Available online at www.sciencedirect.com

ScienceDirect

Information and Computation 00 (2015) 1–18

Inform. Comput.

Deciding Determinism of Unary Languages☆

Ping Lu[a,b,c], Feifei Peng[a,b,c,*], Haiming Chen[a,**], Lixiao Zheng[d]

[a] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*
[b] *Graduate University of Chinese Academy of Sciences*
[c] *University of Chinese Academy of Sciences*
[d] *College of Computer Science and Technology, Huaqiao University, Xiamen Fujian 361021,China*

Abstract

In this paper, we investigate the complexity of deciding determinism of unary languages. First, we give a method to derive a set of arithmetic progressions from a regular expression E over a unary alphabet, and establish relations between numbers represented by these arithmetic progressions and words in $\mathcal{L}(E)$. Next, we define a problem relating to arithmetic progressions and investigate the complexity of this problem. Then by a reduction from this problem we show that deciding determinism of unary languages is **coNP**-complete. Finally, we extend our derivation method to expressions with counting, and prove that deciding whether an expression over a unary alphabet with counting defines a deterministic language is in $\mathbf{\Pi_2^p}$. We also establish a tight upper bound for the size of the minimal DFA for expressions with counting.

Keywords: Regular expressions, Regular expressions with counting, Deterministic languages, Minimal DFA, **coNP**-complete, $\mathbf{\Pi_2^p}$

1. Introduction

XML (Extensible Markup Language) has important applications in data exchange [1], database [2], etc. XML schema languages, e.g., DTD and XML Schema, are used to specify the constraints which XML documents should obey [3]. However, designing a correct schema is not an easy job [4, 5]. One difficulty is the Unique Particle Attribution (UPA) constraint [6], which requires that content models should be deterministic [7, 8]. Intuitively, determinism means that a symbol in the input word should be matched to a unique position in the regular expression without looking ahead in the word [6, 9]. For example, $A \to a^*a$ is a simple example of a DTD. This is not a correct DTD, because the content model a^*a is not deterministic. Consider the word a. Without knowing the length of the word, we do not know that the only symbol a in the word should match the first a or the second one in a^*a.

Deterministic expression is defined in a semantic way, without a known simple syntax definition [8]. It is not easy for users to understand such kind of expressions. Studying properties of deterministic expressions can help reduce

☆Parts of this work have been presented as conference abstracts appeared in DLT 2013 under the title "Deciding Determinism of Unary Languages Is coNP-Complete". Work supported by the National Natural Science Foundation of China under Grants 61472405 and 61070038.

*Part of this work was done while Feifei Peng was a student in China Agricultural University.

**Corresponding author.
E-mail addresses: {luping,chm}@ios.ac.cn

this difficulty. Lots of work [8, 9, 10, 11, 12, 13, 14, 15] studied properties of deterministic expressions and gave methods to help users write deterministic expressions. Meanwhile, studying properties of deterministic languages is equally important. For example, when the user writes a nondeterministic expression E and we know that $L(E)$ is deterministic, we can automatically generate a deterministic expression describing $L(E)$ for the user. However only little progress has been made about determinism of languages.

For standard regular expressions, Brüggemann-Klein and Wood [9] showed that the problem, whether a regular language defined by a standard regular expression can be described by a standard deterministic expression, is decidable. Bex et al. [8], Czerwiński et al. [16], and P. Lu et al. [17] proved that this problem is **PSPACE**-complete. The problem becomes much harder when we consider expressions with counting. Czerwiński et al. [16] also proved that deciding whether a regular language defined by a regular expression with counting can be described by a standard deterministic expression is **EXPSPACE**-complete [16]. Recently Latte et al. [18] had shown that whether a regular language defined by a standard regular expression can be described by a deterministic expression with counting is in 2-**EXPSPACE**. And an **NL** lower bound was given there [18, 17]. In this paper, we try to give a **coNP** lower bound for this problem.

In [19], Gelade et al. showed that for unary languages, deterministic expressions with counting are expressively equivalent to standard deterministic expressions. Hence considering determinism of regular languages described by standard expressions over a unary alphabet can give a lower bound for the problem, whether a regular language can be described by a deterministic expression with counting. Moreover, in the lower bound proofs of [8] and [16], the alphabet size of constructed expressions is at least 4. So it is possible that the complexity of the problem, whether a regular language defined by a standard regular expression over a unary alphabet can be described by a standard deterministic expression, is lower than **PSPACE**. This is our starting point. In the following, unless explicitly stated otherwise, all regular expressions are expressions over the alphabet $\{a\}$.

Our contributions are listed as follows:

(1) We show that deciding whether a standard expression denotes a deterministic language is **coNP**-complete. Then we conclude that deciding whether a language can be defined by a deterministic expression with counting is **coNP**-hard.

(2) For any expression E with counting, we show that there is a DFA with less than $2^{O(|E|)}$ states accepting $\mathcal{L}(E)$. It has been shown that there exists an expression E with counting such that every DFA accepting this language has at least exponential number of states [20, 21]. So our upper bound is tight. For the case $|\Sigma| = 2$, there is an expression E such that the minimal DFA accepting $\mathcal{L}(E)$ has $\Omega(2^{2^{|E|}})$ states [21].

(3) Using the result in (2), we devise a non-deterministic algorithm to check determinism of languages defined by expressions with counting, and show that the problem, whether an expression with counting denotes a deterministic language, is in $\mathbf{\Pi_2^p}$.

The rest of the paper is organized as follows. Section 2 gives some basic definitions and some facts from the number theory, which we will use later. We associate a set of arithmetic progressions with a given regular expression in Section 3. Section 4 shows the complexity of deciding determinism of unary languages. Section 5 deals with expressions with counting. Section 6 gives the conclusion and the future work.

2. Preliminaries

Let $\Sigma = \{a\}$ be an alphabet of symbols. A standard regular expression over Σ is recursively defined as follows: $\emptyset$, ε and a are regular expressions; for any two regular expressions E_1 and E_2, the union $E_1 + E_2$, the concatenation E_1E_2 and the star E_1^* are regular expressions. For a regular expression E, we denote $\mathcal{L}(E)$ as the language specified by E and $|E|$ as the size of E, which is the sum of the number of symbol occurrences in E and the number of used operators.

Expressions with counting, denoted by R(#), extend standard expressions by using counting operator: $E^{[m,n]}$ or $E^{[m,\infty]}$, where ∞ stands for infinity. Since $E^* = E^{[0,\infty]}$, we do not consider the star operator in regular expressions in R(#). The size of an expression E in R(#), denoted by $|E|$, is the sum of the number of symbol occurrences, the number of used operators, and the lengths of the binary encodings of all counting numbers [19].

To define deterministic regular expressions, we need the following notations. We mark each symbol a in E with a different integer i such that each marked symbol a_i occurs only once in the marked expression. For example, $a_1^*a_2$ is a marking of a^*a. The marking of E is denoted by $\overline{E}$. We use $E^\natural$ to denote the result of dropping subscripts from the marked symbols. These notations are extended for words and sets of symbols in an obvious way.

Deterministic regular expressions are defined as follows.

Definition 1 ([9]). *An expression E is deterministic, if and only if, for all words $uxv, uyw \in \mathcal{L}(\overline{E})$ where $|x| = |y| = 1$, if $x \neq y$ then $x^\natural \neq y^\natural$. A regular language is deterministic if it is denoted by some deterministic expression.*

For example, a^*a is not deterministic, since $a_2, a_1a_2 \in \mathcal{L}(a_1^*a_2)$. Deterministic regular expressions denote a proper subclass of regular languages [9].

A nondeterministic finite automaton (NFA) [22] is a 5-tuple $\mathcal{N} = (Q, \{a\}, \delta, q_0, F)$, where Q is a finite set of states, a is the input symbol, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final states, $\delta : Q \times \{a\} \to 2^Q$ is the transition function. The size of an NFA $\mathcal{N}$, denoted as $|\mathcal{N}|$, is defined as the number of states of $\mathcal{N}$. An NFA $\mathcal{N} = (Q, \{a\}, \delta, q_0, F)$ is in Chrobak normal form [23] if the following conditions hold: (1) $Q = \{q_0, q_1, \ldots, q_m\} \cup \{q_{1,0}, q_{1,1}, \ldots, q_{1,i_1}\} \cup \{q_{2,0}, q_{2,1}, \ldots, q_{2,i_2}\} \cup \ldots \cup \{q_{n,0}, q_{n,1}, \ldots, q_{n,i_n}\}$; (2) $F \subseteq Q$; (3) $\delta(q_0, a) = \{q_1\}, \ldots, \delta(q_{m-1}, a) = \{q_m\}, \delta(q_m, a) = \{q_{1,0}, \ldots, q_{n,0}\}$ and $\delta(q_{j,k}, a) = \{q_{j,((k+1) \bmod (i_j+1))}\}$ $(0 \leq j \leq n, 0 \leq k \leq i_j)$. Chrobak [23] had shown that every NFA $\mathcal{N}$ over a unary alphabet can be changed into an equivalent NFA in Chrobak normal form with $O(|\mathcal{N}|^2)$ states.

A deterministic finite automaton (DFA) is an NFA where the transition function δ is defined as $Q \times \{a\} \to Q$. The minimal DFA M is the DFA such that $|M| \leq |D|$ for any DFA D with $\mathcal{L}(D) = \mathcal{L}(M)$. For any regular language, the minimal DFA is unique modulo state renaming [22].

A context-free grammar $G = (V, \{a\}, P, S)$ is a 4-tuple such that [22]: (1) V is a set of variables; (2) $S \in V$ is the start symbol; (3) $P \subseteq V \times (V \cup \{a\})^*$ is a finite set of rules. And a context-free grammar G is in Chomsky normal form [24] if all rules have one of the following two forms: (1) $X \to a$; (2) $X \to BC$, where $B, C \in V$. It has been known that regular expressions over a unary alphabet have the same expressive power as unary context-free grammars in Chomsky normal form [25].

The following notations are basic mathematical operators [26]: $\lfloor x \rfloor = \max\{ n \mid n \leq x, n \in \mathbb{Z}^1\}$; $x \bmod y = x - y\lfloor \frac{x}{y} \rfloor$, for $y \neq 0$; $x \equiv y \pmod p \Leftrightarrow x \bmod p = y \bmod p$; $m|n \Leftrightarrow m > 0$ and $n = mx$ for some integer x; $\gcd(x_1, x_2, \ldots, x_n) = \max\{k | (k|x_1) \wedge (k|x_2) \wedge \ldots (k|x_n)\}$; $\mathrm{lcm}(x_1, x_2, \ldots, x_n) = \min\{k | k > 0 \wedge (x_1|k) \wedge (x_2|k) \wedge \ldots (x_n|k)\}$. Notice that $\gcd(0, 0, \ldots, 0)$ is undefined and $\mathrm{lcm}(x_1, x_2, \ldots, x_n)$ is also undefined when at least one of the parameters is 0. In this paper, we denote $\gcd(0, 0, \ldots, 0) = 0$ and $\mathrm{lcm}(x_1, x_2, \ldots, x_n) = 0$ when one of the parameters is 0. Moreover, if $S = \{s_1, s_2, \ldots, s_n\}$, then we denote $\gcd(S) = \gcd(s_1, s_2, \ldots, s_n)$.

The following fact is important to our proofs.

Lemma 1 ([27, 28]). *Given n $(n \geq 2)$ integers $a_1 > 0, a_2 > 0, \ldots, a_n > 0$, each number of the form $a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n$, with $x_1 \geq 0$, $x_2 \geq 0$, …, and $x_n \geq 0$, is a multiple of $\gcd(a_1, a_2, \ldots, a_n)$. Furthermore, all multiples of $\gcd(a_1, a_2, \ldots, a_n)$ larger than $(\max(a_1, a_2, \ldots, a_n))^2$ can be represented as $a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n$, with $x_1 \geq 0$, $x_2 \geq 0$, …, and $x_n \geq 0$.*

An arithmetic progression, denoted as $\langle l, s \rangle$, is an infinite set of integers $\{s, s + l, s + 2 \cdot l, s + 3 \cdot l, \ldots\}$ [26]. A *covering system* **CS** is a set of arithmetic progressions $\{\langle l_1, s_1 \rangle, \langle l_2, s_2 \rangle, \ldots, \langle l_n, s_n \rangle\}$, with $0 \leq s_i < l_i$ $(1 \leq i \leq n)$, such that every integer $x \geq 0$ satisfies $x \equiv s_i \pmod{l_i}$ for some i $(1 \leq i \leq n)$ [29]. The size of a cover system is defined as $\sum_{i=1}^{n}(l_i + s_i)$. For example, the set of arithmetic progressions $\{\langle 2, 0 \rangle, \langle 4, 1 \rangle, \langle 4, 3 \rangle\}$ forms a covering system, and its size is 14. It is easy to verify that every integer $i \geq 0$ satisfies one of the following conditions: $i \equiv 0 \pmod 2$; $i \equiv 1 \pmod 4$; $i \equiv 3 \pmod 4$.

