Information and Computation 209 (2011) 456–470

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Descriptional and computational complexity of finite automata—A survey☆

Markus Holzer*, Martin Kutrib

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany

ARTICLE INFO

Article history:
Available online 18 November 2010

Keywords:
Finite automata
Determinism
Nondeterminism
Alternation
Descriptional complexity
Computational complexity
Survey

ABSTRACT

Finite automata are probably best known for being equivalent to right-linear context-free grammars and, thus, for capturing the lowest level of the Chomsky-hierarchy, the family of regular languages. Over the last half century, a vast literature documenting the importance of deterministic, nondeterministic, and alternating finite automata as an enormously valuable concept has been developed. In the present paper, we tour a fragment of this literature. Mostly, we discuss developments relevant to finite automata related problems like, for example, (i) simulation of and by several types of finite automata, (ii) standard automata problems such as fixed and general membership, emptiness, universality, equivalence, and related problems, and (iii) minimization and approximation. We thus come across descriptional and computational complexity issues of finite automata. We do not prove these results but we merely draw attention to the big picture and some of the main ideas involved.

1. Introduction

Nondeterministic finite automata (NFAs) were introduced in [75], where their equivalence to deterministic finite automata (DFAs) was shown. Later, the concept of alternation was developed in [14], where also alternating finite automata (AFAs) were investigated, which turned out to be equivalent to DFAs, too. It is well known that NFAs can offer exponential saving in the number of states compared with deterministic finite automata (DFAs). A similar result also holds for AFAs simulated by DFAs with a tight double exponential state bound of 2^{2^n} shown in [14].

Much work has been done in the study of descriptional complexity of simulation of and by several types of automata and on the computational complexity of decision problems related to finite automata. The goal of this research is to obtain tight bounds on simulation results and to classify the computational complexity of problems according to the complexity classes NC^1, L, NL, P, NP, and PSPACE, or others—for basics in computational complexity theory we refer to, e.g. [41]. Our tour on the subjects listed in the abstract cover some (recent) results in the field of descriptional and computational complexity. It obviously lacks completeness and it reflects our personal view of what constitute the most interesting links to descriptional and computational complexity theory. In truth there is much more to the regular languages, DFAs, NFAs, etc., than one can summarize here. For a recent survey on finite automata we refer to [87] and [38].

Our nomenclature of finite automata is as follows: the powerset of a set Q is denoted by 2^Q and the empty word by λ. The *reversal of a word* w over alphabet Σ, referred to w^R, is inductively defined as $w^R := \lambda$, if $w = \lambda$, and $w^R := u^R a$, if $w = au$, for $a \in \Sigma$ and $u \in \Sigma^*$. Then the *reversal of the language* $L \subseteq \Sigma^*$ is set to $L^R := \{ w^R \mid w \in L \}$.

A *nondeterministic finite automaton* (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of *states*, Σ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\delta : Q \times \Sigma \to 2^Q$ is the *transition function*.

☆ This is an extended and revised version of: M. Holzer, M. Kutrib. Descriptional and Computational Complexity of Finite Automata, Proc. 3rd Int. Conf. Language and Automata Theory and Applications, LNCS 5457, 23–42, Springer-Verlag, 2009.

* Corresponding author.
E-mail addresses: holzer@informatik.uni-giessen.de (M. Holzer), kutrib@informatik.uni-giessen.de (M. Kutrib).

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.11.013

A finite automaton is *deterministic* (DFA) if and only if $|\delta(q, a)| = 1$, for all states $q \in Q$ and letters $a \in \Sigma$. In this case we simply write $\delta(q, a) = p$ instead of $\delta(q, a) = \{p\}$ assuming that the transition function is a mapping $\delta : Q \times \Sigma \to Q$. So, any DFA is *complete*, that is, the transition function is total, whereas it may be a partial function for NFAs in the sense that the transition function of nondeterministic machines may map to the empty set.

Let $L(A) = \{ w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \}$ be the *language accepted* by the NFA or DFA A, where the transition function is recursively extended to a $\delta : Q \times \Sigma^* \to 2^Q$. A finite automaton is said to be *minimal* if its number of states is minimal with respect to the accepted language. Note that a sink state is counted for DFAs, since they are always complete, whereas it is not counted for NFAs, since these devices are not necessarily complete. For further details we refer to [41].

We identify the logical values *false* and *true* with 0 and 1 and write $\{0, 1\}^Q$ for the set of finite functions from Q into $\{0, 1\}$, and $\{0, 1\}^{\{0,1\}^Q}$ for the set of Boolean formulas (functions) mapping $\{0, 1\}^Q$ into $\{0, 1\}$.

An *alternating finite automaton* (AFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are as for NFAs, and $\delta : Q \times \Sigma \to \{0, 1\}^{\{0,1\}^Q}$ is the *transition function*. The transition function maps pairs of states and input symbols to Boolean formulas. Before we define the language accepted by the AFA A we have to explain how a word is accepted. As the input is read (from left to right), the automaton "builds" a propositional formula, starting with the formula q_0, and on reading an input a, replaces every $q \in Q$ in the current formula by $\delta(q, a)$. The input is *accepted* if and only if the constructed formula on reading the whole input evaluates to 1 on substituting 1 for q, if $q \in F$, and 0 otherwise. This substitution defines a mapping from Q into $\{0, 1\}$ which is called the *characteristic vector* f_A of A. Then the *language accepted* by the AFA A is defined as $L(A) = \{ w \in \Sigma^* \mid w \text{ is accepted by } A \}$. A slight generalization of AFAs are *Boolean automata* [13], that are similarly defined as AFAs except for the initial state, which is allowed to be an arbitrary propositional formula f_0. Two automata are *equivalent* if and only if they accept the same language. For further details we refer to [14] and [41].

Example 1 [88]. Let $A = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ be an AFA with transition function defined through

$$\delta(q_0, a) = q_1 \wedge q_2,$$
$$\delta(q_1, a) = q_2,$$
$$\delta(q_2, a) = \overline{q}_1 \wedge q_2,$$
$$\delta(q_0, b) = 0,$$
$$\delta(q_1, b) = q_1 \wedge q_2,$$

and

$$\delta(q_2, b) = q_1 \vee \overline{q}_2.$$

On input *aba* the propositional formula evolves as follows. Starting with q_0 after reading the first input symbol a the formula is $q_1 \wedge q_2$. After reading b we obtain $(q_1 \wedge q_2) \wedge (q_1 \vee \overline{q}_2)$, and after reading the last symbol a the formula $(q_2 \wedge (\overline{q}_1 \wedge q_2)) \wedge (q_2 \vee (\overline{\overline{q}_1 \wedge q_2}))$. After substituting the characteristic vector, that is, 0 for q_0, q_1, and 1 for q_2 we have $(1 \wedge (\overline{0} \wedge 1)) \wedge (1 \vee (\overline{\overline{0} \wedge 1}))$ which evaluates to 1. Therefore, the input *aba* is accepted.

We continued with another example that shows that the shortest word accepted by an AFA or Boolean automaton can be of exponential length in the number of states. This interesting property will give rise to some variants on the computational complexity of non-emptiness problems discussed in Section 3.2.

Example 2. Let $p_1, p_2, \ldots, p_n$ the first n be prime numbers. Consider the Boolean automaton $A = (Q, \{a\}, \delta, f_0, F)$ with $Q = Q_1 \cup Q_2 \cup \cdots \cup Q_n$ (the union being disjoint) such that $Q_i = \{q_{i,j} \mid 0 \leq j < p_i\}$, for $1 \leq i \leq n$, and set of accepting states $F = \{ q_{i,0} \mid 1 \leq i \leq n \}$. The initial propositional formula f_0 is

$$q_{1,0} \wedge q_{2,0} \wedge \cdots \wedge q_{n,0},$$

and the transition function is defined by $\delta(q_{i,j}, a) = q_{i,j+1 \bmod p_i}$ for $1 \leq i \leq n$ and $0 \leq j < p_i$. Thus, the Boolean automaton A consists of n independent cycles of lengths $p_1, p_2, \ldots, p_n$. Since the computation starts in the states $q_{1,0}, q_{2,0}, \ldots, q_{n,0}$ simultaneously and these states are the only accepting ones, any word w over the alphabet $\{a\}$ accepted by A obeys the property that the length of w is congruent 0 modulo each prime number p_i, i.e., $|w| = 0 \bmod p_i$, for every p_i with $1 \leq i \leq n$. By the well-known fact that the sum $\sum_{i=1}^{n} p_i$ of the first n prime numbers is exponentially smaller than their product, the stated claim follows. By easy means the Boolean automaton A can be redesigned to become an ordinary AFA with one additional state q_0 that simulates the behaviour of f_0. The claim on the shortest word accepted stills holds true.

2. Descriptional complexity of finite automata simulations

Since regular languages have many representations in the world of finite automata, it is natural to investigate the succinctness of their representation by different types of automata in order to optimize the space requirements. Here we measure the

costs of representations in terms of the states of a minimal automaton accepting a language. More precisely, the *simulation problem* is defined as follows:

- Given two classes of finite automata C_1 and C_2, how many states are sufficient and necessary in the worst case to simulate n-state automata from C_1 by automata from C_2?

In particular, we are interested in simulations between DFAs, NFAs, and AFAs.

It is well known that to any NFA one can always construct an equivalent DFA [75]. This so-called *powerset construction*, where each state of the DFA is associated with a subset of NFA states, turned out to be optimal, in general. That is, the bound on the number of states necessary for the construction is tight in the sense that for an arbitrary n there is always some n-state NFA which cannot be simulated by any DFA with strictly less than 2^n states [71,74]. So, NFAs can offer exponential savings in the number of states compared with DFAs. This gives rise to the following theorem.

Theorem 3 (NFA by DFA simulation). *Let $n \geq 1$ and A be an n-state NFA. Then 2^n states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

The situation becomes more involved when AFAs come into play. Alternating finite automata as we have defined them have been developed in [14]. At the same period in [13] the so-called *Boolean automata* were introduced. Note, that several authors use the notation "alternating finite automata" but rely on the definition of Boolean automata. Though it turned out that both types are almost identical, there are differences with respect to the initial configurations. While for AFAs the computation starts with the fixed propositional formula q_0, a Boolean automaton starts with an arbitrary propositional formula. Clearly, this does not increase their computational capacities. However, it might make the following difference from a descriptional complexity point of view.

