The Complexity of SORE-definability Problems

Ping Lu[1], **Zhilin Wu**[2], **and Haiming Chen**[2]

1 **BDBC, Beihang University, Beijing 100191, China**
luping@buaa.edu.cn

2 **State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China**
{wuzl,chm}@ios.ac.cn

Abstract

Single occurrence regular expressions (SORE) are a special kind of deterministic regular expressions, which are extensively used in the schema languages DTD and XSD for XML documents. In this paper, with motivations from the simplification of XML schemas, we consider the *SORE-definability problem*: Given a regular expression, decide whether it has an equivalent SORE. We investigate extensively the complexity of the SORE-definability problem: We consider both (standard) regular expressions and regular expressions with counting, and distinguish between the alphabets of size at least two and unary alphabets. In all cases, we obtain tight complexity bounds. In addition, we consider another variant of this problem, the bounded SORE-definability problem, which is to decide, given a regular expression E and a number M (encoded in unary or binary), whether there is an SORE, which is equivalent to E on the set of words of length at most M. We show that in several cases, there is an exponential decrease in the complexity when switching from the SORE-definability problem to its bounded variant.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Single occurrence regular expressions, Definability, Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.22

1 Introduction

Background. Deterministic regular expressions are a special kind of regular expressions, which are mainly used in DTD and XML Schema [11]. Intuitively, deterministic regular expressions require that when reading from left to right a word in the language, each symbol in the word can directly match the symbol in the expression without knowing the next symbol or the length of the word [35, 1]. For example, $(a+b)a^*$ is deterministic. Since for each word w in $\mathcal{L}((a+b)a^*)$, if the first symbol in w is a, then it matches the first a in $(a+b)a^*$, and the other occurrences of a match the second one. On the other hand, $a^*(a+b)$ is not deterministic. Because given a word $w = aa$, without knowing the length of the word, we do not know whether the first a in w should match the first a in $a^*(a+b)$ or the second one.

Although deterministic regular expressions ensure the effective processing of XML documents [11, 16], it is not an easy task to design schema for XML documents with deterministic regular expressions. The major obstacle is that they are defined in a semantic manner, and do not have syntax rules to guide the design [1]. Considerable efforts have been made to solve this problem [2, 3, 4, 13, 23, 21, 7]. Among these is the introduction of single occurrence regular expressions (SORE) [2, 3, 4, 13]. SORE are regular expressions where each alphabet symbol appears at most once [2]. For example, $E = abc$ is an SORE, since there is only one of a, b, and c in E. While $E = aa^*b$ is not, because the symbol a appears twice in E.

Motivation. XML schemas defined with SORE are desirable, since they are very simple and intuitive to comprehend, and it is easy to check the conformance of XML documents with respect to them. As a result, though SORE constitute a restricted class of regular expressions, it turns out that deterministic regular expressions used in DTDs of XML documents from the practice are mostly SORE [29]. In view of this, XML schema designers may tempt to know whether the schema they proposed can in fact be changed into an equivalent, but simpler, schema defined with SORE. This brings the *SORE-definability problem*: *Given a regular expression, decide whether there is an SORE defining the same language.*

In this paper, we start an extensive investigation on the complexity of the SORE-definability problem: We consider both (standard) regular expressions and regular expressions with counting, and distinguish between the alphabets of size at least two and unary alphabets. In all cases, we obtain tight complexity bounds.

In addition, we consider another variant of the SORE-definability problem, the bounded SORE-definability problem, which is to decide, given a regular expression E (without or with counting) and a number M (encoded in unary or binary), whether there is an SORE, which is equivalent to E on the set of words of length at most M. The motivation of this problem comes from the observation that in practice, there may exist some additional information about the number of children that a node in XML documents can have, and this information can be utilised to ease the design of XML schemas. Given a node n, suppose that its content model (the format of its child elements) is defined by a regular expression E. Although $\mathcal{L}(E)$, the language defined by E, might be not deterministic (thus not in SORE), if there is a bound M on the number of children of n, then we may still construct a SORE E_1 such that E and E_1 are equivalent over the set of words up to length M. In this way, we can obtain a regular expression E_1 with a simpler structure to define its content model. Moreover, E_1 covers all the possible XML documents. Checking the existence of SORE E_1 such that $E =_{<=M} E_1$ is the bounded-definability problem. In addition, the bounded-definability problem is related to the bounded nonuniversality problem investigated in [10]. The bounded nonuniversality problem is to decide for a given nondeterministic finite-state automaton A and a number M encoded in unary, whether $\mathcal{L}(A) \cap \Sigma^{\leq M} \neq \Sigma^{\leq M}$. Note that this problem can be rephrased as $\mathcal{L}(A) \neq_{\leq M} \Sigma^*$, where $\Sigma = \{a_1, \ldots, a_n\}$ and Σ^* is the abbreviation of $(a_1 + \ldots + a_n)^*$, which is obviously an SORE.

Contributions. The results obtained in this work are summarised in Table 1. In the table, R stands for the set of (standard) regular expressions, and $R(\#)$ stands for the set of regular expressions with counting. We highlight some of them below.

(1) We first show that the SORE-definability problem is **PSPACE**-complete for regular expressions, and **EXPSPACE**-complete for regular expressions with counting.

(2) We then consider the special case of unary alphabets. We show that the SORE-definability problem becomes **coNP**-complete for regular expressions and $\mathbf{\Pi_2^p}$-complete for regular expressions with counting. Moreover, by using the same idea of the lower bound proof, we can show that the definability problem of deterministic regular expressions is $\mathbf{\Pi_2^p}$-complete for regular expressions with counting, which solves an open problem in [27].

(3) For the bounded SORE-definability problem, a bit surprisingly, we show that the complexity is **coNP**-complete for both regular expressions and regular expressions with counting, if the length bound M is encoded in unary. On the other hand, if M is encoded in binary, the complexity is **PSPACE**-complete for regular expressions and **coNEXPTIME**-complete for regular expressions with counting. In addition, when unary alphabets are considered and M is encoded in unary, the bounded SORE-definability problem can be solved in *polynomial time*, even for regular expressions with counting.

These results show that if the length bound M is encoded in unary, there is an exponential decrease when switching from the SORE-definability problem to its bounded variant.

The SORE-definability problem		
	$\|\Sigma\| = 1$	$\|\Sigma\| \geq 2$
R	**coNP**-c (Thm 6)	**PSPACE**-c (Thm 4)
$R(\#)$	$\mathbf{\Pi_2^p}$-c (Thm 7)	**EXPSPACE**-c (Thm 5)

The bounded SORE-definability problem			
R	M is unary	**PTIME** (Thm 15)	**coNP**-c (Cor 14)
	M is binary	**coNP**-c (Cor 19)	**PSPACE**-c (Cor 17)
$R(\#)$	M is unary	**PTIME** (Thm 15)	**coNP**-c (Thm 13)
	M is binary	$\mathbf{\Pi_2^p}$-c (Cor 18)	**coNEXPTIME**-c (Thm 16)

Table 1 The results of this paper: An overview

Related work. Single occurrence regular expressions were introduced in [2, 3, 4]. Most works on SORE focus on inferring an SORE from a set of sample words. The basic idea of these works is that given a set of sample words, we first construct a single occurrence automaton (SOA) A, then derive an SORE E from A. The main technical difficulty is how to construct E from A. To this end, Bex et al. [3, 4] developed an $O(|A|^5)$ algorithm. Recently, Freydenberger et al. [13] reduced the complexity of the construction to linear time.

