Deciding definability by deterministic regular expressions

Wojciech Czerwiński, Claire David, Katja Losemann, Wim Martens

Deciding Definability by Deterministic Regular Expressions[☆,☆☆]

Wojciech Czerwiński[c], Claire David[b], Katja Losemann[a,*], Wim Martens[a]

[a] *Universität Bayreuth*
[b] *Université Paris-Est Marne-la-Vallée*
[c] *University of Warsaw*

Abstract

We investigate the complexity of deciding whether a given regular language can be expressed by a deterministic regular expression. Our main technical result shows that deciding if the language of a given regular expression (or, non-deterministic finite automaton) can be defined by a deterministic regular expression is PSPACE-complete. The problem becomes EXPSPACE-complete if the input language is represented as a regular expression with counters and NL-hard if the input language is given by a minimal deterministic finite automaton.

Keywords: regular expressions, determinism, definability, complexity

1. Introduction

Schema information is highly advantageous when managing and exchanging XML data. Primarily, schema information is crucial for automatic error detection in the data itself (which is called validation, see, e.g., [2, 3, 4, 5]) or in the procedures that transform the data [6, 7, 8]. Furthermore, schemas provide information for optimization of XML querying and processing [9, 10], they are inevitable when integrating data through schema matching [11], and they provide users with a high-level overview of the structure of the data. From a software development point of view, schemas are very useful to precisely specify pre- and post-conditions of software routines that process XML data.

In their core, XML schemas specify the structure of well-formed XML documents through a set of constraints which are very similar to extended context-free grammar productions. Such schema are usually abstracted as a set of rules of the form

$$\mathit{Type} \to \mathit{Content}$$

[☆]This work was supported by grant number MA 4938/2-1 of the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).
[☆☆]This is the full version of Czerwiński et al. [1].
*Corresponding author

where *Content* is a regular expression that defines the allowed content inside the element type specified in the left-hand side. As such, regular expressions are a central component of schema languages for XML.

The two most prevalent schema languages for XML data, *Document Type Definition (DTD)* [2] and *XML Schema Definition (XSD)* [12], both developed by the World Wide Web Consortium, do not allow arbitrary regular expressions to define *Content*. Instead, they require these expressions to be *deterministic*. We refer to such deterministic regular expressions as *DREs*. In order to get a good understanding of schema languages for XML, it is thus important to develop a good understanding on DREs. Furthermore, since the concept of determinism in regular expressions is rather foundational, we believe our results to be relevant in a larger scope as well.

Intuitively, a regular expression is deterministic if, when reading a word from left to right without looking ahead, it is always clear where in the expression the next symbol can be matched. For example, the expression $(a + b)^*b(a + b)$ is not deterministic, because if we read a word that starts with b, it is not clear whether this b should be matched in the expression if we do not know what the remainder of the word will be. As such, determinism in regular expressions is very similar to determinism in finite automata: Let N be the automaton where we consider each alphabet symbol in an expression as a state and where we consider transitions between positions in the expression that can be matched by successive symbols. Then the expression is deterministic if and only if N (which is known as the Glushkov automaton of the expression) is deterministic.

Deterministic regular expressions or DREs have therefore been a subject of research since their foundations were laid by Brüggemann-Klein and Wood [13, 14]. Their important contribution is a characterization of languages definable by DREs in terms of structural properties on the minimal DFA. In particular, this characterization showed that some regular languages cannot be defined with a DRE. One such language is defined by the expression $(a + b)^*b(a + b)$. Furthermore, Brüggemann-Klein and Wood showed that it is decidable whether a given regular language is definable by a DRE. Since then, DREs have been studied in the context of language approximations [15], learning [16], descriptional complexity [17, 18] and static analysis [19, 20]. Recently, it was shown that testing if a regular expression is deterministic can be done in linear time [21].

Determinism has also been studied for a more general class of regular expressions which allows a *counting operator* [22, 23, 24]. This operator allows to write the expression $a^{10,100}$ defining the language that contains words of length 10 to 100 and labeled with only a's. The motivation for the counting operator again comes from schema languages, because the operator can be used to define expressions in XML Schema. Determinism for expressions with counters seems to pose more challenges than without the counting operator. For example, already testing whether an expression with counters is deterministic is non-trivial [25].

Contributions. In this paper we study the following problem:

Given a regular expression, can it be determinized?

This rather fundamental question has first been studied about 20 years ago [13] but the precise complexity is still open, despite the rich body of research discussed above. The best known upper bound is from Brüggeman-Klein and Wood, who showed that the problem is in EXPTIME (by exhibiting an algorithm that works in polynomial time on the minimal DFA [14]) and the best known lower bound is PSPACE-hardness [15, 26]. The main result of this paper settles this question and proves that this problem is PSPACE-complete. Our proof is rather technical and provides deeper insights in the decision algorithm of Brüggemann-Klein and Wood. A central insight, which is a cornerstone of our proof, is that the recursion depth of the algorithm is only polynomial in the size of a smallest NFA for the given regular language.

Since regular expressions with counters are important in the context of W3C XML Schema, we also study the complexity of deciding if a given expression with counters can be written as a DRE. This problem turns out to be EXPSPACE-complete. We complement these completeness results by proving that it is NL-hard to decide if a given DFA can be written as a DRE. At the moment, it is not clear to us whether this lower bound can be improved. The problem is known to be in polynomial time by [14].

2. Definitions

For a finite set S, we denote its cardinality by $|S|$. By Σ we always denote an alphabet, i.e., a finite set of symbols. A (Σ-)*word* w over alphabet Σ is a finite sequence of symbols $a_1 \cdots a_n$, where $a_i \in \Sigma$ for each $i = 1, \ldots, n$. The set of all Σ-words is denoted by Σ^*. The *length* of a word $w = a_1 \cdots a_n$ is n and is denoted by $|w|$. The empty word is denoted by ε. A *language* is a set of words.

2.1. (Nondeterministic) Finite Automata

A *(nondeterministic) finite automaton* (or *NFA*) N is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. We sometimes denote that $q_2 \in \delta(q_1, a)$ as $q_1 \xrightarrow{a} q_2 \in \delta$ to emphasize that, when N is in state q_1, it can go to state q_2 when reading an a. A *run of N on word $w = a_1 \cdots a_n$* is a sequence $q_0 \cdots q_n$ where, for each $i = 1, \ldots, n$, we have $q_{i-1} \xrightarrow{a_i} q_i \in \delta$. Word w is *accepted* by N if there is such a run which is *accepting*, i.e., if $q_n \in F$. The *language of N*, also denoted $L(N)$, is the set of words accepted by N. By δ^* we denote the extension of δ to words, i.e., $\delta^*(q, w)$ is the set of states which can be reached from q by reading w. The *size* $|N|$ of an NFA is the total number of transitions, i.e., $\sum_{q,a} |\delta(q, a)|$. An NFA is *deterministic*, or a *DFA*, when every $\delta(q, a)$ has at most one element. Throughout the paper, we use the notation $\mathcal{P}_N$ for the power set automaton of N and $[N]$ for the minimal DFA for $L(N)$. It is well-known that $[N]$ is unique for N and that it can be obtained by merging states of $\mathcal{P}_N$ [27]. In this paper, we assume that all states of an automaton are *useful* unless mentioned otherwise, that is, every state appears in some accepting run. This implies that every state can be reached from the initial state and that,

from each state in an automaton, an accepting state can be reached. This also implies that we use minimal DFAs without sink state and that $\mathcal{P}_N$ by default only contains the useful subsets of states of N. We sometimes abuse notation and also denote by $\emptyset$ the minimal DFA with no states.

Furthermore, we often see an NFA as a *graph*, which is obtained by considering its states as nodes and its transitions as (labeled) directed edges. Then, we also refer to a connected sequence of transitions in N as a *path*.

2.2. Regular Expressions and Variants

We define the class of *regular expressions (RE)* over Σ as follows: ε and every Σ-symbol is a regular expression; and whenever r and s are regular expressions then so are $(r \cdot s)$, $(r + s)$, and $(s)^*$. In addition, we allow $\emptyset$ as a regular expression, but we do not allow $\emptyset$ to occur in any other regular expression. For readability, we usually omit concatenation operators and parentheses in examples. The *language* defined by an RE r, denoted by $L(r)$, is defined as usual. Whenever we say that expressions or automata are *equivalent*, we mean that they define the same language. The *size* $|r|$ of r is defined to be the total number of occurrences of alphabet symbols, epsilons, and operators, i.e., the number of nodes in its parse tree.

The *regular expressions with counters (RE(#))* extend REs with a *counting operator*. That is, each RE-expression is an RE(#)-expression. Furthermore, when r and s are RE(#)-expressions then so are $(r \cdot s)$, $(r + s)$, (r^*), and $r^{k,\ell}$ for $k \in \mathbb{N}$ and $\ell \in \mathbb{N}^+ \cup \{\infty\}$ with $k \leq \ell$. Here, $\mathbb{N}^+$ denotes $\mathbb{N} \setminus \{0\}$. For a language L, define $L^{k,\ell} = \bigcup_{i=k}^{\ell} L^i$ and $L(r^{k,\ell}) = \bigcup_{i=k}^{\ell} L(r)^i$. Thus, $L(r^*) = L(r^{0,\infty})$. The size of an expression in RE(#) is the number of nodes in its parse tree, plus the sizes of all numbers $k, \ell \in \mathbb{N}$, where a natural number k has size $\lceil \log k \rceil$ if $k > 0$ and size 1 if $k = 0$.

