

Chapter 1

Descriptive Complexity — An Introductory Survey

Markus Holzer and Martin Kutrib

Institut für Informatik, Universität Giessen,

Arndtstr. 2, 35392 Giessen, Germany,

E-mail: {holzer,kutrib}@informatik.uni-giessen.de

The purpose of the paper is to give an introductory survey of the main aspects and results regarding the relative succinctness of different representations of languages, such as finite automata, regular expressions, push-down automata and variants thereof, context-free grammars, and descriptive systems from a more abstract perspective. Basic properties of these descriptive systems and their size measures are addressed. The trade-offs between different representations are either bounded by some recursive function, or reveal the phenomenon that the gain in economy of description can be arbitrary. In the latter case there is no recursive function serving as upper bound. We discuss developments relevant to the descriptive complexity of formal systems. The results presented are not proved but we merely draw attention to the big picture and some of the main ideas involved.

1.1 Introduction

In the field of theoretical computer science the term *descriptive complexity* has a well known meaning as it stands. Since the beginning of computer science descriptive complexity aspects of systems (automata, grammars, rewriting systems, etc.) have been a subject of intensive research [111]—since more than a decade the Workshop on “Descriptive Complexity of Formal Systems” (DCFS), formerly known as the Workshop on “Descrip-

tional Complexity of Automata, Grammar, and Related Structures,” has contributed substantially to the development of this field of research. The broad field of descriptonal complexity of formal systems includes, but is not limited to, various measures of complexity of automata, grammars, languages and of related descriptonal systems, succinctness of descriptonal systems, trade-offs between complexity and mode of operation, etc., to mention a few.

The time has come to give an introductory survey of the main aspects and results regarding the relative succinctness of different representations of languages by finite automata, pushdown automata and variants thereof, context-free grammars, and descriptonal systems from a more abstract perspective. Our tour mostly focuses on results that were found at the advent of descriptonal complexity, for example, [52, 53, 59, 60, 98, 109, 112]. To this end, we have to unify the treatment of different research directions from the past. See also [38] for a recent survey of some of these results. Our write up obviously lacks completeness and it reflects our personal view of what constitute the most interesting relations of the aforementioned devices from a descriptonal complexity point of view. In truth there is much more to the subject in question, than one can summarize here. For instance, the following current active research directions were not addressed in this summary: we skipped almost all results from the descriptonal complexity of the operation problem which was revitalized in [137] after the dawn in the late 1970’s. Moreover we will discuss anything on the subject of magic numbers a research field initiated in [73], and on the related investigations of determinization of nondeterministic finite automata accepting subregular languages done in [14] and others, and finally we left out the interesting field of research on the transition complexity of nondeterministic finite automata which has received a lot of attention during the last years [26, 46, 69, 70, 97].

In the next section, basic notions are given, and the basic properties of descriptonal systems and their complexity measures are discussed and presented in a unified manner. A natural and important measure of descriptonal complexity is the size of a representation of a language, that is, the length of its description. Section 1.3 is devoted to several aspects and results with respect to complexity measures that are recursively related to the sizes. A comprehensive overview of results is given concerning the question: how succinctly can a regular or a context-free language be represented by a descriptor of one descriptonal system compared with the representation by an equivalent descriptor of the other descriptonal sys-

tem? Section 1.4 generalizes this point of view. Roughly speaking some, say, structural resource is fixed and its descriptive power is studied by measuring other resources. So, the complexity measures are not necessarily recursively related to the sizes of the descriptors. Here we stick with context-free grammars and subclasses as descriptive systems. Finally, Section 1.5 deals with the phenomenon of non-recursive trade-offs, that is, the trade-offs between representations of languages in different descriptive systems are not bounded by any recursive function. With other words, the gain in economy of description can be arbitrary. It turned out that most of the proofs appearing in the literature are basically relying on one of two fundamental schemes. These proof schemes are presented in a unified manner. Some important results are collected in a compilation of non-recursive trade-offs.

1.2 Descriptive Systems and Complexity Measures

We denote the set of nonnegative integers by \mathbb{N} , and the powerset of a set S by 2^S . In connection with formal languages, strings are called *words*. Let Σ^* denote the set of all words over a finite alphabet Σ . The *empty word* is denoted by λ , and we set $\Sigma^+ = \Sigma^* - \{\lambda\}$. For the *reversal of a word* w we write w^R and for its *length* we write $|w|$. A *formal language* L is a subset of Σ^* . In order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most by the empty word, that is, $L - \{\lambda\} = L' - \{\lambda\}$. Throughout the article two automata or grammars are said to be *equivalent* if and only if they accept or generate the same language. We use \subseteq for *inclusions* and \subset for *strict inclusions*.

We first establish some notation for descriptive complexity. In order to be general, we formalize the intuitive notion of a representation or description of a family of languages. A *descriptive system* is a collection of encodings of items where each item *represents* or *describes* a formal language. In the following, we call the items *descriptors*, and identify the encodings of some language representation with the representation itself. A formal definition is:

Definition 1.1. A *descriptive system* \mathcal{S} is a set of finite descriptors, such that each descriptor $D \in \mathcal{S}$ describes a formal language $L(D)$, and the underlying alphabet $\text{alph}(D)$ over which D represents a language can be read off from D . The *family of languages represented (or described)*

by \mathcal{S} is $\mathcal{L}(\mathcal{S}) = \{L(D) \mid D \in \mathcal{S}\}$. For every language L , the set $\mathcal{S}(L) = \{D \in \mathcal{S} \mid L(D) = L\}$ is the set of its descriptors in \mathcal{S} .

Example 1.2. Pushdown automata (PDA) can be encoded over some fixed alphabet such that their input alphabets can be extracted from the encodings. The set of these encodings is a descriptonal system \mathcal{S} , and $\mathcal{L}(\mathcal{S})$ is the family of context-free languages (CFL). \square

Now we turn to measure the descriptors. Basically, we are interested in defining a complexity measure as general as possible to cover a wide range of approaches, and in defining it as precise as necessary to allow a unified framework for proofs. So, we consider a *complexity measure* for a descriptonal system \mathcal{S} to be a total, recursive mapping $c : \mathcal{S} \rightarrow \mathbb{N}$. The properties total and recursive are straightforward.

Example 1.3. The family of context-free grammars is a descriptonal system. Examples for complexity measures are the number productions appearing in a grammar, or the number of nonterminals, or the total number of symbols, that is, the length of the encoding. \square

Common notions as the *relative succinctness of descriptonal systems* and our intuitive understanding of descriptonal complexity suggest to consider the *size of descriptors*. From the viewpoint that a descriptonal system is a collection of encoding strings, the length of the strings is a natural measure for the size. We denote it by **length**. In fact, we will use it to obtain a rough classification of different complexity measures. We distinguish between measures that (with respect to the underlying alphabets) are recursively related with **length** and measures that are not.

Definition 1.4. Let \mathcal{S} be a descriptonal system with complexity measure c . If there is a total, recursive function $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ such that $\text{length}(D) \leq g(c(D), |\text{alph}(D)|)$, for all $D \in \mathcal{S}$, then c is said to be an *s-measure*.

Example 1.5. Let us consider a widely accepted measure of complexity for finite automata, that is, their number of states, which is denoted by **state**. The formal definition of a finite automaton is given in the next section. Is **state** an s-measure? What makes a difference between the number of states (say, for deterministic finite automata (DFA)) and the lengths of encoding strings? The answer is obvious, encoding strings are over some fixed alphabet whereas the input alphabet of DFAs is not fixed

a priori. The number of transitions depends on the input alphabet while the number of states does not. But states and transitions both determine the lengths of encoding strings. Nevertheless, when finite automata are addressed then, actually, a fixed given input alphabet is assumed tacitly. Since we regarded this aspect in the definition of s-measures, the answer to the first question is yes, the number of states of finite automata is an s-measure. To this end, given a deterministic finite automaton A , we may choose $g(\text{state}(A), \text{alph}(A)) = k \cdot \text{state}(A) \cdot \text{alph}(A)$, where $\text{state}(A) \cdot \text{alph}(A)$ is the number of transition rules, and k is a mapping that gives the length of a rule dependent on the actual encoding alphabet, the number of states and the number of input symbols.

Similarly, we can argue for other types of finite automata as nondeterministic or alternating ones either with one-way or two-way head motion, etc. If the number of transition rules depends on the number of states and the number of input symbols (and, of course, on the type of the automaton in question), and the length of the rules is bounded dependent on the type of the automaton, then *state* is an s-measure. \square

Whenever we consider the relative succinctness of two descriptive systems \mathcal{S}_1 and \mathcal{S}_2 , we assume the intersection $\mathcal{L}(\mathcal{S}_1) \cap \mathcal{L}(\mathcal{S}_2)$ to be non-empty.

Definition 1.6. Let \mathcal{S}_1 be a descriptive system with complexity measure c_1 , and \mathcal{S}_2 be a descriptive system with complexity measure c_2 . A total function $f : \mathbb{N} \rightarrow \mathbb{N}$, is said to be an *upper bound* for the increase in complexity when changing from a descriptor in \mathcal{S}_1 to an equivalent descriptor in \mathcal{S}_2 , if for all $D_1 \in \mathcal{S}_1$ with $L(D_1) \in \mathcal{L}(\mathcal{S}_2)$ there exists a $D_2 \in \mathcal{S}_2(L(D_1))$ such that $c_2(D_2) \leq f(c_1(D_1))$.

If there is no recursive function serving as upper bound, the *trade-off* is said to be *non-recursive*. That is, whenever the trade-off from one descriptive system to another is non-recursive, one can choose an arbitrarily large recursive function f but the gain in economy of description eventually exceeds f when changing from the former system to the latter.

Definition 1.7. Let \mathcal{S}_1 be a descriptive system with complexity measure c_1 , and \mathcal{S}_2 be a descriptive system with complexity measure c_2 . A total function $f : \mathbb{N} \rightarrow \mathbb{N}$, is said to be a *lower bound* for the increase in complexity when changing from a descriptor in \mathcal{S}_1 to an equivalent descriptor in \mathcal{S}_2 , if for infinitely many $D_1 \in \mathcal{S}_1$ with $L(D_1) \in \mathcal{L}(\mathcal{S}_2)$ there exists a *minimal* $D_2 \in \mathcal{S}_2(L(D_1))$ such that $c_2(D_2) \geq f(c_1(D_1))$.

1.3 Measuring Sizes

This section is devoted to several aspects of measuring descriptors with s-measures. A main field of investigation deals with the question: how succinctly can a language be represented by a descriptor of one descriptional system compared with the representation by an equivalent descriptor of the other descriptional system? An upper bound for the trade-off gives the maximal gain in economy of description, and conversely, the maximal blow-up (in terms of descriptional complexity) for simulations between the descriptional systems. A maximal lower bound for the trade-off terms the costs which are necessary in the worst cases.

1.3.1 *Descriptional Systems for Regular Languages*

Regular languages are represented by a large number of descriptional systems. So, it is natural to investigate the succinctness of their representations with respect to s-measures in order to optimize the space requirements. In this connection, many results have been obtained. On the other hand, the descriptional complexity of regular languages still offers challenging open problems. In the remainder of this subsection we collect and discuss some of these results and open problems.

1.3.1.1 *Finite Automata*

Here we measure the costs of representations by several types of finite automata in terms of the number of states, which is an s-measure by Example 1.5. Probably the most famous result of this nature is the simulation of nondeterministic finite automata by DFAs. Since several results come up with tight bounds in the exact number of states, it is advantageous to recall briefly the definitions of finite automata on which the results rely.

Definition 1.8. A *nondeterministic finite automaton* (NFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, 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 $\delta : Q \times \Sigma \rightarrow 2^Q$ is the *transition function*.

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 $\delta : Q \times \Sigma \rightarrow Q$.

The *language accepted* by the finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ is defined as $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$, where the transition function δ is naturally extended to $\delta : Q \times \Sigma^* \rightarrow 2^Q$.

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. Note that, therefore, a rejecting sink state is counted for DFAs, whereas it is not counted for NFAs. For further details we refer to [67].

It is well known that for any NFA one can always construct an equivalent DFA [119]. 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 [98, 109, 112]. So, NFAs can offer exponential savings in the number of states compared with DFAs.

Theorem 1.9 (NFA to DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state NFA. Then 2^n states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

For the particular cases of *finite* or *unary* regular languages the situation is significantly different. The conversion for finite languages over a binary alphabet was solved in [103] with a tight bound in the exact number of states. The general case of finite languages over a k -letter alphabet was shown in [124] with an asymptotically tight bound.

Theorem 1.10 (Finite NFA to DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state NFA accepting a finite language over a binary alphabet. Then $2 \cdot 2^{\frac{n}{2}} - 1$ if n is even, and $3 \cdot 2^{\frac{n-1}{2}} - 1$ if n is odd, states are sufficient and necessary in the worst case for a DFA to accept $L(A)$. If A accepts a finite language over a k -letter alphabet, for $k \geq 3$, then $\Theta(k^{\frac{n}{1+\log_2 k}})$ states are sufficient and necessary in the worst case.*

Thus, for finite languages over a two-letter alphabet the costs are only $\Theta(2^{\frac{n}{2}})$. The situation is similar when we turn to the second important special case, the unary languages. The general problem of evaluating the costs of unary automata simulations was raised in [128], and has led to emphasize some relevant differences with the general case. For state complexity issues of unary finite automata *Landau's function*

$F(n) = \max\{\text{lcm}(x_1, \dots, x_k) \mid x_1, \dots, x_k \geq 1 \text{ and } x_1 + \dots + x_k = n\}$, 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 [89, 90] the asymptotic growth rate $\lim_{n \rightarrow \infty} (\ln F(n) / \sqrt{n \cdot \ln n}) = 1$ 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 [22, 23]. Its proof is based on a normalform (Chrobak normalform) for unary NFAs introduced in [22]. 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 1.11 (Unary NFA to DFA conversion). *Let $n \geq 1$ be an integer 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 $L(A)$.*

For languages that are unary *and* finite in [103] it has been shown that nondeterminism does not help at all. Finite unary DFAs are up to one additional state as large as equivalent minimal NFAs. Moreover, it is well known that nondeterminism cannot help for all languages.

Example 1.12. Any NFA accepting the language

$$L_n = \{w \in \{a, b\}^* \mid |w| = i \cdot n, \text{ for } i \geq 0\},$$

for an integer $n \geq 1$, has at least n states, and L_n is accepted by an n -state DFA as well. □

On the one hand, we have seen that for certain languages unlimited nondeterminism cannot help. On the other hand, for unary languages accepted by NFAs in Chrobak normalform, that is, by NFAs that make at most *one* nondeterministic step in every computation, one can achieve a trade-off which is strictly less than 2^n but still exponential. This immediately brings us to the question in which cases nondeterminism can help to represent a regular language succinctly. A model with very weak nondeterminism are deterministic finite automata with multiple entry states (N DFA) [33, 134]. Here the sole guess appears at the beginning of the computation, that is, by choosing one out of k initial states. So, the nondeterminism is not only

limited in its amount but also in the situation at which it may appear. Converting an NFA with k initial states into a DFA by the powerset construction shows immediately that any reachable state contains at most k states of the NFA. This gives an upper bound for the conversion. In [66] it has been shown that this upper bound is tight.

Theorem 1.13 (NFA to DFA conversion). *Let $n, k \geq 1$ be integers satisfying $k \leq n$, and A be an n -state NFA with k entry states. Then $\sum_{i=1}^k \binom{n}{i}$ states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

So, for $k = 1$ we obtain DFAs while for $k = n$ we are concerned with the special case that needs $2^n - 1$ states. Interestingly, NFAs can be exponentially concise over NFAs. The following lower bound has been derived in [79].