Lemma 4 (Boolean automata by AFA simulation). *Let $n \geq 1$ and A be an n-state Boolean automaton. Then $n+1$ states are sufficient for an AFA to accept $L(A)$.*

In the first step of the simulation, the additional state of the AFA is used to derive the successors of the initial propositional formula of the Boolean automaton from the fixed initial propositional formula q_0 of the AFA. The additional state is unreachable afterwards. It is an open problem whether or not the additional state is really necessary, that is, whether the bound of $n+1$ is tight. See [23] for more details on alternating finite automata having an initial state that is unreachable after the first step.

Next we turn to the simulation of AFAs by NFAs and DFAs. The tight bound of 2^{2^n} states for the deterministic simulation of n-state AFAs has already been shown in the famous fundamental papers [14] for AFAs and [13,65] for Boolean automata.

Theorem 5 (AFA by DFA simulation). *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then 2^{2^n} states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

The original proofs of the upper bound rely on the fact that an AFA or a Boolean automaton can enter only finitely many internal situations, which are given by Boolean functions depending on n Boolean variables associated with the n states. The number of 2^{2^n} such functions determines the upper bound.

The proofs provide little insight in the way an NFA can perform the simulation. In [23] the constructions of simulating NNFAs are presented which implies the same upper bound. Basically, an NNFA is an NFA with multiple entry states, where initially one is nondeterministically chosen. Let $A = (Q, \Sigma, \delta, q_0, F)$ be an n-state AFA with $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ and characteristic vector f_A. Then we consider the NNFA $A' = (\{0,1\}^Q, \Sigma, \delta', Q_0, \{f_A\})$, where the set of initial states is $Q_0 = \{\, u \in \{0,1\}^Q \mid u(q_0) = 1 \,\}$, and the transition function is defined by

$$\delta'(u,a) = \{\, u' \in \{0,1\}^Q \mid \delta(q,a)(u') = u(q) \text{ for every } q \in Q \,\},$$

for all $u \in \{0,1\}^Q$ and $a \in \Sigma$. So, the NNFA simulates the AFA by guessing the sequence of functions of the form $\{0,1\}^Q$ that appear during the evaluation of the propositional formula computed by the AFA in reverse order. Since there are 2^n such functions we obtain the upper bound stated in Theorem 6. Moreover, since the powerset constructions works also fine for the NNFA by DFA simulation, the presented construction also reveals the upper bound for the AFA simulation by DFAs already stated in Theorem 5. The construction for Boolean automata is derived from above by considering the initial Boolean formula f_0 of the Boolean automaton and to change the set of initial states of the NNFA accordingly. To this end, it suffices to define Q_0 to be $\{\, u \in \{0,1\}^Q \mid f_0(u) = 1 \,\}$. From the construction we derive the upper bound of the next theorem.

Theorem 6 (AFA by NNFA simulation). *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then 2^n states are sufficient and necessary in the worst case for an NNFA to accept $L(A)$.*

The matching lower bound of Theorem 5 is shown in [14] for AFAs by witness languages in a long proof. Before we come back to this point for Boolean automata, we turn to an interesting aspect of AFAs and Boolean automata. One can observe that the construction of the simulating NNFA is backward deterministic [14]. So, the reversal of a language accepted by an n-state AFA or Boolean automaton is accepted by a *not necessarily complete* 2^n-state DFA which in turn can be simulated by a (2^n+1)-state *complete* DFA. This result has significantly be strengthened in [65], where it is shown that the reversal of *every* n-state DFA language is accepted by a Boolean automaton with $\lceil \log_2(n) \rceil$ states. With other words, *with restriction to reversals* of regular languages a Boolean automaton can always save exponentially many states compared with a DFA. The next theorem summarizes these results.

Theorem 7 (Reversed AFA by DFA simulation)**.** *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then 2^n+1 states are sufficient and necessary in the worst case for a DFA to accept the reversal of $L(A)$. If the minimal DFA accepting the reversal of $L(A)$ does not have a rejecting sink state, then 2^n states are sufficient. Moreover, the reversal of every language accepted by an n-state DFA is accepted by a Boolean automaton with $\lceil \log_2(n) \rceil$ states.*

The theorem left open whether the reversal of *every* n-state DFA language is also accepted by some *AFA* with $\lceil \log_2(n) \rceil$ states. However, we know that $\lceil \log_2(n) \rceil + 1$ states are sufficient for this purpose.

Now we are prepared to argue for the matching lower bound of Theorem 5 for Boolean automata in a simple way. It is well known that for any $m \geq 1$ there is an m-state DFA A such that any DFA accepting the reversal of $L(A)$ has 2^m states [65]. Setting $m = 2^n$ we obtain a 2^n-state DFA language $L(A)$ whose reversal is accepted by a Boolean automaton with n states by Theorem 7. On the other hand, the reversal of $L(A)$ takes at least 2^{2^n} states to be accepted deterministically.

Next we argue that the upper bound of Theorem 6 cannot be improved in general. To this end, let A be an n-state AFA or Boolean automaton such that any equivalent DFA has 2^{2^n} states. Let m be the minimal number of states for an equivalent NNFA. Since the NNFA can be simulated by a DFA with at most 2^m states, we conclude $2^m \geq 2^{2^n}$, that is, the NNFA has at least $m \geq 2^n$ states.

So far, we have only considered nondeterministic finite automata with multiple entry states. It is known that any such NNFA can be simulated by an NFA having one more state. The additional state is used as new sole initial state which is appropriately connected to the successors of the old initial states. On the other hand, in general this state is needed. For example, consider the language $\{ a^n \mid n \geq 0 \} \cup \{ b^n \mid n \geq 0 \}$ which is accepted by a 2-state NNFA but takes at least three states to be accepted by an NFA. Nevertheless, it is an open problem whether there are languages accepted by n-state AFAs or Boolean automata such that any equivalent NFA has at least 2^n+1 states. In [23] it is conjectured that this bound presented in the following theorem is tight.

Lemma 8 (AFA by NFA simulation)**.** *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then 2^n+1 states are sufficient for an NFA to accept $L(A)$. Moreover, for every $n \geq 1$ there is an n-state AFA or Boolean automaton A such that any NFA accepting $L(A)$ has at least 2^n states.*

We now direct our attention to the question whether alternation can always help to represent a regular language succinctly. It is well known that nondeterminism cannot help for all languages. For example, any NFA accepting the language $L_n = \{a^n\}^*$, for a constant $n \geq 1$, has at least n states, and L is accepted by an n-state DFA as well. So, how about the worst case of the language representation by alternating finite automata? The situation seems to be more sophisticated. Theorem 7 says that for reversals of n-state DFA languages we can *always* achieve an exponential saving of states. Interestingly, this potential gets lost when we consider the n-state DFA languages itself (instead of their reversals). The next theorem and its corollary are from [66].

Theorem 9. *For every $n \geq 1$ there exists a minimal DFA A with n states such that any AFA or Boolean automaton accepting $L(A)$ has at least n states.*

The DFAs $A_n = (\{q_0, q_1, \ldots, q_{n-1}\}, \{a, b\}, \delta, q_1, F)$ witness the theorem for $n \geq 2$, where $F = \{ q_i \mid 0 \leq i \leq n-1$ and i even $\}$ and the transition function given by

$$\delta(q_i, a) = q_{(i+1) \bmod n} \quad \text{and} \quad \delta(q_i, b) = \begin{cases} q_i & \text{for } 0 \leq i \leq n-3 \\ q_{n-1} & \text{for } i \in \{n-2, n-1\}. \end{cases}$$

Each DFA A_n has the property that any DFA A'_n accepting the reversal of $L(A)$ has at least 2^n states. Moreover, A_n and A'_n both are minimal, complete and do not have a rejecting sink state [65]. Assume that $L(A)$ is accepted by some AFA or Boolean automaton with $m < n$ states. Then the reversal of $L(A)$ would be accepted by some DFA having at most 2^m states by Theorem 7. This is a contradiction since $2^m < 2^n$.

Corollary 10. *Let A be an n-state DFA such that any DFA accepting the reversal of $L(A)$ has at least 2^n states and no rejecting sink state. Then any AFA or Boolean automaton accepting $L(A)$ has at least n states.*

Up to now we dealt with simulations whose costs optimality is witnessed by regular languages which may be built over alphabets with two or more letters. For the particular case of *unary regular languages*, that is, languages over a single letter alphabet, the situation turned out to be significantly different. The problem of evaluating the costs of unary automata simulations was raised in [79], and has led to emphasize some relevant differences with the general case. So, we next turn to draw a part of that picture, which is complemented by the sophisticated studies in [70] which reveal tight bounds also for many other types of finite automata and, in addition, is a valuable source for further references.

Unary NFAs can be much more concise than DFAs, but yet not as much as for the general case. For state complexity issues of unary finite automata Landau's function $F(n) = \max\{ \operatorname{lcm}(x_1, \ldots, x_k) \mid x_1, \ldots, x_k \geq 1 \text{ and } x_1 + \cdots + x_k = n \}$, which gives the maximal order of the cyclic subgroups of the symmetric group on n elements, plays a crucial role. Here, lcm denotes the least common multiple. Since F depends on the irregular distribution of the prime numbers, we cannot expect to express $F(n)$ explicitly by n. In [62,63] the asymptotic growth rate

$$\lim_{n\to\infty} (\ln F(n)/\sqrt{n \cdot \ln n}) = 1$$

was determined, which for our purposes implies the (sufficient) rough estimate $F(n) \in e^{\Theta(\sqrt{n\cdot\ln n})}$. The following asymptotic tight bound on the unary NFA by DFA simulation was presented in [17,18]. Its proof is based on a normal-form for unary NFAs introduced in [17]. Each n-state unary NFA can be replaced by an equivalent $O(n^2)$-state NFA consisting of an initial deterministic tail and some disjoint deterministic loops, where the automaton makes only a single nondeterministic decision after passing through the initial tail, which chooses one of the loops.

