Diss. ETH No. 18871

Complexity Classes of Finite Automata

A dissertation submitted to the
Swiss Federal Institute of Technology (ETH)
Zürich

for the degree of
Doctor of Sciences

presented by
Richard Královič
Mgr., Comenius University Bratislava
born on 7 September 1980
citizen of Slovakia

accepted on the recommendation of
Prof. Dr. Juraj Hromkovič, examiner
Prof. Dr. Georg Schnitger, co-examiner
Prof. Dr. Peter Widmayer, co-examiner

2010

Abstract

In this thesis, we present a complexity theory of finite automata in an analogous way to the well-known complexity theory of Turing machines. While most models of finite automata have equivalent computational power, there are large differences in their descriptional complexity. We provide a comprehensive overview of known results regarding the state complexity of finite automata and we use these results to build a hierarchy of the corresponding complexity classes.

The main motivation for the complexity theory of finite automata is to obtain a deeper understanding of the relationship between determinism, nondeterminism, and randomization. Indeed, while questions of such kind are usually very hard to answer in the scope of Turing machines, similar questions are often feasible to solve in simpler computational models such as finite automata.

At first, we focus on non-randomized models of finite automata. Here, we analyze determinism, nondeterminism, and self-verifying nondeterminism in connection with one-way, rotating, sweeping, and two-way automata. Apart from collecting previously known results into a unifying framework, we prove an exponential gap in state complexity between determinism and self-verifying nondeterminism for rotating and sweeping automata. To do so, we use the technique of *hardness propagation*, which allows us to prove separation results for state complexity classes in a systematic way.

The main focus of this thesis is the analysis of randomized models of finite automata. Here, many results can be obtained by relating the power of the randomized models to the non-randomized ones. Furthermore, several interesting facts about the relationship between different randomized models can be obtained by adapting corresponding results from communication complexity.

Our main contribution is the analysis of the relationship between randomized sweeping automata running in linear and in exponential expected time. While bounded-error sweeping automata are very powerful if their running time is not restricted (they can accept even non-regular languages), restricting their running time to be linear causes a significant decrease in their computational power. More precisely, we prove that even LasVegas sweeping automata (i. e., sweeping automata using the weakest form of randomization with zero probability of error) can be exponentially more succinct than bounded-error sweeping automata restricted to linear time. To prove our results, we adapt the hardness propagation lemmas used in the non-randomized case to randomized models.

To sum up, we show that it is possible to build a nontrivial complexity theory even for a very simple computational model such as finite automata. In such a way, we can learn more about the relationship between determinism, nondeterminism, and various models of randomization. Furthermore, there are several interesting open problems remaining.

Zusammenfassung

In dieser Arbeit stellen wir eine Komplexitätstheorie endlicher Automaten analog zu der bekannten Komplexitätstheorie der Turingmaschinen vor. Obwohl die meisten Modelle endlicher Automaten dieselbe Berechnungsstärke besitzen, gibt es grosse Unterschiede in ihrer Beschreibungskomplexität. Wir geben einen Überblick der bekannten Ergebnisse bezüglich der Zustandskomplexität endlicher Automaten und benutzen diese Resultate dann, um eine Hierarchie der zugehörigen Komplexitätsklassen aufzubauen.

Die Hauptmotivation für eine Komplexitätstheorie endlicher Automaten ist es, ein tiefergehendes Verständnis der Beziehungen zwischen Determinismus, Nichtdeterminismus und Randomisierung zu erlangen. Obwohl derartige Fragen im Bezug auf Turingmaschinen in der Regel sehr schwer zu beantworten sind, fällt dies in einfacheren Berechnungsmodellen, wie bei den endlichen Automaten, oftmals leichter.

Zunächst betrachten wir nicht-randomisierte Modelle endlicher Automaten. Hierzu untersuchen wir Determinismus, Nichtdeterminismus und selbstverifizierenden Nichtdeterminismus in Verbindung mit Einweg-, Rotating-, Sweeping- und Zweiweg-Automaten. Zusätzlich zur Einordnung bereits bekannter Ergebnisse in einen einheitlichen Rahmen beweisen wir eine exponentielle Lücke bezüglich der Zustandskomplexität zwischen Determinismus und selbstverifizierendem Nichtdeterminismus für Rotating- und Sweeping-Automaten. Um dies zu tun, bedienen wir uns der Technik der *Hardness Propagation*, die es uns ermöglicht, Separationsergebnisse systematisch zu beweisen.

Der Schwerpunkt dieser Arbeit liegt auf der Analyse randomisierter Modelle endlicher Automaten. Viele Ergebnisse erhalten wir dadurch, dass wir die Mächtigkeit randomisierter Modelle mit der nicht-randomisierter Modelle in Beziehung setzen. Desweiteren können viele interessante Tatsachen über die Beziehungen zwischen verschiedenen randomisierten Modellen aus Ergebnissen der Kommunikationskomplexität abgeleitet werden.

Der Hauptbeitrag dieser Arbeit ist die Analyse der Beziehungen zwischen randomisierten Sweeping-Automaten mit linearer Laufzeit und solchen mit exponentieller Laufzeit. Obwohl Sweeping-Automaten mit beschränktem Fehler sehr mächtig sind, wenn ihre Laufzeit nicht begrenzt ist (in diesem Fall können sie sogar nicht-reguläre Sprachen akzeptieren), wird ihre Berechnungsstärke stark verringert, wenn sie nur eine lineare Laufzeit besitzen dürfen. Wir beweisen, dass sogar LasVegas-Sweeping-Automaten (also Sweeping-Automaten, die nur die schwächste Form der Randomi-

sierung mit einer Fehlerwahrscheinlichkeit von Null nutzen) exponentiell kleiner sind als Sweeping-Automaten mit beschränktem Fehler, wenn deren Laufzeit linear begrenzt ist. Um dies zu beweisen, wenden wir Lemmata, die wir schon zur Hardness Propagation im nicht-randomisierten Fall benutzt haben, auf randomisierte Modelle an.

Zusammengefasst zeigen wir, dass es möglich ist, selbst für einfache Berechnungsmodelle wie endliche Automaten eine nicht-triviale Komplexitätstheorie zu konstruieren. Hierdurch können wir vieles über die Beziehungen zwischen Determinismus, Nichtdeterminismus und verschiedenen Modellen der Randomisierung lernen. Desweiteren wirft dies viele interessante bislang unbeantwortete Fragen auf.

Contents

List of Figures

Chapter 1

Introduction

A considerable part of the computer science is devoted to the analysis of the power of different computational models. Since the advent of computer science, many different abstract models of machines have been proposed and analyzed, many of them differing in their computational power. On one side of this spectrum, there are the very powerful Turing machines, together with other equivalently powerful models, such as register machines, phrase grammars, lambda calculus, and two-counter automata. Naturally, it has been asked how the computational power changes with introducing certain restrictions to the computational model. In this direction, models such as linear-bounded automata (equivalent to context-sensitive grammars), push-down automata (equivalent to context-free grammars), and finite automata (equivalent to regular grammars) have been analyzed and shown to be different – this result is widely known as the the Chomsky hierarchy.

Computational models are often viewed as some kind of machines that are given some input instance (sometimes called *input word*) and are supposed to *decide* if this instance belongs to the considered problem (also called *language*). It is possible to use also different frameworks, e. g. the more general model where the machine is able to give arbitrary output. Nevertheless, restricting to the *decision problems* usually does not cause significant loss of generality.

Most computational models can be defined in several variants. For example, any model of automata or Turing machines can be considered in the deterministic or in the nondeterministic setting. In the former case, the computation of the automaton is uniquely defined for each input word. In the latter case, however, the automaton is allowed to make nondeterministic guesses during the computation; the input word is accepted if there exists at least one computation that accepts it. Hence, the nondeterministic setting can be viewed as a scenario where the automaton is given some (maybe bogus) "proof" that the input instance belongs to the recognized problem, and this proof has to be verified.

Since every deterministic automaton can be viewed as a special case of a nondeterministic automaton of the same type, it is obvious that deterministic computations

cannot be more powerful than nondeterministic ones. A natural question is whether there is a difference in computation power of determinism and nondeterminism. For several models, the answer is known. For example, there is no difference in the computational power of determinism and nondeterminism in Turing machines and in finite automata, but deterministic push-down automata are strictly weaker than nondeterministic ones [GG66]. For linear bounded automata, however, the relationship between determinism and nondeterminism is a long-standing open problem (see e. g. [Mon81, HH73]).

Apart from determinism and nondeterminism, several other variants of computation, such as randomized, alternating, and quantum computational models, have been widely studied. Randomized machines can use some external source of randomness to make their decisions. Hence, there are many possible computations (as in nondeterminism), but probability of any computation is uniquely defined by the behavior of the automaton. There are, however, several possible ways how to define the language accepted by a randomized machine. In the most general setting, the language recognized by the machine consists of all words that are accepted with probability at least p for some fixed constant $0 \leq p \leq 1$; every word that is accepted with probability less than p is said to be rejected by the machine. This model is called as *a randomized computation with non-isolated cut-point* p (see e. g. [Rab63]). Such model of randomized computations can be very powerful, but it does not guarantee an efficient way of computing the result: Since the probability of acceptance of some word in the recognized language and the probability of acceptance of some word not in the language can be arbitrarily close, there is no efficient way how to obtain an answer that is correct with reasonable high probability.

It is not difficult to see that the value of p is not very important: Indeed, for most computation models it holds that any machine M with cut-point p can be easily transformed into machine M' with cut-point p', for any $0 < p < 1$ and $0 < p' < 1$ (see, e. g., [MPP01, Paz71]). For example, if $p' < p$, then M' will immediately reject the input with probability $\frac{p-p'}{p}$, otherwise (i. e., with probability $1 - \frac{p-p'}{p}$) machine M' simulates M. Hence, M' accepts any word in the language recognized by M with probability at least

$$\left(1 - \frac{p - p'}{p}\right) p = p',$$

and, by similar argumentation, accepts any word not in the language with probability at most p'. For this reason, randomized computations are sometimes defined with cut-point $1/2$ only and are called as *randomized computations with unbounded error*.[1]

The drawback of computations with non-isolated cut-points is addressed in the model of *bounded-error randomized computations*. Here, it is required that every

[1] Sometimes the randomized machines are defined in such a way that only rational probabilities can be generated, or, (equivalently) only fair-coin flips can be used. In this case, the equivalence between arbitrary cut-point and cut-point $1/2$ does not always hold (as claimed in [Rab63] for the case of finite automata), or is nontrivial to prove (as shown in [Wat99] for the case of space-bounded randomized Turing machines).

word is either accepted with probability at least $p + \varepsilon$ or at most $p - \varepsilon$, for $0 \leq p \leq 1$ and some fixed $\varepsilon > 0$; this model of randomized computation is also called as the *model with isolated cut-point p* (introduced in [Rab63]).

Bounded-error randomized machines can still err in both sides (i. e., accepting some word not in the recognized language or rejecting some word in the language with with non-zero probability). We have, however, a fixed constant ε that isolates the probabilities of these two cases. Hence, we can efficiently compute the correct answer with high probability by using the method of *amplification* (see e. g. [Hro05]). In particular, we can run several independent computations on the same input word and let the machine "vote" if the word should be accepted or rejected, i. e., we just take the result that occurred more often. Furthermore, the probability of error decreases exponentially in the number of computations done. Hence, the amplification technique allows us to compute the correct answer with arbitrarily high probability, just by repeating the computation a constant number of times; this constant depends on the parameter ε of the bounded-error automaton. For this reason, the exact value of ε is usually[2] not interesting, and bounded-error automata are sometimes defined with $p = 1/2$ and $\varepsilon = 1/6$, i. e., they are required to accept every word either with probability at least $2/3$ or at most $1/3$.

Since bounded-error machines can make errors on both sides, it is not trivial to compare them with nondeterminism. Indeed, their relationship to nondeterministic machines is different for various models. E. g., in case of Turing machines, nondeterminism and bounded-error randomization have the same computational power, but bounded-error two-way finite automata are more powerful than nondeterministic two-way finite automata.

Another widely considered model of randomized computations is model with bounded one-sided error, usually called *one-sided error Monte-Carlo randomization*. This model is, in fact, a restriction of nondeterminism: A Monte-Carlo machine with one-sided error is required to accept every word either with probability at least ε (for some fixed $\varepsilon > 0$), or with probability zero. Hence, any one-sided error Monte-Carlo machine can be simulated by a corresponding nondeterministic machine in a straightforward way – every randomized decision (that occurs with non-zero probability) is replaced by a nondeterministic one.

Every time a Monte-Carlo machine with bounded one-sided error accepts the input word, it is guaranteed that this word is in the recognized language. Since the probability of error on any word that is in the recognized language is bounded by a fixed constant $1 - \varepsilon$, it is possible to use the amplification technique to make the error probability arbitrary small. This indicates that, similarly to the case of two-sided bounded-error machines, the exact value of ε is usually not important; one-sided error Monte-Carlo machines are sometimes defined with fixed value $\varepsilon := 1/2$.

[2] There are computation models (e. g. one-way finite automata) that are not capable of repeating the computation. Here, the amplification technique cannot be used in the straightforward way. Nevertheless, it still may be possible to apply the amplification technique by parallel simulation of the computations.

It is possible to restrict the model of randomized computations even more. The weakest model of randomized computations, called *Las Vegas* (a name introduced in [Bab79]) or *zero probability of error*, imposes very strict constraints on the randomized machine: The answer provided by the machine has to be always correct. On the other hand, the machine is allowed to not provide any answer (e. g. by outputting special "*I do not know*" message) with probability bounded by a fixed constant $\varepsilon < 1$. Again, the model is sometimes defined for fixed $\varepsilon = 1/2$, as the amplification technique allows to efficiently decrease the probability of error below $1/2$ for most computation models.

Essentially, any computation model can be viewed as some computational device that is using certain resources. E. g., for Turing machines, we can measure the number of tape cells used during the computation or the total number of computation steps performed. In this way, we can define the space and time complexity of Turing machines. Another possible complexity measure is the amount of information needed to describe the computational device, what leads to the definition of the *descriptional complexity*. Analogously, these complexity measures can be applied to other models of computation as well.

In general, we can ask which problems are solvable by certain computational model within some restrictions on its resources. In this way, we can define several different *complexity classes* for one given computational model. The study of the relationship between different complexity classes is of crucial importance for computer science, and the most prominent open problems belong to this area. For example, the famous P *vs.* NP problem asks about the relationship of deterministic and nondeterministic Turing machines restricted to polynomial running time. This problem has far reaching consequences: Problems in P, i. e., those problems that are solvable by deterministic Turing machines running in polynomial time, are considered *efficiently solvable*, but many real-world problems are known to be NP-*hard* (i. e., problems that any problem in NP can be reduced to in polynomial deterministic time). Hence, disproving P = NP would show that these hard problems are not efficiently solvable by deterministic algorithms.

The relationship between randomized and non-randomized classes of Turing machines is probably even more relevant. For example, the class BPP, defined as the class of problems solvable by bounded-error Turing machines restricted to polynomial running time, is considered to be the largest class of efficiently solvable problems. Thus, proving that BPP = NP would show that all problems in NP are efficiently solvable; on the other hand, proof of BPP $\neq$ NP would suggest that there is no efficient solution for any NP-hard problem.

Another well known open problem concerns the power of LasVegas Turing machines. An interesting open problem (introduced in [Gil77]), is the relationship between the class ZPP, i. e., the class of problems solvable by LasVegas Turing machines with polynomial running time, and the class P. While it is known that space complexity classes of LasVegas Turing machines are equal to the corresponding classes of nondeterministic Turing machines (as proven in [MS99]), there are no indices suggesting that this is the case for restricted running time. Proving that ZPP = P would

show that randomized computations with zero probability of error do not add any extra power for Turing machines with polynomial running time.

Unfortunately, proving these relationship between complexity classes of Turing machines seems to be extremely hard. Hence, to obtain better understanding of the relative power of determinism, nondeterminism, and randomization, it is necessary to focus on simpler computational models, where the problems are much more tractable. In our work, we focus on various variants of finite automata.

Since for most variants of finite automata superlinear time does not add any power, our main attention is on the measure of *state complexity*, which is closely related to the descriptional complexity. Furthermore, the state complexity of finite automata can be viewed as an analogy to the space complexity of Turing machines.

In Section 1.1 and Section 1.2, we formally define several variants of finite automata, as well as the complexity classes induced by their state complexity. Since we will not deal with randomized models and time complexity classes of finite automata until Chapter 3, we postpone their definitions until then. In Section 1.3, we present a short summary of previous work related to the complexity analysis of finite automata.

Chapter 2 is devoted to the study of the relationship between various non-randomized complexity classes of finite automata, i. e., classes related to determinism and nondeterminism. Up to know, most of the work has been focused on investigating the relationship between two-way deterministic and nondeterministic finite automata. Other variants of finite automata have been examined only as intermediate steps. Hence, our study of these automata has been rather fragmentary. In Chapter 2, we take the time to have focus on the relationship between the numerous variants of finite automata itself. We introduce a list of language operators and study the closure properties of the analyzed complexity classes with respect to these operators. We introduce a technique of *hardness propagation*, a general framework for proving lower bounds on the size of finite automata. This technique allows us to use the language operators to build hard languages for more powerful automata classes out of a simple, minimally hard, 'core' family. This way, we can find witnesses for previously known complexity class separations in a systematic way. Moreover, we use these technique to show a hierarchy between the analyzed complexity classes of finite automata. For most of the classes, our results yield a complete characterization of their relationship to other classes.

We deal with randomized models of finite automata in Chapter 3. After introducing the randomized models and their state complexity classes, we relate them to the classes discussed in Chapter 2. This allows us to provide a complete characterization of several important randomized classes. Among others, our results imply that LasVegas automata can be exponentially more succinct than deterministic ones for certain models of restricted two-way head motion.

This exponential gap between determinism and LasVegas randomization, however, heavily depends on the exponential running time of the LasVegas machines. While long (more precisely, superlinear) running time makes no difference for non-randomized models of finite automata, we show later that this is not the case for randomized models.

Our results for finite automata corresponds well with the fact that LasVegas randomization has equal power as nondeterminism for space-bounded computations of Turing Machines, as shown in [MS99]. On the other hand, an interesting question is whether the exponential running time is necessary to gain this power, i. e., what is the relationship between deterministic and LasVegas automata with polynomial running time. This question is still open and can be viewed as an analogy of the ZPP vs. P problem.

In Chapter 3, however, we provide a partial answer to this problem. More precisely, we focus on the randomized finite automata restricted to *linear* running time. We show that this restriction may enforce exponential blow-up in state complexity of the automata, i. e., we separate the complexity classes corresponding to linear-time and exponential-time LasVegas finite automata. More generally, we prove that the restriction on running time cannot be compensated by a more powerful model of randomization. In particular, we exhibit problems that are easily solvable by LasVegas automata with exponential running time, but any (even the most powerful bounded-error) finite automata running in linear time must be exponentially larger.

1.1 Models of Finite Automata

In this section, we introduce the models of finite automata used in this thesis. At first, we introduce some general notation used. Let Σ be an alphabet, i. e., any finite set of symbols. By Σ^* we denote the set of all finite strings over Σ. If $z \in \Sigma^*$, then $|z|$, z_t, z^t, and z^R are its length, t-th symbol (if $1 \leq t \leq |z|$), t-fold concatenation with itself (if $t \geq 0$), and reverse.

A *language* L over Σ is any subset of Σ^*, the *complement* of L is $\overline{L} := \Sigma^* - L$, the *reverse* of L is $L^R := \{w^R \mid w \in L\}$. If $w \in L$, we say that w is a *positive instance* of L, otherwise w is a *negative instance* of L. An automaton *recognizes* (or *solves* or *accepts*) a language iff it accepts exactly the strings of that language.

1.1.1 Deterministic Two-Way Finite Automata

At first, we introduce the deterministic models of finite automata. The most general one, introduced in [RS59], is the *two-way* finite automaton. Informally, it is a deterministic machine with finite-state control and single head operating operating on the input tape, where the input word, surrounded by end-markers, is written. The head can move in both directions, but the machine is not allowed to modify the content of the input tape.

There are several possible ways to define when the input word is accepted by the automaton. The model is, however, quite robust, i. e., different ways of accepting usually do not change the power of the model. To be consistent with other models of finite automata, we require the automaton to accept by entering a special state (called the *accept state*) after reading the right end-marker of the input.

More formally, we say that a *deterministic two-way finite automaton* (shortly denoted as 2DFA) over an alphabet Σ and a set of states Q is any triple $M = (q_s, \delta, q_a)$ of a *start* state $q_s \in Q$, an *accept* state $q_a \in Q$, and a *transition function* δ which partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to $Q \times \{-1, 0, +1\}$, for some *end-markers* $\vdash, \dashv \notin \Sigma$. An input $z \in \Sigma^*$ is presented to M surrounded by the end-markers, as $\vdash z \dashv$. The computation starts at q_s and on $\vdash$. The next state is always equal to the first component of $\delta(q, a)$, where q is the current state and a is the symbol located under the head. The second component of $\delta(q, a)$ determines the head motion. Value -1 means that the new head position is the one left adjacent to the old one, value 1 means that the new position is right adjacent to the old one, and value 0 specifies that the head is not moved in the current computation step. The automaton is neither allowed to move the head to the left while reading $\vdash$, nor allowed to move the head to the right while reading $\dashv$. The only exception is a transition from $\dashv$ to the accept state q_a and moving the head to the right; in this case, and only in this case, the automaton *accepts* the input word z. Note that the computation can either loop, or hang, or fall off $\dashv$ into q_a (and *accept* z).

1.1.2 Deterministic Sweeping Finite Automata

It turns out that the state complexity of two-way finite automata is very hard to analyse. Indeed, the relationship between determinism and nondeterminism, a problem introduced in [SS78], is a long standing open problem. Hence, a model with restricted possibility of head movement, called the *deterministic sweeping finite automaton*, was proposed in [Sip80b]. Informally, the deterministic sweeping finite automaton is a special case of the deterministic two-way finite automaton that can change the direction of the head motion at the endmarkers only.

The formal definition of the *sweeping deterministic finite automaton* (shortly denoted as SDFA) is analogous to the definition of a two-way finite automaton. The *transition function* δ, however, partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to Q. The computation on an input $z \in \Sigma^*$, presented to M as $\vdash z \dashv$, starts at the *start* state q_s and on $\vdash$. The next state is always derived from δ and the current state and symbol. The next position is always the adjacent one in the direction of motion; except when the current symbol is $\vdash$ or when the current symbol is $\dashv$ and the next state is not q_a, in which cases the next position is the adjacent one towards the other end-marker. Again, the computation can either loop, or hang, or fall off $\dashv$ into q_a. In the last case we call it *accepting* and say that M *accepts* z.

More generally, for any input string $z \in \Sigma^*$ and state p, the *left computation of* M *from* p *on* z is the unique sequence

$$\text{LCOMP}_{M,p}(z) := (q_t)_{1 \le t \le m},$$

where $q_1 := p$; every next state is $q_{t+1} := \delta(q_t, z_t)$, provided that $t \le |z|$ and the value of δ is defined; and m is the first t for which this provision fails. If $m = |z| + 1$, we say that the computation *results* z *in* q_m; otherwise, $1 \le m \le |z|$ and the computation

hangs at q_m and *results in* $\perp$. The *right computation of* M *from* p *on* z is denoted by $\text{RCOMP}_{M,p}(z)$ and defined symmetrically, i. e., $\text{RCOMP}_{M,p}(z) = \text{LCOMP}_{M,p}(z^{\text{R}})$.

The *traversals of* M *on* z are the members of the unique sequence $(c_t)_{1 \le t < m}$ where $c_1 := \text{LCOMP}_{M,p_1}(z)$ for $p_1 := \delta(q_s, \vdash)$; every next traversal c_{t+1} is either $\text{RCOMP}_{M,p_{t+1}}(z)$ if t is odd and c_t results in a state q_t such that $\delta(q_t, \dashv) = p_{t+1} \neq q_a$, or $\text{LCOMP}_{M,p_{t+1}}(z)$ if t is even and c_t results in a state q_t such that $\delta(q_t, \vdash) = p_{t+1}$; and m is either the first t for which c_t cannot be defined or ∞ if c_t exists for all t. Then, the *computation of* M *on* z, denoted by $\text{COMP}_M(z)$, is the concatenation of $(q_s), c_1, c_2, \ldots$ and possibly also (q_a) if m is finite and even and c_{m-1} results in a state q_{m-1} such that $\delta(q_{m-1}, \dashv) = q_a$. An example of a computation of SDFA is depicted in Figure 1.1a.

1.1.3 Deterministic Rotating Finite Automata

When dealing with sweeping automata, it is usually necessary to argue about left and right computations separately, although arguments for both directions are analogous. Hence, allowing the left computations only makes it possible to express core ideas of several proofs in a simpler way. This leads to the definition of *deterministic rotating finite automata*. Historically, this model was introduced (in slightly different form) in [SS78] under the name *series finite automata*, and later was generalized into the sweeping finite automata. In fact, it is usually much easier to argue about the rotating automata, and it is possible to generalize many of the arguments to sweeping automata in a straightforward way.

The definition of a *rotating deterministic finite automaton* (shortly denoted as RDFA) is analogical to the definition of SDFA. The only difference is in the definition of the head motion. The next position of the head of a RDFA is always the adjacent one to the right, except when the current symbol is $\dashv$ and the next state is not q_a, in which case it is the one to the right of $\vdash$.

The formal definition of the computation of a RDFA M on z is similar to the definition for a SDFA, too. The *traversals of* M *on* z are always defined in terms of LCOMP, i. e., as the members of the unique sequence $(c_t)_{1 \le t < m}$ where $c_1 := \text{LCOMP}_{M,p_1}(z)$ for $p_1 := \delta(q_s, \vdash)$; every next traversal c_{t+1} is defined as $\text{LCOMP}_{M,p_{t+1}}(z)$ if c_t results in a state q_t such that $\delta(q_t, \dashv) = p_{t+1}$; and m is either the first t for which c_t cannot be defined or ∞ if c_t exists for all t. The *computation of* M *on* z, denoted by $\text{COMP}_M(z)$, is the concatenation of $(q_s), c_1, c_2, \ldots$ and possibly also (q_a) if m is finite and c_{m-1} results in a state q_{m-1} such that $\delta(q_{m-1}, \dashv) = q_a$. An example of a computation of RDFA is depicted in Figure 1.1b.

1.1.4 Deterministic One-Way Finite Automata

The well-known model of computation called one-way deterministic automata, introduced in [Huf54, Mea55, Moo56], can be viewed as a special case of RDFAs. We say that M is an *one-way deterministic finite automaton* (1DFA for short) if it is a RDFA that halts immediately after reading $\dashv$: The value of δ on any state q and on $\dashv$ is

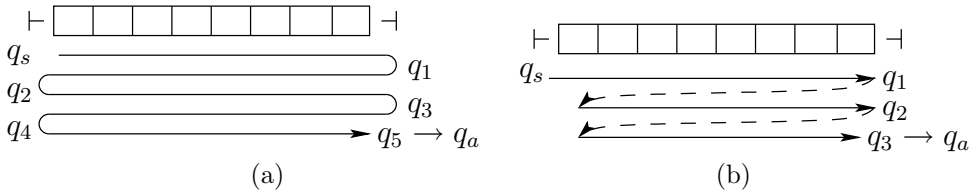

Figure 1.1: Computation of a (a) sweeping and (b) rotating automaton.

always either q_a or undefined. If it is q_a, we say q is a *final* state; if it is undefined, we say q is *nonfinal*. The state $\delta(q_s, \vdash)$, if defined, is called *initial*. An example of a one-way computation is shown in Figure 1.2.