Theorem 1.14 (NFA to NFA conversion). *Let $n \geq 1$ be an integer and A be an n -state NFA. Then $\Omega(2^n)$ states are necessary in the worst case for a NFA to accept $L(A)$.*

The concept of limited nondeterminism in finite automata is more generally studied in [82]. There, a bound on the number of nondeterministic steps allowed during a computation as well as on the maximal number of choices for every nondeterministic step is imposed. While the maximal number of choices is three, the bound on the number of steps is given by a function that depends on the number of states. This implies that in any computation the NFAs can make a finite number of nondeterministic steps only. But the situations at which nondeterminism appears are not restricted *a priori*. The order of magnitude of the functions considered is strictly less than the logarithm, that is, for a bounding function f we have $f \in o(\log)$. The upper bound for the costs of the conversion into a DFA follows from the powerset construction. Due to the restrictions any reachable state contains at most $3^{f(n)}$ states of the NFA. The next theorem summarizes this observation and the lower bound shown in [82].

Theorem 1.15 (Limited NFA to DFA conversion). *Let $n \geq 1$ be an integer, A be an n -state NFA, and $f : \mathbb{N} \rightarrow \mathbb{N}$ be function of order $o(\log)$. Then $\sum_{i=0}^{3^{f(n)}} \binom{n}{i}$ states are sufficient and $\sum_{i=0}^{2^{f(\sqrt{n})}} \binom{O(\sqrt{n})}{i}$ is a lower bound for the worst case state complexity for a deterministic finite automaton to accept $L(A)$.*

Note that the upper bound $\sum_{i=0}^{3^{f(n)}} \binom{n}{i}$ is of order $o(2^n)$, if $f(n) \in o(\log n)$. The precise bound for the conversion is an open problem.

Next, we turn to finite automata whose descriptive capacity is stronger than NFAs due to their additional resources. We start with alternating finite automata which have been developed in [21], and recall their definition from that paper. To this end, 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 $\{0, 1\}$.

Definition 1.16. An *alternating finite automaton* (AFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where $Q, \Sigma, q_0,$ and F are as for NFAs, and $\delta : Q \times \Sigma \rightarrow \{0, 1\}^{\{0,1\}^Q}$ 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 $L(A) = \{w \in \Sigma^* \mid w \text{ is accepted by } A\}$.

Example 1.17. [136] Let $A = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ be an AFA with transition function defined through

$$\begin{aligned} \delta(q_0, a) &= q_1 \wedge q_2, & \delta(q_0, b) &= 0, \\ \delta(q_1, a) &= q_2, & \delta(q_1, b) &= q_1 \wedge q_2, \\ \delta(q_2, a) &= \overline{q_1} \wedge q_2, & \delta(q_2, b) &= q_1 \vee \overline{q_2}. \end{aligned}$$

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 $(q_2 \wedge (\overline{q_1} \wedge q_2)) \wedge (q_2 \vee (\overline{q_1} \wedge q_2))$.

After substituting the characteristic vector, that is, 0 for $q_0, q_1,$ and 1 for q_2 we have $(1 \wedge (\overline{0} \wedge 1)) \wedge (1 \vee (\overline{0} \wedge 1))$ which evaluates to 1. Therefore, the input aba is accepted. □

It is worth mentioning that sometimes in the literature an equivalent definition of AFAs appears, where the state set is partitioned into existential and universal states.

At the same period as alternating finite automata were developed in [15] 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 descriptive complexity point of view.

Lemma 1.18 (Boolean automata to AFA conversion). *Let $n \geq 1$ be an integer 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 [30] for more details on alternating finite automata having an initial state that is unreachable after the first step.

Next we consider the descriptive capacity of AFAs compared with NFAs and DFAs. The tight bound of 2^{2^n} states for the conversion of n -state AFAs into DFAs has already been shown in the fundamental papers [21] for AFAs and [15, 92] for Boolean automata.

Theorem 1.19 (AFA/Boolean automata to DFA conversion). *Let $n \geq 1$ be an integer 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 descriptonal capacity of AFAs compared with NFAs. In [30] constructions are presented that show how to convert an AFA into an equivalent nondeterministic finite automaton with multiple entry states (NNFA). Let $A = (Q, \Sigma, \delta, q_0, F)$ be an n -state AFA with $Q = \{q_0, q_1, \dots, q_{n-1}\}$ 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 $\delta'(u, a) = \{u' \mid \delta(u', a) = u\}$, for all $u, u' \in \{0, 1\}^Q$ 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 1.20. Moreover, since the powerset construction works also fine for the determinization of NNFAs, the presented construction also reveals the upper bound for the AFA to DFA conversion already stated in Theorem 1.19. 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 $\{u \in \{0, 1\}^Q \mid f_0(u(q_0), u(q_1), \dots, u(q_{n-1})) = 1\}$. From the construction we derive the upper bound of the next theorem.

Theorem 1.20 (AFA to NNFA conversion). *Let $n \geq 1$ be an integer 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 1.19 is shown in [21] 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 [21]. 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 [92], 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 to a DFA. The next theorem summarizes these results.

Theorem 1.21 (Reversed AFA to DFA conversion). *Let $n \geq 1$ be an integer 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 leaves 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 1.19 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 [92]. 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 1.21. 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 1.20 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 [30] it is conjectured that this bound presented in the following theorem is tight.

Lemma 1.22 (AFA to NFA conversion). *Let $n \geq 1$ be an integer and A be an n -state AFA or Boolean automaton. Then $2^n + 1$ states are sufficient and 2^n is a lower bound for the worst case state complexity for an NFA to accept $L(A)$.*

We now direct our attention to the question whether alternation can always help to represent a regular language succinctly. We have seen already that nondeterminism cannot help for all languages. So, how about the worst case of the language representation by alternating finite automata? The situation seems to be more sophisticated. Theorem 1.21 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 [93].

Theorem 1.23. *For every integer $n \geq 1$ 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 $A_n = (\{q_0, q_1, \dots, q_{n-1}\}, \{a, b\}, \delta, q_1, F)$ witness the theorem for all integers $n \geq 2$, where $F = \{q_i \mid 0 \leq i \leq n-1 \text{ and } i \text{ even}\}$ 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 [92]. 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 1.21. This is a contradiction since $2^m < 2^n$.

Corollary 1.24. *Let $n \geq 1$ be an integer and 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.*

We have already seen that unary NFAs can be much more concise than DFAs, but yet not as much as for the general case. So, we next continue to draw that part of the picture with respect to alternating finite automata. In general, the simulation of AFAs by DFAs 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 1.21. The lower bound can be seen by considering the single word language $L_n = \{a^{2^n-1}\}$. For all $n \geq 1$,

the language L_n is accepted by some minimal $(2^n + 1)$ -state DFA and an n -state AFA. So, we derive the next theorem.

Theorem 1.25 (Unary AFA to DFA conversion). *Let $n \geq 1$ be an integer 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 require 2^n states for any NFA accepting it.

Corollary 1.26 (Unary AFA to NFA simulation). *Let $n \geq 1$ be an integer 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 1.23 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 1.25 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 1.27 (Unary DFA to AFA conversion). *Let $n \geq 1$ be an integer 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 1.11 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 1.21 in combination with Lemma 1.18 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 1.11 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 1.25.

Theorem 1.28 (Unary NFA to AFA conversion). *Let $n \geq 1$ be an integer and A be an n -state NFA accepting a unary language. Then $\Theta(\sqrt{n \cdot \ln n})$ states are sufficient and necessary in the worst case for an AFA to accept $L(A)$.*

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

Corollary 1.29. *Let $n \geq 1$ be an integer and 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.*

The final resource we investigate here with respect to its descriptive capacity is two-way head motion. We denote deterministic and nondeterministic finite automata that may move their head to the right as well as to the left by 2DFA and 2NFA. We first sketch the development of results for general regular languages. Then we turn to unary languages again.

Concerning the simulation of 2DFA by DFA an $\Theta(n^n)$ asymptotically tight bound was shown in [127]. Moreover, the proof implied that any n -state 2NFA can be simulated by an NFA with at most $n2^{n^2}$ states. The well-known proof of the equivalence of two-way and one-way finite automata via crossing sequences reveals a bound of $O(2^{2n \log n})$ states [67]. Recently in [77] it was noted that a straightforward elaboration on [127] shows that the cost can be brought down to even $n(n+1)^n$. However, this bound still wastes exponentially many states, as [10] shows that $8^n + 2$ states suffice by an argument based on length-preserving homomorphisms. Recently, the problem was finally solved in [77] by establishing the tight bound of the following theorem.

Theorem 1.30 (2NFA/2DFA to NFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2NFA or an n -state 2DFA. Then $\binom{2n}{n+1}$ states*

are sufficient and necessary in the worst case for an NFA to accept $L(A)$.

Furthermore, the following tight bounds in the exact number of states for the 2DFA and 2NFA conversion to not necessarily complete DFAs have been shown in [78].

Theorem 1.31 (2NFA/2DFA to DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2DFA. Then $n(n^n - (n-1)^n)$ states are sufficient and necessary in the worst case for a DFA to accept $L(A)$. Moreover, if A is an n -state 2NFA, then $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \binom{n}{i} \binom{n}{j} (2^i - 1)^j$ states are sufficient and necessary in the worst case.*

The bounds reveal that two-way head motion is a very powerful resource with respect to descriptive complexity. Interestingly, when simulating two-way devices by NFAs, it does not matter whether the two-way device is nondeterministic or not. From this point of view, two-way head motion can compensate for nondeterminism.

Nevertheless, challenging problems are still open. The question of the costs for trading two-way head motion for nondeterminism, that is, the costs for simulating (two-way) NFA by 2DFAs is unanswered for decades. It was raised by Sakoda and Sipser in [122]. They conjectured that the upper bound is exponential. The best lower bound currently known is $\Omega(n^2/\log n)$. It was proved in [4], where also an interesting connection with the open problem whether L equals NL is given. In particular, if $L = NL$, then for some polynomial p , all integers m , and all n -state 2NFAs A , there exists a $p(mk)$ -state 2DFA accepting a subset of $L(A)$ including all words whose lengths do not exceed m . However, not only are the exact bounds of that problem unknown, but we cannot even confirm the conjecture that they are exponential.

The picture was complemented by the sophisticated studies on unary languages. The problem of Sakoda and Sipser has partially been solved for the unary case in [22] as follows.

Theorem 1.32 (Unary NFA to 2DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2DFA accepting a unary language. Then $\Theta(n^2)$ states are sufficient and necessary in the worst case for a 2DFA to accept $L(A)$.*

The result has been shown with the surprisingly simple witness languages $L_n = \{a^k \mid k = n \cdot i + (n-1) \cdot j, \text{ for } i, j \geq 1\}$, for all integers $n \geq 1$.

An upper bound for the remaining case has been shown in [31].

Theorem 1.33 (Unary 2NFA to 2DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2NFA accepting a unary language. Then $O(n^{\lceil \log_2(n+1) + 3 \rceil})$ states are sufficient for a 2DFA to accept $L(A)$.*

In [22, 23] the same costs as for simulating unary NFAs by DFAs are derived for removing two-way head motion from deterministic automata.

Theorem 1.34 (Unary 2DFA to NFA/DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2DFA accepting a unary language. Then $e^{\Theta(\sqrt{n \cdot \ln n})}$ states are sufficient and necessary in the worst case for an NFA or a DFA to accept $L(A)$.*

It turned out that, again, the same costs appear for removing two-way head motion from nondeterministic automata [108].

Theorem 1.35 (Unary 2NFA to NFA/DFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2NFA accepting a unary language. Then $e^{\Theta(\sqrt{n \cdot \ln n})}$ states are sufficient and necessary in the worst case for an NFA or a DFA to accept $L(A)$.*

The result was shown for the conversion to DFAs. This gives an upper bound also for the conversion to NFAs. The lower bound for the second case follows from the tight bounds of Theorem 1.34.

So, nondeterministic as well as two-way automata are hard to simulate for DFAs even if they accept unary languages. Since the bounds are the same, it seems that two-way motion is equally powerful as nondeterminism, but both together cannot increase the descriptive capacity. This observation is confirmed by the bounds for simulations by NFAs, where similarly as in the general case it does not matter whether the two-way device is nondeterministic or not. Nevertheless, from this point of view, two-way head motion can compensate for nondeterminism. Since unary 2DFAs can simulate NFAs increasing the number of states only polynomially, which is not possible the other way around, two-way motion turned out to be more powerful than nondeterminism.

Finally, we present the results concerning the relations (needless to say, with respect to descriptive complexity) between AFAs and 2NFAs as well as 2DFAs. These problems have been suggested to be investigated in [22]. The results are all derived in [108]. Starting with the 2NFA simulation by AFAs we conclude that any unary n -state 2NFA can be converted in a

DFA having at most $e^{\Theta(\sqrt{n \cdot \ln n})}$ states by Theorem 1.35. Next, the DFA is converted into an AFA with at most $\Theta(\sqrt{n \cdot \ln n})$ states by Theorem 1.27. For the lower bound, we argue as follows. Given a unary NFA that causes the maximal blow-up when converted to a DFA, we obtain an AFA from the DFA having at least $\Theta(\sqrt{n \cdot \ln n})$ states by Theorem 1.27. Moreover, for the 2NFA simulation by AFAs we can apply the same upper bound. Since in [22, 23] it has been shown that the witness languages for the fact that there is a unary NFA that causes the maximal blow-up when converted to a DFA are also accepted by n -state 2DFAs, the lower bound also applies for the 2DFA conversion.

Theorem 1.36 (Unary 2FA to AFA conversion). *Let $n \geq 1$ be an integer and A be an n -state 2NFA or an n -state 2DFA accepting a unary language. Then $\Theta(\sqrt{n \cdot \ln n})$ states are sufficient and necessary in the worst case for an AFA to accept $L(A)$.*

For the converse simulations a result from [11] is used. It says that any 2NFA accepting a single word language $L_n = \{a^{2^n-1}\}$ must have $\Omega(2^n)$ states, while it is accepted by some n -state AFA. On the other hand, in the order of magnitude we can derive the matching upper bound from the unary AFA to DFA conversion.

Theorem 1.37 (Unary AFA to 2FA conversion). *Let $n \geq 1$ be an integer and A be an n -state AFA accepting a unary language. Then $\Theta(2^n)$ states are sufficient and necessary in the worst case for a 2NFA or 2DFA to accept $L(A)$.*

In conclusion of this part of the section some approaches not discussed are worth mentioning. In [88] the state complexity of weak restarting automata is considered. The determinization of several finite automata for subregular language families is investigated in [14]. Several papers dealt with descriptive complexity questions of unambiguous descriptors [39, 71, 94, 120, 125, 130]. In order to attack and to solve the problem of Sakoda and Sipser for subclasses, sweeping automata are investigated in [3, 95, 110, 128]. Moreover, K -visit 2NFAs are studied in [84], and [9, 75] considered the problem for positional simulations.

1.3.1.2 Regular Expressions

One of the most basic theorems in formal language theory is that every regular expression can be effectively converted into an equivalent finite automaton, and *vice versa* [85]. Regular expressions are defined as follows.

Definition 1.38. Let Σ be an alphabet. Then \emptyset , λ , and every letter $a \in \Sigma$ are *regular expressions*. If r and s are regular expressions, then $(r + s)$, $(r \cdot s)$, and (r^*) are also regular expressions. The language $L(r)$ defined by a regular expression r is defined as follows: $L(\emptyset) = \emptyset$, $L(\lambda) = \{\lambda\}$, $L(a) = \{a\}$, $L(r+s) = L(r) \cup L(s)$, $L(r \cdot s) = L(r) \cdot L(s)$, and $L(r^*) = L(r)^*$.