Theorem 11 (Unary NFA by DFA simulation). *Let $n \geq 1$ and A be an n-state NFA accepting a unary language. Then $e^{\Theta(\sqrt{n\cdot\ln n})}$ states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

In general, the deterministic simulation of AFAs may cost a double exponential number of states. The unary case is cheaper. Since every unary language coincides trivially with its reversal, the upper bound of the following theorem is immediately derived from Theorem 7. The lower bound can be seen by considering the single word language $L_n = \{a^{2^n-1}\}$. For each $n \geq 1$, the language L_n is accepted by some minimal $(2^n + 1)$-state DFA. The construction of an equivalent n-state AFA is omitted here. So, we derive the next theorem.

Theorem 12 (Unary AFA by DFA simulation). *Let $n \geq 1$ and A be an n-state AFA accepting a unary language. Then $2^n + 1$ states are sufficient and necessary in the worst case for a DFA to accept $L(A)$. If the minimal DFA does not have a rejecting sink state, then 2^n states are sufficient.*

Interestingly, to some extent for unary languages it does not matter in general whether we simulate an AFA deterministically or nondeterministically. The tight bounds differ at most by one state. The upper bound of this claim follows since any DFA is also an NFA and NFAs are not necessarily complete. The lower bound is again witnessed by the single word languages L_n, which requires 2^n states for any NFA accepting it.

Corollary 13 (Unary AFA by NFA simulation). *Let $n \geq 1$ and A be an n-state AFA accepting a unary language. Then 2^n states are sufficient and necessary in the worst case for an NFA to accept $L(A)$.*

Theorem 9 revealed that alternation cannot help to reduce the number of states of DFAs or NFAs in all cases. The same is true for nondeterministic simulations of DFAs in general and in the unary case. The latter can be seen by the unary languages $\{a^n\}^*$, for $n \geq 1$. However, for unary languages alternation does help. By Theorem 12 we know already that any AFA simulating an n-state DFA accepting a unary language has not less than $\lceil \log_2(n) \rceil - 1$ states. Once more the unary single word languages L_n are witnesses that this saving can be achieved. This gives rise to the next theorem.

Theorem 14 (Unary DFA by AFA simulation). *Let $n \geq 1$ and A be an n-state DFA accepting a unary language. Then $\lceil \log_2(n) \rceil - 1$ states are necessary for an AFA to accept $L(A)$. Moreover, there exists a minimal DFA A with n states accepting a unary language such that any minimal AFA accepting $L(A)$ has exactly $\lceil \log_2(n) \rceil - 1$ states.*

Finally, we derive the *always possible* savings for unary NFA by AFA simulations as follows. Given some n-state NFA accepting a unary language, by Theorem 11 we obtain an equivalent DFA that has at most $e^{\Theta(\sqrt{n\cdot\ln n})} = 2^{\Theta(\sqrt{n\cdot\ln n})}$ states. Now Theorem 7 in combination with Lemma 4 says essentially that there is an equivalent AFA with $\Theta(\sqrt{n \cdot \ln n})$ states. In order to see that these savings are optimal in general, consider a unary n-state NFA such that any equivalent DFA must have $e^{\Theta(\sqrt{n\cdot\ln n})}$ states. Since the bound of Theorem 11 is tight such automata exist. Clearly, any equivalent AFA has at least $\Theta(\sqrt{n \cdot \ln n})$ states. Otherwise there would be an equivalent DFA with less than $e^{\Theta(\sqrt{n\cdot\ln n})}$ states by Theorem 12.

Theorem 15 (Unary NFA by AFA Simulation)**.** *Let $n \geq 1$ and A be a minimal n-state NFA accepting a unary language. Then $\Theta(\sqrt{n \cdot \ln n})$ states are sufficient for an AFA to accept $L(A)$. Moreover, there exists an n-state NFA accepting a unary language such that any equivalent AFA requires $\Theta(\sqrt{n \cdot \ln n})$ states.*

The justification of the second part of the theorem gives rise to the following corollary.

Corollary 16. *Let A be a unary n-state NFA such that any equivalent DFA has at least $e^{\Theta(\sqrt{n \cdot \ln n})}$ states. Then any AFA accepting $L(A)$ has at least $\Theta(\sqrt{n \cdot \ln n})$ states.*

In the remainder of this section we focus on a structural property of AFAs and the role played by the number of accepting states. The next theorem shows that negations in the Boolean functions defining an AFA can be avoided at the cost of increasing the number of states by a factor of two [23].

Theorem 17. *For every AFA $A = (Q, \Sigma, \delta, q_0, F)$ one can construct an equivalent AFA $A' = (Q', \Sigma, \delta', q_0', F')$ with $|Q'| = 2|Q|$ such that δ' maps from $Q' \times \Sigma$ to $2^{2^{Q'}}$, that is, the transition function can be identified with Boolean formulas without negations.*

While the family of languages accepted by DFAs with k accepting states is strictly contained in the family of languages accepted by DFAs with $k+1$ accepting states, for $k \geq 0$, it is known that for NFAs two states are always sufficient. In particular, any regular language not containing the empty word is accepted by an NFA with one accepting state, while regular languages containing the empty word may require two accepting states. The situation for AFAs is in contrast to the situation for DFAs but parallels the situation for NFAs. More precisely, the following theorem has been shown in [23].

Theorem 18. *Let $n \geq 1$. For every n-state AFA A accepting a λ-free regular language one can construct an equivalent n-state AFA A' without accepting state. If $L(A)$ contains the empty word, then A' has one sole accepting state that coincides with the start state.*

3. Computational complexity of some decision problems for finite automata

We recall what is known from the computational complexity point of view on some standard problems for regular languages. We assume that a regular language is specified by a DFA, NFA, or AFA unless otherwise stated. The problems considered in this section are all decidable, as most problems for finite automata, and they will be grouped as mentioned in the abstract. These problems have finite automata as inputs. Therefore we need an appropriate coding function $\langle\cdot\rangle$ which maps a finite automaton A and a string w to a word $\langle A, w\rangle$ over a fixed alphabet Σ. We do not go into the details of $\langle\cdot\rangle$, but assume it fulfills certain standard properties; for instance, that the coding of the input alphabet symbols as well as the coding of the states is of logarithmic length on the alphabet size and on the number of states.

3.1. The fixed and general membership problem

Our tour on problems for regular languages is started with the definition of the fixed and general membership problem: The former problem is device independent by definition and is commonly referred to in the literature as the *fixed membership problem* for regular languages:

- *Fix* a finite automaton A. For a given word w, does the word w belong to the language $L(A)$, i.e., is $w \in L(A)$?

A natural generalization is the *general membership problem*, which is defined as follows:

- *Given* a finite automaton A and a word w, i.e., a suitable coding $\langle A, w\rangle$, does the word w belong to the language $L(A)$, i.e., is $w \in L(A)$?

Obviously, the fixed membership problem for regular languages reduces to the general membership problem for any suitable class of automata, that describes the family of regular languages. On the other hand, the complexity of the general membership problem may depend on the given language descriptor. For instance, it is easy to see that the general membership problem for DFAs is in L, and in fact, complete for L under weak reductions. The problem is NL-complete for NFAs [57], and becomes P-complete for AFAs as shown in [55]. These completeness results even hold for finite automata accepting languages over a singleton, i.e., unary languages. We summarize these results in the following theorem:

Theorem 19 (General membership)**.** *The general membership problem for DFAs is* L*-complete with respect to constant depth reducibilities. Moreover, the problem is* NL*-complete for NFAs and becomes* P*-complete for AFAs. The results remain valid for automata accepting unary languages.*

For the fixed membership problem there is much more to tell, since there is a deep and nice connection between this problem and circuit complexity theory. There exist several characterizations, in terms of formal logic, semigroup theory, and operations on languages, of the regular languages in the circuit complexity classes AC^0, ACC^0, and NC^1—see, e.g. [8,9,68]. Here AC^0 (NC^1, respectively) is the class of languages accepted by uniform circuit families of constant (logarithmic, respectively) depth, polynomial size, with AND- and OR-gates of unbounded (bounded, respectively) fan-in and ACC^0 is the class of languages accepted by AC^0 circuits with additional MODULO-gates. Hence $\mathsf{AC}^0 \subseteq \mathsf{ACC}^0 \subseteq \mathsf{NC}^1$, and note that AC^0 is distinct from NC^1 by Ajtai [4] and Furst et al. [24]. It is conjectured that ACC^0 is a proper subset of NC^1, too.

First, observe that by a divide and conquer approach it is easy to see that regular languages in general belong to NC^1. On the other hand, the NC^1 lower bound (under weak reductions such as constant depth reducibilities) was established in the landmark paper of Barrington [7], where it was shown that bounded width polynomial size programs over finite monoids recognize exactly the languages in NC^1. To this end, it was shown how to simulate the AND-, OR-, and NOT-gates of an NC^1-circuit with programs over the symmetric group S_5 on five elements, whose program lengths are exponential in the depth of the circuits. The main idea behind the simulation in Barrington's proof is that the symmetric group S_5 is non-solvable, i.e., there are cyclic permutations σ and τ (composed of a single cycle) such that $\sigma\tau\sigma^{-1}\tau^{-1}$ is also a cyclic permutation and thus is not equal to the identity 1. This property is used to simulate AND-gates. The simulation of NOT-gates is straightforward, and OR-gates are done by AND-gates and negations using DeMorgan's Law. For further details the interested reader is referred to [7]. Programs over monoids are a straightforward generalization of the concept of recognizability. Here language $L \subseteq \Sigma^*$ is *recognizable* if and only if there exists a finite monoid M, a morphism $\varphi : \Sigma^* \to M$, and a subset $N \subseteq M$ such that $L = \varphi^{-1}(N)$. In other words, an input word $w = a_1 a_2 \dots a_n$ is translated *via* the morphism φ to the word $\varphi(a_1)\varphi(a_2)\dots\varphi(a_n)$ of monoid elements, that is evaluated in the monoid to yield a value whose N membership is tested. A *program over a monoid M*, for short *M-program*, takes an input word $a_1 a_2 \dots a_n$ of length n and transforms it into a word $\sigma = \sigma_1\sigma_2\dots\sigma_m$, for some m, over monoid elements by querying the input positions in arbitrary order and transforming the read letter into a monoid element, which is multiplied in the monoid to yield the value of the program. More formally, an M-program for input of length n, is a sequence of instructions $P_n = \langle i_1, \varphi_1\rangle\langle i_2, \varphi_2\rangle \dots \langle i_m, \varphi_m\rangle$, where for each j with $1 \le j \le m$ we have $1 \le i_j \le n$ and $\varphi_j : \Sigma \to M$, and an accepting subset $N \subseteq M$. On input $w = a_1 a_2 \dots a_n$, the program produces the word $\varphi(w) = \varphi_1(a_{i_1})\varphi_2(a_{i_2})\dots\varphi_m(a_{i_m})$ of monoid elements, that are multiplied in M and whose N membership determines whether the input is recognized or not, i.e., the word w is recognized if and only if $\varphi(w)$ is in N. The words recognized by the M-program P_n define a language $L \subseteq \Sigma^n$ in the obvious way.