Covering systems were introduced by Paul Erdős [30, 29]. This is an interesting topic in mathematics and there are many unsolved problems about covering systems [31]. Here, we are only concerned with the problem whether a set of arithmetic progressions forms a covering system. This problem has been shown to be **coNP**-complete [32, 33].

[1]Here $\mathbb{Z}$ denotes the set of all integers. That is $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. In the following, we also denote by $\mathbb{N}$ the set of natural numbers. That is $\mathbb{N} = \{0, 1, 2, \ldots\}$.

In this paper, we primarily discuss unary languages. For a regular language $\mathcal{L}$ over the alphabet $\{a\}$, there is a correspondence between words in $\mathcal{L}$ and their lengths. For convenience when we say the word n, we mean the word a^n.

3. The arithmetic progressions of unary languages

In this section, we handle the following problem: Given a standard expression E[2], how to construct a set $\mathcal{P} = \{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$ of arithmetic progressions such that there exists a number $M \in \mathbb{N}$ satisfying $|\mathcal{L}(E) \setminus \mathcal{P}| + |\mathcal{P} \setminus \mathcal{L}(E)| \leq M$. In next section, we will define the equal difference covering system (**EDCS**) over arithmetic progressions, show that $\mathcal{L}(E)$ is deterministic, if and only if, $\mathcal{P}$ forms an **EDCS**, and obtain the complexity of deciding determinism of $\mathcal{L}(E)$.

$\mathcal{P}$ is computed by the following function:

Definition 2. *The function $\mathcal{S}(E)$ is defined as*

$$\begin{aligned}
&\mathcal{S}(\varepsilon) = \{\langle 0, 0\rangle\}\\
&\mathcal{S}(a) = \{\langle 0, 1\rangle\}, \quad a \in \Sigma\\
&\mathcal{S}(E_1 + E_2) = \mathcal{S}(E_1) \cup \mathcal{S}(E_2)\\
&\mathcal{S}(E_1E_2) = \{\langle \gcd(l_i, l_j), s_i + s_j\rangle | \langle l_i, s_i\rangle \in S(E_1) \wedge \langle l_j, s_j\rangle \in S(E_2)\}\\
&\mathcal{S}(E_1^*) = \{\langle l, 0\rangle | S(E_1) = \{\langle l_1, s_1\rangle, \ldots, \langle l_n, s_n\rangle\} \wedge l = \gcd(l_1, \ldots, l_n, s_1, \ldots, s_n)\}.
\end{aligned}$$

The intuition behind the construction of $\mathcal{S}(E)$ is Lemma 1. The cases ε, a, and $E_1 + E_2$ are obvious. For the case E_1E_2, let $\langle l_1, s_1\rangle \in \mathcal{S}(E_1)$ and $\langle l_2, s_2\rangle \in \mathcal{S}(E_2)$. Then all numbers $k_1 \cdot l_1 + s_1 + k_2 \cdot l_2 + s_2$ with $k_1, k_2 \in \mathbb{N}$ should be in $\mathcal{S}(E)$. By Lemma 1, these numbers can be written as $k \cdot \gcd(l_1, l_2) + s_1 + s_2$, where $k \in \mathbb{N}$. The case E_1^* is similar.

Example 1. *Let $E = (aaa + aa)^* + (aaa)^*((aa)^*aaa + (aaa)^*aa)$. The process of computing $\mathcal{S}(E)$ is shown in Table 1. E_1 in the table stands for subexpressions of E. At last, $\mathcal{S}(E) = \{\langle 1, 0\rangle, \langle 1, 3\rangle, \langle 3, 2\rangle\}$. It is easy to see that $\mathcal{S}(E)$ contains all natural numbers. However, $a \notin \mathcal{L}(E)$ and a is the only word, which is not in $\mathcal{L}(E)$.*

E_1	$\mathcal{S}(E_1)$	E_1	$\mathcal{S}(E_1)$
a	$\langle 0, 1\rangle$	$(aa + aaa)^*$	$\langle 1, 0\rangle$
aa	$\langle 0, 2\rangle$	$(aa)^*aaa$	$\langle 2, 3\rangle$
aaa	$\langle 0, 3\rangle$	$(aaa)^*aa$	$\langle 3, 2\rangle$
$aa + aaa$	$\langle 0, 2\rangle\langle 0, 3\rangle$	$(aa)^*aaa + (aaa)^*aa$	$\langle 2, 3\rangle\langle 3, 2\rangle$
$(aa)^*$	$\langle 2, 0\rangle$	$(aaa)^*((aa)^*aaa + (aaa)^*aa)$	$\langle 1, 3\rangle\langle 3, 2\rangle$
$(aaa)^*$	$\langle 3, 0\rangle$	$(aaa + aa)^* + (aaa)^*((aa)^*aaa + (aaa)^*aa)$	$\langle 1, 0\rangle\langle 1, 3\rangle\langle 3, 2\rangle$

Table 1: The process of computing $\mathcal{S}(E)$

In Example 1, we can see that words in $\mathcal{L}(E)$ are all contained in $\mathcal{S}(E)$, and there are only finite many numbers in $\mathcal{S}(E)$, not contained in $\mathcal{L}(E)$. In the following, we will show that this is generally held.

Observation 1. *Let E be an expression. For any $\langle l, s\rangle \in \mathcal{S}(E)$, we have $s \in \mathcal{L}(E)$.*

[2]In this section and next section, we mainly consider standard regular expressions, and when we say a regular expression E, we mean a standard regular expression E. We will handle the corresponding problem for expressions in $R(\#)$ in Section 5.

Proposition 1. *Let E be an expression. For any $w \in \mathcal{L}(E)$, there exist an arithmetic progression $\langle l, s\rangle \in \mathcal{S}(E)$ and an integer $k \in \mathbb{N}$ such that $|w| = k \cdot l + s$.*

Proof. We prove it by induction on the structure of E. The cases $E = \varepsilon$ or a (where $a \in \Sigma$) are obvious.

$\mathbf{E = E_1 + E_2}$: For any $w \in \mathcal{L}(E)$, we know that $w \in \mathcal{L}(E_1)$ or $w \in \mathcal{L}(E_2)$. If $w \in \mathcal{L}(E_1)$, then by the inductive hypothesis there exist $\langle l, s\rangle \in \mathcal{S}(E_1)$ and an integer $k_1 \in \mathbb{N}$ such that $|w| = k_1 \cdot l + s$. From the definition of $\mathcal{S}$, we know that $\mathcal{S}(E) = \mathcal{S}(E_1) \cup \mathcal{S}(E_2)$. Then there exist $\langle l, s\rangle \in \mathcal{S}(E)$ and k_1 such that $|w| = k_1 \cdot l + s$. The case $w \in \mathcal{L}(E_2)$ can be proved in a similar way.

$\mathbf{E = E_1E_2}$: For any $w \in \mathcal{L}(E)$, there are $w_1 \in \mathcal{L}(E_1)$ and $w_2 \in \mathcal{L}(E_2)$ such that $w = w_1w_2$. By the inductive hypothesis there exist $\langle l_1, s_1\rangle \in \mathcal{S}(E_1)$, $\langle l_2, s_2\rangle \in \mathcal{S}(E_2)$, and two integers $k_1, k_2 \in \mathbb{N}$ such that $|w_1| = k_1 \cdot l_1 + s_1$ and $|w_2| = k_2 \cdot l_2 + s_2$. From the definition of $\mathcal{S}$, we know that $\langle \gcd(l_1, l_2), s_1 + s_2\rangle \in \mathcal{S}(E)$. Moreover,

$$\begin{aligned}|w| &= |w_1w_2|\\ &= k_1 \cdot l_1 + s_1 + k_2 \cdot l_2 + s_2\\ &= k_1 \cdot k_1' \cdot \gcd(l_1, l_2) + s_1 + k_2 \cdot k_2' \cdot \gcd(l_1, l_2) + s_2\\ &= (k_1 \cdot k_1' + k_2 \cdot k_2') \cdot \gcd(l_1, l_2) + s_1 + s_2.\end{aligned}$$

Let $k = k_1 \cdot k_1' + k_2 \cdot k_2'$. Then $k \in \mathbb{N}$ and $|w| = k \cdot \gcd(l_1, l_2) + s_1 + s_2$.

$\mathbf{E = E_1^*}$: Suppose $\langle l, 0\rangle \in \mathcal{S}(E)$. If $w = \varepsilon$, since $\langle l, 0\rangle \in \mathcal{S}(E)$, we have $|w| = 0 \cdot l + 0$. Otherwise $w \neq \varepsilon$. Then there are $w_1 \in \mathcal{L}(E_1), \ldots, w_n \in \mathcal{L}(E_1)$ such that $w = w_1 \ldots w_n$. By the inductive hypothesis there exist $\langle l_1, s_1\rangle \in \mathcal{S}(E_1)$, $\ldots$, $\langle l_n, s_n\rangle \in \mathcal{S}(E_1)$, $k_1, \ldots, k_n \in \mathbb{N}$ such that $|w_1| = k_1 \cdot l_1 + s_1, \ \ldots, |w_n| = k_n \cdot l_n + s_n$. From the definition of $\mathcal{S}$, $w \in \mathcal{L}(E)$, and $w \neq \varepsilon$, it is easy to prove that $l \neq 0$. Then for any $\langle l', s'\rangle \in \mathcal{S}(E_1)$, there are $k_{1,1}, k_{1,2}, \ldots, k_{n,1}, k_{n,2} \in \mathbb{N}$ such that $l_1 = k_{1,1} \cdot l$, $s_1 = k_{1,2} \cdot l$, $l_2 = k_{2,1} \cdot l$, $s_2 = k_{2,2} \cdot l, \ldots, l_n = k_{n,1} \cdot l$, and $s_n = k_{n,2} \cdot l$. Hence

$$\begin{aligned}|w| &= |w_1w_2 \ldots w_n|\\ &= k_1 \cdot l_1 + s_1 + k_2 \cdot l_2 + s_2 + \ldots + k_n \cdot l_n + s_n\\ &= k_1 \cdot k_{1,1} \cdot l + k_{1,2} \cdot l + k_2 \cdot k_{2,1} \cdot l + k_{2,2} \cdot l + \ldots + k_n \cdot k_{n,1} \cdot l + k_{n,2} \cdot l\\ &= (k_1 \cdot k_{1,1} + k_{1,2} + k_2 \cdot k_{2,1} + k_{2,2} + \ldots + k_n \cdot k_{n,1} + k_{n,2}) \cdot l.\end{aligned}$$

Let $k = k_1 \cdot k_{1,1} + k_{1,2} + k_2 \cdot k_{2,1} + k_{2,2} + \ldots + k_n \cdot k_{n,1} + k_{n,2}$. Then $k \geq 0$ and $|w| = k \cdot l$. Since $\langle l, 0\rangle \in \mathcal{S}(E)$ and $k \in \mathbb{N}$, the statement holds. □

Proposition 2. *Let E be an expression. For any $\langle l, s\rangle \in \mathcal{S}(E)$ there exists $L \in \mathbb{N}$ such that $L + t \cdot l + s \in \mathcal{L}(E)$ for any $t \in \mathbb{N}$.*

Proof. We prove it by induction on the structure of E. The cases $E = \varepsilon$ or a (where $a \in \Sigma$) are obvious, since $L = 0$ satisfies the conditions.

$\mathbf{E = E_1 + E_2}$: Suppose $\langle l, s\rangle \in \mathcal{S}(E)$. From the definition of $\mathcal{S}$, we have $\langle l, s\rangle \in \mathcal{S}(E_1)$ or $\langle l, s\rangle \in \mathcal{S}(E_2)$. If $\langle l, s\rangle \in \mathcal{S}(E_1)$, then by the inductive hypothesis there exists $L' \in \mathbb{N}$ such that $L' + t \cdot l + s \in \mathcal{L}(E_1)$ for any $t \in \mathbb{N}$. Let $L = L'$. Because $E = E_1 + E_2$, we know that $L + t \cdot l + s \in \mathcal{L}(E)$. Then the conclusion holds. The case $\langle l, s\rangle \in \mathcal{S}(E_2)$ can be proved in a similar way.