There are also works investigating how to automatically construct deterministic regular expressions from (non-deterministic) regular expressions given by users. This problem entails the definability problem of deterministic regular expressions: Given a regular expression, decide whether there exists an equivalent deterministic regular expression. It was shown in [1, 12, 26] that this problem is **PSPACE**-complete. For regular expressions with counting, there are two notations of determinism, *i.e.,* weak and strong determinism (see [14]). The definability problem of weakly (resp. strongly) deterministic regular expressions with counting can also be defined similarly. It was shown in [24] that the definability problem of weakly deterministic regular expressions with counting is in **2-EXPSPACE** when the inputs are regular expressions, and in **3-EXPSPACE** when the inputs are regular expressions with counting, whereas the exact complexity of these problems are still open.

Researchers also investigated how to decide whether a given regular expression is deterministic [5, 6, 23, 21, 7, 16] (Note that this problem is different from the definability problem of deterministic regular expressions in the sense that we do not check the existence of an equivalent albeit potentially different deterministic regular expression). Remarkably, it was shown in [16] that the problem of whether a regular expression is deterministic can be decided in linear time. In addition, in [16], it was also shown that the problem of whether a regular expression with counting is weakly deterministic can be decided with the same complexity bound. On the other hand, the work [8] provided a linear time algorithm to decide whether a regular expression with counting is strongly deterministic. Moreover, in [18, 19], efficient matching algorithms were provided for strongly deterministic regular expressions with counting by using finite automata with counters.

Outline. This paper is structured as follows. Section 2 fixes some notations. Section 3 is devoted to the SORE-definability problem. The bounded SORE-definability problem is investigated in Section 4. We conclude this paper in Section 5.

2 Preliminaries

For a natural number $n \in \mathbb{N}$, let $[n]$ denote $\{1, \cdots, n\}$. In addition, for two natural numbers $m, n \in \mathbb{N}$ such that $m \leq n$, let $[m, n]$ denote the set $\{m, m+1, \cdots, n\}$.

An *alphabet* Σ is a finite set of symbols $\{a_1, a_2, \dots, a_n\}$. We will use $a, b, \cdots$ to denote symbols from Σ. A *word* over Σ is a sequence of symbols from Σ. We will use $u, v, w, \cdots$ to denote words and ε to denote the empty word. A *language* over Σ is a set of words on Σ. For two languages L_1 and L_2, we use $L_1 \cdot L_2$ to denote the language $\{uv \mid u \in L_1, v \in L_2\}$. In addition, for a language L, we define $L^0 = \{\varepsilon\}$, and $L^{n+1} = L \cdot L^n$ for each natural number n. We also use L^* to denote $\bigcup\limits_{n \in \mathbb{N}} L^n$. A (standard) *regular expression* over Σ is inductively defined as follows: ε and a are regular expressions for any $a \in \Sigma$; for any regular expressions E_1 and E_2, the disjunction $E_1 + E_2$, the concatenation $E_1 \cdot E_2$ (or $E_1 E_2$), and the star E_1^* are also regular expressions. The semantics of a regular expression E is given by a language $\mathcal{L}(E)$ defined as follows: $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(a) = \{a\}$, $\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$, $\mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2)$, and $\mathcal{L}(E_1^*) = (\mathcal{L}(E_1))^*$. Let R denote the set of all regular expressions. For a regular expression E, let Σ_E denote the set of all symbols from Σ that appear in E.

Next, we define deterministic regular expressions. Before that, we introduce some notations. Given a regular expression E, we replace every symbol a in E by a subscripted symbol a_i such that a_i appears only once. Let $\overline{E}$ denote the resulting expression obtained by this replacement. For instance, let $E = (\varepsilon + a) \cdot a^*$, then $\overline{E} = (\varepsilon + a_1) \cdot a_2^*$. A regular expression E is *deterministic* iff the following condition holds: for every two words $uxw, uyv \in \mathcal{L}(\overline{E})$, if $x = a_i$ and $y = a_j$, then $i = j$. A regular language L is deterministic, if there exists a deterministic regular expression E such that $\mathcal{L}(E) = L$. For $E = (\varepsilon + a) \cdot a^*$, consider two words $a_1 a_2, a_2 \in \mathcal{L}(\overline{E})$. Let $u = \varepsilon$, $x = a_1$, $w = a_2$, $y = a_2$, and $v = \varepsilon$, then $uxw, uyv \in \mathcal{L}(\overline{E})$, $x = a_1$, $y = a_2$, but $1 \neq 2$. By the definition, $(\varepsilon + a) \cdot a^*$ is not deterministic. But the language $\mathcal{L}((\varepsilon + a) \cdot a^*)$ is deterministic, since $\mathcal{L}((\varepsilon + a) \cdot a^*) = \mathcal{L}(a^*)$ and a^* is deterministic.

Next, we define the Glushkov automata of regular expressions [15, 30]. Given a regular expression E, we first define the following sets: $first(E) = \{a \mid aw_1 \in \mathcal{L}(E)\}$, $follow(E, a) = \{b \mid w_1 ab w_2 \in \mathcal{L}(E)\}$, and $last(E) = \{a \mid w_1 a \in \mathcal{L}(E)\}$. Intuitively, $first(E)$ comprises the first symbols of words in $\mathcal{L}(E)$, $follow(E, a)$ is the set of symbols, which can appear immediately after a in a word from $\mathcal{L}(E)$, and $last(E)$ contains the last symbols of words in $\mathcal{L}(E)$. Then the Glushkov automaton of E, denoted by $G_E = (Q_E \cup \{q_I\}, \Sigma, \delta_E, q_I, F_E)$, is constructed as follows:

1. Q_E is the set of symbols in $\overline{E}$, and q_I is the initial state;
2. For any $a \in \Sigma_E$, let $\delta_E(q_I, a) = \{a_i \mid a_i \in first(\overline{E})\}$;
3. For any $a, b \in \Sigma_E$, let $\delta_E(a_i, b) = \{b_j \mid b_j \in follow(\overline{E}, a_i)\}$;
4. $F_E = \begin{cases} \{a_i \mid a_i \in last(\overline{E})\} \cup \{q_I\} & \text{if } \varepsilon \in L(E), \\ \{a_i \mid a_i \in last(\overline{E})\} & \text{otherwise.} \end{cases}$

Given a regular expression E, the Glushkov automaton G_E can be constructed in polynomial time [6]. See Example 1 in the next section for an example of Glushkov automata.