Thus, Barrinton's result shows that the word problem over the symmetric group S_5 is already NC^1-hard and moreover, one can state the following result on fixed membership for regular languages. The next statement on the NC^1-completeness of the fixed membership problem for regular languages should be understood in the way that checking membership for a fixed regular language can be done in NC^1, and that there is at least one regular language for which it is NC^1-hard.

Theorem 20 (Fixed membership)**.** *The fixed membership problem for regular languages is NC^1-complete with respect to constant depth reducibilities.*

The strong algebraic background on this result has triggered further studies on M-programs over monoids satisfying certain restrictions. For instance, one of the best investigated restriction is aperiodicity. Here a monoid is *aperiodic* if and only if all elements x from the monoid satisfy $x^t = x^{t+1}$, for some $t \ge 0$. It is well known that a language L has a aperiodic syntactic monoid if and only if L is *star-free*, i.e., it can be obtained from the elementary languages $\{a\}$, for $a \in \Sigma$, by applying Boolean operations and finitely many concatenations, where complementation is with respect to Σ^*. These languages are exhaustively studied in, e.g. [69] and [78]. We mention in passing that first it was shown in [81] that the aperiodicity problem (given a finite automaton with input alphabet Σ, does it accept an aperiodic or star-free language?) for DFAs is coNP-hard and belongs to PSPACE. Later this result was improved to PSPACE-completeness [15]. In fact, using some algebraic background developed in [85] on the parameterization of aperiodic and solvable monoids one can show the following result [9].

Theorem 21 (Fixed membership)

1. *The fixed membership problem for regular languages recognized by aperiodic monoids belongs to AC^0.*
2. *The fixed membership problem for regular languages recognized by solvable monoids belongs to ACC^0.*

A closer look reveals that one can obtain even more, namely a tight connection between the parameterization of AC^0 in terms of circuit depth k and a parameterization of aperiodic monoids, namely Brzozowski's dot-depth hierarchy [19,22]. Here the class D^k of dot-depth k languages is the union of the classes D_m^k inductively defined as

$$D_m^0 = \textsc{Bool}(\{\Sigma^*\}),$$
$$D_m^1 = \textsc{Bool}(\{\ \Sigma^* a \Sigma^* \mid a \in \Sigma\ \})$$

and

$$D_m^k = \textsc{Bool}(\{\ L_0 a_1 L_1 \dots a_r L_r \mid a_i \in \Sigma, L_i \in D_m^{k-1}, \text{and } r \le m\ \}),$$

for $k \geq 2$, where $\text{BOOL}(\cdot)$ refers to the Boolean closure. Finally, a monoid has dot-depth k if all the languages recognized by it belong to D^k. Instead of using a divide-and-conquer approach as for regular languages in general, the inductive definition of dot-depth k monoids or languages allows a straightforward decomposition of language membership that gives a one-to-one correspondence between dot-depth and circuit depth. By simple induction one can show that any language in D_m^k belongs to AC^0 of depth k, using unbounded AND- and OR-gates to search over the possible partitions of the input—compare with the definition of D_m^k given above. The result proven in [9] reads as follows:

Theorem 22 (Fixed membership)**.** *The fixed membership problem for regular languages recognized by dot-depth k monoids is solvable in* AC^0 *by a family of depth k circuits.*

Since the AC^0-hierarchy is strict, the fixed membership problem for regular languages recognized by dot-depth k monoids nicely parametrizes this strict hierarchy. It is worth mentioning, that there exists a non-trivial formal language family, with an even easier fixed membership problem than the aperiodic regular languages, namely the family of D0L languages. This language family is well studied in the literature [77], and in [20,21] it was shown that the fixed membership problem for D0L languages is solvable in AC^0 by a family of depth k circuits, for some constant k.

3.2. Emptiness, universality, equivalence, and related problems

In this subsection we consider non-emptiness, universality, equivalence, and some related problems such as intersection emptiness or bounded universality or equivalence, for finite automata in more detail. Obviously, these standard problems are related to each other and we will briefly discuss their relations and moreover some consequences to the complexity of some non-trivial properties for problems on DFAs, NFAs, and AFAs. The *non-emptiness problem for NFAs* is defined as follows:

- Given a nondeterministic finite automaton A, is $L(A) \neq \emptyset$?

Moreover, the *universality problem for NFAs* is:

- Given a nondeterministic finite automaton A with input alphabet Σ, is the language $L(A)$ universal, i.e., $L(A) = \Sigma^*$?

The *equivalence problem for NFAs* is defined for two devices as:

- Given two nondeterministic finite automata A_1 and A_2, is $L(A_1) = L(A_2)$?

This notation naturally generalizes to other types of finite automata.

Intuitively, the universality problem can be much harder than the corresponding emptiness problem, which may also be true for the equivalence problem and the universality problem. For instance, it is easy to see that emptiness reduces to non-universality if the automata class are logspace effectively closed under arbitrary homomorphism and concatenation with regular languages. Here a class of automata C is logspace effectively closed under arbitrary homomorphism, if for any automaton A from C with n states and any homomorphism h, one can construct within deterministic logspace an automaton B from C that accepts language $h(L(A))$. This implies that the number of states of B is bounded by some polynomial $p_R(n)$. Similarly logspace effective closure under concatenation with regular languages is defined. More general conditions for logspace many-one reductions of universality or emptiness to equivalence, where one of the languages is a fixed language, were studied in detail in a series of papers by Hunt III and co-authors [47–49].

Now let us come to the complexity of the emptiness problem for finite automata. In general, if automata are logspace effectively closed under intersection with regular sets, then the general membership logspace many-one reduces to the non-emptiness problem for the same type of automata, because $w \in L(A)$ if and only if $L(A) \cap \{w\} \neq \emptyset$. Conversely, non-emptiness logspace many-one reduces to general membership, if the automata are logspace effectively closed under homomorphism, since $L(A) \neq \emptyset$ if and only if $h(L(A)) \neq \emptyset$ if and only if $\lambda \in h(L(A))$, where $h(a) = \lambda$, for $a \in \Sigma$. In [57] the following result on the non-emptiness problem for NFAs was shown, which even holds for DFAs—since nondeterministic space complexity classes are closed under complementation [53,84] the result also holds for the emptiness problem. Moreover, non-emptiness for AFAs was considered in [55] and [36].

Theorem 23 (Non-emptiness)**.** *The non-emptiness problem for NFAs and DFAs is* NL*-complete, and it is* PSPACE*-complete for AFAs. The results remain valid for automata accepting unary languages, except for DFAs accepting unary languages, whose non-emptiness problem becomes* L*-complete.*

A natural variant of non-emptiness is *intersection non-emptiness*. This is the problem to decide whether $\bigcap_{1 \leq i \leq n} L(A_i) \neq \emptyset$, for given finite automata $A_1, A_2, \ldots, A_n$. If the *number* of automata in the input instance is bounded by some function $g(n)$, where n denotes the length of the description of the $g(n)$ many finite automata, then this problem is referred to as the $g(n)$*-bounded intersection non-emptiness problem*. For easier readability we abbreviate the former problem by $\emptyset \neq \bigcap C$ and

the latter one by $\emptyset \neq \bigcap^{g(n)} C$, where C is from {DFA, NFA, AFA}. Trivially, non-emptiness logspace many-one reduces to intersection non-emptiness, even to k-bounded intersection non-emptiness for constant k. The results on these problems read as follows: In [60] it was shown that $\emptyset \neq \bigcap$ DFA is PSPACE-complete. Since $\emptyset \neq \bigcap$ AFA can be decided within nondeterministic polynomial space, $\emptyset \neq \bigcap$ NFA and $\emptyset \neq \bigcap$ AFA are PSPACE-complete, too. Recently it was shown in [6] that the *infinite cardinality intersection problem*, i.e., given automata $A_1, A_2, \ldots, A_n$ from C, do there exist infinitely many words in $\bigcap_{1 \leq i \leq n} L(A_i)$, is also PSPACE-complete for DFAs. Further PSPACE-complete problems on NFAs based on pattern and power acceptance were identified in [6].

For DFAs and NFAs these intractable intersection emptiness problems become feasible, if the number of finite automata is bounded by some constant k, but remains intractable for AFAs. More precisely, both the k-bounded intersection non-emptiness problems $\emptyset \neq \bigcap^k$ DFA and $\emptyset \neq \bigcap^k$ NFA are NL-complete, for each k with $k \geq 1$ [25] and $\emptyset \neq \bigcap^k$ AFA remains obviously PSPACE-complete. Moreover, for the bounded intersection non-emptiness problem in general it was shown in [64] that both $\emptyset \neq \bigcap^{g(n)}$ DFA and $\emptyset \neq \bigcap^{g(n)}$ NFA are complete for $\mathrm{NSPACE}(g(n) \cdot \log n)$. In particular, both $\emptyset \neq \bigcap^{\log^{k-1} n}$ DFA and $\emptyset \neq \bigcap^{\log^{k-1} n}$ NFA are $\mathrm{NSPACE}(\log^k n)$-complete, for $k \geq 1$. Observe, that these were the first natural complete problems for these complexity classes. Finally, what can be said about the (bounded) intersection non-emptiness problem for the automata under consideration, when restricted to a unary [1] input alphabet? As a consequence of [25] and [83] both $\emptyset \neq \bigcap$ Tally-DFA and $\emptyset \neq \bigcap$ Tally-NFA are NP-complete, while $\emptyset \neq \bigcap$ Tally-AFA again remains PSPACE-complete [36]—the abbreviations of the problem instances are self-explaining. The latter result also holds for the bounded variant, even for constant k. In [64] it is briefly mentioned that $\emptyset \neq \bigcap^k$ Tally-DFA is L- and $\emptyset \neq \bigcap^k$ Tally-NFA is NL-complete. On the other hand, *completeness* results for the bounded intersection non-emptiness problem are not known for unary languages, as in the general case. Nevertheless, involved upper and lower bounds by simultaneously bounded complexity classes (time, space, and number of nondeterministic steps) were shown in [64].