In the following, we introduce *determinism* for REs. *Deterministic regular expressions (DREs)* put a restriction on the class of REs. Let $\bar{r}$ stand for the RE obtained from r by replacing, for every integer i and alphabet symbol a, the i-th occurrence of a in r by a_i (counting occurrences from left to right). For example, for $r = b^*a(b^*a)^*$ we have $\bar{r} = b_1^*a_1(b_2^*a_2)^*$. A regular expression r is *deterministic* (or *one-unambiguous* [14] or a *DRE*) if there are no words wa_iv and wa_jv' in $L(\bar{r})$ such that $a \in \Sigma$ and $i \neq j$. The expression $(a + b)^*a$ is not deterministic since both words a_2 and a_1a_2 are in $L((a_1 + b_1)^*a_2)$. The equivalent expression $b^*a(b^*a)^*$ is deterministic. Brüggemann-Klein and Wood showed that not every regular expression is equivalent to a deterministic one and, therefore, that the set of DREs forms a strict subset of REs. We call a regular language *DRE-definable* if there exists a DRE that defines it. The canonical example for a language that is not DRE-definable is $(a + b)^*b(a + b)$ [14].

2.3. Problem of Interest

In this paper, we investigate variants of the following problem.

– DRE-Definability: Given a regular language L, is L DRE-definable?

We consider the problem for various representations of regular languages: regular expressions, regular expressions with counters, NFAs, and DFAs. Whenever we consider such a variation, we put the respective representation between braces. For example, DRE-Definability(RE) is the problem: Given a regular expression r, is $L(r)$ DRE-definable?

We study the complexity of DRE-definability for NFAs and REs in Section 3 and the complexity of DRE-definability for RE(#)s and minimal DFAs in Section 4.

3. DRE-Definability for REs and NFAs

DRE-Definability was first studied by Brüggemann-Klein and Wood who showed that the problem can be solved in polynomial time in the size of the minimal DFA of a language [14]. Their algorithm (henceforth referred to as the Bkw-Algorithm) is not at all trivial and gives good insight in DRE-definable regular languages. In the following we present this algorithm and show how one can implement the algorithm for NFAs in PSPACE.

3.1. The Bkw *Algorithm for DRE-Definability*

We recall the Bkw-Algorithm together with some definitions and known results. Moreover, we define *level automata* which we use to analyse the Bkw-Algorithm.

Orbits and gates. For a state q in an NFA N, the *orbit of* q, denoted $\mathcal{O}(q)$, is the maximal strongly connected component of N that contains q. We call q a *gate* of $\mathcal{O}(q)$ if q is accepting or q has an outgoing transition that leaves $\mathcal{O}(q)$. If an orbit consists only of one state q and q has no self-loops, we say that it is a *trivial* orbit. We say that a transition $q_1 \xrightarrow{a} q_2$ is an *inter-orbit* transition if q_1 and q_2 belong to different orbits. The *orbit automaton of state* q is the sub-automaton of N consisting of $\mathcal{O}(q)$ in which the initial state is q and the accepting states are the gates of $\mathcal{O}(q)$. We denote the orbit automaton of q by N_q. The *orbit language of* q is $L(N_q)$. The *orbit languages of* N are the orbit languages of states of N.

Orbit property. An NFA N has the *orbit property* if, for every pair of gates q_1 and q_2 in the same orbit in N, the following properties hold:

1) q_1 is accepting if and only if q_2 is accepting; and,
2) for all states q outside the orbit of q_1 and q_2, there is a transition $q_1 \xrightarrow{a} q$ if and only if there is a transition $q_2 \xrightarrow{a} q$.

Consistent symbols. A symbol $a \in \Sigma$ is *N-consistent* if there is a state $f(a)$, such that every accepting state q of N has a transition $q \xrightarrow{a} f(a)$. We refer to the corresponding transitions as *consistent transitions* of N. A set $S \subseteq \Sigma$ is *N-consistent* if every symbol in S is N-consistent. Whenever we consider N-consistent sets S in the remainder of the paper we assume that they are maximal, i.e., there does not exist an $a \in \Sigma$ that is not in S and is N-consistent. Henceforth, we always refer to *the* N-consistent set. For the set S of N-consistent symbols, the *S-cut* of N, denoted N_S, is obtained by removing all consistent transitions from N. Using these notions, Brüggemann-Klein and Wood give the following characterization of the class of DRE-definable languages.

Theorem 1 (Brüggemann-Klein and Wood [14]). *Let D be a minimal DFA and S the set of D-consistent symbols. Then the following are equivalent:*

1) *$L(D)$ is DRE-definable;*
2) *D has the orbit property and all orbit languages of D are DRE-definable;*
3) *D_S has the orbit property and all orbit languages of D_S are DRE-definable.*

Furthermore, if D consists of a single, nontrivial orbit and $L(D)$ is DRE-definable, then there is at least one D-consistent symbol.

Towards a polynomial time algorithm for DRE-Definability, they show the following result:

Lemma 2 (Brüggemann-Klein and Wood [14]). *Let D be a minimal DFA and S be the set of D-consistent symbols.*

(1) If D_S has the orbit property, then $(D_S)_q$ is minimal for each state q in D.

(2) If p and q are states in the same orbit of D_S, then $L((D_S)_p)$ is DRE-definable if and only if $L((D_S)_q)$ is DRE-definable.

Point 1 of the above lemma is immediate from combining Lemmas 5.9 and 5.10 from [14]. Point 2 is immediate from the fact that DRE-definable regular languages are closed under derivatives [14]. Notice that, in general, D_S does not have to be a minimal DFA. In particular, it can have states that are not reachable from the initial state. However, these results lead to a recursive test that decides whether the language of a minimal DFA is DRE-definable. We present this test in Algorithm 1. Notice that Lemma 2 ensures that we never have to effectively minimize the DFA that we give to the recursive call in line 11 of the algorithm.

In the remainder of this article, D always denotes a minimal DFA. We now investigate the recursion depth of Algorithm 1 and examine how, for a state q of D, the orbit of q evolves during the recursion. Therefore, we define *level automata* which exactly describe the structure of the orbit of some state q at a specific point in time of the algorithm. Remember that, in one iteration of Algorithm 1 we always delete two kinds of transitions, if they are present: the consistent transitions (which we delete to obtain D_S from D) and the inter-orbit transitions in D_S (which we delete to obtain $(D_S)_q$).

Algorithm 1 The Bkw-Algorithm [14].

```
   Algorithm Bkw
2: Input: Minimal DFA D = (Q, Σ, δ, q0, F)
   Output: true if L(D) is DRE-definable, else false
4: S ← the maximal set of D-consistent symbols
   if D has only one trivial orbit then return true
6: if D has precisely one orbit and S = ∅ then return false
   compute the orbits of D_S
8: if D_S does not have the orbit property then return false
   for each orbit O in D_S do
10:    choose a state q in O
       if not Bkw((D_S)_q) then return false
12: return true
```

Level automata. For a state q of a minimal DFA D and $k \in \mathbb{N}$ we inductively define the *level k automaton of D for the state q*, denoted $\mathrm{lev}_k(D, q)$, as follows:

- $\mathrm{lev}_0(D, q) = D$.
- Let S be the maximal set of consistent symbols in D. Then

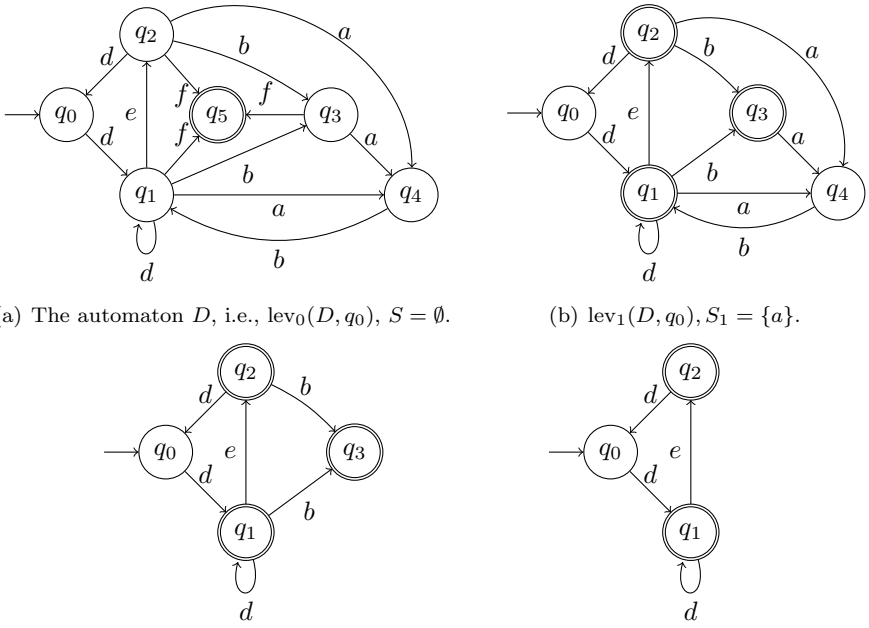
$$\mathrm{lev}_1(D, q) = \begin{cases} (D_S)_q & \text{if } D \text{ has more than one orbit and} \\ & \qquad D_S \text{ has the orbit property;} \\ (D_S)_q & \text{if } S \neq \emptyset \text{ and } D_S \text{ has the orbit property;} \\ \emptyset & \text{otherwise.} \end{cases}$$

- For $k > 1$, let $B := \mathrm{lev}_{k-1}(D, q)$ and S_{k-1} be the maximal set of consistent symbols in B. Then

$$\mathrm{lev}_k(D, q) = \begin{cases} (B_{S_{k-1}})_q & \text{if } S_{k-1} \neq \emptyset \text{ and } B_{S_{k-1}} \text{ fulfills the orbit property} \\ \emptyset & \text{otherwise.} \end{cases}$$

The above definition follows precisely the construction in Algorithm 1 if state q is chosen every time in line 10. The definition makes clear that the top level recursion of the Bkw-Algorithm (in which we construct $\mathrm{lev}_1(D, q)$) is slightly different from the others: the input DFA D of the top level can have multiple orbits, whereas this is not the case for deeper recursive levels. According to Lemma 2, $\mathrm{lev}_k(D, q)$ is always minimal.