$\mathbf{E = E_1E_2}$: Suppose $\langle l, s\rangle \in \mathcal{S}(E)$. From the definition of $\mathcal{S}$, there are $\langle l_1, s_1\rangle \in \mathcal{S}(E_1)$ and $\langle l_2, s_2\rangle \in \mathcal{S}(E_2)$ such that $l = \gcd(l_1, l_2)$ and $s = s_1 + s_2$. By the inductive hypothesis there exist $L_1, L_2 \in \mathbb{N}$ such that $L_1 + t_1 \cdot l_1 + s_1 \in \mathcal{L}(E_1)$ and $L_2 + t_2 \cdot l_2 + s_2 \in \mathcal{L}(E_2)$ for any $t_1, t_2 \in \mathbb{N}$. Let $L = L_1 + L_2 + 2 \cdot (\max(l_1, l_2))^2$. Then for any integer $t \in \mathbb{N}$, there are $k_1, k_2 \in \mathbb{N}$ such that

$$\begin{aligned}&L + t \cdot l + s\\ &= L_1 + L_2 + 2 \cdot (\max(l_1, l_2))^2 + t \cdot \gcd(l_1, l_2) + s_1 + s_2\\ &= L_1 + L_2 + k_1 \cdot l_1 + k_2 \cdot l_2 + s_1 + s_2\\ &= L_1 + k_1 \cdot l_1 + s_1 + L_2 + k_2 \cdot l_2 + s_2\end{aligned}$$

From the inductive hypothesis and $E = E_1E_2$, we have $L + t \cdot l + s \in \mathcal{L}(E)$. Then the conclusion holds.

$\mathbf{E = E_1^*}$: Suppose $\langle l, 0\rangle \in \mathcal{S}(E)$ and $\mathcal{S}(E_1) = \{\langle l_1, s_1\rangle, \ldots, \langle l_n, s_n\rangle\}$. Then $l = \gcd(l_1, l_2, \ldots, l_n, s_1, s_2, \ldots, s_n)$. Because $\mathcal{S}(E_1) = \{\langle l_1, s_1\rangle, \ldots, \langle l_n, s_n\rangle\}$, by the inductive hypothesis there are $L_1, \ldots, L_n \in \mathbb{N}$ such that for any $t_1, \ldots, t_n \in \mathbb{N}$, $L_1 + t_1 \cdot l_1 + s_1 \in \mathcal{L}(E_1), \ldots, L_n + t_n \cdot l_n + s_n \in \mathcal{L}(E_1)$ hold. Let $L = \sum_{i=1}^{n} L_i + 2 \cdot (\max(l_1, \ldots, l_n, s_1, \ldots, s_n))^2 + \sum_{i=1}^{n} s_i$. From Lemma 1, we know that for any integer $t \in \mathbb{N}$, there are $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{N}$ such that

$$\begin{aligned} & L + t \cdot l \\ = & (\sum_{i=1}^{n} L_i + 2 \cdot (\max(l_1, \ldots, l_n, s_1, \ldots, s_n))^2 + \sum_{i=1}^{n} s_i) + t \cdot l \\ = & \sum_{i=1}^{n} (L_i + s_i) + \sum_{i=1}^{n} (x_i \cdot l_i + y_i \cdot s_i) \\ = & \sum_{i=1}^{n} (L_i + x_i \cdot l_i + s_i) + \sum_{i=1}^{n} (y_i \cdot s_i). \end{aligned}$$

By the inductive hypothesis, we have $L_i + x_i \cdot l_i + s_i \in \mathcal{L}(E_1)$ $(1 \le i \le n)$. Moreover, from Observation 1, we know that $\sum_{i=1}^{n} (y_i \cdot s_i) \in \mathcal{L}(E_1^*)$. Hence $L + t \cdot l \in \mathcal{L}(E_1^*)$. Therefore the conclusion holds. □

From the above properties, we can conclude that there exists a number $M \in \mathbb{N}$ satisfying $|\mathcal{L}(E) \backslash \mathcal{P}| + |\mathcal{P} \backslash \mathcal{L}(E)| \le M$. Moreover, we had built the relation between words in $\mathcal{L}(E)$ and arithmetic progressions in $\mathcal{S}(E)$, and then we can check determinism of $\mathcal{L}(E)$ by investigating properties of $\mathcal{S}(E)$.

Now we analyze the time used to compute $\mathcal{S}(E)$. Given an expression E, we compute $\mathcal{S}(E)$ in the following way: We first construct the syntax tree of E, after that we use a bottom-up traversal to compute $\mathcal{S}$ for each node. It is known that for two m-bit numbers, the greatest common divisor can be computed in $O(m^2)$ time [34]. In our computation, the maximum number in $\mathcal{S}(E)$ is not larger than $|E|$. Then representing each number in $\mathcal{S}(E)$ only needs $O(\log |E|)$ bits. Moreover, for each node the algorithm for computing the greatest common divisor has to run $O(|E|^2)$ times, especially for the case $E_1 \cdot E_2$. Then computing $\mathcal{S}$ for each node takes $O(|E|^2 \cdot \log^2 |E|)$ time. Therefore the total time to compute $\mathcal{S}(E)$ is $O(|E|^3 \cdot \log^2 |E|)$.

Given an NFA $\mathcal{N}$, Sawa [35] also gave an algorithm to construct a set of arithmetic progressions such that the union of these arithmetic progressions is the language accepted by $\mathcal{N}$. The algorithm runs in $O(n^2 \cdot (n + m))$ time, where n is the number of states in $\mathcal{N}$ and m is the number of transitions in $\mathcal{N}$. The advantage of our method is that it works merely on original expressions and reaches some kind of the lower bound for the algorithm in [35], since there is an expression E_n such that $|E_n| = n$ and every NFA describing $\mathcal{L}(E_n)$ has $\Omega(n \cdot (\log n)^2)$ transitions [36, 37]. But the price is that we add words in languages. However, we will see later that adding such words does not affect determinism of languages.

4. Determinism of unary languages

In the previous section, we had derived a set of arithmetic progressions from a given expression E. We will show how to use these arithmetic progressions to check determinism of $\mathcal{L}(E)$ in this section.

4.1. Decision problems for Covering Systems

The *covering problem* (**CP**) is the following problem: Whether a given set of arithmetic progressions $\{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$, with $0 \le s_i < l_i$ $(1 \le i \le n)$, forms a covering system? The complexity of **CP** is shown in the following theorem.

Theorem 1 ([32],[33])**.** **CP** *is* **coNP**-*complete*[3].

Similarly, an equal difference covering system (**EDCS**) is a set $\mathcal{P}$ of arithmetic progressions, $\{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$, with $0 \le s_i < l_i$ $(1 \le i \le n)$, such that there exist two integers (y, x) $(0 \le x < y)$ satisfying the following condition: For any integer k $(k \ge 0)$, $x \equiv k \pmod{y}$ if and only if $k \equiv s_i \pmod{l_i}$ for some i $(1 \le i \le n)$. Its size is defined as $\sum_{i=1}^{n} (l_i + s_i)$. We define (y, x) as the answer to $\mathcal{P}$. Since $l_i > 0$ $(0 \le i \le n)$, it is easy to see that $y > 0$. Let $\mathcal{P} = \{\langle 4, 1\rangle, \langle 4, 3\rangle\}$. It is straightforward that $\mathcal{P}$ is an **EDCS**, but is not a **CS**. The answer to $\mathcal{P}$ is $(2, 1)$. Intuitively, the union of the numbers represented by an **EDCS** forms an arithmetic progression, while the union of the numbers represented by a **CS** contains all non-negative integers. The arithmetic progression represented by union of arithmetic progressions is studied in the evenly spaced integer topology [38].

[3]This theorem also holds for more general cases [33], where both of l and s are represented by binary numbers. However, we concentrate on unary languages of standard expressions here. For this restricted case, the theorem also holds [32].

The equal difference covering problem (**EDCP**) is defined as follows: Whether a given set of arithmetic progressions $\{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$, with $0 \le s_i < l_i$ $(1 \le i \le n)$, forms an **EDCS**?

The complexity of **EDCP** can be easily proved by a reduction from **CP**. We show the proofs here for the sake of completeness.

At first, we show how to compute the answer to an **EDCS**.

Lemma 2. *Suppose $\mathcal{P}$ is an* **EDCS**. *The answer to $\mathcal{P}$ is unique.*

For a given set $\mathcal{P}$ of arithmetic progressions $\{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$, we denote $L = \gcd(l_1, l_2, \ldots, l_n)$ and suppose $0 < p_1 < p_2 < p_3 \ldots < p_m$ are the distinct divisors of L.

Lemma 3. *Suppose $\mathcal{P}$ is an* **EDCS** *and the answer to $\mathcal{P}$ is (y, x). Then there is an integer k such that $y = p_k$, $k = \max\{l | \forall i \forall j(\langle l_i, s_i\rangle \in \mathcal{P} \wedge \langle l_j, s_j\rangle \in \mathcal{P} \wedge s_i \equiv s_j \pmod{p_l})\}$ and $x = (s_1 \bmod p_k)$.*

Proof. For any arithmetic progression $\langle l_i, s_i\rangle$, we have $x \equiv s_i \pmod y$ and $x \equiv (s_i + l_i) \pmod y$. Then $y | l_i$. So $y | L$ and there is an integer k such that $y = p_k$. For any s_i, s_j $(1 \le i, j \le n)$, by the definition of **EDCS**, we have $s_i \equiv x \pmod{p_k}$, $s_j \equiv x \pmod{p_k}$, and $p_k | (s_i - s_j)$. Let $k' = \max\{l | \forall i \forall j(\langle l_i, s_i\rangle \in \mathcal{P} \wedge \langle l_j, s_j\rangle \in \mathcal{P} \wedge s_i \equiv s_j \pmod{p_l})\}$. Because $\mathcal{P}$ is an **EDCS** and (p_k, x) is the answer to $\mathcal{P}$, we can show that $p_{k'} = p_k$. Moreover, since $\mathcal{P}$ is an **EDCS**, we have $x < p_k$ and there is an integer i such that $x + i \cdot p_k = s_1$. Hence $x = (s_1 \bmod p_k)$. □

Given a set $\mathcal{P}$ of arithmetic progressions, if we know $\mathcal{P}$ is an **EDCS**, then we can find the answer to $\mathcal{P}$ from the arithmetic progressions in $\mathcal{P}$. Since we define the size of a cover system as $\sum_{i=1}^{n}(l_i + s_i)$, the answer to $\mathcal{P}$ is polynomial-time computable. It is easy to see that the converse of the lemma does not hold. Consider the following set of arithmetic progressions: $\{\langle 3, 0\rangle, \langle 4, 0\rangle\}$. $y = 1$ and $x = 0$ satisfy all the conditions, but obviously this set is not an **EDCS**.

Bickel et al. [29] gave a method to construct a covering system from a set $\mathcal{P}$ of arithmetic progressions, where the union of numbers represented by these arithmetic progressions contains an arithmetic progression. Inspired by this idea, we can construct a covering system from an **EDCS**, and vice versa.

Theorem 2. **EDCP** *is* **coNP**-*complete.*

Proof. At first, we prove that the problem is **coNP**-hard. This can be proved by a reduction from **CP**. Given a set $\mathcal{P}$ of arithmetic progressions $\{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$, we construct the set $\mathcal{P}_1 = \{\langle 3 \cdot l_1, 3 \cdot s_1\rangle, \langle 3 \cdot l_2, 3 \cdot s_2\rangle, \ldots, \langle 3 \cdot l_n, 3 \cdot s_n\rangle\}$. Using the definition of **CS** and **EDCS**, we can show that: $\mathcal{P}$ is a **CS**, if and only if, $\mathcal{P}_1$ is an **EDCS** and the answer to $\mathcal{P}_1$ is $(3, 0)$. From Theorem 1 we conclude that **EDCP** is **coNP**-hard.