For convenience, parentheses in regular expressions are sometimes omitted and the concatenation is simply written as juxtaposition. The priority of operators is specified in the usual fashion: Concatenation is performed before union, and star before both product and union.

In the literature one finds a lot of different complexity measures for regular expressions. The measure **size** is defined to be the total number of symbols (including \emptyset , λ , alphabetic symbols from Σ , all operation symbols, and parentheses) of a completely bracketed regular expression (for example, used in [1], where it is called length). Another measure related to the reverse polish notation of a regular expression is **rpn**, which gives the number of nodes in the syntax tree of the expressions (parentheses are not counted). This measure is equal to the length of a (parenthesis-free) expression in postfix notation [1]. The alphabetic width **a-width** is the total number of alphabetic symbols from Σ (counted with multiplicity) [105, 28].

In order to clarify our definitions we give a small example [29].

Example 1.39. Let $r = ((0 + ((1 \cdot 0)^*)) \cdot (1 + \lambda))$ be a regular expression. Then $\text{size}(r) = 20$, $\text{rpn}(r) = 10$, because the expression in postfix notation reads $010 \cdot * + 1\lambda + \cdot$, and $\text{a-width}(r) = 4$. \square

Further not so well known measures are the ordinary length **o-length** [29], the width **width** [28], the length (dual to width) **length** [28], and the sum **sum** [50]. To our knowledge all these measures, except for the first one, never have been studied again since their introduction. See Table 1.1 for the inductive definition of the measures. We have also included the definition of the non s-measure star height **height**, which comes into play at the end of this subsection.

Next we restrict ourself to the first three mentioned measures **size**, **rpn**, and **a-width**. As usual, the size of a regular language L , that is $\text{size}(L)$, is

Table 1.1 Inductive definitions of the measures *size*, *rpn*, *a-width*, *o-length*, *width*, *length*, *sum*, and *height* for regular expressions. \circ refers to the measure to be defined.

Measure	\emptyset	λ	$a \in \Sigma$	$(r \cdot s)$	$(r)^*$	$(r + s)$
<i>size</i>	1	1	1	$\circ(r) + \circ(s) + 3$	$\circ(r) + 3$	$\circ(r) + \circ(s) + 3$
<i>rpn</i>	1	1	1	$\circ(r) + \circ(s) + 1$	$\circ(r) + 1$	$\circ(r) + \circ(s) + 1$
<i>a-width</i>	0	0	1	$\circ(r) + \circ(s)$	$\circ(r)$	$\circ(r) + \circ(s)$
<i>o-length</i>	1	1	1	$\circ(r) + \circ(s) + 2$	$\circ(r) + 3$	$\circ(r) + \circ(s) + 3$
<i>width</i>	0	0	1	$\max\{\circ(r), \circ(s)\}$	$\circ(r)$	$\circ(r) + \circ(s)$
<i>length</i>	0	0	1	$\circ(r) + \circ(s)$	$\circ(r)$	$\max\{\circ(r), \circ(s)\}$
<i>sum</i>	1	1	1	$\circ(r) + \circ(s)$	$\circ(r)$	$\begin{cases} 1, & \text{if } L(r) \subseteq \Sigma \text{ and } L(s) \subseteq \Sigma \\ \circ(r) + \circ(s), & \text{otherwise} \end{cases}$
<i>height</i>	0	0	0	$\max\{\circ(r) + \circ(s)\}$	$\circ(r) + 1$	$\max\{\circ(r), \circ(s)\}$

defined to be the minimum *size* among all regular expressions denoting L . The notions $\text{rpn}(L)$, $\text{a-width}(L)$, and $\text{height}(L)$ are analogously defined. One can easily show that the measures *rpn* and *a-width* are linearly related to *size*, thus these are all *s-measures*. Relations between these measures have, for example, been studied in [28, 29, 45, 50, 56, 72].

Theorem 1.40 (Relation on basic regular expression measures).

Let L be a regular language. Then

- (1) $\text{size}(L) \leq 3 \cdot \text{rpn}(L)$ and $\text{size}(L) \leq 8 \cdot \text{a-width}(L) - 3$,
- (2) $\text{a-width}(L) \leq \frac{1}{2} \cdot (\text{size}(L) + 1)$ and $\text{a-width}(L) \leq \frac{1}{2} \cdot (\text{rpn}(L) + 1)$,
- (3) $\text{rpn}(L) \leq \frac{1}{2} \cdot (\text{size}(L) + 1)$ and $\text{rpn}(L) \leq 4 \cdot \text{a-width}(L) - 1$.

Because there are so many results on these measures, we further have to narrow our focus on some important aspects. We discuss some (recent) results on the conversion from regular expressions to finite automata and *vice versa* in more detail. Moreover, for our presentation we concentrate on the measure *a-width* for regular expressions.

Converting regular expressions to finite automata is well understood. The following theorem is due to [37] and [91, 92]. The upper bounds are obtained by effective constructions. For the regular expression to NFA conversion the so called *Glushkov* or *position* automaton is constructed [37], which has the property that the initial state has no incoming transitions.

Theorem 1.41 (Regular expression to FA conversion). Let $n \geq 1$ be an integer and r be a regular expression with $\text{a-width}(r) = n$. Then $n + 1$ states are sufficient and necessary in the worst case for an NFA to accept $L(r)$. In case of a DFA $2^n + 1$ states are sufficient.

The tightness of the bound for NFAs can be seen by the unary witness languages $L_n = \{a^n\}$, for integers $n \geq 1$, for which any NFA needs at least $n + 1$ states. In general, techniques to prove lower bounds on the number of states for NFAs are presented in [8, 36, 47, 68]. More on the conversion from regular expressions to finite automata can be found in [121, 136]. The papers [20, 45, 56, 72] also may serve as good references for recent developments.

What concerns the other conversion direction? Probably the most popular algorithm for converting a finite automaton into an equivalent regular expression is the state elimination technique, which is a variant of the algorithm of McNaughton and Yamada [105]. All known algorithms covering the general case of infinite languages are based on the classical ones, which are compared in the survey [121]. The drawback is that all of these (structurally similar) algorithms return expressions of size $2^{O(n)}$ in the worst case [29, 48], assuming an alphabet size at most polynomial in the number of states. Recently a family of languages over a *binary* alphabet was exhibited for which this exponential blow up is inevitable [48].

Theorem 1.42 (FA to regular expression conversion). *Let $n \geq 1$ be an integer and A be an n -state finite automaton with input alphabet size at most polynomial in n . Then alphabetic width $2^{\Theta(n)}$ is sufficient and necessary in the worst case for a regular expression to describe $L(A)$.*

Note that the conversion problem for finite automata accepting unary languages becomes linear for DFAs and polynomial for NFAs [29]; in the latter case the proof is based on the Chrobak normalform for NFAs accepting unary languages. When changing the representation of a *finite* language from a DFA or NFA to an equivalent regular expression, a tight bound of order $n^{\Theta(\log n)}$ was shown in [50]. Unary finite languages accepted by finite automata can easily be converted to regular expressions of linear alphabetic width.

Before we close this section we want to comment on lower bound techniques for regular expressions. In the literature one can find at least three techniques. The first one is due to Ehrenfeucht and Zeiger, which however requires, in its original version, a largely growing alphabet. Recently, a variation of this method was used in [32] to get similar but weaker lower bounds. A technique based on communication complexity that applies only for finite languages is proposed in [50]. The most general technique, up to now, was developed in [48], where the s-measure *a*-width is related to the

non s-measure star height **height** of regular languages—for a definition of star height we refer to Table 1.1.

Theorem 1.43 (Star height versus alphabetic width). *Let L be a regular language. Then $\text{height}(L) \leq 3 \cdot \log(\mathbf{a}\text{-width}(L) + 1) + 1$.*

As mentioned above, the measure **height** is not an s-measure. This is easily seen because every finite language has star height 0, but can be of arbitrary size. Thus, the difference in the above theorem can be arbitrarily large. Nevertheless, language families are known, where this bound is tight up to a constant factor: define the languages L_n inductively by $L_1 = \lambda$ and $L_n = (0L_{n-1}1)^*$, for all integers $n \geq 1$. Then $\mathbf{a}\text{-width}(L_n)$ is clearly at most $2n$, but it is known from [104] that $\text{height}(L_{2^n}) = n$, for all integers $n \geq 1$. Thus, the previous theorem can be used for proving lower bounds on $\mathbf{a}\text{-width}(L)$ by determining the star height of the language L .

The star height of regular languages has been intensively studied in the literature for more than 40 years, see [63, 83] for a recent treatment. Determining the star height can be in some cases reduced to the easier task of determining the cycle rank of a certain digraph, a digraph connectivity measure defined in [27] in the 1960s. Observe, that measuring the connectivity of digraphs is a very active research area [6, 7, 74].

1.3.2 Pushdown Automata and Context-Free Grammars

In the previous section the relative succinctness of descriptive systems representing the regular languages has been discussed. A more general treatment deals with descriptive systems having a non-empty intersection. For example, one can consider languages that are deterministic *and* linear context free in order to study the relative succinctness of deterministic pushdown automata and linear context-free grammars. A more particular approach is to represent regular languages by pushdown automata or context-free grammars.

1.3.2.1 Pushdown Automata

Measuring the size of a pushdown automaton (PDA) by its number of states, as is done for finite automata, is clearly ineligible. It is well known that every pushdown automaton can effectively be converted into an equivalent one having just one sole state [67]. But, in general, one has to pay with an increase in the number of stack symbols, and determinism is not

preserved. For deterministic pushdown automata accepting by empty pushdown, the expressive power is known to increase strictly with the number of states [57]. A language is accepted by a one-state deterministic pushdown automaton if and only if it is a *simple* language, that is, it is generated by a context-free grammar of a very restricted form. So, measuring the size of a (deterministic) pushdown automaton by its number of stack symbols is also too crude. In fact, it is also possible to reduce the number of stack symbols if one pays with an increase in the number of states. The precise relations between states and stack symbols have been shown in [40] and [42].

Theorem 1.44. *For every (real-time) PDA with n states and t stack symbols and for every integer m in the range $1 \leq m < n$, there is an equivalent (real-time) PDA with m states and $t\lceil n/m \rceil^2 + 1$ stack symbols.*

The conversion preserves real-time behavior but not determinism. However, the construction is more or less the best possible, in the sense that an expansion in the stack alphabet to size $t\lceil n/m \rceil^2$ is sometimes unavoidable even if real-time behavior need not to be preserved, and also in the sense that no general state-reduction procedure can preserve determinism.

Theorem 1.45. *For every pair of positive integers n and t , there is a deterministic real-time PDA with n states and t stack symbols such that*

- (1) *every equivalent PDA with m states has at least $t\lceil n/m \rceil^2$ stack symbols, and*
- (2) *every equivalent PDA having fewer than n states is not deterministic.*

These results immediately raise the question of the converse transformations, that is, for transformations that reduce the number of stack symbols. The question has been answered in [42]. In particular, determinism can be preserved, but if it is allowed to introduce nondeterminism, states can be saved.

Theorem 1.46.

- (1) *For every (deterministic) PDA with n states and t stack symbols and for every integer r in the range $2 \leq r < t$, there is an equivalent (deterministic) PDA with r stack symbols and $O(n \cdot t/r)$ states.*
- (2) *For every PDA with n states and t stack symbols and for every integer r in the range $2 \leq r < t$, there is an equivalent PDA with r stack symbols and $O(n\sqrt{t/r})$ states.*

Both transformations do not preserve real-time behavior. However, again, these transformations are essentially the best possible ones. There are no transformations preserving determinism which always increase the number of states by less than $O(n \cdot t/r)$, there are no transformations which always increase the number of states by less than $O(n\sqrt{t/r})$, and there are no constructions which always preserve real-time behavior at all.

Theorem 1.47. *For every pair of positive integers n and t ,*

- (1) *there is a deterministic PDA with n states and t stack symbols such that every equivalent deterministic PDA with r stack symbols has at least $n \cdot t/r$ states,*
- (2) *there is a PDA with n states and t stack symbols such that every equivalent PDA with r stack symbols has at least $n\sqrt{t/r}$ states, and*
- (3) *there is a deterministic real-time PDA with n states and t stack symbols such that every equivalent real-time PDA has at least t stack symbols.*

So, besides the input alphabet, the number of states as well as the number of stack symbols have to be considered to measure the size of a pushdown automaton. But even their product is insufficient. For example, for all integers $n \geq 1$ the language $L_n = (a^n)^*$ can be accepted by a PDA with two states and two stack symbols that, in one move, is able to push n symbols on the stack. So, in addition, we have to take into account the lengths of the right-hand sides of the transition rules which can get long when a PDA pushes lots of symbols during single transitions.

Given a pushdown automaton $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ with state set Q , input alphabet Σ , stack alphabet Γ and transition δ mapping $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$, we consider size to be the measure of complexity defined by $|Q| \cdot |\Sigma| \cdot |\Gamma| \cdot h$, where h is the length of the longest word pushed in a single transition. In order to see that size is an s-measure for pushdown automata we observe that there are at most $|Q| \cdot (|\Sigma| + 1) \cdot |\Gamma|$ different left-hand sides and at most $|Q| \cdot |\Gamma|^h$ different right-hand sides of transition rules. So, there are at most $\text{size}(A)^{h+2}$ transition rules, and we may choose $g(\text{size}(A), \text{alph}(A)) = k \cdot \text{size}(A)^{h+2}$, where k is a mapping that gives the length of a rule dependent on the actual encoding alphabet, the number of states, input symbols, and stack symbols.

In order to avoid technical encumbrance sometimes PDAs are considered that are allowed to push at most two symbols during every transition. In [57] these PDAs are called *moderate*. It is well known that every (deterministic) PDA can effectively be converted into a moderate

one. But how about the changes in size? For the conversion every transition rule of the form $\delta(p, a, t) \ni (q, u_1u_2 \cdots u_m)$ with $m > 2$ is replaced by the rules $\delta(p, a, t) \ni (p_1, u_{m-1}u_m)$, $\delta(p_1, \lambda, u_{m-1}) \ni (p_2, u_{m-2}u_{m-1})$, $\delta(p_2, \lambda, u_{m-2}) \ni (p_3, u_{m-3}u_{m-2})$, \dots , $\delta(p_{m-2}, \lambda, u_2) \ni (q, u_1u_2)$, where p_1, p_2, \dots, p_{m-2} are new states not appearing in any other rule. Therefore, roughly $m + 4$ symbols are replaced by $6(m - 1)$ symbols. This implies that every PDA A can be converted into a moderate one having size $\Theta(\text{size}(A))$.

Next we turn to some fundamental results in connection with the representation of regular languages by pushdown automata. In [129] the decidability of regularity for deterministic pushdown automata has been shown by a deep proof. This effective procedure revealed the following upper bound for the trade-off in descriptive complexity when deterministic pushdown automata accepting regular languages are converted into DFAs. Given a deterministic pushdown automaton with $n > 1$ states and $t > 1$ stack symbols that accepts a regular language. Then the number of states which is sufficient for an equivalent DFA is bounded by an expression of the order $t^{n^{n^t}}$. Later this triple exponential upper bound has been improved by one level of exponentiation in [132].

Theorem 1.48. *Let A be a deterministic pushdown automaton with n states, t stack symbols, and h is the length of the longest word pushed in a single transition. If $L(A)$ is regular then $2^{2^{O(n^2 \log n + \log t + \log h)}}$ states are sufficient for a DFA to accept $L(A)$.*

In the levels of exponentiation this bound is tight, since the following double exponential lower bound has been obtained in [109]. It is open whether the precise lower bound or the precise upper bound can be improved in order to obtain matching bounds.