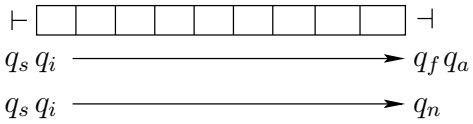

Figure 1.2: The computation of a one-way automaton. The computation starts in the start state q_s on $\vdash$ and the first symbol of the input word is read when the automaton is in the initial state q_i. After reading the last symbol of the input word, the automaton either reaches a final state q_f and enters the accept state q_a afterwards, or it reaches a non-final state q_n and hangs.

Note that this definition is equivalent to the other widely-used definition of one-way automata: Here, the one-way automaton M is a five-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is the set of states of M, Σ is the input alphabet, δ is the transition function, q_0 is the initial state and F is the set of final states. The input word z is presented to M without end-markers, the computation starts in q_0 on the first symbol z. An input word z is accepted by M if and only if some state from F is reached after reading the last symbol of z.

1.1.5 Nondeterministic Finite Automata

For every model of deterministic finite automata introduced so far (i. e., 1DFAs, RDFAs, SDFAs, and 2DFAs), it is possible to define its nondeterministic counterpart. In this way, we obtain *nondeterministic one-way finite automata* (1NFAs for short), *nondeterministic rotating finite automata* (RNFAs for short), *nondeterministic sweeping finite automata* (SNFAs for short), and *nondeterministic two-way finite automata* (2NFAs for short). More precisely, a nondeterministic automaton M is allowed to make more than one move at each step. In this case, every input word z admits a set of compu-

tations of M. The input word z is accepted if and only if at least one of the possible computations is accepting.

More formally, this means that δ partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to the set of *all non-empty subsets* of $Q \times \{-1, 0, 1\}$ in the case of two way automata, or to the set of all non-empty subsets of Q in the case of sweeping, rotating, and one-way automata. Then, $\text{LCOMP}_{M,p}(z)$, $\text{RCOMP}_{M,p}(z)$, and $\text{COMP}_M(z)$ of a nondeterministic sweeping automaton M are actually *sets of computations*, and M accepts z if and only if $\text{COMP}_M(z)$ contains at least one accepting computation. Analogous definition is used for rotating and one-way nondeterministic automata.

All models of finite automata introduced so far can be categorized according to two independent properties: The capability of the head motion (two-way, sweeping, rotating, one-way) and the mode of computation (nondeterministic, deterministic). Each combination of these two properties yields a reasonable model of automata.

Easily, each deterministic machine can be viewed also as a nondeterministic machine of the same type. Furthermore, one-way automata are just a special case of rotating automata. Any rotating automaton with k states can be easily transformed into a corresponding two-way automaton with at most $2k$ states by simulating each "jump" to the beginning of the input word by one right-to-left traversal. Easily, any sweeping automaton with k states can be simulated by a corresponding two-way automaton with at most $2k$ states, i. e., without asymptotic blowup in the state complexity. Hence, we can arrange all models introduced so far in a "map" as shown in Figure 1.3, where each model is at least as powerful as its left and bottom neighbor.[3]

N	1NFA		RNFA		SNFA	2NFA
∪|						
D	1DFA		RDFA		SDFA	2DFA
	1	⊆	R	⊆	S	⊆ 2

Figure 1.3: Models of finite automata.

So far, we have not introduced any models of randomized automata. Since we do not deal with randomization until Chapter 3, we postpone their definition until then.

1.2 Complexity Classes

It is a well known result that all models of finite automata introduced in the previous subsection accept exactly the class of regular languages (equivalent computational power of 2NFAs and 1DFAs, proven in [Kol72], implies this fact). Nevertheless, different

[3]The map provided in Figure 1.3 is rather informal, since we have not defined what it means that some model is "at least as powerful" as another one yet. We specify this, however, more precisely in Section1.2 by introducing complexity classes of finite automata.

models of automata require different number of states to accept the same language. Hence, we focus on the analysis of the state complexity of finite automata, i. e., on the complexity measure defined by the number of states.

In this section, we introduce complexity classes induced by the state complexity of finite automata. A state of a finite automaton represents certain information that is stored in the memory of the automaton. Hence, the state complexity of finite automata can be viewed as an analogy of the space complexity of Turing machines.

Because the state complexity of finite automata is a static complexity measure, the state complexity of any fixed language is a fixed constant. Hence, it does not make any sense to analyze the asymptotic complexity of a single language. To overcome this problem, we consider *families of languages* instead of single languages, in a similar way as in [SS78, KKM07, KKM08].

Let $M_1, M_2, \ldots$ be finite automata and let $L_1, L_2, \ldots$ be languages. A *family of automata* $\mathcal{M} = (M_n)_{n \geq 1}$ solves a *family of languages* $\mathcal{L} = (L_n)_{n \geq 1}$ iff, for all n, M_n solves L_n. The automata of $\mathcal{M}$ are "*small*" iff, for some polynomial p and all n, M_n has at most $p(n)$ states.

The state-complexity class 1D consists of every family of languages that can be solved by a family of small 1DFAs. The classes RD, SD, 2D, 1N, RN, SN, 2N are defined similarly, by replacing 1DFAs with RDFAs, SDFAs, 2DFAs, 1NFAs, RNFAs, SNFAs, 2NFAs. The naming convention is from [SS78]; in general, class $\mathfrak{C}$ consists of every family of languages that can be solved by a family of small $\mathfrak{C}$FAs.

Any language operator can be generalized to work on language families in a straightforward way, the operator is just separately applied to all languages in the family. In particular, we say that a *complement* of a family of languages $\mathcal{L} = (L_n)_{n \geq 1}$ is defined as $\overline{\mathcal{L}} := (\overline{L_n})_{n \geq 1}$, and a *reverse* of $\mathcal{L}$ is defined as $\mathcal{L}^{\mathsf{R}} := (L_n^{\mathsf{R}})_{n \geq 1}$.

If $\mathfrak{C}$ is a class, then co-$\mathfrak{C}$ consists of all families of languages whose complement is in $\mathfrak{C}$, and re-$\mathfrak{C}$ consists of all families of languages whose reverse is in $\mathfrak{C}$.

The analysis of the relationship between various complexity classes can be often simplified by the following straightforward observation:

Observation 1.1. *Let $\mathfrak{C}_1$, $\mathfrak{C}_2$ be any classes of language families. It holds that:*

1. re-(re-$\mathfrak{C}_1$) = $\mathfrak{C}_1$
2. co-(co-$\mathfrak{C}_1$) = $\mathfrak{C}_1$
3. *If* $\mathfrak{C}_1 \subseteq \mathfrak{C}_2$, *then* re-$\mathfrak{C}_1 \subseteq$ re-$\mathfrak{C}_2$.
4. *If* $\mathfrak{C}_1 \subseteq \mathfrak{C}_2$, *then* co-$\mathfrak{C}_1 \subseteq$ co-$\mathfrak{C}_2$.
5. *If* $\mathfrak{C}_1 \not\supseteq \mathfrak{C}_2$, *then* re-$\mathfrak{C}_1 \not\supseteq$ re-$\mathfrak{C}_2$.
6. *If* $\mathfrak{C}_1 \not\supseteq \mathfrak{C}_2$, *then* co-$\mathfrak{C}_1 \not\supseteq$ co-$\mathfrak{C}_2$.
7. *If* $\mathfrak{C}_1 \subseteq \mathfrak{C}_2 \not\supseteq \mathfrak{C}_3 \subseteq \mathfrak{C}_4$, *then* $\mathfrak{C}_1 \not\supseteq \mathfrak{C}_4$.

1.2.1 Self-Verifying Complexity Classes

Due to close connection to randomized automata models, we also consider the "*self-verifying*" classes $1\Delta := 1\text{N} \cap \text{co-}1\text{N}$, $\text{R}\Delta := \text{RN} \cap \text{co-R}\Delta$, $\text{S}\Delta := \text{SN} \cap \text{co-SN}$, and $2\Delta := 2\text{N} \cap \text{co-}2\text{N}$. In general, the naming convention is that $\text{X}\Delta := \text{XN} \cap \text{co-XN}$, for any X.

The notion of the *self-verifying nondeterminism* has been introduced in [HS96] and [ĎHRS97]. Here, however, it was considered as a separate model of finite automata: The *self-verifying automaton* is able to make nondeterministic choices, and is able to give three types of answers: "*yes*", "*no*", and "*I do not know*". Whenever the answer is "*yes*" or "*no*", it has to be correct, i. e., any possible computation is required to provide either correct answer or the "*I do not know*" answer. Furthermore, for each input there must exist at least one computation that gives the correct answer. In other words, self-verifying nondeterminism differs from the ordinary nondeterminism by the semantics of the negative answer: Whereas in the ordinary nondeterminism, a negative answer does not provide any information about the input word z, the answer *no* of a self-verifying machine indicates a proof that z is a negative instance of the solved problem.

Following our naming convention, we call the one-way, rotating, sweeping, and two-way self-verifying automaton as 1ΔFA, RΔFA, SΔFA, and 2ΔFA, respectively. More formally, the self-verifying automata are just nondeterministic machines augmented by a special *reject* state q_r, whose behavior is analogous to the behavior of the accept state. Then, any computation can either either loop, or hang, or fall off $\dashv$ into q_a or q_r. In this last case we call it *accepting* (*rejecting*) and say that M *accepts* z (*rejects* z).

It is not difficult to see that this definition is equivalent to the definition via intersecting nondeterministic and co-nondeterministic classes. Indeed, consider any problem $\mathcal{L}$ in $\text{X}\Delta$ for any $\text{X} \in \{1, \text{R}, \text{S}, 2\}$, i. e., there exist small families $\mathcal{M}^{(1)}$, $\mathcal{M}^{(2)}$ of XNFAs solving $\mathcal{L}$ and $\overline{\mathcal{L}}$, respectively. Then it is possible to construct a small family $\mathcal{M}$ of XΔFA solving $\mathcal{L}$ in the following way: Consider any XNFAs $M_i^{(1)}$, $M_i^{(2)}$ with at most k states solving L_i and $\overline{L_i}$, respectively. We can construct an $O(k)$-state XΔFA M_i solving L_i in the following way: At the beginning of the computation, M_i nondeterministically guesses if the input word z is a positive or negative instance of the solved problem. Afterwards, it simulates $M_i^{(1)}$ ($M_i^{(2)}$) and accepts z (rejects z) if the simulated automaton accepts the input word, respectively. If the simulated automaton does not accept the input word, M neither accepts nor rejects z. Clearly, M is a correct self-verifying automaton, since every word can be either accepted or rejected, and no word can be both accepted and rejected.

For the other direction, it is easy to observe that any self-verifying automaton M solving language L can be transformed into nondeterministic automaton M' of the same type solving L, just by ignoring the special meaning of the reject state.[4] Similarly, it is possible to construct M' solving $\overline{L}$ just by declaring the reject state

[4] To be formally precise, it is necessary to remove any transition from the reject state q_r while reading $\dashv$; in M, this leads to the rejection of the input word, but M' should just hang.

q_r of M to be the accept state of M' and ignoring the special meaning of the accept state q_a of M. In both cases, M' has the same number of states as M.

Similarly as for the automata models, we can arrange all the introduced classes in a grid (see Figure 1.4), where each class is superset of its left and bottom neighbor. Compared to Figure 1.3, we can add the self-verifying classes to the map, too. The fact that $\mathrm{X}\Delta \subseteq \mathrm{X}\mathrm{N}$ holds for any $\mathrm{X} \in \{1, \mathrm{R}, \mathrm{S}, 2\}$ is easy to observe. The fact that $\mathrm{X}\mathrm{D} \subseteq \mathrm{X}\Delta$ follows easily from $\mathrm{X}\mathrm{D} \subseteq \mathrm{X}\mathrm{N}$ and the fact that $\mathrm{X}\mathrm{D}$ is closed under complement for all presented classes $\mathrm{X}\mathrm{D}$ – a fact that we discuss in Subsection 2.4.1.

N	$1\mathrm{N}$		RN		SN		$2\mathrm{N}$
$\cup$ |							
Δ	1Δ		$\mathrm{R}\Delta$		$\mathrm{S}\Delta$		2Δ
$\cup$ |							
D	$1\mathrm{D}$		RD		SD		$2\mathrm{D}$
	1	$\subseteq$	R	$\subseteq$	S	$\subseteq$	2

Figure 1.4: Basic complexity classes of finite automata.

Our work is dedicated to exploring the relationship among different complexity classes of finite automata. Hence, we will fill more and more details in the Figure 1.4, including the relationships of the co- and re- classes, as well as the randomized classes. In particular, our results show an interesting hierarchy inside the complexity classes of finite automata. Moreover, we analyse the closure properties of the considered classes as well.

In this thesis, we focus on the state complexity of finite automata. There are, however, several different ways how to measure the size of automata, such as the number of bits needed to describe the automaton (i. e., the descriptional complexity) or the number of transitions in the transition function of the automaton. Although these measures are not equivalent to the state complexity, they are polynomially related to the state complexity if the size of the alphabet is polynomial. Furthermore, it is easy to see that any language family with polynomial descriptional complexity or transition complexity has a polynomial alphabet. In our definition of the state complexity classes, problems with polynomially related complexity always fall into the same class. Hence, the definition of the complexity classes does not depend on the chosen measure if we consider only language families with alphabets of polynomial size. Even though we deal also with exponentially large alphabets in this thesis, it is not difficult to see that all presented results can be easily adapted for automata restricted to the binary alphabet as well. Hence, all relationships between different complexity classes presented in this thesis hold also for the other measures of automata size.

1.3 Known Results

Deterministic one-way finite automata were introduced in [Huf54, Mea55, Moo56]. Later, Rabin and Scott generalized their definition to nondeterministic ones in [RS59]. Here, they introduced the well-known *subset construction*, which they used to prove that the computational power of 1DFAs and 1NFAs is equal. Two-way deterministic automata were introduced in [RS59], too, as well as the proof of equivalent computing power of 1DFAs and 2DFAs (a simpler proof of this fact was published in [She59]). Nondeterministic variant of two-way finite automata was introduced in [Kol72], together with the proof of their computational equivalence with 1DFAs. (An alternative proof for this fact is in [Var89]). Since 2NFAs are obviously the strongest model introduced in Figure 1.3, and 1DFAs are the weakest model there, the result of [Kol72] implies that all models of finite automata introduced in Figure 1.3 have equal computational power.

The fact that all models of finite automata introduced so far have the same computational power encourages to compare their succinctness, i. e., to compare the size of different automata types required to solve the same language. Indeed, the study of the succinctness of finite automata was initiated very early and a lot of research has been made in this direction.

In [MF71] it was proven that a 1DFA may require 2^k states to simulate a k-state 1NFA, hence, the subset construction of [RS59] is tight. The gap in the state complexity between 2DFAs and 1DFAs is superexponential: It was shown in [MF71] that a 1DFA may require k^k states to simulate a 2DFA with $O(k)$ states. This was already quite close to the upper bound of [She59], which proved that any k-state 2DFA can be simulated by a 1DFA with $(k+2)^{k+1}$ states. The exponential gap in the state complexity between 1NFAs and 1DFAs, as well as superexponential gap between 2DFAs and 1DFAs, was proven independently in [Moo71].

While the relationship between determinism and nondeterminism was solved very soon for one-way finite automata, this is not the case for two-way automata. The question about the gap in state complexity between 2DFAs and 2NFAs was introduced in [SS78], and it is still an open problem. Hardness of this problem is emphasized also by its relationship to the L *vs.* NL *problem*: Results of [Sip80b,Ber80] show that if there is an exponential gap in the state complexity of 2DFAs and 2NFAs and, furthermore, it is possible to prove this gap using words of length polynomial in the number of states of the 2NFA, then the classes of problems solvable by deterministic and nondeterministic Turing machines working in logarithmic space (L and NL, respectively) are not equal. As the L vs. NL problem is one of the major open problems in the complexity theory, it gives additional motivation for the study of the relationship between 2DFAs and 2NFAs.

Since the relationship between determinism and nondeterminism for two-way automata was too hard to be attacked directly, the model of sweeping automata was proposed in [Sip80b]. Here it was also proven that, in the case of sweeping automata, the nondeterminism can be exponentially more succinct than determinism. In fact, a stronger result was proven in [Sip80b], showing that SDFAs may require exponen-

tially more states that 1NFAs (the exact value of the gap between SDFAs and 1NFAs was proved later in [Leu01]). Furthermore, the relationship with the L *vs.* NL problem propagates to the sweeping case, too: If the exponential gap in state complexity between determinism and nondeterminism for sweeping automata is reachable using only words of polynomial length, then L $\neq$ NL. On the other hand, it was shown in [SS78] that 1NFAs may require exponentially more states than SDFAs.

To prove the gap between SNFAs and SDFAs, the technique of *generic words* was introduced in [Sip80b]. Unfortunately, this technique relies heavily on long words, so it is not possible to solve the L vs. NL problem with it. Nevertheless, it is a powerful technique for proving lower bounds on the state complexity of rotating and sweeping automata. We employ it in the technique of hardness propagation and we generalize it for the case of time-bounded randomized automata.

The ideas of [Sip80b] can be used to prove an exponential gap in state complexity of sweeping and two-way deterministic automata, too. In fact, [Sip80b] cites a personal communication with J. Seiferas as a proof of this fact. An independent proof was, however, published in [Ber80, Mic81]. A partial answer to the relationship between SNFAs and 2DFAs was provided in [Kap06]. Here it was proven that a sweeping nondeterministic automaton may require exponentially more states than a two-way deterministic automaton. The opposite direction, i. e., the question if a 2DFA may require exponentially more states than a SNFA for some language, is still open. A positive answer would, however, immediately imply an exponential gap between determinism and nondeterminism for two-way automata.

The notion of self-verifying nondeterminism was introduced in [HS96] in the context of communication complexity and in [ĎHRS97] for finite automata, where the close connection to LasVegas randomization was explained. The exponential gap in the state complexity of 1DFAs and 1ΔFAs, as well as of 1ΔFAs and 1NFAs, was proved also in [SS78].

The relationship between self-verifying nondeterminism on one hand, and the determinism and nondeterminism on the other hand, is open for two-way automata. In fact, showing an exponential gap on either side would imply the existence of an exponential gap between determinism and nondeterminism immediately: Any 2ΔFA can be trivially simulated by a 2NFA with the same number of states. Furthermore, it was proven in [GMP07] that any k-state 2DFA accepting a language L can be transformed into a $O(k)$-state 2DFA accepting $\overline{L}$ (hence improving the upper bound of $O(k^2)$ states proved in [Sip80a]). For this reason, any k-state 2DFA can be simulated by an $O(k)$-state 2ΔFA. A quadratic gap in the state complexity between nondeterminism and self-verification for two-way automata was proven in [HS01b].

The relationship between self-verification and determinism/nondeterminism for rotating and sweeping automata is one of the main problems addressed in this thesis. More precisely, we prove an exponential gap between determinism and self-verification, as well as an exponential gap between self-verification and nondeterminism for both the rotating and the sweeping case. These results were also published in [KKM07, KKM08].

Using the above-mentioned results, we can express the basic relationships among the complexity classes in Figure 1.4, as depicted in Figure 1.5. Here, a solid arrow $\mathfrak{C} \to \mathfrak{C}'$ represents the fact that the class $\mathfrak{C}'$ is strict superset of $\mathfrak{C}$. A dashed arrow $\mathfrak{C} \to \mathfrak{C}'$ denotes that $\mathfrak{C}' \not\subseteq \mathfrak{C}$, i. e., there is some family of languages that belongs to $\mathfrak{C}'$, but does not belong to $\mathfrak{C}$. Hence, drawing a solid arrow $\mathfrak{C} \to \mathfrak{C}'$ is equivalent to drawing a dashed arrow and saying that $\mathfrak{C} \subseteq \mathfrak{C}'$.

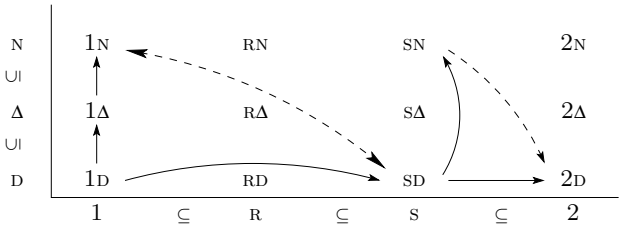

Figure 1.5: Known relationships between complexity classes of finite automata. A solid arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \subsetneq \mathfrak{C}'$, a dashed arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C}' \not\subseteq \mathfrak{C}$.

It is easy to observe that the arrows are in some sense transitive: If there is a dashed arrow $\mathfrak{C}_1 \to \mathfrak{C}_2$, then we can draw a dashed arrow $\mathfrak{C}_1 \to \mathfrak{C}_3$ for any $\mathfrak{C}_3 \supseteq \mathfrak{C}_2$, and also a dashed arrow $\mathfrak{C}_4 \to \mathfrak{C}_2$ for any $\mathfrak{C}_4 \subseteq \mathfrak{C}_1$. Using such transitivity, results in Figure 1.5 immediately imply that $1\Delta \subsetneq S\Delta \subsetneq 2\Delta$ and $1N \subsetneq SN \subsetneq 2N$.

In our work, we focus on the complexity classes that ignore polynomial differences in the automata size. In some sense, this is an analogy of the well-known polynomial complexity classes of Turing machines, and it makes the complexity classes more robust. It is possible, however, to analyse also the *exact* increase in state complexity when converting machine of one kind into another. An overview of exact values of such *trade-offs* between one-way and two-way deterministic and nondeterministic finite automata can be found in [Kap06]. Known results on *trade-offs* between different models are discussed in [GKK+02] as well.

For every complexity class, we have defined its complement and reverse in Section 1.2. In Chapter 2, we add all complements and reverses of classes in Figure 1.5 into the map, and provide a characterization of their relationships. In fact, it is possible to perform similar analysis for any other language operator instead of complement and reverse. This approach has been extensively studied (see, e. g., [JJS04, Jir05] and references therein).

Since the relationship between 2N and 2D is a long-standing open problem, several other approaches have been proposed for attacking it. One of them is to analyze the *degree of non-obliviousness* of two-way finite automata, as proposed in [HS03b, ĎHJ+01]. Here, the degree of non-obliviousness of a two-way finite automaton M is measured as the number of different orders of the input symbols that can appear in computations of M on words of length n. It is shown, by reduction to the case

of sweeping automata, that 2DFAs with non-obliviousness of degree $o(n)$ may require exponentially more states than 1NFAs.

Another extensively studied approach is to restrict the two-way automata to unary languages only, i. e., languages over a single-symbol alphabet. Here, the situation is often quite different than in the general case. For example, it was proven in [Chr86] that the cost of transformation from 1NFAs to 1DFAs is subexponential for unary languages; more precisely, it was proven that for any k-state 1NFA accepting an unary language there exists an equivalent 1DFA with $O\left(e^{\sqrt{k \log k}}\right)$ states and that this bound is asymptotically tight. Exact trade-offs between 1NFAs and 1DFAs for unary languages was further studied in [Gef07].

Some of the most prominent open problems regarding two-way automata were solved for the case of unary languages. Another result of [Chr86] shows that for any k-state 1NFA accepting a unary language there exists an equivalent 2DFA with $O(k^2)$ states, and that this bound is tight. Furthermore, a tight subexponential bound $O\left(e^{\sqrt{k \log k}}\right)$ for simulation of 2DFAs by 1DFAs was proven in [Chr86] as well. A subexponential upper bound on the simulation of 2NFAs by 2DFAs for unary languages was shown in [GMP03]: More precisely, it was shown that any k-state 2NFA accepting a unary language has an equivalent SDFA with $O\left(k^{\lceil \log_2(k+1)+3 \rceil}\right)$ states. Furthermore, an upper bound $O\left(e^{\sqrt{k \log k}}\right)$ for the simulation of unary 2NFAs by 1DFAs, as well as an upper bound $O(k^2)$ for the simulation of unary 2NFAs by SNFAs, was shown in [MP01].

Another interesting results for unary languages deals with the problem of complementation. Without imposing the restriction of unary languages, the question if 2N equals co-2N is open. In fact, it is conjectured that 2N $\neq$ co-2N and proving this conjecture would imply 2D $\neq$ 2N. On the other hand, it was proven in [GMP07] that, for any k-state 2NFA accepting language L, there exists a 2NFA accepting $\overline{L}$ with only $O(k^8)$ states.

To conclude the overview of previous research relevant to this thesis, we note that relationship between nondeterminism, randomization, and determinism was studied also for different models of automata, such as finite automata with two-dimensional input [ĎHI00]. The definition of the randomized models of finite automata, as well as the overview of results concerning these models, is postponed until Chapter 3, which is devoted to randomized finite automata.

Chapter 2

Determinism vs. Nondeterminism

In this chapter, we compare the state complexity classes of deterministic, nondeterministic, and self-verifying finite automata. In particular, we focus on refining the hierarchy between the classes in Figure 1.5, as well as their complements and reverses. The main result of this chapter is the separation of SD and SΔ. Furthermore, we analyse the closure properties of the introduced complexity classes.

To prove our results, we introduce a technique of *hardness propagation* for proving separations between the complexity classes. Using this technique, we are able to generate witness languages of these separations in a systematic way, what makes this technique interesting in itself. Furthermore, we generalize the hardness propagation for randomized models in the next chapter.

The high-level idea of the hardness propagation is the following one. To separate classes $\mathfrak{C}_{small}$ and $\mathfrak{C}_{large}$, it is sufficient to provide a *witness*, i. e., a family of languages $\mathcal{L} \in \mathfrak{C}_{large} - \mathfrak{C}_{small}$. While proving that a certain $\mathcal{L}$ is in $\mathfrak{C}_{large}$ can be often done by a straightforward construction of the corresponding family of small automata, proving that $\mathcal{L} \notin \mathfrak{C}_{small}$ is usually more difficult. It may be, however, feasible to prove that $\mathcal{L}$ is not in some class $\mathfrak{C}_{tiny}$ that is even more restricted than $\mathfrak{C}_{small}$. If we are able to find a language family operator $\mathcal{O}$ such that 1) for any language family such that $\mathcal{L} \notin \mathfrak{C}_{tiny}$, it holds that $\mathcal{O}(\mathcal{L}) \notin \mathfrak{C}_{small}$, and 2) $\mathfrak{C}_{large}$ is closed under $\mathcal{O}$, we directly obtain a witness $\mathcal{O}(\mathcal{L}) \in \mathfrak{C}_{large} - \mathfrak{C}_{small}$ of a desired separation. In this way, we can build a harder language ($\mathcal{O}(\mathcal{L})$) out of an easier one ($\mathcal{L}$); the hardness of $\mathcal{L}$ for class $\mathfrak{C}_{tiny}$ is *propagated* to the class $\mathfrak{C}_{small}$.

Obviously, the hardness propagation can be done in multiple steps. We start with a simple language family that is hard for some very restricted class, and, using several appropriate language operators, we propagate the hardness to more powerful classes.

We structure this chapter as follows. At first, we introduce language operators that are later used in the hardness propagation. Next, we introduce several models of finite automata used as intermediate steps. Afterwards we prove several hard-

ness propagation lemmas, which are used in the next section to fill the map of class relationship.

Results presented in this chapter have been published in [KKM07, KKM08].

2.1 Language Operators

We have introduced complement and reverse in Section 1.1. Now we define some more language operators that are helpful in the hardness propagation.