Next, we show that **EDCP** is in **coNP**. This can be proved by a reduction to **CP**. Suppose $\mathcal{P}$ is the set of arithmetic progressions $\{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$. Then let $p = \gcd(l_1, l_2, \ldots, l_n)$. From the definition of **EDCP**, we know that $p > 0$ must hold. Suppose $0 < p_1 < p_2 < \ldots < p_m$ are the distinct divisors of p. We look for an integer k such that $k = \max\{l | \forall i \forall j(\langle l_i, s_i\rangle \in \mathcal{P} \wedge \langle l_j, s_j\rangle \in \mathcal{P} \wedge s_i \equiv s_j \pmod{p_l})\}$. If there is not such k, then from Lemma 3 we know that $\mathcal{P}$ is not an **EDCS**. Hence suppose there is a k satisfying the condition. Denote $s = (s_1 \bmod p_k)$. We construct the following set $\mathcal{Q}$ of arithmetic progressions: $\{\langle \frac{l_1}{p_k}, \frac{s_1 - s}{p_k}\rangle, \langle \frac{l_2}{p_k}, \frac{s_2 - s}{p_k}\rangle, \ldots, \langle \frac{l_n}{p_k}, \frac{s_n - s}{p_k}\rangle\}$. We can show that: $\mathcal{Q}$ is a **CS**, if and only if, $\mathcal{P}$ is an **EDCS**. Then to decide whether $\mathcal{P}$ is an **EDCS**, we only need to check whether $\mathcal{Q}$ is a **CS**. Since **CP** is in **coNP** and the computations of p, p_k and s take polynomial time, **EDCP** is in **coNP**. □

4.2. The complexity of determinism of unary languages

In this section, we will discuss the complexity of deciding determinism of unary languages. For an expression E, suppose $\mathcal{S}(E) = \{\langle l_1, s_1\rangle, \langle l_2, s_2\rangle, \ldots, \langle l_n, s_n\rangle\}$, and define $\mathcal{L}(\mathcal{S}(E)) = \bigcup_{\langle l,s\rangle \in \mathcal{S}(E)} \{k \cdot l + s | k \in \mathbb{N}\}$.

From Proposition 1 and Proposition 2, we know that $\mathcal{L}(E) \subseteq \mathcal{L}(\mathcal{S}(E))$, and there are only finite many words w such that $w \in \mathcal{L}(\mathcal{S}(E))$ and $w \notin \mathcal{L}(E)$. The relation between determinism of $\mathcal{L}(\mathcal{S}(E))$ and $\mathcal{L}(E)$ can be established by the following lemma.

Lemma 4 ([8])**.** *For any deterministic language L, the following statements hold:* (1) *If string $w \in L$, then the language $L \setminus \{w\}$ is deterministic;* (2) *If string $w \notin L$, then the language $L \cup \{w\}$ is deterministic.*

Corollary 1. *Given an expression E, $\mathcal{L}(\mathcal{S}(E))$ is deterministic if and only if $\mathcal{L}(E)$ is deterministic.*

Then to check determinism of $\mathcal{L}(E)$, we only need to check determinism of $\mathcal{L}(\mathcal{S}(E))$. Now we study how to check determinism of $\mathcal{L}(\mathcal{S}(E))$. For a unary language, the corresponding minimal DFA consists of a chain of states or a chain followed by a cycle [19]. And deterministic regular languages have the following characterization.

Lemma 5 ([19])**.** *Let $\mathcal{L}$ be a regular language, then $\mathcal{L}$ is a deterministic language if and only if $\mathcal{L}$ is finite or the cycle of the minimal DFA of $\mathcal{L}$ has at most one final state.*

From the definition of $\mathcal{L}(\mathcal{S}(E))$ and this characterization, we can easily see that to check determinism of $\mathcal{L}(\mathcal{S}(E))$ we only need to check whether $\mathcal{S}(E)$ is an **EDCS**.

But the tuples $\langle l, s\rangle \in \mathcal{S}(E)$ may not satisfy the basic condition $0 \leq s < l$. Then we simplify $\mathcal{S}(E)$ in the following ways: (1) Delete all arithmetic progressions in the form $\langle 0, s\rangle$; (2) For $\langle l, s\rangle$ with $l \leq s$, we replace $\langle l, s\rangle$ with $\langle l, s \bmod l\rangle$. After the simplification, every arithmetic progression $\langle l, s\rangle \in \mathcal{S}(E)$ satisfies $0 \leq s < l$. This process just deletes or adds a finite number of words in $\mathcal{L}(\mathcal{S}(E))$. From Lemma 4, we know that it does not change determinism of the language. From now on, when we say $\mathcal{S}(E)$, we mean the simplified one.

Theorem 3. *Suppose $\mathcal{L}(\mathcal{S}(E))$ is infinite. $\mathcal{L}(\mathcal{S}(E))$ is deterministic if and only $\mathcal{S}(E)$ is an* **EDCS**.

Proof. ($\Rightarrow$) Since $\mathcal{L}(\mathcal{S}(E))$ is deterministic, the cycle of the minimal DFA $\mathcal{M}$ of $\mathcal{L}(\mathcal{S}(E))$ has at most one final state. Let the size of the cycle be p. Since $\mathcal{L}(\mathcal{S}(E))$ is infinite, $p \neq 0$. Denote the start state as q_0 and the only final state in the cycle as q_1. Suppose w is the shortest word such that $\delta(q_0, w) = q_1$. Let $s = (w \bmod p)$. We can prove that (p, s) is the answer to $\mathcal{S}(E)$. Actually, this can be easily deduced from the following relation between (p, s) and any $\langle l_i, s_i\rangle \in \mathcal{S}(E)$: $p|l_i$ and $w \equiv s_i \pmod p$. It remains to verify this relation.

For any $\langle l_i, s_i\rangle$ $(1 \leq i \leq n)$, since q_1 is the only final state in the cycle, there is an integer k'' such that $k'' \cdot l_i + s_i > w$, $\delta(q_0, k'' \cdot l_i + s_i) = q_1$ and $\delta(q_0, k'' \cdot l_i + l_i + s_i) = q_1$. Then $p|l_i$ and there is an integer k_1 such that $w + k_1 \cdot p = k'' \cdot l_i + s_i$. Hence $w \equiv s_i \pmod p$.

($\Leftarrow$) Suppose $\mathcal{S}(E)$ is an **EDCS**, and (p, s) is the answer to $\mathcal{S}(E)$. If $\mathcal{L}(\mathcal{S}(E))$ is not deterministic, then there are two final states p_1 and p_2 in the cycle of the minimal DFA of $\mathcal{L}(\mathcal{S}(E))$. Denote the start state as q_0. Then there are words k_1 and k_2 such that $\delta(q_0, k_1) = q_1$, $\delta(q_0, k_2) = q_2$, $k_1 \in \mathcal{L}(\mathcal{S}(E))$ and $k_2 \in \mathcal{L}(\mathcal{S}(E))$. Let F be the set of final states of the minimal DFA of $\mathcal{L}(\mathcal{S}(E))$. Because p_1 and p_2 are not equivalent, there is a word k such that $\delta(q_1, k) \in F \wedge \delta(q_2, k) \notin F$ or $\delta(q_1, k) \notin F \wedge \delta(q_2, k) \in F$ holds. Suppose the case $\delta(q_1, k) \in F \wedge \delta(q_2, k) \notin F$ holds. From $\mathcal{S}(E)$ is an **EDCS** and $\delta(q_1, k) \in F$, we can show that $p|k$. On the other hand, from $\delta(q_2, k) \notin F$, we have $k + k_2 \not\equiv s \pmod p$. Then $p \nmid k$, which is a contradiction. Hence $\mathcal{L}(\mathcal{S}(E))$ is deterministic. The case $\delta(q_1, k) \notin F \wedge \delta(q_2, k) \in F$ can be proved in a similar way. □

From Theorem 2 and Theorem 3, we can obtain the complexity of checking determinism of $\mathcal{L}(E)$ for an expression E as follows.

Theorem 4. *Given a regular expression E, the problem of deciding whether $\mathcal{L}(E)$ is deterministic is* **coNP**-*complete.*

For any expression $E = E_1^*$, we have $|\mathcal{S}(E_1^*)| = 1$ from the definition of $\mathcal{S}$. Then $\mathcal{S}(E_1^*)$ is an **EDCS**. Hence we can easily obtain the following theorem.

Theorem 5 ([14])**.** *Let $\mathcal{L}$ be any unary language. Then $\mathcal{L}^*$ is deterministic.*[4]

5. The complexity of determinism of unary languages in R(#)

In this section, we consider the case for expressions with counting.

[4] This theorem also follows from Lemma 4.1 in [39] and Lemma 4.

5.1. The main idea

Suppose E is an expression in R(#) and the minimal DFA for $\mathcal{L}(E)$ is $\mathcal{M}$. To check determinism of $\mathcal{L}(E)$, we first give an upper bound for the size of $\mathcal{M}$, and then develop an algorithm to check non-determinism of $\mathcal{L}(E)$.

At first, we give an upper bound for the size of $\mathcal{M}$. Pighizzini et al. [28] had proved that for any unary context-free grammar G in Chomsky normal form with at least two variables, there will be a DFA $\mathcal{D}$ with less than 2^{h^2} states accepting $\mathcal{L}(G)$, where h is the number of variables in G. To use this result, we need to derive from E a context-free grammar satisfying the required conditions.

Based on the structure of E, we can inductively construct a unary context-free grammar $G = (V_E, \{a\}, P_E, S)$ with at most $2 \cdot |E|$ variables accepting $\mathcal{L}(E)$. The key point of the construction is how to handle subexpressions like $E_1^{[m,n]}$ and $E_1^{[m,\infty]}$. A straightforward way to change $E_1^{[m,n]}$ and $E_1^{[m,\infty]}$ into a grammar in Chomsky normal form will introduce exponentially many variables [22, 24]. To avoid this, we use the ideas of bisection. The construction is quite standard. Take the case $E = E_1^{[2,3]}$ as an example. Suppose $\mathcal{L}(A_{E_1}) = \mathcal{L}(E_1)$, $(2)_{10} = (10)_{bin}$ [5], and $(3-2)_{10} = (1)_{bin}$. Then we can use the following grammar to represent $\mathcal{L}(E)$:

$$A_E \to D_1F_0 \qquad D_1 \to B_1D_0 \qquad D_0 \to \varepsilon \qquad F_0 \to B'_0$$
$$B_1 \to B_0B_0 \qquad B_0 \to A_{E_1} \qquad B'_0 \to A_{E_1} \qquad B'_0 \to \varepsilon$$

From the above construction, we can show that there are at most $2 \cdot |E|$ variables in G. However, G is not in Chomsky normal form. We can use the algorithm in [24] to change G into a new grammar G' such that G' is in Chomsky normal form and $\mathcal{L}(G') = \mathcal{L}(G) \setminus \{\varepsilon\}$. Moreover, variables in G' are the same as the ones in G. Then from [28], we know that there is a DFA $\mathcal{D}$ with no more than $2^{4\cdot|E|^2}$ states accepting $\mathcal{L}(E)$. Therefore, the size of $\mathcal{M}$ cannot be larger than $2^{4\cdot|E|^2}$.

Using this bound, we can obtain an algorithm to check non-determinism of $\mathcal{L}(E)$. At first, we give another characterization of determinism of regular languages. Suppose the initial state of $\mathcal{M}$ is q_0.

Lemma 6. *Suppose $\mathcal{L}(\mathcal{M})$ is infinite. Then $\mathcal{L}(\mathcal{M})$ is not deterministic if and only if there are three words w_1, w_2, and w_3 such that $|w_1| > |\mathcal{M}|$, $|w_2| > |\mathcal{M}|$, $|w_3| > |\mathcal{M}|$, $w_1 \in \mathcal{L}(\mathcal{M})$, $w_2 \in \mathcal{L}(\mathcal{M})$, $w_1 \cdot w_3 \in \mathcal{L}(\mathcal{M})$, and $w_2 \cdot w_3 \notin \mathcal{L}(\mathcal{M})$.*

Proof. From Lemma 5 and the definition of equivalent states, we know that $\mathcal{L}(\mathcal{M})$ is not deterministic, if and only if, there are two non-equivalent final states, denoted by $q_1 = \delta(q_0, w'_1)$ and $q_2 = \delta(q_0, w'_2)$, in the cycle of $\mathcal{M}$, if and only if, there is a word w' such that one of $\delta(q_1, w')$ and $\delta(q_2, w')$ is a final state, while the other is not. We assume, *w.l.o.g*, that $\delta(q_1, w')$ is a final state. Suppose the length of the cycle of $\mathcal{M}$ is C. Let $w_1 = w'_1 + |\mathcal{M}| \cdot C$, $w_2 = w'_2 + |\mathcal{M}| \cdot C$, and $w_1 = w' + |\mathcal{M}| \cdot C$. From the structure of $\mathcal{M}$, it can be verified that w_1, w_2, and w_3 satisfy the required conditions. Then the conclusion holds. □

Following this lemma, to check non-determinism of $\mathcal{L}(E)$, we first guess three sufficiently long words w_1, w_2, and w_3 such that the sizes of w_1, w_2, and w_3 are larger than the size of $\mathcal{M}$. Let $q_1 = \delta(q_0, w_1)$ and $q_2 = \delta(q_0, w_2)$. Then q_1 and q_2 are states in the cycle of $\mathcal{M}$. If $w_1 \in \mathcal{L}(E)$, $w_2 \in \mathcal{L}(E)$, $w_1 \cdot w_3 \in \mathcal{L}(E)$, and $w_2 \cdot w_3 \notin \mathcal{L}(E)$, then q_1 and q_2 are two distinct final states in the cycle, and we can conclude that $\mathcal{L}(E)$ is not deterministic.