Another problem closely connected to non-emptiness is the so-called *short word problem*, which was investigated in [64], too. The main idea underlying short words is that in general the shortest word accepted by an AFA can be of exponential length in the coding of this automaton (cf. Example 2). Thus, the natural question arises whether the automaton accepts words which are "short" in some sense. Regarding words of linear length as short, we can define the *short word problem* as follows: given a finite automaton A, is there a word w of length less than or equal to the coding of A, such that $w \in L(A)$? The short word problem (as long as not combined with some further restriction) is only interesting for AFAs, since the linear restriction does not change the complexity of the non-emptiness problem for languages given by DFAs and NFAs (even for automata accepting unary languages only), as the shortest word accepted or rejected is always linear in the number of states. Thus we abbreviate the short word problem for AFAs by $\emptyset \neq \mathrm{AFA}_{lin}$ and $\emptyset \neq$ Tally-AFA_{lin} when the automata accept unary languages. It was shown shown in [36] that $\emptyset \neq \mathrm{AFA}_{lin}$ is NP- and $\emptyset \neq$ Tally-AFA_{lin} is P-complete. Considering the combination of the (bounded) intersection non-emptiness problem with the short word restriction leads to more interesting results. We refer to these problems as $\emptyset \neq \bigcap^{g(n)} C_{lin}$ and $\emptyset \neq \bigcap^{g(n)}$ Tally-C_{lin}, respectively. The problems $\emptyset \neq \bigcap^{g(n)} \mathrm{DFA}_{lin}$ and $\emptyset \neq \bigcap^{g(n)} \mathrm{NFA}_{lin}$ are complete for simultaneously time and space bounded classes between $\mathrm{NTISP}(\mathrm{pol}\, n, \log n) = \mathrm{NL}$ and $\mathrm{NTISP}(\mathrm{pol}\, n, \mathrm{pol}\, n) = \mathrm{NP}$, namely for $\mathrm{NTISP}(\mathrm{pol}\, n, g(n) \cdot \log n)$—see [64]. For $g(n) = \log^k n$ these classes are the nondeterministic counterparts of the SC^k-hierarchy. The restriction of these problems with respect to DFAs (NFAs, respectively) to short words, always leads to L-complete (NL-complete, respectively) sets, regardless of the function $g(n)$. The corresponding problems for AFAs, namely $\emptyset \neq \bigcap^{g(n)} \mathrm{AFA}_{lin}$ and $\emptyset \neq \bigcap^{g(n)}$ Tally-AFA_{lin} are P-complete, also regardless of $g(n)$.

Now let us consider the next standard problem, the universality problem. As previously mentioned, emptiness and universality are closely related to each other by the complementation operation. The universality problem for DFAs was shown to be NL-complete [16]. For NFAs and AFAs, respectively, the problem under consideration was investigated in [2,72] and [36], respectively, in more detail. For the results on automata accepting unary languages we refer to [83] and [36]. We summarize these results in the following theorem.

Theorem 24 (Universality). *The universality problem for DFAs is* NL*-complete, and for NFAs and AFAs it is* PSPACE*-complete. For automata accepting unary languages, the universality problem is* L*-complete for DFAs,* coNP*-complete for NFAs, and* PSPACE*-complete for AFAs.*

Recently it was shown in [58] that the universality problem for NFAs remains PSPACE-complete even if all states are final or both initial and final, respectively. Observe that finite automata having only final states accept prefix closed languages, while finite automata where all states are both initial and final accept infix closed languages. In fact, prefix and infix closed languages can be characterized by these properties on finite automata.

Next we consider two variants of universality. The first one is the *union universality problem*, that is to decide for given automata $A_1, A_2, \ldots, A_n$, whether $\bigcup_{1 \leq i \leq n} L(A_i) = \Sigma^*$? Trivially, universality logspace many-one reduces to the union uni-

[1] Unary languages are also sometimes called tally languages. While the former term "unary" is mostly used in the literature on automata and formal language theory, but not exclusively, the latter phrase "tally" is more commonly used in complexity theory. Therefore, the problem notation used in the literature involving automata accepting unary languages mostly use the word tally instead of unary. We will not deviate from this and refer by, e.g. Tally-DFA to DFAs accepting unary languages only. We do similar for the other automata classes considered.

versality problem for any class of automata. For DFAs this problem is readily seen to be PSPACE-complete by a reduction from the intersection emptiness problem for DFAs, which was discussed in detail earlier. For NFAs and AFAs the union universality problem is PSPACE-complete, too, since it is already PSPACE-hard for a single automaton, and containment can easily be seen since NFAs and AFAs are logspace effective closed under union. Thus, further variants of this problem, comparable to variants of the intersection emptiness problem, are not worth studying. Another, not so well-known generalization of the universality problem is the bounded universality problem first studied in [16]. The *bounded universality problem* is the problem of deciding for a given finite automaton A and a unary integer n, whether $L(A) \cap \Sigma^{\leq n} = \Sigma^{\leq n}$? The *bounded non-universality problem* is defined accordingly. In [16] it was shown that the bounded universality problem for NFAs is coNP-complete, while it is NL-complete for DFAs. Thus, the complexity of non-bounded universality is significantly lower than that of the equivalence problem, which is discussed below. Regarding the problem of computing the lexically first witness string that proves bounded non-universality for NFAs, an Δ_2^{P} upper bound, and NP- and coNP-hardness lower bounds were shown in [16]. Computing any witness string, thus dropping the lexically first criterion, leads to a problem that is computationally equivalent to the bounded non-universality problem and, thus, is an NP-complete problem for NFAs. For AFAs the bounded universality is seen to be coNP-complete.

The last standard problem we are interested in, is the equivalence problem. Besides the emptiness problem, the equivalence problem is certainly one of the most important decision problems that has been investigated extensively in the literature. That equivalence is harder than emptiness is (partially) true for DFAs and NFAs, because equivalence is NL-complete for deterministic [16] and PSPACE-complete for NFAs. However, in case of AFAs equivalence remains as hard as emptiness as shown in [55]. Automata on a unary input alphabet were investigated in [36,83]. As the reader may notice, universality and equivalence are computational equivalent with respect to logspace many-one reductions for the finite automata types under consideration.

Theorem 25 (Equivalence)**.** *The equivalence problem for DFAs is* NL*-complete, and for NFAs and AFAs it is* PSPACE*-complete. For automata accepting unary languages, the equivalence problem is* L*-complete for DFAs,* coNP*-complete for NFAs, and* PSPACE*-complete for AFAs.*

Most of the presented results on emptiness, universality, and equivalence date back to the pioneering papers [72,82,83] and [35,46–49,80], where mostly problems on regular-like expressions were investigated. Obviously, a lower bound on the computational complexity of a problem for ordinary regular expressions implies the same lower bound for NFAs, since any regular expression of size n can be converted into an equivalent $(n+1)$-state NFA [52]. Most of these results on regular expressions are summarized in [26]—for instance, one can read the following entry, literally taken from [26], on inequivalence for regular expressions:

> "[The inequivalence for regular expressions r_1 and r_2, i.e., deciding whether $L(r_1) \neq L(r_2)$, is ...] PSPACE-complete, even if $|\Sigma| = 2$ and $L(r_2) = \Sigma^*$. In fact, PSPACE-complete if r_2 is any fixed expression representing an "unbounded" language [49]. NP-complete for fixed r_2 representing any infinite "bounded" language, but solvable in polynomial time for fixed r_2 representing any finite language. The general problem remains PSPACE-complete if r_1 and r_2 both have "star-height" k for fixed $k \geq 1$ [49], but is NP-complete for $k = 0$ ("star-free") [44,83]. Also NP-complete if one of both of r_1 and r_2 represent bounded languages (a property that can be checked in polynomial time) [49] or if $|\Sigma| = 1$ [83]. For related results and intractable generalizations, see cited references, [45], and [48]."

Here a language L is *bounded* if and only if there exist words $w_1, w_2, \ldots, w_k$, for some k, such that $L \subseteq w_1^* w_2^* \ldots w_k^*$. In [51] it was shown that boundedness is a necessary and sufficient condition for context-free languages to be sparse. A language $L \subseteq \Sigma^*$ is *sparse*, if there exists a polynomial p such that for all n we have $|L \cap \Sigma^{\leq n}| \leq p(n)$, where $\Sigma^{\leq n}$ is the set of all words over Σ of length at most n. While boundedness for languages specified by regular expressions is easily shown to be solvable in polynomial time *via* an inductive proof [49], it is not that clear, whether this also holds for NFAs. Here the equivalence of boundedness and sparseness for context-free languages comes into play. The *sparseness problem*, i.e., given an automaton A, is $L(A)$ sparse?, was shown to be NL-complete for both DFAs and NFAs [50], and for AFAs it is PSPACE-complete. For automata accepting unary languages the problem under consideration is trivial. Hence, the boundedness problem for NFAs is efficiently solvable.

Next we summarize some results on some problems related to universality and equivalence, namely the segment equivalence and the closeness problem:

1. The *segment equivalence problem* is defined as follows: given two automata A_1 and A_2 and n, is $L(A_1) \cap \Sigma^{\leq n} = L(A_2) \cap \Sigma^{\leq n}$? If n is coded in binary, it is called the *binary-encoded segment equivalence problem*. Segment and binary-encoded segment equivalence were studied in [50]. There it was shown that segment equivalence for DFAs is NL-complete, whereas for NFAs the problem becomes coNP-complete. As in case of ordinary equivalence one can show that the complexity of segment equivalence for AFAs is the same as for NFAs, hence coNP-complete, if the input alphabet contains at least two letters. For automata accepting unary languages it is easy to see that the segment equivalence problem is L-complete for DFAs, NL-complete for NFAs, and P-complete for AFAs. Moreover, for the

binary-encoded segment equivalence problem it was shown that both NFAs and AFAs induce a PSPACE-complete problem [50].

