1\}}, \mathcal{L}(d)$ *is an EXREG language}. Hence, the predicate "is an EXREG language" is productive (not recursively enumerable) for SRE.*

Proof Let $L_f = \{h(w_1) \cdot 11 \cdot h(w_2) \cdot 11 \cdots h(w_n) \cdot 11 \mid w_1, ..., w_n \in \{0, 1\}^*, |w_1| = |w_2| = ... = |w_n|, n \geq 0\}$ where h is a 1-1 homomorphism defined by $h(0) = 00$ and $h(1) = 01$. Here, the string 11 is treated as the special maker # of L defined in Lemma 3.4. So the proof of Lemma 3.4 can also prove that $\forall w \in \{0, 1\}^*$, the language $L_f \cdot \{w\}$ is not an EXREG language. This satisfies condition 2 of Theorem 3.1. □

Besides the predicate "$= \{0, 1\}^*$", this proof technique can also be applied to reductions of other sources. For example, one theorem in [18] states that the predicate "$= \{0, 1, \lambda\}^{2^{cn}}$" is Co-NEXPTIME-hard for context-free grammars generating finite languages. Here we state that theorem with a slight modification. The proof is the same as in [18]. Let CFG_{fin} be the set of context-free grammars over terminal alphabet $\{0, 1\}$ generating finite languages.

Theorem 3.2 [18] *There exists a constant* $c > 0$ *such that* $\mathbf{Co-NEXPTIME} \leqslant_{ptime} \{d \mid d \in CFG_{fin}, \mathcal{L}(d) = \{0, 1, \lambda\}^{2^{cn}}$ *where* $n = |d|\}$.

The following theorem shows that for context-free grammars generating bounded languages, the predicate "is an EXREG language" is Co-NEXPTIME-hard. Moreover we show that even for an easily recognizable subset D of $\mathbf{CFG}(\{\mathbf{0}, \mathbf{1}\})$ whose elements only generate bounded languages, the predicate "is an EXREG language" is already Co-NEXPTIME-hard. This means that the problem is hard independent of the complexity of testing whether a context-free grammar generates a bounded language. Results of this type occur throughout this paper and have many applications, especially for promise problems. We first give the definition of a bounded language.

Definition 11 A language L is *bounded* if $L \subseteq \{w_1\}^* \cdot \{w_2\}^* \cdots \{w_m\}^*$ for some strings $w_1, w_2, ..., w_m \in \{0, 1\}^*$, $m \geq 1$.

Theorem 3.3 *There exists a subset D of* ***CFG({0,1})*** *such that*

1. $D \in \mathbf{P}$;
2. $\forall d \in D$, $\mathcal{L}(d)$ *is bounded; and*
3. ***Co-NEXPTIME*** $\leqslant_{ptime} \{d \mid d \in D$, $\mathcal{L}(d)$ *is an EXREG language*$\}$.

Proof of 3: $L_f = \{0^n 1^n \mid n \geq 0\}$ is not an EXREG language and it is bounded by $\{0\}^* \cdot \{1\}^*$. For any $g \in CFG_{fin}$, let c and n be the same as defined in Theorem 3.2. We can efficiently construct a context-free grammar H such that

$$\mathcal{L}(H) = \{0\}^* \cdot \{1\}^* \cdot \{11\} \cdot h(\mathcal{L}(g))$$
$$\cup$$
$$L_f \cdot \{11\} \cdot h(\{0, 1, \lambda\}^{2^{cn}})$$

where $h : \{0, 1\}^* \mapsto \{0, 1\}^*$ is the homomorphism defined by $h(0) = 00$ and $h(1) = 01$. If $\mathcal{L}(g) = \{0, 1, \lambda\}^{2^{cn}}$, $\mathcal{L}(H) = \{0\}^* \cdot \{1\}^* \cdot \{11\} \cdot h(\mathcal{L}(g))$ which is regular. So $\mathcal{L}(H)$ is an EXREG language. Otherwise, assume that $\mathcal{L}(H)$ is an EXREG language. There exists a string $w \notin \mathcal{L}(g)$ such that $\mathcal{L}(H) \cap \{0, 1\}^* \cdot \{11h(w)\} = L_f \cdot \{11h(w)\}$. Hence, $L_f \cdot \{11h(w)\}$ is an EXREG since EXREG languages are closed under intersection with regular sets. From Lemma 3.3, this is a contradiction. So $\mathcal{L}(H)$ is not an EXREG language. □