By extending the function $\mathcal{S}$, we can improve the upper bound for $|\mathcal{M}|$ to $2^{O(|E|)}$. Since Kilpeläinen and Tuhkanen [20, 21] had shown that the lower bound is $2^{\Omega(|E|)}$, our upper bound is tight.

5.2. The arithmetic progressions of regular expressions in R(#)

At first, we define the $\mathcal{S}$ function for expressions in R(#). We can define it as before, but using such definition cannot improve the upper bound $2^{4\cdot|E|^2}$. We need more details about the structure of words in $\mathcal{L}(E)$ as done in the proof of Lemma 4 in [28]. So we define the following intermediate function $\mathcal{S}_1$. Note that in this definition, the pairs in $\mathcal{S}_1(E)$ are not in the form $\langle l, s\rangle$ as before, but pairs of the form $\langle\{l_1, l_2, ..., l_n\}, s\rangle$ representing the sets of numbers, which can be written as $k_1 \cdot l_1 + k_2 \cdot l_2 + ... + k_n \cdot l_n + s$.

[5] $(m)_{10} = (b_ib_{i-1}\dots b_0)_{bin}$ stands for that the binary encoding of the decimal number m is $b_ib_{i-1}\dots b_0$.

Definition 3. *The function $\mathcal{S}_1(E)$ is defined as*

$$\begin{aligned}
&\mathcal{S}_1(\varepsilon) = \{\langle\{0\}, 0\rangle\}\\
&\mathcal{S}_1(a) = \{\langle\{0\}, 1\rangle\}, \quad a \in \Sigma\\
&\mathcal{S}_1(E_1 + E_2) = \mathcal{S}_1(E_1) \cup \mathcal{S}_1(E_2)\\
&\mathcal{S}_1(E_1E_2) = \{\langle L_1 \cup L_2, s_1 + s_2\rangle | \langle L_1, s_1\rangle \in \mathcal{S}_1(E_1) \wedge \langle L_2, s_2\rangle \in \mathcal{S}_1(E_2)\}\\
&\mathcal{S}_1(E_1^{[m,n]}) = \bigcup_{m\le i\le n} \{\langle \bigcup_{1\le j\le i} L_j, \sum_{j=1}^{i} s_j\rangle | \langle L_j, s_j\rangle \in \mathcal{S}_1(E_1)\}\\
&\mathcal{S}_1(E_1^{[m,\infty]}) = \{\langle \bigcup_{\langle L_k, s_k\rangle \in \mathcal{S}_1(E_1)} ((\bigcup_{l' \in L_k} \{l' + s_k\}) \cup \{s_k\}), \sum_{j=1}^{m} s_j\rangle | \langle L_j, s_j\rangle \in \mathcal{S}_1(E_1)\}.
\end{aligned}$$

The first four cases are easy to understand, we give a simple example to show the computations of the last two cases. Suppose $\mathcal{S}_1(E_1) = \{\langle\{2, 3\}, 1\rangle, \langle\{4, 5\}, 2\rangle\}$. Then

$$\begin{aligned}
&\mathcal{S}_1(E_1^{[2,3]})\\
&= \{\langle\{2, 3\}, 1 + 1\rangle, \langle\{2, 3, 4, 5\}, 1 + 2\rangle, \langle\{4, 5\}, 2 + 2\rangle,\\
&\quad \langle\{2, 3\}, 1 + 1 + 1\rangle, \langle\{2, 3, 4, 5\}, 1 + 1 + 2\rangle, \langle\{2, 3, 4, 5\}, 1 + 2 + 1\rangle,\\
&\quad \langle\{2, 3, 4, 5\}, 1 + 2 + 2\rangle,\\
&\quad \langle\{2, 3, 4, 5\}, 2 + 1 + 1\rangle, \langle\{2, 3, 4, 5\}, 2 + 1 + 2\rangle, \langle\{2, 3, 4, 5\}, 2 + 2 + 1\rangle,\\
&\quad \langle\{4, 5\}, 2 + 2 + 2\rangle\}\\
&= \{\langle\{2, 3\}, 2\rangle, \langle\{2, 3\}, 3\rangle, \langle\{4, 5\}, 4\rangle, \langle\{4, 5\}, 6\rangle,\\
&\quad \langle\{2, 3, 4, 5\}, 3\rangle, \langle\{2, 3, 4, 5\}, 4\rangle, \langle\{2, 3, 4, 5\}, 5\rangle\}
\end{aligned}$$

and,

$$\begin{aligned}
&\mathcal{S}_1(E_1^{[2,\infty]})\\
&= \{\langle\{2 + 1, 3 + 1, 1, 4 + 2, 5 + 2, 2\}, 1 + 1\rangle,\\
&\quad \langle\{2 + 1, 3 + 1, 1, 4 + 2, 5 + 2, 2\}, 1 + 2\rangle,\\
&\quad \langle\{2 + 1, 3 + 1, 1, 4 + 2, 5 + 2, 2\}, 2 + 1\rangle,\\
&\quad \langle\{2 + 1, 3 + 1, 1, 4 + 2, 5 + 2, 2\}, 2 + 2\rangle\}\\
&= \{\langle\{1, 2, 3, 4, 6, 7\}, 2\rangle, \langle\{1, 2, 3, 4, 6, 7\}, 3\rangle, \langle\{1, 2, 3, 4, 6, 7\}, 4\rangle\}.
\end{aligned}$$

The following example shows how to compute $\mathcal{S}_1(E)$ in a bottom-up manner.

Example 2. *Let $E = ((aaa + aa)^{[1,2]})^{[0,\infty]} + (aaa)^{[2,\infty]}((aa)^{[0,\infty]}aaa + (aaa)^{[0,\infty]}aa)$. The process of computing $\mathcal{S}_1(E)$ is shown in Table 2.*

The reason, why we define $\mathcal{S}_1(E_1^{[m,\infty]})$ in such a way, will be clear after the proof of Proposition 3.

We will give some bounds for the numbers occurring in tuples of $\mathcal{S}_1(E)$. Suppose $\langle L, s\rangle \in \mathcal{S}_1(E)$. The bound for s is given by the following function.

Definition 4. *The function $\mathcal{R}(E)$ is defined as*

$$\begin{aligned}
&\mathcal{R}(\varepsilon) = 0\\
&\mathcal{R}(a) = 1\\
&\mathcal{R}(E_1 + E_2) = \max(\mathcal{R}(E_1), \mathcal{R}(E_2))\\
&\mathcal{R}(E_1E_2) = \mathcal{R}(E_1) + \mathcal{R}(E_2)\\
&\mathcal{R}(E_1^{[m,n]}) = n \cdot \mathcal{R}(E_1)\\
&\mathcal{R}(E_1^{[m,\infty]}) = m \cdot \mathcal{R}(E_1).
\end{aligned}$$

According to the definition, we can obtain the following bounds. The proofs are quite straightforward.

E_1	$\mathcal{S}(E_1)$	E_1	$\mathcal{S}(E_1)$
a	$\langle\{0\},1\rangle$	$(aa+aaa)^{[1,2]}$	$\langle\{0\},2\rangle,\langle\{0\},3\rangle,\langle\{0\},4\rangle,\langle\{0\},5\rangle,\langle\{0\},6\rangle$
aa	$\langle\{0\},2\rangle$	$((aa+aaa)^{[1,2]})^{[0,\infty]}$	$\langle\{2,3,4,5,6\},0\rangle$
aaa	$\langle\{0\},3\rangle$	$(aa)^{[0,\infty]}aaa$	$\langle\{0,2\},3\rangle$
$aa+aaa$	$\langle\{0\},2\rangle\langle\{0\},3\rangle$	$(aaa)^{[0,\infty]}aa$	$\langle\{0,3\},2\rangle$
$(aa)^{[0,\infty]}$	$\langle\{2\},0\rangle$	$(aa)^{[0,\infty]}aaa+(aaa)^{[0,\infty]}aa$	$\langle\{0,2\},3\rangle\langle\{0,3\},2\rangle$
$(aaa)^{[0,\infty]}$	$\langle\{3\},0\rangle$	$(aaa)^{[2,\infty]}((aa)^{[0,\infty]}aaa+(aaa)^{[0,\infty]}aa)$	$\langle\{0,2,3\},9\rangle\langle\{0,3\},8\rangle$
$(aaa)^{[2,\infty]}$	$\langle\{3\},6\rangle$	$((aa+aaa)^{[1,2]})^{[0,\infty]}+(aaa)^{[2,\infty]}((aa)^{[0,\infty]}aaa+(aaa)^{[0,\infty]}aa)$	$\langle\{2,3,4,5,6\},0\rangle,\langle\{0,2,3\},9\rangle\langle\{0,3\},8\rangle$

Table 2: The process of computing $\mathcal{S}_1(E)$

Observation 2. *Let E be an expression in R(#). We have $\mathcal{R}(E) \le 2^{|E|}$.*

Lemma 7. *Let E be an expression in R(#). For any $\langle L, s\rangle \in \mathcal{S}_1(E)$, we have $s \le \mathcal{R}(E)$.*

Lemma 8. *Let E be an expression in R(#). There exists a tuple $\langle L, s\rangle \in \mathcal{S}_1(E)$ such that $s = \mathcal{R}(E)$.*

Lemma 9. *Let E be an expression in R(#). For any $\langle L, s\rangle \in \mathcal{S}_1(E)$, we have $s \in \mathcal{L}(E)$.*

Lemma 10. *Let E be an expression in R(#). For any $\langle L, s\rangle = \langle\{l_1, l_2, \ldots, l_r\}, s\rangle \in \mathcal{S}_1(E)$, we have $l_i \le 2^{|E|}$ $(1 \le i \le r)$.*

The following proposition is central to the proof of the upper bound $2^{O(|E|)}$. To simplify the proof, we define the following indicator function.

$$\mathbf{1}_S(x) = \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{if } x \notin S; \end{cases}$$

Proposition 3. *Let E be an expression in R(#) and $\langle L, s\rangle = \langle\{l_1, \ldots, l_r\}, s\rangle \in \mathcal{S}_1(E)$. For any $x_1, \ldots, x_r \in \mathbb{N}$, we have $f(L, s) = \sum_{i=1}^{r} x_i \cdot l_i + s \in \mathcal{L}(E)$.*

Proof. We prove it by induction on the structure of E. The cases $E = \varepsilon$ or a (where $a \in \Sigma$) are obvious.

$\mathbf{E = E_1 + E_2}$: From the definition of $\mathcal{S}_1$, we have $\langle L, s\rangle \in \mathcal{S}_1(E_1)$ or $\langle L, s\rangle \in \mathcal{S}_1(E_2)$. If $\langle L, s\rangle \in \mathcal{S}_1(E_1)$, then by the inductive hypothesis, for any $x_1, \ldots, x_r \in \mathbb{N}$, we have $f(L, s) \in \mathcal{L}(E_1)$. Then since $E = E_1 + E_2$, we have $f(L, s) \in \mathcal{L}(E)$. The case $\langle L, s\rangle \in \mathcal{S}_1(E_2)$ can be proved in a similar way.