Example 3. Figure 1 provides an example to illustrate the notion of level automata. Consider the minimal DFA D from Figure 1(a) and its state q_0. By definition, $\mathrm{lev}_0(D, q_0)$ is the automaton D itself. In order to build $\mathrm{lev}_1(D, q_0)$ (see Figure 1(b)), observe that D has two orbits and its set of consistent symbols S is empty since no transitions leave state q_5. Furthermore, D_S, which equals D,



(a) The automaton D, i.e., $\mathrm{lev}_0(D, q_0)$, $S = \emptyset$.

(b) $\mathrm{lev}_1(D, q_0), S_1 = \{a\}$.

(c) The S_1-cut of $\mathrm{lev}_1(D, q_0)$.

(d) $\mathrm{lev}_2(D, q_0), S_2 = \emptyset$.

Figure 1: An example of level automata for a minimal DFA D.

fulfills the orbit property since all transitions that leave $\mathcal{O}(q_0)$ go to state q_5. As such, $\mathrm{lev}_1(D, q_0)$ equals $(D_\emptyset)_{q_0}$, the orbit automaton of q_0 in D. We now explain how to obtain $\mathrm{lev}_2(D, q_0)$ (see Figure 1(d)). First notice that $S_1 = \{a\}$ is the maximal set of consistent symbols in $\mathrm{lev}_1(D, q_0)$. Furthermore, the S_1-cut of $\mathrm{lev}_1(D, q_0)$ (illustrated in Figure 1(c) without unreachable states) fulfills the orbit property. The automaton $\mathrm{lev}_2(D, q_0)$ is the orbit automaton of q_0 in the S_1-cut of $\mathrm{lev}_1(D, q_0)$ (that is, $\mathrm{lev}_2(D, q_0) = (\mathrm{lev}_1(D, q_0)_{S_1})_{q_0}$). Finally, observe that $\mathrm{lev}_2(D, q_0)$ has only one orbit and no consistent symbols which implies that $\mathrm{lev}_3(D, q_0) = \emptyset$. Also, in accordance with the Bkw-Algorithm, this means that $L(D)$ is not DRE-definable.

The following lemma summarizes the link between DRE-definability and level automata.

Lemma 4. *Let D be a minimal DFA. Then the following are equivalent:*

(1) $L(D)$ is DRE-definable;

(2) for every state q of D and $k \in \mathbb{N}$, $L(lev_k(D, q))$ is DRE-definable;

(3) for every state q of D and $k \in \mathbb{N}$, $L(lev_k(D, q))$ is DRE-definable and $lev_k(D, q)_{S_k}$ has the orbit-property.

PROOF. The implications (3) ⇒ (2) and (2) ⇒ (1) are trivial.

It remains to show that (1) ⇒ (3) holds. Assume that there exist a state q of D and $k \in \mathbb{N}$ such that $\mathrm{lev}_k(D,q)$ is not DRE-definable or $\mathrm{lev}_k(D,q)_{S_k}$ does not have the orbit-property, where S_k is the set of consistent symbols in $\mathrm{lev}_k(D,q)$. By (2) in Lemma 2, we know that the choice of a state in line 10 of the BKW-Algorithm is arbitrary. Therefore, consider a run of the BKW-Algorithm on D where, at each recursion level $i < k$, state q is chosen in line 10 when the current orbit $\mathcal{O}$ contains q. Then, at level k, the algorithm is called on $\mathrm{lev}_k(D,q)$ and returns false by definition of the BKW-Algorithm, i.e., $L(D)$ is not DRE-definable. □

3.2. The Recursion Depth of BKW

Now, we are ready to prove deeper properties of the BKW-Algorithm which are the basis of further results in the paper. First we observe that, once a state becomes a gate in the BKW-Algorithm, its outgoing transitions disappear in deeper recursion levels.

Lemma 5. *Let D be a minimal DFA and q be a gate in $\mathrm{lev}_k(D,q)$ for some $k > 0$. Then either $\mathrm{lev}_{k+1}(D,q) = \emptyset$ or q has strictly less outgoing transitions in $\mathrm{lev}_{k+1}(D,q)$. In the latter case, q is also a gate in $\mathrm{lev}_{k+1}(D,q)$.*

PROOF. Since $k > 0$, $\mathrm{lev}_k(D,q)$ has precisely one orbit. Therefore there exist no inter-orbit transitions in $\mathrm{lev}_k(D,q)$. Thus, since q is a gate, it is accepting. Assume that the set of consistent symbols S_k in $\mathrm{lev}_k(D,q)$ is not empty and that $\mathrm{lev}_k(D,q)_{S_k}$ fulfills the orbit property. By definition, $\mathrm{lev}_{k+1}(D,q) = (\mathrm{lev}_k(D,q)_{S_k})_q$ which is obtained from $\mathrm{lev}_k(D,q)$ by first removing transitions that are outgoing of an accepting state and labeled with a symbol in S_k and, secondly, taking the orbit automaton of q afterwards. Since q is accepting and $S_k \neq \emptyset$, it follows that q must have consistent transitions in $\mathrm{lev}_k(D,q)$ which are not in $\mathrm{lev}_k(D,q)_{S_k}$, and thus are not in $\mathrm{lev}_{k+1}(D,q)$. If $\mathrm{lev}_k(D,q)$ has no consistent symbols or $\mathrm{lev}_k(D,q)_{S_k}$ does not fulfill the orbit property, then $\mathrm{lev}_{k+1}(D,q) = \emptyset$ by definition. □

Since in a minimal DFA every state has at most $|\Sigma|$ outgoing transitions the following results holds.

Lemma 6. *Let D be a minimal DFA and q be a gate in $\mathrm{lev}_k(D,q)$. Let $n = |\Sigma| + 1$ if $k = 0$ and let $n = |\Sigma|$ if $k > 0$. Then either q is a trivial orbit in $\mathrm{lev}_{k+n}(D,q)$ or $\mathrm{lev}_{k+n}(D,q) = \emptyset$.*

PROOF. Choose n as defined above and assume that $\mathrm{lev}_{k+n}(D,q) \neq \emptyset$, as otherwise the lemma is shown. By definition, q exists and is a gate in $\mathrm{lev}_{k+\ell}(D,q)$ for every $0 \leq \ell \leq n$. Because q has at most $|\Sigma|$ outgoing transitions in $\mathrm{lev}_k(D,q)$, the assumption holds directly by Lemma 5. □

From Lemma 6, we can infer how long it takes for a state p to become a gate.

Lemma 7. *Let D be a minimal DFA and p be a state of $lev_k(D,p)$ for some $k \in \mathbb{N}$. Let ℓ be the length of the shortest path from p to a gate in $lev_k(D,p)$. Then either $lev_{k+|\Sigma|\cdot\ell+1}(D,p) = \emptyset$ or p is a gate in $lev_{k+|\Sigma|\cdot\ell+1}(D,p)$.*

Proof. Let ℓ_j be the length of the shortest path from p to a gate in $\mathrm{lev}_j(D,p)$. If $\ell_j = 0$, p is a gate. If $\ell_j > 1$, we show that within $|\Sigma|$ recursion levels ($|\Sigma|+1$ levels if $j = 0$), this value decreases strictly.

Let q be a gate, such that there is a path of length ℓ_j from p to q in $\mathrm{lev}_j(D,p)$ and let the corresponding path be:

$$p = p_0 \to p_1 \to \cdots \to p_{\ell_j - 1} \to p_{\ell_j} = q.$$

First notice that ℓ_j cannot increase as long as no transition on the path is removed. (If q is a gate at some level, it remains a gate at the next level). Assume now that some transition $p_i \to p_{i+1}$ is removed from $\mathrm{lev}_{j+m}(D,p)$ for some $m \in \mathbb{N}$. By definition of level automata, the transition is either consistent (out-going from an accepting state), or an inter-orbit transition. In both cases p_i is a gate in $\mathrm{lev}_{j+m}(D,p)$ and thus $\ell_{j+m} < \ell_j$. Therefore, ℓ_j cannot increase when considering ascending levels.