Proof of 1, 2: Let H be constructed in a certain way so that H has a special format. For example, H must contain two non-terminals such that the first non-terminal generates $\{0\}^* \cdot \{1\}^* \cdot \{11\} \cdot h(\mathcal{L}(g))$, and the second non-terminal generates $L_f \cdot \{11\} \cdot h(\{0, 1, \lambda\}^{2^{cn}})$. Let D be the set of H. It is easy to see $\forall d \in D$, $\mathcal{L}(d)$ is bounded. Since L_f is fixed and H is constructed with a special format, we can determine g from H in polynomial time in $|H|$. $CFG_{fin} \in \mathbf{P} \Rightarrow D \in \mathbf{P}$. □

We can also apply the proof technique to reductions from the emptiness problem. In [19], the emptiness problem for EDT0L systems is shown to be PSPACE-complete. We modify the proof of this PSPACE-completeness result and get a stronger theorem. It shows that for a polynomial time recognizable subset D of **EDT0L** whose elements only generate $\emptyset$ or singleton languages (i.e., $\{w\}$ where $w \in \{0, 1\}^*$), the emptiness problem is already PSPACE-hard.

Theorem 3.4 *There exists a subset D of **EDT0L** such that*

1. $D \in \mathbf{P}$;
2. $\forall d \in D$, $|\mathcal{L}(d)| \leq 1$; *and*
3. ***PSPACE*** $\leqslant_{ptime} \{d \mid d \in D, \mathcal{L}(d) = \emptyset\}$.

Proof In [20] Theorem 3.4, Xie et al. showed that for a polynomial time recognizable subset R of $(\cup, \cdot.*)$-regular expressions where each element in R only generates $\{0, 1\}^*$ or $\{0, 1\}^* - \{w\}$ where $w \in \{0, 1\}^*$, the predicate "$= \{0, 1\}^*$" is PSPACE-hard. In this proof, let R be the same as defined in [20]. It is well-known that $(\cup, \cdot.*)$-regular expressions can be transformed into NFAs in polynomial time. Let N be the set of NFAs transformed from R. For any NFA $M = (Q, \{0, 1\}, \sigma, q_0, F) \in N$ where

1. Q is the finite set of states;
2. $F \subseteq Q$ is the set of accepting states;
3. $\{0, 1\}$ is the input alphabet;
4. $\sigma : (Q \times \{0, 1, \lambda\}) \mapsto 2^Q$ is the transition function; and
5. q_0 is the initial state.

Here 2^Q denotes the power set of Q. We give every state in Q a distinct name $q_0, q_1, q_2, ..., q_{|Q|-1}$ and define the total order $<$ by $q_0 < q_1 < q_2 < ... < q_{|Q|-1}$. From M, we can efficiently construct an EDT0L system $G = (Q - F, F, P, q_0)$ where

1. $Q - F$ is the terminal alphabet;
2. F is the nonterminal alphabet;
3. q_0 is the axiom; and
4. $P = \{P_0, P_1\}$ where P_0 and P_1 are finite substitutions and for each $x \in \{0, 1\}$, $\forall q \in Q$, if $\sigma(q, x) = \{q_{i_1}, q_{i_2}, ...q_{i_k}\}$ where $q_{i_1} < q_{i_2} < ... < q_{i_k}$, then $q_{i_1}q_{i_2}q_{i_3}...q_{i_k} \in P_x(q)$.

We define a function $\hat{\sigma} : (Q \times \{0, 1\}^*) \mapsto 2^Q$ such that

1. $\hat{\sigma}(q, \lambda) = \{q\}$;
2. $\hat{\sigma}(q, wa) = \bigcup_{q' \in \hat{\sigma}(q,w)} \sigma(q', a)$.