$\mathbf{E = E_1E_2}$: From the definition of $\mathcal{S}_1$, there are $\langle L_1, s_1\rangle = \langle\{l_{1,1}, \ldots, l_{1,r_1}\}, s_1\rangle \in \mathcal{S}_1(E_1)$ and $\langle L_2, s_2\rangle = \langle\{l_{2,1}, \ldots, l_{2,r_2}\}, s_2\rangle \in \mathcal{S}_1(E_2)$ such that $L = L_1 \cup L_2$ and $s = s_1 + s_2$. Denote $T = L_1 \cap L_2$. By the inductive hypothesis, for any $x^1_1, \ldots, x^1_{r_1} \in \mathbb{N}$, and $x^2_1, \ldots, x^2_{r_2} \in \mathbb{N}$, we know that $f(L_1, s_1) \in \mathcal{L}(E_1)$ and $f(L_2, s_2) \in \mathcal{L}(E_2)$. In fact, $f(L_1, s_1)$ and $f(L_2, s_2)$ correspond to $\sum_{i=1}^{r_1} x^1_i \cdot l_{1,i} + s_1$ and $\sum_{i=1}^{r_2} x^2_i \cdot l_{2,i} + s_2$, respectively. The full expansions will mess the following proof, so we use these simple notations. It will be clear from the context what $f(L', s')$ means. Then for any $x_1, \ldots, x_r \in \mathbb{N}$, there are $y^1_1, y^2_1, \ldots, y^1_r, y^2_r \in \mathbb{N}$ such that $x_1 = y^1_1 + y^2_1, \ldots, x_r = y^1_r + y^2_r$, and

$$\begin{aligned} & f(L, s) \\ &= \sum_{i=1}^{r} (x_i \cdot l_i) + s \\ &= \sum_{i=1}^{r} [(\mathbf{1}_{L_1 \setminus T}(l_i) + \mathbf{1}_T(l_i) + \mathbf{1}_{L_2 \setminus T}(l_i)) \cdot x_i \cdot l_i] + s_1 + s_2 \\ &= \sum_{i=1}^{r} (\mathbf{1}_{L_1 \setminus T}(l_i) \cdot x_i \cdot l_i) + \sum_{i=1}^{r} (\mathbf{1}_T(l_i) \cdot x_i \cdot l_i) + \sum_{i=1}^{r} (\mathbf{1}_{L_2 \setminus T}(l_i) \cdot x_i \cdot l_i) + s_1 + s_2 \end{aligned}$$

$$= \sum_{i=1}^{r}(\mathbf{1}_{L_1\setminus T}(l_i)\cdot x_i\cdot l_i) + \sum_{i=1}^{r}(\mathbf{1}_T(l_i)\cdot(y_i^1+y_i^2)\cdot l_i) + \sum_{i=1}^{r}(\mathbf{1}_{L_2\setminus T}(l_i)\cdot x_i\cdot l_i) + s_1 + s_2$$
$$= \sum_{i=1}^{r}(\mathbf{1}_{L_1\setminus T}(l_i)\cdot x_i\cdot l_i) + \sum_{i=1}^{r}(\mathbf{1}_T(l_i)\cdot y_i^1\cdot l_i) + s_1$$
$$+ \sum_{i=1}^{r}(\mathbf{1}_T(l_i)\cdot y_i^2\cdot l_i) + \sum_{i=1}^{r}(\mathbf{1}_{L_2\setminus T}(l_i)\cdot x_i\cdot l_i) + s_2.$$

By the inductive hypothesis, we know that $\sum_{i=1}^{r}(\mathbf{1}_{L_1\setminus T}(l_i)\cdot x_i\cdot l_i) + \sum_{i=1}^{r}(\mathbf{1}_T(l_i)\cdot y_i^1\cdot l_i) + s_1 \in \mathcal{L}(E_1)$ and $\sum_{i=1}^{r}(\mathbf{1}_T(l_i)\cdot y_i^2\cdot l_i) + \sum_{i=1}^{r}(\mathbf{1}_{L_2\setminus T}(l_i)\cdot x_i\cdot l_i) + s_2 \in \mathcal{L}(E_2)$. Hence $f(L,s)\in\mathcal{L}(E)$.

$\mathbf{E} = \mathbf{E_1^{[m,n]}}$: Suppose $\langle L,s\rangle \in \mathcal{S}_1(E)$, there exist $\langle L_1,s_1\rangle \in \mathcal{S}_1(E_1)$, …, $\langle L_i,s_i\rangle \in \mathcal{S}_1(E_1)$ such that $m \le i \le n$, $L = \bigcup_{1\le j\le i} L_j$, and $s = \sum_{j=1}^{i} s_j$. By the inductive hypothesis, for any $x_{1,1},\dots,x_{1,r_1},\dots,x_{i,1},\dots,x_{i,r_i} \in \mathbb{N}$, we have $f(L_1,s_1)\in\mathcal{L}(E_1)$, …, and $f(L_i,s_i)\in\mathcal{L}(E_1)$. Then for any $x_1,\dots,x_r\in\mathbb{N}$,

$$f(L,s)$$
$$= \sum_{k=1}^{r}(x_k\cdot l_k) + s$$
$$= \sum_{k=1}^{r}[(\sum_{j=1}^{i}(\mathbf{1}_{L_j}(l_k)\cdot x_k^j))\cdot l_k] + \sum_{j=1}^{i} s_j$$
$$= \sum_{j=1}^{i}[\sum_{k=1}^{r}(\mathbf{1}_{L_j}(l_k)\cdot x_k^j\cdot l_k) + s_j].$$

By the inductive hypothesis, we know that $[\sum_{k=1}^{r}(\mathbf{1}_{L_j}(l_k)\cdot x_k^j\cdot l_k) + s_j] \in \mathcal{L}(E_1)$. Because $m\le i\le n$, we know that $f(L,s)\in\mathcal{L}(E)$.

$\mathbf{E} = \mathbf{E_1^{[m,\infty]}}$: Let $\mathcal{S}_1(E_1) = \{\langle L_1,s_1\rangle,\dots,\langle L_n,s_n\rangle\}$. Then from the definition of $\mathcal{S}_1$, we know that $L = \bigcup_{\langle L_k,s_k\rangle\in\mathcal{S}_1(E_1)}((\bigcup_{l'\in L_k}\{l'+s_k\})\cup\{s_k\})$. And there exist $\langle L_1',s_1'\rangle,\dots,\langle L_m',s_m'\rangle\in\mathcal{S}_1(E_1)$ such that $s=\sum_{j=1}^{m}s_j'$. By the inductive hypothesis for any $x_{1,1},\dots,x_{1,r_1},\dots,x_{n,1},\dots,x_{n,r_n}\in\mathbb{N}$, we have $f(L_1,s_1),\dots,f(L_n,s_n)\in\mathcal{L}(E_1)$. Let $T_1 = \bigcup_{\langle L_k,s_k\rangle\in\mathcal{S}_1(E_1)}(\bigcup_{l'\in L_k}\{l'+s_k\})$ and $T_2 = \bigcup_{\langle L_k,s_k\rangle\in\mathcal{S}_1(E)}\{s_k\}$. Then for any $x_1,\dots,x_r\in\mathbb{N}$,

$$f(L,s)$$
$$= \sum_{k=1}^{r}(x_k\cdot l_k) + s$$
$$= \sum_{k=1}^{r}[(\mathbf{1}_{T_1}(l_k) + \mathbf{1}_{T_2\setminus T_1}(l_k))\cdot x_k\cdot l_k] + \sum_{j=1}^{m} s_j'$$
$$= \sum_{k=1}^{r}(\mathbf{1}_{T_1}(l_k)\cdot x_k\cdot l_k) + \sum_{k=1}^{r}(\mathbf{1}_{T_2\setminus T_1}(l_k)\cdot x_k\cdot l_k) + \sum_{j=1}^{m} s_j'.$$

If $\mathbf{1}_{T_1}(l_k)\neq 0$ for some integer k ($1\le k\le r$), by the inductive hypothesis, $\mathbf{1}_{T_1}(l_k)\cdot l_k = l_k = l' + s_k \in \mathcal{L}(E_1)$. Hence $\sum_{k=1}^{r}(\mathbf{1}_{T_1}(l_k)\cdot x_k\cdot l_k)\in\mathcal{L}(E_1^{[0,\infty]})$. From Lemma 9, we know that $\sum_{k=1}^{r}(\mathbf{1}_{T_2\setminus T_1}(l_k)\cdot x_k\cdot l_k)\in\mathcal{L}(E_1^{[0,\infty]})$, and $\sum_{j=1}^{m}s_j'\in\mathcal{L}(E_1^{[m,\infty]})$. Therefore $f(L,s)\in\mathcal{L}(E)$.

□

From the proof in the last case, we can see the reason for using $\bigcup_{l'\in L_k}\{l'+s_k\}$ to define $\mathcal{S}_1(E_1^{[m,\infty]})$, instead of $\bigcup_{l'\in L_k}\{l'\}$. The inductive hypothesis cannot ensure $l'\in\mathcal{L}(E_1)$. Having this property, we can define the $\mathcal{S}$ function as follows.

Definition 5. *The function $\mathcal{S}(E)$ is defined as*

$$\mathcal{S}(E) = \{\langle \gcd(L), s\rangle | \langle L,s\rangle\in\mathcal{S}_1(E)\}.$$

If $\mathcal{S}(E) = \{\langle l_1, s_1\rangle, \ldots, \langle l_n, s_n\rangle\}$, we also define $\gcd(\mathcal{S}(E)) = \gcd(l_1, l_2, \ldots, l_n, s_1, \ldots, s_n)$. Now we can prove that the function $\mathcal{S}$ can be computed as before.

Lemma 11. *Let E be a regular expression in $R(\#)$.*
(1) $E = \varepsilon : \mathcal{S}(\varepsilon) = \{\langle 0, 0\rangle\}$
(2) $a \in \Sigma : \mathcal{S}(a) = \{\langle 0, 1\rangle\}, \ a \in \Sigma$
(3) $E = E_1 + E_2 : \mathcal{S}(E_1 + E_2) = \mathcal{S}(E_1) \cup \mathcal{S}(E_2)$
(4) $E = E_1E_2$:

$$\mathcal{S}(E_1E_2) = \{\langle \gcd(l_1, l_2), s_1 + s_2\rangle | \langle l_1, s_1\rangle \in \mathcal{S}(E_1) \wedge \langle l_2, s_2\rangle \in \mathcal{S}(E_2)\}$$

(5) $E = E_1^{[m,n]}$:

$$\mathcal{S}(E_1^{[m,n]}) = \bigcup_{m \le i \le n} \{\langle \gcd(l_1, l_2, \ldots, l_i), \sum_{j=1}^{i} s_j\rangle | \langle l_j, s_j\rangle \in \mathcal{S}(E_1)\}$$

(6) $E = E_1^{[m,\infty]}$:

$$\mathcal{S}(E_1^{[m,\infty]}) = \{\langle \gcd(\mathcal{S}(E_1)), \sum_{j=1}^{m} s_j\rangle | \langle l_j, s_j\rangle \in \mathcal{S}(E_1)\}.$$

Moreover, the $\mathcal{S}$ function has properties similar to the ones in Proposition 1 and Proposition 2.

Proposition 4. *Let E be an expression in $R(\#)$. For any $w \in \mathcal{L}(E)$, there exist an arithmetic progression $\langle l, s\rangle \in \mathcal{S}(E)$ and an integer $k \in \mathbb{N}$ such that $|w| = k \cdot l + s$.*

Proof. Since the proof of this proposition is almost the same as the one for Proposition 1, we omit the details here. □

Proposition 5. *Let E be an expression in $R(\#)$. For any $\langle l, s\rangle \in \mathcal{S}(E)$ there exists $L \in \mathbb{N}$ such that $L + t \cdot l + s \in \mathcal{L}(E)$ for any $t \in \mathbb{N}$, and $L \le 2^{2|E|+1}$.*

Proof. For any $\langle l, s\rangle \in \mathcal{S}(E)$, there exists a tuple $\langle\{l_1, l_2, \ldots, l_r\}, s\rangle \in \mathcal{S}_1(E)$ such that $l = \gcd(l_1, l_2, \ldots, l_r)$. From Proposition 3, we have $\sum_{i=1}^{r}(x_i \cdot l_i) + s \in \mathcal{L}(E)$ for any $x_1, \ldots, x_r \in \mathbb{N}$. And for any $t \in \mathbb{N}$, $2 \cdot (\max(l_1, l_2, \ldots, l_r))^2 + t \cdot l$ can be expressed as $\sum_{i=1}^{r}(y_i \cdot l_i)$ with $y_i \in \mathbb{N}$ by Lemma 1. Hence $2 \cdot (\max(l_1, l_2, \ldots, l_r))^2 + t \cdot l + s \in \mathcal{L}(E)$. Let $L = 2 \cdot (\max(l_1, l_2, \ldots, l_r))^2$. Then $L + t \cdot l + s \in \mathcal{L}(E)$. And from Lemma 10, we have $L = 2 \cdot (\max(l_1, l_2, \ldots, l_r))^2 \le 2 \cdot (2^{|E|})^2 \le 2^{2|E|+1}$. □

As before, we define the language $\mathcal{L}(\mathcal{S}(E)) = \bigcup_{\langle l,s\rangle \in \mathcal{S}(E)} \{k \cdot l + s | k \in \mathbb{N}\}$. From the definition of $\mathcal{S}$, we know that $\mathcal{S}(E)$ consists of finite many arithmetic progressions. From this we will construct an NFA in Chrobak normal form and then a DFA $\mathcal{B}$ such that $|\mathcal{B}| \le 2^{O(|E|)}$ and $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{S}(E))$ in next section. From above properties, we know that $\mathcal{L}(E) \subseteq \mathcal{L}(\mathcal{S}(E))$ and there exists $M \in \mathbb{N}$ such that $|\mathcal{L}(\mathcal{S}(E)) \setminus \mathcal{L}(E)| \le M$. Then we can construct a DFA for $\mathcal{L}(E)$ from $\mathcal{B}$ by changing finite many final states, which are not in the cycle, into non-final states. Moreover, we obtain the desired upper bound for the minimal DFA for $\mathcal{L}(E)$.