Single occurrence regular expressions (SORE)[3, 4] are a special kind of deterministic regular expressions, which require that every symbol in the alphabet appears at most once. Moreover, SORE use the following operators: the disjunction ($+$), the concatenation ($\cdot$), the iteration ($^+$), and the optional ($?$), where the iteration E^+ and the optional $E?$ are the abbreviations of $E \cdot E^*$ and $\varepsilon + E$, respectively. Since $\mathcal{L}(E^*) = \mathcal{L}((E^+)?)$, SORE do not use the star operation. For example, a^+bc is an SORE, but aa^+ is not, since the symbol a appears twice. Additionally, we require that the iteration (resp. the optional) cannot be nested in an SORE, that is, the expressions of the form $(E?)?$ or $(E^+)^+$ are forbidden. We also forbid the expressions of the form $((E^+)?)^+$ or $((E?)^+)?$. These constraints ensure that for a fixed alphabet Σ, there are only a fixed number of SORE over Σ. Nevertheless, these constraints do not affect the expressive power of SORE. For an SORE, its Glushkov automaton can also be constructed in polynomial time [2].

A single occurrence automaton (SOA) $S = (Q, \Sigma, \delta, q_I, F)$ over an alphabet Σ is defined as follows [13]: $Q \subseteq \Sigma \cup \{q_I\}$, where q_I is the initial state; $\delta : Q \times \Sigma \to Q$ is the transition function such that whenever $\delta(q_1, b) = q_2$, we have $q_2 = b$; and $F \subseteq Q$ is the set of final states. Although SOA defined here are slightly different from those in [13], one can easily add a sink state to each SOA defined in this paper to obtain an SOA in [13]. Later on, we will ignore this difference and apply the algorithms in [13] directly on SOA in this paper.

A regular expression with counting is an extension of regular expressions, which additionally allows using the counting modalities $E^{[m,n]}$ or $E^{[m,\infty]}$. For a regular expression with counting E, the language $\mathcal{L}(E^{[m,n]}) = \bigcup_{i \in [m,n]} (\mathcal{L}(E))^i$. The language $\mathcal{L}(E^{[m,\infty]})$ can be defined similarly. Let $R(\#)$ denote the set of regular expressions with counting.

3 The SORE-definability problem

In this section, we study the complexity of the SORE-definability problem: Given a regular expression E, decide whether there exists an SORE E^c such that $\mathcal{L}(E^c) = \mathcal{L}(E)$. We first consider the general case of non-unary alphabets, then the special case of unary alphabets.

3.1 Non-unary alphabets

We start with regular expressions and consider regular expressions with counting later on[1].

To solve the SORE-definability problem, our main idea is to construct for a regular expression E, an SORE E^c such that $\mathcal{L}(E) = \mathcal{L}(E^c)$ iff there exists an SORE E_1 satisfying that $\mathcal{L}(E) = \mathcal{L}(E_1)$. With such an SORE E^c, we can solve the SORE-definability problem by checking whether $\mathcal{L}(E) = \mathcal{L}(E^c)$.

The construction of the SORE E^c is divided into two steps: We first construct an SOA S_E from E, then an SORE E^c from S_E.

Given a regular expression E, let $G_E = (Q_E \cup \{q_I\}, \Sigma, \delta_E, q_I, F_E)$ be the Glushkov automaton of E, we construct the SOA $S_E = (\Sigma_E \cup \{q_I\}, \Sigma_E, \delta'_E, q_I, F'_E)$ as follows:

1. For any $a \in \Sigma_E$, let $\delta'_E(q_I, a) = \{a \mid \exists i.\ a_i \in \delta_E(q_I, a)\}$;
2. For any $a, b \in \Sigma_E$, let $\delta'_E(a, b) = \{b \mid \exists i, j.\ b_j \in \delta_E(a_i, b)\}$;
3. $F'_E = \begin{cases} \{a \mid \exists i.\ a_i \in F_E\} \cup \{q_I\} & \text{if } q_I \in F_E, \\ \{a \mid \exists i.\ a_i \in F_E\} & \text{otherwise.} \end{cases}$

Intuitively, the SOA S_E is constructed from G_E by merging for each $a \in \Sigma_E$, all the states a_i of G_E into one state a and modifying the transition relation correspondingly.

▶ **Example 1.** Let $E = (a + b)^* a(a + c + d)^*$. Then $\overline{E} = (a_1 + b_2)^* a_3(a_4 + c_5 + d_6)^*$. The Glushkov automaton G_E and the SOA S_E constructed from G_E are given in Figure 1. Note that the states a_1, a_3, and a_4 in G_E are merged into one state a in S_E, and the other states remain the same (modulo names). ◀

The SOA S_E enjoys the following property.

▶ **Lemma 2.** *Given a regular expression E, $\mathcal{L}(E)$ can be defined by an SOA iff $\mathcal{L}(G_E) = \mathcal{L}(S_E)$.*

To construct the desired SORE E^c, we apply the algorithm REWRITE in [3, 4] or Soa2Sore in [13] to S_E, and get an SORE E^c enjoying the following property: S_E can be represented by an SORE iff $\mathcal{L}(S_E) = \mathcal{L}(E^c)$. From Lemma 2, we deduce the following fact.

[1] Several results in this section are based on Chapter 7 of the PhD. thesis of Ping Lu (cf. [25]).

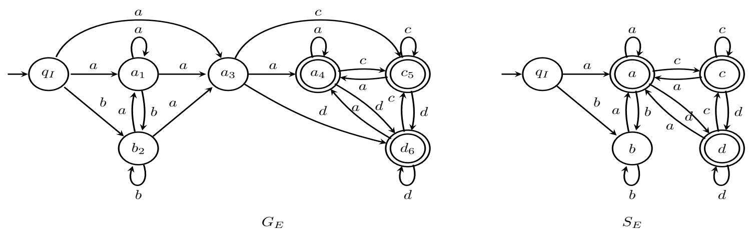

Figure 1 G_E and S_E.

▶ **Proposition 3.** $\mathcal{L}(E)$ can be defined by an SORE iff $\mathcal{L}(E) = \mathcal{L}(E^c)$.

The arguments for Proposition 3 proceed as follows: The "if" direction is trivial. For the "only if" direction, suppose that $\mathcal{L}(E)$ is defined by an SORE E_1. Then it can also be defined by an SOA, since the Glushkov automaton of E_1 is an SOA. By Lemma 2, we must have that $\mathcal{L}(E_1) = \mathcal{L}(E) = \mathcal{L}(G_E) = \mathcal{L}(S_E)$. Hence, S_E is represented by the SORE E_1. From the property of E^c, we know that $\mathcal{L}(S_E) = \mathcal{L}(E^c)$. We conclude that $\mathcal{L}(E) = \mathcal{L}(S_E) = \mathcal{L}(E^c)$.