2. The closeness problem measures the similarity of languages in terms of the density of their symmetric difference, i.e., two languages L_1 and L_2 are *close* if and only if $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is sparse. Thus, the *closeness problem* is to decide whether for given two automata A_1 and A_2, the symmetric difference of $L(A_1)$ and $L(A_2)$ is sparse. In [50] it was shown that the closeness problem for DFAs is NL-complete and for NFAs it is PSPACE-complete. Moreover, PSPACE-completeness also holds for the closeness problem for AFAs. Note, that the closeness problem for automata accepting unary languages is trivial.

Along the lines of development in computational complexity theory, authors began to study functional problems and classes, see, e.g. [3,59,86]. One of the easiest functional problems for finite state devices is *census*. Here for a given finite automaton A and 1^n, one asks how many words up to length n are accepted by A? Other well-known functional problems are *census of the complement*, *ranking*, *maximal word*, and *maximal relative word*—for a precise definition of these problems we refer to [3]. For DFAs and NFAs it was shown in [3] that most of these problems are complete for logarithmic space bounded counting classes like #L, spanL, or optL, while for AFAs these problems turn out to be complete for their polynomially time bounded counterparts #P or optP [36].

We have seen that most problems for NFAs and AFAs are intractable, while some problems for DFAs are effectively solvable. In the remainder of this subsection we consider two results of Hunt III and co-authors [35,48], which show that the above mentioned behavior on the computational complexity of DFA and NFA based problems is not accidental. It thus explains in part, why most problems for NFAs and AFAs are intractable. We feel that these nice results demand more attention, therefore we present them here. The results of [35,48] given here parallel Greibach's well-known theorem on the undecidability of certain properties for context-free languages [31]. They are slightly adapted and read in our notation as follows:

Theorem 26 (Hardness for DFA and NFA problems)**.** *Let Σ be an alphabet with $|\Sigma| \geq 2$ and $P : 2^{\Sigma^*} \to \{0, 1\}$ be any non-trivial predicate on the regular languages. Assume that the set P_{left} of all languages $\delta_x(L)$, where L is a regular language, $P(L)$ is true, and $x \in \Sigma^*$, is not equal to the family of all regular languages—here $\delta_x(L) = \{ y \in \Sigma^* \mid xy \in L \}$ refers to the left quotient of L with respect to the word x from Σ^*. Then the P-problem for NFAs, that is, to determine whether for a given nondeterministic finite automaton A the predicate P on $L(A)$ is true, is* PSPACE*-hard, assuming $P(\Sigma^*)$ to be true. Moreover, the corresponding P-problem for DFAs is at least* NL*-hard.*

For NFAs we recall the astonishing simple proof of this result—for DFAs the construction is more involved, because DFAs are not logspace effective closed under union in general. Without loss of generality assume that the input alphabet of the finite automata is $\Sigma = \{a, b\}$. Let L_f be a regular language, which is not a member of P_{left}. Define the homomorphism $h(a) = aa$ and $h(b) = ab$ and set $L_1 = (\{aa, ab\}^* \cdot ba \cdot L_f) \cup (\Sigma^* \setminus (\{aa, ab\}^* \cdot ba \cdot \Sigma^*))$. Then for a given NFA A we construct automaton B of the same type satisfying $L(B) = h(L(A)) \cdot ba \cdot \Sigma^* \cup L_1$. Then we claim that $P(L(B))$ is *true* if and only if $L(A) = \Sigma^*$. We distinguish two cases:

1. If $L(A) = \Sigma^*$, then $L(B) = \Sigma^*$ and by assuMption $P(L(B))$ is *true*.
2. On the other hand, if $L(A) \subset \Sigma^*$, then there is a word x in $\Sigma^* \setminus L(A)$. But then it is easy to see that $\delta_{h(x)\cdot ba}(L(B)) = L_f$, which was chosen not to satisfy predicate P_{left}, i.e., $P(L(B))$ is *false*.

This proves the stated claim.

The result of the above given theorem can be used to obtain different characterizations of the famous "LBA problem" [61], that is, the question whether deterministic and nondeterministic linear space bounded Turing machines are computational equivalent? For further reading on the LBA problem we refer to, e.g. [35,61,73]. Moreover, in [48] an extensive list of PSPACE-hard properties is provided. Examples are problems associated with various subclasses of regular languages such as, e.g. deciding whether a given automaton accepts (i) a prefix-, suffix-, or infix-closed language, (ii) variants of definite languages, (iii) variants of comet languages, and (iv) variants of testable languages, etc. Recently in [12] generalizations of decision problems related to those mentioned in (i), namely variants of convex languages were studied from the computational complexity point of view, obtaining PSPACE-completeness results for NFAs.

Finally, we summarize some results on the operation problem from the computational complexity perspective. For a survey on the descriptional complexity of the operation problem for DFAs and NFAs we refer to [87] and [37,38]. Let $\circ$ be a fixed operation on languages that preserves regularity. Then the $\circ$-*operation problem* is defined as follows: given finite automata A_1, A_2, and A_3, is $L(A_1) \circ L(A_2) = L(A_3)$? Obviously, this problem generalizes to unary language operations. It turned out that both the concatenation operation problem and the Kleene star operation for DFAs are PSPACE-complete [56]. A converse problem to the $\circ$-operation problem is the *minimum* $\circ$-*problem*. That is, given a finite automaton A and an integer k, are there finite automata A_1 and A_2 of the same type with $|A_1| + |A_2| \leq k$ such that $L(A_1) \circ L(A_2) = L(A)$? For DFAs this problem is NP-complete for union and intersection, and PSPACE-complete for concatenation and Kleene star. Interestingly, the minimum reverse-operation problem is shown to be solvable in polynomial time if the integer k is given in unary, although DFAs are not logspace effective closed under reversal. The latter problem is associated to the *diversity*

problem for DFAs defined in [76]. The diversity is the number of equivalence classes induced by the relation $\equiv_A$ of the DFA A with states Q and transition function δ, which is $x \equiv_A y$ if and only if for all states q in Q we have $\delta(q, x)$ is accepting if and only if $\delta(q, y)$ is. These problems are closely related to descriptional complexity issues discussed so far.

3.3. Minimization of finite automata

The study of the minimization problem for finite automata dates back to the early beginnings of automata theory. Here we focus mainly on some recent developments related to this fundamental problem—for further reading we refer to [56] and references therein. The minimization problem is also of practical relevance, because regular languages are used in many applications, and one may like to represent the languages succinctly. The decision version of the minimization problem, for short the *NFA-to-NFA minimization problem*, is defined as follows:

- Given a *nondeterministic* finite automaton A and a natural number k in binary, that is, an encoding $\langle A, k\rangle$, is there an equivalent k-state *nondeterministic* finite automaton?

This notation naturally generalizes to other types of finite automata, for example, the DFA-to-NFA minimization problem. It is well known that for a given n-state DFA one can efficiently compute an equivalent minimal automaton in $O(n \log n)$ time [40]. More precisely, the DFA-to-DFA minimization problem is complete for NL, even for DFAs without inaccessible states [16]. This is contrary to the nondeterministic case since the NFAs minimization problem is known to be computationally hard [56], which is also true for AFAs. The PSPACE-hardness result for NFAs was shown by a reduction from the union universality problem to the NFA-to-NFA minimization problem. For some further problems related to minimization we refer also to [32].

Theorem 27 (Minimization). *The DFA-to-DFA minimization problem is* NL*-complete, while the NFA-to-NFA minimization problem is* PSPACE*-complete, even if the input is given as a deterministic finite automaton. The AFA-to-AFA minimization problem is* PSPACE*-complete, too.*

In order to better understand the very nature of nondeterminism one may ask for minimization problems for restricted types of finite automata. Already in [56] it was shown that for the restricted class of unambiguous finite automata (UFA) some minimization problems remain intractable. An NFA is said to be *unambiguous* if for every word that belongs to the accepted language there is at most one accepting computation. To be more precise, the UFA-to-UFA and the DFA-to-UFA minimization problems are NP-complete. We mention in passing that in [16] necessary and sufficient conditions were provided to distinguish between exponential, polynomial, bounded, and k-bounded ambiguity, and it was shown that these *ambiguity problems*, i.e., determining whether the degree of ambiguity of a given NFA is exponential, polynomial, bounded, k-bounded, where k is a fixed integer, or unambiguous are all NL-complete.

Later in [67] it was shown that the minimization of finite automata equipped with a very small amount of nondeterminism is already computationally hard. To this end, a reduction from the NP-complete minimal inferred DFA problem [28,56] to the the minimization problems for multiple initial state deterministic finite automata with a fixed number of initial states (MDFA) as well as for nondeterministic finite automata with fixed finite branching has been shown. Prior to this, the MDFA-to-DFA minimization problem in general was proven to be PSPACE-complete in [39]. Here the minimal inferred DFA problem [28] is defined as follows: given a finite alphabet Σ, two finite subsets $S, T \subseteq \Sigma^*$, and an integer k, is there a k-state DFA that accepts a language L such that $S \subseteq L$ and $T \subseteq \Sigma^* \setminus L$? Such an automaton can be seen as a consistent "implementation" of the sets S and T. Recently, the picture was completed in [11] by getting much closer to the tractability frontier for nondeterministic finite automata minimization. There a class of NFAs is identified, the so called δ-nondeterministic finite automata (δNFA), such that the minimization problem for any class of finite automata that contains δNFAs is NP-hard, even if the input is given as a DFA. Here the class of δNFAs contains all NFAs with the following properties: (i) the automaton is unambiguous, (ii) the maximal product of the degrees of nondeterminism over the states in a possible computation is at most 2, and (iii) there is at most one state q and a letter a such that the degree of nondeterminism of q and a is 2. It is worth mentioning that for every n-state δNFA there is an equivalent DFA with at most $O(n^2)$ states.