From Lemma 6, we know that after at most $n = |\Sigma|$ recursion levels ($|\Sigma|+1$ if we start from level $j = 0$) the orbit automaton $\mathrm{lev}_{j+n}(D,q)$ is either empty or trivial. But this means that the transition $p_{\ell_j - 1} \to q$ is an inter-orbit transition in $\mathrm{lev}_{j+n}(D,q)$, i.e., $p_{\ell_j - 1}$ is a gate. Thus $\ell_{j+n} < \ell_j$, which proves that ℓ_j strictly decreases within $|\Sigma|$ recursion levels ($|\Sigma|+1$ if $j = 0$). Altogether, after at most $j = \ell \cdot |\Sigma| + 1$ recursion levels, either $\mathrm{lev}_{k+j}(D,p)$ is empty or $\ell_j = 0$, which means p is a gate in $\mathrm{lev}_{k+j}(D,p)$. □

Next, we want to combine Lemma 7 with a simple observation about NFAs versus their minimal DFAs, namely that paths to accepting states are short.

Lemma 8. *Let N be an NFA with size n. Then for every state of $[N]$ there is a path leading to some accepting state of length at most $n-1$.*

Proof. We prove the assumption by examining the states in $\mathcal{P}_N$. Let $N = (Q, \Sigma, \delta, q_0, F)$ and $p = \{q_1, \ldots, q_k\}$ be a non-empty subset of Q (that is, a state in $\mathcal{P}_N$). Notice that, we consider only NFAs where all states are useful. Therefore, for every state $q \in Q$, there exists a path to some accepting state $q^f \in F$ of length at most $n-1$. In particular, this means that there is such a path from q_1 to q_1^f. Then, by definition of $\mathcal{P}_N$, there exists a path from p to some state p^f that contains q_1^f of length at most $n-1$ in $\mathcal{P}_N$. Clearly, p^f is accepting in $\mathcal{P}_N$. Let w be a word such that $\delta^*_{\mathcal{P}_N}(p,w) = p^f$ and $|w| = n-1$. We have $\delta^*_{[N]}([p],w) = [p^f]$, which concludes the proof. □

Combining Lemma 6, 7 and 8 we have an upper bound on how long states can be present in the recursion of the Bkw-Algorithm, compared to the size of an NFA for the language.

Lemma 9. *Let N be an NFA with size n. Then, $lev_{n\cdot|\Sigma|+2}([N], p) = \emptyset$ for every state p of $[N]$.*

Proof. Let p be a state in $[N]$. By Lemma 8 there exists a path from p to some accepting state q in $[N]$ of length at most $n-1$. Clearly, q is a gate in $[N]$. Thus the length of the shortest path from p to a gate in $[N]$ is smaller or equal to $n-1$. Furthermore, $[N] = \mathrm{lev}_0([N], p)$ by definition. Therefore, by Lemma 7, state p is a gate in $\mathrm{lev}_{(n-1)\cdot|\Sigma|+1}([N], p)$, or $\mathrm{lev}_{(n-1)\cdot|\Sigma|+1}([N], p)$ is empty. In the latter case the lemma is shown. If p is a gate in $\mathrm{lev}_{(n-1)\cdot|\Sigma|+1}([N], p)$ then, by Lemma 6, p is a trivial orbit in $\mathrm{lev}_{n\cdot|\Sigma|+1}([N], p)$. Thus it follows that $\mathrm{lev}_{n\cdot|\Sigma|+2}([N], p) = \emptyset$, which concludes the proof. □

Summarized, we know that the recursion depth of the Bkw-Algorithm is polynomial in the size of a minimal NFA for a language.

Theorem 10. *Let N be an NFA with size n. The recursion depth of Algorithm 1 on $[N]$ is at most $n \cdot |\Sigma| + 2$.*

3.3. Consistency Violations

In the following, we analyze the possible causes of failure for the Bkw-Algorithm. We identify three properties such that the Bkw-Algorithm fails if and only if one of them holds for some orbit automaton at some level k. Then, our PSPACE algorithm searches for one of these properties (resp. violations) in the input automaton.

From the Bkw-Algorithm, we can immediately see that there are two situations in which it can reject: (in line 6) at some point in the recursion, when the automaton consists only of one orbit which has no consistent symbols or (in line 8) at some point in the recursion, when the S-cut of the automaton does not have the orbit property. The latter means that there exist two gates of the same orbit in the S-cut, such that either they do not have the same transitions to the outside or one of them is accepting while the other one is not. We now formalize these different types of violations and prove afterwards that the Bkw-Algorithm fails if and only if one of these violations is found at some point in the recursion. Let D be a minimal DFA and S be its set of consistent symbols. Then D can have the following violations:

Out-Consistency Violation: There exist gates q_1 and q_2 in the same orbit $\mathcal{O}$ of D_S and there exists a state q outside $\mathcal{O}$ such that there is a transition $q_1 \xrightarrow{a} q$ and no transition $q_2 \xrightarrow{a} q$.

Acceptance Consistency Violation: There exist gates q_1 and q_2 in the same orbit of D_S such that q_1 is accepting and q_2 is not.

Orbit Consistency Violation. There exists an accepting state q_1 such that, for every symbol a, there exists another accepting state q_2 in $\mathcal{O}(q_1)$ in D, such that for every state q, at most one of the transitions $q_1 \xrightarrow{a} q$ and $q_2 \xrightarrow{a} q$ exists.[1]

[1] Notice that $\delta(q_1, a)$ may be empty if D only has useful states.

Notice that, similar to the Bkw-Algorithm the first two violations focus on D_S and the last one on D. Altogether we also say that a DFA D *has a violation* if and only if it has at least one of the above violations. We show that these violations are a valid characterization of DRE-definable languages.

Theorem 11. *Let D be a minimal DFA. Then it holds that $L(D)$ is not DRE-definable if and only if there exist a state q of D and $k \in \mathbb{N}$ such that $\mathrm{lev}_k(D, q)$ has a violation.*

Proof. We prove the direction from right to left first. Therefore, we distinguish three cases depending on the violation that occurs in $\mathrm{lev}_k(D, q)$.

If, for some state q of D and $k \in \mathbb{N}$, we have that $\mathrm{lev}_k(D, q)_{S_k}$ has an out-consistency violation or an acceptance consistency violation, then $\mathrm{lev}_k(D, q)_{S_k}$ does not fulfill the orbit property. By Lemma 4, this means that $L(D)$ is not DRE-definable.

If $\mathrm{lev}_k(D, q)$ has an orbit consistency violation, then $\mathrm{lev}_k(D, q)$ has no consistent symbol. If $k \geq 1$ then $\mathrm{lev}_k(D, q)$ has only one orbit. This implies that $\mathrm{lev}_k(D, q)$ is not DRE-definable by Theorem 1. By Lemma 4, the language $L(D)$ is not DRE-definable. If $k = 0$, then $\mathrm{lev}_1(D, q)$ is the orbit automaton of q in D. This automaton consists of a single orbit, whose states are the ones in the orbit of q in D. (Because D has no consistent symbols, it holds that $D_S = D$.) Furthermore, $\mathrm{lev}_1(D, q)$ still does not have a consistent symbol because all accepting states of the orbit of q in D are still accepting in $\mathrm{lev}_1(D, q)$. Therefore, again by Lemma 4, the language $L(D)$ is not DRE-definable.

It remains to prove the direction from left to right. Therefore, assume that $L(D)$ is not DRE-definable. By Lemma 4, there exist k and q, such that $L(\mathrm{lev}_k(D, q))$ is not DRE-definable. This means that the Bkw-Algorithm fails when given $\mathrm{lev}_k(D, q)$ as input. Assume it fails in line 6. Then $\mathrm{lev}_k(D, q)$ has no consistent symbol, which means that an orbit consistency violation occurs in $\mathrm{lev}_k(D, q)$. Assume it fails in line 8. Then $\mathrm{lev}_k(D, q)_{S_k}$ does not have the orbit property, which means that either an out-consistency violation occurs in $\mathrm{lev}_k(D, q)_{S_k}$, or an acceptance consistency violation occurs in $\mathrm{lev}_k(D, q)_{S_k}$. This concludes the proof. □

3.4. A PSPACE Algorithm for DRE-Definability

We are now ready to re-examine the complexity of DRE-Definability for NFAs and REs. Our PSPACE algorithm for DRE-definability for REs and NFAs exploits Theorem 11 in the following way. Given an NFA N, we search for a level k and a state p of $[N]$ such that $\mathrm{lev}_k([N], p)$ has a violation. As PSPACE is closed under complement, the result follows.
Notice that, in general $[N]$ can be exponentially larger than N and therefore we cannot simply compute $[N]$ in space polynomial in $|N|$. To overcome this difficulty, we use the following two ideas:

1) Use the fact that the maximal recursion depth of Algorithm 1 on $[N]$ is polynomial in the size of the NFA N (Theorem 10).

2) Adapt Algorithm 1 using Theorem 11 and apply it on the minimal DFA by only partially constructing it on-the-fly from the NFA.

In the following we explain how we can detect if there occurs a violation in the minimal DFA for some NFA on the fly, i.e., without constructing the DFA explicitly. To this end, we fix the following notations for the remainder of the section. By $N = (Q_N, \Sigma, \delta_N, q_N^0, F_N)$ we always denote an NFA. So, in particular, we always denote by Q_N the state set of N. For a set of states $q \subseteq Q_N$, we denote by $[q]$ the corresponding state in the minimal DFA $[N]$. More formally, $[q]$ is the set of words $\{w \mid \exists t \in q \text{ s.t. } \delta_N^*(t, w) \cap F_N \neq \emptyset\}$, i.e., the Myhill-Nerode class of q. Also, whenever we talk about $\mathrm{lev}_k([N], [q])_{S_k}$, the set S_k is the set of consistent symbols in $\mathrm{lev}_k([N], [q])$. The key result (Lemma 23) is to show that we can detect if a violation occurs in a level k for $[N]$ in space polynomial in k and $|N|$. Here are the precise problems we consider. For each of them the input is an NFA N and $k \in \mathbb{N}$:

Out-Cons-Violation:	Given N and k, is there a $q \subseteq Q_N$ such that $\mathrm{lev}_k([N], [q])_{S_k}$ has an out-consistency violation?
Acc-Cons-Violation:	Given N and k, is there a $q \subseteq Q_N$ such that $\mathrm{lev}_k([N], [q])_{S_k}$ has an acceptance consistency violation?
Orbit-Cons-Violation:	Given N and k, is there a $q \subseteq Q_N$ such that $\mathrm{lev}_k([N], [q])$ has an orbit consistency violation?