Before showing the construction of $\mathcal{B}$, we firstly establish a number $M \in \mathbb{N}$ such that $|\mathcal{L}(\mathcal{S}(E)) \setminus \mathcal{L}(E)| \le M$. To this end, we need to know bounds for numbers in $\mathcal{S}(E)$. Suppose $\langle l, s\rangle \in \mathcal{S}(E)$. The bound for s is also given by Lemma 7. The bound for l is computed by the following function.

Definition 6. *The function $\mathcal{I}(E)$ is defined as:*

$$\begin{aligned}
&\mathcal{I}(\varepsilon) = \{0\}\\
&\mathcal{I}(a) = \{0\}\\
&\mathcal{I}(E_1 + E_2) = \mathcal{I}(E_1) \cup \mathcal{I}(E_2)\\
&\mathcal{I}(E_1E_2) = \mathcal{I}(E_1) \cup \mathcal{I}(E_2)\\
&\mathcal{I}(E_1^{[m,n]}) = \mathcal{I}(E_1)\\
&\mathcal{I}(E_1^{[m,\infty]}) = \begin{cases} \{\mathcal{R}(E_1)\} & \text{if } \mathcal{I}(E_1) = \{0\},\\ \mathcal{I}(E_1) & \text{otherwise;}\end{cases}
\end{aligned}$$

The following properties are straightforward.

Observation 3. *Let E be a regular expression in R(#). We have $|\mathcal{I}(E)| \leq |E|$.*

Observation 4. *Let E be a regular expression in R(#). For any $l \in \mathcal{I}(E)$, we have $l \leq 2^{|E|}$.*

Lemma 12. *Let E be a regular expression in R(#). If $\mathcal{I}(E) = \{l_1, l_2, \ldots, l_n\}$, then $\prod_{i=1}^{n} l_i \leq 2^{|E|}$.*

We establish the relations between arithmetic progressions in $\mathcal{S}$ and the numbers in $\mathcal{I}$ as follows.

Lemma 13. *Let E be a regular expression in R(#). If for any $\langle l, s\rangle \in \mathcal{S}(E)$, we have $l = 0$, then $\mathcal{I}(E) = \{0\}$.*

Lemma 14. *Let E be a regular expression in R(#). For any $\langle l, s\rangle \in \mathcal{S}(E)$ ($l \neq 0$), there is an $l_1 \in \mathcal{I}(E)$ such that $l_1 \neq 0$ and $l | l_1$.*

Since we have the bounds for l and s, the bound for $|\mathcal{S}(E)|$ is straightforward.

Lemma 15. *Let E be a regular expression in R(#). We have $|\mathcal{S}(E)| \leq 2^{2|E|}$.*

Then from Proposition 5, we have the following property.

Corollary 2. *Let E be a regular expression in R(#). If $w \notin \mathcal{L}(E)$ and $w \in \mathcal{L}(\mathcal{S}(E))$, then $|w| \leq 2^{2|E|+2}$.*

5.3. The upper bound for the minimal DFA

According to the construction in [23], Proposition 5, and Corollary 2, we can obtain the upper bound for the size of the minimal DFA for an expression in R(#) as follows.

Theorem 6. *Let E be a regular expression in R(#). The number of states of the minimal DFA for $\mathcal{L}(E)$ is at most $2^{2|E|+4}$.*

Proof. From the construction in [23], we can construct a DFA $\mathcal{A}$ with the desired size as follows:

(1) Compute the set $\mathcal{S}(E)$.

(2) Construct an NFA $\mathcal{N}$ in Chrobak normal form such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{S}(E))$, and then use the method in Theorem 4.4 in [23] to construct an equivalent DFA $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{N})$.

(3) Copy the cycle to extend $\mathcal{D}$ such that the chain of the resulting DFA $\mathcal{B}$ has $2^{2|E|+2} + 1$ states, and $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{D})$. One additional state is required for the initial state. Suppose the initial state of $\mathcal{B}$ is q_0.

(4) Delete every word w such that $|w| \leq 2^{2|E|+2}$, $w \notin \mathcal{L}(E)$ and $w \in \mathcal{L}(\mathcal{S}(E))$. That is if $q = \delta(q_0, w)$, q is a final state of $\mathcal{B}$, and $w \notin \mathcal{L}(E)$, then we set q as a non-final state.

Using the bounds for l, s, and $|\mathcal{S}|$, we estimate the sizes of $\mathcal{N}$ and $\mathcal{D}$ in step (2) as follows. Since $\mathcal{N}$ is in Chrobak normal form, from the construction in [23], we have $|\mathcal{N}| \leq \mathcal{R}(E) + |\mathcal{S}| \cdot \max(\mathcal{I}(E)) + 1 \leq 2^{|E|} + 2^{2|E|} \cdot 2^{|E|} + 1 \leq 2^{3|E|+2}$. From [23], Corollary 2, Lemma 12, and Lemma 14, the size of $\mathcal{D}$ can be estimated as follows: $|\mathcal{D}| \leq \mathcal{R}(E) + \prod_{l_i \in \mathcal{I}(E)} l_i + 1 \leq 2^{|E|+1} + 1$.

Since the chain of $\mathcal{B}$ has $2^{2|E|+2} + 1$ states, we have $|\mathcal{B}| \leq 2^{2|E|+2} + 1 + |\mathcal{D}| \leq 2^{2|E|+4}$. Because step (4) does not affect the number of states of $\mathcal{B}$, we conclude that $|\mathcal{A}| = |\mathcal{B}| \leq 2^{2|E|+4}$.

From Proposition 4, $\mathcal{L}(E) \subseteq \mathcal{L}(\mathcal{B})$ holds. And from Corollary 2, we know that step (4) removes all word w such that $w \in (\mathcal{L}(\mathcal{B}) \setminus \mathcal{L}(E))$. Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(E)$.

□

We give an example to show the construction in Theorem 6. Let $E = a(a^{[4,4]})^{[0,\infty]} \cdot aa(a^{[6,6]})^{[0,\infty]} + (aa)^{[0,\infty]} \cdot aaaaaaa + (aaa)^{[0,\infty]} \cdot aaaa$.

(1) $\mathcal{S}(E) = \{\langle 2, 3\rangle, \langle 2, 7\rangle, \langle 3, 4\rangle\}$, and $|E| = 67$.

(2) We get an NFA $\mathcal{N}$ in Chrobak normal form in Figure 1, and a DFA $\mathcal{D}$ in Figure 2.

(3) We extend $\mathcal{D}$ such that the size of the chain of the resulting DFA is $2^{2|E|+2} + 1 = 2^{136} + 1$. Notice that $2^{136} \equiv 4 \pmod 6$. The resulting DFA is shown in Figure 3.

(4) We can check that $w = aaaaa$ ($|w| = 5$) is the only word such that $w \in \mathcal{L}(\mathcal{S}(E)) \setminus \mathcal{L}(E)$. Then we get the final DFA accepting $\mathcal{L}(E)$ in Figure 4.

Since there exists an expression E in R(#) such that every DFA accepting $\mathcal{L}(E)$ has at least $\Omega(2^{|E|})$ states [20, 21], the upper bound $2^{O(|E|)}$ is tight.

5.4. Checking determinism of languages of expressions in R(#)

For an expression E in R(#), since the minimal DFA of $\mathcal{L}(E)$ is of single-exponential size, we can non-deterministically check whether the cycle of the minimal DFA has at least two final states as follows.

Before showing the algorithm, we first consider the complexity of the following problem.

PROBLEM:	Member(E,w)
INPUT:	A regular expression E in R(#), and a binary number w.
	That is $w = b_0 b_1 \dots b_n$ and $(w)_{bin} = b_0 2^n + b_1 2^{n-1} + \dots + b_n$.
QUESTION:	$(w)_{bin} \in \mathcal{L}(E)$?

Following the proof of Lemma 5.1 in [32], we can obtain the complexity of Member(E,w) as follows.

Theorem 7 ([32])**.** Member(E,w) *is* **NP**-*complete.*

Using Lemma 5, Lemma 6, Theorem 6, and Theorem 7, we give the following straightforward algorithm to check whether $\mathcal{L}(E)$ is not deterministic.

(1) Guess three binary numbers w_1, w_2, and w_3 such that
$$2|E| + 4 \le |w_i| \le 2|E| + 5 \ (1 \le i \le 3).$$

(2) Check Member(E,w_1), Member(E,w_2), Member(E,$w_1 + w_3$), and Member(E,$w_2 + w_3$).

(3) If $(w_1)_{bin} \in \mathcal{L}(E)$, $(w_2)_{bin} \in \mathcal{L}(E)$, $(w_1 + w_3)_{bin} \in \mathcal{L}(E)$, and $(w_2 + w_3)_{bin} \notin \mathcal{L}(E)$, then the cycle of the minimal DFA will have at least two distinct final states, and $\mathcal{L}(E)$ is not deterministic.

Since Member(E,w) is **NP**-complete, we get the desired result as follows.

Theorem 8. *Given a regular expression E in R(#), the problem of deciding whether $\mathcal{L}(E)$ is deterministic is in $\mathbf{\Pi_2^p}$.*

Is it possible to prove this problem is $\mathbf{\Pi_2^p}$-hard by a reduction from the corresponding version of **EDCP**, where all numbers are binary numbers? We give a negative answer to this question.

The corresponding $\mathbf{EDCS^{II}}$ is defined almost the same as **EDCS**, excepted that its size is defined as $\sum_{i=1}^{n}(\log(l_i) + \log(s_i))$[6]. The $\mathbf{EDCP^{II}}$ is defined as follows: Does a given set $\mathcal{P}$ satisfying the above conditions form an $\mathbf{EDCS^{II}}$?

Theorem 9. $\mathbf{EDCP^{II}}$ *is* **coNP**-*complete.*

Proof. Since the lower bound follows from Theorem 2, we mainly deal with the upper bound[7]. At first, we need the following property.

[6] If this condition holds, for an arithmetic progression $\langle l, s\rangle$, l and s can be represented by binary numbers such that the input instance $\mathcal{P}$ has polynomial size. For example, the arithmetic progression $\langle 2, 5\rangle$ can be represented by $\langle 10, 101\rangle$, and its size is less than 5.

[7] This algorithm is provided by the reviewers of DLT 2013.

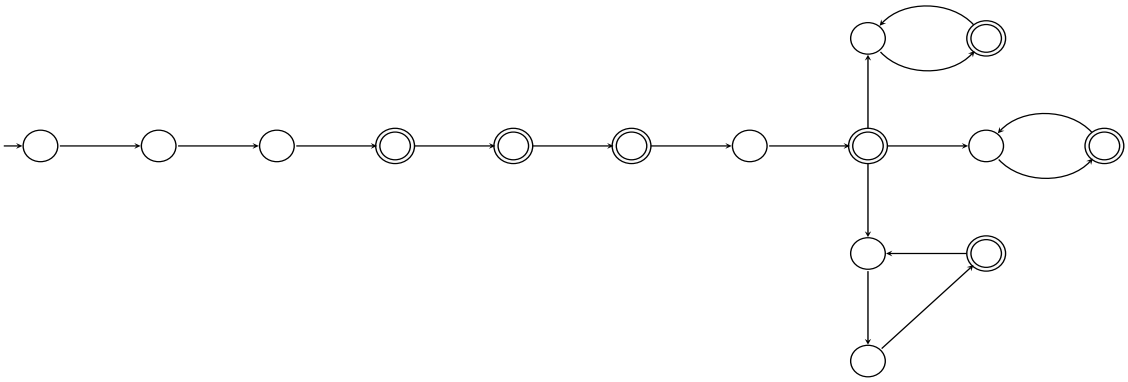

Figure 1: An NFA accepting $\mathcal{L}(\mathcal{S}(E))$ in Chrobak normal form.