Theorem 1.49. *Let $n \geq 1$ be an integer. Then there is a language L_n accepted by a deterministic pushdown automaton of size $O(n^3)$, and each equivalent DFA has at least 2^{2^n} states.*

The theorem is witnessed by languages L_n that are subsets of $\{0, 1, a_1, a_2, \dots, a_n\}^* \{0, 1\}^n$. The subsets are specified by an accepting deterministic pushdown automaton which operates as follows:

- (1) Push the input onto the stack until symbol a_1 appears. If a_1 does not occur, reject the input.
- (2) Set $i = 2$.

- (3) If the next input is 0, pop the stack until the first occurrence of a_i . Else if the next input is 1, pop the stack until the second occurrence of a_i . Else if any other input appears or the occurrences of a_i are not found, reject the input.
- (4) Increment i by one.
- (5) If $i \leq n$, repeat step (3).
- (6) If the digit on top of the stack is 1 and there are no more input symbols, accept the input. Otherwise reject the input.

The natural next step is to consider the trade-offs when nondeterministic or finite-turn pushdown automata accepting regular languages are converted into DFAs. But in [109] it has been shown that for *any* given recursive function f and arbitrarily large integers n , there exists a nondeterministic pushdown automaton of size n representing a regular language, such that any equivalent DFA has at least $f(n)$ states. This implies that there does not exist a recursive function serving as upper bound for the trade-off. We deal with this phenomenon in Section 1.5 in more detail. However, the situation is different when unary languages are considered. It is well known that every unary context-free language is regular [35]. From the viewpoint of descriptive complexity, unary nondeterministic pushdown automata have been investigated in [116] where PDAs having a *strong normalform* are considered. In particular, the normalform PDAs are such that (1) at the start of the computation the stack contains only the bottom-of-stack symbol which is never pushed or popped, (2) the input is accepted if and only if the automaton reaches an accepting state, the stack contains only the bottom-of stack symbol, and all the input has been scanned, (3) if the PDA moves the input head, then no operations are performed on the stack, and (4) every push adds *exactly* one symbol on the stack. The complexity measure used in [116] is the product of the number of states and the number of stack symbols. While this measure is reasonable for the normalform PDAs, it distorts the results with respect to PDAs that are measured, for example, by *size*. However, similarly as for moderate PDAs it is not hard to convert a given PDA into a normalform PDA, whereby the number of stack symbols is increased by at most one, and the number of states is increased linearly in *size* of the given automaton. This implies that every PDA A can be converted into a normalform one having size $\Theta(\text{size}(A))$.

Theorem 1.50. *For every normalform PDA with n states and t stack symbols accepting a unary language there is an equivalent NFA with $2^{2n^2t+1} + 1$ states, and an equivalent DFA with $2^{n^4t^2+2n^2t+1}$ states. Therefore, for every*

arbitrary PDA of size s accepting a unary language there is an equivalent NFA with $2^{O(s^2)}$ states, and an equivalent DFA with $2^{O(s^4)}$ states.

Similar investigations for deterministic pushdown automata have been done in [115]. It is proved that each unary deterministic pushdown automaton of size s can be converted into a DFA with a number of states exponential in s . Moreover, this bound is tight in the order of magnitude.

Theorem 1.51. *For every normalform deterministic pushdown automaton with n states and t stack symbols accepting a unary language there is an equivalent DFA with $2^{n \cdot t}$ states. Therefore, for every arbitrary deterministic PDA of size s accepting a unary language there is an equivalent DFA with $2^{O(s)}$ states.*

Theorem 1.35 says that any unary n -state 2NFA can be simulated by a DFA with $e^{\Theta(\sqrt{n \cdot \ln n})}$ states. This suggests the possibility of a smaller gap between the descriptive complexities of unary deterministic pushdown automata and 2NFAs. However, even in this case the gap can be exponential.

Theorem 1.52. *Let $s \geq 1$ be an integer. Then there is a unary language L_s accepted by a deterministic pushdown automaton of size $8s + 4$, and each equivalent DFA has at least 2^s states. Moreover, each equivalent 2NFA has also at least 2^s states.*

The theorem is witnessed by the languages $L_s = \{a^{2^s}\}^*$ containing multiples of 2^s in unary notation. By a result in [107] it can be shown that every 2NFA accepting L_s must have at least 2^s states. The studies in [115] are complemented by answering the question whether or not for each unary regular language there exists an exponential gap between the sizes of deterministic pushdown automata and finite automata negatively.

Theorem 1.53. *Let $n \geq 1$ be an integer. Then there is a language L_n accepted by a DFA as well as by a 2NFA with 2^n states, and each equivalent deterministic pushdown automaton is at least of size $O(2^n/n^2)$.*

Finite-turn pushdown automata accepting (letter-)bounded languages are studied in [102], where a language is said to be bounded if it is a subset of $a_1^* a_2^* \cdots a_m^*$, for some alphabet $\Sigma = \{a_1, a_2, \dots, a_m\}$. It is known that arbitrary finite-turn pushdown automata accept exactly the ultralinear languages. While it will turn out in Theorem 1.92 that the increase in size when converting arbitrary PDAs accepting ultralinear languages to

finite-turn PDA cannot be bounded by any recursive function, for bounded languages an exponential trade-off is obtained.

Theorem 1.54. *Let $\Sigma = \{a_1, a_2, \dots, a_m\}$ be an alphabet. For every PDA of size s accepting a subset of $a_1^* a_2^* \cdots a_m^*$ there is an equivalent $(m-1)$ -turn pushdown automaton of size $2^{O(s^2)}$.*

In addition, in [102] a conversion algorithm is presented and the optimality of the construction is shown by proving tight lower bounds. Furthermore, the question of reducing the number of turns of a given finite-turn PDA is studied. Again, a conversion algorithm is provided which shows that in this case the trade-off is at most polynomial.

Theorem 1.55. *Let $n \geq 1$ be an integer. Then there is a bounded language $L_n \subseteq a_1^* a_2^* \cdots a_m^*$ which is accepted by an $(m-1)$ -turn pushdown automaton of size $2^{O(n)}$, but cannot be accepted by any PDA making strictly less than $m-1$ turns, and for each integer $k \geq m-1$, every k -turn pushdown automaton accepting L_n is at least of size $2^{O(n)}$, for sufficiently large n .*

Theorem 1.56. *For every normalform k -turn pushdown automaton of size s accepting a bounded language $L \subseteq a_1^* a_2^* \cdots a_m^*$ there is an equivalent normalform $(m-1)$ -turn pushdown automaton of size $O(m^6 s^{4[\log_2 k]+8})$. Therefore, for every arbitrary k -turn pushdown automaton of size s accepting a bounded language $L \subseteq a_1^* a_2^* \cdots a_m^*$ there is an equivalent $(m-1)$ -turn pushdown automaton of size $O(s^{6 \log k})$.*

1.3.2.2 Context-Free Grammars

Next we turn to compare the relative succinctness of pushdown automata and context-free grammars (CFG). Both descriptive systems are known to capture the context-free languages and, therefore, it is natural to ask whether a given context-free language can be more concisely described by an automaton or a grammar.

Definition 1.57. A *context-free grammar* (CFG) is a quadruple $G = (N, T, P, S)$, where N is a finite set of nonterminals, T is a finite set of terminal symbols, $S \in N$ is the axiom, and P is a finite set of productions of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$.

A context-free grammar G is in *Chomsky normalform* (CNF grammar) if every production in P is of the form $A \rightarrow BC$ or $A \rightarrow a$, for $A, B, C \in N$ and $a \in T$.

The *language generated* by a CFG $G = (N, T, P, S)$ is defined as

$$L(G) = \{w \in T^* \mid S \Rightarrow^* w\},$$

where \Rightarrow^* denotes the reflexive, transitive closure of the derivation relation \Rightarrow .

Several measures for CFGs have been proposed in the literature. A detailed treatment can be found in Section 1.4. Here we are particularly interested in s-measures that allow a comparison with PDAs. To this end, for a given context-free grammar G let $\text{size}(G)$ be the product of the number of productions and the length of the longest right-hand side of a production in P . Clearly, size is an s-measure for context-free grammars. On the other hand, the number of nonterminals, that is, $|N|$ is *not* an s-measure for CFGs in general. Assume contrarily it is. Then there is a recursive function g such that $\text{length}(G) \leq g(|N|, T)$, for every CFG G . But for the grammar $G = (\{S\}, \{a\}, \{S \rightarrow a^{g(1, \{a\})+1}\}, S)$ the value $\text{length}(G)$ exceeds $g(|N|, T)$. However, the number of nonterminals *is* an s-measure for context-free grammars in Chomsky normalform. In this case there are at most $|N|$ different left-hand sides of productions, and at most $|N|^2 + |T|$ right-hand sides. So, there are at most $|N|^3 + |N| \cdot |T|$ productions containing at most three nonterminal and terminal symbols, and we may choose $g(|N|, T) = k \cdot (|N|^3 + |N| \cdot |T|)$, where k is a mapping that gives the precise length of a production depending on the actual encoding alphabet, the number of nonterminals and terminal symbols. Note, that in this case the underlying descriptive system is a strict subfamily of the context-free grammars, the grammars in Chomsky normalform.

First, we present some results concerning the conversion of finite automata into CNF grammars. By standard construction every n -state DFA or NFA can be converted into a regular right-linear grammar with n nonterminals. This, in turn, can be converted into a CNF grammar with $n + i$ nonterminals, where i is the number of input symbols. The following lower bound has been obtained in [25].

Theorem 1.58. *Let $n \geq 1$ be an integer. Then there is a language L_n accepted by a DFA with $2^n + n + 1$ states and by an NFA with $2^n + n$ states, and every CNF grammar generating L_n has at least $\Omega(2^n/n)$ nonterminals.*

Upper and lower bound can significantly be improved in the unary case. Moreover, for DFAs they are tight in the order of magnitude [25].

Theorem 1.59. *Let $n \geq 1$ be an integer and A be an n -state DFA accepting a unary language. Then $\Theta(n^{1/3})$ nonterminals are sufficient and necessary*

in the worst case for a CNF grammar to generate $L(A)$. In case of an NFA $\Theta(n^{2/3})$ nonterminals are sufficient for a CNF grammar to generate $L(A)$.

Next, the conversion of pushdown automata into context-free grammars is considered. The so-called *triple construction* is the standard method for converting a PDA which accepts by empty stack into a CFG [67]. Given a PDA with n states and t stack symbols it constructs a CFG with $n^2t + 1$ nonterminals if $n > 1$, and n^2t nonterminals otherwise. The result has been complemented in [116], where it is shown by a modified triple construction that the resulting context-free grammar can be converted in a CNF grammar without introducing new nonterminals when the PDA is in normalform.

Theorem 1.60. *For every normalform PDA with n states and t stack symbols there is an equivalent CNF grammar with $n^2t + 1$ nonterminals. Therefore, for every arbitrary PDA of size s there is an equivalent CNF grammar with $O(s^2)$ nonterminals.*

In the worst case, this number of nonterminals is necessary even if the PDA is real time, deterministic, and accepts by empty stack [41].

Theorem 1.61. *Let $n, t \geq 1$ be integers. Then there is a language $L_{n,t}$ accepted by a deterministic real-time pushdown automaton by empty stack that has n states, t stack symbols, and size $O(nt)$, and each equivalent CFG has at least $n^2t + 1$ nonterminals if $n > 1$, and n^2t nonterminals otherwise.*

It follows that there are context-free languages which can be recognized by pushdown automata of size $O(nt)$, but which cannot be generated by context-free grammars of size smaller than $O(n^2t)$. Moreover, the standard construction for converting a pushdown automaton to an equivalent context-free grammar is optimal with respect to the number of nonterminals.

The situation is better for unary languages [115].

Theorem 1.62. *For every unary normalform deterministic pushdown automaton of size s there is an equivalent CNF grammar at most of size $O(s)$.*

Theorems 1.58, 1.59, and 1.59 dealt with the simulation of finite automata by CNF grammars. Since for unary context-free languages there are always equivalent DFAs and NFAs, the converse simulations are also worth studying. The following results are proven in [116].

Theorem 1.63. *For every unary CNF grammar with n nonterminals there is an equivalent NFA with $2^{2^{n-1}} + 1$ states.*

This upper bound is close to optimal.

Theorem 1.64. *Let $n \geq 1$ be an integer. Then there is a unary CNF with n nonterminals such that every equivalent NFA has at least $2^{n-1} + 1$ states.*

By Theorem 1.63, given a unary CNF with n nonterminals there is an equivalent NFA with $2^{O(n)}$ states. This automaton can be transformed into a DFA applying the powerset construction or the determinization procedure for unary automata presented in [22]. In both cases, the number of states of the resulting DFA is bounded by a function which grows at least double exponential in n . In [116] it is proved that this cost can drastically be reduced.

Theorem 1.65. *For every unary CNF grammar with n nonterminals there is an equivalent DFA with 2^{n^2} states.*

Note, in the particular case $n = 1$ the upper bound given in Theorem 1.65 does not hold. It is not difficult to show that the only non-empty languages generated by unary CNF grammars with one variable are $\{a\}$ and $\{a^k \mid k \geq 1\}$. The minimal DFAs accepting these languages have three and two states, respectively. However, the upper bound stated in Theorem 1.65 is tight.

Theorem 1.66. *For infinitely many integers $n \geq 1$, there is a unary CNF with n nonterminals such that every equivalent DFA has at least $2^{O(n^2)}$ states.*

The relations between the subfamily of finite-turn pushdown automata accepting (letter-)bounded languages and CNF grammars are studied in [102].

Theorem 1.67. *For every CNF grammar with n nonterminals generating a bounded language $L \subseteq a_1^* a_2^* \cdots a_m^*$ there is an equivalent normalform $(m - 1)$ -turn PDA with $2^{O(n)}$ states and $O(1)$ stack symbols.*

For the situation where the given grammar is not necessarily in Chomsky normalform the following result has been shown.

Theorem 1.68. *For every context-free grammar of size s generating a bounded language $L \subseteq a_1^* a_2^* \cdots a_m^*$ there is an equivalent normalform $(m-1)$ -turn PDA with $2^{O(s)}$ states and $O(1)$ stack symbols.*

The lower bound is given by the next theorem.

Theorem 1.69. *Let $n \geq 1$ be an integer. Then there is a unary language L_n generated by a CNF grammar with $n+1$ nonterminals such that for each integer $k \geq 1$, every normalform k -turn pushdown automaton accepting L_n is at least of size $2^{O(n)}$, for sufficiently large n .*

So far, essentially (deterministic) pushdown automata and context-free grammars (in Chomsky normalform) have been discussed. A sub-class of CFGs that characterize the deterministic context-free languages are LR(k) grammars, where $k \geq 1$, can be seen as the length of the lookahead of a corresponding LR(k) parser. Already the class of LR(1) grammars characterizes the deterministic context-free languages. The use of a longer lookahead k does not increase their generative capacity. But from a descriptive complexity point of view the question whether a longer lookahead can reduce the size of a grammar has been answered in [96] affirmatively. The practical relevance of these results is immediate. A sequence of languages L_n is presented such that there is a progressive trade-off in the size of the LR(k) grammars as the length of the lookahead varies.

Theorem 1.70. *Let $n \geq 2$ and $k \geq 0$ be integers satisfying $k \leq n - 9 \log n$. Then there is a language L_n such that every LR(k) grammar generating L_n is at least of size $2^{\Theta(n-k)}$.*

1.4 Measuring Resources

The investigation of several aspects of measuring descriptors with s-measures is responding to an interest to optimize the space requirements. Basically, some resources which are recursively related to the length of the description are measured, and the relative succinctness of different types of descriptors is studied. But what makes the difference between two types of descriptors? Roughly speaking, it is their different equipment with resources. For example, the difference between a DFA and an NFA is made by the resource *nondeterminism*, or the difference between an NFA and a PDA is caused by the resource *stack*. So, in more general terms, we fix some, say, structural resources and study their descriptive power by measuring

other resources. This immediately raises the question which resources are structural and which can be measured. Here, the only restriction we impose is to have descriptive systems with recursive complexity measures. Given a descriptor its complexity must be computable. So, even though they are naturally motivated the s-measures are special cases, only. This section is devoted to descriptive systems measured by measures which are *not* recursively related to their length.

