Journal of Universal Computer Science, vol. 8, no. 2 (2002), 193-234
submitted: 15/9/01, accepted: 31/1/02, appeared: 28/2/02 © J.UCS

Descriptional Complexity of Machines with Limited Resources [1]

Jonathan Goldstine
(Department of Computer Science, Pennsylvania State University
Email: goldstin@cse.psu.edu)

Martin Kappes
(Avaya Labs, Basking Ridge, NJ
Email: mkappes@avaya.com)

Chandra M.R. Kintala
(Avaya Labs, Basking Ridge, NJ
Email: cmk@avaya.com)

Hing Leung
(Department of Computer Science, New Mexico State University
Email: hleung@nmsu.edu)

Andreas Malcher
(Department of Computer Science, University of Frankfurt
Email: malcher@psc.informatik.uni-frankfurt.de)

Detlef Wotschke[2,3]
(Department of Computer Science, University of Frankfurt
Email: wotschke@psc.informatik.uni-frankfurt.de)

Abstract: Over the last 30 years or so many results have appeared on the descriptional complexity of machines with limited resources. Since these results have appeared in a variety of different contexts, our goal here is to provide a survey of these results. Particular emphasis is put on limiting resources (e.g., nondeterminism, ambiguity, lookahead, etc.) for various types of finite state machines, pushdown automata, parsers and cellular automata and on the effect it has on their descriptional complexity. We also address the question of how descriptional complexity might help in the future to solve practical issues, such as software reliability.

Key Words: Descriptional complexity, nondeterminism, ambiguity, formal languages, finite automata, pushdown automata, parsers, cellular automata, software reliability
Category: F.1, F.4

1 Introduction

The great Greek philosopher Socrates is often cited as having said in his famous defense speech (short version): I know that I know nothing! As true as this might

[1] C. S. Calude, K. Salomaa, S. Yu (eds.). *Advances and Trends in Automata and Formal Languages. A Collection of Papers in Honour of the 60th Birthday of Helmut Jürgensen.*
[2] Part of this author's work was done while he was visiting the Network Software Research Department in Avaya Labs, Basking Ridge, NJ.
[3] Corresponding author

be—after all, what do we really know or understand?—it is just as impractical, especially in the current knowledge-based world and environment. Just imagine the executive of any technology company giving a presentation at a major convention and, when asked a question, replying: I don't know anything! Virtually unthinkable!

The executive would like the presentation to be as simple or as easy as possible and only as difficult as necessary. He does not want to be in the position where a potential customer might say: If you can't explain it so that I understand it, then I am not interested and therefore I don't want to hear about it. By the same token, the executive would not want his customers to say: This sounds so simple that there really cannot be anything to it, so we are not interested. Don't waste our time with trivia.

Creating such presentations and descriptions is not easy, especially for complex engineering systems. A modern jet aircraft has hydraulic parts, electronic monitors and software control systems, all interconnected in such a complex fashion that it takes literally thousands of pages of detailed descriptions to show that power to the entertainment system in the jet will automatically be turned off in case of a low fuel level reading in one of the engines.

Similarly, current voice and data communication devices have tens of millions of lines of code and hundreds of interfaces and features to process the signals, activate the features and produce detailed records for billing purposes. It would be a complex task to describe in detail how, for example, hanging up an IP phone will terminate the session, produce the correct records and free communication resources for later use.

This brings us to the area of descriptional complexity (often called conciseness, succinctness, or economy of description), that is, the science—or sometimes the art—of making the description of objects as simple as possible and only as complex as necessary. Modifying Socrates' statement, we would probably say: I don't know what I really know. But I do know that, whatever I know, I only know something well if I can describe it in simple terms.

But, as in the above example of a presentation by a company's executive, determining "as simple as possible" or "only as difficult as necessary" is in general not that easy. In preparing the presentation for the convention it is important to know how much the audience already knows, i.e., how many knowledge resources or tools can be employed, which presentation resources can be used, etc. How simple, how long or how ambiguous can the presentation be in some instances for the sake of brevity or simplicity? How precise does it have to be? These are some of the questions one needs to answer before the presentation can be prepared.

Depending on the particular audience, sometimes the resources for the presentation can be used to their full capacity, sometimes not at all, and in most cases they have to be used in a limited way. This brings us to the area of *Descriptional Complexity and Limited Resources.*

Even in a more scientific or mathematical context, descriptional complexity is a real everyday issue. Often, proofs of the same mathematical theorem differ greatly in length and complexity, and, just as often, this difference is a direct consequence of the mathematical theory employed or of other mathematical results used.

Since descriptional complexity has become a large and very wide-spread area, we would like to maintain a well-defined focus and thus, coming from a computer science and engineering background, will apply the question of simplicity vis-àvis resource use to *Machines*, and this brings us to *Descriptional Complexity of Machines with Limited Resources*. And to define the focus even more closely, we will concentrate on scalable or gradual *trade-offs* with other measures or resources which result from such limitations. In other words, we are interested, at least whenever possible, in questions like: How much does the limitation of one resource cost in terms of another resource, i.e., what are the upper and lower bounds of such costs?

Therefore, we will direct our attention to such well-known and classical devices as finite automata, pushdown automata, syntax parsers, cellular automata, etc. Such formal devices are often used to describe certain aspects of the complex engineering systems we mentioned earlier. For example, finite automata are routinely used to describe the call processing routines in switching equipment, and message sequence charts are used for protocols in data communication equipment. Pushdown automata are, among other things, used to model recursive programs and nested procedures, and parsers are an important part in translating software code to execution code. Cellular automata are realized as hardware implementations of a massively parallel model and they serve to model and analyze phenomena of nature.

Of course, there are other formal devices which have been looked at from the vantage point of descriptional complexity and limited resources, such as Regular Expressions, Alternating and Probabilistic Finite Automata, Quantum Automata, Formal Grammars, Regulated Rewriting Systems, Contextual Grammars, Grammar Systems, Lindenmayer Systems, Rewriting Systems for DNA Computing, Membrane Computing, Logical Formulas, Boolean Circuits, etc.

As much as this paper is meant to be a survey of existing results, just as much, due to the desire to provide a well-defined focus, it can by no means be a complete survey. Hence the restriction to Machines and the selection of topics is somewhat subjective. For these reasons we apologize to authors (and their areas) that do not feel adequately represented herein.

Since, to our knowledge, this is the first survey of its kind, we hope that it might serve as an incentive for further surveys covering the other areas which could not be included here. As a preliminary step in this direction we are currently constructing a database with titles of papers on descriptional complexity which will be continuously updated. A link to this database can be found at

`www.psc.cs.uni-frankfurt.de/english`

Since the intricacies of many results surveyed here and of their proofs can be fully understood only by reading the complete proofs in the original papers, we purposely do not attempt to give a fully unified presentation here. Instead, we would like this survey to serve as a directory or tour guide to the original literature. As a consequence, we will in the various sections of this paper partly try to preserve the flavor, thrust and notation of the original papers, and therefore the various sections do in part differ in style, notation and in the amount of detail provided.

However, some concepts are common to all questions and results exhibited here and should thus be unified. We therefore will briefly define the concepts of descriptional complexity and of upper and lower bounds of descriptional trade-offs.

Definition 1 *A* descriptional system $\mathcal{D}$ *for a class* $\mathcal{S}$ *of languages is a set of finite descriptors, each of them expressing a language in* $\mathcal{S}$*. For every* $L \in \mathcal{S}$ *let* $\mathcal{D}(L){:=}\{D \mid D \in \mathcal{D} \ \wedge D \text{ describes } L\}$.

Definition 2 *A* complexity measure C *for a descriptional system* $\mathcal{D}$ *is a total function* $C : \mathcal{D} \to \mathbb{R}_0^+$*;* $C(D)$ *for a* $D \in \mathcal{D}$ *is the* complexity *of the descriptor* D.

Let $\mathbb{R}_0^+$ be the set of non-negative real numbers, let $\mathcal{S}$ be an infinite class of languages, $\mathcal{D}_1, \mathcal{D}_2$ two descriptional systems with descriptors for languages in $\mathcal{S}$, and $C_1 : \mathcal{D}_1 \to \mathbb{R}_0^+$, $C_2 : \mathcal{D}_2 \to \mathbb{R}_0^+$ two complexity measures.

So, we can ask the following natural questions:

1. Does a function $F : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, $F \geq id$ (where id is the identity function) exist, such that for all $L \in \mathcal{S}$

$$\min\{C_1(D) \mid D \in \mathcal{D}_1(L)\} \quad \leq \quad F(\min\{C_2(D) \mid D \in \mathcal{D}_2(L)\})?$$

In case that F exists, it is an upper bound for the increase (blow-up) in complexity when changing from a minimal description in $\mathcal{D}_2$ for an *arbitrary* language $L \in \mathcal{S}$ to an equivalent minimal description in $\mathcal{D}_1$ (or for the savings in complexity in the reverse direction, respectively).

Notation: $\begin{array}{ccc} \mathcal{D}_2 & \overset{C_2,C_1}{\longrightarrow} & \mathcal{D}_1 \\ n & & \leq F(n) \end{array}$

If $C_1 = C_2$, we will drop C_2 in this notation.

Remark: As will be seen later, F does not necessarily exist!

Examples: Let $\mathcal{D}_1$ = DFA (Deterministic Finite Automata), $\mathcal{D}_2$ = NFA (Nondeterministic Finite Automata) and s be the number of states. Then the following holds:

$$\begin{array}{ccl} NFA & \overset{s}{\longrightarrow} & DFA \\ n & & \leq 2^n \ \text{(subset-construction according to [RS59])} \end{array}$$

$$\begin{array}{ccl} DFA & \overset{s}{\longrightarrow} & NFA \\ n & & \leq n \ \text{(every DFA can be considered to be an NFA)} \end{array}$$

2. Does an infinite sequence $(L_i)_{i=0}^{\infty}$ of distinct languages exist with $L_i \in \mathcal{S}$ for all $i \in \mathbb{N}$ and a function $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, $f \geq id$, such that for all $i \in \mathbb{N}$

$$\min\{C_1(D) \mid D \in \mathcal{D}_1(L_i)\} \quad \geq \quad f(\min\{C_2(D) \mid D \in \mathcal{D}_2(L_i)\})?$$

In case that f exists, it is a lower bound for the increase in complexity ("blow-up") when changing from a description in $\mathcal{D}_2$ to an equivalent one in $\mathcal{D}_1$ (or for savings in the complexity in the reverse direction) for infinitely many languages.

Notation: $\begin{array}{ccc} \mathcal{D}_2 & \overset{C_2,C_1}{\longrightarrow} & \mathcal{D}_1 \\ n & & \geq f(n) \end{array}$

If $C_1 = C_2$, we will drop C_2 in this notation.

Examples:

$$\begin{array}{lll} NFA \xrightarrow{s} DFA & & \\ \quad n \qquad\quad \geq 2^n & \text{(will be shown in Section 2)} \end{array}$$

$$\begin{array}{lll} DFA \xrightarrow{s} NFA & & \\ \quad n \qquad\quad \geq n & \text{(e. g. with } L_n{:=}\{1^n\}) \end{array}$$

Definition 3 *If there is no recursive function serving as an upper bound for the trade-off between two descriptional systems $\mathcal{D}_1$ and $\mathcal{D}_2$, we say that the trade-off is* non-recursive *and write $\mathcal{D}_1 \xrightarrow{nonrec} \mathcal{D}_2$.*

In contrast to Definitions 1 and 2, the last definition does not mention a complexity measure. This is no loss of generality: A non-recursive trade-off means that changing from one descriptional system $\mathcal{D}_1$ to another system $\mathcal{D}_2$ implies a blow-up which is not bounded by any recursive function. Since any *reasonable* complexity measure can be assumed to be bounded by a recursive function, a non-recursive blow-up will exceed any difference caused by applying two reasonable complexity measures. Therefore, non-recursive trade-offs do not depend on specific (reasonable) complexity measures.

In Section 2 we will investigate how the descriptional complexity of finite automata varies with different amounts of nondeterminism. Specifically we will look at the concept of the spectrum of a regular language. The spectrum of a language indicates how "inherently nondeterministic" a language is. As a special case of finite automata with limited nondeterminism we will look at finite automata with up to k initial states.

Section 3 will focus on the descriptional complexity of finite automata with limited ambiguity. In other words, we do not necessarily restrict the total number of choices an automaton can make to process the input but do limit the number of "successful" choices.

Section 4 will look at two-way finite automata, i.e., finite automata which are allowed to move their reading head in two directions. Aside from the classical trade-offs obtained when converting two-way deterministic or nondeterministic finite automata to one-way deterministic ones, we will investigate two-way finite automata with limited resources, e.g., with a limited number of turns of the reading head as well as some restricted models as for instance *sweeping automata* which are allowed to change the head's direction only at either end of the input.

Section 5 will turn our attention to limiting the amount of nondeterminism in Pushdown Automata (PDAs). In order to limit nondeterminism, one first has to find or agree on ways of measuring it. Several ways of measuring nondeterminism will be discussed and examined, among them the so-called *minmax* measure.

Section 6 will study the descriptional complexity of PDAs with limited nondeterminism (based on the *minmax* measure) and limited ambiguity. Rather surprisingly, it turns out that for certain context-free languages, increasing the degree of ambiguity of the PDA (or corresponding grammar) by just one degree can yield a non-recursive reduction in descriptional complexity.

Section 7 will examine how an increase in the length of the lookahead for LL(k) and LR(k) languages can reduce the size of the parser. This is especially interesting in the LR(k) case. Although it is well known that LR(1) grammars already characterize the entire class of LR(k) languages, this result shows that it might be very prudent to enlarge the lookahead window for complexity reasons, even though it is not needed.

Section 8 will investigate the descriptional complexity of a parallel computational model. The main result is that there are non-recursive trade-offs between cellular automata and sequential automata as well as among several cellular models. Moreover, general cellular automata turn out to be very difficult to handle, because it can be shown that almost nothing is decidable. Putting a natural restriction on the general model yields a cellular model which is weaker in its generative capacity, but stronger in terms of manageability.

In Section 9 we will outline the relationship between descriptional complexity and software reliability, an area of tremendous practical interest. Furthermore, we will discuss how future work in descriptional complexity might help to better understand and improve software reliability.

Section 10 will look at some other types of devices and resources through the prism of descriptional complexity. For some of these types of devices the descriptional complexity has been studied in general but not—at least not extensively yet—from the particular aspect of gradually limiting resources.

In Section 11 we will give a short summary and a brief outlook for further research.

Our survey of results in descriptional complexity of machines with limited resources does not yet address the problems, for example, of proving certain properties of jet aircraft or other technological devices based on their descriptions or the complexity of their descriptions. However, we have surveyed the fundamental results and theoretical underpinnings in an area that one day could provide, hopefully, solutions to these problems.

2 Finite Automata with Limited Nondeterminism

2.1 Motivation

Finite automata (often also called finite state machines) are probably the most elementary automata model and are used in virtually every area of computer science, from process modeling in software engineering to protocol specification in distributed systems.