Given a regular expression E, by using Proposition 3, we solve the SORE-definability problem of E as follows:

(1) Construct the SOA S_E;
(2) Construct the candidate SORE E^c;
(3) Check whether $\mathcal{L}(E) = \mathcal{L}(E^c)$. If so, return **true**; otherwise, return **false**.

The correctness of the algorithm follows from Proposition 3. In the following, we analyze the complexity of this algorithm. From the results in [6, 3, 4, 13], we know that steps (1) and (2) can be done in polynomial time. Since the equivalence problem of regular expressions is PSPACE-complete [31], we know that step (3) can be fulfilled in polynomial space. It follows that the SORE-definability problem of regular expressions is in **PSPACE**. For the lower bound, we apply a reduction from the complement of the acceptance problem of polynomial-space bounded Turing machines. Then we get the following result.

▶ **Theorem 4.** *The SORE-definability problem is* **PSPACE**-*complete for regular expressions.*

When E is a regular expression in $R(\#)$, *i.e.,* a regular expression with counting, we can expand E into a (standard) regular expression E', and then use the above algorithm to decide the SORE-definability problem. Since expanding E into E' takes exponential time, and the equivalence problem of regular expressions is **PSPACE**-complete [31], we conclude that the SORE-definability problem of $R(\#)$ is in **EXPSPACE**. For the **EXPSPACE** lower bound, we prove it by a reduction from the universality problem of regular expressions in $R(\#)$, which is known to be **EXPSPACE**-complete [31].

▶ **Theorem 5.** *The SORE-definability problem is* **EXPSPACE**-*complete for* $R(\#)$.

3.2 Unary alphabets

In this section, we mainly consider the complexity of the SORE-definability problem for unary alphabets. As a by-product, we also solve an open problem in [27].

Let $\Sigma = \{a\}$. One can verify that an SORE E_1 over Σ must satisfy one of the following constraints: $\mathcal{L}(E_1) = \mathcal{L}(\varepsilon)$, $\mathcal{L}(E_1) = \mathcal{L}(a)$, $\mathcal{L}(E_1) = \mathcal{L}(a?)$, $\mathcal{L}(E_1) = \mathcal{L}(a^+)$, or $\mathcal{L}(E_1) = \mathcal{L}((a^+)?)$. Then to check whether $\mathcal{L}(E)$ can be defined by an SORE, we only need to check whether $\mathcal{L}(E) = \mathcal{L}(\varepsilon)$, $\mathcal{L}(E) = \mathcal{L}(a)$, $\mathcal{L}(E) = \mathcal{L}(a?)$, $\mathcal{L}(E) = \mathcal{L}(a^+)$, or $\mathcal{L}(E) = \mathcal{L}((a^+)?)$. Since the equivalence problems of regular expressions and regular expressions with counting over a unary alphabet are **coNP**-complete [34] and $\mathbf{\Pi_2^p}$-complete [34, 20] respectively, we get the **coNP** and $\mathbf{\Pi_2^p}$ upper bounds respectively for their SORE-definability problems over a unary alphabet. Moreover, we show the corresponding lower bounds by a reduction from the universality problem of regular expressions over a unary alphabet and the connectedness problem of integer expressions respectively, which are known to be **coNP**-complete [34] and $\mathbf{\Pi_2^p}$-complete [36, 33] respectively. Therefore, we get the following results.

▶ **Theorem 6.** *Over a unary alphabet, the SORE-definability problem is* **coNP***-complete for regular expressions.*

▶ **Theorem 7.** *Over a unary alphabet, the SORE-definability problem is* $\mathbf{\Pi_2^p}$*-complete for* $R(\#)$*.*

The work in [27] showed that the definability problem of deterministic regular expressions over a unary alphabet is in $\mathbf{\Pi_2^p}$ for $R(\#)$, but left the lower bound open. By using a reduction similar to the proof of the lower bound in Theorem 7, we can solve the open problem and get the following result.

▶ **Theorem 8.** *Over a unary alphabet, the definability problem of deterministic regular expressions is* $\mathbf{\Pi_2^p}$*-complete for* $R(\#)$*.*

Although the lower bound in Theorem 7 can also be proved by using the construction in [9], that construction cannot be used to prove the lower bound in Theorem 8, since the language defined by the regular expression constructed in [9] is already deterministic.

4 The Bounded SORE-definability problem

In this section, we will study the complexity of the bounded SORE-definability problem: Given a regular expression E (without or with counting) and a number M, whether there exists an SORE E_1 such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E_1)$, *i.e.*, for every word w such that $|w| \leq M$, $w \in \mathcal{L}(E)$ iff $w \in L(E_1)$. As mentioned in the introduction, if the content model E in a DTD or XML Schema does not denote a deterministic regular language, if there is a bound M on the maximum number of children of nodes in XML documents, then we only need to give a deterministic content model E_1 to ensure that the language $\mathcal{L}(E_1)$ is equivalent to $\mathcal{L}(E)$ within the bound M.

We assume that in the bounded SORE-definability problem, the given regular expression (without or with counting) E and the number M satisfy that $M \geq 2 \cdot |\Sigma_E|$. This assumption is essential for the results in this section and it is open whether this assumption can be lifted.

Similar to the SORE-definability problem, the decision procedure for the bounded SORE-definability problem proceeds as follows:

1. At first, an SOA S_E is constructed from E such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(S_E)$ iff there exists an SOA A such that $\mathcal{L}(A) =_{\leq M} \mathcal{L}(E)$.
2. Then by using the algorithm Soa2Sore in [13], an SORE E^c is constructed from S_E such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$ iff there exists an SORE E' such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E')$.
3. Finally, decide whether $L(E) =_{\leq M} L(E^c)$.

In the following, we will first show how to construct S_E from E and E^c from S_E in Section 4.1, then based on these constructions, we derive the complexity results of the bounded SORE-definability problem in Section 4.2.

4.1 The construction of S_E and E^c

In this section, we assume that E is in $R(\#)$ and demonstrate how to construct an SOA S_E and an SORE E^c from E. The construction for regular expressions without counting is relatively easy and taken as a special case.

For the construction of S_E, one naive approach is to expand the expression E into a regular expression (without counting) E', and construct $S_{E'}$ from E'. But since the expansion of E incurs an exponential blow-up, we would not be able to achieve the tight complexity bounds (see, *e.g.,* Theorem 13 in Section 4.2). In the following, we show how we can circumvent the exponential blow-up and construct a desired SOA S_E in polynomial time.