We first study the complexity of the following subproblems which we use later to solve the above problems (see e.g. Lemma 23). The input is always a subset of an NFA N, non-empty sets $p, q \subseteq Q_N$, $a \in \Sigma$, $k \in \mathbb{N}$ that is relevant to the problem.

Edge:	Given (N, p, q, a, k), is $[p] \xrightarrow{a} [q]$ a transition in $\mathrm{lev}_k([N], [p])$?
Reachability:	Given (N, p, q, k), is $[q]$ reachable from $[p]$ in $\mathrm{lev}_k([N], [p])$?
SameOrbit:	Given (N, p, q, k), are $[p]$ and $[q]$ in the same orbit of $\mathrm{lev}_k([N], [p])$?
InterOrbit:	Given (N, p, k), is there an inter-orbit transition $[p] \xrightarrow{a} [q]$ for some label a and q in $\mathrm{lev}_k([N], [p])$?
Acceptance:	Given (N, p, k), is $[p]$ accepting in $\mathrm{lev}_k([N], [p])$?
IsGate:	Given (N, p, k), is $[p]$ a gate in $\mathrm{lev}_k([N], [p])$?

Notice that SameOrbit and InterOrbit are only non-trivial if $k = 0$. Furthermore, for some of the above problems X we consider a variation called X-Cut in which, with the same input, we want to decide if the problem X is true for automaton $\mathrm{lev}_k([N], [p])_{S_k}$ (instead of $\mathrm{lev}_k([N], [p])$).
We will heavily use the following result:

Theorem 12 (Corollary to Savitch's Theorem). *Let $f(n) \geq \log n$ be a non-decreasing polynomial function. Then NSPACE($f(n)$) $\subseteq$ SPACE($f^2(n)$).*

Our proof is a careful mutual induction on the above defined problems. First we show that EDGE, EDGE-CUT, and ACCEPTANCE can be computed in polynomial space on level 0 and then we prove a set of implications of the sort *if we can solve X on level k, then we can solve Y on level k (or level $k+1$).* All the lemmas have to be carefully put together in the right order.

Lemma 13. *Given N and $p, q \subseteq Q_N$, we can test whether $[p] = [q]$ in space $O(|N|^2)$.*

PROOF. We describe how a non-deterministic Turing Machine can test in non-deterministic space $O(|N|)$ whether $[p] \neq [q]$. The statement then follows from Theorem 12 and the Immerman-Szelepscényi Theorem.

In this proof, we consider the power set automaton of N *with* its useless states. We do this to be able to elegantly state a procedure that is correct even if p or q are not states of $\mathcal{P}_N$. To this end, let $\mathcal{P}'_N = (Q_{\mathcal{P}'_N}, \Sigma, \delta_{\mathcal{P}'_N}, \{q_N^0\}, Q_{\mathcal{P}'_N})$ be the power set automaton of N with useless states. That is, $Q_{\mathcal{P}'_N}$ is the power set of Q_N and $\delta_{\mathcal{P}'_N}(q, a) = \{s \mid \exists t \in q \text{ s.t. } s \in \delta_N(t, a)\}$.

The non-deterministic Turing Machine checks if there exists a Σ-word w such that exactly one of the two states $\delta^*_{\mathcal{P}'_N}(p, w)$ and $\delta^*_{\mathcal{P}'_N}(q, w)$ is accepting. This would show that the two states p and q are in different Myhill-Nerode congruence classes. To test this, the Turing Machine first checks if exactly one of the two states p or q are accepting in $\mathcal{P}'_N$. If so, it immediately accepts. (This situation would correspond to $w = \varepsilon$ above.) Otherwise, it guesses the word $w = a_1 \cdots a_k$ symbol by symbol and, for every $i = 1, \ldots, k$ keeps the current states $\delta^*_{\mathcal{P}'_N}(p, a_1 \cdots a_{i-1})$ and $\delta^*_{\mathcal{P}'_N}(q, a_1 \cdots a_{i-1})$ on tape and updates these two states after every new guess of $a_i \in \Sigma$. This test requires $O(|N|)$ space since every state in $\mathcal{P}'_N$ is a set of states of N. □

In the following, whenever we say that we solve a problem *for N at level k*, we mean that we solve it for the NFA N and arbitrary sets of states $p, q \subseteq Q_N$, $a \in \Sigma$, and k. The basis of our entire mutual induction relies on being able to test if a certain transition is present in the minimal DFA equivalent to N, if it is present in its S-cut, or if some state is accepting.

Lemma 14. EDGE, ACCEPTANCE *and* EDGE-CUT *for N at level 0 can be solved in space $O(|N|^2)$.*

PROOF. We first show how to check, for a given non-empty set $p \subseteq Q_N$, whether $[p]$ is a state in $[N]$, i.e., whether it is useful. As N only has useful states, there exists a path from $[p]$ to some accepting state in $[N]$. Thus it is enough to check whether there is a path from $[\{q_N^0\}]$ to $[p]$. This clearly can be done by a nondeterministic algorithm working in space $O(|N|)$. This algorithm would guess a word symbol by symbol, simulate $\mathcal{P}_N$ on the fly, starting from $\{q_N^0\}$ and test at each step whether the reached state q is equivalent to p, i.e., if $[p] = [q]$ (see e.g., proof of Lemma 13). By Theorem 12, this can also be done by a deterministic algorithm working in space $O(|N|^2)$.

We now turn our attention to EDGE with some input $(N, p, q, a, 0)$. We must decide if $[p] \xrightarrow{a} [q]$ is a transition in $[N]$. Since $[N]$ does not have useless states, this means that we should test two things:

- both $[p]$ and $[q]$ are states in $[N]$;
- $[\delta_{\mathcal{P}_N}(p, a)] = [q]$, where $\mathcal{P}_N$ is the power set automaton of N.

The former can be solved in $O(|N|^2)$, as we mentioned before. It remains to prove the latter. Given p and a, we can easily compute $\delta_{\mathcal{P}_N}(p, a)$ in space $O(|N|)$. According to Lemma 13, we can then decide if $[\delta_{\mathcal{P}_N}(p, a)] = [q]$ in space $O(|N|^2)$.

We now show how to solve ACCEPTANCE for N at level 0 with an input $(N, p, 0)$. Notice that $[p]$ is accepting in $\mathrm{lev}_0([N], [p])$ if and only if $[p]$ is a (useful) state in $[N]$ and if $p \cap F_N \neq \emptyset$. By aforementioned reasoning the former can be done in space $O(|N|^2)$. The latter can be easily done in space $O(|N|)$.

Finally, we focus on EDGE-CUT. Notice that $\mathrm{lev}_0([N], [p])_S = [N]_S$, where S is the set of consistent symbols in $[N]$. Let $(N, p, q, a, 0)$ be the input for EDGE-CUT. We must decide if $[p] \xrightarrow{a} [q]$ is a transition in $[N]_S$, that is, a transition in $[N]$ which is not deleted in the S-cut. We can test whether $[p] \xrightarrow{a} [q]$ is a transition in $[N]$ in space $O(|N|^2)$ by the test for EDGE we just proved. If $[p] \xrightarrow{a} [q]$ is a transition in $[N]$, we test whether $[p]$ is accepting, which we can do in space $O(|N|^2)$ (as we just proved). If $[p]$ is accepting, then $[p] \xrightarrow{a} [q]$ is a transition in $[N]_S$ if and only if a is not consistent in $[N]$. To decide this, we iterate over all states $p' \subseteq Q_N$ of $\mathcal{P}_N$, test whether $[p']$ is accepting in $[N]$, and whether $[p']$ does not have an a-transition to $[q]$. Formally, we test, for every state p' of $\mathcal{P}_N$, whether ACCEPTANCE for $[p']$ returns true and EDGE for $(N, [p'], [q], a, 0)$ returns false. Again, we can solve both in space $O(|N|^2)$, which we just proved for EDGE and ACCEPTANCE. This concludes the proof. □

In the following we show that, if we can solve EDGE or EDGE-CUT at a certain level then we can use it to make more complex tests.

Lemma 15. *Assume that we can solve* EDGE *for N at level k in space $f(k, |N|)$. Then we can solve*

- REACHABILITY *for N at level k in space $f(k, |N|) + O(|N|^2)$;*
- SAMEORBIT *for N at level k in space $f(k, |N|) + O(|N|^2)$; and*
- INTERORBIT *for N at level k in space $f(k, |N|) + O(|N|^2)$.*

Analogously, if we can solve EDGE-CUT *for N at level k in space $f(k, |N|)$, then we can solve* REACHABILITY-CUT, SAMEORBIT-CUT, *and* INTERORBIT-CUT *for N at level k in space $f(k, |N|) + O(|N|^2)$.*

PROOF. We prove the statements for REACHABILITY, SAMEORBIT, and INTERORBIT. The CUT-variants of the problems are proved completely analogous where we use always EDGE-CUT instead of using EDGE.