Nondeterminism is a fundamental concept in automata theory, in some sense formalizing the human behavior of guessing. For some automata models, the use of nondeterminism allows for an increased generative power or higher efficiency such as faster processing time or less (dynamic) space consumption. For finite automata, deterministic and nondeterministic automata both accept the class of regular languages and are thus equal in generative power. Furthermore, both operate in real time and, as there is no storage medium in addition to the finite state control, the dynamic complexity of deterministic and nondeterministic finite automata is also equal. Yet, the descriptional complexity, i.e., the size, of minimal deterministic and nondeterministic finite automata describing the same language may vary dramatically. In fact, the use of nondeterminism in finite automata sometimes allows for exponentially more concise representations of languages.

Not all nondeterministic finite automata appear to make equal use of nondeterminism. As first shown for other automata models in [KF80], nondeterminism in finite automata is a resource that may be used in different amounts and that can be accurately quantified. In this section, we will summarize results about the descriptional complexity of finite automata with limited nondeterminism. We will not only focus on the case in which all regular languages are considered but will also state results about trade-offs between NFAs with limited nondeterminism and DFAs when the languages accepted by these automata are limited to unary or binary alphabets, as well as when these languages are finite.

In order to precisely illustrate, describe and summarize the models and concepts used and the results obtained, we would like to recall some definitions. A nondeterministic finite automaton is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q and Σ are non-empty finite sets of states and input symbols, respectively, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a non-empty set of final states and $\delta : Q \times \Sigma \to 2^Q$ denotes the transition function. An automaton is called a deterministic finite automaton (DFA) if $|\delta(p, \sigma)| = 1$ for all $p \in Q, \sigma \in \Sigma$. A move μ is a triple $\mu = (p, \sigma, q) \in (Q \times \Sigma \times Q)$, where $q \in \delta(p, \sigma)$. This move is called nondeterministic if $|\delta(p, \sigma)| > 1$. A computation for $w = \sigma_1 \dots \sigma_n \in \Sigma^*$ is a sequence of moves $\mu_1 \dots \mu_n$, where $\mu_i = (p_{i-1}, \sigma_i, p_i)$ and p_0 is the initial state of M. It is called accepting if $p_n \in F$. The language accepted by an FA M is $T(M) = \{w \in \Sigma^* \mid \text{ there is an accepting computation for } w \text{ in } M\}$. Two automata M and M' are equivalent if $T(M) = T(M')$. As usual, the size of an automaton is measured by the size of its finite state control, the number of states of the automaton.

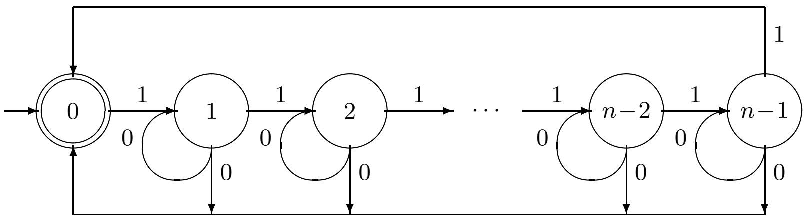

Figure 1: NFA M_n with n states accepting a language for which an equivalent DFA needs at least 2^n states.

The landmark paper [MF71] raised descriptional complexity as a research topic and addressed the economy of description by various devices. Among the results, it proved that there is an infinite sequence of languages $(L_n)_{n\geq 1}$ such that there is an NFA for L_n with n states and each equivalent DFA has at least 2^n states. In other words, it shows that in general no improvements to the subset construction are possible which converts an n-state NFA to an equivalent DFA with 2^n states as presented in [RS59]. The NFA M_n for L_n is depicted in Figure 1. It should be noted that the same result was independently proven by [Moo71] using another sequence of languages, and meanwhile some more sequences $(L_n)_{n\geq 1}$ are known such that the representation of L_n by a DFA requires at least 2^n states whereas there is an NFA for L_n with n states. Some of these sequences also possess other features, for instance a very few number of transitions [Leu98b]. Using the notation from the introduction, these results can be summarized as follows:

$$\begin{array}{ll} NFA \xrightarrow{s} DFA & \\ \quad n \qquad\quad \leq 2^n & \text{[RS59]} \end{array}$$

$$\begin{array}{ll} NFA \xrightarrow{s} DFA & \\ \quad n \qquad\quad \geq 2^n & \text{[MF71], [Moo71], [Leu98b]} \end{array}$$

In order to study situations where the resource of nondeterminism is limited, we need to measure nondeterminism. A natural candidate for quantifying the amount of nondeterminism an automaton uses in a computation $\mu_1 \ldots \mu_n$ is the number of nondeterministic moves in that computation. Crediting the automaton with its best effort, the amount of nondeterminism the automaton uses on an accepted input w can be measured by the minimal amount of nondeterminism needed in an accepting computation for w (this measure was suggested in [KW80]). The nondeterminism used by an NFA M is considered limited (also called finite) if there is an integer c such that for each accepted word there is an accepting computation with at most c nondeterministic moves. In this case, c is measuring the amount of nondeterminism M uses.

Returning to the NFAs proposed by [MF71], it can be seen that nondeterminism in the NFA M_n for all $n \geq 2$ is unlimited. For instance, when accepting the input 10^m, $m \geq 1$ at least m nondeterministic moves are made in any accepting computation.

So, the question arises whether the use of limited amounts of nondeterminism also enables NFAs to be more concise than DFAs, even if the amount of nondeterminism is very limited. As will be outlined in the remainder of this section, results in [KW80] and [GKW90] prove that there are cases where even very limited nondeterminism helps, but the extent to which limited nondeterminism may help is bounded. Furthermore, as results in [GKW90] show, there are languages such that NFAs, unlimited in nondeterminism, may be exponentially more concise compared to DFAs, but any NFA using a limited amount of nondeterminism has the same size as a DFA for the language.

2.2 Towards Finite Nondeterminism

An early indication that NFAs with finite nondeterminism can be more concise than DFAs, even if nondeterminism is limited and the described languages are finite, was given in [Man73]. This paper proved that in the case of a binary alphabet, each NFA with n states that accepts a finite language can be transformed into a DFA having at most $2 \cdot 2^{n/2} - 1$ states if n is even and at most $3 \cdot 2^{\lfloor n/2 \rfloor} - 1$ states if n is odd. Furthermore, it showed that this upper bound is the best possible in the sense that there is an infinite sequence of languages $(L_n)_{n \geq 1}$ such that there is an NFA with n states and each equivalent DFA has at least $2 \cdot 2^{n/2} - 1$ states if n is even and at least $3 \cdot 2^{\lfloor n/2 \rfloor} - 1$ states if n is odd. As each NFA accepting a finite language only traverses a finite number of nondeterministic moves for every accepting computation, this result already showed that limited nondeterminism can yield (almost) exponential savings, even for finite languages over binary alphabets (the generalized case of finite languages over arbitrary alphabets is studied in [SY97]). Let NFA(fin,bin) denote all NFAs accepting finite languages over binary alphabets. We have:

$$\begin{array}{ccc} NFA(\text{fin,bin}) \xrightarrow{s} & DFA & \\ n & \leq \begin{cases} 2 \cdot 2^{n/2} - 1 & \text{if } n \text{ is even,} \\ 3 \cdot 2^{\lfloor n/2 \rfloor} - 1 & \text{if } n \text{ is odd.} \end{cases} & \text{[Man73]} \\ NFA(\text{fin,bin}) \xrightarrow{s} & DFA & \\ n & \geq \begin{cases} 2 \cdot 2^{n/2} - 1 & \text{if } n \text{ is even,} \\ 3 \cdot 2^{\lfloor n/2 \rfloor} - 1 & \text{if } n \text{ is odd.} \end{cases} & \text{[Man73]} \end{array}$$

[KW80] investigated the situation when nondeterminism in NFAs is even further restricted, to the point where the automaton is "almost" deterministic except for a "small" nondeterministic portion. The NFAs considered make at most a finite number of nondeterministic moves bounded by a function depending on the number of states of the automaton. The functions $g(n)$ considered are asymptotically smaller than logarithmic functions, $g(n) \ll \log(n)$ ($f(n) \ll g(n)$ means $\lim_{n\to\infty} f(n)/g(n) = 0$). Furthermore, only NFAs with at most three choices in a nondeterministic move, $|\delta(p,\sigma)| \leq 3$ for all $p \in Q$, $\sigma \in \Sigma$ were considered. This implies that, for such an NFA with n states, by applying the subset construction only subsets of states whose cardinality is less than or equal to $3^{g(n)}$ can be reached and thus an equivalent DFA with at most $\sum_{0\leq i\leq 3^{g(n)}} \binom{n}{i} \ll 2^n$ states can be obtained. Hence, the trade-off achievable is less than exponential.

Restricting the number of nondeterministic moves to less than logarithmic in the number of states is indeed a strong limitation. [Leu98a] showed that the number of nondeterministic moves in an NFA with n states, if limited, is bounded by $2^n - 2$ and also presented examples where this bound is almost reached.

[KW80] proved that for any given function $g(n) \ll \log(n)$, there is a sequence of languages $(L_n)_{n\geq c}$ for some constant c such that there is an NFA accepting L_n having n states, making at most $g(n)$ nondeterministic moves on all inputs, and each DFA accepting L_n must have at least $\sum_{0\leq i\leq 2^{g(\sqrt{n})}} \binom{O(\sqrt{n})}{i}$ states.

In other words, for any two functions $g_1(n)$, $g_2(n)$ with $g_1(n) \ll g_2(n) \ll \log(n)$ two sequences of languages $(L_n^1)_{n\geq 1}$ and $(L_n^2)_{n\geq 1}$ can be exhibited such that the succinctness achieved by NFAs making at most $g_2(n)$ guesses on their input over the corresponding minimal DFAs for $(L_n^2)_{n\geq 1}$ is considerably larger than the succinctness achieved by NFAs making at most $g_1(n)$ guesses on their input over the corresponding minimal DFAs for $(L_n^1)_{n\geq 1}$.

To put it yet another way: Even very small amounts of nondeterminism may lead to NFAs that are substantially smaller than equivalent DFAs, and more nondeterminism allows for increased succinctness of the NFAs over equivalent DFAs.

2.3 Spectra of Regular Languages

A logical extension of this idea is to study how the descriptional complexity of finite automata varies with different amounts of nondeterminism allowed when considering a single language. This concept of the so-called spectrum of a regular language was studied in [GKW90].

Apart from introducing the concept of a spectrum, [GKW90] also refined the measure used for nondeterminism in finite automata by introducing "branching" and "guessing". These measures allow for a much more finely grained assessment of the amount of nondeterminism used than simply counting nondeterministic moves.

The refinement in measuring is based on the observation that the amount of nondeterminism used in a nondeterministic configuration (p, σ, q) is not always equal, but varies with the cardinality $|\delta(p, \sigma)|$ of possible successor states. Intuitively, there is no difference in the amount of nondeterminism if traversing two nondeterministic moves with two possible successor states each and traversing a single nondeterministic move with four possible successor states.

In the following, we will present the definitions of branching and guessing as given in [GKW90]. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. The branching and guessing of a move (p, σ, q), $\beta_M(p, \sigma, q)$ and $\gamma_M(p, \sigma, q)$, are given by $\beta_M(p, \sigma, q) = |\delta(p, \sigma)|$ and $\gamma_M(p, \sigma, q) = \log_2(|\delta(p, \sigma)|)$. The branching and guessing $\beta_M(\mu_1 \ldots \mu_r)$ and $\gamma_M(\mu_1 \ldots \mu_r)$ of a computation $\mu_1 \ldots \mu_r$ in M are $\beta_M(\mu_1 \ldots \mu_r) = \beta_M(\mu_1) \cdot \ldots \cdot \beta_M(\mu_r)$ and $\gamma_M(\mu_1 \ldots \mu_r) = \gamma_M(\mu_1) + \ldots + \gamma_M(\mu_r)$. The branching and guessing $\beta_M(w)$ and $\gamma_M(w)$ of a word $w \in T(M) - \{\epsilon\}$ are defined by $\beta_M(w) = \min\{\beta_M(\mu_1 \ldots \mu_r) \mid \mu_1 \ldots \mu_r$ is an accepting computation for $w\}$ and $\gamma_M(w) = \min\{\gamma_M(\mu_1 \ldots \mu_r) \mid \mu_1 \ldots \mu_r$ is an accepting computation for $w\}$, and $\beta_M(\epsilon) = 1$ if $\epsilon \in T(M)$ and $\gamma_M(\epsilon) = 0$ if $\epsilon \in T(M)$. The branching and guessing β_M and γ_M of the automaton M are $\beta_M = \sup\{\beta_M(w) \mid w \in T(M)\}$ and $\gamma_M = \sup\{\gamma_M(w) \mid w \in T(M)\}$. If $T(M) = \emptyset$, then $\beta_M = 1$ and $\gamma_M = 0$.

Intuitively, $\gamma_M(\mu)$ is the number of bits of information needed to single out and record the move μ from among the other moves that the automaton could have selected at any point in a computation where this move occurs.

Notice that for a word w and a computation $\mu_1 \ldots \mu_r$ for w in an automaton M, $1/\beta_M(\mu_1 \ldots \mu_r)$ is the probability that M will choose this computation for w if the probabilities for nondeterministic transitions are uniformly distributed. Furthermore, $\beta_M(\mu_1 \ldots \mu_r)$ reflects the amount of parallelism needed for a real-time simulation of this computation. Since acceptance of a word w can be discovered by only examining computations with branching less than or equal to β_M, for NFAs having finite nondeterminism, β_M reflects the amount of parallelism needed for a real-time simulation of M. Hence, the branching of an automaton M is the minimal amount of nondeterminism M needs to accept all words of $T(M)$. In-depth details on motivations behind these measures can be found in [GKW90].

If $|\delta(p, \sigma)| \leq 2$ for all $p \in Q, \sigma \in \Sigma$, the guessing as defined above counts exactly the number of nondeterministic moves. By the same token, the measure used in [KW80] and the above measures are strongly related. For a computation in any NFA, the guessing is always at least equal to the number of nondeterministic moves in that computation. As only NFAs with $|\delta(p, \sigma)| \leq 3$ for all $p \in Q, \sigma \in \Sigma$ are considered in [KW80], the guessing of a computation is at most by a factor of $\log_2(3)$ higher than the number of nondeterministic moves.

Whereas guessing seems to be the natural choice for measuring nondeterminism, it may fail to be an integer. Therefore, it is technically more convenient to use branching for nondeterminism in finite automata with limited nondeterminism. Notice that (for finite values) guessing is always the binary logarithm of branching for moves, computations, words and automata.

With the measures defined, we can now proceed to the concept of the spectrum of a regular language. The spectrum of a regular language captures the size of minimal NFA descriptions for a language L for any possible finite amount of nondeterminism and for unlimited nondeterminism as well: Let L be a regular language. The spectrum of L, $\sigma(L)$, is the infinite sequence (with endpoint) $\sigma(L) = (\sigma_1(L), \sigma_2(L), \ldots, \sigma_i(L), \ldots; \sigma_\infty(L))$, where $\sigma_i(L)$ is the minimal number of states of an NFA accepting L with branching of at most i.

Although a spectrum is an infinite sequence of integers, it is monotone and bounded and its entries assume just a finite number of values. Hence the entire spectrum contains just a finite amount of information. Furthermore, the spectrum for any regular language is computable (see [GKW90], [Leu98a]).