▶ **Lemma 9.** *Given a regular expression E in $R(\#)$ and a number M encoded in binary, an SOA S_E satisfying the following constraint can be computed in polynomial time: $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E)$ iff there exists an SOA A such that $\mathcal{L}(A) =_{\leq M} \mathcal{L}(E)$.*

Before presenting the construction of S_E, we introduce some additional notations. Let us assume that $M > 0$ in the following. For an expression E in $R(\#)$ and a natural number M, we define $first_M(E) = \{a \in \Sigma_E \mid \exists w_1.\ aw_1 \in \mathcal{L}(E) \cap (\Sigma_E)^{\leq M}\}$, $follow_M(E) = \{(a,b) \in \Sigma_E \times \Sigma_E \mid \exists w_1, w_2.\ w_1abw_2 \in \mathcal{L}(E) \cap (\Sigma_E)^{\leq M}\}$, and $last_M(E) = \{a \in \Sigma_E \mid \exists w_1.\ w_1a \in \mathcal{L}(E) \cap (\Sigma_E)^{\leq M}\}$.

For an expression E in $R(\#)$ and a natural number M, we construct an SOA $S_E = (\Sigma_E \cup \{q_I\}, \Sigma_E, \delta'_E, q_I, F'_E)$ satisfying the following constraints:

- $\{a \in \Sigma_E \mid \delta'_E(q_I, a) = a\} = first_M(E)$;
- $\{(a,b) \in \Sigma_E \times \Sigma_E \mid \delta'_E(a,b) = b\} = follow_M(E)$;
- if $\varepsilon \in L(E)$, then $F'_E = last_M(E) \cup \{q_I\}$; otherwise, $F'_E = last_M(E)$.

Therefore, the construction of S_E is equivalent to the computations of $first_M(E)$, $follow_M(E)$ and $last_M(E)$, which, we will show, can be done in polynomial time.

For the computations of $first_M(E)$, $follow_M(E)$ and $last_M(E)$, for each subexpression E' of E, we will compute an *up-to-M counting abstraction* of E' over Σ_E, denoted by $\mathsf{Abs}_{\Sigma_E,M}(E') = (x, T, F, L)$, in polynomial time, such that

- $x \in \{\mathbf{true}, \mathbf{false}\}$ denotes whether $\varepsilon \in \mathcal{L}(E')$;
- T is a function from $\Sigma_E \times \Sigma_E$ to $[M] \cup \{\infty\}$ such that $T(a,b) = \min(\{|w| \mid w \in \mathcal{L}(E') \cap (\Sigma_E)^{\leq M}, w = w_1abw_2 \text{ for some } w_1, w_2\})$, where $\min(\emptyset) = \infty$ by convention;
- F is a function from Σ_E to $[M] \cup \{\infty\}$ such that $F(a) = \min(\{|w| \mid w \in \mathcal{L}(E') \cap (\Sigma_E)^{\leq M}, w = aw_1 \text{ for some } w_1\})$;
- L is a function from Σ_E to $[M] \cup \{\infty\}$ such that $L(a) = \min(\{|w| \mid w \in \mathcal{L}(E') \cap (\Sigma_E)^{\leq M}, w = w_1a \text{ for some } w_1\})$.

Moreover, given $\mathsf{Abs}_{\Sigma_E,M}(E') = (x, T, F, L)$, we define $N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'))$ as the minimum length of the words in $\mathcal{L}(E') \cap (\Sigma_E)^{\leq M}$, that is, if $x = \mathbf{true}$, then $N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E')) = 0$; otherwise, $N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E')) = \min(\mathsf{rng}(T) \cup \mathsf{rng}(F) \cup \mathsf{rng}(L))$, where $\mathsf{rng}(T)$ denotes the range of T, similarly for $\mathsf{rng}(F)$ and $\mathsf{rng}(L)$. It is assumed that $n < \infty$ for each $n \in [M]$.

Evidently, given $\mathsf{Abs}_{\Sigma_E,M}(E) = (x, T, F, L)$, we have that $first_M(E) = \{a \in \Sigma_E \mid F(a) \neq \infty\}$, $follow_M(E) = \{(a,b) \in \Sigma_E \times \Sigma_E \mid T(a,b) \neq \infty\}$ and $last_M(E) = \{a \in \Sigma_E \mid L(a) \neq \infty\}$.

Next, we show how to compute $\mathsf{Abs}_{\Sigma_E,M}(E')$ from E' by a structural induction on E'.

<u>(1) $E' = \varepsilon$.</u> In this case, $\mathsf{Abs}_{\Sigma_E,M}(E') = (\mathbf{true}, T_\infty, F_\infty, L_\infty)$, where T_∞ denotes the function where $T_\infty(a,b) = \infty$ for each $a, b \in \Sigma_E$.

<u>(2) $E' = a$ for any $a \in \Sigma_E$.</u> In this case, $\mathsf{Abs}_{\Sigma_E,M}(E') = (\mathbf{false}, T_\infty, a \to 1, a \to 1)$, where $a \to 1$ denotes the function that maps a to 1 and maps all the other symbols from Σ_E to ∞.

<u>(3) $E' = E'_1 + E'_2$.</u> In this case, suppose that $\mathsf{Abs}_{\Sigma_E,M}(E'_i) = (x_i, T_i, F_i, L_i)$ for $i = 1, 2$, then $\mathsf{Abs}_{\Sigma_E,M}(E') = (x, T, F, L)$, where

- $x = x_1 \vee x_2$ (x is the disjunction of x_1 and x_2);
- for each $(a,b) \in \Sigma_E \times \Sigma_E$, $T(a,b) = \min(\{T_1(a,b), T_2(a,b)\})$;
- for each $a \in \Sigma_E$, $F(a) = \min(\{F_1(a), F_2(a)\})$;

- for each $a \in \Sigma_E$, $L(a) = \min(\{L_1(a), L_2(a)\})$.

(4) $E' = E'_1 E'_2$ In this case, suppose that $\mathsf{Abs}_{\Sigma_E,M}(E'_i) = (x_i, T_i, E_i, L_i)$ for $i = 1, 2$, then $\mathsf{Abs}_{\Sigma_E,M}(E') = (x, T, F, L)$, where

- $x = x_1 \wedge x_2$ (x is the conjunction of x_1 and x_2);
- for each $(a, b) \in \Sigma_E \times \Sigma_E$, let

$$T(a,b) = \min\left([M] \bigcap \left(\begin{array}{l} \{T_1(a,b) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_2))\} \\ \cup \{T_2(a,b) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \\ \cup \{L_1(a) + F_2(b)\} \end{array}\right)\right),$$

  note that here we assume $\infty + \infty = \infty + n = n + \infty = \infty$ for every natural number n;
- for each $a \in \Sigma_E$, let

$$F(a) = \min\left([M] \bigcap \left(\{F_1(a) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_2))\} \cup \{F_2(a) \mid x_1 = \mathbf{true}\}\right)\right);$$

- for each $a \in \Sigma_E$, let

$$L(a) = \min\left([M] \bigcap \left(\{L_2(a) + N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \cup \{L_1(a) \mid x_2 = \mathbf{true}\}\right)\right).$$