Let L, L_1, L_2 be arbitrary languages over alphabet Σ. Fix a delimiter # that is not in Σ. The languages $L_1 \wedge L_2$, $L_1 \vee L_2$, $L_1 \oplus L_2$, $\bigwedge L$, $\bigvee L$, and $\bigoplus L$ over alphabet $\Sigma \cup \{\#\}$ are defined as follows:

$$\begin{aligned}
L_1 \wedge L_2 &:= \{\#x\#y\# \mid x, y \in \Sigma^*, x \in L_1 \wedge y \in L_2\} \\
L_1 \vee L_2 &:= \{\#x\#y\# \mid x, y \in \Sigma^*, x \in L_1 \vee y \in L_2\} \\
L_1 \oplus L_2 &:= \{\#x\#y\# \mid x, y \in \Sigma^*, x \in L_1 \Leftrightarrow y \notin L_2\} \\
\bigwedge L &:= \{\#x_1\# \ldots \#x_l\# \mid l \geq 0, x_i \in \Sigma^*, (\forall i)(x_i \in L)\} \\
\bigvee L &:= \{\#x_1\# \ldots \#x_l\# \mid l \geq 0, x_i \in \Sigma^*, (\exists i)(x_i \in L)\} \\
\bigoplus L &:= \{\#x_1\# \ldots \#x_l\# \mid x_i \in \Sigma^*, \text{the number of } i \text{ such that } x_i \in L \text{ is odd}\}
\end{aligned} \tag{2.1}$$

We call these operators as *conjunctive concatenation*, *disjunctive concatenation*, *parity concatenation*, *conjunctive iteration*, *disjunctive iteration*, and *parity iteration*, respectively. Informally, language resulting from any of these operations consists of #-delimited *blocks* and a word is in the language if and only if the blocks satisfy the boolean operation used to define the operator.

As noted in Section 1.2, all language operators can be generalized for language families in a straightforward way, by applying the language operator on every member of the language family separately. More formally, let $\mathcal{L}_1 = (L_{1_n})_{n\geq 1}$, $\mathcal{L}_2 = (L_{2_n})_{n\geq 1}$ be language families, then $\mathcal{L}_1 \wedge \mathcal{L}_2 := (L_{1_n} \wedge L_{2_n})_{n\geq 1}$, and $\bigwedge \mathcal{L}_1 := (\bigwedge L_{1_n})_{n\geq 1}$. The definition for $\vee$, $\bigvee$, $\oplus$ and $\bigoplus$ is analogous.

It is easy to check that the following observation holds.

Observation 2.1. *The following equation holds both for languages and for language families L_1, L_2, L:*

$$\begin{aligned}
(L_1 \wedge L_2)^{\mathsf{R}} &= L_2^{\mathsf{R}} \wedge L_1^{\mathsf{R}} & \qquad (\textstyle\bigwedge L)^{\mathsf{R}} &= \textstyle\bigwedge(L^{\mathsf{R}}) \\
(L_1 \vee L_2)^{\mathsf{R}} &= L_2^{\mathsf{R}} \vee L_1^{\mathsf{R}} & \qquad (\textstyle\bigvee L)^{\mathsf{R}} &= \textstyle\bigvee(L^{\mathsf{R}}) \\
(L_1 \oplus L_2)^{\mathsf{R}} &= L_2^{\mathsf{R}} \oplus L_1^{\mathsf{R}} & \qquad (\textstyle\bigoplus L)^{\mathsf{R}} &= \textstyle\bigoplus(L^{\mathsf{R}}) \\
(\overline{L})^{\mathsf{R}} &= \overline{(L^{\mathsf{R}})}
\end{aligned}$$

Similar facts hold for complementation, but the situation is a bit more complicated. For example, the language $\overline{(L_1 \wedge L_2)}$ contains all words from language $\overline{L_1} \vee \overline{L_2}$, plus all words that do not consists of #-delimited blocks, i. e., not from language $R := \#\Sigma^*\#\Sigma^*\#$. We call such words *not well-formed*. Hence, it holds that $\overline{(L_1 \wedge L_2)} =$

$(\overline{L_1} \vee \overline{L_2}) \cup \overline{R}$. Similarly, $\overline{L_1} \vee \overline{L_2} = \overline{(L_1 \wedge L_2)} \cap R$. Nevertheless, both R and $\overline{R}$ are very simple regular languages, and union (intersection) with them usually does not play any significant role when dealing with state-complexity classes of finite automata:

Lemma 2.1. *Let $\mathcal{L}, \mathcal{L}_1$, and $\mathcal{L}_2$ be families of languages and let $\mathfrak{C}$ be a class of language families closed under union and intersection with any family from* 1D. *It holds that:*

$$\overline{(\mathcal{L}_1 \wedge \mathcal{L}_2)} \in \mathfrak{C} \Leftrightarrow \overline{\mathcal{L}_1} \vee \overline{\mathcal{L}_2} \in \mathfrak{C} \qquad \overline{(\bigwedge \mathcal{L})} \in \mathfrak{C} \Leftrightarrow \bigvee \overline{\mathcal{L}} \in \mathfrak{C}$$
$$\overline{(\mathcal{L}_1 \vee \mathcal{L}_2)} \in \mathfrak{C} \Leftrightarrow \overline{\mathcal{L}_1} \wedge \overline{\mathcal{L}_2} \in \mathfrak{C} \qquad \overline{(\bigvee \mathcal{L})} \in \mathfrak{C} \Leftrightarrow \bigwedge \overline{\mathcal{L}} \in \mathfrak{C}$$
$$\overline{(\mathcal{L}_1 \oplus \mathcal{L}_2)} \in \mathfrak{C} \Leftrightarrow \overline{\mathcal{L}_1} \oplus \mathcal{L}_2 \in \mathfrak{C}$$

Proof. To prove the first claim, let

$$\mathcal{L}_1 = (L_{1_n})_{n \geq 1}, \quad \mathcal{L}_2 = (L_{2_n})_{n \geq 1},$$

such that L_{1_i}, L_{2_i} are languages over Σ_i. As already said, it holds that

$$\overline{(\mathcal{L}_1 \wedge \mathcal{L}_2)} = (\overline{\mathcal{L}_1} \vee \overline{\mathcal{L}_2}) \cup \overline{\mathcal{R}}; \qquad \overline{\mathcal{L}_1} \vee \overline{\mathcal{L}_2} = \overline{(\mathcal{L}_1 \wedge \mathcal{L}_2)} \cap \mathcal{R}$$

where $\mathcal{R} := (\#\Sigma_n^*\#\Sigma_n^*\#)_{n \geq 1}$. Easily, $\mathcal{R}, \overline{\mathcal{R}} \in$ 1D, hence the first claim follows. Proof of the remaining claims are analogous, with the exception of using the well-formed language family $\mathcal{R}$ defined as $\mathcal{R} := (\#(\Sigma_n^*\#)^*)_{n \geq 1}$ for the conjunctive iteration and disjunctive iteration. □

All space-complexity classes introduced so far are closed under the intersection with 1D, what is straightforward to verify. Indeed, given a 1DFA M_1 accepting language L_1 and any XFA M_2 accepting language L_2 such that both M_1 and M_2 have at most k states, we can use the well-known *Cartesian-product construction* to construct a XFA accepting $L_1 \cap L_2$ with $O(k^2)$ states.

2.2 Parallel Automata

In this section, we introduce additional models that will be useful as intermediate steps of hardness propagation: Instead of propagating hardness directly from one-way to rotating/sweeping machines, we propagate the hardness from one-way to parallel automata and then from parallel automata to rotating and sweeping automata.

A (*two-sided*) *parallel automaton* (P$_2$1DFA), introduced in [Sip80b], is any triple $M = (\mathfrak{L}, \mathfrak{R}, F)$ where $\mathfrak{L} = \{C_1, \ldots, C_k\}$ and $\mathfrak{R} = \{D_1, \ldots, D_l\}$ are disjoint families of 1DFAs, and $F \subseteq Q_\perp^{C_1} \times \cdots \times Q_\perp^{C_k} \times Q_\perp^{D_1} \times \cdots \times Q_\perp^{D_l}$, where Q^A is the state set of automaton A and $Q_\perp^A := Q^A \cup \{\perp\}$ contains all possible results of runs of A. To run M on z means to run each $A \in \mathfrak{L} \cup \mathfrak{R}$ on z from its initial state and record the result, but with a twist: each $A \in \mathfrak{L}$ reads from left to right (i. e., reads z), while each $A \in \mathfrak{R}$ reads from right to left (i. e., reads z^R). We say that M accepts z iff the tuple of the

results of these computations is in F. More formally, let $C_i(z) \in Q_\perp^{C_i}$ be the result of $\text{LCOMP}_{C_i,q_I^{C_i}}(z)$, and let $D_i(z) \in Q_\perp^{D_i}$ be the result of $\text{LCOMP}_{D_i,q_I^{D_i}}(z^R)$, where q_I^A is the initial state of A. The parallel automaton M accepts z iff

$$(C_1(z), \ldots, C_k(z), D_1(z), \ldots, D_l(z)) \in F.$$

When $\mathfrak{R} = \emptyset$ or $\mathfrak{L} = \emptyset$, we say M is *left-sided* (a P_L1DFA) or *right-sided* (a P_R1DFA), respectively.

A *parallel intersection automaton* ($\cap_2$1DFA, $\cap_\text{L}$1DFA, or $\cap_\text{R}$1DFA) [SS78] is a parallel automaton whose F consists of the tuples where *all* results are final states, i. e., $F = F^{C_1} \times \ldots \times F^{C^k} \times F^{D_1} \times \ldots \times F^{D_l}$, where F^A is the set of final states of automaton A. If F consists of all tuples where *some* result is a final state, the automaton is a *parallel union automaton* ($\cup_2$1DFA, $\cup_\text{L}$1DFA, or $\cup_\text{R}$1DFA) [SS78]. Thus, a $\cap_2$1DFA accepts the input word z if and only if *all* its components accept z; a $\cup_2$1DFA accepts z if and only if *at least one* component does.

The number of states of a parallel automaton M is the sum of the numbers of states over all components of M. In accord with previous definitions, we say that a family of parallel automata $\mathcal{M} = (M_n)_{n \geq 1}$ is 'small' if M_n has at most $p(n)$ states for some polynomial p and all n. Hence, automata of $\mathcal{M}$ have a polynomial number of components with polynomial number of states each.

We use parallel intersection and union automata as intermediate steps of hardness propagation, hence we deal with their complexity classes, too. Following our naming convention, class $\cap_2$1D ($\cap_\text{L}$1D, $\cap_\text{R}$1D, $\cup_2$1D, $\cup_\text{L}$1D, $\cup_\text{R}$1D) contains all language families recognizable by small families of $\cap_2$1DFAs ($\cap_\text{L}$1DFAs, $\cap_\text{R}$1DFAs, $\cup_2$1DFAs, $\cup_\text{L}$1DFAs, $\cup_\text{R}$1DFAs), respectively.

The following lemma explains basic relationships between parallel automata on one hand and rotating and sweeping automata on the other hand. Using these relationships, we are able to provide a map of parallel-automata classes in Figure 2.1.

Lemma 2.2. *The following facts hold:*

1. $1\text{D} \subseteq \cap_\text{L}1\text{D} \cap \cup_\text{L}1\text{D}$, $\cap_\text{L}1\text{D} \cup \cap_\text{R}1\text{D} \subseteq \cap_2 1\text{D}$,
2. $\cap_\text{L}1\text{D} = \text{re-}\cap_\text{R}1\text{D}$, $\cup_\text{L}1\text{D} = \text{re-}\cup_\text{R}1\text{D}$, $\cap_2 1\text{D} = \text{re-}\cap_2 1\text{D}$, $\cup_2 1\text{D} = \text{re-}\cup_2 1\text{D}$
3. $\cap_\text{L}1\text{D} = \text{co-}\cup_\text{L}1\text{D}$, $\cap_\text{R}1\text{D} = \text{co-}\cup_\text{R}1\text{D}$, $\cap_2 1\text{D} = \text{co-}\cup_2 1\text{D}$
4. $\cap_\text{L}1\text{D} \cup \cup_\text{L}1\text{D} \subseteq \text{RD}$, $\cap_2 1\text{D} \cup \cup_2 1\text{D} \subseteq \text{SD}$.
5. *Every* RDFA *(*SDFA*) with k states can be simulated by a k-component* P_L1DFA *(*$\text{P}_2$1DFA*) whose components all have k states.*

Proof. 1. Since one-way automata are in fact special cases of both parallel union and parallel intersection automata, and left-sided and right-sided parallel automata are special cases of two-sided parallel automata, the claim follows.

2. If L can be solved by $\cap_{\text{L}}$1DFA or $\cap_2$1DFA $M = (\mathfrak{L}, \mathfrak{R})$ with m states, then L^{R} can be solved by $\cap_{\text{R}}$1DFA or $\cap_2$1DFA $M' = (\mathfrak{R}, \mathfrak{L})$ with m states, and vice versa. The same holds for parallel union automata.

3. If L can be solved by a k-component $\cap_{\text{L}}$1DFA M with m states, then $\overline{L}$ can be solved by a k-component $\cup_{\text{L}}$1DFA M' with $m + k$ states: to construct M', it is sufficient to 1) make all components of M non-hanging by adding one new state to every component, and 2) make all nonfinal states final and vice versa. Every word $w \notin L$ is rejected by some component of M, hence it is accepted by the corresponding component of M'. The same argument holds for $\cap_{\text{R}}$1DFAs and $\cap_2$1DFAs, too.

4. A RDFA can simulate any $\cap_{\text{L}}$1DFA or $\cup_{\text{L}}$1DFA in a straightforward way, simulating one component per traversal. We assume that every component of the parallel automaton is non-hanging, what can be achieved by adding one new state to every component of the automaton. Then, the set of states of the RDFA consists of all states of the parallel automaton plus one new accept state, so a small family of parallel automata are simulated by a small family of rotating automata. Similar arguments hold for simulation of $\cap_2$1DFA or $\cup_2$1DFA by a SDFA.

5. Proven in [Sip80b] for SDFAs; each component of the $\text{P}_2$1DFA simulates one traversal of the SDFA. The proof for RDFAs is analogous.

□

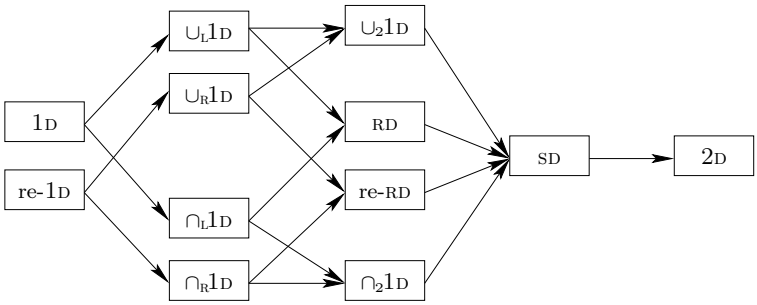

Figure 2.1: Complexity classes of finite automata, zoom to the parallel automata classes. A solid arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \subseteq \mathfrak{C}'$. Note that all complements and reverses of classes corresponding to parallel automata are already included in the map due to Lemma 2.2.

The correctness of the inclusions in Figure 2.1 follows from Lemma 2.2 and from Observation 1.1. The remaining inclusions, i. e., RD ∪ re-RD ⊆ SD ⊆ 2D, are very easy to verify.

2.3 Hardness Propagation

As already introduced, the main part of the hardness propagation consists of a proof that if a family of languages $\mathcal{L}$ is hard for some restricted model of finite automata, a family of more complex languages $\mathcal{O}(\mathcal{L})$ is hard for some more powerful model of automata. We use two basic tools for the construction of hard inputs to finite automata: the *confusing* and the *generic* strings. In this section, we present these tools and use them to prove the core hardness propagation lemmas.

2.3.1 Confusing Strings

We use the idea of confusing strings for the "lower level" of the hardness propagation, i. e., for propagating hardness from one-way deterministic automata to parallel intersection automata. Intuitively, we say that a string y confuses a parallel intersection automaton M if it should be rejected, but for each component of M it is indistinguishable from some word $\tilde{y}$ that should be accepted, or if y should be accepted, but some component of M hangs on y. In any case, existence of the confusing string implies that automaton M can not work correctly.

More formally, let $M = (\mathfrak{L}, \mathfrak{R})$ be a $\cap_2$1DFA and let L be a language over alphabet Σ. We say a string $y \in \Sigma^*$ *confuses* M *on* L if $y \in L$ but some component hangs on it or if $y \notin L$ but every component treats it identically to some word from L:

$$\text{or} \quad \begin{array}{lll} y \in L & \text{and} & (\exists A \in \mathfrak{L} \cup \mathfrak{R})\big(A(y) = \bot\big) \\ y \notin L & \text{and} & (\forall A \in \mathfrak{L} \cup \mathfrak{R})(\exists \tilde{y} \in L)\big(A(y) = A(\tilde{y})\big) \end{array} \tag{2.2}$$

It is not difficult to see that if some y confuses M on L, then M does not solve L. Note, though, that (2.2) is independent of the selection of final states in the components of M. Thus, if $\mathfrak{F}(M)$ is the class of $\cap_2$1DFAs that may differ from M only in the selection of final states, then a y that confuses M on L confuses every $M' \in \mathfrak{F}(M)$, too, and thus no $M' \in \mathfrak{F}(M)$ solves L, either. The converse is also true.

Lemma 2.3. *Let $M = (\mathfrak{L}, \mathfrak{R})$ be a $\cap_2$1DFA and L a language. Then, strings that confuse M on L exist iff no member of $\mathfrak{F}(M)$ solves L.*

Proof. [⇒] Suppose some y confuses M on L. Fix any $M' = (\mathfrak{L}', \mathfrak{R}') \in \mathfrak{F}(M)$. Since (2.2) is independent of the choice of final states, y confuses M' on L, too. *If $y \in L$:* By (2.2), some $A \in \mathfrak{L}' \cup \mathfrak{R}'$ hangs on y. So, M' rejects y, and thus fails. *If $y \notin L$:* If M' accepts y, it fails. If it rejects y, then some $A \in \mathfrak{L}' \cup \mathfrak{R}'$ does not accept y. Consider the $\tilde{y}$ guaranteed for this A by (2.2). Since $A(\tilde{y}) = A(y)$, we know $\tilde{y}$ is also not accepted by A. Hence, M' rejects $\tilde{y} \in L$, and fails again.

[⇐] Suppose no string confuses M on L. Then, no component hangs on a positive instance; and every negative instance is 'noticed' by some component, in the sense that the component treats it differently than all positive instances:

$$\text{and} \quad \begin{array}{l} (\forall y \in L)(\forall A \in \mathfrak{L} \cup \mathfrak{R})\big(A(y) \neq \bot\big) \\ (\forall y \notin L)(\exists A \in \mathfrak{L} \cup \mathfrak{R})(\forall \tilde{y} \in L)\big(A(y) \neq A(\tilde{y})\big). \end{array} \tag{2.3}$$

This allows us to find an $M' \in \mathfrak{F}(M)$ that solves L, as follows. We start with all states of all components of M unmarked. Then we iterate over all $y \notin L$. For each of them, we pick an A as guaranteed by (2.3) and, if the result $A(y)$ is a state, we mark it. When this (possibly infinite) iteration is over, we make all marked states nonfinal and all unmarked states final. The resulting $\cap_2$1DFA is our M'.

To see why M' solves L, consider any string y. *If $y \notin L$:* Then our method examined y, picked an A, and ensured $A(y)$ is either $\perp$ or a nonfinal state. So, this A does not accept y. Therefore, M' rejects y. *If $y \in L$:* Towards a contradiction, suppose M' rejects y. Then some component A^* does not accept y. By (2.3), $A^*(y) \neq \perp$. Hence, $A^*(y)$ is a state, call it q^*, and is nonfinal. Thus, at some point, our method marked q^*. Let $\hat{y} \notin L$ be the string examined at that point. Then, the selected A was A^* and $A(\hat{y})$ was q^*, and thus no $\tilde{y} \in L$ had $A^*(\tilde{y}) = q^*$ due to (2.3). But this contradicts the fact that $y \in L$ and $A^*(y) = q^*$. □

Note that Lemma 2.3 is valid also for the empty $\cap_2$1DFA $M = (\emptyset, \emptyset)$. In this case, M solves Σ^*, since every word is accepted by all components of M. Class $\mathfrak{F}(M)$ contains only M. If $L \neq \Sigma^*$, any word $y \notin L$ confuses M on L, since every component of M (vacuously, since there is no such component) treats it identically to some word in L. Conversely, if some y confuses M on L, $y \notin L$ so $L \neq \Sigma^*$.

Confusing strings can be used to prove that a certain language is hard for $\cap_2$1DFAs. At first we use this technique to propagate hardness from 1D to $\cap_L$1D.

Lemma 2.4. *If no* 1DFA *with at most m states can solve language L, then no $\cap_L$*1DFA *with at most m-state components can solve language $\bigvee L$. Similarly for $\bigoplus L$.*

Proof. Suppose no m-state 1DFA can solve L. By induction on k, we prove a stronger claim that every $\cap_L$1DFA with k m-state components is confused on $\bigvee L$ by some *well-formed* string y, i. e., $y \in \#(\Sigma^*\#)^*$. The proof for $\bigoplus L$ is similar.

If $k = 0$: Fix any such $\cap_L$1DFA $M = (\mathfrak{L}, \emptyset) = (\emptyset, \emptyset)$. By definition, $\# \notin \bigvee L$. Furthermore, # confuses M on $\bigvee L$, because all components of M (vacuously, since $\mathfrak{L} = \emptyset$) treats it identically to some word in $\bigvee L$. Since # is well-formed, the claim holds.

If $k \geq 1$: Fix any such $\cap_L$1DFA $M = (\mathfrak{L}, \emptyset)$. Pick any $D \in \mathfrak{L}$ and remove it from M to get $M_1 = (\mathfrak{L}_1, \emptyset) := (\mathfrak{L} - \{D\}, \emptyset)$. By the inductive hypothesis, some well-formed y confuses M_1 on $\bigvee L$.

Case 1: $y \in \bigvee L$. Then some $A \in \mathfrak{L}_1$ hangs on y. Since $A \in \mathfrak{L}$, too, y confuses M as well, so the inductive step is complete.

Case 2: $y \notin \bigvee L$. Then every $A \in \mathfrak{L}_1$ treats y identically to a positive instance:

$$(\forall A \in \mathfrak{L} - \{D\})(\exists \tilde{y} \in \textstyle\bigvee L)\big(A(y) = A(\tilde{y})\big). \tag{2.4}$$

Now we define a single-component $\cap_L$1DFA $M_2 = (\{D'\}, \emptyset)$. If D hangs on y (i. e., $D(y) = \perp$), we define D' to be a single-state automaton that hangs immediately. Otherwise, D' is derived from D by changing its initial state to $D(y)$. In any case, it holds that $D'(z) = D(yz)$ for any non-empty word z.

By the assumption of the lemma, no member of $\mathfrak{F}(M_2)$ solves L. So, by Lemma 2.3, some x confuses M_2 on L. We claim that $yx\#$ confuses M on $\bigvee L$. Since $yx\#$ is well-formed, the induction is again complete. To prove the confusion, we examine cases:

Case 2a: $x \in L$. Then $yx\# \in \bigvee L$, since y is well-formed and $x \in L$. And D' hangs on x (since x is confusing for M_2 and D' is the only component), thus $D(yx\#) = D'(x\#) = \bot$. So, component D of M hangs on $yx\# \in \bigvee L$. So, $yx\#$ confuses M on $\bigvee L$.

Case 2b: $x \notin L$. Then $yx\# \notin \bigvee L$, because y is well-formed and not in $\bigvee L$, and x does not contain #. And, since x is confusing for M_2, D' treats it identically to some $\tilde{x} \in L$: $D'(x) = D'(\tilde{x})$. Then, each component of M treats $yx\#$ identically to a positive instance of $\bigvee L$:

- D treats $yx\#$ as $y\tilde{x}\#$: $D(y\tilde{x}\#) = D'(\tilde{x}\#) = D'(x\#) = D(yx\#)$. And we know $y\tilde{x}\# \in \bigvee L$, because y is well-formed and $\tilde{x} \in L$.
- each $A \neq D$ treats $yx\#$ as $\tilde{y}x\#$, where $\tilde{y}$ the string guaranteed for A by (2.4): $A(\tilde{y}x\#) = A(yx\#)$. And we know $\tilde{y}x\# \in \bigvee L$, since $\tilde{y} \in \bigvee L$ and x does not contain #.

Overall, $yx\#$ is again a well-formed confusing string for M on $\bigvee L$, as required. □

Corollary 2.5. *If $\mathcal{L} \notin$ 1D, then $\bigvee\mathcal{L} \notin \cap_{\text{L}}$1D and $\bigoplus\mathcal{L} \notin \cap_{\text{L}}$1D.*

Proof. Assume the contrary, i. e., $\bigvee\mathcal{L} \in \cap_{\text{L}}$1D. Hence, there is a family $\mathcal{M} = (M_n)_{n\geq 1}$ of small $\cap_{\text{L}}$1DFAs solving $\bigvee\mathcal{L}$, i. e., M_n has at most $p(n)$ states overall for some polynomial $p(n)$. Obviously, M_n has at most $p(n)$ state components, so by Lemma 2.4 there exists a family $\mathcal{M}' = (M'_n)_{n\geq 1}$ of 1DFAs solving $\mathcal{L}$ such that M'_n has at most $p(n)$ states. Hence, $\mathcal{L} \in$ 1D, a contradiction. The proof for $\bigoplus$ is analogous. □

In fact, Lemma 2.4 is stronger than Corollary 2.5, since it proves that even arbitrarily high number of components cannot help to solve $\mathcal{L}$ if the components are small. We discuss the implications of this fact in Section 2.5.

Next, we prove a hardness propagation from $\cap_{\text{L}}$1D to $\cap_{\text{2}}$1D.

Lemma 2.6. *Let $m \geq 1$. If L_1 has no $\cap_{\text{L}}$1DFA with at most m states and L_2 has no $\cap_{\text{R}}$1DFA with at most m states, then $L_1 \vee L_2$ has no $\cap_{\text{2}}$1DFA with at most m states. Similarly for $L_1 \oplus L_2$.*

Proof. Let $M = (\mathfrak{L}, \mathfrak{R})$ be a $\cap_{\text{2}}$1DFA with at most m states; we show that M does not accept $L_1 \vee L_2$. By contradiction, assume that M accepts $L_1 \vee L_2$. Let $M_1 := (\mathfrak{L}', \emptyset)$ and $M_2 := (\emptyset, \mathfrak{R}')$ be the $\cap_{\text{2}}$1DFAs derived from the two 'sides' of M after changing the initial state of each $A \in \mathfrak{L} \cup \mathfrak{R}$ to $A(\#)$. Note that we can assume that $A(\#) \neq \bot$: Otherwise, A hangs on every well-formed word, hence M accepts an empty language. This implies that both L_1 and L_2 are empty, hence they can be accepted by a $\cap_{\text{L}}$1DFA (a $\cap_{\text{R}}$1DFA) with 1 state; a contradiction.

By the lemma's assumption, no member of $\mathfrak{F}(M_1)$ solves L_1 and no member of $\mathfrak{F}(M_2)$ solves L_2. So, by Lemma 2.3, some y_1 confuses M_1 on L_1 and some y_2 confuses M_2 on L_2. We claim that $\#y_1\#y_2\#$ confuses M on $L_1 \vee L_2$ and thus M fails.

Case 1: $y_1 \in L_1$ or $y_2 \in L_2$. Assume $y_1 \in L_1$ (if $y_2 \in L_2$, we work similarly). Then $\#y_1\#y_2\# \in L_1 \vee L_2$ and some $A' \in \mathfrak{L}'$ hangs on y_1. The corresponding $A \in \mathfrak{L}$ has $A(\#y_1\#y_2\#) = A'(y_1\#y_2\#) = \bot$. So, $\#y_1\#y_2\#$ confuses M on $L_1 \vee L_2$.