The spectrum $\sigma(L)$ of a regular language L indicates how "inherently nondeterministic" the language is: The entry $\sigma_1(L)$ reflects the minimal size of an incompletely specified DFA for L (which can be converted to a regular DFA with at most one additional state). At the other extreme, the entry $\sigma_\infty(L)$ shows the minimal number of states an NFA needs for L with arbitrary amount of nondeterminism. The middle entries describe the situation where only limited amounts of nondeterminism may be used.

The question is what entries are possible in the middle of a spectrum, especially in cases where unlimited nondeterminism enables exponential savings between NFAs and DFAs, like for example in the case of the languages exhibited in [MF71] and [Moo71].

Intuitively, two extreme cases could occur: Either no amount of limited nondeterminism helps at all, which implies that NFAs with limited nondeterminism are just as large as a DFA for that language, or it might be that already very small amounts of nondeterminism allow for much more concise representations of the language by NFAs. Indeed, [GKW90] showed that both of these cases can truly arise.

Let us first focus on the case where limited nondeterminism does not help. Interesting in itself, the paper presents a construction converting any regular language L to a language L' such that limited nondeterminism does not help for L', i.e., all NFAs with limited nondeterminism for L' are as large as an NFA for L' not using any nondeterminism at all. Applying this construction to languages such as exhibited in [MF71] or [Moo71] yields an upper bound on spectra of regular languages as follows:

For every regular language L, $\sigma(L) \leq (2^n - 1, 2^n - 1, \ldots, 2^n - 1, \ldots; n)$, where $n = \sigma_\infty(L)$. Furthermore, this bound is approximately the best possible in the sense that, for each $n \geq 1$, there is a regular language L_n with $\sigma(L_n) = (2^{n-1}, 2^{n-1}, \ldots, 2^{n-1}, \ldots; n)$.

Thus, indeed there are languages for which limited nondeterminism does not lead to a reduction of the number of states necessary to describe the language, whereas using arbitrary nondeterminism leads to exponentially more concise automata.

On the other hand, [GKW90] also investigated how many states can be saved at best when only limited amounts of nondeterminism are permitted and hence how to establish a lower bound on spectra. This lower bound is obtained by a modified version of the subset construction which is more effective in the case of small amounts of nondeterminism. Furthermore, they exhibited a sequence of languages proving that the presented construction cannot be significantly improved.

For any regular language L, if $\sigma_1(L) \geq 2^n - 1 > 1$ then $\sigma_i(L) \geq 2^{n/i}$ for $1 < i \leq n/\log_2(n)$, $\sigma_i(L) \geq n$ for $n/\log_2(n) \leq i \leq \infty$, and these lower bounds are approximately the best possible in the sense that, for each $n > 1$, there is a regular language L_n such that $\sigma_1(L_n) = 2^n$, $\sigma_i(L_n) < 2i \cdot 2^{n/i}$ for $1 < i \leq n$, $\sigma_i(L_n) < 4n$ for $n \leq i < \infty$, $\sigma_\infty(L_n) = n + 1$.

Hence, there are cases in which already small amounts of nondeterminism result in substantial savings. The most rapid rate at which a spectrum can decrease is for the i-th entry to be essentially just the i-th root of the first entry until the maximum decrease has been attained after about $n/\log_2(n)$ steps.

Using proving techniques from communication complexity as exhibited in [Kla98], [Kla00] recently showed that there are also languages where finite nondeterminism gradually helps like above, but only after a certain "threshold" amount of nondeterminism has been reached. So, there are indeed languages situated between the bounds on spectra presented in [GKW90].

It should be stressed that the above results rely on the use of the refined measurements for nondeterminism as in fact an NFA making at most a single nondeterministic move in every accepting computation can achieve exponential savings in the number of states compared to a DFA. In this case, however, the number of possible successor states in this move is very high, close to all states of the NFA (see [Kap00] for details). As the results from [KW80] show, this is not the case if the number of successor states in all nondeterministic moves is bounded by three (or any other finite constant).

2.4 Finite Automata with Multiple Initial States

Another possible restriction to nondeterminism is to limit the points at which nondeterminism may occur in a computation. Allowing only a single guess before the computation actually starts, we come to MDFAs, finite automata with multiple initial states and a deterministic transition function. This model was studied in [Kap00] and [HSY00]. The only nondeterminism occurring in MDFA is to guess the initial state whereas the rest of the computation proceeds deterministically. Consequently, the branching of such an automaton is measured by the number of initial states and thus is always finite.

By the subset construction, for each MDFA with n states and k initial states there is an equivalent DFA with at most $\sum_{1 \leq i \leq k} \binom{n}{i}$ states. [HSY00] showed that this bound is indeed the best possible by exhibiting an infinite sequence of languages providing exactly this trade-off. Denoting the class of all MDFA with at most k initial states by MDFA($\beta \leq k$), we have:

$$\begin{array}{ccl} MDFA(\beta \leq k) & \xrightarrow{s} & \quad DFA \\ n & & \leq \sum_{1 \leq i \leq k} \binom{n}{i} \text{ [RS59]} \\ MDFA(\beta \leq k) & \xrightarrow{s} & \quad DFA \\ n & & \geq \sum_{1 \leq i \leq k} \binom{n}{i} \text{ [HSY00]} \end{array}$$

Hence, in the extreme case that each state is an initial state (a special case first considered in [GK74] and [VG79]), there are MDFA with n states and n initial states such that every equivalent DFA needs at least $2^n - 1$ states. On the other hand, using the above result by [GKW90] that there are cases where finite nondeterminism does not help and the fact that each MDFA can easily be converted to an equivalent NFA with finite nondeterminism and at most one additional state, [Kap00] pointed out that there are also cases where NFAs can achieve exponential savings over DFAs and that NFAs are also exponentially concise over MDFAs.

$$\begin{array}{ll} NFA \xrightarrow{s} & MDFA \\ n & \geq \Omega(2^n) \text{ [Kap00]} \end{array}$$

In the case of finite nondeterminism however, [Kap00] proved that for each NFA with n states and branching k there is an equivalent MDFA with $kn+1$ states and k initial states. In other words, the nondeterminism in an NFA with finite branching can be shifted from the transition function into the initial states. Let $NFA(\beta \leq k)$ denote the class of all NFAs with branching of at most $k < \infty$. We have:

$$\begin{array}{ll} NFA(\beta \leq k) \xrightarrow{s} & MDFA \\ n & \leq kn+1 \text{ [Kap00]} \end{array}$$

2.5 Unary Languages

An important special case are unary languages. [Man73] showed that for finite unary languages NFAs are just as good (or bad) as DFAs (except for one additional state needed for completely specifying the DFA). Thus, for finite unary languages nondeterminism does not help at all. For infinite languages, NFAs can be more concise than DFAs, yet not as much as in the binary case. [Chr86] proved that, in the unary case, for each NFA with n states it is possible to construct a DFA with at most $O(e^{\sqrt{n \log(n)}})$ states and that this is the best possible construction in the sense that for each n there is a unary NFA with n states such that any equivalent DFA has at least $\Omega(e^{\sqrt{n \log(n)}})$ states. Initial results indicate that the concept of the spectrum of a regular language can also be successfully applied to the unary case [Oet92].

3 Finite Automata with Limited Ambiguity

In the previous section, we studied the trade-offs in descriptional complexity between NFAs with varying amounts of nondeterminism. The measure of nondeterminism is computed by considering only accepting computations that consume the least amount of guessing. Another measure called ambiguity takes into account all of the accepting computations.

An NFA is said to be k-ambiguous if every string in the language is accepted with at most k different accepting computations. An unambiguous NFA (UFA) is a 1-ambiguous NFA. A UFA is allowed to use nondeterminism, but, like a DFA, it can only accept strings in a unique way.

Schmidt [Sch78] first showed that there are exponential trade-offs in succinctness between DFAs, UFAs and NFAs. Stearns and Hunt [SH85] proved slightly stronger results which we summarize as follows:

$$\begin{array}{ll} UFA \xrightarrow{s} & DFA \\ n & \geq 2^{cn} \end{array}$$

$$\begin{array}{ccc} NFA & \xrightarrow{s} & UFA \\ n & & \geq 2^{n-1} \end{array}$$

An NFA is said to be finitely ambiguous (FNA) if the NFA is k-ambiguous for some positive integer k. An NFA is polynomially ambiguous (PNA) if there exists a polynomial p such that every string x in the language is accepted with at most $p(|x|)$ accepting computations. Given an NFA $M = (Q, \Sigma, \delta, q_0, F)$, any input string of length n can have at most $|Q|^n$ different accepting computations. Therefore, every NFA is exponentially ambiguous in the sense that any string in the language is accepted with at most exponentially that many accepting computations.

Considering the finite language $L_n = \{x\#y \mid x, y \in \{0,1\}^n, x \neq y\}$ introduced by Schmidt [Sch78], the following result can be obtained:

$$\begin{array}{ccc} FNA & \xrightarrow{s} & UFA \\ n & & \geq 2^{c\sqrt{n}} \end{array}$$

Leung [Leu98b] studied the language $(0 + (01^*)^{n-1}0)^*$ and showed that:

$$\begin{array}{ccc} NFA & \xrightarrow{s} & PNA \\ n & & \geq 2^n - 1 \end{array}$$

Using techniques from communication complexity, Hromkovič et al. [HKK$^+$] also proved an exponential trade-off in size between PNAs and NFAs. However, it is an open problem whether there are exponential trade-offs in size between FNAs and PNAs. A partial result is proved in [HKK$^+$] which shows that there is a family KON_m of languages such that KON_m can be accepted by a linearly ambiguous PNA with size $m+2$, but any k-ambiguous FNA for KON_m has size at least $2^{(m-1)/k} - 2$.

Ravikumar and Ibarra [RI89] considered succinctness questions for unary languages. It is shown that there can be exponential trade-offs in size between DFAs and UFAs, and between UFAs and FNAs. According to the result of Chrobak [Chr86], every n-state NFA over a unary alphabet can be accepted by an $O(n^2)$-state FNA. That is, polynomially ambiguous (or exponentially ambiguous) NFAs are not needed in succinct representations of unary regular languages.

A rather obvious question comes to mind, namely whether the measures of nondeterminism and ambiguity are related. It is shown in [GLW92] that sublinear unbounded amounts of nondeterminism are possible for NFAs, and when that happens the degree of ambiguity must be infinite.

[HKK$^+$] proposed studying the fine structure of languages with regard to finite ambiguity. Similar to the spectrum concept in the previous section, we may want to consider how the size of NFAs varies as the amount of ambiguity increases. Specifically, one may wonder if there are exponential trade-offs in size between k-ambiguous NFAs and $(k+1)$-ambiguous NFAs.

4 Two-Way Finite Automata from the Perspective of Limited Resources

It is well known that both two-way deterministic finite automata (2DFA) and two-way nondeterministic finite automata (2NFA) describe regular languages. There are a number of "simulation" results (by Rabin-Scott [RS59], Shepherdson [She59], Vardi [Var89] and Birget [Bir93]) which showed how one model of two-way finite automata can be simulated by other models of finite automata which include the conversions from 2DFAs to DFAs, 2DFAs to NFAs, 2NFAs to DFAs and 2NFAs to NFAs. However, there are only very few "separation" results.

When comparing the descriptional complexity between different formal models, we consider two models to be "separated" when one model is shown to be exponentially more succinct than the other model for representing the same sequence of objects.

For the problem of comparing the descriptional complexity of different models of two-way finite automata, the main problem ([Sip80]) of whether 2DFAs and 2NFAs can be separated is still open. That is, is there a family of regular languages L_n such that to describe L_n, a 2DFA requires an exponential number of states whereas a 2NFA requires only a polynomial number of states?

The main problem has been open for a long time and seems to be very difficult to resolve. An alternative approach is to compare the different models from the perspective of limited resources. Another approach is to study restricted models of two-way finite automata.

On the other hand, there are a few separation results between two-way finite automata and one-way finite automata. Meyer and Fischer [MF71] showed that there is a family of $O(n)$-state 2DFAs such that the smallest equivalent DFA has at least n^n states. Sakoda and Sipser [SS78] showed that there is a family of $(2n)$-state 2NFAs such that the smallest equivalent DFA has at least $2^{(n-2)^2}$ states. In [Dam97], it is shown that there is a family of $O(n)$-state 2DFAs such that the smallest equivalent NFA has at least 2^n states.

4.1 Sweeping Automata

Sweeping automata are a restricted model of 2DFAs. A sweeping automaton is a 2DFA that changes direction only at either end of the input. Sipser showed that one-way NFAs and sweeping automata are separated. Specifically, there exists a family of n-state NFAs over an alphabet of size 2^{n^2} such that the equivalent sweeping automaton has at least 2^n states. With respect to the regular languages $L_n = (0+(01^*)^{n-1}0)^*$ over a binary alphabet, Leung [Leu] showed a similar tight result separating NFAs from sweeping automata. In [Mic81] and [Ber80] it is shown that 2DFAs can be exponentially more succinct than sweeping automata.

4.2 Two-way finite automata over unary alphabets

We summarize the main results on descriptional complexity of two-way finite automata over unary alphabets as follows:

– Any n-state 2NFA can be converted to a $O(n^{\log n+4})$-state 2DFA [GMP01].

– Any n-state 2NFA can be converted to a $O(e^{\sqrt{n \log n}})$-state one-way DFA [MP00a].
– Any n-state one-way NFA can be converted to a $O(n^2)$-state 2DFA, and the conversion is tight [Chr86].

4.3 Two-way Las Vegas finite automata

In [HS01b], the model of two-way Las Vegas finite (2LVFA) automata was introduced. A Las Vegas finite automaton is a nondeterministic automaton where each transition is assigned a probability. A string is accepted (rejected) if the probability for arriving at an accepting (rejecting) state is at least $\frac{1}{2}$. It is further required that a Las Vegas automaton cannot make mistakes: If there is a computation that ends in an accepting (rejecting) state, then the string processed must (not) be in the language of the automaton. This new probabilistic two-way finite automata model is studied against 2DFAs and 2NFAs. The following results are obtained. There are languages M_k such that M_k can be accepted by $(2k+O(1))$-state 2NFAs, but any 2LVFA requires at least k^2-2 states to accept. There are languages S_k such that S_k can be accepted by $O(k)$-state 2LVFAs, but any 2DFA requires $\Omega(k^2/\log k)$ states to accept. In both cases, the trade-offs are quadratic, not exponential.

4.4 Two-way finite automata using limited resources

Damanik [Dam97] considered 2DFAs with the number of turns restricted. An automaton in the class 2DFA(k) is allowed to use at most k left moves for accepting an input. The two-way spectrum (which is similar to the spectrum concept presented in section 2) of a regular language is studied. Klauck [Kla98] studied k-visit 2NFAs with limited amounts of nondeterminism. Note that a k-visit 2NFA is allowed to visit each input bit at most k times. It is shown that for any positive integers s and k there is a language $F_{s,k} \subset \{0,1\}^n$ which can be accepted by a k-visit 2DFA with $kn^{O(s)}$ states, whereas any equivalent $(k-1)$-visit 2NFA using s guess bits requires $2^{\Omega(n/(s^2k^3 \log n))}$ states.

4.5 Other works

Berman and Lingas [BL77] showed how the descriptional complexity question for two-way finite automata relates to the open problem in computational complexity of whether deterministic logarithmic space (L) is properly contained in nondeterministic logarithmic space (NL).