(5) $E' = (E'_1)^{[m,n]}$ or $E' = (E'_1)^{[m,\infty]}$. Since the analysis of $E' = (E'_1)^{[m,\infty]}$ is almost the same as $E' = (E'_1)^{[m,n]}$, we only show the analysis of $E' = (E'_1)^{[m,n]}$. Let $\mathsf{Abs}_{\Sigma_E,M}(E'_1) = (x_1, T_1, F_1, L_1)$. Then $\mathsf{Abs}_{\Sigma_E,M}(E') = (x, T, F, L)$, where

- if $m \geq 2$, then
  - $x = x_1$,
  - for each $(a, b) \in \Sigma_E \times \Sigma_E$, let

$$T(a,b) = \min\left([M] \bigcap \left(\begin{array}{l} \{T_1(a,b) + (m-1)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \cup \\ \{L_1(a) + F_1(b) + (m-2)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\} \end{array}\right)\right),$$

    note that here we assume that $0 \times \infty = 0$ and $n \times \infty = \infty$ for each $n > 0$,
  - for each $a \in \Sigma_E$, let

$$F(a) = \min\left([M] \bigcap \{F_1(a) + (m-1)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\}\right),$$

  - for each $a \in \Sigma_E$, let

$$L(a) = \min\left([M] \bigcap \{L_1(a) + (m-1)N_{\min}(\mathsf{Abs}_{\Sigma_E,M}(E'_1))\}\right);$$

- if $m = 1$ and $n = 1$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(E'_1)$;
- if $m = 1$ and $n \geq 2$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(E'_1 + (E'_1)^{[m+1,n]})$, which can be computed from $\mathsf{Abs}_{\Sigma_E,M}(E'_1)$ and $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m+1,n]})$ by the aforementioned construction for the $+$ operator (note that $(E'_1)^{[m+1,n]}$ satisfies that $m + 1 \geq 2$);
- if $m = 0$ and $n = 0$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(\varepsilon)$;
- if $m = 0$ and $n \geq 1$, then $\mathsf{Abs}_{\Sigma_E,M}((E'_1)^{[m,n]}) = \mathsf{Abs}_{\Sigma_E,M}(\varepsilon + (E'_1)^{[m+1,n]})$ (note that $(E'_1)^{[m+1,n]}$ satisfies that $m + 1 \geq 1$).

The above computation of $\mathsf{Abs}_{\Sigma_E,M}(E)$ can be done in polynomial time, since each $\mathsf{Abs}_{\Sigma_E,M}(E')$ occupies only polynomial space and the computation takes at most $O(|E|)$ steps.

▶ **Example 10.** We will use the following example to show all the constructions. Let $E = ((a + b) \cdot (c + d) \cdot (\varepsilon + (ae)^{[5,5]}))^{[2,\infty]}$ and $M = 9$. The computation of $\mathsf{Abs}_{\Sigma_E,M}(E')$ is shown in Table 2, where $\mathbf{T}$ stands for **true**, $\mathbf{F}$ stands for **false**, and the pairs (a, b) such that $T(a, b) = \infty$ are omitted, similarly for F and L. Consider the computation of $\mathsf{Abs}_{\Sigma_E,M}((ae)^{[5,5]})$. It is easy to verify that $\mathsf{Abs}_{\Sigma_E,M}(ae) = (\mathbf{false}, \{(a, e) \to 2\}, a \to 2, e \to 2)$. Since $M = 9$, and ae is the shortest word in $\mathcal{L}(ae)$, we have that $2 + (m - 1) \times 2 = 2 + 4 \times 2 = 10 > M$. Therefore, $\mathsf{Abs}_{\Sigma_E,M}((ae)^{[5,5]}) = (\mathbf{false}, T_\infty, T_\infty, T_\infty)$, which means that any word in $\mathcal{L}((ae)^{[5,5]})$ cannot be a sub-word of w in $\mathcal{L}(E)$ such that $|w| \leq M$. Let $\mathsf{Abs}_{\Sigma_E,M}(E) = (x, T, F, L)$. Then $follow_M(E) = \{(a', b') \in \Sigma_E \times \Sigma_E \mid T(a', b') \neq$

E'	$\mathsf{Abs}_{\Sigma_E,M}(E')$	E'	$\mathsf{Abs}_{\Sigma_E,M}(E')$
a	$(\mathbf{F}, T_\infty, a \to 1, a \to 1)$	$c+d$	$(\mathbf{F}, T_\infty, \left\{\begin{smallmatrix} c\to 1 \\ d\to 1 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 1 \\ d\to 1 \end{smallmatrix}\right\})$
b	$(\mathbf{F}, T_\infty, b \to 1, b \to 1)$	ae	$(\mathbf{F}, \{(a,e)\to 2\}, a\to 2, e\to 2)$
c	$(\mathbf{F}, T_\infty, c \to 1, c \to 1)$	$(a+b)(c+d)$	$(\mathbf{F}, \left\{\begin{smallmatrix} (a,c)\to 2 \\ (b,c)\to 2 \\ (a,d)\to 2 \\ (b,d)\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 2 \\ b\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 2 \\ d\to 2 \end{smallmatrix}\right\})$
d	$(\mathbf{F}, T_\infty, d \to 1, d \to 1)$	$(ae)^{[5,5]}$	$(\mathbf{F}, T_\infty, T_\infty, T_\infty)$
e	$(\mathbf{F}, T_\infty, e \to 1, e \to 1)$	$\varepsilon + (ae)^{[5,5]}$	$(\mathbf{T}, T_\infty, T_\infty, T_\infty)$
$a+b$	$(\mathbf{F}, T_\infty, \left\{\begin{smallmatrix} a\to 1 \\ b\to 1 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 1 \\ b\to 1 \end{smallmatrix}\right\})$	$(a+b)(c+d)(\varepsilon+(ae)^{[5,5]})$	$(\mathbf{F}, \left\{\begin{smallmatrix} (a,c)\to 2 \\ (b,c)\to 2 \\ (a,d)\to 2 \\ (b,d)\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 2 \\ b\to 2 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 2 \\ d\to 2 \end{smallmatrix}\right\})$
$((a+b)(c+d)(\varepsilon+(ae)^{[5,5]}))^{[2,\infty]}$		$(\mathbf{F}, \left\{\begin{smallmatrix} (a,c)\to 4 & (c,a)\to 4 \\ (b,c)\to 4 & (d,a)\to 4 \\ (a,d)\to 4 & (c,b)\to 4 \\ (b,d)\to 4 & (d,b)\to 4 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} a\to 4 \\ b\to 4 \end{smallmatrix}\right\}, \left\{\begin{smallmatrix} c\to 4 \\ d\to 4 \end{smallmatrix}\right\})$	

Table 2 The computation of $\mathsf{Abs}_{\Sigma_E,M}(E')$

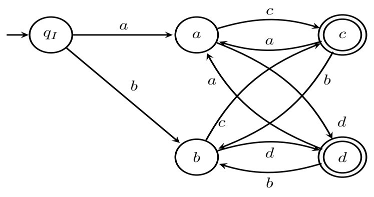

Figure 2 The SOA S_E

$\infty\} = \{(a,c),(b,c),(a,d),(b,d),(c,a),(d,a),(c,b),(d,b)\}$. Similarly, $first_M(E) = \{a' \in \Sigma_E \mid F(a') \neq \infty\} = \{a,b\}$, and $last_M(E) = \{a' \in \Sigma_E \mid L(a') \neq \infty\} = \{c,d\}$. From the sets $first_M$, $follow_M$, and $last_M$, we construct an SOA S_E illustrated in Figure 2.