First we show that REACHABILITY can be solved in (deterministic) space $f(k, |N|) + O(|N|^2)$. The proof is analogous to the one of Savitch's Theorem (that shows that graph reachability is in space $O(\log^2 n)$). Thereby, the proof of Savitch's Theorem can be sketched as follows. Let G be a graph, x and y nodes in G, and ℓ an integer, then the predicate $\text{PATH}(G, x, y, \ell)$ denotes whether there is a path from x to y in G of length at most ℓ. The proof then shows that this predicate can be decided deterministically in space $O(\log^2 |G|)$ which is done by a recursive algorithm that searches for a mid-point in the path. In our case, we are given N, $p, q \subseteq Q_N$, and k, and we want to know whether there exists a path from $[p]$ to $[q]$ in $\text{lev}_k([N], [p])$. Then the procedure runs in the same way as in the proof of Savitch's Theorem. The only difference is that in the case where we need to test if there is a transition we use the procedure for solving EDGE for N at level k. Since the size of $[N]$ is $O(2^{|N|})$, we obtain the $f(k, |N|) + O(|N|^2)$ upper bound.

Next we show how to implement SAMEORBIT for N at level k for given N, $p, q \subseteq Q_N$, and k. We know that $[p]$ and $[q]$ are in the same orbit if and only if REACHABILITY is true for (N, p, q, k) and for (N, q, p, k). That is, $[q]$ should be reachable from $[p]$ and vice versa in $\text{lev}_k([N], [p])$. Therefore, if we can solve REACHABILITY for N at level k in space $f(k, |N|) + O(|N|^2)$ then we can also solve SAMEORBIT for N at level k in space $f(k, |N|) + O(|N|^2)$.

Finally, we show how to solve INTERORBIT for N at level k. Given that SAMEORBIT is in space $f(k, |N|) + O(|N|^2)$, we can solve INTERORBIT by enumerating all $a \in \Sigma$ and testing whether $[p]$ and $[\delta_{\mathcal{P}_N}(p, a)]$ are not in the same orbit, i.e., checking if SAMEORBIT returns false. This requires space $f(k, |N|) + O(|N|^2 + |N| + \log |\Sigma|) = f(k, |N|) + O(|N|^2)$ in total. □

The next lemmas prove that we can compute the structure of the automaton at level $k + 1$ under the assumption that the automaton at level k is already computed. In this way, the next lemmas show a single induction step of our overall proof for DRE-Definability(NFA) is in PSPACE.

Lemma 16. *Assume that we can solve* EDGE *and* ACCEPTANCE *for N at level k in space $f(k, |N|)$. Furthermore, assume that $\text{lev}_k([N], [p])$ has no violation. Then we can solve* EDGE-CUT *for N at level k in space $f(k, |N|) + O(|N|)$.*

PROOF. Let the input for EDGE-CUT at level k be (N, p, q, a, k). Then there is a transition $[p] \xrightarrow{a} [q]$ in $\text{lev}_k([N], [p])_{S_k}$ if and only if all of the following hold:

- there is a transition $[p] \xrightarrow{a} [q]$ in $\text{lev}_k([N], [p])$;
- $[p] \xrightarrow{a} [q]$ is not deleted in the S_k-cut; and
- $\text{lev}_k([N], [p])$ has no violation.

From the lemma statement, we know that $\text{lev}_k([N], [p])$ has no violation and that we can test whether there is a transition $[p] \xrightarrow{a} [q]$ in $\text{lev}_k([N], [p])$ in space $f(k, |N|)$. In order to check whether it is deleted in the S_k-cut we test whether $[p]$

is accepting in $\mathrm{lev}_k([N],[p])$. This can also be done in space $f(k,|N|)$ according to lemma statement. Afterwards, we check whether there is a state $p' \subseteq Q_N$ in $[N]$, such that $[p']$ is accepting, and does not have an a-transition to $[q]$ in $\mathrm{lev}_k([N],[p])$. Again, both can be solved in space $f(k,|N|)$ by assumption. Overall we need space $O(|N|)$ to keep p, q, p' and a in the memory and $f(k,|N|)$ to solve Edge and Acceptance, which concludes the proof. □

Lemma 17. *Assume that we can solve* Edge-Cut *and* Acceptance *for N at level k in space $f(k,|N|)$. Furthermore, assume that $lev_k([N],[p])$ has no violation. Then we can solve* Acceptance *for N at level $k+1$ in space $f(k,|N|)+O(|N|^2)$.*

Proof. Let $(N,p,k+1)$ be the input of the Acceptance problem. Then $[p]$ is an accepting state in $\mathrm{lev}_{k+1}([N],[p])$ if and only if $\mathrm{lev}_k([N],[p])_{S_k}$ has the orbit property and either

- $[p]$ is accepting in $\mathrm{lev}_k([N],[p])$; or
- $[p]$ has an outgoing inter-orbit transition in $\mathrm{lev}_k([N],[p])_{S_k}$.

Since $\mathrm{lev}_k([N],[p])$ has no violation we know that $\mathrm{lev}_k([N],[p])_{S_k}$ has the orbit property. By Lemma 15 we can solve whether $[p]$ has an outgoing inter-orbit transition in $\mathrm{lev}_k([N],[p])_{S_k}$ in space $f(k,|N|)+O(|N|^2)$. □

Lemma 18. *Assume that we can solve* Edge-Cut *for N at level k in space $f(k,|N|)$. Furthermore, assume that $lev_k([N],[p])$ has no violation. Then we can solve* Edge *for N at level $k+1$ in space $f(k,|N|)+O(|N|^2)$.*

Proof. Let the input for Edge at level $k+1$ be N, p, $q \subseteq Q_N$, and $a \in \Sigma$. Then there is a transition $[p] \xrightarrow{a} [q]$ in $\mathrm{lev}_{k+1}([N],[p])$ if and only if all of the following hold:

- there is a transition $[p] \xrightarrow{a} [q]$ in $\mathrm{lev}_k([N],[p])_{S_k}$;
- $[p]$ and $[q]$ are in the same orbit in $\mathrm{lev}_k([N],[p])_{S_k}$; and
- $\mathrm{lev}_k([N],[p])$ has no violation.

By assumption we know that $\mathrm{lev}_k([N],[p])$ has no violation and that we can test whether there is a transition $[p] \xrightarrow{a} [q]$ in $\mathrm{lev}_k([N],[p])_{S_k}$ in space $f(k,|N|)$. By Lemma 15 we can check whether $[p]$ and $[q]$ are in the same orbit in $\mathrm{lev}_k([N],[p])_{S_k}$ in space $f(k,|N|)+O(|N|^2)$. □

Lemma 19. *Assume that we can solve* Edge *and* Acceptance *for N at level k in space $f(k,|N|)$. Furthermore, assume that $lev_k([N],[p])$ has no violation. Then we can solve* Edge *and* Acceptance *for N at level $k+1$ in space $f(k,|N|)+O(|N|^2)$.*

PROOF. According to Lemma 16 we can solve EDGE-CUT for N at level k in space $f(k, |N|) + O(|N|)$. Then by Lemma 17 we can solve ACCEPTANCE for N at level $k+1$ in space $f(k, |N|) + O(|N|^2)$. Finally, we can solve EDGE for N at level $k+1$ in space $f(k, |N|) + O(|N|^2)$ by Lemma 18. □

Lemma 20. *Assume that for $0 \leq i \leq k-1$ all automata $lev_i([N], [p])$ have no violation. Then* EDGE *and* ACCEPTANCE *for N at level k are in space $O((k+1)|N|^2)$.*

PROOF. To prove the desired upper bound we show that there exists a constant $c > 0$ such that EDGE and ACCEPTANCE for N at level k can be solved using at most space $c(k+1)|N|^2$. By Lemma 14 we know that there exists a constant c_1 such that EDGE and ACCEPTANCE for N at level 0 can be solved in space $c_1|N|^2$. Similarly, let c_2 be the constant from Lemma 18, such that EDGE and ACCEPTANCE for N at level $k+1$ can be solved in space $f(k, |N|) + c_2|N|^2$. Notice that, c_2 does not depend on k. We take $c = \max\{c_1, c_2\}$.

The proof is by induction on k. For the base case, $k = 0$, the lemma statement holds directly by Lemma 14. Assume that the lemma is true for k, i.e., EDGE and ACCEPTANCE can be solved for N at level k in space $c(k+1)|N|^2$. By Lemma 18, EDGE and ACCEPTANCE can be solved for N at level $k+1$ in space $c(k+1)|N|^2 + c_2|N|^2 \leq c(k+2)|N|^2$. This concludes the proof. □

Lemma 21. *Assume that, for $0 \leq i \leq k-1$, all automata $lev_i([N], [p])$ have no violation. Then* ISGATE-CUT *for N at level k is in space $O((k+1)|N|^2)$.*

PROOF. Let N and $p \subseteq Q_N$ be the input for ISGATE-CUT at level k. Then state $[p]$ is a gate in $\mathrm{lev}_k([N], [p])_{S_k}$ if and only if $\mathrm{lev}_k([N], [p])$ has no violation and at least one of the following hold:

- $[p]$ is accepting in $\mathrm{lev}_k([N], [p])_{S_k}$;
- $[p]$ has an outgoing inter-orbit transition in $\mathrm{lev}_k([N], [p])_{S_k}$.