A context-free grammar $G = (N, T, P, S)$ is *right-linear* or *regular* if every production in P is of the form $A \rightarrow uB$ or $A \rightarrow u$, for $A, B \in N$ and $u \in T^*$. The states of a finite automaton correspond roughly to the number of nonterminals of a regular grammar and *vice versa*. This leads to the idea of considering the number of nonterminals as a measure for arbitrary context-free grammars. In the forthcoming we stick with context-free grammars and subclasses as descriptive systems, while the below given definitions easily generalize to arbitrary phrase structure grammars. For a context-free grammar $G = (N, T, P, S)$, we define the following three measures [52]:

$$\text{var}(G) = |N|,$$

$$\text{prod}(G) = |P|,$$

and

$$\text{symb}(G) = \sum_{(A \rightarrow \alpha) \in P} (|A| + |\alpha| + 1).$$

In order to clarify the definitions we present an example [52].

Example 1.71. Let $n \geq 1$ be an integer. Consider the context-free grammar $G = (\{S, A\}, \{a\}, P, S)$ with the three productions $S \rightarrow A^n$, $A \rightarrow a$, and $A \rightarrow aa$. Then $L(G) = \{a^i \mid n \leq i \leq 2n\}$ and $\text{var}(G) \leq 2$, $\text{prod}(G) \leq 3$, and $\text{symb}(G) \leq n + 9$. In fact, there exist an equivalent context-free grammar G' with $\text{var}(G') = 1$ because language $L(G)$ is finite and can be easily generated by a CFG with one nonterminal only. \square

As already mentioned previously, the measure var is not an s-measure. The same holds true for the measure prod , while symb is obviously recursively related to the size of the CFG. Observe, that it heavily depends on the underlying descriptive system, if a measure becomes an s-measure. For instance, the number of variables is *not* even an s-measure for regular grammars in general, but it becomes an s-measure if the descriptive system is chosen to be that of *regular grammars in normalform*, that is, every

production is of the form $A \rightarrow aB$ or $A \rightarrow a$. Further measures based on other criteria induced by grammatical levels, derivation trees, derivation steps, etc., are introduced and studied in [51, 52, 54, 55].

Before we summarize some results on the measures **var** and **prod** we find it worth mentioning, that basic algorithmic problems for most of the measures on context-free grammars and languages are undecidable. For instance, to determine the complexity of a given language, to construct a minimal equivalent grammar, to decide minimality of a given grammar, and so on. For further readings on this topic we refer to [52, 51, 54, 19]. Now we turn to the first result on the number of variables for context-free grammars [51].

Theorem 1.72 (var for CFL). *Let $n \geq 1$ be an integer. Then there exists a regular language L_n over a binary alphabet, such that $\text{var}(G) \geq n$ for every context-free grammar generating L_n .*

Thus, the **var**-measure for context-free grammars induces a dense and strict hierarchy of increasing levels of difficulties. This property of inducing a dense hierarchy is known in the literature as the *connectedness property* with respect to an alphabet [52]. If we consider the classification of context-free grammars in terms of the number of productions, we find a similar statement as above [52]. For every alphabet T and all integers $n \geq 1$ there is a finite language L_n over T , such that $\text{var}(G) \geq n$ for every context-free grammar generating L_n . Thus, the **prod**-measure is also connected, even for unary alphabets. The witness language to show the result on the number of productions is the finite language $\{a^{2^i} \mid 0 \leq i \leq n-1\}$. On the other hand, if the alphabet is unary, the situation for **var** changes drastically [51].

Theorem 1.73 (var for unary CFL). *Let L be a unary context-free grammar. Then two nonterminals are sufficient and necessary in the worst case for an equivalent context-free grammar.*

Concerning the smallest level of the number of nonterminals, in [53] it is mentioned that the language generated by $G = (\{S\}, T, P, S)$, where $P = \{S \rightarrow \alpha \mid \alpha \in F\}$ for some finite $F \subseteq (\{S\} \cup T)^*$, is equal to the iterated S -substitution of F , that is, $L(G) = F^{\uparrow^S}$. Here for a letter a and two languages L_1 and L_2 , the a -substitution of L_2 in L_1 , denoted by $L_1 \uparrow^a L_2$, is defined by

$$L_1 \uparrow^a L_2 = \{u_1 v_1 u_2 \dots u_k v_k u_{k+1} \mid u_1 a u_2 a \dots a u_{k+1} \in L_1, \\ a \text{ does not occur in } u_1 u_2 \dots u_{k+1}, \text{ and } v_1, v_2, \dots, v_k \in L_2\},$$

and the *iterated a-substitution* of language L , denoted by L^{\uparrow^a} , is defined by

$$L^{\uparrow^a} = \{ w \in L \cup (L \uparrow^a L) \cup (L \uparrow^a L \uparrow^a L) \cup \dots \mid \text{word } w \text{ has no occurrence of letter } a \},$$

where any further bracketing is omitted since a -substitution is obviously associative. The definition of iterated substitution expressions gives a nice and convenient way to specify context-free languages in terms of expressions. For the characterization of context-free languages by means of expressions we refer also to [106] and [135]. Although these approaches are very similar there are subtle differences; see the former reference for the relation between McWhirter's expressions and Gruska's substitution model. The relation between auxiliary symbols in substitution expressions and the number of nonterminals in context-free grammars is discussed in [53] in more detail (cf. [44]).

Next, let us restrict our attention to regular grammars. Here the the smallest level with respect to the number of nonterminals is more handy to describe. One can easily show that if a regular language L is generated by regular grammar with one nonterminal, then there exist two regular sets R_1 and R_2 such that $L = R_1^* R_2$. This result can further be generalized to higher levels.

We continue with some results on the comparison between context-free and regular grammars. The natural question arises, whether the former have advantages compared with the latter according to the measures considered so far? This question was answered in [52], where the following result was shown.

Theorem 1.74 (var and prod for CFL versus REG). *There is a regular language L such that any regular grammar generating L has strictly more nonterminals than a minimal equivalent context-free grammar. The statement remains valid for the number of productions.*

Observe, that the strict increase in complexity already happens for non-self-embedding context-free grammars. Here a context-free grammar is *self-embedding* if there is a nonterminal X such that there is a derivation $X \Rightarrow^* \alpha X \beta$, for both non-empty α and β . It is well known that non-self-embedding context-free grammars generate regular languages only. A closer look on the previous theorem reveals even more. In fact, an easy example given in [52] shows that the difference between the number of nonterminals in equivalent context-free and regular grammars can be arbitrarily large.

For all integers $n \geq 1$, consider the language $L_n = (a^*bab^*)^n$. Two nonterminals are sufficient for a context-free grammar while each regular grammar needs at least $2n$ nonterminals in order to generate L_n . Analogously one can show that four productions are sufficient for a context-free grammar while each regular grammar needs at least $2n$ productions. Further examples showing that the measures can yield essential different classifications, when the underlying grammar is varied, can be found in [52]. In this way so called *bounded* complexity measure results are obtained.

Another field of research that is related to bounded complexity measures is the study of the descriptive complexity of various types of grammars that can be used to describe context-free languages. For example, context-free grammars and their normalform restrictions such as λ -free normal form, Chomsky normalform, Greibach normalform, position restricted grammars, etc. Transformations of context-free grammars into normalforms may change their complexity with respect to the measures under consideration. In a series of papers these questions were addressed [19, 44, 52, 80, 81, 117, 118]. Some of the most interesting results are presented next. We start with some results based on restricted context-free grammars. A context-free grammar is *restricted* if it is λ -free and does not contain any unit productions.

Theorem 1.75 (Bounded complexity for restricted CFGs). *Let G be a context-free grammar. Then there is an equivalent restricted context-free grammar G' such that*

- (1) $\text{var}(G') \leq \text{var}(G)$,
- (2) $\text{prod}(G') \leq \frac{1}{2} \cdot \text{prod}^2(G)$,
- (3) $\text{symb}(G') \leq \frac{1}{2} \cdot \text{symb}^2(G)$.

The latter two bounds cannot be improved by more than a constant.

If each two equivalent minimal descriptors from different descriptive systems have the same complexity with respect to a measure, this measure is called *dense*. So, the measure var is dense for context-free and restricted context-free grammars. In the next theorem we will see that this is not true in general, because the var is *not* dense for (restricted) context-free and context-free grammars in Greibach normalform.

A context-free grammar $G = (N, T, P, S)$ is in *Greibach normalform* [43] if every production in P is of the form $A \rightarrow a\alpha$, for $A \in N$, $a \in T$, and $\alpha \in N^*$.

Theorem 1.76 (Bounded complexity for CFGs in normalforms).

Let G be a restricted context-free grammar. Then

- (1) there is an equivalent context-free grammar G' in Chomsky normalform such that $\text{symb}(G') \leq 7 \cdot \text{symb}(G)$,
- (2) there is an equivalent context-free grammar G' in Greibach normalform such that $\text{var}(G') \leq 2 \cdot \text{var}(G)$ and this bound is the best possible, $\text{prod}(G') \in O(\text{prod}^3(G) \cap \Omega(\text{prod}^2(G)))$, and $\text{symb}(G') \in O(\text{symb}^3(G) \cap \Omega(\text{symb}^2(G)))$.

In the remainder, we turn our attention to the relation between context-free grammars and other grammars from the Chomsky hierarchy such as monotone grammars, that is, grammars with productions whose right-hand side is not shorter than the left-hand side, or arbitrary phrase structure grammars with respect to the measures under consideration. The next theorem shows that the gap between the number of nonterminals for a context-free grammar and a monotone grammar representation can be arbitrarily large. Consider the languages $L_n = \bigcup_{i=1}^{n-1} b(a^i b)^+$, for all integers $n \geq 3$. By standard arguments one can show that every context-free grammar generating L_n needs at least n nonterminals. On the other hand, the monotone grammar $G_n = (\{S, A\}, \{a, b\}, P_n, S)$ with

$$P_n = \{S \rightarrow ba^i b, S \rightarrow Aa^i b, Aa^i b \rightarrow Aa^i ba^i b, A \rightarrow b \mid 1 \leq i \leq n-1\}$$

generates the L_n with two nonterminals only. This result can be slightly strengthened as follows:

Theorem 1.77. *Let $n \geq 1$ be an integer. Then there exists a regular language L_n such that every context-free grammar generating L_n has at least n nonterminals, and there is an equivalent monotone grammar with 2 nonterminals.*

A similar situation appears if we consider the measure prod . Note that the witness language for the next theorem is a finite language. Further readings on the measure prod for finite languages can be found in [16, 18, 17].

Theorem 1.78. *Let $n \geq 1$ be an integer. Then there exists a finite language L_n such that every context-free grammar generating L_n has at least n productions, and there is an equivalent monotone grammar with 5 nonterminals.*

The argument for this statement is not too complicated [114]. We have already mentioned that every context-free grammar G generating the (finite) language $L_n = \{a^{2i} \mid 0 \leq i \leq n-1\}$ has at least n productions. Now we consider the modified language

$$L'_n = \{b^{n-i-1}a^{2i}b^{i+1} \mid 0 \leq i \leq n-1\}.$$

We have $L_n = h(L'_n)$ for the erasing homomorphism $h : \{a, b\}^* \rightarrow \{a\}^*$ defined by $h(a) = a$ and $h(b) = \lambda$. Moreover, every context-free grammar generating $h(L(G))$ clearly needs at most as many productions as G has. So, since $n \leq \text{prod}(G)$ every context-free grammar G' generating L'_n has at least n productions. In order to conclude that n productions are enough we construct the grammar $G = (\{S\}, \{a, b\}, P, S)$ with the set of productions

$$P = \{S \rightarrow b^{n-i-1}a^{2i}b^{i+1} \mid 0 \leq i \leq n-1\}.$$

Grammar G' generates L'_n and has n productions. Finally, it is easy to see that the monotone grammar $G' = (\{S, T, A\}, \{a, b\}, P', S)$ with productions

$$P' = \{S \rightarrow T^{n-1}ab, T \rightarrow A, T \rightarrow b, Aa \rightarrow aaA, Ab \rightarrow bb\}$$

also generates L'_n . Since P contains only five elements the statement follows.

1.5 Non-Recursive Trade-Offs

In order to motivate the main topic of this section we first deduce a property of any descriptive system \mathcal{S} when it is measured by an s-measure c . Let $D \in \mathcal{S}$ be a descriptor. Since c is an s-measure there is a recursive function g such that $\text{length}(D) \leq g(c(D), \text{alph}(D))$. But this implies that with respect to $\text{alph}(D)$ there are only finitely many descriptors in \mathcal{S} having the same size as D . Otherwise, applying g to infinitely many descriptors would yield to the same result. But for any coding alphabet there are only finitely many descriptors whose length does not exceed $g(c(D), \text{alph}(D))$. So, we know that for any size, \mathcal{S} contains only finitely many descriptors over the same alphabet.

Assume now there are two descriptive systems \mathcal{S}_1 and \mathcal{S}_2 , and two s-measures c_1 for \mathcal{S}_1 and c_2 for \mathcal{S}_2 . Given a descriptor from \mathcal{S}_1 a natural question is for the maximal blow-up in complexity when this descriptor is converted into an equivalent one from \mathcal{S}_2 . Clearly, if a general upper bound for the trade-off is known, the blow-up is given by that function.

Our question is somehow simpler. We are not interested in a general upper bound but in an upper bound that may use the given alphabet as an additional parameter, say $h(n, \Sigma)$. We can precisely determine h as follows. For all n and alphabets Σ , let $\mathcal{D}_{n, \Sigma}$ denote the finite subset of all descriptors from \mathcal{S}_1 over alphabet Σ whose complexity is n . For each $D_1 \in \mathcal{D}_{n, \Sigma}$ set $m(D_1) = \min\{c_2(D_2) \mid D_2 \in \mathcal{S}_2(L(D_1))\}$. Then $h(n, \Sigma)$ is set to $\max\{m(D_1) \mid D_1 \in \mathcal{D}_{n, \Sigma}\}$. So, we have a function at hand that answers our question. Unfortunately, it may happen that this function is not effectively computable. What does this mean? This means that the size blow-up caused by such a conversion cannot be bounded above by *any* recursive function. With other words, one can choose an arbitrarily large recursive function but the gain in economy of description eventually exceeds it. This qualitatively new phenomenon, nowadays known as *non-recursive trade-off*, was first observed by Meyer and Fischer [109] between context-free grammars and finite automata.

In the sequel we often use the following second property of measures.

Definition 1.79. Let \mathcal{S} be a descriptive system with s-measure c . If for any alphabet Σ , the set of descriptors in \mathcal{S} describing languages over Σ is recursively enumerable in order of increasing size, then c is said to be an *sn-measure*.

In fact, the non-recursive trade-offs are independent of particular sn-measures. Any two sn-measures c_1 and c_2 for some descriptive system \mathcal{S} are related by a function

$$h(n, \Sigma) = \max\{c_2(D) \mid D \in \mathcal{S} \text{ with } c_1(D) = n \text{ and } \text{alph}(D) = \Sigma\}.$$

By the properties of sn-measures, h is recursive. So, a non-recursive trade-off exceeds any difference caused by applying two sn-measures.