By using S_E, we can construct the candidate SORE E^c for E in Example 12. ◀

Computation of E^c. Again, we use the algorithm Soa2Sore [13] mentioned in Section 3 to compute an SORE E^c from S_E in polynomial time. As a result of the assumption that $M \geq 2 \cdot |\Sigma_E|$, the SORE E^c enjoys the following property.

▶ **Proposition 11.** $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$ iff there exists an SORE E' such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E')$.

The arguments for Proposition 11 proceed as follows: The "only if" direction is trivial. For the "if" direction, suppose that there exists an SORE E' such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E')$. Then from the fact that $\mathcal{L}(E')$ can be defined by an SOA, according to Lemma 9, we know that $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E)$. Therefore, we have $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E')$. From the assumption $M \geq 2 \cdot |\Sigma_E|$, we can further show that $\mathcal{L}(S_E) = \mathcal{L}(E')$. Moreover, in [13], it was proved that the SORE E^c satisfies the "SORE-descriptive" property, *i.e.* there does not exist an SORE E'' such that $\mathcal{L}(S_E) \subseteq \mathcal{L}(E'') \subset \mathcal{L}(E^c)$. Therefore, we must have $\mathcal{L}(S_E) = \mathcal{L}(E^c)$. From $\mathcal{L}(S_E) =_{\leq M} \mathcal{L}(E)$, we conclude that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$.

▶ **Example 12.** Let us continue Example 10. By using the algorithm Soa2Sore in [13], from the SOA S_E in Figure 2, we obtain the following SORE: $((a+b)(c+d))^+$. It is easy to verify that $\mathcal{L}(E) =_{\leq 9} \mathcal{L}(((a+b)(c+d))^+)$. Then E can be represented by an SORE within the bound $M = 9$. On the other hand, by the results in Section 3, we can check that $\mathcal{L}(E)$ cannot be defined by an SORE. ◀

4.2 The Complexity

In this section, we establish the complexity results of the bounded SORE-definability problem. Given a regular expression E in $R(\#)$ and a number M, by using Proposition 11, we can develop the following algorithm to decide the bounded SORE-definability problem:

(1) Compute the set $\mathsf{Abs}_{\Sigma_E,M}(E)$;
(2) Construct $first_M(E)$, $follow_M(E)$, $last_M(E)$, and the candidate SORE E^c;
(3) Check whether $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$. If so, return **true**; otherwise, return **false**.

The correctness of this algorithm follows from Proposition 11. In the following, we analyse the complexity of the algorithm. From the computations of $\mathsf{Abs}_{\Sigma_E,M}(E)$, $first_M$, $follow_M$, $last_M$, and E^c in Section 4.1, we know that steps (1) and (2) can be done in polynomial time. For step (3), we distinguish between the unary and binary encoding of M.

M is encoded in unary Since M is encoded in unary, to check whether $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$, we first guess a word w such that $|w| \leq M$, then check whether $w \in (\mathcal{L}(E) \setminus \mathcal{L}(E^c))$ or $w \in (\mathcal{L}(E^c) \setminus \mathcal{L}(E))$. Since checking whether $w \in \mathcal{L}(E)$ and $w \in \mathcal{L}(E^c)$ is in **PTIME** [22], we deduce that the bounded SORE-definability problem for $R(\#)$ is in **coNP**. By a reduction from the bounded universality problem, which is known to be **coNP**-complete [10], we also show that the bounded SORE-definability problem for $R(\#)$ is **coNP**-hard.

▶ **Theorem 13.** *The bounded SORE-definability problem is* **coNP***-complete for* $R(\#)$ *and natural numbers* M *encoded in unary.*

One may wonder whether the bounded SORE-definability problem would be easier to solve, if E is a regular expression. The answer to this question is negative, since the expression constructed in the lower-bound proof of Theorem 13 is already a regular expression.

▶ **Corollary 14.** *The bounded SORE-definability problem is* **coNP***-complete for regular expressions and natural numbers* M *encoded in unary.*

If $|\Sigma_E| = 1$, then $(\Sigma_E)^{\leq M}$ contains at most $M + 1$ words. Therefore, in this case, when M is encoded in unary, step (3) can be done in **PTIME** and we have the following result.

▶ **Theorem 15.** *The bounded SORE-definability problem is in* **PTIME** *for unary regular expressions in* $R(\#)$ *and natural numbers* M *encoded in unary.*

From the aforementioned results and the ones in Section 3, we can see that if M is encoded in unary, then there is an exponential decrease in the complexity when switching from the SORE-definability problem to its bounded variant. For instance, for $R(\#)$, the SORE-definability problem is **EXPSPACE**-complete, while the bounded SORE-definability problem is **coNP**-complete if M is encoded in unary.

M is encoded in binary As mentioned above, $\mathsf{Abs}_{\Sigma_E,M}(E)$ can be computed in polynomial time even if M is encoded in binary. Nevertheless, since M is encoded in binary, the complexity of step (3) may increase exponentially.

Similar to the algorithm for Theorem 13, step (3) can also be done by guessing a word w such that $|w| \leq M$. Since M is encoded in binary, w can be exponentially large. Then checking whether $w \in \mathcal{L}(E)$ and $w \in \mathcal{L}(E^c)$ can be done in exponential time and we get a **coNEXPTIME** upper bound for bounded SORE-definability problem problem of $R(\#)$. For the **coNEXPTIME** lower bound, we get a reduction from the complement of the acceptance problem of nondeterministic exponential-time Turing machines. Therefore, we obtain the following result.

▶ **Theorem 16.** *The bounded SORE-definability problem is* **coNEXPTIME***-complete for* $R(\#)$ *and natural numbers* M *encoded in binary.*

Next, we consider the case when E is a regular expression. Step (1) and (2) can still be done in polynomial time. To check whether $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E^c)$, we can construct two

NFA from E and E^c respectively, guess a word w such that $|w| \leq M$, and check whether $w \in \mathcal{L}(E) \setminus \mathcal{L}(E^c)$ or $w \in \mathcal{L}(E^c) \setminus \mathcal{L}(E)$. The nondeterministic algorithm uses polynomial space. By Savitch's Theorem [32], we conclude that the bounded SORE-definability problem for regular expressions is in PSPACE. For the lower bound, we can directly use the same reduction in Theorem 4, and let $M = 2^{|E|}$. The correctness follows from the following arguments: Since the number of states of the minimum DFA for E is at most M [17], it is easy to check that there exists an SORE E_1 such that $\mathcal{L}(E) =_{\leq M} \mathcal{L}(E_1)$ iff there exists an SORE E_2 such that $\mathcal{L}(E) = \mathcal{L}(E_2)$.