The situation for the minimization problem in general is, in fact, even worse. Recent work [30] shows that the DFA-to-NFA problem cannot be approximated within a factor of $\sqrt{n}/\text{polylog}n$ for state minimization and $n/\text{polylog}n$ for transition minimization, provided some cryptographic assumption holds. Moreover, the NFA-to-NFA minimization problem was classified to be inapproximable within $o(n)$, unless $\mathsf{P} = \mathsf{PSPACE}$, if the input is given as an NFA with n states [30]. That is, no polynomial-time algorithm can determine an approximate solution of size $o(n)$ times the optimum size. Even the DFA-to-NFA minimization problem remains inapproximable within a factor of at least $n^{1/3-\epsilon}$, for all $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ [34], for alphabets of size $O(n)$, and not approximable within $n^{1/5-\epsilon}$ for a binary alphabet, for all $\epsilon > 0$. Under the same assumption, it was shown that the transition minimization problem for binary input alphabets is not approximable within $n^{1/5-\epsilon}$, for all $\epsilon > 0$. The results in [34] proved approximation hardness results under weaker (and more familiar) assumptions than [30]. Further results on the approximability of the minimization problem when the input is specified as regular expression or a truth table can be found in [30,34].

For finite languages, NFA-to-NFA minimization can be done by the following algorithm: anondeterministic Turing machine with an NFA equivalence oracle for finite languages can guess an NFA with at most k states, and ask the oracle whether the

guessed automaton is equivalent to the input automaton, and accept if and only if the oracle answer is yes. Since equivalence for finite languages specified by NFA is coNP-complete [83], the minimization problem belongs to Σ_2^P, regardless of whether a deterministic or nondeterministic finite state device is given. Recently, the NFA-to-NFA minimization problem for finite languages was shown to be DP-hard, even if the input is a DFA accepting a finite language. This improved the previously known NP-hardness result, which follows from [5]. The complexity class DP includes both NP and coNP, and is a subset of Σ_2^P. This nicely contrasts with a recent result on the NP-completeness of minimization for finite languages given by truth tables [33]. Hence, the DFA-to-NFA minimization problem for finite languages is more complicated than that with truth tables as input, unless NP = coNP. Whether this lower bound can be substantially raised to, for example Σ_2^P-hardness, is open.

The unary NFA-to-NFA minimization problem is coNP-hard [83], and similarly as in the case of finite languages contained in Σ_2^P. The number of states of a minimal NFA equivalent to a given unary cyclic DFA cannot be computed in polynomial time, unless NP $\subseteq$ DTIME($n^{O(\log n)}$) [54]. Note that in the latter case the corresponding decision version belongs to NP. Inapproximability results for the problem in question have been found during the last years, if the input is a unary NFA: The problem cannot be approximated within $\sqrt{n}/\ln n$ [29], and if one requires in addition the explicit construction of an equivalent NFA, the inapproximability ratio can be raised to $n^{1-\epsilon}$, for every $\epsilon > 0$, unless P = NP [30]. On the other hand, if a unary *cyclic* DFA with n states is given, the nondeterministic state complexity of the considered language can be approximated within a factor of $O(\log n)$. The picture on the unary NFA-to-NFA minimization problem was completed in [33]. Some of the aforementioned (in)approximability results, which only hold for the *cyclic* case, generalize to unary languages in general. In particular, it was shown that for a given n-state NFA accepting a unary language, it is impossible to approximate the nondeterministic state complexity within $o(n)$, unless P = NP. Observe that this bound is tight. In contrast, it is proven that the NFA-to-NFA minimization problem can be *constructively* approximated within $O(\sqrt{n})$, where n is the number of states of the given DFA. Here by *constructively approximated* we mean that we can build the nondeterministic finite automaton, instead of only approximately determining the number of states needed. This solves an open problem stated in [56] on the complexity of converting a DFA to an approximately optimal NFA in the case of unary languages.

Estimating the size, in terms of the number of states, of a minimal NFA for a regular language is stated as an open problem in [1]; see also, e.g. [43]. It has been shown, that upper or lower bounds on the state size of minimal NFAs with a guaranteed relative error better than $\sqrt{n}/\text{poly}(\log(n))$ cannot be obtained in polynomial time, provided some cryptographic assumption holds [30]. Several authors have introduced methods for proving NFA state lower bounds; see, e.g. the fooling set [27], the extended fooling set [10,42], and the biclique edge cover technique [32]. These lower bound techniques read as follows:

Theorem 28 (NFA lower bound techniques). *Let $L \subseteq \Sigma^*$ be a regular language and suppose there exists a set of pairs $S = \{ (x_i, y_i) \mid 1 \leq i \leq n \}$.*

Fooling set and extended fooling set technique: *If $x_i y_i \in L$ for $1 \leq i \leq n$, and $x_i y_j \notin L$, for $1 \leq i, j \leq n$, and $i \neq j$, then any NFA accepting L has at least n states. Here S is called a fooling set for L. The statement remains valid if the latter condition is replaced by $i \neq j$ implies $x_i y_j \notin L$ or $x_j y_i \notin L$, for $1 \leq i, j \leq n$. In this case, S is called an extended fooling set for L.*

Biclique edge cover technique: *Let $X = \{ x_i \mid \exists y_i : (x_i, y_i) \in S \}$ and set $Y = \{ y_i \mid \exists x_i : (x_i, y_i) \in S \}$. If the bipartite graph $G = (X, Y, E_L)$ with edge set $E_L = \{ (x, y) \in X \times Y \mid xy \in L \}$ has bipartite dimension d, then any NFA accepting language L has at least d states. Here the bipartite dimension of a bipartite graph G is the size of the smallest biclique edge cover, that is an edge cover of G by bicliques only, if it exists and is infinite otherwise.*

Although the bounds provided by these techniques are not always tight and in fact can be arbitrarily worse compared to the nondeterministic state complexity, they give good results in many cases. The corresponding decision problems on the aforementioned lower bound techniques are defined as follows:

- The *fooling set problem* asks, whether for a given finite automaton A and a natural number k in binary, there is a fooling set S for the language $L(A)$ of size at least k?

The *extended fooling set* and the *biclique edge cover problem* are analogously defined. While the fooling set problem for DFAs is NP-hard and contained in PSPACE, the extended fooling set and the biclique edge cover problem for DFAs is PSPACE-complete [32]. The complexity of the fooling set and the extended fooling set problem does not increase if the regular language is specified as an NFA. The proofs for the upper bounds on the complexity carry over to this setup, too. Currently, we do not know whether this also holds true for the biclique edge cover problem for NFAs. The best upper bound we are aware of is coNEXPTIME, obtained by an explicit construction.

References

[1] H.N. Adorna, Some descriptional complexity problems in finite automata theory, in: Philippine Computing Science Congress (PCSC 2005), Computing Society of the Philippines, 2005.
[2] A.V. Aho, J.E., Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974, pp. 27–32.