Case 2: $y_1 \notin L_1$ and $y_2 \notin L_2$. Then $\#y_1\#y_2\# \notin L_1 \vee L_2$, each component of M_1 treats y_1 identically to a positive instance of L_1, and each component of M_2 treats y_2 identically to a positive instance of L_2:

$$(\forall A' \in \mathfrak{L}')(\exists \tilde{y}_1 \in L_1)\big(A'(y_1) = A'(\tilde{y}_1)\big), \tag{2.5}$$

$$(\forall A' \in \mathfrak{R}')(\exists \tilde{y}_2 \in L_2)\big(A'(y_2) = A'(\tilde{y}_2)\big). \tag{2.6}$$

It is then easy to verify that every $A \in \mathfrak{L}$ treats $\#y_1\#y_2\#$ as $\#\tilde{y}_1\#y_2\# \in L_1 \vee L_2$ ($\tilde{y}_1$ as guaranteed by (2.5)), and every $A \in \mathfrak{R}$ treats $\#y_1\#y_2\#$ as $\#y_1\#\tilde{y}_2\# \in L_1 \vee L_2$ ($\tilde{y}_2$ as guaranteed by (2.6)). Therefore, $\#y_1\#y_2\#$ confuses M on $L_1 \vee L_2$, again.

The proof for the parity concatenation is analogous. It is necessary, however, to split Case 1 into two sub-cases:

Case 1a: Either $y_1 \in L_1$ or $y_2 \in L_2$. This case is completely analogous to Case 1 of the proof for disjunctive concatenation.

Case 1b: Both $y_1 \in L_1$ and $y_2 \in L_2$. Since L_2 can not be accepted by a $\cap_{\mathrm{R}}$1DFA with $m \geq 1$ states, it is nontrivial, i. e., there exists some $\tilde{y}_2 \notin L_2$ that does not contain #. Hence $\#y_1\#\tilde{y}_2\# \in L_1 \oplus L_2$ and, by similar arguments as for the disjunctive concatenation, $\#y_1\#\tilde{y}_2\#$ confuses M.

□

Corollary 2.7. *If $\mathcal{L} \notin \cap_{\mathrm{L}}$1D, then $\mathcal{L} \vee \mathcal{L}^{\mathrm{R}} \notin \cap_2$1D and $\mathcal{L} \oplus \mathcal{L}^{\mathrm{R}} \notin \cap_2$1D.*

Proof. Let $\mathcal{L} = (L_n)_{n\geq 1}$. Assume the contrary, i. e., $\mathcal{L} \vee \mathcal{L}^{\mathrm{R}} \in \cap_2$1D. Hence, there exists a family $\mathcal{M} = (M_n)_{n\geq 1}$ of small $\cap_2$1DFAs solving $\mathcal{L} \vee \mathcal{L}^{\mathrm{R}}$, i. e., M_n has at most $p(n)$ states for some polynomial $p(n)$. By Lemma 2.6 there is either a $\cap_{\mathrm{L}}$1DFA M_n^L solving L_n with at most $p(n)$ states, or a $\cap_{\mathrm{R}}$1DFA M_n^R solving L_n^{R} with at most $p(n)$ states. If the second case occurs, we can obtain M_n^L solving L_n with at most $p(n)$ states from M_n^R by swapping its left and right components. Hence, there exists a family $\mathcal{M}' = (M_n^L)_{n\geq 1}$ of $\cap_{\mathrm{L}}$1DFAs solving $\mathcal{L}$ such that M_n^L has at most $p(n)$ states. So, $\mathcal{L} \in \cap_{\mathrm{L}}$1D, a contradiction. Proof for $\oplus$ is analogous. □

Similarly as in the case of Lemma 2.4, Lemma 2.6 can be formulated for automata with m-state components instead of m-state automata, too.

The following auxiliary lemma is quite simple, and it does not use the idea of confusing strings. Nevertheless, it allows us to construct more elaborate hard languages out of simpler ones, so we present it for the sake of completeness. Intuitively, it just shows that concatenation operations do not decrease the hardness of the language.

Lemma 2.8. *Let L' be nontrivial, $\pi \in \{\cap, \cup, \mathrm{P}\}$, $_\sigma \in \{_{\mathrm{L}}, _{\mathrm{R}}, _2\}$. If L has no π_σ1DFA with m-states, then neither $L \wedge L'$ has. Similarly for $L \vee L'$ and $L \oplus L'$.*

Proof. We prove only the first claim, for $\pi = \cap$ and $_\sigma = {}_{\mathrm{L}}$. Fix any $y' \in L'$. Given a $\cap_{\mathrm{L}}$1DFA M' solving $L \wedge L'$ with m-states, we build a $\cap_{\mathrm{L}}$1DFA M solving L with m-states: We just modify each component A' of M' so that the modified A' works on y exactly as A' on $\#y\#y'\#$. Then, M accepts $y \Leftrightarrow M'$ accepts $\#y\#y'\# \Leftrightarrow y \in L$. The modifications for proving the other claims are straightforward. □

2.3.2 Generic Strings

In this subsection, we proceed to the "upper level" of the hardness propagation. We show how to propagate hardness from $\cap_{\mathrm{L}}$1DFAs to RDFAs. Furthermore, we generalize these results for sweeping automata, too, showing a hardness propagation from $\cap_2$1DFAs to SDFAs.

Intuition. For the case of rotating automata, our goal is to show that if some language L is hard for $\cap_{\mathrm{L}}$1DFAs, then the language $\bigwedge L$ is hard for RDFAs. Intuitively, this fact is not very surprising: Language $\bigwedge L$ consists of arbitrarily many #-delimited blocks, and the goal of the RDFA is to verify if *each* of these blocks belongs to L. Since the RDFA is finite, it can carry only finite information between different traversals. Obviously, this finite information does not suffice to cover every #-delimited block. Hence, the RDFA must be able to verify the correctness of every block in several traversals that do not interact with each other. This corresponds well to the model of parallel intersection automata: Essentially, the RDFA accepting $\bigwedge L$ must act as a $\cap_{\mathrm{L}}$1DFA. This leads to the conclusion that if the language $\bigwedge L$ can be accepted by a small RDFA, then the language L can be accepted by a small $\cap_{\mathrm{L}}$1DFA.

While the intuition behind this step of hardness propagation is not too difficult, it is not trivial to formalize it. To do so, the technique of *generic strings*, which are a powerful tool for proving lower bounds on the state complexity of rotating and sweeping automata, was introduced in [Sip80b]. In fact, the results presented in this subsection refine the results of [Sip80b] by explicitly separating the two steps of hardness propagation. While this is not a significant contribution by itself, it simplifies the analysis of the complexity classes of finite automata, allows us to separate nondeterminism and self-verifying nondeterminism for rotating and sweeping automata, and opens the way for the generalization of the generic strings for time-bounded randomized models of finite automata.

Formal proof. The rest of this subsection is devoted to the formalization of the above-described intuitive approach. Let A be a 1DFA over alphabet Σ and states Q, and $y, z \in \Sigma^*$. The *left views of A on y* is the set of states reached on the right boundary of y by left computations of A:

$$\textsc{lviews}_A(y) := \{q \in Q \mid (\exists p \in Q)[\textsc{lcomp}_{A,p}(y) \text{ results in } q]\}.$$

The (*left*) *mapping of A on y and z* is the partial function

$$\textsc{lmap}_A(y, z) : \textsc{lviews}_A(y) \to Q$$

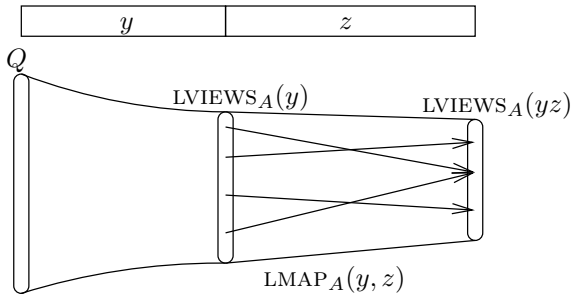

Figure 2.2: LVIEWS and LMAP.

which, for every $q \in \text{LVIEWS}_A(y)$, is defined only if $\text{LCOMP}_{A,q}(z)$ does not hang and, if so, returns the state that this computation results in. (See Figure 2.2.)

It is straightforward to verify that this function is a *partial surjection* from the set $\text{LVIEWS}_A(y)$ to the set $\text{LVIEWS}_A(yz)$. First, the values of $\text{LMAP}_A(y,z)$ are all in $\text{LVIEWS}_A(yz)$. Indeed: Let r be a value of $\text{LMAP}_A(y,z)$. Then some $q \in \text{LVIEWS}_A(y)$ is such that $\text{LMAP}_A(y,z)(q) = r$. Since $q \in \text{LVIEWS}_A(y)$, we know some $c := \text{LCOMP}_{A,p}(y)$ results in q. Since $\text{LMAP}_A(y,z)(q) = r$, we know $d := \text{LCOMP}_{A,q}(z)$ results in r. Overall, the computation $\text{LCOMP}_{A,p}(yz)$ must be exactly the concatenation of c and d. So, it results in the same state as d, namely r. Therefore $r \in \text{LVIEWS}_A(yz)$.

Second, the values of $\text{LMAP}_A(y,z)$ cover the entire $\text{LVIEWS}_A(yz)$. Indeed: Suppose $r \in \text{LVIEWS}_A(yz)$. Then some $c' := \text{LCOMP}_{A,p}(yz)$ results in r. Let q be the state of c' right after crossing the y-z boundary. Clearly, (i) the computation $\text{LCOMP}_{A,p}(y)$ results in q, and (ii) the computation $\text{LCOMP}_{A,q}(z)$ results in the same state as c', namely r. By (i), we know that $q \in \text{LVIEWS}_A(y)$. By (ii), we know that $\text{LMAP}_A(y,z)(q) = r$. Therefore, r is a value of $\text{LMAP}_A(y,z)$.

Therefore, $\text{LMAP}_A(y,z)$ partially surjects $\text{LVIEWS}_A(y)$ onto $\text{LVIEWS}_A(yz)$. This immediately implies Fact 2.1. Fact 2.2 is equally simple.

Fact 2.1. *For all A, y, z as above*: $|\text{LVIEWS}_A(y)| \geq |\text{LVIEWS}_A(yz)|$.

Fact 2.2. *For all A, y, z as above*: $\text{LVIEWS}_A(yz) \subseteq \text{LVIEWS}_A(z)$.

Proof. Suppose $r \in \text{LVIEWS}_A(yz)$. Then some computation $c := \text{LCOMP}_{A,p}(yz)$ results in r. If q is the state of c after crossing the y-z boundary, then $\text{LCOMP}_{A,q}(z)$ is a suffix of c and results in r. So, $r \in \text{LVIEWS}_A(z)$. □

In the following, we are going to prove that if there is no small $\cap_{\text{L}}$1DFA solving L, then there is no small P_{L}1DFA solving $\bigwedge L$. Using this result, we can easily obtain the hardness propagation from $\cap_{\text{L}}$1DFAs to RDFAs, since, due to Lemma 2.2, P_{L}1DFAs are at least as strong as RDFAs. Hence, we prove a stronger result of hardness propagation from $\cap_{\text{L}}$1DFAs to P_{L}1DFAs, but we are not dealing with the complexity classes of P_{L}1DFAs. We provide, however, some discussion about those classes in Section 2.5.

Consider any $\text{P}_\text{L}\text{1DFA}$ $M = (\mathfrak{L}, \emptyset, F)$ and any language L. We say that $uu' \in L$ is a right-extension of $u \in L$. A string $y \in L$ is called L-*generic (for* M*) over* L, if the size of $\text{LVIEWS}_A(y)$ cannot be decreased by any right-extension of y for any component $A \in \mathfrak{L}$:

$$y \in L \quad \text{and} \quad (\forall yz \in L)(\forall A \in \mathfrak{L})\big(|\text{LVIEWS}_A(y)| = |\text{LVIEWS}_A(yz)|\big) \tag{2.7}$$

Note that we can use equality in (2.7) due to Fact 2.1.

It is easy to see that L-generic strings always exist: Consider $y \in L$ such that

$$\sum_{A \in \mathfrak{L}} |\text{LVIEWS}_A(y)|$$

is minimal possible. For any $yz \in L$, Fact 2.1 ensures that no component of the sum is increased. Since the sum cannot be decreased by the definition of y, string y is L-generic.

The next lemma shows an important property of L-generic strings. Intuitively, it shows that the behavior of the parallel automaton is very limited after reading a generic string. Later we exploit this limitation to build a small $\cap_\text{L}\text{1DFA}$ for language L from a small $\text{P}_\text{L}\text{1DFA}$ accepting $\bigwedge L$.

Lemma 2.9. *Suppose a* $\text{P}_\text{L}\text{1DFA}$ $M = (\mathfrak{L}, \emptyset, F)$ *solves* $\bigwedge L$ *and* y *is* L*-generic for* M *over* $\bigwedge L$*. Then,* $x \in L$ *iff* $\text{LMAP}_A(y, xy)$ *is total and injective for all* $A \in \mathfrak{L}$*.*

Proof. [⇒] Let $x \in L$. Then $yxy \in \bigwedge L$, because $y \in \bigwedge L$ and $x \in L$. So, yxy is a right-extension of y inside $\bigwedge L$. Since y is L-generic, $|\text{LVIEWS}_A(y)| = |\text{LVIEWS}_A(yxy)|$, for all $A \in \mathfrak{L}$. Hence, each partial surjection $\text{LMAP}_A(y, xy)$ has domain and codomain of the same size. This is possible only if the function is a bijection, i. e., it is both total and injective.

[⇐] Suppose that, for each $A \in \mathfrak{L}$, the partial surjection $\text{LMAP}_A(y, xy)$ is total and injective. Then it bijects the set $\text{LVIEWS}_A(y)$ into the set $\text{LVIEWS}_A(yxy)$, which is actually a subset of $\text{LVIEWS}_A(y)$ (Fact 2.2). Clearly, this is possible only if this subset is the set itself. Hence, $\text{LMAP}_A(y, xy)$ is a permutation π_A of $\text{LVIEWS}_A(y)$.

Now pick $k \geq 1$ such that each π_A^k is an identity for each A. Clearly, it is always possible to find such k; for example, it is sufficient to choose $k = m!$, where m is the maximal number of states over all components of $\mathfrak{L}$. Let $z := y(xy)^k$. It is easy to verify that $\text{LMAP}_A\big(y, (xy)^k\big)$ equals $\text{LMAP}_A(y, xy)^k = \pi_A^k$, and is therefore the identity on $\text{LVIEWS}_A(y)$. This means that, reading through z, the left computations of A do not notice the suffix $(xy)^k$ to the right of the prefix y. So, no A can distinguish between y and z: it either hangs on both or results both in the same state.

To show this more formally, suppose A accepts $z = y(xy)^k$ and let

$$c := \text{LCOMP}_{A,p}\big(y(xy)^k\big)$$

be its computation, where p is its initial state. Then c results in a final state r. Easily, c can be split into subcomputations $c' := \text{LCOMP}_{A,p}(y)$, which results in some state

q, and $c'' := \textsc{lcomp}_{A,q}\big((xy)^k\big)$, which results in r. By the selection of q and r and the fact that π_A^k is an identity, we know

$$r = \textsc{lmap}_A\big(y, (xy)^k\big)(q) = \pi_A^k(q) = q.$$

Hence, c results $z = y(xy)^k$ in the same state in which c' results y. (Intuitively, in reading $(xy)^k$ to the right of y, the full computation c achieves nothing more than what is already achieved on y by its prefix c'.) Since r is final, A accepts y.

Conversely, any accepting computation of A on y can be extended into an accepting computation on z – this time by pumping up (as opposed to pumping down) and by using the computations that cause π_A^k to be an identity.

Thus, M does not distinguish between y and z, either: it either accepts both or rejects both. But M accepts y (because $y \in \bigwedge L$), so it accepts z. Hence, every #-delimited block of z is in L. In particular, $x \in L$. □

If $M = (\mathfrak{L}, \mathfrak{R}, F)$ is a $\textsc{p}_2\textsc{1dfa}$, we can also work symmetrically with right computations and left-extensions: we can define $\textsc{rviews}_A(y)$ and $\textsc{rmap}_A(z, y)$ for $A \in \mathfrak{R}$, derive Facts 2.1, 2.2 for $\textsc{rviews}_A(y)$ and $\textsc{rviews}_A(zy)$, and define R-generic strings. In particular, $\textsc{rviews}_A(y)$ is the set of states of A reached on the left boundary of y after reading it backwards and $\textsc{rmap}_A(z, y)$ is the function that maps states of $\textsc{rviews}_A(y)$ into the states reached after reading z backwards. We can then construct strings that are simultaneously L- and R-generic; we call such strings *generic*. Indeed, it is easy to verify that if $y_L\#$ is L-generic over $\bigwedge L$ and $\#y_R$ is R-generic over $\bigwedge L$, then $y_L\#y_R$ is a generic string over $\bigwedge L$.

Generic strings can be used to extend Lemma 2.9 for $\textsc{p}_2\textsc{1dfa}$s. The following lemma can be proved by a straightforward extension of the proof of Lemma 2.9:

Lemma 2.10. *Suppose a* $\textsc{p}_2\textsc{1dfa}$ $M = (\mathfrak{L}, \mathfrak{R}, F)$ *solves* $\bigwedge L$ *and* y *is generic for* M *over* $\bigwedge L$. *Then,* $x \in L$ *iff* $\textsc{lmap}_A(y, xy)$ *is total and injective for all* $A \in \mathfrak{L}$ *and* $\textsc{rmap}_A(yx, y)$ *is total and injective for all* $A \in \mathfrak{R}$.

Now we can use the properties of generic strings to prove hardness propagation from $\cap_{\text{L}}\textsc{1d}$ to RD. The following lemma proves a stronger result of hardness propagation from $\cap_{\text{L}}\textsc{1dfa}$s to $\textsc{p}_{\text{L}}\textsc{1dfa}$s; the actual hardness propagation to RD is stated as the following corollary.

Lemma 2.11. *If* L *has no* $\cap_{\text{L}}\textsc{1dfa}$ *with at most* $k \cdot \binom{m}{2}$ *components of at most* $\binom{m}{2}$ *states each, then* $\bigwedge L$ *has no* $\textsc{p}_{\text{L}}\textsc{1dfa}$ *with at most* k *components of at most* m *states each.*

Proof. Let $M = (\mathfrak{L}, \emptyset, F)$ be a $\textsc{p}_{\text{L}}\textsc{1dfa}$ solving $\bigwedge L$ with at most k components of at most m states each. Let y be L-generic for M over $\bigwedge L$. We build a $\cap_{\text{L}}\textsc{1dfa}$ M' solving L.

By Lemma 2.9, an arbitrary x is in L iff $\textsc{lmap}_A(y, xy)$ is total and injective for all $A \in \mathfrak{L}$; i. e., iff for all $A \in \mathfrak{L}$ and every two distinct $p, q \in \textsc{lviews}_A(y)$,

$$\textsc{lcomp}_{A,p}(xy) \text{ and } \textsc{lcomp}_{A,q}(xy) \text{ result } xy \text{ into different states.} \tag{2.8}$$

So, checking $x \in L$ reduces to checking (2.8) for each A and two-set of states of $\textsc{lviews}_A(y)$. The components of M' perform exactly these checks. To describe them, let us first define the following relation on the states of an $A \in \mathfrak{L}$:

$$r \asymp_A s \iff \textsc{lcomp}_{A,r}(y) \text{ and } \textsc{lcomp}_{A,s}(y) \text{ result } y \text{ into different states,}$$

and restate our checks as follows: for all $A \in \mathfrak{L}$ and all distinct $p, q \in \textsc{lviews}_A(y)$,

$$\textsc{lcomp}_{A,p}(x) \text{ and } \textsc{lcomp}_{A,q}(x) \text{ result } x \text{ into states that relate under } \asymp_A. \quad (2.8')$$

Now, building 1DFAs to perform these checks is easy. For each $A \in \mathfrak{L}$ and $p, q \in \textsc{lviews}_A(y)$, the corresponding 1DFA has 1 state for each two-set of states of A. The initial state is $\{p, q\}$. At each step, the automaton applies A's transition function on the current symbol and each state in the current two-set. If either application returns no value or both return the same value, it hangs; otherwise, it moves to the resulting two-set. A state $\{r, s\}$ is final iff $r \asymp_A s$.

Since for every $A \in \mathfrak{L}$ we constructed at most $\binom{m}{2}$ components of M', each with at most $\binom{m}{2}$ states, the proof is completed. □

Corollary 2.12. *If $\mathcal{L} \notin \cap_{\text{L}}\text{1D}$, then $\bigwedge\mathcal{L} \notin$ RD.*

Proof. Let $\mathcal{L} = (L_n)_{n\geq 1}$. Assume the contrary, i. e., $\bigwedge\mathcal{L} \in$ RD. Hence, there is a family $\mathcal{M} = (M_n)_{n\geq 1}$ of small RDFAs solving $\bigwedge\mathcal{L}$, i. e., M_n has at most $p(n)$ states for some polynomial $p(n)$. By Lemma 2.2, there is a P_{L}1DFA M'_n with at most $p(n)$ components of at most $p(n)$ states each such that M'_n is equivalent to M_n. By Lemma 2.11, there is a $\cap_{\text{L}}$1DFA M''_n solving L_n with at most $p(n)\binom{p(n)}{2}$ components of at most $\binom{p(n)}{2}$ states each. Hence, there exists a small family $\mathcal{M}'' = (M''_n)$ of $\cap_{\text{L}}$1DFAs solving $\mathcal{L}$, a contradiction. □

Hardness propagation from $\cap_2$1D to SD is analogous to Lemma 2.11 and Corollary 2.12. The proof is very similar to the case of rotating automata; RCOMP and RVIEWS are used similarly as LCOMP and LVIEWS.

Lemma 2.13. *If L has no $\cap_2$1DFA with at most $k \cdot \binom{m}{2}$ components of at most $\binom{m}{2}$ states each, then $\bigwedge L$ has no $\text{P}_2$1DFA with at most k components of at most m states each.*

Corollary 2.14. *If $\mathcal{L} \notin \cap_2$1D, then $\bigwedge\mathcal{L} \notin$ SD.*

To conclude this subsection, we provide an overview of the presented hardness propagation lemmas in Figure 2.3.

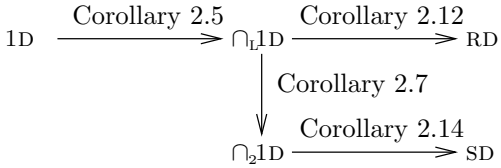

Figure 2.3: The structure of the hardness propagation lemmas.

2.4 Map of the Complexity Classes

In this section, we explore the relationship between the complexity classes of finite automata introduced so far. In particular, we present a complete map of relationships between these classes. We, however, analyze their closure properties before filling the map. We do this for the following reason: Positive closure properties help us to collapse many complexity classes, what simplifies the map of the classes significantly. Furthermore, we need these properties for finding language families witnessing the separation of the classes when using the hardness propagation technique.

	1D	1Δ	1N	∪L1D	RD	∪21D	SD	SΔ	SN	2D	2Δ	2N	
$\overline{\cdot}$	+	+	−	−	+	−	+	+	−	+	+	?	1
$\cdot^R$	−	+	+	−	−	+	+	+	+	+	+	+	2
∧	+	+	+	+	+	−	+	+	+	+	+	+	3
∨	+	+	+	+	+	+	+	+	+	+	+	+	4
⊕	+	+	−	−	+	−	+	+	−	+	+	?	5
⋀	+	+	+	−	−	−	−	?	?	+	+	+	6
⋁	+	+	+	+	−	+	−	?	?	+	+	+	7
⨁	+	+	−	−	?	−	?	?	−	+	+	?	8
	A	B	C	D	E	F	G	H	H	I	J	K	

Figure 2.4: Closure properties: '+' means closure; '-' means non-closure; '?' means we do not know.

At first, we focus on the positive closure properties mentioned in Figure 2.4. In the next sections, we use them to simplify the map of complexity classes by collapsing some classes, as well as to separate the remaining classes by hardness propagation. Last section is devoted to the proofs of the negative closure properties.

Note that we have omitted some of the introduced complexity classes from Figure 2.4, namely RΔ, RN, ∪R1D, ∩L1D, ∩R1D, and ∩21D. Nevertheless, the closure properties of all these classes follow easily from the closure properties presented in Figure 2.4: As we show later, it holds that RΔ = SΔ and RN = SN. Furthermore, the closure properties of any class co-$\mathfrak{C}$ and re-$\mathfrak{C}$ are related to the closure properties of $\mathfrak{C}$, as described by the following observations and lemmas. Due to Lemma 2.2, the closure properties of the omitted classes of parallel automata follow.

The following observation is a straightforward corollary of Observation 2.1:

Observation 2.2. *Let $\mathcal{O}$ be any operator used in Figure 2.4, and $\mathfrak{C}$ any class of language families. Then* re-$\mathfrak{C}$ *is closed under $\mathcal{O}$ iff $\mathfrak{C}$ is closed under $\mathcal{O}$.*

Lemma 2.15. *Let $\mathfrak{C}$ be any class used in Figure 2.4. If $\mathfrak{C}$ is closed under $\oplus$ or $\bigoplus$, then $\mathfrak{C}$ is closed under complement.*

Proof. Fix a class $\mathfrak{C}$ of language families solvable by small families of XFAs, where X represents any automata model corresponding to the classes in Figure 2.4, and assume that $\mathfrak{C}$ is closed under $\bigoplus$. Let $\mathcal{L} = \{L_n\}_{n\geq 1} \in \mathfrak{C}$. The language family $\bigoplus\mathcal{L}$ is solvable by a family $\mathcal{M} = \{M_n\}_{n\geq 1}$ of small XFAs. We need to prove that $\overline{\mathcal{L}} \in \mathfrak{C}$, i. e., to find a small family $\mathcal{M}' = \{M'_n\}_{n\geq 1}$ of XFAs solving $\{\overline{L_n}\}_{n\geq 1}$.

Now we discuss how to construct automaton M'_n solving $\overline{L_n}$ from automaton M_n solving $\bigoplus L_n$. The case of $L_n = \emptyset$ is trivial, so assume that L_n is not empty and fix some $w \in L_n$. Easily, for any word u it holds that $u \notin L_n$ iff $\#w\#u\# \in \bigoplus L_n$. Hence, it is sufficient that M'_n, given an input u, simulates M_n on input $\#w\#u\#$. I. e., automaton M'_n simulates M_n on every symbol of the input word except on the endmarkers. On $\vdash$, M'_n simulates the behavior of M on $\vdash\#w\#$ and on $\dashv$, M'_n simulates M on $\#\dashv$. It is straightforward to verify that such a simulation is possible for all considered models of automata by adding only a constant number of new states (in case of parallel automata, a constant number of new states in each component). Hence, this construction transforms k-state automata into $O(k)$ state automata, and thus the resulting automata family $\mathcal{M}'$ is small.

The proof for $\oplus$ is analogous; it is sufficient to consider family $\mathcal{L} \oplus \mathcal{L}$ instead of $\bigoplus\mathcal{L}$. □

Lemma 2.16. *Let $\mathfrak{C}$ be any class used in Figure 2.4. Class* co-$\mathfrak{C}$ *is closed under $\overline{\cdot}$ ($\cdot^{\mathrm{R}}$, $\wedge$, $\vee$, $\oplus$, $\bigwedge$, $\bigvee$, $\bigoplus$) iff $\mathfrak{C}$ is closed under $\overline{\cdot}$ ($\cdot^{\mathrm{R}}$, $\vee$, $\wedge$, $\oplus$, $\bigvee$, $\bigwedge$, $\bigoplus$), respectively.*

Proof. The claim for the complement is trivial. The claim for $\cdot^{\mathrm{R}}$ follows directly from Observation 2.1. The claim for $\wedge$, $\vee$, $\bigwedge$, and $\bigvee$ follows from Lemma 2.1, and the fact that all classes in Figure 2.4 are closed under union and intersection with any language family from 1D, what is not difficult to verify by straightforward constructions.