If $\mathcal{L}(M) = \{0, 1\}^*$, then for any $w \in \{0, 1\}^*$, $\hat{\sigma}(q_0, w)$ contains a state in F. $\Rightarrow$ $P(q_0) \cap (Q - F)^* = \emptyset$. $\Rightarrow$ $\mathcal{L}(G) = \emptyset$. If $\mathcal{L}(M) = \{0, 1\}^* - \{t\}$, then t is the only string in $\{0, 1\}^*$ such that $\hat{\sigma}(q_0, t)$ contains no state in F. Let $t = t_1t_2t_3...t_k$ where t_i $(1 \leq i \leq k) \in \{0, 1\}$. Then $\mathcal{L}(G) = P_{t_k}(P_{t_{k-1}}...P_{t_2}(P_{t_1}(q_0)))$. Since P_0 and P_1 are morphisms, clearly $|\mathcal{L}(G)| = 1$. Let D be the set of G we construct. Then $R \in \mathbf{P} \Rightarrow N \in \mathbf{P} \Rightarrow D \in \mathbf{P}$. ◻

With Theorem 3.4, we can study the predicate "is an EXREG language" for EDT0L systems and get the following theorem. It shows that even for an easily recognizable subset E of **EDT0L** whose elements only generate bounded languages, the predicate "is an EXREG language" is already PSPACE-hard. This means that the problem is hard independent of the complexity of testing whether an EDT0L system generates a bounded language. Comparing with Theorems 3.1 and 3.3, the proof of Theorem 3.5 is easier and requires less. It suggests that if a class of languages $\mathcal{C}$ is effectively closed under concatenation and there exists a language $L_f \in \mathcal{C}$ such that $\forall w \in \{0, 1\}^*$, the language $L_f \cdot \{w\}$ or $\{w\} \cdot L_f$ is not an EXREG language, then the predicate "is an EXREG language" is as hard as "$= \emptyset$" for $\mathcal{C}$. Note that the closure property here does not need to be efficient, since the construction in Theorem 3.5 only requires concatenation with a fixed language.

Theorem 3.5 *There exists a subset E of **EDT0L** such that*

1. $E \in \mathbf{P}$;
2. $\forall d \in E$, $\mathcal{L}(d)$ *is bounded; and*
3. ***PSPACE*** $\leqslant_{ptime} \{d \mid d \in E, \mathcal{L}(d)$ *is an EXREG language*$\}$.

Proof Consider the set D mentioned in Theorem 3.4. Recall that for any $G \in D$, $|\mathcal{L}(G)| \leq 1$. For any $G \in D$, we can construct an EDT0L system H such that

$$\mathcal{L}(H) = \{0^n 1^n \mid n > 0\} \cdot \mathcal{L}(G)$$

If $\mathcal{L}(G) = \emptyset$, then $\mathcal{L}(H) = \emptyset$. So $\mathcal{L}(H)$ is an EXREG language. Otherwise, $\mathcal{L}(G) = \{w\}$. Then $\mathcal{L}(H) = \{0^n 1^n \mid n > 0\} \cdot \{w\}$. According to Lemma 3.3, $\mathcal{L}(H)$ is not an EXREG language. Let E be the set of H. Clearly, $\forall d \in E$, $\mathcal{L}(d)$ is bounded. From H we can determine the system G efficiently. $D \in \mathbf{P} \Rightarrow E \in \mathbf{P}$. $\square$