1.5.1 Proving Non-Recursive Trade-Offs

Before we present examples of non-recursive trade-offs, we turn to the question of how to prove them. Roughly speaking, most of the proofs appearing in the literature are basically relying on one of two different schemes. One fundamental technique is due to Hartmanis [59]. In [60] a generalization is developed that relates semi-decidability to trade-offs. Next we present a slightly generalized and unified form of this technique [87].

Theorem 1.80. Let \mathcal{S}_1 and \mathcal{S}_2 be two descriptive systems for recursive languages such that any descriptor D in \mathcal{S}_1 and \mathcal{S}_2 can effectively be con-

verted into a Turing machine that decides $L(D)$, and let c_1 be a measure for \mathcal{S}_1 and c_2 be an sn-measure for \mathcal{S}_2 . If there exists a descriptive system \mathcal{S}_3 and a property P that is not semi-decidable for descriptors from \mathcal{S}_3 , such that, given an arbitrary $D_3 \in \mathcal{S}_3$, (i) there exists an effective procedure to construct a descriptor D_1 in \mathcal{S}_1 , and (ii) D_1 has an equivalent descriptor in \mathcal{S}_2 if and only if D_3 does not have property P , then the trade-off between \mathcal{S}_1 and \mathcal{S}_2 is non-recursive.

Let us give evidence that the theorem is true. Assume contrarily that the trade-off is bounded by some recursive function f . Let D_1 be a descriptor in \mathcal{S}_1 . If D_1 has an equivalent descriptor D_2 in \mathcal{S}_2 , then $c_2(D_2) \leq f(c_1(D_1))$. Since f and c_1 are recursive, the value $f(c_1(D_1))$ can be computed. Next, we can recursively enumerate the finite number of descriptors in \mathcal{S}_2 , whose underlying alphabet is $\text{alph}(D_1)$ and whose size is at most $f(c_1(D_1))$. All these descriptors can effectively be converted into Turing machines that decide for any input whether it belongs to the described languages. The same holds for the descriptor D_1 . By comparing the enumerated descriptors with D_1 on successive inputs over the alphabet $\text{alph}(D_1)$, we can detect whether *none* of the descriptors is equivalent to D_1 . As a result, a Turing machine is constructed that halts if and only if D_1 has no equivalent descriptor in \mathcal{S}_2 . So, the set $R = \{D_1 \in \mathcal{S}_1 \mid D_1 \text{ has no equivalent descriptor in } \mathcal{S}_2\}$ is recursively enumerable. Now the theorem follows due to the following contradiction. Given a descriptor $D_3 \in \mathcal{S}_3$ we construct the descriptor D_1 in \mathcal{S}_1 , and semi-decide whether it belongs to the set R . If the answer is in the affirmative, there is no equivalent descriptor in \mathcal{S}_2 and, thus, D_3 has property P .

Example 1.81. Let \mathcal{S}_1 be the family of linear context-free grammars, and \mathcal{S}_2 be the set of deterministic finite automata. Clearly, both descriptive systems meet the preconditions of Theorem 1.80. Since the regularity of linear context-free grammars is not semi-decidable [13], we set \mathcal{S}_3 to be \mathcal{S}_1 and property P is to be a descriptor describing a non-regular language. So, any linear context-free grammar D_1 has an equivalent DFA if and only if language $L(D_1)$ is regular, that is, if it does not have property P . We conclude that the trade-off between linear context-free grammars and DFAs is non-recursive. \square

Since we may apply Theorem 1.80 for any pairs of descriptive systems whose first component represents the linear context-free and whose second component represents the regular languages the following theorem is derived from Example 1.81.

Theorem 1.82. *The trade-offs between linear context-free grammars and deterministic finite automata, between one-turn pushdown automata and nondeterministic finite automata, etc., are non-recursive.*

On the one hand, the method presented can serve as a powerful tool. Several known proofs are simplified. Some more or less new non-recursive trade-offs follow immediately by known undecidability results. On the other hand, to apply Theorem 1.80 the crucial hard part is to find suitable descriptive systems \mathcal{S}_3 having the required properties. The example presented before considered linear context-free languages for which regularity is not semi-decidable. Another valuable descriptive system is the set of Turing machines for which only trivial problems are decidable and a lot of problems are not semi-decidable. When Theorem 1.80 is applied, one has to be a little bit careful about the negation of property P . For example, finiteness is not semi-decidable for Turing machines. Not finite means infinite, which is also not semi-decidable for Turing machines. On the other hand, emptiness is not semi-decidable, but its negation is, that is, whether the Turing machine accepts at least one input.

In order to utilize non-semi-decidable properties of Turing machines in [58] complex Turing machine computations have been encoded in small grammars. These encodings and variants thereof are of tangible advantage for our purposes. Basically, we consider *valid computations of Turing machines*. Roughly speaking, these are histories of accepting Turing machine computations. It suffices to consider deterministic Turing machines with one single tape and one single read-write head. Without loss of generality and for technical reasons, we assume that the Turing machines can halt only after an odd number of moves, accept by halting, make at least three moves, and cannot print blanks. A valid computation is a string built from a sequence of configurations passed through during an accepting computation.

Let Q be the state set of some Turing machine M , where q_0 is the initial state, $T \cap Q = \emptyset$ is the tape alphabet containing the blank symbol, and $\Sigma \subset T$ is the input alphabet. Then a configuration of M can be written as a word of the form T^*QT^* such that $t_1t_2 \cdots t_iqt_{i+1} \cdots t_n$ is used to express that M is in state q , scanning tape symbol t_{i+1} , and t_1, t_2 to t_n is the support of the tape inscription. For the purpose of the following, valid computations are now defined in three different forms:

(1) $\text{VALC}_A(M)$ is the set of strings of the form

$$\$w_1\$w_2^R\$w_3\$w_4^R\$ \cdots \$w_{2n-1}\$w_{2n}^R\$.$$

(2) $\text{VALC}_C(M)$ is the set of strings of the form

$$\$w_1\$w_2\$ \cdots \$w_{2n}\$.$$

(3) $\text{VALC}_R(M)$ is the set of strings of the form

$$\$w_1\$w_3\$ \cdots \$w_{2n-1}\#w_{2n}^R\$ \cdots \$w_4^R\$w_2^R\$.$$

In all three cases, $\$, \# \notin T \cup Q$, $w_i \in T^*QT^*$ are configurations of M , w_1 is an initial configuration of the form $q_0\Sigma^*$, w_{2n} is an halting, that is, accepting configuration, and w_{i+1} is the successor configuration of w_i .

The set of *invalid computations* $\text{INVALC}_i(M)$, for $i \in \{A, C, R\}$, is the complement of $\text{VALC}_i(M)$ with respect to the alphabet $\{\#, \$\} \cup T \cup Q$.

Later we exploit a result on the following decomposition of $\text{VALC}_A(M)$: $\text{VALC}_{A1}(M)$ is the set of strings of the form $\$w_1\$w_2^R\$ \cdots \$w_{2n-1}\$w_{2n}^R\$$, where w_1 is an initial and w_{2n} is an accepting configuration, and w_{2i+1} is the successor configuration of w_{2i} , for $1 \leq i \leq n-1$. $\text{VALC}_{A2}(M)$ is the set of strings of the form $\$w_1\$w_2^R\$ \cdots \$w_{2n-1}\$w_{2n}^R\$$, where w_1 is an initial and w_{2n} is an accepting configuration, and w_{2i} is the successor configuration of w_{2i-1} , for $1 \leq i \leq n$.

The next lemma summarizes some of the important properties of valid computations.

Lemma 1.83. *Let M be a Turing machine and $i \in \{A, C, R\}$.*

- (1) *If $L(M)$ is finite, then $\text{VALC}_i(M)$ is finite.*
- (2) *If $L(M)$ is finite, then $\text{INVALC}_i(M)$ is regular.*
- (3) *If $L(M)$ is infinite, then $\text{VALC}_i(M)$ is not context free.*
- (4) *If $L(M)$ is infinite, then $\text{INVALC}_i(M)$ is not regular.*
- (5) *$\text{VALC}_R(M)$ can be represented by the intersection of two deterministic linear context-free languages, such that both deterministic pushdown automata and both linear context-free grammars can effectively be constructed from M .*
- (6) *$\text{VALC}_A(M)$ can be represented by the intersection of two deterministic context-free languages, such that both deterministic pushdown automata can effectively be constructed from M .*
- (7) *$\text{VALC}_{A1}(M)$ and $\text{VALC}_{A2}(M)$ are deterministic context-free languages, such that their deterministic pushdown automata can effectively be constructed from M .*
- (8) *$\text{INVALC}_i(M)$ is a linear context-free language, such that its grammar can effectively be constructed from M .*

Assertions (1), (2), (4), (6), and (7) are immediate observations. Assertion (3) is shown by pumping lemma [58]. The two deterministic linear context-free languages for (5) are constructed in [2]. For INVALC_A , assertion (8) has been shown in [58]. Similarly, it can be proved for INVALC_R . For INVALC_C observe that the complement of the language $\{w\$w \mid w \in \{a, b\}^*\}$ is linear context free [34, 123].

Before we discuss some more applications and results in detail, we turn to the generalized and unified form of the second technique that emerges from known proofs.

Theorem 1.84. *Let \mathcal{S}_1 and \mathcal{S}_2 be two descriptive systems, c_1 be a measure for \mathcal{S}_1 and c_2 be an sn-measure for \mathcal{S}_2 . If there exists a recursive function $\varphi : \mathbb{N} \rightarrow \mathbb{N}$, such that given an arbitrary Turing machine M ,*

- (i) there exists an effective procedure to construct a descriptor D_1 in \mathcal{S}_1 ,*
- (ii) if M halts on blank tape, then D_1 has an equivalent descriptor in \mathcal{S}_2 ,*

and for all equivalent descriptors D_2 in \mathcal{S}_2 it holds $\varphi(c_2(D_2)) \geq t$, where t is the number of tape cells used by M , then the trade-off between \mathcal{S}_1 and \mathcal{S}_2 is non-recursive.

Again, let us give evidence that the theorem is true. Assume contrarily that the trade-off is bounded by some recursive function f . Given some Turing machine M , we first construct a descriptor D_1 in \mathcal{S}_1 . Since c_1 and f are recursive, the value $f(c_1(D_1))$ can be computed. Next, we can recursively enumerate the finite number of descriptors D_2 in \mathcal{S}_2 , whose underlying alphabet is $\text{alph}(D_1)$ and whose size is at most $f(c_1(D_1))$. Since c_2 and φ are recursive, their maximum value $t' = \varphi(c_2(D_2))$ can be computed. If Turing machine M halts on blank tape, then there is at least one descriptor D_2 in the list such that $\varphi(c_2(D_2)) \geq t$, where t is the number of tape cells used by M . Since $t' \geq \varphi(c_2(D_2)) \geq t$ it suffices to simulate M on blank tape until it exceeds the tape cell bound t' , or it runs into a loop without exceeding the tape cell bound, or it halts, in order to decide whether M halts on blank tape at all. From the contradiction follows that the trade-off is non-recursive.

The preconditions of the previous theorem are different compared with Theorem 1.80. In particular, it is not requested that the descriptors can effectively be converted into Turing machines, and the descriptive systems need not to be for recursive languages.

The next trade-off exploits the following crucial lemma on the size of deterministic pushdown automata [133].

Lemma 1.85. *If for some deterministic pushdown automaton A with state set Q and set of stack symbols T the string w is the shortest string such that wa and wb are accepted, then there is a positive constant k such that $|Q| \cdot |T| \geq (\log |w|)^k$.*

Example 1.86. The trade-off between unambiguous context-free grammars and deterministic pushdown automata is non-recursive.

Let M be an arbitrary Turing machine with state set Q , tape alphabet T , and initial state q_0 . From M two deterministic pushdown automata A_1 and A_2 for the languages $\text{VALC}_{A_1}(M)$ and $\text{VALC}_{A_2}(M)$ can effectively be constructed. Since the deterministic context-free languages are effectively closed under intersection with regular sets, the languages $L_1 = \$q_0\$(T^*QT^*\$)^* \cap \text{VALC}_{A_1}(M)$ and $L_2 = \$q_0\$(T^*QT^*\$)^* \cap \text{VALC}_{A_2}(M)$ are also effective deterministic context-free languages. Let a and b be new symbols. Then we can construct an unambiguous context-free grammar D_1 for the language $L = L_1a \cup L_2b$.

Now assume that M halts on empty input. Then the intersection $L_1 \cap L_2$ contains exactly one string v , which is the valid computation of M on empty input. Moreover, there does not exist a string of length $2|v|$ which is a prefix in both languages. So, by inspecting the first $2|v|$ input symbols a deterministic pushdown automaton can decide to which language the input may still possibly belong. We conclude that there exists a deterministic pushdown automaton D_2 for L . By Lemma 1.85 we obtain that the product of the number of states and the number of stack symbols of any equivalent deterministic pushdown automaton is greater than $(\log |v|)^k$, for some positive k . Therefore, the recursive function φ for the application of Theorem 1.84 can easily be determined. \square

1.5.2 A Compilation of Non-Recursive Trade-Offs

Before we turn to collect some important results in a compilation of non-recursive trade-offs, we draw the attention to a general question in connection with descriptive complexity. We have seen that there is a non-recursive trade-off between linear context-free languages and finite automata. On the other hand, the trade-off between deterministic context-free languages and finite automata is recursive [129]. So, what makes the difference between linear and deterministic context-free languages? Though the answer might be the power of nondeterminism, a closer look at the problem might clarify descriptive complexity to be a finer apparatus compared

with computational complexity. This observation is emphasized by showing that, for example, between two separated Turing machine space classes there is always a non-recursive trade-off (see also [60]).

Example 1.87. Denote the languages accepted by deterministic Turing machines obeying a space bound $s(n)$ by $\text{DSPACE}(s(n))$. If $\text{DSPACE}(s_1(n)) \supset \text{DSPACE}(s_2(n))$ for a constructible bound s_1 , then the trade-off between s_1 -space bounded Turing machines and s_2 -space bounded Turing machines is non-recursive. \square

The example can be shown with the help of Theorem 1.80. It can be modified to work for several other Turing machine classes which are separated by bounding some resource. For example, $\text{P} \neq \text{NP}$ if and only if the trade-off between NP and P is non-recursive (see [60–62] for further relations between descriptonal and computational complexity).

Now we turn to results that deal mainly with descriptonal systems at the lower end of the Chomsky hierarchy. A cornerstone of descriptonal complexity is the result of Meyer and Fischer [109] who showed for the first time a non-recursive trade-off. It appears between context-free grammars and finite automata. Nowadays, we can derive that result immediately from the non-recursive trade-off between linear context-free grammars and finite automata, but originally, the proof follows the scheme presented in Theorem 1.84.

Theorem 1.88. *The trade-off between context-free grammars and finite automata is non-recursive.*

Since, for example, the sizes of context-free grammars and pushdown automata are recursively related, there is a non-recursive trade-off between pushdown automata and finite automata, too. Similarly, this remark holds for several results below. Once a non-recursive trade-off has been shown, it is interesting to consider language families in between. We know already by Example 1.86 that there is a non-recursive trade-off between unambiguous context-free grammars and deterministic pushdown automata. Considering the remaining gap between general and unambiguous context-free grammars we encounter the problem that unambiguity is not semi-decidable for context-free grammars. Therefore, we cannot enumerate the unambiguous context-free grammars. This implies that there does not exist any sn -measure for them and, thus, neither Theorem 1.80 nor Theorem 1.84 can be applied directly. However, by a variant of the technique of Theorem 1.84 the gap has been closed in [126]. The proof uses Ogden’s lemma [113] in

order to solve the crucial part to show that the sizes of unambiguous grammars depend on the lengths of strings in certain witness languages.