By assumption $\mathrm{lev}_k([N], [p])$ has no violation. By Lemma 20 we can solve ACCEPTANCE for N at level k in space $O((k+1)|N|^2)$. By the same lemma, we also get that EDGE for N at level k is in space $O((k+1)|N|^2)$. Therefore, by Lemma 15, INTERORBIT for N at level k is in space $O((k+1)|N|^2)$. □

We are now ready to show that we can compute all the aforementioned properties of the level k automaton in polynomial space. For technical reasons, we need the assumption that all lower level automata $\mathrm{lev}_i([N], [p])$ have no violation. This is because, otherwise, the level k automaton would be empty. The proof is basically a careful induction that puts together the previous lemmas.

Lemma 22. *Assume that, for $0 \leq i \leq k-1$, all automata $lev_i([N], [p])$ have no violation. Then* EDGE, REACHABILITY, SAMEORBIT, INTERORBIT, ACCEPTANCE *and* EDGE-CUT, REACHABILITY-CUT, SAMEORBIT-CUT, INTERORBIT-CUT, *and* ISGATE-CUT *for N at level k can be solved in space $O((k+1)|N|^2)$.*

Proof. By Lemma 20 we can solve Edge and Acceptance for N at level k in space $O((k+1)|N|^2)$. For IsGate-Cut the statement holds by Lemma 21. Thus we have that, for Lemma 15 and 16, $f(k,|N|) \in O((k+1)|N|^2)$. Therefore, the assumption holds for Reachability, SameOrbit, InterOrbit, Reachability-Cut, SameOrbit-Cut, InterOrbit-Cut, and Edge-Cut. □

By Lemma 22 we can decide on-the-fly which transitions are present and which states are accepting in a level k automaton (still assuming that no violations occur in more shallow levels). Since these properties give the entire structure of a level k automaton, we can now also test for violations on level k.

Lemma 23. *Assume that, for $0 \leq i \leq k-1$, all automata $lev_i([N],[p])$ have no violation. Then* Out-Cons-Violation, Acc-Cons-Violation *and* Orbit-Cons-Violation *for N at level k can be solved in space $O((k+1)|N|^2)$.*

Proof. Let N be the input for Out-Cons-Violation at level k. We know an out-consistency violation occurs at level k of N if and only if there exist $p, q \subseteq Q_N$ such that all of the following hold:

- all automata $\mathrm{lev}_i([N],[p])$ for $0 \leq i \leq k-1$ have no violation;
- both $[p]$ and $[q]$ are gates in $\mathrm{lev}_k([N],[p])_{S_k}$;
- both $[p]$ and $[q]$ are in the same orbit of $\mathrm{lev}_k([N],[p])_{S_k}$; and
- in $\mathrm{lev}_k([N],[p])_{S_k}$ there exist a symbol $a \in \Sigma$ and $[q']$ outside the orbit of $[p]$ such that $[p] \xrightarrow{a} [q']$ but $[q] \not\xrightarrow{a} [q']$.

By assumption all $\mathrm{lev}_i([N],[p])$ for $0 \leq i \leq k-1$ have no violation. According to Lemma 22 we can solve IsGate-Cut and SameOrbit-Cut for N at level k in space $O((k+1)|N|^2)$. The last point can then be solved by enumerating all $a \in \Sigma$ and states $q' \subseteq Q_N$ where for each we check whether

- $[p] \xrightarrow{a} [q']$ exists while $[q] \xrightarrow{a} [q']$ does not; and
- $[p]$ and $[q']$ are in different orbits.

By Lemma 22, this can be done in space $O((k+1)|N|^2)$, which concludes the proof for Out-Cons-Violation.

Now, let N be the input for Acc-Cons-Violation at level k. We know an Acc-Cons-Violation occurs at level k of N if and only if there exist $p, q \subseteq Q_N$ such that all of the following hold:

- all automata $\mathrm{lev}_i([N],[p])$ for $0 \leq i \leq k-1$ have no violation;
- both $[p]$ and $[q]$ are in the same orbit of $\mathrm{lev}_k([N],[p])_{S_k}$;
- both $[p]$ and $[q]$ are gates of $\mathrm{lev}_k([N],[p])_{S_k}$; and
- exactly one of $[p]$ and $[q]$ is accepting in $\mathrm{lev}_k([N],[p])_{S_k}$.

By assumption all $\mathrm{lev}_i([N],[p])$ for $0 \leq i \leq k-1$ have no violation. According to Lemma 22 and Lemma 21, SameOrbit-Cut, IsGate-Cut and Acceptance at level k for N can be solved in $O((k+1)|N|^2)$, which concludes the proof.

Finally, let N be the input for Orbit-Cons-Violation at level k. We know an Orbit-Cons-Violation occurs at level k of N if and only if there exists a set $q_1 \subseteq Q_N$ such that:

- all automata $\mathrm{lev}_i([N],[q_1])$ for $0 \leq i \leq k-1$ have no violation;
- $[q_1]$ is accepting in $\mathrm{lev}_k([N],[q_1])$; and
- for every symbol a, there exists an accepting state $[q_2] \in \mathcal{O}([q_1])$ such that, for every state $[q] \in [N]$, at most one of the transitions $[q_1] \xrightarrow{a} [q]$ and $[q_2] \xrightarrow{a} [q]$ exists in $\mathrm{lev}_k([N],[q_1])$.

By assumption all $\mathrm{lev}_i([N],[q_1])$ for $0 \leq i \leq k-1$ have no violation. According to Lemma 22 SameOrbit, Acceptance and Edge can be solved in $O((k+1)|N|^2)$. Additionally, we need space $O(|N|)$ to keep $[q_1]$, $[q_2]$, $[q]$, and a in the memory. □

We now have all the ingredients to prove our main result.

Theorem 24. DRE-Definability(*NFA*) *and* DRE-Definability(*RE*) *are PSPACE-complete.*

Proof. DRE-Definability(RE) is known to be PSPACE-hard [15, 26]. Since an RE can be translated in polynomial time into an equivalent NFA, the lower bound also holds for NFAs.

Furthermore, the upper bound for REs follows from the upper bound for NFAs. We therefore show that DRE-Definability for an NFA N is in space $O(|N|^4)$, which proves the theorem. We assume w.l.o.g. that $|\Sigma| \leq |N|$. According to Theorem 11, a language $L(N)$ is not DRE-definable if and only if one of Out-Cons-Violation, Acc-Cons-Violation and Orbit-Cons-Violation occurs at some level k for N. According to Theorem 10, the recursion depth of Algorithm 1 is at most $|N|^2+2$, i.e., only levels k from 0 to $|N|^2+2$ are relevant for the problem.

Therefore, our PSPACE algorithm checks for violations starting from level 0 and moving to higher levels up to $|N|^2+2$. For every single level k violations can be detected in space $O((k+1)|N|^2)$ by Lemma 23. Notice that, when can apply the above lemma because we know that all smaller levels are checked before and do not contain any violation. Altogether, we can solve DRE-Definability for an NFA N in space $O((k+1)|N|^2)$ for $k = |N|^2+2$, i.e., in $O(|N|^4)$. □

4. Definability for DFAs and for RE(#)s

In this section we give minor results for variations of DRE-Definability where the input is an RE(#) or a minimal DFA. First, we show that DRE-Definability (RE(#)) is EXPSPACE-complete.

Theorem 25. DRE-Definability(*RE(#)*) *is EXPSPACE-complete.*

Proof. The upper bound is immediate from Theorem 24 and the fact that we can translate an RE(#) into an RE of exponential size by unfolding the counters.

We prove the lower bound by a reduction from the universality problem for RE(#). This problem is known to be EXPSPACE-complete which is due to Meyer and Stockmeyer. In [28], they showed that universality for regular expressions with squaring (i.e., regular expressions that only allow counters $(2, 2)$) is EXPSPACE-complete. The reduction, which is analogous to the reduction in [26] that shows DRE-Definability(RE) is PSPACE-hard, is defined as follows: Let r be the expression $\Sigma^*\#(a+b)^*a(a+b)+e\#\Sigma^*$ where e is a RE(#) and # is a symbol that is not used in the language of e. To prove correctness for the reduction, observe that $L((a+b)^*a(a+b))$ is not DRE-definable. Therefore, $L(r)$ is DRE-definable if and only if $L(e) = \Sigma^*$. □

As mentioned before, DRE-Definability can be solved in polynomial time when the input is a minimal DFA [14]. We prove that the problem is NL-hard. The precise complexity for arbitrary regular languages remains open.

Theorem 26. DRE-Definability(*minDFA*) *is NL-hard.*

Proof. We prove this result via a log-space reduction from the complement of the reachability problem in directed acyclic graphs (DAGs). This problem asks, given a DAG $G = (V, E)$, a source node s, and a target node t, whether t is reachable from s by a directed path. The DAG reachability problem is well-known to be NL-complete [29].

In this proof, we use the fact that finite languages are always DRE-definable. (This can be checked through the Bkw-Algorithm which discovers immediately that all orbits are trivial.) For the reduction, let $G = (V, E)$, and nodes $s, t \in V$ be an instance of DAG reachability. We construct a minimal DFA D such that $L(D)$ is DRE-definable if and only if vertex t is reachable from vertex s in graph G.