If $\mathfrak{C}$ is closed under $\oplus$ ($\bigoplus$), Lemma 2.15 implies that $\mathfrak{C} =$ co-$\mathfrak{C}$. Hence co-$\mathfrak{C}$ is also closed under $\oplus$ ($\bigoplus$), respectively. The same argument can be used to show that if co-$\mathfrak{C}$ is closed under $\oplus$ ($\bigoplus$), so is $\mathfrak{C}$. □

2.4.1 Positive Closure Properties

All of the positive closure properties in Figure 2.4 can be either proven by straightforward constructions or were proven before. Since none of these constructions is hard, we describe only the main ideas.

Complement. At first we focus on the closures under complement, i. e., row 1 of Figure 2.4. Obviously, any self-verifying class is closed under complement. The closure of 2D under complement was proven in [Sip78] and (an improved construction) [GMP07]. The proof of closure of 1D under complement is very easy – it is sufficient to make all non-final states final and vice versa.

It is not difficult to see that any rotating or sweeping automaton can be modified into an equivalent one that avoids infinite runs: Indeed, any computation of a RDFA (SDFA) with k states consists of at most k (left) traversals. For the case of nondeterministic automata, a shortest accepting computation (if there exists some) consists of at most k (left) traversals. Hence, the modified automaton can count the number of traversals and terminate after reaching the upper limit. Since a k-state automaton is transformed into an $O(k^2)$-state one, a small family of automata is transformed into a small family.

The closure of RD and SD under complement follows by straightforward negation of the answer of the corresponding automaton that avoids infinite runs.

Reverse. Easily, any k-state two-way automaton accepting language L can be transformed into a $(k+1)$-state automaton of the same kind accepting language L^{R}: at first the new automaton moves its head to the right endmarker, and then simulates the original automaton with swapped directions of moves.

A similar construction works for sweeping automata: In the first traversal, the head is moved to the right endmarker, then the original automaton is simulated. For a two-sided parallel automaton $M = (\mathfrak{L}, \mathfrak{R}, F)$, it is enough to swap $\mathfrak{L}$ with $\mathfrak{R}$.

It is well known that 1N is closed under reverse. Here, the core idea is that a 1NFA "guesses" the computation of the simulated automaton backwards and, in every step, verifies if the guess is correct. For the model of 1NFAs without endmarkers, it has been proven in [HK03, Jir05] that any 1NFA with n states accepting language L can be converted into a 1NFA with $n+1$ states accepting L^{R} and that this bound is tight. It is, however, not difficult to check that, in our model of one-way automata (i. e., the model with endmarkers), any 1NFA can be reversed with no increase in the state complexity.

The closure of 1Δ under complement follows from the closure of 1N and Observation 2.1.

Parallel Automata. It is not difficult to check that parallel union automata are closed under both $\vee$ and $\bigvee$. Indeed, if M_1 and M_2 are parallel union automata for languages L_1, L_2, we can modify all components of M_1 to consider only the left #-delimited block and all components of M_2 to consider only the right #-delimited block. Afterwards, the union of all components of M_1 and M_2 forms the automaton for $L_1 \vee L_2$. For $\bigvee L_1$, it is sufficient to modify every component of M_1 to work on all #-delimited blocks separately and exit into the final state iff the original component exited into the final state on at least one block.

Proving that $\cup_{\text{L}}$1D is closed under $\wedge$ is only slightly more involved. Let $M_1 = (\mathfrak{L}_1, \emptyset, F_1)$ and $M_2 = (\mathfrak{L}_2, \emptyset, F_2)$ be $\cup_{\text{L}}$1DFAs with at most k states accepting languages L_1, L_2. We construct a $\cup_{\text{L}}$1DFA M' accepting language $L_1 \wedge L_2$ with at most k^2 components, each with $O(k)$ states: for every pair $C_1 \in \mathfrak{L}_1$, $C_2 \in \mathfrak{L}_2$, there is one component D of M' that simulates C_1 on the first #-delimited block of input word and C_2 on the second one. Component D exits into a final state iff both C_1 and C_2 do. It is easy to see that D can be constructed with $O(k)$ states. Furthermore, for every $x \in L_1$ and $y \in L_2$, some component of M' accepts #x#y#: the component consisting of $C_1 \in \mathfrak{L}_1$ accepting x and $C_2 \in \mathfrak{L}_2$ accepting y does. On the other hand, if some component of M' accepts #x#y#, then $x \in L_1$ and $y \in L_2$. Hence, the construction is correct.

Remaining Closure Properties. It remains to show positive closure properties of rows 3–8, columns A–C,E,G–K of Figure 2.4, i. e., the closure properties of one-way, rotating, sweeping and two-way automata under $\wedge$, $\vee$, $\oplus$, $\bigwedge$, $\bigvee$, $\bigoplus$.

All these properties can be proved using straightforward constructions based on the same idea: The newly constructed automaton simulates the original automaton on each #-delimited block of the input word separately, and decides according to the results of this simulation. To do so, the new automaton needs enough movement power.

It is easy to see that both one-way and two-way automata can simulate automata of the same type on each #-delimited block of the input word, regardless of the number of such blocks. Rotating and sweeping automata, however, are not able to do so for inputs consisting of many blocks, since they cannot "remember" the position of the processed block and return to it in another traversal. Nevertheless, if the input consists of two blocks only, they are able to simulate another automaton on the left (or right) block only. Hence, rotating and sweeping automata have sufficient movement power for proving closures of $\wedge$, $\vee$, and $\oplus$ only, while one-way and two-way automata have sufficient power also for $\bigwedge$, $\bigvee$, and $\bigoplus$.

The ability to simulate an automaton of the same kind on each #-delimited block is enough to prove positive closure properties at rows 3,6 and columns A–C,E,G–K in Figure 2.4, i. e., the closure properties related to $\wedge$ and $\bigwedge$. The same technique can be applied in a straightforward way for properties related to $\vee$ and $\bigvee$ if the simulated automaton never reaches an infinite loop. Infinite loops can be always avoided with polynomial blowup in state complexity for one-way automata (trivially), rotating and sweeping automata (as we have discussed in the paragraph about the complement), as well as for two-way deterministic automata [Sip78]. In this way, it is possible to obtain a proof for positive closure properties at rows 4,7 and columns A–C,E,G–I in Figure 2.4. Hence, only the case of 2N and 2Δ remains to be considered: A 2NFA accepting $\bigwedge L$ can nondeterministically choose one block of the input word and simulate the corresponding 2NFA accepting L. Here, a possible infinite loop of the simulated machine does not pose a problem. A two-way self-verifying automaton accepting $\bigwedge L$ can either verify that the input word is in the accepted language in the

same way as 2NFA, or verify that the word is not in the language in the same way as a 2NFA for $\bigwedge \overline{L}$. An analogous construction works also for $L_1 \wedge L_2$.

To use our technique for closure properties under $\oplus$ and $\bigoplus$, the new automaton needs the ability to negate the answer of the simulated automaton. This is trivial for deterministic classes and can be done easily for all classes closed under complement: the new automaton simulates both the original automaton and the complemented original automaton on every block. It is not difficult to see that the possibility of infinite loops does not pose a problem for two-way self-verifying automata, since the new automaton can, for every block of the input, nondeterministically guess if this block is in the corresponding language and verify this guess by running the simulated machine. Hence, the closure properties in row 5,8 and columns A–C,E,G–K are correct, too.

2.4.2 Collapsing Classes

Now we are ready to augment the map in Figure 1.5 with complements and reverses of all of the classes and draw a new map with collapsed classes. To do so, we use the positive closure properties in Figure 2.4: 1N = re-1N, 1Δ = co-1Δ = re-1Δ, 1D = co-1D, RD = co-RD, SD = co-SD = re-SD, SΔ = co-SΔ = re-SΔ, SN = re-SN, 2D = co-2D = re-2D, 2Δ = co-2Δ = re-2Δ, and 2N = re-2N. Furthermore, we use the following lemma to show that RΔ = SΔ and RN = SN.

Lemma 2.17. *Let $M = (q_s, \delta, q_a)$ be a* SNFA *over a set of k states Q solving language L over alphabet Σ. Then there exists an $O(k^3)$-state* RNFA *M' solving L.*

Proof. We construct M' in the following way: every left-to-right traversal of M is simulated by M' in a straightforward way. To simulate a right-to-left traversal of M, automaton M' guesses the computation of M backwards, in a similar way as in the proof that 1N is closed under reverse. While doing so, M' needs to remember the starting state of the right-to-left traversal (which is checked at the end of the traversal simulation), and the guessed last state of the right-to-left traversal (from which the next left-to-right traversal is started).

More formally, $M' = (q_s, \delta', q_a)$ is an automaton over set of states

$$Q' = \{q, (q_R, q, q_L) \mid q, q_L, q_R \in Q\}$$

such that δ' is defined as follows:

$$\begin{array}{rcll}
\delta'(q,a) & = & \delta(q,a) & \forall a \in \{\vdash\} \cup \Sigma \\
\delta'(q,\dashv) & \ni & q_a & \Leftrightarrow \ \delta(q,\dashv) \ni q_a \\
\delta'(q,\dashv) & \ni & (q,q',q') & \forall q,q' \in Q \\
\delta'((q,w,q'),a) & \ni & (q,w',q') & \Leftrightarrow \ \delta(w',a) \ni w, a \in \Sigma \\
\delta'((q,w,q'),\dashv) & = & \delta(q',\vdash) & \Leftrightarrow \ \delta(q,\dashv) \ni w \\
\delta'((q,w,q'),\dashv) & = & \emptyset & \text{otherwise}
\end{array}$$

States from Q are used for simulation of left-to-right traversals. At the end of the traversal, the current state is saved as the first component of the state (q, q', q'), and

the state at the end of the right-to-left traversal q' is guessed nondeterministically. Afterwards, the computation of M is guessed nondeterministically backwards. Hence, $\text{LCOMP}_{M',(q,q',q')}(z)$ can exit into (q, w, q') if and only if $\text{RCOMP}_{M,w}(z)$ can exit into q'. So, M' can avoid hanging at $\dashv$ iff M could traverse the input word from right to left starting at state q, continuing with w, and exiting into q'. In that case, M' simulates movement of M' on $\vdash$ and continues with the simulation of a left-to-right traversal. □

Combination of the above-mentioned results yields the map of the complexity classes depicted in Figure 2.5. It is possible to add the classes of union and intersection parallel automata to the figure, too. Nevertheless, for the sake of clarity we present them separately in Figure 2.6, which "zooms" into these classes.

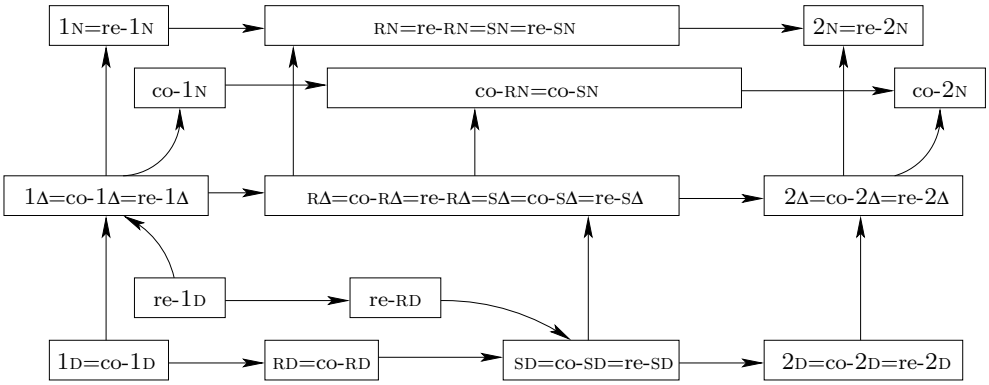

Figure 2.5: Complexity classes of finite automata. An arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \subseteq \mathfrak{C}'$.

In fact, Figure 2.6 augments Figure 2.1 with the relationships to the classes of one-way automata: Since 1D is closed under complement and every 1DFA is a special case of a 1NFA, we have that $1\text{D} \subseteq 1\Delta$. Furthermore, it is easy to see that $\cup_2 1\text{D} \subseteq 1\text{N}$: Any $\cup_2$1DFA can be simulated by a 1NFA that nondeterministically selects one of the components of the parallel automaton and simulates it. Because every 1DFA is also a 1NFA and every 1NFA can be reversed without any increase of the state complexity, this construction transforms a small family of $\cup_2$1DFAs into a small family of 1NFAs. The remaining relationships added in Figure 2.6 follows from Observation 1.1, because 1Δ is closed under complement.

2.4.3 Separating Classes

In this section, we provide results separating the state-complexity classes of finite automata. More precisely, we prove the results displayed in Figure 2.8. As we show later, these separations allow us to compare any two classes in Figures 2.5 and 2.6 except for those corresponding to two-way automata.

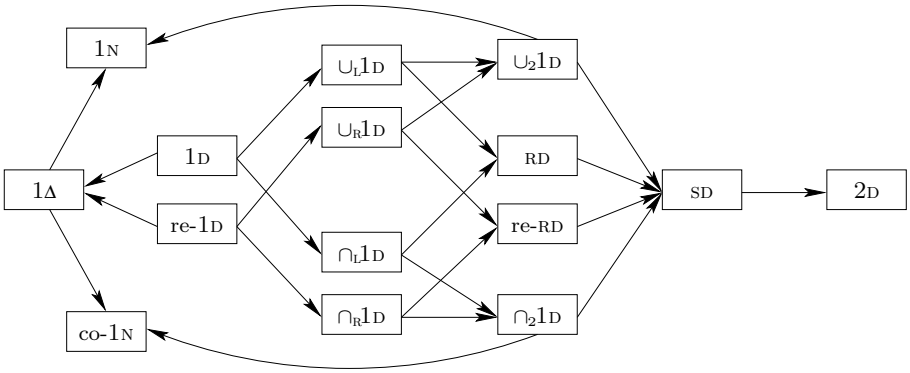

Figure 2.6: Complexity classes of finite automata, zoom to the parallel automata classes. An arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \subseteq \mathfrak{C}'$. Note that all complements and reverses of classes corresponding to parallel automata are already included in the map due to Lemma 2.2.

Theorem 2.18. *Let $\mathfrak{C}_1$, $\mathfrak{C}_2$ be any two classes among those in Figures 2.5, 2.6 distinct from those corresponding to two-way automata. Then $\mathfrak{C}_1 \subsetneq \mathfrak{C}_2$ iff there exists a path from $\mathfrak{C}_1$ to $\mathfrak{C}_2$ in Figures 2.5, 2.6. Otherwise, $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are incomparable.*

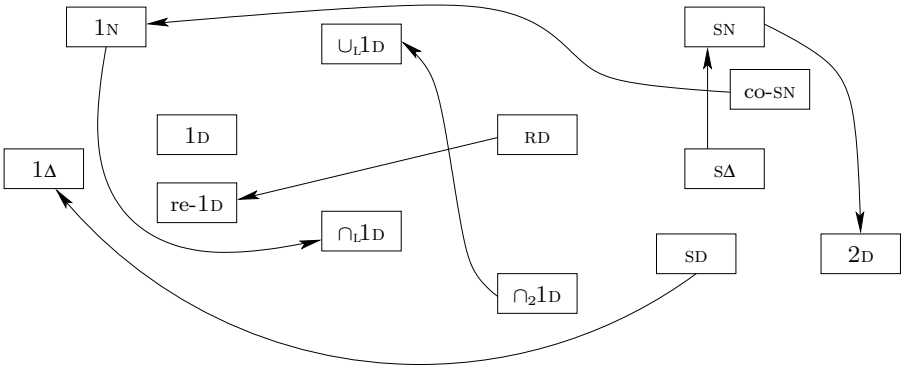

Figure 2.7: Separations of the complexity classes. An arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \not\supseteq \mathfrak{C}'$.

To prove Theorem 2.18, we use the separation results depicted in Figure 2.7. We prove several of them using the technique of hardness propagation on a core language family $\mathcal{J} = (J_n)_{n \geq 1}$ defined as

$$J_n := \{\alpha i \mid \alpha \subseteq [n] \text{ and } i \in \alpha\}, \tag{2.9}$$

where $[n] := \{1, \ldots, n\}$. The basic membership properties of $\mathcal{J}$ are stated in the following lemma:

Lemma 2.19. *$\mathcal{J} := (J_n)_{n\geq 1}$ is not in* 1D *but is in* re-1D, 1N, co-1N, $\cap_{\mathrm{L}}$1D, $\cup_{\mathrm{L}}$1D.

Proof. Any 1DFA solving J_n needs at least 2^n states: Let M be any 1DFA solving J_n. It is easy to verify that if M is in the initial state and reads symbol $\alpha \subseteq [n]$, then the state reached by M must be different for every α.

On the other hand, it is not difficult to verify that J_n can be solved by a 1NFA with $n+2$ states, by a $\cap_{\mathrm{L}}$1DFA with n components of 4 states each, and by a $\cup_{\mathrm{L}}$1DFA with n components of 4 states each. Also, $(J_n)^{\mathrm{R}}$ can be solved by a 1DFA with $n+2$ states and $\neg J_n$ can be solved by a 1NFA with $n+2$ states. □

Now we are ready to prove all results in Figure 2.7:

Lemma 2.20. *All facts in Figure 2.7 are correct.*

Proof.

- RD $\not\supseteq$ re-1D: $\mathcal{L} := \bigwedge\bigvee\mathcal{J}$ is the witness. Due to Lemma 2.19, $\mathcal{J} \notin$ 1D, so Corollary 2.5 yields that $\bigvee\mathcal{J} \notin \cap_{\mathrm{L}}$1D and Corollary 2.12 ensures that $\mathcal{L} = \bigwedge\bigvee\mathcal{J} \notin$ RD. Since $\mathcal{J}^{\mathrm{R}} \in$ 1D (Lemma 2.19), $\bigvee\mathcal{J}^{\mathrm{R}} \in$ 1D (A7 of Figure 2.4), hence A6 of Figure 2.4 yields that $\bigwedge\bigvee\mathcal{J}^{\mathrm{R}} \in$ 1D and, due to Observation 2.1, $\mathcal{L} \in$ re-1D.
- $\cap_2$1D $\not\supseteq \cup_{\mathrm{L}}$1D: $\mathcal{L} := (\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J})^{\mathrm{R}}$ is the witness. Since $\bigvee\mathcal{J} \notin \cap_{\mathrm{L}}$1D, applying Corollary 2.7 yields that $\mathcal{L} \notin \cap_2$1D. Since $\mathcal{L} = (\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J}^{\mathrm{R}})$ (Observation 2.1), $\mathcal{J}^{\mathrm{R}} \in$ 1D $\subseteq \cup_{\mathrm{L}}$1D (Lemma 2.19, Figure 2.6), and $\mathcal{J} \in \cup_{\mathrm{L}}$1D (Lemma 2.19), D7 and D4 of Figure 2.4 yields that $\mathcal{L} \in \cup_{\mathrm{L}}$1D.
- SD $\not\supseteq$ 1Δ: $\mathcal{L} := \bigwedge\big((\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J})^{\mathrm{R}}\big)$ is the witness. Since $(\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J})^{\mathrm{R}} \notin \cap_2$1D, applying Corollary 2.14 yields that $\mathcal{L} \notin$ SD. Since $(\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J})^{\mathrm{R}} \in \cup_{\mathrm{L}}$1D $\subseteq$ 1N, C6 of Figure 2.4 yields that $\mathcal{L} \in$ 1N. It remains to show that $\mathcal{L} \in$ co-1N. Since 1N is closed under union and intersection with any family from 1D, we can use Lemma 2.19, Lemma 2.1, Observation 2.1, and C2, C3, C6, C7 of Figure 2.4 to derive the following: $\overline{\mathcal{J}} \in$ 1N, $\bigwedge\overline{\mathcal{J}} \in$ 1N, $\overline{\bigvee\mathcal{J}} \in$ 1N, $\overline{\bigvee\mathcal{J}}^{\mathrm{R}} \in$ 1N, $\overline{\bigvee\mathcal{J}^{\mathrm{R}}} \in$ 1N, $\overline{\bigvee\mathcal{J}} \wedge \overline{\bigvee\mathcal{J}^{\mathrm{R}}} \in$ 1N, $\overline{(\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J}^{\mathrm{R}})} \in$ 1N, $\bigvee(\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J}^{\mathrm{R}}) \in$ 1N, $\overline{\mathcal{L}} = \overline{\bigwedge((\bigvee\mathcal{J}) \vee (\bigvee\mathcal{J}^{\mathrm{R}}))} \in$ 1N.
- 1N $\not\supseteq \cap_{\mathrm{L}}$1D: Language family $\mathcal{D} = (D_n)_{n\geq 1}$, representing the *disjointness problem*, defined as
$$D_n = \{\alpha\beta \mid \alpha, \beta \subseteq [n], \alpha \cap \beta = \emptyset\}, \tag{2.10}$$
witnesses this separation. To prove that any 1NFA M solving D_n needs at least 2^n states, consider accepting computations of M on words from the set $\{\alpha\overline{\alpha}\}$, where $\alpha \subseteq [n]$ and $\overline{\alpha} = [n] - \alpha$. If M has less than 2^n states, there exist $\alpha_1 \neq \alpha_2$ such that M is in the same state when reading the second symbol of the word $\alpha_1\overline{\alpha_1}$ and of the word $\alpha_2\overline{\alpha_2}$. Without loss of generality, we may assume that α_1 and α_2 differ in element x, more precisely that $x \in \alpha_1$ and $x \notin \alpha_2$. Then M accepts the word $\alpha_1\overline{\alpha_2}$, which is a contradiction, since $x \in \alpha_1 \cap \overline{\alpha_2}$. Furthermore, it is not difficult to construct a $\cap_{\mathrm{L}}$1DFA with n components of 4 states each that solves J_n: for each $i \in [n]$, there is one component that verifies if $i \notin \alpha$ or $i \notin \beta$ for the input word $\alpha\beta$.

- sΔ $\not\supseteq$ sn: Since sΔ is closed under complement and sn is not [Kap06], sΔ $\neq$ sn. Since sΔ $\subseteq$ sn, the claim follows.
- co-sn $\not\supseteq$ 1n: Proven in [Kap06].
- sn $\not\supseteq$ 2d: Proven in [Kap06].

□

The claim of Theorem 2.18 easily follows from the results gathered in Figures 2.5, 2.6, and 2.8:

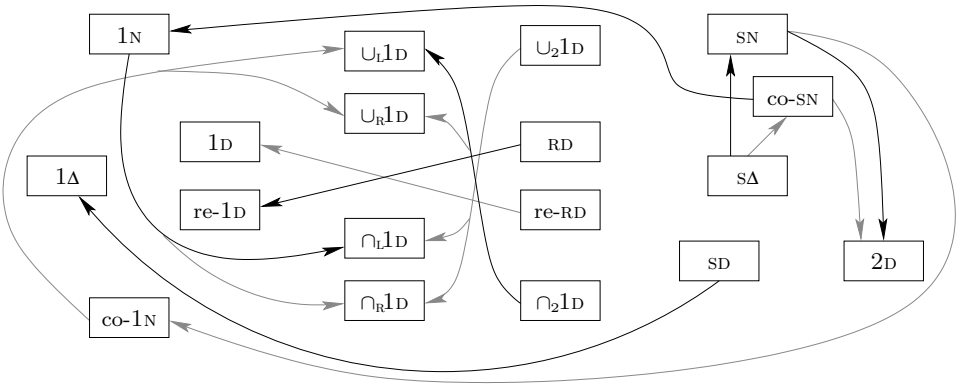

Figure 2.8: Separations of complexity classes, all relationships. An arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \not\supseteq \mathfrak{C}'$.

Proof of Theorem 2.18. All separations in Figure 2.8 follows easily from those in Figure 2.7 using Observation 1.1(5, 6).

Any two classes of one-way, parallel, rotating and sweeping automata considered in the claim of Theorem 2.18 can be compared using the facts in Figures 2.5, 2.6, and 2.8, using Observation 1.1(7).

For example, to prove that 1n and $\cap_R$1d are incomparable, we have that $\cap_R$1d $\subseteq$ $\cap_2$1d $\subseteq$ sd $\not\supseteq$ 1Δ $\subseteq$ 1n, and 1n $\not\supseteq$ $\cap_R$1d. Any other relationship can be proven in a similar way. □

So far, we have proven a complete characterization of the relationship of all introduced complexity classes, except of those corresponding to the two-way automata. Nevertheless, using the same arguments as in the proof of Theorem 2.18, it is possible to derive a separation between two-way and sweeping automata: sd $\subsetneq$ 2d, sΔ $\subsetneq$ 2Δ, and sn $\subsetneq$ 2n.

2.4.4 Negative Closure Properties

In this section, we prove the correctness of all negative closure properties claimed in Figure 2.4.

Reverse and Complement. All negative closure properties under reverse and complement follow directly from Theorem 2.18.

Parity Operators. All negative closure properties under $\oplus$ and $\bigoplus$ (i. e., at rows 5,8 of Figure 2.4) follow easily from the negative closure properties under complement. Indeed, Lemma 2.15 implies that any considered class of finite automata that is not closed under complement is not closed neither under $\oplus$ nor under $\bigoplus$.

Remaining Properties. The remaining properties are easy to prove by using previous results:

- **F6:** Due to Theorem 2.18, there exists $\mathcal{L} \in \cup_2 1\text{D} - \cap_2 1\text{D}$. Corollary 2.14 yields that $\bigwedge\mathcal{L} \notin \text{SD}$, hence $\bigwedge\mathcal{L} \notin \cup_2 1\text{D}$.
- **D6:** Due to Lemma 2.16 and Lemma 2.2, this is equivalent to proving that $\cap_{\text{L}} 1\text{D}$ is not closed under $\bigvee$. Due to Theorem 2.18, there exist $\mathcal{L} \in \cap_{\text{L}} 1\text{D} - 1\text{D}$. Corollary 2.5 yields that $\bigvee\mathcal{L} \notin \cap_{\text{L}} 1\text{D}$.
- **F3:** For similar reasons as in D6, it is sufficient to prove that $\cap_2 1\text{D}$ is not closed under $\vee$. Due to Theorem 2.18, there exist $\mathcal{L} \in \cap_2 1\text{D} - \cap_{\text{L}} 1\text{D}$. Since $\cap_2 1\text{D}$ is closed under reverse, $\mathcal{L}^{\text{R}} \in \cap_2 1\text{D}$. Due to Corollary 2.7, $\mathcal{L} \vee \mathcal{L}^{\text{R}} \notin \cap_2 1\text{D}$.
- **E6, G6:** Due to Theorem 2.18, there exist $\mathcal{L} \in \text{RD} - \cap_{\text{L}} 1\text{D}$ ($\mathcal{L} \in \text{SD} - \cap_2 1\text{D}$), respectively. Due to Corollary 2.12 (2.14), it holds that $\bigwedge\mathcal{L} \notin \text{RD}$ ($\bigwedge\mathcal{L} \notin \text{SD}$), respectively.
- **E7, G7:** Since RD and SD are closed under complement, the claim follows from E6, G6 and Lemma 2.16.