Theorem 1.89. *The trade-off between context-free grammars and unambiguous context-free grammars is non-recursive.*

By measuring the amount of ambiguity and nondeterminism in pushdown automata in [12] and [64] infinite hierarchies in between the deterministic and nondeterministic context-free languages are obtained. The classes of pushdown automata with ambiguity and branching bounded by k are denoted by $\text{PDA}(\alpha \leq k)$ and $\text{PDA}(\beta \leq k)$, where branching is a measure of nondeterminism. If both resources are bounded at the same time, we write $\text{PDA}(\alpha \leq k, \beta \leq k')$. Intuitively, the corresponding language families are close together. Nevertheless, there are non-recursive trade-offs between the levels of the hierarchies.

Theorem 1.90. *Let $k \geq 1$ be an integer. Then the following trade-offs are non-recursive:*

- (1) *between $\text{PDA}(\alpha \leq k + 1, \beta \leq k + 1)$ and $\text{PDA}(\alpha \leq k)$, and*
- (2) *between $\text{PDA}(\alpha \leq 1, \beta \leq k + 1)$ and $\text{PDA}(\beta \leq k)$.*

The proofs of both theorems are similar generalizations of the proof of Theorem 1.89. They follow the scheme of Theorem 1.84.

In [59, 60] simple new proofs of some of the presented theorems have been given. The technique of these proofs follows the scheme of Theorem 1.80. Furthermore, Hartmanis [59] raised the question whether the trade-off between two descriptive systems is caused by the fact that in one system it can be proved what is accepted, but that no such proofs are possible in the other system. For example, consider descriptive systems for the deterministic context-free languages. It is easy to verify whether a given pushdown automaton is deterministic, but there is no uniform way to verify that a nondeterministic pushdown automaton accepts a deterministic context-free language. Sticking with this example, one may ask whether the trade-off is affected if so-called *verified nondeterministic pushdown automata* are considered which come with an attached proof that they accept deterministic languages. The following theorem summarizes the results.

Theorem 1.91. *The following trade-offs are non-recursive:*

- (1) *between verified and deterministic pushdown automata,*
- (2) *between pushdown automata and verified pushdown automata, and*

(3) *between verified ambiguous and unambiguous context-free grammars.*

So far, some of the presented results can be shown by using some variant of the valid computations. In fact, whenever the expressive capacities of two systems can be separated by valid computations, the proof of non-recursive trade-offs is more or less immediate by an application of Theorem 1.80. For example, consider the Boolean closure of context-free languages and the context-free languages. The two systems are separated by VALC_A . The same is true for context-sensitive and context-free grammars and many other pairs of systems. The situation changes if the weaker system also contains a descriptor for the valid computations. Consider Example 1.87 which induces a non-recursive trade-off between space bounded Turing machine classes and deterministic context-sensitive grammars ($\text{DSPACE}(n)$).

Coming back to the observation that space might be a rough measure of complexity, it is interesting and natural to investigate infinite hierarchies of separated language classes where, intuitively, the classes are closer together. For example, $\text{LL}(k+1)$ grammars are known to describe strictly more languages than $\text{LL}(k)$ grammars, that is, the length of the lookahead induces an infinite hierarchy. Nevertheless, the trade-offs between the levels of the hierarchy are recursive [5]. On the other hand, we have seen that there are non-recursive trade-offs between the hierarchy levels of unambiguous and nondeterministic pushdown automata. In [101] the trade-offs between $(k+1)$ -turn and k -turn pushdown automata are investigated. The results are summarized as follows:

Theorem 1.92. *Let $k \geq 1$ be an integer. Then the following trade-offs are non-recursive:*

- (1) *between nondeterministic 1-turn pushdown automata and finite automata,*
- (2) *between nondeterministic $(k+1)$ -turn pushdown automata and nondeterministic k -turn pushdown automata,*
- (3) *between nondeterministic pushdown automata and nondeterministic finite-turn pushdown automata, and*
- (4) *between nondeterministic k -turn and deterministic k -turn pushdown automata.*

So, there are infinite hierarchies such that between each two levels there are non-recursive trade-offs. Other results of such flavor have been obtained in [87] where deterministic and nondeterministic one-way k -head finite automata (k -DFA, k -NFA) are considered.

Theorem 1.93. *Let $k \geq 2$ be an integer. The trade-off between k -DFA and nondeterministic pushdown automata is non-recursive.*

Theorem 1.94. *Let $k \geq 1$ be an integer. Then the following trade-offs are non-recursive:*

- (1) *between $(k + 1)$ -DFA and k -DFA,*
- (2) *between $(k + 1)$ -NFA and k -NFA, and*
- (3) *between $(k + 1)$ -DFA and k -NFA.*

Theorem 1.95. *Let $k \geq 2$ be an integer. Then the following trade-offs are non-recursive:*

- (1) *between 2-NFA and k -DFA, and*
- (2) *between k -DFA and nondeterministic pushdown automata.*

In [76] the problem whether there are non-recursive trade-offs between the levels of the hierarchies defined by two-way k -head finite automata (cf. also [86]) has been answered in the affirmative.

Theorem 1.96. *Let $k \geq 1$ be an integer. The trade-off between (non)deterministic (unary) two-way $(k + 1)$ -head finite automata and (non)deterministic (unary) two-way k -head finite automata is non-recursive.*

Now we briefly consider non-classical descriptive systems. In [65] *deterministic restarting automata*, an automaton model inspired from linguistics are investigated. Variants of deterministic and monotone restarting automata build a strict hierarchy whose top is characterized by the *Church-Rosser languages* and whose bottom is characterized by the deterministic context-free languages. It is shown that between PDAs and any level of the hierarchy there are non-recursive trade-offs. Interestingly, the converse is also true for the Church-Rosser languages. Moreover, there are non-recursive trade-offs between the family of Church-Rosser languages and any other level of the hierarchy.

Theorem 1.97. *The following trade-offs are non-recursive:*

- (1) *between nondeterministic (one-turn) pushdown automata and Church-Rosser languages (deterministic $R(R)WW$ -automata),*
- (2) *between Church-Rosser languages and nondeterministic pushdown automata,*

- (3) between Church-Rosser languages and deterministic pushdown automata,
- (4) between deterministic RWW-automata (Church-Rosser languages) and deterministic monotone RRWW-automata,
- (5) between deterministic RWW- and deterministic RRW-automata,
- (6) between deterministic RWW- and deterministic RW-automata,
- (7) between deterministic RWW- and deterministic RR-automata, and
- (8) between deterministic RWW- and deterministic R-automata.

Metilinear CD grammar systems [24] are context-free CD grammar systems where each component consists of metilinear productions. The maximal number of nonterminals in an initial production defines the width of the CD grammar system. In [131] it is proved that there are non-recursive trade-offs between CD grammar systems of width $m + 1$ and m . Furthermore, non-recursive trade-offs appear between CD grammar systems of width m and $(2m - 1)$ -linear context-free grammars.

Further results are known for *(one-way) cellular automata ((O)CA)* and *iterative arrays (IA)* [99, 100].

Theorem 1.98. *The following trade-offs are non-recursive:*

- (1) between real-time OCA and finite automata,
- (2) between real-time OCA and pushdown automata,
- (3) between real-time CA and real-time OCA, and
- (4) between linear-time OCA and real-time OCA.
- (5) Between real-time IA and finite automata,
- (6) between real-time IA and pushdown automata,
- (7) between linear-time IA and real-time IA,
- (8) between real-time IA and real-time OCA, and
- (9) between real-time OCA and real-time IA.

The proofs all follow the scheme of Theorem 1.80. It is worth mentioning that results (1) and (2) of Theorem 1.97 as well as results (8) and (9) of Theorem 1.98 say that there are non-recursive trade-offs between one system and another system and *vice versa*.

Finally, the phenomenon of non-recursive trade-offs between descriptive systems is investigated in an abstract and more axiomatic fashion in [49]. The aim is to categorize non-recursive trade-offs by bounds on their growth rate, and to show how to deduce such bounds in general. Also criteria are identified which, in the spirit of abstract language theory, allow

to deduce non-recursive tradeoffs from effective closure properties of language families on the one hand, and differences in the decidability status of basic decision problems on the other. A qualitative classification of non-recursive trade-offs is developed in order to obtain a better understanding of this very fundamental behavior of descriptive systems.

References

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms* (Addison-Wesley).
2. Baker, B. S. and Book, R. V. (1974). Reversal-bounded multipushdown machines, *J. Comput. System Sci.* **8**, pp. 315–332.
3. Berman, P. (1980). A note on sweeping automata, in *International Colloquium on Automata, Languages and Programming (ICALP 1980)*, LNCS, Vol. 85 (Springer), pp. 91–97.
4. Berman, P. and Lingas, A. (1977). On the complexity of regular languages in terms of finite automata, Tech. Rep. 304, Polish Academy of Sciences.
5. Bertsch, E. and Nederhof, M.-J. (2001). Size/lookahead tradeoff for LL(k)-grammars, *Inform. Process. Lett.* **80**, pp. 125–129.
6. Berwanger, D., Dawar, A., Hunter, P. and Kreutzer, S. (2006). Dag-width and parity games, in *Theoretical Aspects of Computer Science (STACS 2006)*, LNCS, Vol. 3884 (Springer), pp. 524–536.
7. Berwanger, D. and Grädel, E. (2005). Entanglement – a measure for the complexity of directed graphs with applications to logic and games, in *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2004)*, LNCS, Vol. 3452 (Springer), pp. 209–223.
8. Birget, J.-C. (1992a). Intersection and union of regular languages and state complexity, *Inform. Process. Lett.* **43**, pp. 185–190.
9. Birget, J.-C. (1992b). Positional simulation of two-way automata: Proof of a conjecture of R. Kannan and generalizations, *J. Comput. System Sci.* **45**, pp. 154–179.
10. Birget, J.-C. (1993). State-complexity of finite-state devices, state compressibility and incompressibility, *Math. Systems Theory* **26**, pp. 237–269.
11. Birget, J.-C. (1996). Two-way automata and length-preserving homomorphisms, *Math. Systems Theory* **29**, pp. 191–226.
12. Borchardt, I. (1992). *Nonrecursive Tradeoffs between context-free grammars with different constant ambiguity*, Diploma thesis, Universität Frankfurt (in German).
13. Bordihn, H., Holzer, M. and Kutrib, M. (2005). Unsolvability levels of operation problems for subclasses of context-free languages, *Int. J. Found. Comput. Sci.* **16**, pp. 423–440.
14. Bordihn, H., Holzer, M. and Kutrib, M. (2009). Determinization of finite automata accepting subregular languages, *Theoret. Comput. Sci.* **410**, pp. 3209–3222.

15. Brzozowski, J. A. and Leiss, E. L. (1980). On equations for regular languages, finite automata, and sequential networks, *Theoret. Comput. Sci.* **10**, pp. 19–35.
16. Bucher, W. (1981). A note on a problem in the theory of grammatical complexity, *Theoret. Comput. Sci.* **14**, pp. 337–344.
17. Bucher, W., Maurer, H. A. and Čulik II, K. (1984). Context-free complexity of finite languages, *Theoret. Comput. Sci.* **28**, pp. 277–285.
18. Bucher, W., Maurer, H. A., Čulik II, K. and Wotschke, D. (1981). Concise description of finite languages, *Theoret. Comput. Sci.* **14**, pp. 227–246.
19. Cerný, A. (1977). Complexity and minimality of context-free grammars and languages, in *Mathematical Foundations of Computer Science (MFCS 1977)*, LNCS, Vol. 53 (Springer), pp. 263–271.
20. Champarnaud, J.-M., Ouardi, F. and Ziadi, D. (2007). Normalized expressions and finite automata, *Int. J. Algebra Comput.* **17**, pp. 141–154.
21. Chandra, A., Kozen, D. and Stockmeyer, L. (1981). Alternation, *J. ACM* **21**, pp. 114–133.
22. Chrobak, M. (1986). Finite automata and unary languages, *Theoret. Comput. Sci.* **47**, pp. 149–158.
23. Chrobak, M. (2003). Errata to “finite automata and unary languages”, *Theoret. Comput. Sci.* **302**, pp. 497–498.
24. Csuhaj-Varjú, E., Dassow, J., Kelemen, J. and Păun, G. (1984). *Grammar Systems: A Grammatical Approach to Distribution and Cooperation* (Gordon and Breach).
25. Domaratzki, M., Pighizzini, G. and Shallit, J. (2002). Simulating finite automata with context-free grammars, *Inform. Process. Lett.* **84**, pp. 339–344.
26. Domaratzki, M. and Salomaa, K. (2006). Lower bounds for the transition complexity of NFAs, in *Mathematical Foundations of Computer Science (MFCS 2006)*, LNCS, Vol. 4162 (Springer), pp. 315–326.
27. Eggan, L. C. (1963). Transition graphs and the star height of regular events, *Michigan Math. J.* **10**, pp. 385–397.
28. Ehrenfeucht, A. and Zeiger, H. P. (1976). Complexity measures for regular expressions, *J. Comput. System Sci.* **12**, pp. 134–146.
29. Ellul, K., Krawetz, B., Shallit, J. and Wang, M.-W. (2005). Regular expressions: New results and open problems, *J. Autom., Lang. Comb.* **10**, pp. 407–437.
30. Fellah, A., Jürgensen, H. and Yu, S. (1990). Constructions for alternating finite automata, *Internat. J. Comput. Math.* **35**, pp. 117–132.
31. Geffert, V., Mereghetti, C. and Pighizzini, G. (2003). Converting two-way nondeterministic unary automata into simpler automata, *Theoret. Comput. Sci.* **295**, pp. 189–203.
32. Gelade, W. and Neven, F. (2008). Succinctness of the complement and intersection of regular expressions, in *Theoretical Aspects of Computer Science (STACS 2008)*, Dagstuhl Seminar Proceedings, Vol. 08001 (IBFI, Schloss Dagstuhl, Germany), pp. 325–336.
33. Gill, A. and Kou, L. T. (1974). Multiple-entry finite automata, *J. Comput. System Sci.* **9**, pp. 1–19.