Birget ([Bir93], [Bir96]) studied trade-off results for length-preserving homomorphisms and in [Bir93] also studied trade-off results with respect to two-way alternating finite automata.

With respect to a strong equivalence concept called “positional simulation”, Kannan [Kan83] showed that, to simulate an n-state 2NFA, a 2DFA may need $\geq 2^{\log n^{\log \log n}}$ states, which is not a polynomial. Birget [Bir92b] showed that for every 2NFA there is a 2DFA which simulates it positionally.

(Remark: Let A and B be two-way finite automata. A is said to simulate B positionally if for every state p in B there is a state q in A such that, for all strings u and v, A accepts uv when started in the initial configuration uqv if and only if B accepts uv when started in the initial configuration upv.)

5 Limiting Nondeterminism in Pushdown Automata

In contrast to finite automata and Turing machines, pushdown automata (PDAs) can recognize additional languages through the use of nondeterminism. Thus, the first question we would ask from the perspective of descriptional complexity—"What effect can removing nondeterminism have on the size of a PDA?"—must include the provision, "when it is possible to do so." As is often the case when this additional provision is required, the answer is that the effect on size is recursively unbounded (i.e., grows faster than any recursive function) [Val76]. To progress past this point, we need a way to measure nondeterminism. We can then ask, "What effect can removing a specified amount of nondeterminism have on the size of a PDA?"

Several ways to measure nondeterminism in PDAs have appeared in the literature. Before we address questions involving the descriptional complexity of PDAs in Section 6, we would like to discuss these different measures of nondeterminism. In this section, we will consider three ways to measure a pushdown automaton's use of nondeterminism. One of the measures is static; the other two are dynamic. The static measurement is based only on the structure of the automaton, whereas the dynamic measurements are based on the automaton's behavior.

The effect, in terms of descriptional complexity, of changing the amount of nondeterminism in a PDA has received little attention except in the special case of PDAs that make only a bounded number of nondeterministic moves, whatever the length of their input. (We will say such PDAs have *finite* nondeterminism.) Since the question of how to measure nondeterminism does not arise in this case, we will compare the different measures in the context of concrete complexity rather than descriptional complexity. Thus, the material in this section will bear a greater resemblance to time and space hierarchies of complexity classes than to descriptional complexity.

5.1 Measuring nondeterminism in PDAs

If we wish to measure nondeterminism in a pushdown automaton dynamically, we must decide how to "charge" a PDA for its use of nondeterminism in handling each input string. The simplest choices would be to charge a PDA for the maximum amount of nondeterminism it ever uses while processing a particular input string, or to charge it only for the amount of nondeterminism it uses during its "best" computation on the string. We will consider both choices. Following Salomaa and Yu [SY93], we call these two types of measures *maxmax* measures and *minmax* measures, respectively. The second "max" refers to maximizing the cost, for each value of n, over all input strings of length n.

We must also decide whether or not to take into account nondeterminism that occurs during unsuccessful computations. We choose to ignore any "unsuccessful" use of nondeterminism.

Finally, we must decide whether it should cost more to choose one of many things (e.g., to roll dice) than to choose one of two things, that is, to make a binary choice (e.g., to flip a coin). We will call measures that ignore this distinction "coarse" measures, and will call those in which the cost of rolling dice is the same as the cost of an equivalent combination of coin flips, or in which dice are forbidden altogether (i.e., which require all guesses to be binary), "fine" measures. Fine measures are more precise (and also, we believe, more natural); for example, fine measures can distinguish languages that require k guesses from those that require $k+1$, while coarse measures cannot.

For time- or space-bounded Turing machines, deciding whether to use a maxmax or minmax measure of nondeterminism is of little consequence. (For a well-behaved function $f(n)$ and an input string w, a Turing machine could compute $f(|w|)$ within the available resource bound, and terminate any computation that attempts to use more than that amount of nondeterminism. Thus, a machine whose nondeterminism satisfies a minmax bound can be converted into a machine that satisfies the more stringent maxmax bound.) While this is also true for PDAs that have finite nondeterminism, we will see that there is a considerable difference between maxmax and minmax measures for PDAs in general.

The study of nondeterminism as a measurable resource was initiated by Kintala and Fischer in a 1977 paper that applied the minmax measure to real-time Turing machines [KF77]. In a paper the following year [Kin78], Kintala considered PDAs with finite nondeterminism, in effect applying a fine measure to a restricted class of PDAs for which the max/min distinction is irrelevant. In a 1981 paper [VS81], Vermeir and Savitch extended the measurement of nondeterminism to PDAs having infinite amounts of nondeterminism. However, instead of following the lead of Kintala and Fischer, they used a (coarse) maxmax measure rather than a minmax measure. While a maxmax measure is simple to handle technically, it tends to degenerate: within any accepting computation containing a sufficiently large number of nondeterministic moves, we can find a pair of segments which contain at least one of the nondeterministic moves, and which can be "pumped up" or "iterated", as in the pumping or iteration lemma for CFLs. Hence, in this measure, if the nondeterminism in a PDA is infinite, then it grows linearly. Furthermore, in a coarse measure, all finite amounts of nondeterminism are equivalent, since each can be realized by a single nondeterministic move having a large fan-out. Thus, the maxmax measure in [VS81] assumes only three non-equivalent values: zero (yielding the deterministic CFLs), finite (yielding finite unions of deterministic CFLs), and linear (yielding all CFLs).

In part because of the degeneration of this dynamic measure, Vermeir and Savitch also considered a static measure, the "(nondeterministic) depth" of a PDA. Consider the context-free languages

$$L_1 = \{ww^R \mid w \in \{0,1\}^*\}\,,$$
$$L_2 = \{0^i 1^j 2 \mid j = i \text{ or } j = 2i\}^*\,,$$

where w^R is the left-right reversal of the string w. In the maxmax measure, both L_1 and L_2 require a linear number of guesses. Yet there is a sense in which L_1 appears to involve less nondeterminism than L_2: L_1 can be recognized by a PDA having two deterministic pieces (one for pushing, one for popping) with a nondeterministic jump between them. In the depth measure, this jump is counted just once. In general, the depth of a PDA is the maximum number of jumps that can be made in moving irreversibly from one deterministic submachine to another in an optimal decomposition into deterministic submachines. Thus, L_1 has a nondeterministic depth of 1, while L_2 is said to have infinite nondeterministic depth, because a PDA that recognizes it cannot be decomposed into deterministic submachines connected by irreversible jumps. Vermeir and Savitch show that the depth measure is equivalent to counting the number of markers that must be inserted into the strings of a CFL to make it deterministic; thus, a single marker at the middle of each string suffices to make L_1 deterministic. Unlike the maxmax measure, the depth measure produces an infinite hierarchy of CFL complexity classes. (See [SY94] for a corrected and extended treatment of the depth measure.)

5.2 The minmax measure

In a 1993 paper, Salomaa and Yu [SY93] pointed out that the first hierarchy in [VS81] collapsed not because the measure was dynamic, but because it was a maxmax measure, and they introduced the minmax measure for PDAs. (They used a coarse measure, but we will discuss the fine version of the minmax measure, as in [GLW97].)

There are other reasons to use a minmax measure instead of a maxmax measure beside the desire to find a nondegenerate measure. For example, the minmax measure is based on a nondeterministic automaton's best rather than worst effort, in the same way that its basic behavior (recognizing a language) is. And the minmax measure reflects the amount of parallelism (the number of processes running simultaneously) that is sufficient to simulate the nondeterminism (see [GLW97]). However, the minmax measure is much more difficult to work with because of the fact that only a PDA's best efforts matter. Thus, if we take a computation that recognizes a string while using as little nondeterminism as possible, and pump up some nondeterministic segments of it, then it is true that we can obtain a computation that uses a linear amount of nondeterminism to recognize a long string. However, there may be another computation that recognizes the same long string in a completely different way using just a small amount of nondeterminism.

Despite these technical difficulties, Salomaa and Yu were able to prove that the recognition of a particular CFL requires more than a finite but less than a linear amount of nondeterminism in the minmax measure [SY94] (see also [SY93]), and Salomaa, Wood and Yu were able to prove that the recognition of a particular CFL requires at least $\Omega(\log n)$ nondeterminism in this measure [SWY94].

On the other hand, as long ago as 1978, Kintala asked whether every PDA that recognizes the CFL L_1 using binary guesses must infinitely often make at least $n/2$ guesses on inputs of length n. Yet because of the technical difficulties inherent in the minmax measure, this question remained open for nearly two decades. In fact, for several years it could not be proved that the recognition of any CFL, even L_2, requires a linear amount of nondeterminism in the minmax measure.

To see how minimization can result in complex behavior even in a PDA as simple as a one-counter automaton (a PDA with a unary stack alphabet), consider the CFL

$$L_3 = \{\, x10^i1y \in (0+1)^* \mid i \leq |x| \,\} \,.$$

The obvious way for a PDA to recognize this language is to count the length of its input until it guesses (upon reading a 1) that it is time to count down against consecutive 0's. The only strings in the language that are relevant to the calculation of the rate of consumption of nondeterminism by a PDA are those strings whose length is minimal among all strings on which the PDA expends the same amount of nondeterminism. In this case, these are the strings

$$w_j = 10^1 10^3 10^7 10^{15} 1 \cdots 10^{2^{j-1}-1} 11, \quad j \geq 1 .$$

Since the PDA requires j binary guesses to recognize the string w_j of length 2^j, the rate of consumption of nondeterminism even in this simple PDA is $\lfloor \log_2 n \rfloor$, and while we conjecture that no PDA can recognize this language using nondeterminism at a rate that is $o(\lfloor \log_2 n \rfloor)$, we do not know how to prove it.

More recently, some further progress in handling the minmax measure was made in [GLW97], from which the preceding example was taken. The following results may be found there.

1. Through the use of constructions similar to but more complex than those in L_3, it can be shown that, for every unbounded monotone recursive function $f(n)$, there is a CFL that contains an amount of nondeterminism that is $O(f(n))$ but not $O(1)$. From this it follows that the minmax measure produces an infinite hierarchy of CFL complexity classes.
2. By an argument similar to one used by Kintala (see Lemma 3.2 of [Kin78]), it can be shown that the CFL L_2 requires a linear amount of nondeterminism. The argument is based on the fact that, for a fixed but large value of n, there is a string of the form $w_\varepsilon = 0^n 1^{\varepsilon_1 n} 2 \cdots 0^n 1^{\varepsilon_k n} 2$ in L_2 for each vector of 1's and 2's, $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_k)$, of length k. Since a PDA for L_2 must use guesses to distinguish these 2^k strings from each other, the decision tree representing these guesses must have at least 2^k leaves. Hence, there must be a path to a leaf that contains at least k guesses, and therefore a string w_ε on which the PDA makes at least k guesses. Thus, the proof is basically a counting argument.
3. If a CFL L has finite nondeterministic depth, then the number of guesses made by a PDA recognizing L can be reduced by the factor $1/k$ for every positive integer k. Since the language L_1 has depth 1, it follows that the answer to Kintala's question about the number of guesses required to recognize L_1 is, strictly speaking, *no*. However, the more informative answer is that a PDA recognizing L_1 must make a linear number of guesses. This result is more difficult to prove than the corresponding result for L_2 since it cannot be proved by a counting argument. (A PDA for L_1 must distinguish the prefixes of an input string, but since strings only have a linear number of prefixes, this only shows that a decision tree must have at least a linear number of leaves, and hence at least a logarithmic height. Thus, a counting argument only establishes a logarithmic lower bound on the number of guesses.) The proof in [GLW97] of a linear lower bound for L_1 applies a somewhat complicated pumping argument to the computations of the PDA.

6 Pushdown Automata with Limited Nondeterminism and Ambiguity

Now that the various measures for nondeterminism in PDAs have been introduced, we will summarize results about the descriptional complexity of PDAs with limited nondeterminism and ambiguity. It follows from the standard conversion techniques between PDAs and context-free grammars (CFGs) [HU79] that the results presented here for PDAs with limited ambiguity apply to CFGs with limited ambiguity.

All results involving limited nondeterminism mentioned in this chapter are based on fine minmax measures as introduced in the previous section. Following the lead set by the literature in this field, we will use two such measures, namely "branching" for finite amounts of nondeterminism and "guessing" for infinite amounts of nondeterminism.

The definitions of branching and guessing for PDAs are very similar to those introduced for NFAs in Section 2. However, in order to convey the results summarized in this chapter more precisely, we will present an informal definition of these measures. A formal definition can be found in [GLW97].

The branching (guessing) degree (in short: branching or guessing) of a single move in a PDA is the (binary logarithm of the) number of the next configurations that can be entered from the given configuration. The branching (guessing) degree of a computation is the product (sum) of the branching (guessing) of all moves in the computation. For an accepted input w the branching (guessing) degree is the minimal branching (guessing) among all accepting computations for w. For any function $f(n)$, we say that a PDA has branching (guessing) of at most $f(n)$ if, for all $n \in \mathbb{N}$, the branching (guessing) of all accepted inputs up to length n is less than or equal to $f(n)$. PDAs with a branching of one (or, equivalently, with a guessing of zero) are also called deterministic PDAs (DPDAs).

Apart from nondeterminism, we will also summarize trade-off results between PDAs using different amounts of ambiguity as a resource. As for NFAs, a PDA is said to be k-ambiguous if every string accepted has at most k different accepting computations. 1-ambiguous PDAs are also called unambiguous PDAs (UPDAs). It follows from the standard conversion techniques between PDAs and CFGs that for every k-ambiguous PDA there is an equivalent k-ambiguous CFG and vice versa, where, in order for a CFG to be k-ambiguous, it can have at most k derivation trees for every word in the language.

Let us first focus on finite nondeterminism. As already mentioned in the previous section, [Val76] showed that there is no recursive function which bounds the savings in descriptional complexity when using nondeterministic instead of deterministic PDAs. In other words, for any recursive function it is possible to find a deterministic context-free language (more precisely an infinite sequence of such languages) such that the difference between the size of the smallest PDA for the language and the smallest DPDA cannot be bounded by this function.

In fact, the nondeterministic PDAs used in [Val76] are even unambiguous. Hence, we have:

$$UPDA \stackrel{nonrec}{\longrightarrow} DPDA$$

Furthermore, the UPDAs used in [Val76] have a branching degree of two. Thus, denoting an ambiguity degree of at most k by $(\alpha \leq k)$, a branching degree of at most k' by $(\beta \leq k')$, the class of all PDAs with ambiguity of at most k and branching of at most k' by $PDA(\alpha \leq k, \beta \leq k')$, and omitting any of the two parameters if they are unbounded, we have:

$$PDA(\alpha \leq 1, \beta \leq 2) \stackrel{nonrec}{\longrightarrow} PDA(\beta \leq 1)$$

[Her97] considered arbitrary amounts of finite nondeterminism and proved that the generative capacity of PDAs increases when allowing additional amounts of nondeterminism. Moreover, he proved the interesting fact that for each $k \in \mathbb{N}$, the class of all languages that can be accepted by a PDA with branching of at most k coincides with the class of languages generated by the union of at most k deterministic context-free languages. Apart from the increase in generative capacity, [Her97] pointed out that the savings in size can be non-recursive when describing a language by a PDA with branching of degree $(k+1)$ that actually can be represented by a PDA with branching of degree k. This is even the case if the PDA with higher branching is required to be unambiguous while the PDA with lower branching may have arbitrary ambiguity. Hence, we obtain:

$$PDA(\alpha \leq 1, \beta \leq k+1) \overset{nonrec}{\longrightarrow} PDA(\beta \leq k) \text{ for all } k \in \mathbb{N}$$

The non-recursive trade-offs between PDAs with branching of $(k+1)$ and k hold even if the PDAs with higher branching are required to have only a single state and to operate in real time (cf. [Her99]).