Based on the graph G we build an automaton $D = (Q, \Sigma, \delta, q_0, \{q_f\})$ as follows. The set Q of states is the disjoint union of the vertices V of G, plus two distinguished states q_0 (which is D's initial state) and q_f (which is D's only accepting state). The alphabet Σ is defined as $(V \uplus \{q_0, q_f\})^2$. The transitions of D are defined as follows. Let $V = \{v_1, \ldots, v_n\}$ be the vertices of G:

- for each edge $(v_i, v_j) \in E$, the transition $\delta(v_i, (v_i, v_j)) = v_j$ is in D;
- for each vertex v_i, the transitions $\delta(q_0, (q_0, v_i)) = v_i$ and $\delta(v_i, (v_i, q_f)) = q_f$ are in D; and
- the transition $\delta(t, (t, s)) = s$ is in D.

As such, every transition has its unique label. This concludes the reduction, which can be conducted in logarithmic space.
In order to complete the proof we need to show the following two facts:

1) D is minimal,

2) $L(D)$ is DRE-definable if and only if t is not reachable from s in G.

Fact 1) is immediate because all transitions are labeled by a unique symbol, which means that D has no Myhill-Nerode equivalent states.

We now show 2), that $L(D)$ is DRE-definable if and only if t is not reachable from s in G. We first show that, if there is no path from s to t in G then $L(D)$ is DRE-definable. To this end, assume that there is no path from s to t in G. We argue that the underlying graph of D is a DAG. Towards contradiction, assume that there is a cycle in D. It clearly contains neither q_0 nor q_f, thus it uses only transitions which originate from the edges of G and possibly the transition $t \to s$. Since G is a DAG, the transition $t \to s$ is necessarily used. Thus the rest of the cycle forms a path from s to t, which contradicts our assumption. Thus, the underlying graph of D is a DAG. This means that D defines a finite and therefore DRE-definable language.

It remains to show the implication from left to right, i.e., that the DRE-definability of D implies that t is not reachable from s. Towards contradiction, assume that t is reachable from s in G. Thus, s and t are in the same orbit of D which is not the orbit of q_f because q_f has no outgoing transitions. Therefore, the transitions $s \xrightarrow{(s,q_f)} q_f$ and $t \xrightarrow{(t,q_f)} q_f$ are inter-orbit transitions and s and t are gates of the same orbit. However, their transitions to q_f are not out-consistent and D does not fulfill the orbit property, i.e., $L(D)$ is not DRE-definable. This contradicts our assumption and concludes the proof. □

5. Conclusions

We have pinned down the exact complexity of testing whether a regular expression can be determinized (or, equivalently, be expressed by a deterministic regular expression) and considered several minor variations of this problem:

- Deciding if a regular expression can be determinized; or if the language of a given RE or NFA can be expressed by a deterministic regular expression is PSPACE-complete ([15] and Theorem 24).
- Deciding if the language of a given RE(#) can be expressed by a deterministic regular expression is EXPSPACE-complete (Theorem 25).
- Deciding if the language of a given minimal DFA can be expressed by a deterministic regular expression is in PTIME [14]. We proved that this problem is NL-hard (see Theorem 26).

The precise complexity of the last question remains open. Our proofs provide additional insights on such DRE-definable languages and on the decision algorithm of Brüggemann-Klein and Wood.

We would like to conclude by mentioning an open problem regarding the determinization of regular expressions. What is the worst-case blow-up in the

determinization process when converting a regular expression to an equivalent deterministic one? At the moment, we only know that a single exponential blow-up cannot be avoided [18] but the best known upper bound is double exponential. While our proofs seem to give some insight in how to improve this upper bound, testing whether a language is DRE-definable and actually constructing a minimal equivalent DRE are quite different matters. It is not yet clear to us how our techniques can be leveraged to obtain better upper bounds for this question.

Note. Several results in this work are obtained independently by Ping et al. [30]. Ping et al. present an alternative proof for the PSPACE upper bound for Theorem 24 which additionally shows that only quadratic space is needed. Furthermore, they show that DRE-Definability(minDFA) is in NL when the size of the automaton's alphabet is at most logarithmic in the number of states.

Acknowledgement. We thank Wouter Gelade for bringing this problem to our attention. We also thank an anonymous reviewer for the Journal of Computer and System Sciences for comments that led to a shorter and simpler proof of the EXPSPACE lower bound for DRE-Definability(RE(#)) (Theorem 25).

[1] W. Czerwinski, C. David, K. Losemann, W. Martens, Deciding definability by deterministic regular expressions, in: Proceedings of the 16th International Conference on Foundations of Software Science and Computation Structures (FOSSACS), Springer, 2013, pp. 289–304.

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language XML 1.0 (fifth edition), Tech. rep., World Wide Web Consortium (W3C), w3C Recommendation at http://www.w3.org/TR/xml/ (November 2008).

[3] L. Segoufin, V. Vianu, Validating streaming XML documents, in: Proceedings of the 21th Symposium on Principles of Database Systems (PODS), ACM, 2002, pp. 53–64.

[4] A. Balmin, Y. Papakonstantinou, V. Vianu, Incremental validation of XML documents, ACM Transactions on Database Systems (TODS) 29 (4) (2004) 710–751.

[5] C. Konrad, F. Magniez, Validating XML documents in the streaming model with external memory, ACM Transactions on Database Systems (TODS) 38 (4) (2013) 27.

[6] T. Milo, D. Suciu, V. Vianu, Typechecking for XML transformers, Journal of Computer and System Sciences (JCSS) 66 (1) (2003) 66–97.

[7] W. Martens, F. Neven, On the complexity of typechecking top-down XML transformations, Theoretical Computer Science (TCS) 336 (1) (2005) 153–180.

[8] S. Maneth, A. Berlea, T. Perst, H. Seidl, XML type checking with macro tree transducers, in: Proceedings of the 24th Symposium on Principles of Database Systems (PODS), ACM, 2005, pp. 283–294.

[9] F. Neven, T. Schwentick, On the complexity of XPath containment in the presence of disjunction, DTDs, and variables, Logical Methods in Computer Science (LMCS) 2 (3) (2006) 1–30.

[10] P. T. Wood, Containment for XPath fragments under DTD constraints, in: Proceedings of the 9th International Conference on Database Theory (ICDT), Springer, 2003, pp. 297–311.

[11] M. Arenas, P. Barceló, L. Libkin, F. Murlak, Foundations of Data Exchange, Cambridge University Press, 2014.

[12] D. Fallside, P. Walmsley, XML Schema Part 0: Primer (second edition), Tech. rep., World Wide Web Consortium (W3C), w3C Recommendation at http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ (October 2004).

[13] A. Brüggemann-Klein, D. Wood, Deterministic regular languages, in: Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Springer, 1992, pp. 173–184.

[14] A. Brüggemann-Klein, D. Wood, One-unambiguous regular languages, Information and Computation 142 (2) (1998) 182–206.

[15] G. J. Bex, W. Gelade, W. Martens, F. Neven, Simplifying XML Schema: effortless handling of nondeterministic regular expressions, in: Proceedings of the International Conference on Management of Data (SIGMOD), ACM, 2009, pp. 731–744.

[16] G. J. Bex, W. Gelade, F. Neven, S. Vansummeren, Learning deterministic regular expressions for the inference of schemas from XML data, ACM Transactions on the Web 4 (4) (2010) 14:1–14:32.

[17] W. Gelade, F. Neven, Succinctness of the complement and intersection of regular expressions, ACM Transactions on Computational Logic (TOCL) 13 (1) (2012) 4:1–4:19.

[18] K. Losemann, W. Martens, M. Niewerth, Descriptional complexity of deterministic regular expressions, in: Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS), Springer, 2012, pp. 643–654.

[19] H. Chen, L. Chen, Inclusion test algorithms for one-unambiguous regular expressions, in: Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing (ICTAC), Springer, 2008, pp. 96–110.

[20] D. Colazzo, G. Ghelli, C. Sartiani, Efficient inclusion for a class of XML types with interleaving and counting, Information Systems (IS) 34 (7) (2009) 643–656.

[21] B. Groz, S. Maneth, S. Staworko, Deterministic regular expressions in linear time, in: Proceedings of the 31st Symposium on Principles of Database Systems (PODS), ACM, 2012, pp. 49–60.

[22] P. Kilpeläinen, R. Tuhkanen, One-unambiguity of regular expressions with numeric occurrence indicators, Information and Computation (IC) 205 (6) (2007) 890–916.

[23] W. Gelade, M. Gyssens, W. Martens, Regular expressions with counting: weak versus strong determinism, SIAM Journal on Computing (SICOMP) 41 (1) (2012) 160–190.

[24] D. Hovland, Regular expressions with numerical constraints and automata with counters, in: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC), Springer, 2009, pp. 231–245.

[25] P. Kilpeläinen, Checking determinism of XML Schema content models in optimal time, Inf. Syst. 36 (3) (2011) 596–617.

[26] B. Groz, XML security views: Queries, updates and schemas, Ph.D. thesis, Université Lille 1 (2012).

[27] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Pearson Education, 2007.

[28] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: Proceedings of the 13th Annual Symposium on Switching and Automata Theory (SWAT), IEEE Computer Society, 1972, pp. 125–129.

[29] N. D. Jones, Space-bounded reducibility among combinatorial problems, Journal of Computer and System Sciences (JCSS) 11 (1) (1975) 68–85.

[30] P. Lu, J. Bremer, H. Chen, Deciding determinism of regular languages, Theory of Computing Systems (2014) 1–43.