2.5 Parallel Automata Classes

It is possible to define classes of language families recognized by small families of general parallel automata (P_{L}1DFAs, P_{R}1DFAs, $\text{P}_2$1DFAs). Following our naming convention, we denote these classes as $\text{P}_{\text{L}}1\text{D}$, $\text{P}_{\text{R}}1\text{D}$, and $\text{P}_2 1\text{D}$.

Up to now, we have not discussed these complexity classes, even though we have provided results about complexity of general parallel automata (lower bounds in Lemma 2.11 and Lemma 2.13). The reason for this is that the complexity classes of general parallel automata are rather unnatural from our point of view: Consider a parallel automaton M with k components of at most m states each. The description of such an automaton contains the set of accepting k-tuples of M, and this set can be an arbitrary subset of all m^k possible tuples. Hence, it might be necessary to use exponential number of bits to describe a small automaton. E. g., if $m = k$, the automaton has only m^2 states, but it is necessary to use m^m bits to describe the set of accepting tuples.

The descriptional complexity of general parallel automata can be exponential in the number of states. This holds even if we consider only automata over the binary alphabet. Hence, it is not surprising that the complexity classes of these automata do not fit nicely into the map we have presented in this chapter. On one hand, it is possible to prove that P_L1D $\not\supseteq$ re-1D and $\text{P}_2$1D $\not\supseteq$ 1Δ. The proof goes in the same way as the proof of RD $\not\supseteq$ re-1D and SD $\not\supseteq$ 1Δ in Lemma 2.20; this is possible because Corollary 2.12 and Corollary 2.14 can be formulated for classes P_L1D and $\text{P}_2$1D, too. On the other hand, P_L1DFAs can be very powerful. As indicated by the following lemma, small P_L1DFAs can accept even some language families not in 2N.

Lemma 2.21. 2N $\not\supseteq$ P_L1D

Proof. Consider all possible languages containing only binary words of length n. Let $L_n \subseteq \{0,1\}^n$ be one of these languages that is most difficult for 2NFA, i. e., the one that requires a 2NFA with highest number of states. We claim that the language family $\mathcal{L} := \{L_n\}_{n\geq 1}$ witnesses the claim of the lemma.

It is easy to see that L_n can be accepted by an n-component P_L1DFA with $O(n)$ states per component: The i-th component reads the input word and remembers its i-th symbol. In this way, complete information about the input is stored in the tuple of the result states. Hence, any $L \subseteq \{0,1\}^n$ can be accepted by a P_L1DFA with $O(n^2)$ states.

On the other hand, assume that $\mathcal{L} \in$ 2N. Then, there is some polynomial $p(n)$ such that, for every n, there is a 2NFA with $p(n)$ states accepting L_n. By the choice of L_n, any $L \subseteq \{0,1\}^n$ can be accepted by a 2NFA with $p(n)$ states. There are, however, 2^{2^n} different languages that are subsets of $\{0,1\}^n$, but there are only $2^{O\left(p^2(n)\right)}$ different 2NFAs with $p(n)$ states. Hence, we have a contradiction for large enough n. □

Putting together these results with Theorem 2.18 yields that P_L1D is a strict superset of 1D, $\cap_\text{L}$1D, $\cup_\text{L}$1D, and RD and it is incomparable with all other classes in Figures 2.5 and 2.6. Similarly, $\text{P}_2$1D is a strict superset of SD and all classes included in SD and it is incomparable with all other classes in Figures 2.5, 2.6.[1]

We used the notion of parallel automata as an intermediate step in the hardness propagation. Consistently with our goals, we defined a family of a parallel automata to be small if the automata contain only polynomial number of components with polynomial number of states each. There is an alternative definition to this, used in [KKM08], which relaxes the constraint on the number of components. In this way, small parallel automata must only have polynomially small components, the number of components is not relevant.

All hardness propagation results presented in this chapter hold also for this alternative definition of parallel automata, what is in fact a stronger result than the one we presented. Nevertheless, we opted not to consider these alternative complexity classes.

[1] This claim follows easily from Theorem 2.18 for all classes except 2D. For 2D, it is trivial to adapt the proof of $\text{P}_2$1D $\not\supseteq$ 1Δ to $\text{P}_2$1D $\not\supseteq$ 2D.

The reason for this is similar to the reason why we did not include the classes of general parallel automata into the map of complexity classes. The alternative classes of parallel automata are rather unnatural, as they correspond to automata with possibly large descriptional complexity; this problem arises even for parallel intersection and parallel union automata. As a consequence, such alternative classes do not fit easily in the hierarchy of classes that we have built; e. g., it is an open problem if a parallel automaton with exponentially many small components can be simulated by a small sweeping automaton.

On the other side, separating complexity classes of both variants of parallel automata pose an interesting open problem. Since this apparently cannot be done using the technique of generic words, completely new technique for proving lower bounds on parallel automata seems to be necessary to achieve the separation.

Chapter 3

Randomization

So far, we have dealt only with deterministic and nondeterministic models of finite automata. It is possible, however, to define also randomized models of automata, in a similar way as it is done for Turing machines. In this chapter, we focus on these randomized models.

Essentially, a randomized automaton (sometimes called also probabilistic automaton) is just a nondeterministic automaton that makes its nondeterministic choices according to some probability distribution. More precisely, the difference between a nondeterministic automaton and a randomized automaton of the corresponding type is as follows. Consider a nondeterministic automaton over a set of states Q and alphabet Σ. The transition function δ of this automaton partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to the set of all possible subsets of feasible actions A. The action of a one-way, rotating, and sweeping automata is completely described by the new state, hence $A = Q$ in this case. For two-way automata, the action consists of the new state and the movement, hence $A = Q \times \{-1, 0, 1\}$ for two-way automata. The transition function δ of a randomized machine partially maps $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to the set of all *probability distributions* over the set of actions A, i. e., all total functions from A to the real numbers that obey the axioms of probability. Hence, on any input word $z \in \Sigma^*$, the computation of M on z is a *probability distribution* over all possible computations. The expected length of a computation drawn from this distribution is called the *expected running time* of M on z. Easily, this way of defining a randomized automaton applies for one-way, rotating, sweeping, as well as for two-way automata.

Randomized automata are sometimes required to use only rational numbers in their transition functions (see e. g. [Wan92]). In some sense, this is more natural than allowing arbitrary real numbers, since real numbers might have infinite descriptional complexity. Indeed, as we note later (and as shown in [dLMSS56] for a more general computation model), certain models of finite automata can accept even languages that are not recursively enumerable if arbitrary real probabilities can be used.

Nevertheless, the results presented in this chapter are robust with regard to the restriction to rational probabilities: While all upper bounds are achievable with the

(potentially weaker) models with rational probabilities, all lower bounds are valid also for the (potentially stronger) models with real probabilities. Since lower bounds on state complexity of randomized automata is our main focus in this chapter and some of our lower bound proofs require the use of real numbers, we use the more general definition of randomized automata that allows to use arbitrary real numbers in the transition function.

Even more restricting variant of definition of randomized automata is sometimes used (see e. g. [Fre75, Gil77]). Here, the automaton is required to use *fair coin flips* only, i. e., the source of randomness provides only a sequence of random bits. Nevertheless, it is not difficult to check that any rational probability can be generated by using several coin flips (see e. g. [Con01]), hence the models with coin flips and with rational probabilities are computationally equivalent.[1] This might be not the case for the equivalence of complexity classes: A straightforward conversion of a small family of automata using rational probabilities might yield a family of coin-flipping automata that is not small. No separation results are, however, known for these two models. Nevertheless, all relationships between complexity classes presented in this chapter are robust in this respect. I. e., all presented relationships hold for automata using real probabilities, for automata using rational probabilities, as well as for coin-flipping automata.

In this chapter, we introduce various models of randomized finite automata and provide several results concerning the complexity classes of these models, what gives some insight into the power of randomized computations.

The next section is dedicated to the definition of the randomized models. In Section 3.2, we provide results concerning the complexity classes of the introduced models. Most of these results are rather straightforward implications of previous works, but together they augment the hierarchy of the complexity classes in an interesting way. Furthermore, several open problems are sketched here.

Results shown in Section 3.2 suggest that, for rotating, sweeping, and two-way automata, even LasVegas randomization is quite powerful. This fact, however, depends on exponential running time of the LasVegas automata. Hence, it is a natural question to ask if the exponential running time is indispensable for the power of the LasVegas machines. We address this problem in Section 3.4 for rotating automata. Here, we show that restricting rotating automata to linear expected running time significantly decreases their power. We further generalize this result for sweeping automata in Section 3.5.

[1] This is true if the randomized automata are allowed to make transitions without head movement (called also λ-transitions). Not allowing this is rather unnatural, since the amount of randomness accessible by the automaton would be very restricted in such a case. On the other side, it is possible to prove that any coin-flipping randomized automaton with λ-transitions can be simulated by a corresponding randomized automaton with rational probability distributions without λ-transitions and that this can be done with no increase in the state complexity. The proof can be done by using the theory of Markov chains in a straightforward way; it is sufficient to prove that a coin-flipping automaton can generate only rational probabilities within λ-transitions.

3.1 Randomized Models

As we have already mentioned, a randomized automaton is essentially a nondeterministic automaton that follows each nondeterministic choice with certain probability; these probabilities are part of the transition function δ. Hence, the last missing part of the definition of randomized automata is to define which words are accepted by the automaton. There are several ways of doing that, yielding several different models of randomized automata.

3.1.1 Monte-Carlo Automata with Two-Sided Error

The most general way of defining the language accepted by a randomized finite automaton M is to define the language as the set of all words accepted by M with probability greater than $1/2$. This definition applies to one-way, rotating, sweeping, as well as two-way randomized finite automata. We call the resulting computation models as *one-way (rotating, sweeping, two-way) Monte-Carlo finite automata with two-sided unbounded error* and denote them as $1\mathrm{P}_2^{\mathrm{U}}\mathrm{FAs}$ ($\mathrm{RP}_2^{\mathrm{U}}\mathrm{FAs}$, $\mathrm{SP}_2^{\mathrm{U}}\mathrm{FAs}$, $2\mathrm{P}_2^{\mathrm{U}}\mathrm{FAs}$).

One-way Monte-Carlo automata with two-sided error were introduced in [Rab63], as the first randomized model of finite automata. Here, even more general definition was used, allowing arbitrary constant α instead of $1/2$ as the cut-point. Nevertheless, as already explained in Chapter 1, this makes no difference to the computational power. Indeed, any Monte-Carlo finite automaton with two sided unbounded error defined with cut-point α can be simulated by a finite automaton of the same type with cut-point $1/2$ at the cost of adding only constant number of states.[2]

In a similar way as for the non-randomized models, we can define the complexity classes induced by small families of one-way Monte-Carlo automata with two-sided unbounded error. Following our naming convention, we denote these classes as $1\mathrm{P}_2^{\mathrm{U}}$, $\mathrm{RP}_2^{\mathrm{U}}$, $\mathrm{SP}_2^{\mathrm{U}}$, and $2\mathrm{P}_2^{\mathrm{U}}$.

Monte-Carlo automata with two-sided unbounded error can be very powerful. Indeed, when they are not restricted to rational numbers in the transition function, even $1\mathrm{P}_2^{\mathrm{U}}\mathrm{FAs}$ can accept languages that are not recursively enumerable [Rab63]. (This is not very surprising, since there are uncountably many different automata.) Even when restricted to rational numbers, $1\mathrm{P}_2^{\mathrm{U}}\mathrm{FAs}$ can accept non-regular languages [Fre81]. This power, however, depends on the fact that the probability between accepting a word in the language and a word not in the language can be arbitrarily close. Hence, the model with unbounded error does not capture the notion of efficient computability.

To overcome this deficiency, the computation model with bounded error has been introduced in [Rab63] (called model with *isolated cut-point* there). In this model, the finite automaton M accepting a language L is required to accept all words in L with probability at least $1/2 + \varepsilon$ and accept all words not in L with probability

[2]This claim holds for automata using real probabilities. For automata restricted to rational probabilities, the claim holds only for rational cut-points. For coin-flip automata, the construction works for rational cut-points, but the number of states added depends on the value of the denominator of the cut-point.

at most $1/2 - \varepsilon$ for some fixed constant $\varepsilon > 0$.[3] Again, this definition applies to one-way, rotating, sweeping, as well as two-way randomized finite automata. We call the resulting computation models as *one-way (rotating, sweeping, two-way) Monte-Carlo finite automata with two-sided bounded error* and denote them as 1P$_2$FAs (RP$_2$FAs, SP$_2$FAs, 2P$_2$FAs). We call the parameter ε as *the error bound*.

It is possible to define the complexity classes corresponding to Monte-Carlo finite automata with two-sided bounded error in a straightforward way, i. e., as the classes of all language families solvable by a small family of the corresponding bounded-error automata. In such definition, however, the error bound ε may be different for different automata in the family. In this way, we introduce classes $1\mathrm{P}_2^{\mathrm{N}}$, $\mathrm{RP}_2^{\mathrm{N}}$, $\mathrm{SP}_2^{\mathrm{N}}$, and $2\mathrm{P}_2^{\mathrm{N}}$ corresponding to one-way, rotating, sweeping, and two-way Monte-Carlo finite automata with two-sided bounded error. We use the superscript N to emphasize that the error is bounded *nonuniformly*, i. e., the parameter ε may decrease arbitrarily fast in the family of small automata.

As opposed to the model with unbounded error, the model with bounded error captures the notion of computability. Indeed, using the technique of amplification (we show more details about how to use amplification with finite automata later in Lemma 3.1), arbitrary small error can be achieved. In [Rab63] it was proven that all languages accepted by 1P$_2$FAs are regular, even without the restriction to rational numbers in transition function. However, the nonuniformity of the error bound may cause that the use of the amplification technique causes huge blowup in space complexity, rendering it inefficient.

Even though the computation models with nonuniformly bounded error do not guarantee efficient solvability, they are quite interesting in the context of our work. These models fit very naturally in the lower bound proofs presented later. They are also the least restricted models for which the corresponding proof techniques work. Hence, using the models with nonuniformly bounded error gives us the strongest results achievable by these proof techniques.

The model of bounded-error Monte-Carlo computations is sometimes defined with some fixed error bound, e. g. $\varepsilon = 1/6$ (see e. g. [Hro05]). While the value of ε is usually irrelevant when dealing with the individual automata, using the definition with fixed ε yield different complexity classes. Here, the error bound ε is required to be the same for all automata in the family. We call the corresponding complexity classes as *Monte-Carlo with two-sided uniformly bounded error*, and denote them as $1\mathrm{P}_2^{\varepsilon}$, $\mathrm{RP}_2^{\varepsilon}$, $\mathrm{SP}_2^{\varepsilon}$, and $2\mathrm{P}_2^{\varepsilon}$. I. e., for every fixed $0 < \varepsilon < 1$, the class $1\mathrm{P}_2^{\varepsilon}$ contains all language families that are solvable by a family of small one-way Monte-Carlo automata with two-sided error bounded by ε; the definition for rotating, sweeping, and two-way automata is analogous.

For one-way, rotating, and sweeping automata, the complexity class induced by uniformly bounded two-sided Monte-Carlo is the same for any $\varepsilon \in (0, 1)$. Indeed, it

[3] Similarly as in the case of unbounded error, allowing arbitrary cut-point instead of $1/2$ does not increase the computational power of the model.

is possible to use the amplification technique, as described by the following lemma. Nevertheless, an analogous result for two-way automata is not known.

Lemma 3.1. *Consider arbitrary $\varepsilon_1, \varepsilon_2 \in (0,1)$. There exists an integer c that depends only on $\varepsilon_1, \varepsilon_2$ such that for any* $\mathrm{SP_2FA}$ *M_1 with k states working with error bound ε_1 there exists an equivalent* $\mathrm{SP_2FA}$ *M_2 with $O(k^c)$ states working with error bound ε_2. An analogous result holds for* $\mathrm{RP_2FAs}$ *and* $\mathrm{1P_2FAs}$.

Proof. We focus on the case of sweeping automata; our proof, however, works smoothly for rotating and one-way automata, too.

The basic idea is to use the method of probability amplification [Hro05]. More precisely, we construct an automaton M_2 that simulates c independent runs of M_1 and accepts if and only if a majority of the simulated runs of M_1 accepts. Formula (2.13) in [Hro05] ensures that if

$$c \geq \frac{2 \ln \delta}{\ln(1 - 4\varepsilon_1)}, \tag{3.1}$$

then M_2 gives the correct answer with probability at least $1 - \delta$. To guarantee that the automaton M_2 works with error bound ε_2, it is sufficient to choose $\delta := 1/2 - \varepsilon_2$. Hence, M_2 has to simulate c independent runs of M_1 for

$$c \geq \frac{2 \ln(\frac{1}{2} - \varepsilon_2)}{\ln(1 - 4\varepsilon_1)}. \tag{3.2}$$

The simulation of c independent runs can be done in parallel. Automaton M_2 consists of Cartesian product of c independent copies of M_1. In every computation step, M_2 performs one step of M_1 in every copy independently. Automaton M_2 accepts iff majority of the simulated components accepts.

More formally, let Q is the set of states of M_1. Automaton M_2 has the set of states $(Q \cup \{\bot, \top\})^c \cup \{q'_a\}$, where q'_a is a new accept state of M_2 and S^c denotes Cartesian product applied c times, i. e., $S^c = \overbrace{S \times \ldots \times S}^{c}$. The symbol $\bot$ is used to denote that the corresponding simulation hangs; $\top$ means that the corresponding simulation already accepted. The start state of M_2 is $(\overbrace{q_s, \ldots, q_s}^{c})$, where q_s is the start state of M_1.

Automaton M_2 simulates c independent runs of M_1 in a straightforward way on every symbol except $\dashv$. Hence, the transition function δ^{M_2} of M_2 is defined as

$$\delta^{M_2}((q_1, \ldots, q_c), a)((q'_1, \ldots, q'_c)) = \prod_{i=1}^{c} \delta^{M_1}(q_i, a)(q'_i)$$

for any $a \neq \dashv$, where δ^{M_1} is the transition function of M_1, extended in the following way:

$$\begin{array}{lcll}
\delta^{M_1}(\bot, a)(\bot) & = & 1 & \\
\delta^{M_1}(\bot, a)(q) & = & 0 & \forall q \neq \bot \\
\delta^{M_1}(\top, a)(\top) & = & 1 & \\
\delta^{M_1}(\top, a)(q) & = & 0 & \forall q \neq \top \\
\delta^{M_1}(q, a)(\bot) & = & 1 & \text{if transition function of } M_1 \text{ is not defined for (q,a)} \\
\delta^{M_1}(q, a)(q') & = & 0 & \text{if transition function of } M_1 \text{ is not defined for (q,a)} \\
 & & & \text{and } q' \neq \bot
\end{array}$$

When reading $\dashv$, we define the transition function of δ^{M_2} in a similar way, except that we implement the accepting condition of M_2 and take care to properly remember which simulated runs accepted:

$$\begin{array}{ll}
\delta^{M_2}((q_1, \ldots, q_c), \dashv)(q'_a) & = 1 \quad \text{if more than } c/2 \text{ of } q_i \text{ are equal to } \top \\
\delta^{M_2}((q_1, \ldots, q_c), \dashv)(q'_a) & = 0 \quad \text{if at most } c/2 \text{ of } q_i \text{ are equal to } \top \\
\delta^{M_2}((q_1, \ldots, q_c), \dashv)((h(q'_1), \ldots, h(q'_c))) & = \prod_{i=1}^{c} \delta^{M_1}(q_i, a)(q'_i) \\
 & \quad \text{if at most } c/2 \text{ of } q_i \text{ are equal to } \top \\
\delta^{M_2}((q_1, \ldots, q_c), \dashv)((q'_1, \ldots, q'_c)) & = 0 \quad \text{if not defined otherwise}
\end{array}$$

where $h(q_a) = \top$ and $h(q) = q$ for any $q \neq q_a$.

It is easy to check that the probability that M_2 accepts an input word z is equal to the probability that more than one half of c independent runs of M_1 accepts z. According to [Hro05], if (3.2) holds, this probability is at least $1/2 + \varepsilon_2$ if the input z is in the solved language and at most $1/2 - \varepsilon_2$ otherwise. □

In the proof of Lemma 3.1, we have used parallel simulation of the independent runs, since this is the only valid option for one-way automata. For rotating and sweeping automata we can also use sequential simulation, which is more efficient: Instead of yielding $O(k^c)$-state automata, it produces automata with $O(kc)$ states. Here, the idea is to simulate the independent runs sequentially, one after another. One problem that has to be solved when using this approach is the fact that rotating and sweeping Monte-Carlo automata may loop infinitely when not accepting the input word. Nevertheless, it is possible to prove that any accepting computation of a RP$_2$FA or a SP$_2$FA has at most exponential length with very high probability. Hence, it is possible to implement the sequential simulation by stopping every simulated run after exponential expected time, what can be done by stopping the simulation with probability α^n after every simulated run (for some constant $\alpha < 1$, where n is the length of the input word). Nevertheless, the gain obtained by the sequential simulation is not relevant with respect to the polynomial complexity classes we are dealing with.

3.1.2 Monte-Carlo Automata with One-Sided Error

Another well known variant of Monte-Carlo computation models is a more restricting one, in the sense that the randomized machine may err on one side only. More precisely, any word in the solved language must be accepted with a nonzero probability, and any word not in the solved language must be accepted with zero probability. In this way, we define the *one-way (rotating, sweeping, two-way) Monte-Carlo finite automata with one-sided unbounded error* and denote them as $1\mathrm{P}_1^{\mathrm{U}}$FAs ($\mathrm{RP}_1^{\mathrm{U}}$FAs, $\mathrm{SP}_1^{\mathrm{U}}$FAs, $2\mathrm{P}_1^{\mathrm{U}}$FAs). In the terminology of [Rab63], these automata are probabilistic automata with (non-isolated) cut-point 0. Consistently with our naming convention, we denote the corresponding complexity classes as $1\mathrm{P}_1^{\mathrm{U}}$, $\mathrm{RP}_1^{\mathrm{U}}$, $\mathrm{SP}_1^{\mathrm{U}}$, and $2\mathrm{P}_1^{\mathrm{U}}$.

It is easy to see that the Monte-Carlo randomization with one-sided (unbounded) error is a special case of nondeterminism. Indeed, any Monte-Carlo randomized machine with one-sided error can be trivially transformed into an equivalent nondeterministic machine, just by replacing any random choice that occurs with nonzero probability by a nondeterministic choice.

In a similar way as for Monte-Carlo machines with two-sided error, we can consider a bounded-error variant of Monte-Carlo machines with one-sided error. Here, a finite automaton M accepting a language L is required to accept all words in L with probability at least ε and accept all words not in L with zero probability, where $\varepsilon > 0$ is some fixed constant. Applying this definition to one-way, rotating, sweeping, as well as two-way randomized finite automata yields the computation models of *one-way, rotating, sweeping, and two-way Monte-Carlo finite automata with one-sided bounded error*, which we denote as $1\mathrm{P}_1$FAs, RP_1FAs, SP_1FAs, and $2\mathrm{P}_1$FAs.

Analogously as for the Monte-Carlo automata with two-sided error, we proceed with the definition of nonuniformly bounded classes corresponding to one-way, rotating, sweeping, and two-way Monte-Carlo finite automata with one-sided error, which we denote as $1\mathrm{P}_1^{\mathrm{N}}$, $\mathrm{RP}_1^{\mathrm{N}}$, $\mathrm{SP}_1^{\mathrm{N}}$, and $2\mathrm{P}_1^{\mathrm{N}}$. Furthermore, we define the corresponding uniformly bounded classes, denoted as $1\mathrm{P}_1^{\varepsilon}$, $\mathrm{RP}_1^{\varepsilon}$, $\mathrm{SP}_1^{\varepsilon}$, and $2\mathrm{P}_1^{\varepsilon}$. It is straightforward to check that Lemma 3.1 can be easily adapted for machines with one-sided error. Furthermore, it can be extended to two-way machines, too, by using sequential simulation and the technique used in [MS99] to avoid problems with detection of termination. This fact is stated as the following lemma:

Lemma 3.2. *Consider arbitrary $\varepsilon_1, \varepsilon_2 \in (0,1)$. There exists an integer c that depends only on $\varepsilon_1, \varepsilon_2$ such that for any $1\mathrm{P}_1$FA M_1 with k states working with error bound ε_1 there exists an equivalent $1\mathrm{P}_1$FA M_2 with $O(k^c)$ states working with error bound ε_2. An analogous result holds for RP_1FAs, SP_1FAs and $2\mathrm{P}_1$FAs.*

Proof. The claim for one-way, rotating, and sweeping automata can be proven in a similar way as in Lemma 3.1. Indeed, it is sufficient that M_2 simulates c independent computations of M_1 such that

$$c > \frac{\ln(1-\varepsilon_2)}{\ln(1-\varepsilon_1)}$$

and accepts iff at least one simulated computation accepts the input. It is easy to see that probability of accepting any word not in the solved language is zero and probability of rejecting any word in the solved language is at most

$$(1-\varepsilon_1)^c \leq 1-\varepsilon_2.$$

Hence, M_2 works with error bound ε_2.

For two-way automata, the result of [MS99] ensures that there exists an automaton M_1' equivalent to M_1 that halts with probability 1, works with error bound $1/2$, and has $O(k)$ states. Then, it is possible to construct M_2 that simulates c independent runs of M_1' in a sequential way. In this way, we obtain the desired M_2 with only $O(kc)$ states. Note that this method is usable for rotating and sweeping automata as well. □

It is not difficult to see that Monte-Carlo randomized machines with one-sided error can be simulated by Monte-Carlo machines with two-sided error without significant increase in the state complexity:

Lemma 3.3. *Any k-state Monte-Carlo finite automaton M with one-sided error accepting language L can be transformed into an equivalent $(k+O(1))$-state Monte-Carlo finite automaton M' with two sided error of the corresponding type.*

Proof. It is sufficient that M' accepts immediately at the start of the computation with some probability α and simulates M otherwise. When dealing with unbounded error, we set $\alpha = 1/2$. In this case, every word not in L is accepted with probability $1/2$, and every word in L is accepted with probability strictly greater than $1/2$.

For bounded error models with error bound ε, we can set $\alpha = (1-\varepsilon)/(2-\varepsilon)$; in this way, the resulting M' has error bound $\varepsilon/(4-2\varepsilon)$. Indeed, any word not in the solved language is accepted with probability α, any word in the language is accepted with probability at least $\alpha + (1-\alpha)\varepsilon$, and for the error bound of M' it holds that

$$\frac{1}{2} - \alpha = \alpha + (1-\alpha)\varepsilon - \frac{1}{2} = \frac{\varepsilon}{4-2\varepsilon}.$$

□

Due to Lemma 3.3, we have $1\mathrm{P}_1^{\mathrm{U}} \subseteq 1\mathrm{P}_2^{\mathrm{U}}$, $1\mathrm{P}_1^{\mathrm{N}} \subseteq 1\mathrm{P}_2^{\mathrm{N}}$, $1\mathrm{P}_1^{\varepsilon} \subseteq 1\mathrm{P}_2^{\varepsilon}$, $\mathrm{RP}_1^{\mathrm{U}} \subseteq \mathrm{RP}_2^{\mathrm{U}}$, $\mathrm{RP}_1^{\mathrm{N}} \subseteq \mathrm{RP}_2^{\mathrm{N}}$, $\mathrm{RP}_1^{\varepsilon} \subseteq \mathrm{RP}_2^{\varepsilon}$, $\mathrm{SP}_1^{\mathrm{U}} \subseteq \mathrm{SP}_2^{\mathrm{U}}$, $\mathrm{SP}_1^{\mathrm{N}} \subseteq \mathrm{SP}_2^{\mathrm{N}}$, $\mathrm{SP}_1^{\varepsilon} \subseteq \mathrm{SP}_2^{\varepsilon}$, $2\mathrm{P}_1^{\mathrm{U}} \subseteq 2\mathrm{P}_2^{\mathrm{U}}$, $2\mathrm{P}_1^{\mathrm{N}} \subseteq 2\mathrm{P}_2^{\mathrm{N}}$, and $2\mathrm{P}_1^{\varepsilon} \subseteq 2\mathrm{P}_2^{\varepsilon}$.