For infinite nondeterminism, as shown in [Her99], PDAs with linear guessing allow for non-recursively smaller descriptions than those with sublinear growth. Denoting all PDAs with guessing of at most $f(n)$ by $PDA(\gamma \leq f(n))$, we have for all $c > 0$ and sublinear functions $f(n)$:

$$PDA(\gamma \leq cn) \overset{nonrec}{\longrightarrow} PDA(\gamma \leq f(n))$$

As in the finite case, the result also holds even if the PDAs with linear guessing are required to have only a single state and to operate in real time.

Furthermore, starting from the fact that there are context-free languages which require at least sublinear nondeterminism, [Her99] established an infinite hierarchy of languages with respect to sublinear nondeterminism and proved that there are also non-recursive trade-offs between classes of PDAs allowing different sublinear degrees of nondeterminism. Along the same lines, non-recursive trade-offs between PDAs with sublinear and finite nondeterminism were shown.

In short, in many cases increasing the amount of nondeterminism allowed in a PDA leads to non-recursive savings in size (and to increased generative capacity).

Let us now focus on the resource of ambiguity in PDAs. Although ambiguity and nondeterminism are related, they nevertheless constitute two different resources. The relationship between nondeterminism and ambiguity in PDAs is not as simple as in NFAs. For more details see [Her97].

[SS77] proved that there is no recursive function bounding the savings which can be achieved by arbitrary PDAs over unambiguous PDAs. We have:

$$PDA \overset{nonrec}{\longrightarrow} UPDA$$

More precisely, the PDAs used in [SS77] are ambiguous of degree two. Hence, denoting the class of all PDAs with ambiguity degree of at most k, for some integer k, by $PDA(\alpha \leq k)$, the above result can be more precisely expressed as

$$PDA(\alpha \leq 2) \overset{nonrec}{\longrightarrow} UPDA.$$

This result was generalized to arbitrary degrees of ambiguity in [Bor92] as follows. For an arbitrary integer k, allowing PDAs to have ambiguity of degree $(k+1)$ instead of degree k can yield non-recursive savings:

$$PDA(\alpha \leq k+1) \stackrel{nonrec}{\longrightarrow} PDA(\alpha \leq k) \text{ for all } k \in \mathbb{N}$$

It should be noted that this result for PDAs was proven in [Bor92] using CFGs but as an immediate consequence applies to PDAs as well.

[Her97] also studied the relationship among the resources of nondeterminism and ambiguity with respect to the descriptional complexity of PDAs. It was shown that allowing an additional amount of ambiguity can always yield non-recursive savings over descriptions not having this additional amount, even if the branching can increase from $(k+1)$ to unbounded:

$$PDA(\alpha \leq k+1, \beta \leq k+1) \stackrel{nonrec}{\longrightarrow} PDA(\alpha \leq k) \text{ for all } k \in \mathbb{N}$$

Apart from the results in descriptional complexity mentioned above, [Her97] also investigated the generative power of PDAs with limited nondeterminism and limited ambiguity and established a double, infinite hierarchy of languages with respect to nondeterminism and ambiguity. Using the union of languages from [Mau68] and [Kin78], for each $k, k' \in \mathbb{N} \cup \{\infty\}$ with $k \leq k'$ there is a language that can be accepted by a PDA in $PDA(\alpha \leq k, \beta \leq k')$ but not by any PDA in $PDA(\alpha < k)$ or $PDA(\beta < k')$. In other words, for each $k, k' \in \mathbb{N} \cup \{\infty\}$ with $k \leq k'$, there is a language which is *inherently ambiguous* of degree k and *inherently nondeterministic* of degree k'.

It should be mentioned that [Har80] proved some of the non-recursive trade-offs mentioned here very elegantly by using the valid and invalid computations of a Turing machine and a corollary to Rice's Theorem [HU79], namely that the set of all Turing machines accepting infinite sets is not recursively enumerable. Moreover, [Har83] presented a meta-theorem for non-recursive trade-offs which generalizes [Har80].

One is tempted to argue that, for example, the non-recursive trade-off between PDAs and DPDAs is a consequence of the following fact: When we are given a deterministic PDA, we can quickly decide that the PDA is deterministic and thus know that the language accepted is deterministic context-free. When given a (nondeterministic) PDA accepting a deterministic language, however, we have no way of telling whether the language is deterministic. Actually, this problem is undecidable. Thus, the trade-off between PDAs and DPDAs might be non-recursive because the DPDA comes essentially with a proof that the language accepted is deterministic.

[Har80] posed the question what would happen if we consider nondeterministic PDAs which come with a proof attached (written in some appropriate formal system) that the language accepted is deterministic if it is, indeed, deterministic. Such PDAs are called *verified* PDAs. Let the class of verified PDAs be denoted by VPDA. The following two theorems can be proven [Har80]:

$$PDA \stackrel{nonrec}{\longrightarrow} VPDA$$
$$VPDA \stackrel{nonrec}{\longrightarrow} DPDA$$

The first statement says that attaching *only a proof* that the language accepted is deterministic (as opposed to actually specifying a particular DPDA) can still cause a non-recursive blow-up in descriptional complexity over PDAs which come without such a proof. The second statement, on the other hand, says that actually specifying a particular DPDA can still cause a non-recursive blow-up in descriptional complexity over PDAs which come only with a proof—but no specific DPDA—that the language is deterministic.

7 Correct Prefix Parsers, LL(k) and LR(k) Parsers

Syntax parsers are an essential part in translating software or source code to execution code. Correct prefix parsers, i.e., parsers which detect errors at the earliest possible moment, have received special attention because of their practical importance. However, such features as, e.g., early error detection might have their price in terms of parser size which is an important practical aspect as well.

Using the concept of scanning pushdown automata, [GHSU77] exhibited an infinite sequence of languages for which correct prefix parsers have to be exponentially larger than the smallest deterministic parsers which do not have to operate in correct prefix mode. This rather dramatic trade-off is achieved because the exponentially smaller deterministic parsers can delay the recognition of an error until the very end of the input and thus, since the inputs are unbounded in length, arbitrarily long.

Thus, one might wonder how the size of the smallest parser will be affected if one delays the error detection "just a little bit". More precisely, can one gradually decrease the delay in error recognition while at the same time increasing the parser size only gradually as well?

Based on the idea of spectra of nondeterminism (cf. Section 2.3), Füssel [Füs92] introduced the notion of time-delay spectra. Intuitively, for a given language L, the k-th entry in the delay spectrum contains the size of the smallest deterministic parser which detects an error with time delay of at most k. In short, there are languages for which gradually increasing the time delay allows gradually decreasing the parser size.

In practice, many parsers are constructed on the basis of LL(k) and LR(k) grammars. Such parsers seem to be generally quite large in size, and much attention has focused on ways to reduce the size of such parsers, e.g., in [Blu01].

LR(1) grammars can describe all deterministic context-free languages, and hence any increase in the length of the lookahead will not increase the expressive power of LR(k) grammars. Therefore, at least in theory, there is no need to use longer lookaheads for LR(k) grammars. On the other hand, the class of LL(k) languages is properly contained in the class of LL($k+1$) languages. That is, more languages can be described using longer lookaheads for LL grammars.

Parr and Quong [PQ96] studied the use of LL and LR grammars in practice. They argued that programmers implement translators, not just syntactic parsers. Using example languages of practical interest, they showed that there is a need to use LL and LR grammars with lookahead lengths larger than one.

Leung and Wotschke [LW00] studied the relationship between the length of the lookahead and the size of the smallest LR(k) grammar. The size of a grammar is defined to be the total number of symbols used in all the productions of the grammar. For example, a context-free production with 4 symbols on the right hand side contributes a value of 5 to the size of the grammar. It is shown that there is a sequence of languages L_n such that, as the length of the lookahead varies, there is a gradual trade-off in the size of the LR(k) grammars describing the same language . Specifically, it is proved that any LR(k) grammar for L_n has size $2^{\Theta(n-k)}$ for $0 \leq k \leq n - 9 \log n$. That is, a linear decrease in the lookahead length causes the grammar size to increase exponentially. The technical difficulty lies in the lower bound proof that any LR(k) grammar for L_n has size $2^{\Omega(n-k)}$.

Bertsch and Nederhof [BN01] performed a similar study for LL(k) grammars and obtained results that complement the findings in [LW00]. They found another sequence of languages such that a linear decrease in the lookahead length, whenever such decrease is possible, causes the LL grammar to increase exponentially in size.

The two studies demonstrate from a theoretical perspective that using longer lookaheads in LL and LR grammars can be practically very useful for parsing certain languages, even though such increase in lookaheads is not necessary in the LR(k) case.

8 Descriptional Complexity and Cellular Automata

The preceding sections studied descriptional complexity of finite automata and pushdown automata from different points of view, using, for example, nondeterminism or ambiguity. Finite and pushdown automata are sequential computational models, since they have one finite state control, an input tape, a storage medium in the case of PDAs, and since at most one input symbol is read and processed in every time step. It is a natural generalization to consider *systems* of sequential automata. Some questions immediately arising are, for example, whether the input is processed in a parallel or sequential mode and whether the cooperation between different automata is organized in a synchronous or asynchronous way. One may ask in what manner the communication structures between different automata are designed and how appropriate restrictions on the "amount of information" communicated can be formulated. In addition, we are very interested in the benefits of such systems with regard to their descriptional complexity.

Some research in this direction was done for communicating finite state machines, for example in [BZ83] and [Kle96], where particular emphasis is placed on the descriptional complexity of such systems. Results about systems of communicating grammars, so-called grammar systems, are summarized in [DPR97] and [CVDKP94]. In general, such systems are very powerful in terms of their generative capacity. For example, in [Kle96] it is shown that every recursively enumerable language can be accepted by only two suitable communicating DFAs.

All systems just mentioned have in common that they solve their computing task acting in a distributed way, i.e., there are several different processors and the computing task is partitioned into several subtasks which themselves are solved by several distinct communicating processors. In this section we turn our attention to cellular automata (CAs), a massively parallel model of computation. In contrast to several distinct processors, a cellular automaton consists of many identical simple processors, namely DFAs, which are homogeneously connected and arranged in an array and have a very restricted form of communication: The only communication between cells is to check the current states of the right and left neighboring cell. The system works synchronously at discrete time steps by applying a local transition function to each automaton at the same time.

Cellular automata seem to be a very promising model considering practical as well as theoretical aspects. First of all, cellular automata capture some features which can be biologically motivated. For example, the growth of a plant or the contraction of a muscle is probably not governed by global rules. Both systems consist of many essentially identical physiological cells which can only communicate with a bounded number of neighboring cells. The cooperation between cells is described by a local rule. Starting from some cells which get a certain stimulus, the total system evolves according to these local rules. We can observe that local interactions induce a global behavior, i.e., the plant grows and the muscle contracts. Hence the theoretical model might be useful to describe and study phenomena of the real world. Much research in this direction is underway and there are many applications in different fields of natural and even social sciences where "real" systems are modeled and analyzed by cellular automata. More detailed information on these practical issues can be found in [BMS01], [Wei97], and [Gar95].

Also, from a theoretical point of view, cellular automata are an interesting topic, because on the one hand we can study, in addition to existing models, another parallel computational model in comparison with sequential models. Where are the advantages and limits of this form of parallelism? On the other hand, cellular automata enable us to investigate a massively parallel model in contrast to a "distributed" parallel model as described above. Yet another appealing property of CAs is their simplicity. Since all cells are identical and homogeneously connected through a simple mechanism, one might hope that cellular automata are easy to realize as hardware. Much work has been done in this field of hardware implementation and we refer to the summary in [BMS01]. Despite their simplicity, when we utilize cellular automata to recognize formal languages, the generative capacity is still rather high, since even the simplest cellular model can recognize certain context-sensitive languages. Formal language aspects of cellular automata will be discussed in more detail below.

Thus, cellular automata are an interesting model both from a theoretical and practical point of view. Succinctness results, upper and lower bounds concerning the trade-off between several systems, and the design of efficient minimization algorithms are theoretical problems for which solutions have practical relevance. Therefore, investigating the descriptional complexity of cellular automata is an important topic, on which, as far as we know, not much research has been done yet. In [GMNP97] the descriptional complexity of systolic binary tree automata is studied. This model is likewise a massively parallel system, but in contrast to CAs the model operates on tree structures and, to accept an input of length n, more than n processors must be provided.

We now want to focus on formal language aspects of cellular automata, and so we start with an informal definition of two-way and one-way cellular automata as will be needed below. A formal definition and results on the generative capacity can be found, for example in [Kut01] and [DM99]. A two-way cellular automaton is a set of many identical deterministic finite automata, called cells, which are arranged in a line. Each cell is homogeneously connected with its left and right neighbor. The next state of each cell depends on the current state of the cell itself and the current states of its left and right neighbor. We say that a cellular automaton is one-way (OCA) if each cell is only connected with its right neighbor. The transition rule is applied synchronously to each cell at the same time. Hence the automaton evolves in discrete time steps and the local transition rule achieves a global behavior. In order to investigate the generative capacity of CAs, we will shortly describe how formal languages can be recognized by cellular automata: We require as many cells as the input is long and in the beginning the input is written into the cells, each letter into one cell. The input is bounded by a special boundary symbol at both ends. Now the automaton works as described. We say that the input is accepted if there is a time step at which the first cell enters an accepting state from a previously defined set of accepting states. The language accepted by a cellular automaton A is the set of all words which are accepted by A. We now introduce a restriction on the available time within which the first cell must enter an accepting state: We say that an automaton has time complexity t if all words in the language are accepted within $t(|u|)$ many time steps where u denotes the input. The corresponding time complexity class is denoted by $\mathcal{L}_t(\mathrm{CA})$ and contains all languages which are accepted by a CA with time complexity t. Some important and well-investigated language classes are realtime ($t(n) = n$), lineartime ($t(n) = m \cdot n$ for a rational number $m \geq 1$), and arbitrary time (t is not restricted) languages. The corresponding language classes are denoted by $\mathcal{L}_{rt}(\mathrm{CA})$, $\mathcal{L}_{lt}(\mathrm{CA})$, and $\mathcal{L}(\mathrm{CA})$. In the one-way case we replace CA by OCA. Results on the generative capacity of different types of cellular automata and their relation to the Chomsky hierarchy are summarized in the following diagram: (Let REG, DCFL, LCFL, CFL, DCSL denote the families of regular, deterministic context-free, linear context-free, context-free, and deterministic context-sensitive languages.)

$$\mathcal{L}_{rt}(\mathrm{OCA}) \subset \mathcal{L}_{lt}(\mathrm{OCA})^R = \mathcal{L}_{rt}(\mathrm{CA}) \subseteq \mathcal{L}_{lt}(\mathrm{CA}) \subseteq \mathcal{L}(\mathrm{OCA}) \subseteq \mathcal{L}(\mathrm{CA})$$

$$\mathrm{REG} \subset \mathrm{LCFL} \subset \mathcal{L}_{rt}(\mathrm{OCA}),\ \mathrm{DCFL} \subset \mathcal{L}_{rt}(\mathrm{CA})$$

$$\mathrm{CFL} \subset \mathcal{L}(\mathrm{OCA}),\ \mathrm{DCSL} = \mathcal{L}(\mathrm{CA})$$