4 Conclusion

In the theory of formal languages, developing pumping lemmas for classes of languages remains an important research topic. A pumping lemma for a class of languages $\mathcal{C}$ is often used to show particular languages are not in $\mathcal{C}$. In contrast, we use EXREGs as an example to show that having a pumping lemma could be "harmful" and lead to productiveness and complexity results. In this paper, we show that the predicate "is an EXREG language" is productive, hence not recursively enumerable, for SRE, N 1-rbd 1-CMs, linear context-free grammars, and context-free grammars. We also show that the predicate "is an EXREG language" is Co-NEXPTIME-hard for a polynomial time recognizable set of context-free grammars only generating bounded languages, and is PSPACE-hard for a polynomial time recognizable set of EDT0L systems generating bounded languages. To obtain these results, a pumping lemma for EXREGs is needed to show that there exists a language L_f such that for any single string w, $\{w\} \cdot L_f$ or $L_f \cdot \{w\}$ is not an EXREG language. The proof technique used in this paper requires very little and can be applied to reductions of many sources (for example, "$= \{0, 1\}^*$", "$= \emptyset$", and "$= \{0, 1, \lambda\}^{2^{cn}}$"). So we believe it has the potential to be used for many classes of languages.

Acknowledgements We thank Dr. Paliath Narendran and the anonymous reviewers for their time and advice. Their valuable input makes this paper more complete.

Funding Not applicable

Declarations

Ethics approval Not applicable

Conflicts of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Foundations Comput. Sci. **14**(6), 1007–1018 (2003). https://doi.org/10.1142/S012905410300214X
2. 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
3. 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
4. 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, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_24
5. Lindenmayer, A.: Mathematical models for cellular interactions in development i. filaments with one-sided inputs. J. Theoretical Biology. **18**(3), 280–299 (1968)
6. Kari, L., Rozenberg, G., Salomaa, A.: In: Rozenberg, G., Salomaa, A. (eds.) L Systems, pp. 253–328. Springer, Berlin, Heidelberg (1997)
7. Levine, A.: Edt0l solutions to equations in group extensions. J. Algebra. **619**, 860–899 (2023). https://doi.org/10.1016/j.jalgebra.2022.11.031
8. Duncan, A., Evetts, A., Holt, D.F., Rees, S.: Using edt0l systems to solve some equations in the solvable baumslag-solitar groups. J. Algebra. **630**, 434–456 (2023). https://doi.org/10.1016/j.jalgebra.2023.04.020
9. Ghorani, M., Garhwal, S., Moghari, S.: Lattice-valued tree pushdown automata: pumping lemma and closure properties. Int. J. Approximate Reasoning. **142**, 301–323 (2022). https://doi.org/10.1016/j.ijar.2021.12.002
10. Lucero, J.C.: Pumping lemmas for classes of languages generated by folding systems. Natural Comput. **20**, 321–327 (2019). https://doi.org/10.1007/s11047-019-09771-5
11. Chattopadhyay, A., Mazowiecki, F., Muscholl, A., Riveros, C.: Pumping lemmas for weighted automata. Logical Methods Comput. Sci. **17**(3) (2021). https://doi.org/10.46298/lmcs-17(3:7)2021
12. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading, MA (1979)
13. Rogers, H., Jr.: Theory of recursive functions and effective computability. MIT Press, Cambridge, MA, USA (1987)
14. Soare, R.I.: Recursively enumerable sets and degrees. Springer, Berlin, Heidelberg (1987)

15. Xie, J., Hunt, H.B., III.: On the undecidability and descriptional complexity of synchronized regular expressions. Acta Informatica. **60**(3), 257–278 (2023). https://doi.org/10.1007/s00236-023-00439-3
16. 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
17. Câmpeanu, C., Santean, N.: On the intersection of regex languages with regular languages. Theoretical Comput. Sci. **410**(24), 2336–2344 (2009). https://doi.org/10.1016/j.tcs.2009.02.022
18. Hunt, H.B., III., Rosenkrantz, D.J., Szymanski, T.G.: On the equivalence, containment, and covering problems for the regular and context-free languages. J. Comput. Syst. Sci. **12**(2), 222–268 (1976). https://doi.org/10.1016/S0022-0000(76)80038-4
19. Jones, N.D., Skyum, S.: Complexity of some problems concerning L systems. Math. Syst. Theory. **13**(1), 29–43 (1979). https://doi.org/10.1007/BF01744286
20. Xie, J., Hunt, H.B., III., Stearns, R. E.: On the computational and descriptional complexity of multi-pattern languages. Available at SSRN (2023). https://doi.org/10.2139/ssrn.4493700

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.