[3] C. Álvarez, B. Jenner, A very hard log-space counting class, Theoret. Comput. Sci. 107 (1993) 3–30.
[4] M. Ajtai, Σ_1^1 formulae on finite structures, Ann. Pure. Appl. Logic 24 (1983) 1–48.
[5] J. Amilhastre, P. Janssen, M.C. Vilarem, FA minimisation heuristics for a class of finite languages, Workshop on Implementing Automata (WIA 2001), LNCS, vol. 2214, Springer, 2001, pp. 1–12.
[6] T. Anderson, J. Loftus, N. Rampersad, N. Santean, J. Shallit, Detecting palindroms, patterns, and borders in regular languages, Inform. and Comput. 207 (2009) 1096–1118.
[7] D.A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, J. Comput. System Sci. 38 (1989) 150–164.
[8] D.A.M. Barrington, N. Immerman, H. Straubing, On uniformity within NC^1, J. Comput. System Sci. 41 (1990) 274–306.
[9] D.M. Barrington, D. Thérien, Finite monoids and the fine structure of NC^1, J. ACM 35 (1988) 941–952.
[10] J.C. Birget, Intersection and union of regular languages and state complexity, Inform. Process. Lett. 43 (1992) 85–190.
[11] H. Björklund, W. Martens, The tractability frontier for NFA minimization, International Colloqium on Automata, Languages and Programming (ICALP 2008), LNCS, vol. 5126, Springer, 2008, pp. 27–38.
[12] J. Brzozowski, J. Shallit, Z. Xu, Decision problems for convex languages, in: Language and Automata Theory and Applications (LATA 2009), LNCS, vol. 5457, Springer, 2009, pp. 247–258.
[13] J.A. Brzozowski, E.L. Leiss, On equations for regular languages, finite automata, and sequential networks, Theoret. Comput. Sci. 10 (1980) 19–35.
[14] A. Chandra, D. Kozen, L. Stockmeyer, Alternation, J. ACM 21 (1981) 114–133.
[15] S. Cho, D.T. Huynh, Finite-automaton aperiodicity is PSPACE-complete, Theoret. Comput. Sci. 88 (1991) 99–116.
[16] S. Cho, D.T. Huynh, The parallel complexity of finite-state automata problems, Inform. Comput. 97 (1992) 1–22.
[17] M. Chrobak, Finite automata and unary languages, Theoret. Comput. Sci. 47 (1986) 149–158.
[18] M. Chrobak, Errata to "finite automata and unary languages", Theoret. Comput. Sci. 302 (2003) 497–498.
[19] R.S. Cohen, J.A. Brzozowski, Dot-depth of star-free events, J. Comput. System Sci. 5 (1971) 1–16.
[20] C. Damm, M. Holzer, K.J. Lange, The parallel complexity of iterated morphisms and the arithmetic of small numbers, in: Mathematical Foundations of Computer Science (MFCS 1992), LNCS, vol. 629, Springer, 1992, pp. 227–235.
[21] C. Damm, M. Holzer, K.J. Lange, P. Rossmanith, Deterministic 0L Languages are of very Low Complexity: D0L is in AC^0, Developments in Language Theory (DLT 1994), World Scientific, 1994, pp. 305–313.
[22] S. Eilenberg, Automata, Languages, and Machines (Volume 59-B), Academic Press, 1976.
[23] A. Fellah, H. Jürgensen, S. Yu, Constructions for alternating finite automata, Int. J. Comput. Math. 35 (1990) 117–132.
[24] M.L. Furst, J.B. Saxe, M. Sipser, Parity, circuits, and the polynomial-time hierarchy, Math. Systems Theory 17 (1984) 13–27.
[25] Z. Galil, Hierarchies of complete problems, Acta Inform. 6 (1976) 77–88.
[26] M.R. Garey, D.S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, 1979.
[27] I. Glaister, J. Shallit, A lower bound technique for the size of nondeterministic finite automata, Inform. Process. Lett. 59 (1996) 75–77.
[28] E.M. Gold, Complexity of automaton identification from given data, Inform. Control 37 (1978) 302–320.
[29] G. Gramlich, Probabilistic and nondeterministic unary automata, Mathematical Foundations of Computer Science (MFCS 2003), LNCS, vol. 2747, Springer, 2003, pp. 460–469.
[30] G. Gramlich, G. Schnitger, Minimizing NFA's and regular expressions, in: Symposium on Theoretical Aspects of Computer Science (STACS 2005), LNCS, vol. 3404, Springer, 2005, pp. 399–411.
[31] S. Greibach, A note on undecidable properties of formal languages, Math. Systems Theory 2 (1) (1968) 1–6.
[32] H. Gruber, M. Holzer, Finding lower bounds for nondeterministic state complexity is hard (extended abstract), Developments in Language Theory (DLT 2006), LNCS, vol. 4036, Springer, 2006, pp. 363–374.
[33] H. Gruber, M. Holzer, Computational complexity of NFA minimization for finite and unary languages, in: Language and Automata Theory and Applications (LATA 2007), Technical Report 35/07, Universitat Rovira i Virgili, Tarragona, 2007, pp. 261–272.
[34] H. Gruber, M. Holzer, Inapproximability of nondeterministic state and transition complexity assuming $P \neq NP$, Developments in Language Theory (DLT 2007), LNCS, vol. 4588, 2007, Springer, pp. 205–216.
[35] J. Hartmanis, H.B. Hunt III, The LBA problem and its importance in the theory of computing, in: SIAM AMS Vol. 7 on Complexity of Computing, 1974, 1–26.
[36] M. Holzer, On Emptiness and Counting for Alternating Finite Automata, Developments in Language Theory (DLT 1996), World Scientific, 1996, pp. 88–97.
[37] M. Holzer, M. Kutrib, Nondeterministic descriptional complexity of regular languages, Int. J. Found. Comput. Sci. 14 (2003) 1087–1102.
[38] M. Holzer, M. Kutrib, Nondeterministic finite automata – recent results on the descriptional and computational complexity, Int. J. Found. Comput. Sci. 20 (2009) 563–580.
[39] M. Holzer, K. Salomaa, S. Yu, On the state complexity of k-entry deterministic finite automata, J. Autom. Lang. Comb. 6 (2001) 453–466.
[40] J. Hopcroft, An $n \log n$ Algorithm for Minimizing the State in a Finite Automaton, The Theory of Machines and Computations, Academic Press, 1971, pp. 189–196.
[41] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.
[42] J. Hromkovič, Communication Complexity and Parallel Computing, Springer, 1997.
[43] J. Hromkovič, G. Schnitger, On the hardness of determining small NFA's and of proving lower bounds on their sizes, in: Developments in Language Theory (DLT 2008), LNCS, vol. 5257, Springer, 2008, pp. 34–55.
[44] H.B. Hunt III, On the time and tape complexity of languages, Ph.D. thesis, Cornell University, Ithaca, New York, USA, 1973.
[45] H.B. Hunt III, On the time and tape complexity of languages I, in: Symposium on Theory of Computing (STOC 1973), ACM Press, 1973, pp. 10–19.
[46] H.B. Hunt III, Observations on the complexity of regular expressions problems, J. Comput. System Sci. 19 (1979) 222–236.
[47] H.B. Hunt III, D.J. Rosenkrantz, On equivalence and contaiment problems for formal languages, J. ACM 24 (1977) 387–396.
[48] H.B. Hunt III, D.J. Rosenkrantz, Computational parallels between the regular and context-free languages, SIAM J. Comput. 7 (1978) 99–114.
[49] H.B. Hunt III, D.J. Rosenkrantz, T.G. Szymanski, On the equivalence, containment, and covering problems for the regular and context-free languages, J. Comput. System Sci. 12 (1976) 222–268.
[50] D.T. Huynh, Complexity of Closeness Sparseness and Segment Equivalence for Context-free and Regular Languages, in: Informatik Festschrift zum 60, Geburtstag von Günter Hotz, Teubner, 1992, pp. 235–251.
[51] O.H. Ibarra, B. Ravikumar, On sparseness ambiguity and other decision problems for acceptors and transducers, in: Symposium on Theoretical Aspects of Computer Science (STACS 1986), LNCS, vol. 210, Springer, 1986, pp. 171–179.
[52] L. Ilie, S. Yu, Follow automata, Inform. Comput. 186 (2003) 140–162.
[53] N. Immerman, Nondeterministic space is closed under complementation, SIAM J. Comput. 17 (1988) 935–938.
[54] T. Jiang, E. McDowell, B. Ravikumar, The structure and complexity of minimal NFAs over a unary alphabet, Internat. J. Found. Comput. Sci. 2 (1991) 163–182.
[55] T. Jiang, B. Ravikumar, A note on the space complexity of some decision problems for finite automata, Inform. Process. Lett. 40 (1991) 25–31.
[56] T. Jiang, B. Ravikumar, Minimal NFA problems are hard, SIAM J. Comput. 22 (1993) 1117–1141.
[57] N. Jones, Space-bounded reducibility among combinatorial problems, J. Comput. System Sci. 11 (1975) 68–85.
[58] J.Y. Kao, A. Malton, N. Rampersad, J. Shallit, On NFA's where all states are final, initial, or both, Theoret. Comput. Sci. 410 (2009) 5010–5021.
[59] J. Köbler, U. Schöning, J. Torán, On counting and approximation, Acta Inform. 26 (1989) 363–379.
[60] D. Kozen, Lower Bounds for Natural Proof Systems, Foundations of Computer Science (FOCS 1977), IEEE Press, 1977.
[61] S.Y. Kuroda, Classes of languages and linear bounded automata, Inform. Control 7 (1964) 207–223.
[62] E. Landau, Über die Maximalordnung der Permutationen gegebenen Grades, Archiv der Math. und Phys. 3 (1903) 92–103.
[63] E. Landau, Handbuch der Lehre von der Verteilung der Primzahlen, Teubner, 1909.

[64] K.J. Lange, P. Rossmanith, The emptiness problem for intersections of regular languages, in: Mathematical Foundations of Computer Science (MFCS 1992), LNCS, vol. 629, Springer, 1992, pp. 346–354.
[65] E. Leiss, Succinct representation of regular languages by Boolean automata, Theoret. Comput. Sci. 13 (1981) 323–330.
[66] E. Leiss, Succinct representation of regular languages by Boolean automata. II, Theoret. Comput. Sci. 38 (1985) 133–136.
[67] A. Malcher, Minimizing finite automata is computationally hard, Theoret. Comput. Sci. 327 (2004) 375–390.
[68] P. McKenzie, P. Péladeau, D. Thérien, NC^1: The automata-theoretical viewpoint, Comput. Complexity 1 (1991) 330–359.
[69] R. McNaughton, S. Papert, Counter-free Automata, Research Monographs, vol. 65, MIT Press, 1971.
[70] C. Mereghetti, G. Pighizzini, Optimal simulations between unary automata, SIAM J. Comput. 30 (2001) 1976–1992.
[71] A.R. Meyer, M.J. Fischer, Economy of description by automata, grammars, and formal systems, in: IEEE Symposium on Switching and Automata Theory (SWAT 1971), IEEE Press, 1971, pp. 188–191.
[72] A.R. Meyer, L.J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential time, in: Symposium on Switching and Automata Theory (SWAT 1972), IEEE Press, 1972, pp. 125–129.
[73] B. Monien, On the LBA problem, in: Fundamentals on Computation Theory (FCT 1981), LNCS vol. 117, Springer (1981) 265–280.
[74] F.R. Moore, On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata, IEEE Trans. Comput. 20 (1971) 1211–1214.
[75] M.O. Rabin, D. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (1959) 114–125.
[76] R.L. Rivest, R.E. Schapire, Diversity-based Inference of Finite Automata (Extended Abstract), Foundations of Computer Science (FOCS 1987), IEEE Press, 1987, pp. 78–87.
[77] G. Rozenberg, A. Salomaa, The Mathematical Theory of L Systems, Pure and Applied Mathematics, vol. 90, Academic Press, 1980.
[78] M.P. Schützenberger, On finite monoids having only trivial subgroups, Inform. Control 8 (1965) 190–194.
[79] M. Sipser, Lower bounds on the size of sweeping automata, J. Comput. System Sci. 21 (1980) 195–202.
[80] R.E. Stearns, H.B. Hunt III, On the equivalence and containment problems for unambiguous regular expressions, regular grammars, and finite automata, SIAM J. Comput. 14 (1985) 598–611.
[81] J. Stern, Complexity of some problems from the theory of automata, Inform. Control 66 (1985) 163–176.
[82] L.J. Stockmeyer, The Complexity of Decision Problems in Automata Theory and Logic, Ph.D. Thesis, MIT, Cambridge, Massasuchets, USA, 1974.
[83] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time, Symposium on Theory of Computing (STOC 1973), ACM Press, 1973, pp. 1–9.
[84] R. Szelepcsényi, The method of forced enumeration for nondeterministic automata, Acta Inform. 26 (1988) 279–284.
[85] D. Thérien, Classification of finite monoids: the language approach, Theoret. Comput. Sci. 14 (1981) 195–208.
[86] L.G. Valiant, The complexity of computing the permanent, Theoret. Comput. Sci. 8 (2) (1979) 189–201.
[87] S. Yu, State complexity of regular languages, J. Autom. Lang. Comb. 6 (2001) 221–234.
[88] S. Yu, Regular Languages, Handbook of Formal Languages, vol. 1, Springer, 1997, pp. 41–110.