Some research on the descriptional complexity of cellular automata was started in [Mal01], and the main result is that one can prove non-recursive trade-offs between cellular automata and DFAs as well as PDAs and also among several restricted classes of cellular automata. The phenomenon of non-recursively bounded trade-offs was first studied in [MF71] on the basis of the trade-off between context-free grammars and DFAs. Additional trade-offs were shown by Hartmanis in [Har80] where a simplified and elegant proof technique is presented. Because the valid and invalid computations of a Turing machine are realtime-OCA languages, Hartmanis' technique can be successfully applied to prove the following non-recursive trade-offs:

$$\textit{realtime-OCA} \overset{nonrec}{\longrightarrow} DFA,\ \textit{realtime-OCA} \overset{nonrec}{\longrightarrow} PDA$$
$$\textit{realtime-CA} \overset{nonrec}{\longrightarrow} \textit{realtime-OCA},\ \textit{lineartime-OCA} \overset{nonrec}{\longrightarrow} \textit{realtime-OCA}$$

Realtime-OCAs are in a way the simplest cellular model, since we have one-way communication and realtime processing. But even this model can achieve non-recursively bounded savings in comparison with DFAs and PDAs. Moreover, using two-way instead of one-way communication or permitting more time than realtime would allow can lead to an enormous decrease of size. So, for regular or context-free languages, it might be preferable to describe them by realtime-OCAs as well as describing realtime-OCA languages by realtime-CAs and lineartime-OCAs, respectively. But, unfortunately, one has to pay a high price for the conciseness since the model becomes very unwieldy in terms of decidable questions. Since the valid and invalid computations of a Turing machine are realtime-OCA languages, it can simply be proven that almost all decidability questions as, for example, emptiness, finiteness, inclusion, equivalence, and regularity are undecidable and not even semidecidable. The undecidability of emptiness implies immediately that a minimization algorithm for realtime-OCAs, i.e., constructing an equivalent realtime-OCA with a minimal number of states, cannot exist. Hence it is not possible to algorithmically parallelize a given language in the CA sense. Furthermore, making use of the non-semidecidability of infiniteness one can see that no pumping lemma exists for cellular language classes. These results are not satisfactory from a practical perspective, because certain tasks, such as, for example, checking whether two CAs accept the same language, checking whether a finite or regular language is accepted or minimizing a given CA, cannot be done algorithmically, but have to be proven *ad hoc*. We can summarize that the cellular automata investigated so far are very powerful at the expense of manageability. Hence we are motivated to look for appropriate restrictions on CAs yielding cellular models which are easier to handle.

For the CAs considered so far we have to provide as many cells as the input is long which is not very realistic from a practical point of view. It is therefore an obvious restriction to introduce a cellular model which has only a fixed number of cells. This is done in [Mal02] where the model *k-cells OCA* (kC-OCA) is defined and some basic results are obtained which are summarized in the remainder of this section. A realtime-kC-OCA is an OCA with a fixed number of k cells for every input. The input mode is slightly modified since the input is only fed into the rightmost cell. But all other cells behave as in the unrestricted model and the input is accepted if the first cell enters an accepting state. Since the minimal time to read the input and to send all information from the rightmost cell to the leftmost cell is the length of the input plus k, realtime is defined as $t(n) = n + k$.

Limiting the number of cells to a fixed number k has grave consequences with regard to the generative capacity of realtime-kC-OCAs which is reduced to the regular languages. This deficit is due to the simple communication structure, since two different and suitable communicating DFAs can accept different language classes ranging from regular to recursively enumerable languages as is shown in [Kle96]. So, realtime-kC-OCAs are a parallel model for regular languages and the comparison with finite automata becomes now particularly interesting. It can be proven that the blow-up in the number of states, when converting a realtime-kC-OCA to an equivalent DFA, is bounded by a polynomial of degree k.

Constructing the Cartesian product of the k cells, for each realtime-kC-OCA with n states there is a DFA which has at most $\frac{n}{n-1}n^k = O(n^k)$ states. This upper bound is tight in order of magnitude, since one can find an infinite sequence of unary languages $(L(n,k))_{n\geq 1}$ such that a realtime-kC-OCA with n states accepts $L(n,k)$, but every DFA recognizing $L(n,k)$ needs at least $(n-1)^k + (n-1)^{k-1} + 1 = \Omega(n^k)$ states. A consequence from this lower bound is a proper hierarchy concerning the number of states: Each language recognized by an n-state realtime-kC-OCA is trivially recognized by an $(n+1)$-state realtime-kC-OCA. But there is an infinite sequence of languages $(L_n)_{n\geq 1}$ such that each L_n is accepted by an n-state realtime-kC-OCA, but no realtime-kC-OCA having less than n states can recognize L_n. We can summarize the results as follows:

$$\begin{array}{ll} \textit{realtime-kC-OCA} \xrightarrow{s} & DFA \\ n & \leq \frac{n}{n-1}n^k = O(n^k) \\ n & \geq (n-1)^k + (n-1)^{k-1} + 1 = \Omega(n^k) \end{array}$$

Finally, we want to address the problem of minimizing a given realtime-kC-OCA. Since every DFA can be converted to an equivalent realtime-kC-OCA, the construction of a minimal equivalent realtime-kC-OCA implies an optimal parallelization of a regular language in the CA sense with respect to the given k. The minimization problem for finite automata is solvable in time $O(n \log n)$ for DFAs and PSPACE-complete for NFAs [JR93]. In [Mal01] it is shown that minimization is algorithmically unsolvable for unrestricted realtime-OCAs. For realtime-kC-OCAs the following intermediate result can be obtained: The minimization problem is algorithmically solvable, but it is not known whether minimization can be done efficiently, i.e. in polynomial time. Since a minimal realtime-kC-OCA does not have to be necessarily unique, minimization is likely to be a hard computational problem.

9 Descriptional Complexity and Software Reliability

Whereas the previous sections described, summarized and illustrated known results about the descriptional complexity of machines with limited resources, this section will outline how descriptional complexity could help analyze and improve software reliability. Although no results linking descriptional complexity and software reliability have been published yet, work relating these areas could yield results not only useful in software reliability but could also lead to theoretical research topics that go beyond the "classical" problems.

Due to space limitations, defining what software reliability exactly is and how it is measured cannot be addressed in this paper. An introduction to the field and further references can be found in [Lyu95] and [Mus98]. However, it is important to point out that the term software reliability comprises a large variety of topics. To mention a specific one, many of todays software systems are component-based, possibly distributed, and the individual components are not specifically tailored to their use in a particular system, but are general purpose ("off the shelf") components. These components could be from different vendors. In such a scenario, not only the reliabilities of the individual components, but also the reliability of the overall system, influenced by the interaction between the components, becomes crucial.

Clearly, today's complex software components consisting of millions of lines of code are virtually never free of faults in the code. Therefore, apart from trying to help generate code with fewer faults, software reliability engineering has provided techniques that prevent such faults from triggering a software system failure at runtime or to alleviate a potential failure by methods such as checkpointing or rejuvenation. Furthermore, models to measure and analyze software reliability have been developed and applied. Currently, models such as Petri Nets and Markov Chains play prominent roles in this area.

There are other important aspects that are not sufficiently addressed by the above models, like, to name one, the algorithmical behavior of the components.

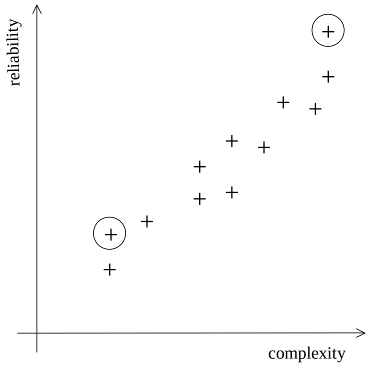

Figure 2: Descriptions of a particular object in a descriptional system. Of interest is the relationship between (one of) the simplest description providing the highest reliability among all simplest descriptions (marked by the circle in the lower left) and (one of) the simplest of all most reliable descriptions (marked by the circle in the upper right) as well as the "path" between those two. The diagram shows a case where more complex descriptions tend to be more reliable.

A first indication that automata-theoretic machine models can help to assess software reliability was [KKK00] (extending results from [Kle96]) which showed that the reliability of a software system comprised of several components cannot be computed in the Turing sense, even if the reliability of each component of such a system is known and if each component is modeled by a finite automaton. Another aspect of software systems that could be helpful in investigating software reliability is the hierarchical design of some systems as modeled for instance in communicating hierarchical finite state machines (c.f. [AKY99]).

Coming back to descriptional complexity, it is clear that, from an intuitive viewpoint, the "complexity" of a software system and its reliability are related. The simpler I can express an object like a software system, the fewer mistakes I am likely to make in expressing it and the easier it should be to express myself correctly. Therefore, the use of a descriptional system allowing for easy representations of the software under consideration could help to improve reliability. On the other hand, for a software system, "simple" does not necessarily mean "concise". When using different descriptional systems, descriptions in one of the systems may have features not present in the other system, for instance, it may be simpler to verify or prove certain properties which cannot be easily (or even not at all) algorithmically checked in the other descriptional system. Thus, the descriptional system allowing for the smallest description of an object may not always be the best choice. Different descriptional systems may be appropriate in different situations, regardless of the tradeoffs possible when only considering "size". Consequently, different measures for the "simplicity" of such systems need to be found for different situations.

The descriptional complexity of software systems is loosely related to the area of software metrics. Software metrics also measure the "complexity" of software with respect to certain properties by means of static measures (i.e., in most cases based on the program code). These metrics are used for assessing relevant aspects of the software in the development or maintenance cycle. A variety of

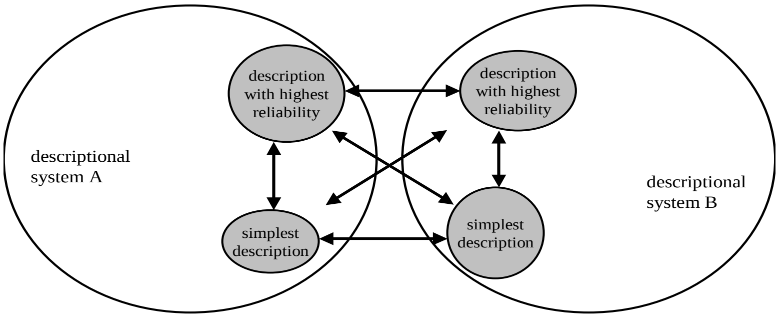

Figure 3: Situation when studying two descriptional systems.

metrics have been proposed to address different properties under consideration. As most of those are computationally intractable, the metrics used often capture the properties heuristically and not in a formal sense. Furthermore, in contrast to descriptional complexity, the focus in software metrics is on measuring the complexity of a particular program (and maybe comparing it with other programs), not on the consideration of minimal functionally equivalent programs when using different measures or descriptional systems and also not on obtaining upper and lower bounds between descriptional systems.

In order to start research relating descriptional complexity and software reliability, first and foremost, models for software systems need to be found and (possibly different) measures for their “simplicity” and their “reliability” have to be identified. As already pointed out, the simplest description might not always be identical with the most concise one. Furthermore, “simplicity” and “reliability” are two properties of a description which are likely to be measured by different means. Hence, when considering single descriptional systems, tradeoffs between “simplicity” on one hand and “reliability” on the other are to be investigated. In particular, the relationship between (one of) the simplest description providing the highest reliability among all simplest descriptions for that object and (one of) the simplest of all most reliable descriptions needs to be studied as shown in Figure 2. Furthermore, the question whether reliability can be “traded in” for simplicity and vice-versa, i.e., the path between these two extremal points, should be studied. Notice that in some cases it might be that the simplest description of an object is also the most reliable one.

When even considering two descriptional systems, a situation as illustrated in Figure 3 occurs, where the relation between (one of) the most reliable and (one of) the simplest description of the object under consideration within System A as well as their relations to (one of) the most reliable and (one of) the simplest description of that object in System B need to be clarified. Apart from these new dimensions to descriptional complexity, even determining which descriptions in a particular system describe the same object (and hence are considered “equivalent”) is a difficult task. Two descriptions considered equivalent in one scenario might not be equivalent in another one since, depending on the situation, equivalence of two descriptions could be defined differently.

It is not clear whether a model capable of capturing important aspects on one hand but yet being manageable in terms of deriving results on the other can be found immediately. Yet, it is important to take first steps in this direction and to come up with models of descriptional systems and measures for their complexity that allow to study some relevant aspects of software systems and to formally prove such properties.

10 Other Types of Devices and Resources

In this section, we will consider descriptional complexity and resource trade-offs for other types of machines. First, we will examine the trade-off between the number of states and the number of stack symbols in pushdown automata (PDAs), often called the *state stack-symbol product*.

Then we will examine some other types of machines for which interesting descriptional complexity results are known, even though little is yet known about their resource trade-offs. In particular, we shall consider alternating finite automata (AFAs), probabilistic finite automata (PFAs), quantum finite automata (QFAs), and degree automata (DAs). Based on the example of DAs, NFAs and DFAs, we will discuss the concept of *concurrent conciseness*.

We will also cite some references on the state complexity of operations on regular languages. Although this may appear to fall outside the scope of research in the area of limited resources, some of the results may in fact suggest interesting research questions in that area.

Finally, we will look briefly at descriptional systems that are not machines, such as rewriting systems and regular expressions.

10.1 The state-stack symbol product of PDAs

A natural measure of descriptional complexity for a finite automaton is the number of states, since the next move depends on the current state as well as the next input symbol. A natural measure of descriptional complexity for a pushdown automaton is the product of the number of states and the number of stack symbols, since the next move depends on the current state and the top stack symbol, as well as the next input symbol. In practice, we might wish to reduce the number of states in a PDA to a specified value while keeping the increase in the number of stack symbols as small as possible, or we might wish to reduce the number of stack symbols to a specified value while keeping the increase in the number of states as small as possible. The interaction between these two parameters, states and stack symbols, can be summarized as follows.

Every PDA having n states and p stack symbols can be transformed into an equivalent PDA that has n' states and $O((n/n')^2 p)$ stack symbols, for any desired value of n', $1 \leq n' < n$ [GPW82]. Furthermore, this result is optimal in the sense that there is a PDA (for every p and n) such that any equivalent PDA that has n' states must have at least $(n/n')^2 p$ stack symbols.

Similarly, every PDA having n states and p stack symbols can be transformed into an equivalent PDA that has p' stack symbols for any desired value of p', $2 \leq p' < p$. If this transformation is required to preserve determinism (that is, if the transformation is not permitted to introduce additional nondeterminism beyond that which was present in the original PDA), then the transformed PDA will have $O(np/p')$ states; if additional nondeterminism is allowed in the transformed PDA, then only $O(n\sqrt{p/p'})$ states are needed [GPW93]. These results on the cost of trading stack symbols for states are optimal in the same sense that the bound on trading states for stack symbols was optimal.