▶ **Corollary 17.** *The bounded SORE-definability problem is* **PSPACE***-complete for regular expressions and natural numbers M encoded in binary.*

It is interesting to observe that the complexities of the SORE-definability problem and its bounded variant are the same for regular expressions, while the complexities of the two problems are different for $R(\#)$. This distinction is attributed to the following facts: (1) given a regular expression E, $\mathcal{L}(E) = \Sigma_E^*$, if and only if, $\mathcal{L}(E) =_{\leq 2^{|E|}} \Sigma_E^*$; while (2) given $E \in R(\#)$, $\mathcal{L}(E) = \Sigma_E^*$, if and only if, $\mathcal{L}(E) =_{\leq 2^{2^{|E|}}} \Sigma_E^*$ [31]. That is, to decide whether $\mathcal{L}(E) = \Sigma_E^*$ for $E \in R(\#)$, we have to check double-exponentially many words in $\mathcal{L}(E)$, while for regular expressions, we only need to check exponentially many words. So the bounded SORE-definability problem is simpler than the SORE-definability problem for $R(\#)$.

Given a regular expression E in $R(\#)$ over a unary alphabet, since the minimum DFA for E has at most $M = 2^{2 \cdot |E|+4} + 1$ states [27], we have the following results.

▶ **Corollary 18.** *Over a unary alphabet, the bounded SORE-definability problem is* $\mathbf{\Pi_2^p}$*-complete for $R(\#)$ and natural numbers M encoded in binary.*

▶ **Corollary 19.** *Over a unary alphabet, the bounded SORE-definability problem is* **coNP***-complete for regular expressions and natural numbers M encoded in binary.*

5 Conclusion

In this paper, we study the complexity of the SORE-definability problem as well as its bounded variant. The results of the paper were summarised in Table 1. As a by-product of the results obtained in this paper, we also solved an open problem in [27] and showed that over a unary alphabet, the definability problem of deterministic regular expressions is $\mathbf{\Pi_2^p}$-complete for regular expressions with counting.

There are several directions for the future work. An obvious question left open in this paper is whether the assumption $M \geq 2 \cdot |\Sigma_E|$ for the bounded SORE-definability problem can be lifted. Without this assumption, given an SOA A, it is unclear whether it is still in **PTIME** to decide whether there exists an SORE E_1 such that $\mathcal{L}(E_1) =_{\leq M} \mathcal{L}(A)$. Another interesting question is to investigate the definability problem for single occurrence regular expressions with counting (SORE(#)). The main technical challenge is how to obtain a candidate SORE(#). Similarly, one can also consider the CHARE-definability problem (see [3, 28] for the definition of CHARE).

Acknowledgement

Haiming Chen and Ping Lu are supported by the NSFC grant under No. 61472405. Ping Lu is also partially funded by Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University. Zhilin Wu is supported by the the NSFC grants under No. 61472474 and 61572478.

References

1 Geert Jan Bex, Wouter Gelade, Wim Martens, and Frank Neven. Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In *SIGMOD*, pages 731–744. ACM, 2009.

2 Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Transactions on the Web*, 4(4):14, 2010.

3 Geert Jan Bex, Frank Neven, Thomas Schwentick, and Karl Tuyls. Inference of concise DTDs from XML data. In *VLDB*, pages 115–126, 2006.

4 Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems*, 35(2):11, 2010.

5 Anne Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.

6 Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.

7 Haiming Chen and Ping Lu. Assisting the design of XML Schema: diagnosing nondeterministic content models. In *Web Technologies and Applications*, pages 301–312. Springer, 2011.

8 Haiming Chen and Ping Lu. Checking determinism of regular expressions with counting. *Information and Computation*, 241:302–320, 2015.

9 Dmitry Chistikov and Rupak Majumdar. Unary pushdown automata and straight-line programs. In *ICALP*, pages 146–157. Springer, 2014.

10 Sang Cho and Dung T Huynh. The parallel complexity of finite-state automata problems. *Information and Computation*, 97(1):1–22, 1992.

11 World Wide Web Consortium. http://www.w3.org/wiki/UniqueParticleAttribution.

12 Wojciech Czerwiński, Claire David, Katja Losemann, and Wim Martens. Deciding definability by deterministic regular expressions. In *FOSSACS*, pages 289–304, 2013.

13 Dominik D Freydenberger and Timo Kötzing. Fast learning of restricted regular expressions and DTDs. *Theory of Computing Systems*, 57(4):1114–1158, 2015.

14 Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM Journal on Computing*, 41(1):160–190, 2012.

15 Victor Mikhaylovich Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1, 1961.

16 Benoît Groz, Sebastian Maneth, and Sławek Staworko. Deterministic regular expressions in linear time. In *PODS*, pages 49–60. ACM, 2012.

17 John E Hopcroft and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation, first edition.* Pearson, 1979.

18 Dag Hovland. Regular expressions with numerical constraints and automata with counters. In *ICTAC*, pages 231–245, 2009.

19 Dag Hovland. The membership problem for regular expressions with unordered concatenation and numerical constraints. In *LATA*, pages 313–324, 2012.

20 Dung T Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^p-complete. *Theoretical Computer Science*, 33(2-3):305–326, 1984.

21 Pekka Kilpeläinen. Checking determinism of XML Schema content models in optimal time. *Information Systems*, 36(3):596–617, 2011.

22 Pekka Kilpeläinen and Rauno Tuhkanen. Regular Expressions with Numerical Occurrence Indicators-preliminary results. In *SPLST*, pages 163–173, 2003.

23 Pekka Kilpeläinen and Rauno Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Information and Computation*, 205(6):890–916, 2007.

24 Markus Latte and Matthias Niewerth. Definability by Weakly Deterministic Regular Expressions with Counters is Decidable. In *MFCS*, pages 369–381, 2015.

25 Ping Lu. *Research on deterministic regular languages (in Chinese)*. PhD thesis, University of Chinese Academy of Sciences, May 2014.

26 Ping Lu, Joachim Bremer, and Haiming Chen. Deciding Determinism of Regular Languages. *Theory of Computing Systems*, 57(1):97–139, 2015.

27 Ping Lu, Feifei Peng, Haiming Chen, and Lixiao Zheng. Deciding determinism of unary languages. *Information and Computation*, 245:181–196, 2015.

28 Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

29 Wim Martens, Frank Neven, Thomas Schwentick, and Geert Jan Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

30 Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 1(EC-9):39–47, 1960.

31 Albert R Meyer and Larry J Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT (FOCS)*, pages 125–129, 1972.

32 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

33 Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

34 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

35 Eric van der Vlist. *XML Schema*. O'Reilly Media, Inc., 2002.

36 Klaus W Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.