34. Ginsburg, S. and Greibach, S. A. (1966). Deterministic context-free languages, *Inform. Control* **9**, pp. 620–648.
35. Ginsburg, S. and Rice, H. G. (1962). Two families of languages related to ALGOL, *J. ACM* **9**, pp. 350–371.
36. Glaister, I. and Shallit, J. (1996). A lower bound technique for the size of nondeterministic finite automata, *Inform. Process. Lett.* **59**, pp. 75–77.
37. Glushkov, V. M. (1961). The abstract theory of automata, *Russian Math. Surveys* **16**, pp. 1–53.
38. Goldstine, J., Kappes, M., Kintala, C. M. R., Leung, H., Malcher, A. and Wotschke, D. (2002). Descriptive complexity of machines with limited resources, *J. UCS* **8**, pp. 193–234.
39. Goldstine, J., Leung, H. and Wotschke, D. (1992). On the relation between ambiguity and nondeterminism in finite automata, *Inform. Comput.* **100**, pp. 261–270.
40. Goldstine, J., Price, J. K. and Wotschke, D. (1982a). On reducing the number of states in a PDA, *Math. Systems Theory* **15**, pp. 315–321.
41. Goldstine, J., Price, J. K. and Wotschke, D. (1982b). A pushdown automaton or a context-free grammar – which is more economical? *Theoret. Comput. Sci.* **18**, pp. 33–40.
42. Goldstine, J., Price, J. K. and Wotschke, D. (1993). On reducing the number of stack symbols in a PDA, *Math. Systems Theory* **26**, pp. 313–326.
43. Greibach, S. A. (1965). A new normal-form theorem for context-free phrase structure grammars, *J. ACM* **12**, pp. 42–52.
44. Greibach, S. A. (1973). The hardest context-free language, *SIAM J. Comput.* **2**, pp. 304–310.
45. Gruber, H. and Gulan, S. (2009). Simplifying regular expressions: A quantitative perspective, IFIG Research Report 0904, Institut für Informatik, Justus-Liebig-Universität, Gießen, Germany.
46. Gruber, H. and Holzer, M. (2005). A note on the number of transitions of nondeterministic finite automata, in *Theoretag Automaten und Formale Sprachen* (Universität Tübingen, Tübingen, Germany), pp. 24–25.
47. Gruber, H. and Holzer, M. (2006). Finding lower bounds for nondeterministic state complexity is hard, in *Developments in Language Theory (DLT 2006)*, LNCS, Vol. 4036 (Springer), pp. 363–374.
48. Gruber, H. and Holzer, M. (2008). Provably shorter regular expressions from deterministic finite automata, in *Developments in Language Theory (DLT 2008)*, LNCS, Vol. 5257 (Springer), pp. 383–395.
49. Gruber, H., Holzer, M. and Kutrib, M. (2009). On measuring non-recursive trade-offs, in *Descriptive Complexity of Formal Systems (DCFS 2009)* (Otto-von-Guericke-Universität Magdeburg, Germany), pp. 187–198.
50. Gruber, H. and Johannsen, J. (2008). Optimal lower bounds on regular expression size using communication complexity, in *Foundations of Software Science and Computational Structures (FoSSaCS 2008)*, LNCS, Vol. 4962 (Springer), pp. 273–286.
51. Gruska, J. (1967). On a classification of context-free languages, *Kybernetika* **3**, pp. 22–29.

52. Gruska, J. (1969). Some classifications of context-free languages, *Inform. Control* **14**, pp. 152–179.
53. Gruska, J. (1971a). A characterization of context-free languages, *J. Comput. System Sci.* **5**, pp. 353–364.
54. Gruska, J. (1971b). Complexity and unambiguity of context-free grammars and languages, *Inform. Control* **18**, pp. 502–519.
55. Gruska, J. (1976). Descriptive complexity (of languages) - a short survey, in *Mathematical Foundations of Computer Science (MFCS 1976)*, LNCS, Vol. 45 (Springer), pp. 65–80.
56. Gulan, S. and Fernau, H. (2008). An optimal construction of finite automata from regular expressions, in *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, Dagstuhl Seminar Proceedings, Vol. 08002 (IBFI, Schloss Dagstuhl, Germany), pp. 211–222.
57. Harrison, M. A. (1978). *Introduction to Formal Language Theory* (Addison-Wesley).
58. Hartmanis, J. (1967). Context-free languages and Turing machine computations, *Proc. Symposia in Applied Mathematics* **19**, pp. 42–51.
59. Hartmanis, J. (1980). On the succinctness of different representations of languages, *SIAM J. Comput.* **9**, pp. 114–120.
60. Hartmanis, J. (1983). On Gödel speed-up and succinctness of language representations, *Theoret. Comput. Sci.* **26**, pp. 335–342.
61. Hartmanis, J. and Baker, T. P. (1979a). Relative succinctness of representations of languages and separation of complexity classes, in *Mathematical Foundations of Computer Science (MFCS 1979)*, LNCS, Vol. 74 (Springer), pp. 70–88.
62. Hartmanis, J. and Baker, T. P. (1979b). Succinctness, verifiability and determinism in representations of polynomial-time languages, in *Foundations and Computer Science (FOCS 1979)* (IEEE), pp. 392–396.
63. Hashiguchi, K. (1988). Algorithms for determining relative star height and star height, *Inform. Comput.* **78**, pp. 124–169.
64. Herzog, C. (1997). Pushdown automata with bounded nondeterminism and bounded ambiguity, *Theoret. Comput. Sci.* **181**, pp. 141–157.
65. Holzer, M., Kutrib, M. and Reimann, J. (2007). Non-recursive trade-offs for deterministic restarting automata, *J. Autom., Lang. Comb.* **12**, pp. 195–213.
66. Holzer, M., Salomaa, K. and Yu, S. (2001). On the state complexity of k-entry deterministic finite automata, *J. Autom., Lang. Comb.* **6**, pp. 453–466.
67. Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley).
68. Hromkovič, J. (1997). *Communication Complexity and Parallel Computing* (Springer).
69. Hromkovič, J. and Schnitger, G. (2005). NFAs with and without ϵ -transitions, in *International Colloquium on Automata, Languages and Programming (ICALP 2005)*, LNCS, Vol. 3580 (Springer), pp. 385–396.
70. Hromkovič, J., Seibert, S. and Wilke, T. (2001). Translating regular expres-

- sions into small ϵ -free nondeterministic finite automata, *J. Comput. System Sci.* **62**, pp. 565–588.
71. Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H. and Schnitger, G. (2002). Communication complexity method for measuring nondeterminism in finite automata, *Inform. Comput.* **172**, pp. 202–217.
 72. Ilie, L. and Yu, S. (2003). Follow automata, *Inform. Comput.* **186**, pp. 140–162.
 73. Iwama, K., Kambayashi, Y. and Takaki, K. (2000). Tight bounds on the number of states of DFAs that are equivalent to n -state NFAs, *Theoret. Comput. Sci.* **237**, pp. 485–494.
 74. Johnson, T., Robertson, N., Seymour, P. D. and Thomas, R. (2001). Directed tree-width, *J. Combinatorial Theory* **82**, pp. 138–154.
 75. Kannan, R. (1983). Alternation and the power of nondeterminism, in *Symposium on Theory of Computing (STOC 1983)* (ACM Press), pp. 344–346.
 76. Kapoutsis, C. A. (2004). From $k + 1$ to k heads the descriptive trade-off is non-recursive, in *Descriptive Complexity of Formal Systems (DCFS 2004)*, pp. 213–224.
 77. Kapoutsis, C. A. (2005). Removing bidirectionality from nondeterministic finite automata, in *Mathematical Foundations of Computer Science (MFCS 2005)*, LNCS, Vol. 3618 (Springer), pp. 544–555.
 78. Kapoutsis, C. A. (2006). *Algorithms and Lower Bounds in Finite Automata Size Complexity*, Phd thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
 79. Kappes, M. (2000). Descriptive complexity of deterministic finite automata with multiple initial states, *J. Autom., Lang. Comb.* **5**, pp. 269–278.
 80. Kelemenová, A. (1981). Grammatical levels of the position restricted grammars, in *Mathematical Foundations of Computer Science (MFCS 1981)*, LNCS, Vol. 118 (Springer), pp. 347–359.
 81. Kelemenová, A. (1984). Complexity of normal form grammars, *Theoret. Comput. Sci.* **28**, pp. 299–314.
 82. Kintala, C. M. and Wotschke, D. (1980). Amounts of nondeterminism in finite automata, *Acta Inform.* **13**, pp. 199–204.
 83. Kirsten, D. (2005). Distance desert automata and the star height problem, *RAIRO Inform. Théor.* **39**, pp. 455–509.
 84. Klauck, H. (1998). Lower bounds for computation with limited nondeterminism, in *Conference on Computational Complexity (IEEE)*, pp. 141–152.
 85. Kleene, S. C. (1956). Representation of events in nerve nets and finite automata, in *Automata Studies* (Princeton University Press), pp. 3–42.
 86. Kutrib, M. (2005a). On the descriptive power of heads, counters, and pebbles, *Theoret. Comput. Sci.* **330**, pp. 311–324.
 87. Kutrib, M. (2005b). The phenomenon of non-recursive trade-offs, *Int. J. Found. Comput. Sci.* **16**, pp. 957–973.
 88. Kutrib, M. and Reimann, J. (2008). Succinct description of regular languages by weak restarting automata, *Inform. Comput.* **206**, pp. 1152–1160.

89. Landau, E. (1903). Über die Maximalordnung der Permutationen gegebenen Grades, *Archiv der Math. und Phys.* **3**, pp. 92–103.
90. Landau, E. (1909). *Handbuch der Lehre von der Verteilung der Primzahlen* (Teubner).
91. Leiss, E. L. (1980). Constructing a finite automaton for a given regular expression, *Bull. EATCS* **10**, pp. 54–59.
92. Leiss, E. L. (1981). Succinct representation of regular languages by Boolean automata, *Theoret. Comput. Sci.* **13**, pp. 323–330.
93. Leiss, E. L. (1985). Succinct representation of regular languages by Boolean automata. II, *Theoret. Comput. Sci.* **38**, pp. 133–136.
94. Leung, H. (1998). Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata, *SIAM J. Comput.* **27**, pp. 1073–1082.
95. Leung, H. (2001). Tight lower bounds on the size of sweeping automata, *J. Comput. System Sci.* **63**, pp. 384–393.
96. Leung, H. and Wotschke, D. (2000). On the size of parsers and LR(k)-grammars, *Theoret. Comput. Sci.* **242**, pp. 59–69.
97. Lifshits, Y. (2003). A lower bound on the size of ϵ -free NFA corresponding to a regular expression, *Inform. Process. Lett.* **85**, pp. 293–299.
98. Lupanov, O. B. (1963). A comparison of two types of finite sources (in Russian), *Problemy Kybernetiki* **9**, pp. 321–326, German translation (1966): Über den Vergleich zweier Typen endlicher Quellen, *Probleme der Kybernetik* **6**, pp. 328–335.
99. Malcher, A. (2002). Descriptive complexity of cellular automata and decidability questions, *J. Autom., Lang. Comb.* **7**, pp. 549–560.
100. Malcher, A. (2004). On the descriptive complexity of iterative arrays, *IEICE Trans. Inf. Syst.* **E87-D**, pp. 721–725.
101. Malcher, A. (2007). On recursive and non-recursive trade-offs between finite-turn pushdown automata, *J. Autom., Lang. Comb.* **12**, pp. 265–277.
102. Malcher, A. and Pighizzini, G. (2007). Descriptive complexity of bounded context-free languages, in *Developments in Language Theory (DLT 2007)*, *LNCS*, Vol. 4588 (Springer), pp. 312–323.
103. Mandl, R. (1973). Precise bounds associated with the subset construction on various classes of nondeterministic finite automata, in *Princeton Conference on Information Sciences and Systems (CISS 1973)*, pp. 263–267.
104. McNaughton, R. (1969). The loop complexity of regular events, *Inform. Sci.* **1**, pp. 305–328.
105. McNaughton, R. and Yamada, H. (1960). Regular expressions and state graphs for automata, *IRE Trans. Elect. Comput.* **EC-9**, pp. 39–47.
106. McWhirter, I. P. (1971). Substitution expressions, *J. Comput. System Sci.* **5**, pp. 629–637.
107. Mereghetti, C. and Pighizzini, G. (2000). Two-way automata simulations and unary languages, *J. Autom., Lang. Comb.* **5**, pp. 287–300.
108. Mereghetti, C. and Pighizzini, G. (2001). Optimal simulations between

- unary automata, *SIAM J. Comput.* **30**, pp. 1976–1992.
109. Meyer, A. R. and Fischer, M. J. (1971). Economy of description by automata, grammars, and formal systems, in *Switching and Automata Theory (SWAT 1971)* (IEEE), pp. 188–191.
 110. Micali, S. (1981). Two-way deterministic finite automata are exponentially more succinct than sweeping automata, *Inform. Process. Lett.* **12**, pp. 103–105.
 111. Minsky, M. L. (1967). *Computation: Finite and Infinite Machines* (Prentice-Hall).
 112. Moore, F. R. (1971). On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata, *IEEE Trans. Comput.* **20**, pp. 1211–1214.
 113. Ogden, W. F. (1968). A helpful result for proving inherent ambiguity, *Math. Systems Theory* **2**, pp. 191–194.
 114. Păun, G. (1984). *Probleme actuale în teoria limbajelor formale* (Editura științifică enciclopedică).
 115. Pighizzini, G. (2008). Deterministic pushdown automata and unary languages, in *Implementation and Application of Automata (CIAA 2008)*, LNCS, Vol. 5148 (Springer), pp. 232–241.
 116. Pighizzini, G., Shallit, J. and Wang, M.-W. (2002). Unary context-free grammars and pushdown automata, descriptive complexity and auxiliary space lower bounds, *J. Comput. System Sci.* **65**, pp. 393–414.
 117. Pirická, A. (1974). Complexity and normal forms of context-free languages, in *Mathematical Foundations of Computer Science (MFCS 1974)*, LNCS, Vol. 28 (Springer), pp. 292–297.
 118. Pirická-Kelemenová, A. (1975). Greibach normal form complexity, in *Mathematical Foundations of Computer Science (MFCS 1975)*, LNCS, Vol. 32 (Springer), pp. 344–350.
 119. Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems, *IBM J. Res. Dev.* **3**, pp. 114–125.
 120. Ravikumar, B. and Ibarra, O. H. (1989). Relating the type of ambiguity of finite automata to the succinctness of their representation, *SIAM J. Comput.* **18**, pp. 1263–1282.
 121. Sakarovitch, J. (2006). The language, the expression, and the (small) automaton, in *Implementation and Application of Automata (CIAA 2005)*, LNCS, Vol. 3845 (Springer), pp. 15–30.
 122. Sakoda, W. J. and Sipser, M. (1978). Nondeterminism and the size of two way finite automata, in *Symposium on Theory of Computing (STOC 1978)* (ACM Press), pp. 275–286.
 123. Salomaa, A. (1973). *Formal Languages* (Academic Press).
 124. Salomaa, K. and Yu, S. (1997). NFA to DFA transformation for finite languages over arbitrary alphabets, *J. Autom., Lang. Comb.* **2**, pp. 177–186.
 125. Schmidt, E. M. (1978). *Succinctness of Descriptions of Context-Free, Regular and Finite Languages*, Ph.D. thesis, Cornell University, Ithaca, New York.
 126. Schmidt, E. M. and Szymanski, T. G. (1977). Succinctness of descriptions

- of unambiguous context-free languages, *SIAM J. Comput.* **6**, pp. 547–553.
127. Shepherdson, J. C. (1959). The reduction of two-way automata to one-way automata, *IBM J. Res. Dev.* **3**, pp. 198–200.
 128. Sipser, M. (1980). Lower bounds on the size of sweeping automata, *J. Comput. System Sci.* **21**, pp. 195–202.
 129. Stearns, R. E. (1967). A regularity test for pushdown machines, *Inform. Control* **11**, pp. 323–340.
 130. Stearns, R. E. and Hunt III, H. B. (1985). On the equivalence and containment problems for unambiguous regular expressions, regular grammars, and finite automata, *SIAM J. Comput.* **14**, pp. 598–611.
 131. Sunckel, B. (2005). On the descriptonal complexity of metalinear CD grammar systems, *Int. J. Found. Comput. Sci.* **16**, pp. 1011–1025.
 132. Valiant, L. G. (1975). Regularity and related problems for deterministic pushdown automata, *J. ACM* **22**, pp. 1–10.
 133. Valiant, L. G. (1976). A note on the succinctness of descriptions of deterministic languages, *Inform. Control* **32**, pp. 139–145.
 134. Veloso, P. A. S. and Gill, A. (1979). Some remarks on multiple-entry finite automata, *J. Comput. System Sci.* **18**, pp. 304–306.
 135. Yntema, M. K. (1971). Cap expressions for context-free languages, *Inform. Control* **18**, pp. 311–318.
 136. Yu, S. (1997). Regular languages, in *Handbook of Formal Languages*, Vol. 1, chap. 2 (Springer), pp. 41–110.
 137. Yu, S. (2001). State complexity of regular languages, *J. Autom., Lang. Comb.* **6**, pp. 221–234.