The transformation that reduces the number of stack symbols without using nondeterminism is straightforward. However, the transformation that uses nondeterminism to achieve greater efficiency is slightly more complex: it encodes each symbol of the larger stack alphabet into two "halves," u and v, over the smaller alphabet, so that the original stack symbol can be recovered from the combined string uv but not from either u or v alone. To decipher uv, the new PDA pops only the first part, u, off its stack; it guesses that the second part is v and it records its guess on the stack; and it continues its computation under the assumption that it has seen uv on its stack. The PDA verifies its guess when it later pops that part of the stack, and it empties its stack before accepting the input string to ensure that it has checked all of its guesses. Since the finite-state control only has to decipher half of uv at a time, the increase in the number of states is only about the square root of what it is in the "deterministic" transformation. (Note that, because of the optimality of this construction, any attempt to extend this approach—for example, by splitting the encoding into three thirds rather than two halves—will prove fruitless.)

10.2 Other types of finite automata; concurrent conciseness

In this subsection, we will look briefly at alternating finite automata (AFAs), probabilistic finite automata (PFAs), quantum finite automata (QFAs), and degree automata (DAs), and we will discuss the concept of *concurrent conciseness*.

The extent to which alternation can produce succinct descriptions of certain regular languages was studied by Leiss in [Lei81] and [Lei85]. Each AFA can be converted to a DFA with a doubly exponential blow-up in size, and this upper bound is tight. On the other hand, there is an infinite sequence of languages $(L_n)_{n\geq 1}$ for which alternation does not help: each L_n is recognized by an n-state DFA, while recognizing L_n with an AFA also requires n states.

Probabilistic finite automata were investigated in [Rab63] and [Paz71], where it was shown that PFAs with isolated cutpoints describe precisely the class of all regular languages. The upper bound obtained there on the cost of converting such PFAs to DFAs is the following: each n-state PFA with an ϵ-isolated cutpoint can be converted to a DFA having at most $(1 + (1/2\epsilon))^{n-1}$ states. Whether this upper bound can be improved or whether it is in fact tight has proved to be a difficult question, despite the attention it has attracted from various researchers: see the survey in [Fre91]. The most recent result is due to Ambainis [Amb96], who presents an infinite sequence $(L_n)_{n\geq 1}$ of languages such that there is, for each n, a PFA with isolated cutpoint that accepts L_n using just n states, although recognizing L_n with a DFA requires $\Omega(2^{n/\log n})$ states.

A thorough discussion of the effects of adding probabilistic moves to finite automata can be found in [Fre91], where PFAs, two-way PFAs, multi-head PFAs, multi-counter PFAs, and probabilistic Turing machines are compared to their deterministic counterparts. Additional information on these topics may be found in [Fre81] and [KF91]. Probabilistic transitions are studied in the context of one-way and two-way Las Vegas finite automata in [HS01a] and [HS01b] (cf. Section 4.3). Milani and Pighizzini obtain an interesting result on PFAs with unary alphabets in [MP00b]: the upper bound in this case can be reduced to $O(e^{\sqrt{n \log n}})$, and this bound is tight. This result is particularly interesting since $O(e^{\sqrt{n \log n}})$ is also a tight upper bound for the conversion of unary NFAs, of two-way DFAs, and of two-way NFAs to unary DFAs: see [Chr86] and [MP00a]. Thus, the maximal savings in the size of the set of states that can be achieved through the use of nondeterminism, of two-way motion, and of probabilistic transitions, all differ by no more than a constant factor in the unary case.

Next, we look briefly at the descriptional complexity of QFAs. An introduction to quantum computing can be found in [Gru99], and results related to descriptional complexity may be found in the detailed survey in [Gru00]. Because this field of research is still in an early stage of development, it is not yet clear how to construct an appropriate model of a QFA, and several models have appeared in the literature. For one model, the so-called *measure-many QFAs*, it is known that QFAs can achieve exponential savings compared with DFAs and PFAs [AF98]. On the other hand, it is shown in [AF98] and [Nay99] that there exists an infinite sequence of regular languages that require exponentially larger QFAs than DFAs.

We turn now to a surprisingly simple extension of NFAs that can sometimes achieve an exponential savings in the number of states required to accept a language, the *degree automaton* [KPW93]. A degree automaton is an NFA that accepts a word in a way that depends on the entire set of states that it might be in after processing the word: it accepts if the fraction of states in this set that are accepting states exceeds a preset *cutpoint* λ. More formally, we have the following definition.

Definition 4 *Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA and, for $x \in \Sigma^*$, let*

$$t_M(x) := \{\, q \mid q \in \delta(q_0, x) \,\} \quad \text{and} \quad f_M(x) := t_M(x) \cap F.$$

The acceptance degree $d_M(x)$ *of x is defined to be*

$$d_M(x) := \begin{cases} \#f_M(x)/\#t_M(x) & \text{if } \#t_M(x) > 0; \\ 0 & \text{if } \#t_M(x) = 0. \end{cases}$$

The degree-language *DL over M for some $\lambda \in [0, 1)$ is defined to be*

$$\mathrm{DL}(M, \lambda) := \{\, x \mid x \in \Sigma^*,\ d_M(x) > \lambda \,\}.$$

The pair (M, λ) is called a degree automaton *(DA).*

In [KPW93], an infinite sequence of regular languages L_n is exhibited such that each L_n can be accepted by a DA M_n with cutpoint $\frac{1}{2}$, i.e., $L_n = DL(M_n, \frac{1}{2})$, using just $4n + 7$ states, although an NFA requires 2^n states to accept L_n. (Note that the reduction in the number of states does not occur at the expense of the cutpoint, which remains constant, independent of n. Thus, the descriptional complexity of the DAs really does grow at a rate proportional to the number of states, $4n + 7$.)

We conclude this subsection with a brief discussion of *concurrent conciseness* (see [KW86] and [KPW93]), which involves trade-offs based on comparing three or more different descriptions. We explain this concept by using the example of DAs, NFAs and DFAs. As we have just seen, there is an exponential trade-off between DAs and NFAs, and we know from Section 2 that there is an exponential trade-off between NFAs and DFAs as well. In studying concurrent conciseness, we consider questions such as this. *Are there sequences of languages for which NFAs yield some preselected savings S_1 over DFAs (not exceeding exponential savings, of course), and for which DAs yield savings S_2 over NFAs, such that the combined savings resulting from S_1 and S_2 is exponential (the maximum possible savings between DAs and DFAs)?* This question can be answered in the affirmative for DAs, NFAs, and DFAs, as well as for some other collections of three or more different devices.

10.3 State complexity of operations on regular languages

Many results may be found in [Bir92a], [YZS94], [CSY00], [Yu01] and [CCSY01] concerning the effect that various language operations can have on the number of states required to recognize a regular language. These results may appear to fall outside the scope of research on the consequences of limiting resources. However, they have the potential to lead to some interesting questions about limited resources, as in the following example.

Communication networks usually consist of many finite-state machines that are interconnected in ways that can be modeled by operations on the languages characterized by these machines. The question that then arises is, would it be preferable to describe the entire network in terms of one large machine or in terms of many small machines? The former approach corresponds to actually performing the operations on the component languages or machines so as to combine them into a single complex entity. The latter approach seeks to decompose the entire network into many small components, and this may require a decision about whether to use a large number of simple operations, or to use a smaller number of more complex operations each of which may produce a larger explosion in the size of the state set than would the simpler operations. Thus, when dealing with interconnected finite-state machines, we may wish to understand the trade-offs that relate the number of machines used, their individual complexity, the number of operations connecting these machines, and the complexity of these operations, so that we can obtain a "balanced" characterization of the network which minimizes its descriptional complexity.

10.4 Non-machine devices: rewriting systems, regular expressions

In addition to the machine models that we have discussed, one can also investigate the descriptional complexity of other kinds of models, such as grammars or rewriting systems. Many studies of this sort have been undertaken, and we would exceed the scope of this survey were we to attempt to cite individual papers in this area. Instead, we will confine ourselves to mentioning some topics that have been covered in textbooks or monographs.

Grammars with controlled derivations are context-free grammars that are equipped with a control mechanism, such as one that requires that the derivation rules be applied in some prescribed order. The generative capacity of such systems is studied in [DP89], along with the relative succinctness of these systems.

In [DPR97] and [CVDKP94], systems of communicating context-free grammars, called *grammar systems*, are studied from several perspectives. In particular, the descriptional complexity of systems of communicating context-free grammars is compared with that of a single context-free grammar.

Just as quantum computing is influenced by the laws of physics, the field of DNA computing is guided by discoveries in the life sciences. For an introduction to this rapidly growing area, see [PRS98] and [Pău00].

Since it is often convenient to use regular expressions to describe regular languages, it is natural to investigate the trade-offs between regular expressions and DFAs or NFAs. Descriptional complexity measures for regular expressions were introduced in [EZ76]. Converting an NFA to a regular expression can result in an exponential increase in size. On the other hand, the increase in size required to convert a regular expression of size n to an NFA is bounded by $O(n(\log n)^2)$, and this upper bound is nearly tight [HSW01].

Some additional perspectives on the issue of limiting resources may be obtained by studying the effect of constraining the size of alphabets [Fro92], or of constraining regular expressions to be deterministic or unambiguous [BK93]. Similarly, one can investigate the impact of other constraints and extensions on regular expressions, such as limitations or extensions on the number or type of operators occurring in the expressions as, e.g., in [Abr87].

11 Conclusion and Outlook

In the previous sections, we presented many of the known results on the descriptional complexity of machines with limited resources. To maintain a well-defined focus, we concentrated on the scalable or gradual *trade-offs* among various resources that result when some of these resources are limited. Thus, we were primarily interested in questions such as: *How much does limiting one resource cost in terms of another resource; specifically, what are the upper and lower bounds on these costs?*

In some cases, we discussed results that do not precisely fit this scheme of trade-offs among resources, because we believe they have the potential of leading to results of this kind in the future.

We also presented several ideas in Section 9 concerning the potential relevance of descriptional complexity to practical issues. One of these was software reliability, but we believe that this is only one of several possible applications of descriptional complexity.

We would like to conclude with the following question: Where can descriptional complexity (or at least, descriptional complexity for machines with limited resources) go from here? Possibly in several directions, but one direction that we believe is important is the following.

Interesting as it might be to know that a resource such as the number of states in a probabilistic finite automaton can sometimes be reduced to some minimal value, this fact actually reveals little about the inherent descriptional complexity of a particular probabilistic finite automaton. To understand fully the descriptional complexity of an object, we ought to parameterize all of the resources needed to describe the object and then measure the trade-offs among them as we reduce some at the expense of others.

Thus, for example, if we consider probabilistic automata, we ought to measure all of the following resources: the number of states and of accepting states, and the representation *sizes* of the initial distribution, the probabilistic transition matrix, and the cutpoint, where *size* is some appropriate measure reflecting the *costs* of these representations.

How to define the *joint* or *overall* descriptional complexity of an object in terms of its individual parameters—for example, how to measure the overall descriptional complexity of a probabilistic finite automaton in terms of the descriptional complexity of its state set, its initial distribution, its probabilistic transition matrix, and its cutpoint—may be a cause for debate. Some may argue that one should measure the cost of *writing out* these parameters as strings or in some other appropriate form. Others may argue that these parameters should be weighted according to their importance in a particular setting. Yet whatever position one takes on this issue, it will always be important to understand how these various parameters relate to each other in the first place.

Because results on the descriptional complexity of machines with limited resources are widely scattered and are embedded in many different contexts, it would be nearly impossible to survey all of the results and references in this area. In addition, many of these results lie near the borderlines of this area, so that it is often unclear whether a particular result fits within the scope of our survey. Nonetheless, we have tried to present a reasonably comprehensive and balanced survey of this area.

As we said in the introduction, we hope that our effort will serve as an impetus for further surveys that cover results which we could not include here or which we inadvertently omitted. As a preliminary step in this direction, we are currently constructing a database of titles of papers on descriptional complexity which will be continuously updated. A link to this database can be found at

`www.psc.cs.uni-franfurt.de/english`

References

[Abr87] Karl R. Abrahamson. Succinct representation of regular sets using gotos and Boolean variables. *J. Comput. System Sci.*, 34(1):129–148, 1987.

[AF98] Andris Ambainis and Rūsiņš Freivalds. 1-way quantum finite automata: Strengths, weaknesses and generalizations. In *Annual Symposium on Foundations of Computer Science (Palo Alto, California, USA, 1998)*, pages 332–341. IEEE, Los Alamitos, CA, 1998.

[AKY99] Rajeev Alur, Sampath Kannan, and Mihalis Yannakakis. Communicating hierarchical state machines. In *Automata, languages and programming (Prague, 1999)*, volume 1644 of *LNCS*, pages 169–178. Springer-Verlag, Berlin, 1999.

[Amb96] Andris Ambainis. The complexity of probabilistic versus deterministic finite automata. In *Algorithms and computation (Osaka, 1996)*, volume 1178 of *LNCS*, pages 233–238. Springer-Verlag, Berlin, 1996.

[Ber80] Piotr Berman. A note on sweeping automata. In *Automata, languages and programming (Proc. Seventh Internat. Colloq., Noordwijkerhout, 1980)*, volume 85 of *LNCS*, pages 91–97. Springer-Verlag, Berlin, 1980.

[Bir92a] Jean-Camille Birget. Intersection and union of regular languages and state complexity. *Inform. Process. Lett.*, 43(4):185–190, 1992.

[Bir92b] Jean-Camille Birget. Positional simulation of two-way automata: Proof of a conjecture of R. Kannan and generalizations. *J. Comput. System Sci.*, 45(2):154–179, 1992.

[Bir93] Jean-Camille Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Math. Systems Theory*, 26(3):237–269, 1993.

[Bir96] Jean-Camille Birget. Two-way automata and length-preserving homomorphisms. *Math. Systems Theory*, 29(3):191–226, 1996.

[BK93] Anne Brüggemann-Klein. Regular expressions into finite automata. *Theoret. Comput. Sci.*, 120(2):197–213, 1993.

[BL77] Piotr Berman and Andrzej Lingas. On complexity of regular languages in terms of finite automata. Technical report, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1977.

[Blu01] Norbert Blum. On parsing LL-languages. *Theoret. Comput. Sci.*, 267(1-2):49–59, 2001.

[BMS01] Stefania Bandini, Giancarlo Mauri, and Roberto Serra. Cellular automata: From a theoretical parallel computational model to its application to complex systems. *Parallel Comput.*, 27(5):539–553, 2001.

[BN01] Eberhard Bertsch and Mark-Jan Nederhof. Size/lookahead tradeoff for LL(k)-grammars. *Inform. Process. Lett.*, 80(3):125–129, 2001.

[Bor92] Ingo Borchardt. Nichtrekursive Tradeoffs bei kontextfreien Grammatiken mit verschiedener konstanter Mehrdeutigkeit. Master's thesis, J.W. Goethe-Universität Frankfurt, 1992.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. Assoc. Comput. Mach.*, 30(2):323–342, 1983.

[CCSY01] Cezar Câmpeanu, Karel Culik, II, Kai Salomaa, and Sheng Yu. State complexity of basic operations on finite languages. In *4th International Workshop on Implementing Automata (Potsdam, 1999)*, volume 2214 of *LNCS*, pages 60–70. Springer-Verlag, Berlin, 2001.

[Chr86] Marek Chrobak. Finite automata and unary languages. *Theoret. Comput. Sci.*, 47(2):149–158, 1986.