3.1.3 LasVegas Automata

It is possible to restrict the randomized automata even more and require zero probability of error. In this setting, called *LasVegas* randomized computation, the randomized machine is allowed to give 3 possible answers: *Yes*, *No*, or *I do not know*. Whenever the answer *Yes* or *No* is given, it has to be correct. Furthermore, probability of answering *I do not know* must be bounded by a constant smaller than one, unless we are dealing with the variant with unbounded error.

More precisely, a *Las Vegas* automaton M needs to have a special *reject* state $q_r \in Q$ in addition to the accept state q_a. If M reaches q_r after reading $\dashv$, the computation of M halts, and we say that the computation of M *rejects* the input word (i. e., gives answer *no*). Beside accepting (i. e., giving answer *yes*), a computation of M can also hang or run for an infinite time; in this case, the computation neither accepts nor rejects the input (i. e., gives an answer *I do not know*). An input word $z \in \Sigma^*$ is accepted if and only if M accepts z with nonzero probability and rejects z with probability 0. Similarly, z is rejected if and only if M rejects z with nonzero probability and accepts z with probability 0. Automaton M is a correct LasVegas automaton with unbounded error if every $z \in \Sigma^*$ is either accepted or rejected.

The definition of LasVegas automata applies to one-way automata ($1\mathrm{P}_0^{\mathrm{U}}\mathrm{FAs}$), rotating automata ($\mathrm{RP}_0^{\mathrm{U}}\mathrm{FAs}$), sweeping automata ($\mathrm{SP}_0^{\mathrm{U}}\mathrm{FAs}$), as well as two-way automata ($2\mathrm{P}_0^{\mathrm{U}}\mathrm{FAs}$). We denote the complexity classes induced by small families of these automata as $1\mathrm{P}_0^{\mathrm{U}}$, $\mathrm{RP}_0^{\mathrm{U}}$, $\mathrm{SP}_0^{\mathrm{U}}$, and $2\mathrm{P}_0^{\mathrm{U}}$, respectively.

The model of LasVegas computations with unbounded error is rather unnatural, but we deal with it for the sake of completeness. The more commonly used model is a bounded-error variant of LasVegas computations. A valid LasVegas finite automaton with error bound $\varepsilon > 0$ is required to both accept every word in the solved language and reject every word not in the solved language with probability at least ε. We denote one-way, rotating, sweeping, and two-way LasVegas finite automata with bounded error as $1\mathrm{P}_0\mathrm{FAs}$, $\mathrm{RP}_0\mathrm{FAs}$, $\mathrm{SP}_0\mathrm{FAs}$, and $2\mathrm{P}_0\mathrm{FAs}$. Again, we define both the nonuniformly bounded and uniformly bounded corresponding complexity classes, and denote them as $1\mathrm{P}_0^{\mathrm{N}}$, $\mathrm{RP}_0^{\mathrm{N}}$, $\mathrm{SP}_0^{\mathrm{N}}$, $2\mathrm{P}_0^{\mathrm{N}}$, $1\mathrm{P}_0^{\varepsilon}$, $\mathrm{RP}_0^{\varepsilon}$, $\mathrm{SP}_0^{\varepsilon}$, and $2\mathrm{P}_0^{\varepsilon}$.

In a similar way as for the Monte-Carlo automata, it is possible to use the amplification technique for LasVegas finite automata (using parallel simulation for one-way machines, sequential simulation for two-way machines, and either one for rotating and sweeping machines). Hence, the uniformly bounded complexity classes are the same for any value of $\varepsilon \in (0, 1)$:

Lemma 3.4. *Consider arbitrary $\varepsilon_1, \varepsilon_2 \in (0, 1)$. There exists an integer c that depends only on $\varepsilon_1, \varepsilon_2$ such that for any $1\mathrm{P}_0\mathrm{FA}$ M_1 with k states working with error bound ε_1 there exists an equivalent $1\mathrm{P}_0\mathrm{FA}$ M_2 with $O(k^c)$ states working with error bound ε_2. An analogous result holds for $\mathrm{RP}_0\mathrm{FAs}$, $\mathrm{SP}_0\mathrm{FAs}$ and $2\mathrm{P}_0\mathrm{FAs}$.*

Proof. The parallel simulation works in the same way as in Lemma 3.2. For the sequential simulation, it is possible to adapt the technique of [MS99] in the same way as it was done in [HS01b]. Indeed, it is sufficient to restart the simulated LasVegas machine with a fixed probability after each computation step. In this way, the correct answer is eventually found. Such construction yields a LasVegas automaton M_2 with $O(k)$ states that always gives correct answer. □

The concept of LasVegas finite automata has been introduced in [HS01a] in the uniformly bounded variant. The LasVegas randomization has been, however, extensively studied for other computation models, such as Turing machines, communication complexity, boolean circuits, and ordered binary decision diagrams.

3.2 Results for Unrestricted Running Time

In this section, we present previously known results (as well as their straightforward implications) about the complexity classes of randomized finite automata defined so far. We do not place any further restrictions to the randomized machines yet. In particular, we are not interested in the running time of rotating, sweeping, and one-way randomized finite automata. The impact of restricted running time on the computational power of the automata is analyzed in the remaining sections of this chapter.

3.2.1 Rotating, Sweeping, and Two-Way Automata

Most of the complexity classes of rotating, sweeping, and two-way Monte-Carlo and LasVegas finite automata fit nicely to the map of the complexity classes presented in Section 2.4.

Monte-Carlo automata with One-Sided Error. Obviously, any Monte-Carlo automaton with one-sided error, even one with unbounded error, can be trivially simulated by a corresponding nondeterministic automaton. Hence, it holds that $\mathrm{RP}_1^{\varepsilon} \subseteq \mathrm{RP}_1^{\mathrm{N}} \subseteq \mathrm{RP}_1^{\mathrm{U}} \subseteq \mathrm{RN}$, $\mathrm{SP}_1^{\varepsilon} \subseteq \mathrm{SP}_1^{\mathrm{N}} \subseteq \mathrm{SP}_1^{\mathrm{U}} \subseteq \mathrm{SN}$, and $2\mathrm{P}_1^{\varepsilon} \subseteq 2\mathrm{P}_1^{\mathrm{N}} \subseteq 2\mathrm{P}_1^{\mathrm{U}} \subseteq 2\mathrm{N}$. The result of [MS99] directly proves that, for any k-state 2NFA, there exists an equivalent $O(k)$-state $2\mathrm{P}_1\mathrm{FA}$ with an error bound $1/2$, what implies that $2\mathrm{N} \subseteq 2\mathrm{P}_1^{\varepsilon}$.[4] The same idea can be transformed for sweeping and rotating automata in a straightforward way, which proves that all one-sided error Monte-Carlo classes collapses with nondeterminism for rotating, sweeping, and two-way automata:

$$2\mathrm{P}_1^{\varepsilon} = 2\mathrm{P}_1^{\mathrm{N}} = 2\mathrm{P}_1^{\mathrm{U}} = 2\mathrm{N}$$

and (since we have proven $\mathrm{RN} = \mathrm{SN}$ in Chapter 2)

$$\mathrm{RP}_1^{\varepsilon} = \mathrm{RP}_1^{\mathrm{N}} = \mathrm{RP}_1^{\mathrm{U}} = \mathrm{RN} = \mathrm{SN} = \mathrm{SP}_1^{\varepsilon} = \mathrm{SP}_1^{\mathrm{N}} = \mathrm{SP}_1^{\mathrm{U}}.$$

LasVegas automata. Easily, any LasVegas machine can be trivially transformed into a corresponding one-sided error Monte-Carlo machine. Furthermore, all LasVegas classes are closed under complement, since it is possible to trivially transform any LasVegas automaton A accepting L into a LasVegas automaton of the same type accepting $\overline{L}$ just by exchanging the accept and reject state. Hence, LasVegas classes are always subsets of the corresponding self-verifying classes: $\mathrm{RP}_0^{\mathrm{U}} \subseteq \mathrm{R}\Delta$, $\mathrm{SP}_0^{\mathrm{U}} \subseteq \mathrm{S}\Delta$, and $2\mathrm{P}_0^{\mathrm{U}} \subseteq 2\Delta$. Furthermore, it was shown in [HS01b] that it is possible to adapt the technique of [MS99] to prove that any selfverifying automaton can be simulated by a (uniformly bounded) LasVegas automaton of the corresponding type without

[4] For coin-flipping automata, the construction can yield $O(k^2)$-state $2\mathrm{P}_1\mathrm{FA}$. Nevertheless, this is not significant for the relationship of the corresponding complexity classes.

significant blowup in its state complexity. Hence, all LasVegas classes collapses with self-verification for rotating, sweeping, and two-way automata:

$$2\mathrm{P}_0^{\varepsilon} = 2\mathrm{P}_0^{\mathrm{N}} = 2\mathrm{P}_0^{\mathrm{U}} = 2\Delta$$

and (since we have proven $\mathrm{R}\Delta = \mathrm{S}\Delta$ in Chapter 2)

$$\mathrm{RP}_0^{\varepsilon} = \mathrm{RP}_0^{\mathrm{N}} = \mathrm{RP}_0^{\mathrm{U}} = \mathrm{R}\Delta = \mathrm{S}\Delta = \mathrm{SP}_0^{\varepsilon} = \mathrm{SP}_0^{\mathrm{N}} = \mathrm{SP}_0^{\mathrm{U}}.$$

Monte-Carlo automata with Two-Sided Error. Our knowledge of the classes of rotating, sweeping, and two-way Monte-Carlo finite automata with two-sided error is less deep. It is obvious that $\mathrm{RP}_2^{\varepsilon} \subseteq \mathrm{RP}_2^{\mathrm{N}} \subseteq \mathrm{RP}_2^{\mathrm{U}}$ and $2\mathrm{P}_2^{\varepsilon} \subseteq 2\mathrm{P}_2^{\mathrm{N}} \subseteq 2\mathrm{P}_2^{\mathrm{U}}$. It is possible to prove that a sweeping Monte-Carlo automaton with two-sided error can be simulated by a rotating automaton of the corresponding type without significant blowup in the state complexity.

The basic idea of the proof is similar to the proof used for the nondeterministic case in Lemma 2.17. The rotating automaton M' simulates the left computations of the sweeping automaton M in a straightforward way. To simulate a right computation, the computation RCOMP of M is chosen uniformly at random. Automaton M' continues with the simulation of the next traversal of M with the probability equal to the probability that M performs the chosen computation RCOMP. Otherwise, the simulation of the right-to-left traversal is repeated with another randomly chosen computation.

Lemma 3.5. *Each* SP_2FA *with k states can be simulated by a* RP_2FA *with at most $O(k^3)$ states. The same result holds for* $\mathrm{SP}_2^{\mathrm{U}}$FAs *and* $\mathrm{RP}_2^{\mathrm{U}}$FAs*, too.*

Proof (by T. Mömke). Given a k-state SP_2FA $M = (q_0, \delta, q_a)$ over an alphabet Σ and a set of states Q, we construct an equivalent RP_2FA $M' = (q_0', \delta', q_a')$ over the same alphabet with state set Q' such that $|Q'| = O(k^3)$.

The RP_2FA M' simulates each left computation of M in a straightforward manner. Thus each pair of states q_i, q_j in M has a corresponding pair of states q_i', q_j' in M' and the probability to reach a state q_j from q_i in M on the input string from left to right is exactly the same as the probability to reach q_j' from q_i' in M'.

Now we consider the simulation of a right computation. Let $w = w_1 w_2 \dots w_l$ be the input of length l. Assume that M starts the right computation at w_l in state q_1, i. e., M reached q_1 after finishing the previous left computation and reading $\dashv$. We know that the computation $\mathrm{RCOMP}_{M,q_1}(w)$ consists of a sequence of $l+1$ states. Consider any such sequence $\mathbf{s} = \langle s_1, s_2, \dots, s_l, s_{l+1} \rangle$. The probability that M performs $\mathbf{s}$ is

$$p_{\mathbf{s}} := [s_1 = q_1] \cdot \prod_{i=1}^{l} \delta(s_i, w_{l-i+1})(q_{i+1}),$$

where $[s_1 = q_1]$ is defined to be 1 if $s_1 = q_1$ and 0 otherwise. Now we consider all k^{l+1} sequences $\mathbf{s}$ and to each sequence we assign the probability of the corresponding

computation. Obviously, an invalid sequence has probability zero and the sum of the assigned probabilities over all sequences is 1.

The basic idea is that M' chooses some sequence $\mathbf{s} = \langle s_1, \ldots, s_{l+1} \rangle$ uniformly at random. With probability $p_{\mathbf{s}}$, automaton M' proceeds to the simulation of the next left computation of M starting in state s_{l+1}, where $p_{\mathbf{s}}$ is the probability assigned to the chosen sequence. In this case, we say that M' *accepted* the chosen sequence. With probability $1 - p_{\mathbf{s}}$, automaton M' chooses another sequence and repeats the process.

The probability of a *repetition*, i. e., of not accepting one randomly chosen sequence) is

$$1 - \sum_{(s_1 = q_1, s_2, \ldots, s_{l+1}) \in Q^{l+1}} \frac{1}{k^{l+1}} \cdot \prod_{i=1}^{l} \delta(s_i, w_{l-i+1})(s_{i+1}) = 1 - \frac{1}{k^{l+1}}.$$

The probability that M' makes exactly i repetitions, chooses the sequence $\mathbf{s}$ afterwards, and accepts it, is

$$\frac{1}{k^{l+1}} \cdot p_{\mathbf{s}} \cdot \left(1 - \frac{1}{k^{l+1}}\right)^i$$

Hence, the probability that M' eventually accepts $\mathbf{s}$ is

$$\frac{1}{k^{l+1}} \cdot p_{\mathbf{s}} \cdot \sum_{i \geq 0} \left(1 - \frac{1}{k^{l+1}}\right)^i = \frac{1}{k^{l+1}} \cdot p_{\mathbf{s}} \cdot \frac{1}{1 - \left(1 - \frac{1}{k^{l+1}}\right)} = p_{\mathbf{s}}.$$

We have just proven that the probability that M' accepts sequence $\mathbf{s}$ is the same as the probability that M performs $\mathbf{s}$. Hence, the probability distribution of M' over its set of states is isomorphic to the probability distribution of M every time the simulation of left computation starts. Thus, M' correctly simulates M.

Now we discuss how to implement this idea. Instead of picking a sequence directly, M' can also pick a sequence uniformly at random state by state from left to right and accept that sequence with the assigned probability. At first, M' selects state s_{l+1} uniformly at random. Automaton M' keeps q_1 and s_{l+1} stored in its states. Let $s_{l+1}, s_l, \ldots, s_{i+1}$ be the first $l - i + 1$ states of the sequence chosen by M'. Thus after $l - i + 1$ steps, M' knows q_1, s_{l+1}, s_{i+1} and, since it reads the input from left to right, symbol w_{l-i+1}. Now M' picks a state s_i uniformly at random. With probability $\delta(s_i, w_{l-i+1})(s_{i+1})$, automaton M' proceeds until the right end-marker is reached. With probability $1 - \delta(s_i, w_{l-i+1})(s_{i+1})$, automaton M' resets and starts a new simulation of the right computation. If $\dashv$ is reached, M' resets and starts a new simulation if $s_1 \neq q_1$. Otherwise, simulation of the right computation is finished, and the next left computation of M can be simulated. To do so, M' simulates the move of M from state s_{l+1} on $\vdash$ and proceeds to the first input symbol.

Now let us count the number of states of M'. Obviously, the left computation can be simulated with k states. In the simulation of the right computation, picking a

sequence and choosing whether to restart without storing the first and the last state of the computation only requires a copy of Q. Since we also store the first and the last state q_1 and s_{l+1}, we need k^2 copies of Q in total to simulate right computations of M. Thus M' has $O(k^3)$ states. □

Note that the proof of Lemma 3.5 works for automata using real probabilities as well as for automata using rational probabilities. Furthermore, it can be adapted in a straightforward way for coin-flipping automata.[5]

Since every rotating automaton with k states can be easily simulated by a sweeping automaton of the same type with $k+O(1)$ states, Lemma 3.5 shows that the rotating and sweeping classes of Monte-Carlo finite automata with two-sided error collapse: $\text{RP}_2^{\text{U}} = \text{SP}_2^{\text{U}}$, $\text{RP}_2^{\text{N}} = \text{SP}_2^{\text{N}}$, and $\text{RP}_2^{\varepsilon} = \text{SP}_2^{\varepsilon}$.

Lemma 3.1 implies that the classes $\text{RP}_2^{\varepsilon} = \text{SP}_2^{\varepsilon}$ collapse for all ε. However, we do not know if this is the case for two-way automata, too.

The last mentioned result about rotating, sweeping, and two-way Monte-Carlo finite automata with two-sided error is the one proved in [Fre81]: It is possible to accept some nonregular languages by bounded-error RP_2FAs. This fact easily implies that $\text{RN} \subsetneq \text{RP}_2^{\varepsilon}$, $\text{SN} \subsetneq \text{SP}_2^{\varepsilon}$, and $\text{2N} \subsetneq \text{2P}_2^{\varepsilon}$.

The results regarding rotating, sweeping, and two-way randomized automata are summarized in Figure 3.1. When combined with the results of Chapter 2, we have shown a separation between determinism and LasVegas randomization for rotating and sweeping automata. In fact, the question about their relationship was the main motivation for the analysis of the self-verifying nondeterminism for rotating and sweeping models.

3.2.2 One-Way Randomized Automata

One-way finite automata always run in linear time, since their head movement capabilities do not allow them to run longer.[6] However, less classes of one-way randomized finite automata collapse together and the resulting hierarchy of classes is more complicated. In this subsection, we show the hierarchy presented in Figure 3.2. Using the depicted results, Observation 1.1 allows us to classify the relationship of any pairs of classes in Figure 3.2 as either strict inclusion or incomparability, except for the relationship of 1P_0^{N} and $1\text{P}_2^{\varepsilon}$ (here it is open if they are incomparable of if $1\text{P}_0^{\text{N}} \subsetneq 1\text{P}_2^{\varepsilon}$).

[5] In this case, we need to modify the mechanism for deciding whether to start a new simulation of the right computation or not. Instead of considering the probability $\delta(s_i, w_{l-i+1})(s_{i+1})$ directly, we need to simulate all λ-transitions of M between the selection of states s_{i+1} and s_i. This can be done at the cost of increasing the number of states k-times, i. e., automaton M' is constructed with $O(k^4)$ states. Nevertheless, this increase in the state complexity is not significant from our point of view.

[6] This is not directly true for coin-flipping automata, as they may loop infinitely on λ-transitions. Nevertheless, it is easy to see that it is possible to remove all states that cause such infinite loops. Afterwards, the expected number of λ-transitions between any two head movements is bounded by a constant, thus the automaton runs in linear expected time.

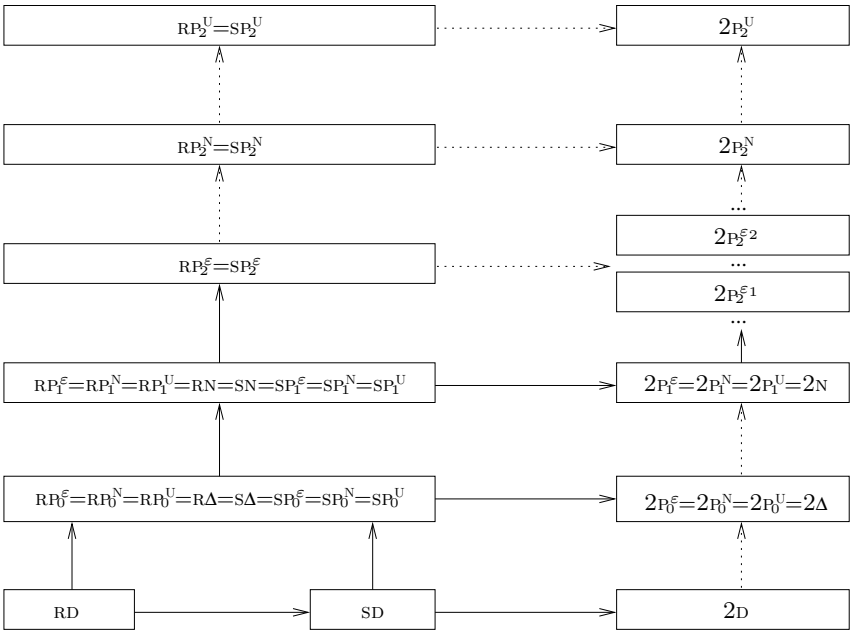

Figure 3.1: Map of the complexity classes of rotating, sweeping, and two-way randomized automata. A solid arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \subsetneq \mathfrak{C}'$, a dotted arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C}' \subseteq \mathfrak{C}$.

Collapsing classes. Lemma 3.1 ensures that the classes $1\mathrm{P}_2^{\varepsilon}$ are equivalent for all ε. Similarly, Lemma 3.2 and Lemma 3.4 ensure the same for $1\mathrm{P}_1^{\varepsilon}$ and $1\mathrm{P}_0^{\varepsilon}$. The fact that $1\mathrm{P}_0^{\varepsilon} = 1\mathrm{D}$ was proven in [HS01a] (and the proof was later refined in [HS03a]). Any $1\mathrm{P}_1^{\mathrm{U}}$FA can be trivially simulated by a 1NFA. Hence, $1\mathrm{P}_1^{\mathrm{U}} \subseteq 1\mathrm{N}$. Since $1\mathrm{P}_0^{\mathrm{U}}$ is closed under complement (as in any LasVegas class, it is sufficient to exchange the accept and the reject state to complement an automaton) and $1\mathrm{P}_0^{\mathrm{U}} \subseteq 1\mathrm{P}_1^{\mathrm{U}}$ trivially holds, $1\mathrm{P}_1^{\mathrm{U}} \subseteq 1\mathrm{N}$ implies also $1\mathrm{P}_0^{\mathrm{U}} \subseteq 1\Delta$. The other way round, it is possible to trivially simulate any 1NFA by a $1\mathrm{P}_1^{\mathrm{U}}$FA of the same size, just by replacing nondeterministic choices by random choices with arbitrary non-zero probabilities. In this way, the probability of accepting a word z is nonzero if and only if there exists an accepting computation for z in the 1NFA. Hence, $1\mathrm{P}_1^{\mathrm{U}} = 1\mathrm{N}$. By similar arguments, we can show that $1\mathrm{P}_0^{\mathrm{U}} = 1\Delta$: Let 1NFA M_1 accepts L with k_1 states and 1NFA M_2 accepts $\overline{L}$ with k_2 states, we can easily construct a $1\mathrm{P}_0^{\mathrm{U}}$FA M with $k_1 + k_2 + O(1)$ states accepting L in the following way. At the beginning of the computation, M decides whether to simulate M_1 or M_2, each with probability $1/2$. Afterwards, nondeterminism is replaced by random choices. If the simulated automaton accepts, M accepts (if M_1 is simulated) or rejects (if M_2 is simulated). Otherwise, M neither accepts nor rejects. Clearly M is always correct, and gives the correct answer with nonzero probability.

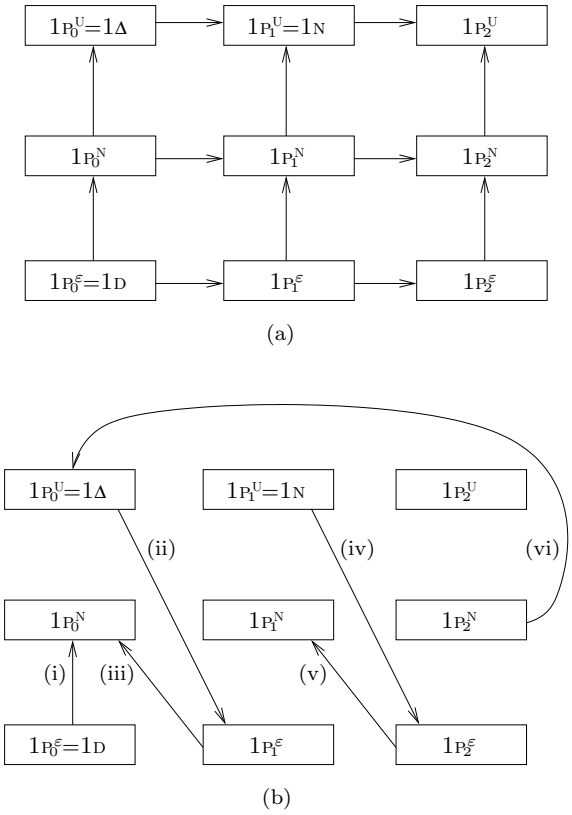

Figure 3.2: Map of the complexity classes of one-way randomized automata. (a) Inclusion results: An arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C} \subseteq \mathfrak{C}'$. (b) Separation results: An arrow $\mathfrak{C} \to \mathfrak{C}'$ means that $\mathfrak{C}' \not\subseteq \mathfrak{C}$.

Inclusion results. All inclusion results in Figure 3.2a either hold trivially or follow directly from Lemma 3.3.

Connection to communication complexity. Before we proceed to show the separation results in Figure 3.2b, we discuss the connection of the state complexity of finite automata with the communication complexity. This connection is not new, the communication complexity was used to prove lower bounds on the state complexity of finite automata e. g. in [HS01a]. To prove the claimed separations, we use some results about communication complexity, too. More precisely, we are interested in the fact that the state complexity of one-way finite automata and the two-party one-way communication complexity is equal for two-symbol languages.

We focus on the following model of communication complexity (used e. g. in [JPS84]): There are two *parties* with unlimited computational power. The first party C_1 receives an input x from a finite set of possible inputs X and sends some message (i. e., a binary string) to the second party C_2. The second party C_2 receives an input y from a finite set of possible inputs Y and the message sent by C_1 and it has to decide if the pair (x, y) satisfies the computed predicate $P(x, y) \subseteq X \times Y$. We say that a communication protocol computes P with communication complexity c if only messages of length at most c bits are used in any computation.

The communicating parties C_1 and C_2 can work in a deterministic, nondeterministic or randomized mode; in case of randomized communication protocols, we can again use either LasVegas, Monte-Carlo with one-sided error, or Monte-Carlo with two-sided error, each with bounded or unbounded error probability. Furthermore, each model of randomized communication protocols can be allowed to use arbitrary real probabilities, can be restricted to rational probabilities only, or can be restricted to use fair coin flips as the only source of randomness. The last case is used e. g. in [JPS84], where each communicating party is allowed to make certain fixed number of coin flips during the computation. Nevertheless, there are no significant differences in the communication complexity of bounded-error protocols using real probabilities, rational probabilities, or coin flips. More precisely, any bounded-error communication protocol using real probabilities can be simulated by a protocol of the same type restricted to coin flips. Indeed, it is possible to round the real probabilities to sufficiently close probabilities that are computable by coin flips. After such change, the probability of accepting any input is sufficiently close to the probability of accepting the same input before the change. Furthermore, we can assume that no nonzero probabilities are rounded to zero and vice-versa. Hence, if the rounding errors are sufficiently small with respect to the error bound, the resulting communication protocol is again a bounded error protocol of the same type (although the error bound may be weaker):

Observation 3.1. *Consider any LasVegas, one-sided error Monte-Carlo, or two-sided error Monte-Carlo communication protocol with bounded error using real probabilities. There exists an equivalent communication protocol of the same type with the same communication complexity restricted to fair coin flips only.*

It is not difficult to see that there is a straightforward mapping between communication protocols and one-way finite automata. Indeed, any (deterministic, nondeterministic, randomized) communication protocol C computing a predicate $P(x, y)$ can be transformed into a (deterministic, nondeterministic, corresponding randomized) finite automaton M accepting language

$$L_P = \{xy \mid x \in X, y \in Y \text{ such that } P(x, y) \text{ holds}\}.$$

Furthermore, if C has communication complexity c, then M has $2^c + O(1)$ states. The transformation itself is very straightforward: Automaton M has a special start state, and one state corresponding to every possible message of C. After reading the

first symbol of the input word, M enters the state corresponding to the message sent by C_1. Afterwards, M reads the second symbol and it either hangs or accept (using a special accept state), depending on the corresponding action of C_2.