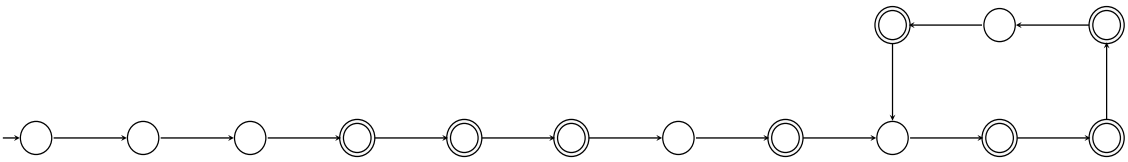

Figure 2: A DFA accepting $\mathcal{L}(\mathcal{S}(E))$.

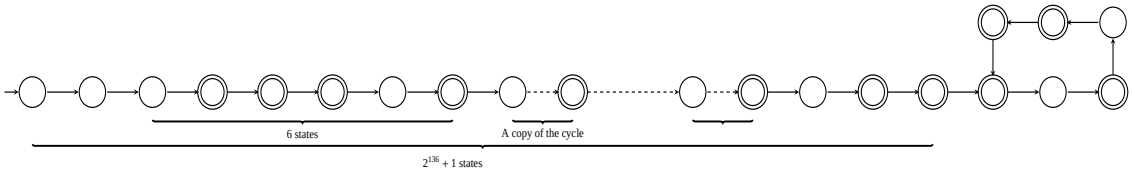

Figure 3: The extended DFA accepting $\mathcal{L}(\mathcal{S}(E))$.

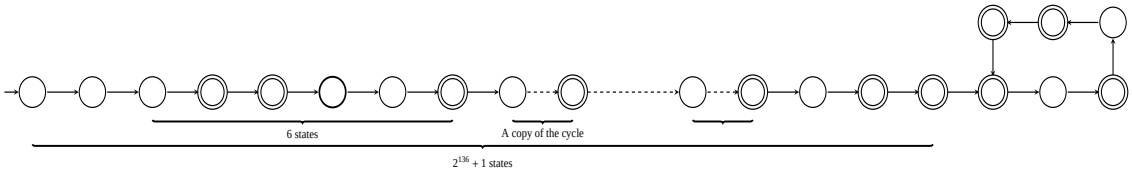

Figure 4: A DFA accepting $\mathcal{L}(E)$.

Claim 1. *Suppose $\mathcal{P}$ is an* $\mathbf{EDCS}^{\mathrm{II}}$ *and the answer to $\mathcal{P}$ is (y, x). Then* $y = \gcd(l_1, l_2, \ldots, l_n, s_1 - s_2, s_1 - s_3, \ldots, s_1 - s_n)$ *and* $x = (s_1 \bmod y)$.

Then checking whether $\mathcal{P}$ is not an $\mathbf{EDCS}^{\mathrm{II}}$ can be done as follows: (1) Compute $y = \gcd(l_1, l_2, \ldots, l_n, s_1 - s_2, s_1 - s_3, \ldots, s_1 - s_n)$ and $x = (s_1 \bmod y)$; (2) Guess an integer $z \in \mathbb{N}$ such that $z \leq \prod_{i=1}^{n} l_i$; (3) Check whether one of the following two conditions holds: (I) $z \equiv x (\mathrm{mod}\ y)$ and $z \not\equiv s_i (\mathrm{mod}\ l_i)$ for any i $(1 \leq i \leq n)$; (II) $z \not\equiv x (\mathrm{mod}\ y)$ and $z \equiv s_i (\mathrm{mod}\ l_i)$ for some i $(1 \leq i \leq n)$.

From Lemma 2 and Claim 1, we know that if $\mathcal{P}$ is an $\mathbf{EDCS}^{\mathrm{II}}$, then (y, x) is the answer. We only need to check whether (y, x) is the answer. From the definition of $\mathbf{EDCS}^{\mathrm{II}}$, we know that (y, x) is not the answer iff there is an integer $z \in \mathbb{N}$ such that condition (I) or condition (II) holds. From the Chinese Remainder Theorem [34], we know that there is such an integer z satisfying $z \leq \prod_{i=1}^{n} l_i$.

It is easy to see that computations of y and x take polynomial time. Since $\log(\prod_{i=1}^{n} l_i) = \sum_{i=1}^{n} \log(l_i) \leq p(n)$, we have $z \leq 2^{p(n)}$. Moreover, we can check whether the equations $z \equiv x (\mathrm{mod}\ y)$, $z \equiv s_1 (\mathrm{mod}\ l_1)$, …, and $z \equiv s_n (\mathrm{mod}\ l_n)$ hold in polynomial time. Then the whole checking algorithm takes polynomial time.

Hence whether $\mathcal{P}$ is not an $\mathbf{EDCS}^{\mathrm{II}}$ can be checked in non-deterministic polynomial time, and checking whether $\mathcal{P}$ is an $\mathbf{EDCS}^{\mathrm{II}}$ is in **coNP**. Therefore $\mathbf{EDCP}^{\mathrm{II}}$ is **coNP**-complete. □

6. Conclusion and future work

In this paper, we show a derivation method to derive a set of arithmetic progressions from a regular expression. There is a close relation between these arithmetic progressions and the language of the expression. Using this relation, we investigate the complexity of deciding determinism of regular languages over a unary alphabet. And we conclude that the problem, whether a regular language defined by a standard regular expression of any alphabet size can be defined by a deterministic expression with counting, is **coNP**-hard. Moreover, by extending the derivation method, we show an upper bound for the size of the minimal DFA of the language of an expression over a unary alphabet with counting. Then we show that checking determinism of the languages of regular expressions over a unary alphabet with counting is in $\mathbf{\Pi}_2^{\mathbf{p}}$. For the general case, this problem has been shown to be **EXPSPACE**-complete [16].

There is one problem remained: the lower bound for the problem of checking determinism of the languages of regular expressions over a unary alphabet in R(#). Theorem 4 shows that this problem is **coNP**-hard. But it is unlikely that this problem is in **coNP**, since the membership problem for standard expressions takes polynomial time, while the problem for expressions in R(#) is **NP**-complete. As noted in [40], the lower bound for checking determinism of the languages of regular expressions can be obtained by a reduction from the universality problem for regular expressions. However, the exact complexity of the universality problem for regular expressions over a unary alphabet in R(#) is unknown.

Acknowledgments. We thank Wim Martens for sending us the full version of the paper [14], and Zhilin Wu for carefully reading an early version of this paper and providing many helpful suggestions. We also thank the anonymous reviewers for providing useful comments to improve the paper.

References

[1] S. Amer-Yahia, Y. Kotidis, A web-services architecture for efficient XML data exchange, in: Data Engineering, 2004. Proceedings. 20th International Conference on, IEEE, 2004, pp. 523–534.
[2] R. Bourret, XML and Databases, `http://www.rpbourret.com/xml/XMLAndDatabases.html` (2000).
[3] S. Gao, C. M. Sperberg-McQueen, H. S. Thompson, N. Mendelsohn, D. Beech, M. Maloney, W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, `http://www.w3.org/TR/xmlschema11-1/` (2012).
[4] G. J. Bex, F. Neven, J. Van den Bussche, DTDs versus XML Schema: a practical study, in: Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004, ACM, 2004, pp. 79–84.
[5] W. Martens, F. Neven, T. Schwentick, G. J. Bex, Expressiveness and complexity of XML Schema, ACM Transactions on Database Systems (TODS) 31 (3) (2006) 770–813.
[6] W. W. W. Consortium, `http://www.w3.org/wiki/UniqueParticleAttribution`.
[7] E. van der Vlist, XML Schema, O'Reilly Media, Inc., 2002.

[8] G. J. Bex, W. Gelade, W. Martens, F. Neven, Simplifying XML Schema: effortless handling of nondeterministic regular expressions, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM, 2009, pp. 731–744.
[9] A. Brüggemann-Klein, D. Wood, One-unambiguous regular languages, Information and Computation 140 (2) (1998) 229–253.
[10] A. Brüggemann-Klein, Regular expressions into finite automata, Theoretical Computer Science 120 (2) (1993) 197–213.
[11] B. Groz, S. Maneth, S. Staworko, Deterministic regular expressions in linear time, in: Proceedings of the 31st symposium on Principles of Database Systems, ACM, 2012, pp. 49–60.
[12] P. Kilpeläinen, Checking determinism of XML Schema content models in optimal time, Information Systems 36 (3) (2011) 596–617.
[13] H. Chen, P. Lu, Assisting the design of XML Schema: diagnosing nondeterministic content models, in: Web Technologies and Applications, Springer, 2011, pp. 301–312.
[14] K. Losemann, W. Martens, M. Niewerth, Descriptional complexity of deterministic regular expressions, in: Mathematical Foundations of Computer Science 2012, Springer, 2012, pp. 643–654.
[15] H. Chen, P. Lu, Checking determinism of regular expressions with counting, Information and Computation 241 (2015) 302–320.
[16] W. Czerwiński, C. David, K. Losemann, W. Martens, Deciding definability by deterministic regular expressions, in: Foundations of Software Science and Computation Structures, Springer, 2013, pp. 289–304.
[17] P. Lu, J. Bremer, H. Chen, Deciding determinism of regular languages, Theory of Computing Systems 57 (1) (2015) 97–139.
[18] M. Latte, M. Niewerth, Definability by weakly deterministic regular expressions with counters is decidable, in: Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I, 2015, pp. 369–381.
[19] W. Gelade, M. Gyssens, W. Martens, Regular expressions with counting: weak versus strong determinism, SIAM Journal on Computing 41 (1) (2012) 160–190.
[20] P. Kilpeläinen, R. Tuhkanen, Regular expressions with numerical occurrence indicators-preliminary results, in: SPLST, 2003, pp. 163–173.
[21] W. Gelade, Succinctness of regular expressions with interleaving, intersection and counting, Theoretical Computer Science 411 (31) (2010) 2987–2998.
[22] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to automata theory, languages, and computation, Pearson, 2007.
[23] M. Chrobak, Finite automata and unary languages, Theoretical Computer Science 47 (1986) 149–158.
[24] E. Rich, Automata, computability and complexity: theory and applications, Pearson Prentice Hall Upper Saddle River, 2008.
[25] S. Ginsburg, H. G. Rice, Two families of languages related to ALGOL, Journal of the ACM (JACM) 9 (3) (1962) 350–371.
[26] R. L. Graham, D. E. Knuth, O. Patashnik, Concrete Mathematics: A Foundation for Computer Science, 1994.
[27] A. Brauer, On a problem of partitions, American Journal of Mathematics (1942) 299–312.
[28] G. Pighizzini, J. Shallit, M.-w. Wang, Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds, Journal of Computer and System Sciences 65 (2) (2002) 393–414.
[29] K. Bickel, M. Firrisa, J. Ortiz, K. Pueschel, Constructions of Coverings of the Integers: Exploring an Erdős problem, Summer Math Institute, Cornell University.
[30] P. Erdős, On integers of the form $2^k + p$ and some related problems, Summa Brasil. Math 2 (1950) 113–123.
[31] R. Guy, Unsolved problems in number theory, Vol. 1, Springer Science & Business Media, 2004.
[32] L. J. Stockmeyer, A. R. Meyer, Word problems requiring exponential time (Preliminary Report), in: Proceedings of the fifth annual ACM symposium on Theory of computing, ACM, 1973, pp. 1–9.
[33] R. G. Michael, S. J. David, Computers and intractability: a guide to the theory of NP-completeness, 1979.
[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al., Introduction to algorithms, Vol. 2, MIT press Cambridge, 2001.
[35] Z. Sawa, Efficient construction of semilinear representations of languages accepted by unary NFA, in: Reachability Problems, Springer, 2010, pp. 176–182.
[36] G. Schnitger, Regular expressions and NFAs without ε-transitions, in: STACS 2006, Springer, 2006, pp. 432–443.
[37] M. Holzer, M. Kutrib, The complexity of regular (-like) expressions, International Journal of Foundations of Computer Science 22 (07) (2011) 1533–1548.
[38] L. A. Steen, J. A. Seebach, L. A. Steen, Counterexamples in topology, Springer, 1978.
[39] Y.-S. Han, K. Salomaa, D. Wood, Prime decompositions of regular languages, in: Developments in Language Theory, Springer, 2006, pp. 145–155.
[40] B. Groz, XML security views: queries, updates and schemas, Ph.D. thesis, Lille 1 (2012).