[CSY00] Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. State complexity of regular languages: finite versus infinite. In *Finite versus infinite*, pages 53–73. Springer-Verlag, London, 2000.

[CVDKP94] Erzsébet Csuhaj-Varjú, Jürgen Dassow, Jozef Kelemen, and Gheorghe Păun. *Grammar systems.* Gordon and Breach Science Publishers, Yverdon, 1994.

[Dam97] David Damanik. Finite automata with restricted two-way motion. Master's thesis, J.W. Goethe-Universität Frankfurt, 1997.

[DM99] Marianne Delorme and Jacques Mazoyer, editors. *Cellular automata.* Kluwer Academic Publishers, Dordrecht, 1999.

[DP89] Jürgen Dassow and Gheorghe Păun. *Regulated rewriting in formal language theory.* Springer-Verlag, Berlin, 1989.

[DPR97] Jürgen Dassow, Gheorghe Păun, and Grzegorz Rozenberg. Grammar systems. In *Handbook of formal languages, Vol. 2*, pages 155–213. Springer-Verlag, Berlin, 1997.

[EZ76] Andrzej Ehrenfeucht and Paul Zeiger. Complexity measures for regular expressions. *J. Comput. System Sci.*, 12(2):134–146, 1976.

[Fre81] Rūsiņš Freivalds. Probabilistic two-way machines. In *Mathematical foundations of computer science, 1981 (Štrbské Pleso, 1981)*, volume 118 of *LNCS*, pages 33–45. Springer-Verlag, Berlin, 1981.

[Fre91] Rūsiņš Freivalds. Complexity of probabilistic versus deterministic automata. In *Baltic computer science*, volume 502 of *LNCS*, pages 565–613. Springer-Verlag, Berlin, 1991.

[Fro92] Matthias Frommknecht. Untere Schranken für die Größe von regulären Ausdrücken. Master's thesis, J.W. Goethe-Universität Frankfurt, 1992.

[Füs92] Hans-Martin Füssel. Komplexität einer frühen Fehlererkennung beim Parsen von deterministisch kontextfreien Sprachen. Master's thesis, J.W. Goethe-Universität Frankfurt, 1992.

[Gar95] Max Garzon. *Models of massive parallelism*. Springer-Verlag, Berlin, 1995.

[GHSU77] Matthew M. Geller, Harry B. Hunt, III, Thomas G. Szymanski, and Jeffrey D. Ullman. Economy of description of parsers, DPDA's, and PDA's. *Theoret. Comput. Sci.*, 4(2):143–153, 1977.

[GK74] Arthur Gill and Lawrence T. Kou. Multiple-entry finite automata. *J. Comput. System Sci.*, 9:1–19, 1974.

[GKW90] Jonathan Goldstine, Chandra M. R. Kintala, and Detlef Wotschke. On measuring nondeterminism in regular languages. *Inform. and Comput.*, 86(2):179–194, 1990.

[GLW92] Jonathan Goldstine, Hing Leung, and Detlef Wotschke. On the relation between ambiguity and nondeterminism in finite automata. *Inform. and Comput.*, 100(2):261–270, 1992.

[GLW97] Jonathan Goldstine, Hing Leung, and Detlef Wotschke. Measuring nondeterminism in pushdown automata. In *STACS 97 (Lübeck)*, volume 1200 of *LNCS*, pages 295–306. Springer-Verlag, Berlin, 1997.

[GMNP97] Jozef Gruska, Angelo Monti, Margherita Napoli, and Domenico Parente. Succinctness of descriptions of SBTA-languages. *Theoret. Comput. Sci.*, 179(1-2):251–271, 1997.

[GMP01] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. In *Mathematical foundations of computer science 2001 (Marianske Lazne)*, volume 2136 of *LNCS*, pages 398–407. Springer-Verlag, Berlin, 2001.

[GPW82] Jonathan Goldstine, John K. Price, and Detlef Wotschke. On reducing the number of states in a PDA. *Math. Systems Theory*, 15(4):315–321, 1982.

[GPW93] Jonathan Goldstine, John K. Price, and Detlef Wotschke. On reducing the number of stack symbols in a PDA. *Math. Systems Theory*, 26(4):313–326, 1993.

[Gru99] Jozef Gruska. *Quantum computing*. McGraw-Hill Publishing Company, Maidenhead, 1999.

[Gru00] Jozef Gruska. Descriptional complexity issues in quantum computing. *J. Autom. Lang. Comb.*, 5(3):191–218, 2000.

[Har80] Juris Hartmanis. On the succinctness of different representations of languages. *SIAM J. Comput.*, 9(1):114–120, 1980.

[Har83] Juris Hartmanis. On Gödel speed-up and succinctness of language representations. *Theoret. Comput. Sci.*, 26(3):335–342, 1983.

[Her97] Christian Herzog. Pushdown automata with bounded nondeterminism and bounded ambiguity. *Theoret. Comput. Sci.*, 181(1):141–157, 1997.

[Her99] Christian Herzog. *Die Rolle des Nichtdeterminismus in kontextfreien Sprachen*. PhD thesis, J.W. Goethe-Universität Frankfurt, 1999.

[HKK$^+$] Juraj Hromkovič, Juhani Karhumäki, Hartmut Klauck, Georg Schnitger, and Sebastian Seibert. Communication complexity method for measuring nondeterminism in finite automata. *Inform. and Comput.*, to appear.

[HS01a] Juraj Hromkovič and Georg Schnitger. On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. *Inform. and Comput.*, 169(2):284–296, 2001.

[HS01b] Juraj Hromkovič and Georg Schnitger. On the power of Las Vegas. II. Two-way finite automata. *Theoret. Comput. Sci.*, 262(1-2):1–24, 2001.

[HSW01] Juraj Hromkovič, Sebastian Seibert, and Thomas Wilke. Translating regular expressions into small ϵ-free nondeterministic finite automata. *J. Comput. System Sci.*, 62(4):565–588, 2001.

[HSY00] Markus Holzer, Kai Salomaa, and Sheng Yu. On the state complexity of k-entry deterministic finite automata. In *Second International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures (DCAGRS 2000)*. University of Western Ontario, London, Ontario, 2000.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Co., Reading, Mass., 1979.

[JR93] Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[Kan83] Ravi Kannan. Alternation and the power of nondeterminism. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (Boston, Massachusetts, 1983)*, pages 344–346. ACM, New York, 1983.

[Kap00] Martin Kappes. Descriptional complexity of deterministic finite automata with multiple initial states. *J. Autom. Lang. Comb.*, 5(3):269–278, 2000.

[KF77] Chandra M. R. Kintala and Patrick C. Fischer. Computations with a restricted number of nondeterministic steps (extended abstract). In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing (Boulder, Colo., 1977)*, pages 178–185. ACM, New York, 1977.

[KF80] Chandra M. R. Kintala and Patrick C. Fischer. Refining nondeterminism in relativized polynomial-time bounded computations. *SIAM J. Comput.*, 9(1):46–53, 1980.

[KF91] Jānis Kaņeps and Rūsiņš Freivalds. Running time to recognize nonregular languages by 2-way probabilistic automata. In *Automata, languages and programming (Madrid, 1991)*, volume 510 of *LNCS*, pages 174–185. Springer-Verlag, Berlin, 1991.

[Kin78] Chandra M. R. Kintala. Refining nondeterminism in context-free languages. *Math. Systems Theory*, 12(1):1–8, 1978.

[KKK00] Martin Kappes, Reinhard Klemm, and Chandra M. R. Kintala. Formal limits on determining reliabilities of component-based software. In *Eleventh International Symposium on Software Reliability Engineering (San Jose, 2000)*, pages 356–364. IEEE, Los Alamitos, CA, 2000.

[Kla98] Hartmut Klauck. Lower bounds for computation with limited nondeterminism. In *Thirteenth Annual IEEE Conference on Computational Complexity (Buffalo, NY, 1998)*, pages 141–152. IEEE, Los Alamitos, CA, 1998.

[Kla00] Hartmut Klauck. *Über beschränkte Interaktion in der Kommunikationskomplexität*. PhD thesis, J.W. Goethe-Universität Frankfurt, 2000.

[Kle96] Reinhard Klemm. *Systems of communicating finite state machines as a distributed alternative to finite state machines*. PhD thesis, The Pennsylvania State University, 1996.

[KPW93] Chandra M. R. Kintala, Kong-Yee Pun, and Detlef Wotschke. Concise representations of regular languages by degree and probabilistic finite automata. *Math. Systems Theory*, 26(4):379–395, 1993.

[Kut01] Martin Kutrib. Automata arrays and context-free languages. In *Where Mathematics, Computer Science, Linguistics and Biology Meet*, pages 139–148. Kluwer Academic Publishers, Dordrecht, 2001.

[KW80] Chandra M. R. Kintala and Detlef Wotschke. Amounts of nondeterminism in finite automata. *Acta Inform.*, 13(2):199–204, 1980.

[KW86] Chandra M. R. Kintala and Detlef Wotschke. Concurrent conciseness of degree, probabilistic, nondeterministic and deterministic finite automata (extended abstract). In *STACS 86 (Orsay, 1986)*, volume 210 of *LNCS*, pages 291–305. Springer-Verlag, Berlin, 1986.

[Lei81] Ernst Leiss. Succinct representation of regular languages by Boolean automata. *Theoret. Comput. Sci.*, 13(3):323–330, 1981.

[Lei85] Ernst Leiss. Succinct representation of regular languages by Boolean automata II. *Theoret. Comput. Sci.*, 38(1):133–136, 1985.

[Leu] Hing Leung. Tight lower bounds on the size of sweeping automata. *J. Comput. System Sci.*, to appear.

[Leu98a] Hing Leung. On finite automata with limited nondeterminism. *Acta Inform.*, 35(7):595–624, 1998.

[Leu98b] Hing Leung. Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. *SIAM J. Comput.*, 27(4):1073–1082, 1998.

[LW00] Hing Leung and Detlef Wotschke. On the size of parsers and LR(k)-grammars. *Theoret. Comput. Sci.*, 242(1-2):59–69, 2000.

[Lyu95] Michael R. Lyu, editor. *Handbook of software reliability engineering.* McGraw-Hill, New York, 1995.

[Mal01] Andreas Malcher. Descriptional complexity of cellular automata and decidability questions. In *Third International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures (DCAGRS 2001)*, pages 123–132. Otto-von-Guericke-Universität Magdeburg, Magdeburg, 2001.

[Mal02] Andreas Malcher. On one-way cellular automata with a fixed number of cells. To be submitted, 2002.

[Man73] Robert Mandl. Precise bounds associated with the subset construction on various classes of nondeterministic finite automata. Princeton Conference on System Sciences, 1973.

[Mau68] Hermann Maurer. The existence of context-free languages which are inherently ambiguous of any degree. Technical report, Department of Mathematics, University of Calgary, 1968.

[MF71] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *IEEE Twelfth Annual Symposium on Switching and Automata Theory*, pages 188–191. IEEE, 1971.

[Mic81] Silvio Micali. Two-way deterministic finite automata are exponentially more succinct than sweeping automata. *Inform. Process. Lett.*, 12(2):103–105, 1981.

[Moo71] Frank R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Trans. Comput.*, C-20:1211–1219, 1971.

[MP00a] Carlo Mereghetti and Giovanni Pighizzini. Optimal simulations between unary automata. *SIAM J. Comput.*, 30(6):1976–1992, 2000.

[MP00b] Massimiliano Milani and Giovanni Pighizzini. Tight bounds on the simulation of unary probabilistic automata by deterministic automata. In *Second International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures (DCAGRS 2000)*. University of Western Ontario, London, Ontario, 2000.

[Mus98] John D. Musa. *Software reliability engineered testing (Software development).* McGraw-Hill, New York, 1998.

[Nay99] Ashwin Nayak. Optimal lower bounds for quantum automata and random access codes. In *Annual Symposium on Foundations of Computer Science (New York, USA, 1999)*, pages 369–377. IEEE, Los Alamitos, CA, 1999.

[Oet92] Stefan Oetken. Beschreibungskomplexität bei regulären Sprachen über unärem Alphabet. Master's thesis, J.W. Goethe-Universität Frankfurt, 1992.

[Pău00] Gheorghe Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1):108–143, 2000.

[Paz71] Azaria Paz. *Introduction to probabilistic automata*. Academic Press, New York, 1971.

[PQ96] Terence J. Parr and Russell W. Quong. LL and LR translators need $k > 1$ lookahead. *ACM SIGPLAN Notices*, 31(2):27–34, 1996.

[PRS98] Gheorge Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA computing*. Springer-Verlag, Berlin, 1998.

[Rab63] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

[RI89] Bala Ravikumar and Oscar H. Ibarra. Relating the type of ambiguity of finite automata to the succinctness of their representation. *SIAM J. Comput.*, 18(6):1263–1282, 1989.

[RS59] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM J. Res. Develop.*, 3:114–125, 1959.

[Sch78] Erik M. Schmidt. *Succinctness of descriptions of context-free, regular and finite languages*. PhD thesis, Cornell University, Ithaca, NY, 1978.

[SH85] Richard E. Stearns and Harry B. Hunt, III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985.

[She59] John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Develop.*, 3:198–200, 1959.

[Sip80] Michael Sipser. Lower bounds on the size of sweeping automata. *J. Comput. System Sci.*, 21(2):195–202, 1980.

[SS77] Erik M. Schmidt and Thomas G. Szymanski. Succinctness of descriptions of unambiguous context-free languages. *SIAM J. Comput.*, 6(3):547–553, 1977.

[SS78] William J. Sakoda and Michael Sipser. Nondeterminism and the size of two-way finite automata. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 275–286. ACM, New York, 1978.

[SWY94] Kai Salomaa, Derick Wood, and Sheng Yu. Pumping and pushdown machines. *RAIRO Inform. Théor. Appl.*, 28(3-4):221–232, 1994.

[SY93] Kai Salomaa and Sheng Yu. Limited nondeterminism for pushdown automata. *Bulletin of the EATCS*, 50:186–193, 1993.

[SY94] Kai Salomaa and Sheng Yu. Measures of nondeterminism for pushdown automata. *J. Comput. System Sci.*, 49(2):362–374, 1994.

[SY97] Kai Salomaa and Sheng Yu. NFA to DFA transformation for finite languages over arbitrary alphabets. *J. Autom. Lang. Comb.*, 2(3):177–186, 1997.

[Val76] Leslie G. Valiant. A note on the succinctness of descriptions of deterministic languages. *Information and Control*, 32(2):139–145, 1976.

[Var89] Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Inform. Process. Lett.*, 30(5):261–264, 1989.

[VG79] Paulo A. S. Veloso and Arthur Gill. Some remarks on multiple-entry finite automata. *J. Comput. System Sci.*, 18(3):304–306, 1979.

[VS81] Dirk Vermeir and Walter J. Savitch. On the amount of nondeterminism in pushdown automata. *Fund. Inform. (4)*, 4(2):401–418, 1981.

[Wei97] Jörg R. Weimar. *Simulation with cellular automata*. Logos-Verlag, Berlin, 1997.

[Yu01] Sheng Yu. State complexity of regular languages. *J. Autom. Lang. Comb.*, 6(2):221–234, 2001.

[YZS94] Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theoret. Comput. Sci.*, 125(2):315–328, 1994.