On the other hand, any finite automaton M accepting language L_P with k states can be transformed into a communication protocol C of the same type computing P with communication complexity $O(\log k)$. Here, C_1 just sends the state of M after reading the first symbol as the message.

Hence, we can summarize our argumentation in the following observation:

Observation 3.2. *Let X, Y be finite sets and $P(X,Y) \subseteq X \times Y$ be any predicate over $X \times Y$. Consider any deterministic, self-verifying, nondeterministic, or randomized computation model using real or rational probabilities. There exists a finite automaton of this type accepting the language*

$$L_P = \{xy \mid x \in X, y \in Y \text{ such that } P(x,y) \text{ holds}\}$$

with k states if and only if there exists a communication protocol of the same type computing P with communication complexity $O(\log k)$.

Note that we have excluded the coin-flipping models from Observation 3.2. The reason for this is that coin-flipping automata and coin-flipping communication protocols are essentially different computation models: Coin-flipping automata have no fixed bound on the number of coin flips, what is not the case for coin-flipping communication protocols. On the other hand, the communication protocols can use very large number of flips without any impact on the communication complexity. Nevertheless, Observation 3.1 allows us to apply lower bounds on coin-flipping protocols to automata using real probabilities.

Separation results. Now we show the separations claimed in Figure 3.2b.

(i) The separation $1\mathrm{P}_0^{\mathrm{N}} \not\subseteq 1\mathrm{D}$ is witnessed by the language family $\mathcal{J}$ (defined by Equation 2.9). Indeed, $\mathcal{J} \notin 1\mathrm{D}$, as proven by Lemma 2.19. However, language J_n can be accepted by a $1\mathrm{P}_0\mathrm{FA}$ with $n+O(1)$ working with error bound $1/n$: The second symbol of the input word is guessed uniformly at random, and the guess is verified. The answer is provided with probability $1/n$ and is always correct.

(ii) The language family $\mathcal{J}$ describes the *membership problem*, where the goal is to determine if a number given as the second symbol of the word belongs to a set described by the first symbol. To witness the separation $1\mathrm{P}_1^{\varepsilon} \not\subseteq 1\Delta$, we define the *nonequality problem* $\mathcal{E} = (E_n)_{n\geq 1}$ defined as as follows:

$$E_n := \{\alpha_1\alpha_2 \mid \alpha_1, \alpha_2 \subseteq [n] \text{ and } \alpha_1 \neq \alpha_2\}, \tag{3.3}$$

where $[n] := \{1, \ldots, n\}$.

It has been proven in [JPS84, Theorem 3.1iii] that the nonequality problem (more precisely, the predicate corresponding to E_n in the sense of Observation 3.2) can be computed by a one-sided error Monte-Carlo communication protocol with error bound $1/2$ with communication complexity $O(\log n)$. (The idea behind this fact is the well-known fingerprinting technique used for equality testing, see e. g. [Hro05].) Hence, due to Observation 3.2, language E_n can be solved by a $1\mathrm{P}_1\mathrm{FA}$ with error bound $1/2$ and with number of states polynomial in n, what implies that $\mathcal{E} \in 1\mathrm{P}_1^{\varepsilon}$.[7]

On the other hand, it is not difficult to see that any 1NFA M accepting the complement of E_n needs at least 2^n states. Indeed, for every $\alpha \subseteq [n]$ there must be some state of M that is reached by some accepting computation after reading the first symbol α of the input word. Furthermore, if some of these states corresponds to two different sets $\alpha_1 \neq \alpha_2$, automaton M accepts also the word $\alpha_1\alpha_2$, what is a contradiction. Hence, $\mathcal{E} \notin 1\Delta$.

(iii) To witness the separation $1\mathrm{P}_0^{\mathrm{N}} \not\subseteq 1\mathrm{P}_1^{\varepsilon}$, we use a communication complexity result from [JPS84] about the language family $\mathcal{J}$. The proof of [JPS84, Theorem 3.1ii] implies that any uniformly bounded one-sided Monte-Carlo communication protocol (i. e., protocol with error bound that does not depend on n) computing the predicate corresponding to J_n in the sense of Observation 3.2 has communication complexity $\Omega(n)$. Hence, Observations 3.2 and 3.1 ensure that $\mathcal{J} \notin 1\mathrm{P}_1^{\varepsilon}$.

On the other side, it is straightforward to see that J_n can be accepted by an $O(n)$ state $1\mathrm{P}_0\mathrm{FA}$ with error bounded by $1/n$. Hence, $\mathcal{J} \in 1\mathrm{P}_0^{\mathrm{N}}$.

(iv) The separation $1\mathrm{P}_2^{\varepsilon} \not\subseteq 1\mathrm{N}$ follows from results presented in [Amb96]. There, the following language family $\{L_m\}_{m\geq 1}$ has been defined. Language L_m is a language over alphabet $\{a_1, \ldots, a_m\}$ such that:

$$L_m = \{w \mid w \text{ contains each of the letters } a_1, \ldots, a_m \text{ exactly } m \text{ times.}\}$$

It is not difficult to check that any 1NFA accepting L_m needs at least $(1+m)^m$ states (to prove our claim, it is sufficient to use analogous argument as employed in (ii) to show that $\overline{E_n}$ is hard for 1NFAs).

On the other hand, it has been proven in [Amb96] that there exists a $1\mathrm{P}_2\mathrm{FA}$ with $O(m\frac{\log^2 m}{\log\log m})$ states that solves L_m with error bound $1/2$. Hence, the language family $\mathcal{L} = \{L_m\}_{m\geq 1}$ is in $1\mathrm{P}_2^{\varepsilon}$, but not in 1N.

(v) To witness the separation $1\mathrm{P}_1^{\mathrm{N}} \not\subseteq 1\mathrm{P}_2^{\varepsilon}$, we use a communication complexity result from [BFS86] about the *disjointness problem* $\mathcal{D} = (D_n)_{n\geq 1}$ defined in Equation 2.10:

$$D_n := \{\alpha\beta \mid \alpha, \beta \subseteq [n] \text{ and } \alpha \cap \beta = \emptyset\}.$$

[7] For coin-flipping automata, we can not use Observation 3.2 directly to propagate the upper bound. Nevertheless, it is easy to see that the upper bound is valid for coin-flipping automata, too.

It has been proven in [BFS86, Theorem 7.2] that any uniformly bounded two-sided Monte-Carlo communication protocol that computes the predicate corresponding to D_n in the sense of Observation 3.2 has communication complexity $\Omega(\sqrt{n})$. Hence, Observations 3.2 and 3.1 ensure that $\mathcal{D} \notin 1\mathrm{P}_2^{\varepsilon}$. Since $1\mathrm{P}_2^{\varepsilon}$ is closed under complement (it is sufficient to make every accepting computation rejecting and every computation that hangs accepting, what can be achieved using one extra state), $\overline{\mathcal{D}} \notin 1\mathrm{P}_2^{\varepsilon}$. On the other hand, it is straightforward to see that $\overline{D_n}$ can be accepted by an $O(n)$-state 1P$_1$FA with error bounded by $1/n$. Hence, $\overline{\mathcal{D}} \in 1\mathrm{P}_1^{\mathrm{N}}$.

(vi) The proof of the separation $1\Delta \not\subseteq 1\mathrm{P}_2^{\mathrm{N}}$ follows from the main result of this thesis presented in Section 3.4. Here we show that there exists a family of languages $\mathcal{L}$ in 1Δ that is not solvable by small nonuniformly-bounded RP$_2$FAs working in linear time. Obviously, any 1P$_2$FA can be trivially simulated by a RP$_2$FA in linear time. Hence, $\mathcal{L}$ does not belong to $1\mathrm{P}_2^{\mathrm{N}}$.

Several separation results in Figure 3.2b follow from other previously published works, which sometimes show stronger results. For example, $1\mathrm{P}_2^{\mathrm{N}} \subsetneq 1\mathrm{P}_2^{\mathrm{U}}$ follows from the fact that $1\mathrm{P}_2^{\mathrm{N}}$ contains only regular languages [Rab63], but $1\mathrm{P}_2^{\mathrm{U}}$ contains some non-regular languages, too [Rab63, Fre81].

It has been shown in [MPP01] that there exists a family of unary languages that is not in 1N, but whose complement is in $1\mathrm{P}_1^{\mathrm{N}}$. Hence the separation $1\mathrm{N} \not\supseteq \text{co-}1\mathrm{P}_1^{\mathrm{N}} \subseteq 1\mathrm{P}_2^{\mathrm{N}}$ holds even for unary languages.

An alternative nonconstructive proof for the separation $1\mathrm{P}_2^{\varepsilon} \not\subseteq 1\mathrm{D}$ has been proven in [Fre08]. Furthermore, the witness language family, whose existence is proved in a nonconstructive way, can be accepted by small randomized *reversible* automata.

3.3 Lower Bounds on 1P$_1$FAs

In this section, we present the proof of the separation $1\Delta \not\subseteq 1\mathrm{P}_1^{\mathrm{N}}$, which will be needed in Section 3.4 to prove lower bounds on rotating randomized automata working in linear expected time. More precisely, we adapt the idea of confusing strings, as used in Section 2.3.1, to randomized automata. In this way, we prove a hardness propagation from 1D to $1\mathrm{P}_1^{\mathrm{N}}$, as stated by the following lemma:

Lemma 3.6. *If no* 1DFA *with at most m states can solve language L, then no* 1P$_1$FA *with at most $m-1$ states can solve language $\bigwedge L$. Similarly for $\bigoplus L$.*

Since Lemma 3.6 states that there is no small 1P$_1$FA working with *any* (arbitrarily small) error bound, the following corollary immediately follows:

Corollary 3.7. *If $\mathcal{L} \notin 1\mathrm{D}$, then $\bigwedge\mathcal{L} \notin 1\mathrm{P}_1^{\mathrm{N}}$ and $\bigoplus\mathcal{L} \notin 1\mathrm{P}_1^{\mathrm{N}}$.*

Using Corollary 3.7, Lemma 2.19, Lemma 2.16, and Figure 2.4[C6,C7], we immediately get that $\bigwedge\mathcal{J} \in 1\Delta - 1\mathrm{P}_1^{\mathrm{N}}$.

In some sense, this result is not very surprising. Intuitively, 1P$_1$FAs are somewhat similar to $\cup_L$1DFAs: A $\cup_L$1DFA consists of several components and the input word is accepted if at least one of them accepts. On the other hand, a 1P$_1$FA can behave differently for different sets of random decisions and the input is accepted if at least one of the random decisions leads to the acceptance. Hence, a 1P$_1$FA can be very roughly viewed as a collection of many deterministic automata, each for one set of possible random decisions.

While this intuition suggests that it is feasible to adapt the technique of confusing strings to 1P$_1$FAs, it is rather imprecise; while $\cup_L$1DFAs have always constant number of components, the number of possible random decisions made by 1P$_1$FAs depends on the length of the input word. We make, however, our argumentation more precise in the rest of this section. We focus only on the case of $\bigwedge L$, the proof for $\bigoplus L$ is analogous.

Formally, the transition function $\delta(q_1, a)(q_2)$ of a randomized automaton describes the probability that the automaton makes a transition from state q_1 to q_2 when reading a symbol a. In this section, we introduce an extension of this notation for one-way automata: We use the expression $\delta(q_1, w)(q_2)$ to denote the probability that the automaton started in state q_1 reaches state q_2 after reading the word w. Furthermore, we assume in the rest of this section that we deal only with one-way randomized automata that do not hang while reading the input:

Definition 3.1. *Let $M = (q_a, \delta, q_s)$ be any randomized automaton such that $\delta(q, a)$ is defined for all q and all $a \neq \dashv$ and that $\delta(q, \dashv)$ is either undefined or leads to q_a with probability 1. Then we say that automaton M is* non-hanging.

Restricting ourselves to non-hanging automata causes no loss of generality, since any randomized automaton can be easily transformed into a non-hanging one by adding a single new state.

Now we formalize the notion of confusion for randomized automata.

Definition 3.2. *Consider any one-way randomized automaton $M = (q_s, \delta, q_a)$ over a set of states Q. We say that a state $q \in Q$ is* confused with respect to language L *if and only if both of these two conditions hold:*

- *There exists some $w_1 \in L$ such that $\delta(q_s, \vdash w_1)(q) \neq 0$, i. e., the probability that M started in q_s reaches q after reading $\vdash w_1$ is nonzero.*
- *There exists some $w_2 \notin L$ such that $\delta(q_s, \vdash w_2)(q) \neq 0$, i. e., the probability that M started in q_s reaches q after reading $\vdash w_2$ is nonzero.*

Definition 3.3. *Consider an one-way randomized automaton $M = (q_s, \delta, q_a)$ and let Q_c be the set of states of M that are confused with respect to language L. We say that M is* confused by an input word w with probability p with respect to language L *if the probability that M started in q_s reaches some state $q \in Q_c$ after reading $\vdash w$ is equal to p. Equivalently, this can be written as*

$$p := \sum_{q \in Q_c} \delta(q_s, \vdash w)(q).$$

Furthermore, we say that M is non-confused by w with probability p with respect to L *if M is confused by w with probability $1-p$ with respect to L.*

Note that the definitions above are valid for any model of one-way randomized automata. For 1P1FAs, we say that a state of 1P1FA M accepting language L is confused if it is confused with respect to L. In a similar way, we extend the notion of confusion and non-confusion probability: We say that 1P1FA M accepting language L is (non-)confused by an input word w with probability p, if it is (non-)confused by w with probability p with respect to L.

Consider any 1P1FA M accepting language L. It is easy to observe that any confused state q of M is nonfinal, i. e., the probability of reaching the accept state from q at the end of the input is zero. Otherwise, the automaton is not Monte-Carlo with one-sided error. On the other hand, recall that 1P1FAs work with bounded error. Hence, to prove that a 1P1FA M cannot accept language L, it is sufficient to construct, for arbitrary given ε, a word from L that confuses M with probability at least $1-\varepsilon$.

More formally, we prove the following lemma, which is in some sense an analogy of Lemma 2.3.

Lemma 3.8. *The following statements are equivalent:*

1. *There exists a non-hanging* 1P1FA *M with m states that solves L.*
2. *There exists some $\varepsilon > 0$ and a non-hanging* 1P1FA *M with m states that solves L such that, for every word $w \in L$, automaton M is non-confused by w with probability at least ε.*
3. *There exists some $\varepsilon > 0$ and a non-hanging one-way randomized automaton M with m states such that, for every word $w \in L$, automaton M is non-confused by w with respect to L with probability at least ε.*

Proof. [1 $\Rightarrow$ 2] Assume that $M = (q_s, Q, q_a)$ works with error bound ε. We claim that M is non-confused by any word $w \in L$ with probability at least ε. Assume that this is not true, i. e., there exists some $w \in L$ such that M is non-confused by w with probability less than ε. Observe that any confused state q' of M is non-final, i. e., $\delta(q', \dashv) = \bot$ or $\delta(q', \dashv)(q_a) = 0$. Otherwise, there exists some word $u \notin L$ such that $\delta(q_a, \vdash u)(q') > 0$, thus $\delta(q_a, \vdash u\dashv)(q_s) > 0$, what contradicts to the fact that M works with one-sided error. Now consider the probability that M accepts w. Every accepting computation must enter the accept state from a non-confused state, since every confused state is non-final. Hence, the probability that M accepts w is upper bounded by the probability that M reaches a non-confused state after reading $\vdash w$, which is less than ε. But this is a contradiction with the error bound of M.

[2 $\Rightarrow$ 3] Since every 1P1FA is a one-way randomized automaton, the implication holds trivially.

[3 $\Rightarrow$ 1] Consider a one-way randomized automaton $M = (q_a, Q, q_s)$ that is, for any word $w \in L$, non-confused by w with respect to L with probability at least ε. We show that it is easily possible to transform M into a 1P1FA solving L with error bound

ε. This transformation is achieved by making every state of M that is non-confused with respect to L and is reachable only by words from L a final state, and making every other state a non-final state. More precisely, we modify the definition of the transition function δ of M as follows. Let $Q_c \subseteq Q$ be the set of states of M confused with respect to L, $Q_y \subseteq Q - Q_c$ be the set of states that are reachable only by words from L, i. e.,

$$Q_y := \{q \in Q \mid \forall w : \delta(q_s, \vdash w)(q) > 0 \Rightarrow w \in L\},$$

and $Q_n \subseteq Q - Q_c$ be the set of states that are reachable only by words not from L, i. e.,

$$Q_y := \{q \in Q \mid \forall w : \delta(q_s, \vdash w)(q) > 0 \Rightarrow w \notin L\}.$$

Easily, Q is a disjoint union of Q_c, Q_y, and Q_n (under the assumption that all states in Q are reachable). Now we modify the transition function of M as follows:

$$\begin{array}{lcll} \delta(q, \dashv)(q_a) & := & 1 & \forall q \in Q_y \\ \delta(q, \dashv)(q') & := & 0 & \forall q \in Q_y, \forall q' \neq q_a \\ \delta(q, \dashv) & := & \bot & \forall q \in Q_n \cup Q_c \end{array}$$

It is easy to see that the modified M accepts all words not in L with zero probability. Now consider any $w \in L$ and analyze the probability that M accepts w. If M reaches a non-confused state after reading $\vdash w$, it accepts with probability 1, since any such reached state is in Q_y. The probability that M reaches a non-confused state is, however, at least ε by the assumed lower bound on confusion probability of M. Hence, M is a 1P$_1$FA with one-sided error bounded by ε. □

We prove Lemma 3.6 by contradiction: Assume that no small 1DFA can solve L, but some small 1P$_1$FA M can solve $\bigwedge L$. We show that M can be confused with arbitrarily high probability, what directly contradicts to Lemma 3.8.[8] At first, we show that the automaton M can be confused from any starting state:

Lemma 3.9. *Assume that no* 1DFA *with at most m states can solve language L. Let $M = (q_s, \delta, q_a)$ be some non-hanging* 1P$_1$FA *with at most m states that solves language $\bigwedge L$. Consider any state q of M. There exists a word $w_1 \in L$, a word $w_2 \notin L$, and some state q_1 of M such that both $\delta(q, w_1\#)(q_1)$ and $\delta(q, w_2\#)(q_1)$ are nonzero.*

Proof. Let Q be the set of states of M. Let $Q_1 \subseteq Q$ be the set of states of M that are reachable from q by word $w_1\#$ for some word $w_1 \in L$, i. e.,

$$Q_1 = \{q_1 \mid \exists w_1 \in L \text{ such that } \delta(q, w_1\#)(q_1) > 0\}.$$

Similarly, let $Q_2 \subseteq Q$ be the set of states reachable by reading $w_2\#$ for some word $w_2 \notin L$:

$$Q_2 = \{q_2 \mid \exists w_2 \notin L \text{ such that } \delta(q, w_2\#)(q_2) > 0\}.$$

[8] In fact, we use only the first two statements of Lemma 3.8 in this section, but we need the third statement later in Subsection 3.5.1.

If $Q_1 \cap Q_2 \neq \emptyset$, the statement of the Lemma holds. Hence, assume by contradiction that Q_1 and Q_2 are disjoint. In that case, we can construct a 1DFA M' accepting L with at most m states as follows. Automaton M' simulates any computation of M that occurs with nonzero probability. More precisely, $M' = (q'_s, \delta', q'_a)$ is an automaton over the set of states $Q \cup \{q'_s, q'_a\}$, where q'_s and q'_a are some new states. The transition function δ' of M' is defined as follows:

$$\begin{array}{lcll} \delta'(q'_a, \vdash) & = & q & \\ \delta'(q', a) & = & q'' & \forall a \notin \{\vdash, \dashv\}, \forall q' \in Q, \text{ for arbitrary } q'' \text{ such that } \delta(q', a)(q'') > 0 \\ \delta'(q', \dashv) & = & q'_s & \forall q' \text{ such that } \delta(q', \#)(q_1) > 0 \text{ for some } q_1 \in Q_1 \\ \delta'(q', \dashv) & = & \bot & \text{otherwise} \end{array}$$

Note that the definition is correct: We assume that M does not hang, hence, for every $q' \in Q$ and $a \notin \{\vdash, \dashv\}$, there exists some q'' such that $\delta(q', a)(q'') > 0$.

Easily, if M' accepts word w, then $\delta(q, w\#)(q_1) > 0$ for some $q_1 \in Q_1$. If $w \notin L$, then $q_1 \in Q_2 \cap Q_1$ by the definition of Q_2, what is a contradiction. Hence, $w \in L$. On the other hand, assume that M' rejects word $w \in L$. The computation of M' hangs at some state q' while reading $\dashv$ and it holds that $\delta(q, w)(q') > 0$. Since we assume that M does not hang, there is some q'' such that $\delta(q', \#)(q'') > 0$, hence $\delta(q, w\#)(q'') > 0$. By definition of Q_1, $q'' \in Q_1$. In this case, however, M' can not reject w, because the third clause of δ' applies. □

Now we show the core of the proof of Lemma 3.6, i. e., we show how it is possible to increase the probability of confusion arbitrarily.

Lemma 3.10. *Assume that no* 1DFA *with at most m states can solve language L. Let $M = (q_s, \delta, q_a)$ be some non-hanging* 1P1FA *with at most m states that solves language $\bigwedge L$. There exists some constant $\alpha < 1$ such that, for any $u \in \bigwedge L$ that non-confuses M with probability p, there exists some $w \in L$ such that $uw\# \in \bigwedge L$ non-confuses M with probability at most αp.*

Proof. Since every finite automaton has finite number of states, Lemma 3.9 immediately implies that there exists some fixed $\beta > 0$ such that, for every state q of automaton M, there exist $w_1 \in L$, $w_2 \notin L$, and a state q' such that $\delta(q, w_1\#)(q') \geq \beta$ and $\delta(q, w_2\#)(q') \geq \beta$. Furthermore, if state q is reachable by some word $z \in \bigwedge L$, i. e., $\delta(q_s, z)(q) > 0$, state q' is confused.

Assume that M is non-confused with probability p after reading word $u \in \bigwedge L$. Let q be the most probable non-confused state of M after reading u, i. e., q is the non-confused state such that $p' := \delta(q_s, \vdash u)(q)$ is maximal. Since M has at most m states, it holds that $p' = \delta(q_s, \vdash u)(q) \geq p/m$. As we have already shown, there exist some $w_1 \in L$, $w_2 \notin L$, and a confused state q' such that $\delta(q, w_1\#)(q') \geq \beta$ and $\delta(q, w_2\#)(q') \geq \beta$.

Now we analyze the probability that M is non-confused after reading $uw_1\#$. This probability can be expressed as $p_1 + p_2 + p_3$, where:

- *p_1 is the probability that M is non-confused and not in state q after reading $\vdash u$, and non-confused after reading $\vdash uw_1\#$.* Obviously, we can upper bound p_1 by the probability that M is in a non-confused state other that q after reading $\vdash u$. Hence $p_1 \leq p - p'$.

- *p_2 is the probability that M is in a non-confused state q after reading $\vdash u$, and is non-confused after reading $\vdash uw_1\#$.* The probability that M is in q after reading $\vdash u$ is equal to p'. If this happens, M ends in a confused state q' with probability at least β, hence $p_2 \leq p'(1-\beta)$.

- *p_3 is the probability that M is confused after reading $\vdash u$ and non-confused after reading $\vdash uw_1\#$.* Easily, if M is in a confused state after reading $\vdash u$, then any state reached after reading complete $\vdash uw_1\#$ is confused. Hence, $p_3 = 0$.

To summarize, the probability that M is non-confused after reading $uw_1\#$ is at most

$$p_1 + p_2 + p_3 \leq (p - p') + p'(1-\beta) + 0 = p - p'\beta \leq p(1 - \beta/m).$$

Hence, the claim of the lemma follows by choosing $\alpha := 1 - \beta/m$. □

The proof of Lemma 3.6 follows easily from Lemma 3.10. Indeed, assume by contradiction that Lemma 3.6 does not hold, i. e., no 1DFA with at most m states solves language L, but there is some $1\mathrm{P}_1\mathrm{FA}$ with at most $m-1$ states that solves $\bigwedge L$. Easily, in such case there exists a non-hanging $1\mathrm{P}_1\mathrm{FA}$ with at most m states that solves $\bigwedge L$ as well. Then, by Lemma 3.8(2), there exist some ε and an m-state $1\mathrm{P}_1\mathrm{FA}$ M that solves $\bigwedge L$ such that M can not be non-confused with probability less than ε. On the other hand, by applying Lemma 3.10 at most $\log \varepsilon / \log \alpha + 1$ times, we can construct some word $w \in \bigwedge L$ that non-confuses M with probability less than ε, what is a contradiction.

3.4 Rotating Automata with Linear Running Time

So far, we have analyzed various models of randomized finite automata without any restriction on running time. We have shown that the use of randomization makes finite automata much stronger. For example, $1\mathrm{P}_2^{\mathrm{U}}\mathrm{FA}$s, $\mathrm{RP}_2\mathrm{FA}$s, $\mathrm{SP}_2\mathrm{FA}$s, and $2\mathrm{P}_2\mathrm{FA}$s can accept nonregular languages. Even the LasVegas randomization, which is the weakest model of randomized computations, allows to construct exponentially more succinct automata: $1\mathrm{D} \subsetneq 1\mathrm{P}_0^{\mathrm{N}}$, $\mathrm{RD} \subsetneq \mathrm{RP}_0^{\varepsilon}$, $\mathrm{SD} \subsetneq \mathrm{SP}_0^{\varepsilon}$, $2\mathrm{D} \subsetneq 2\mathrm{P}_0^{\varepsilon}$.

This power, however, heavily depends on the possibility of very long computations. Indeed, it is easy to see that the construction of [MS99], which proves that $2\mathrm{P}_1^{\varepsilon} = 2\mathrm{N}$, can yield randomized automata with exponential expected running time, and the same holds for the adaptation of this construction for rotating and sweeping automata, as well as for the adaptation for LasVegas randomization [HS01b].

A natural question is to ask how much does the power of the randomized finite automata change if we restrict their running time. This corresponds to the well known

question about the complexity classes of Turing machines: While it is known that LasVegas randomization is as strong as nondeterminism for space-restricted complexity classes [MS99], no such result is known for time-restricted classes. In particular, the question if P equals ZPP is one of the most prominent open problems in complexity theory.

We show that restricting running time of randomized finite automata significantly decreases their power. We focus on proving lower bounds on state complexity of randomized automata running in linear time. In particular, we show that there are some language families solvable by small LasVegas automata in exponential time, but not solvable by small Monte-Carlo automata working with two-sided bounded error in linear time. Hence, we show that even the very powerful model with two-sided error cannot compensate for the time restriction.

At first, we define time complexity of randomized finite automata, as well as the corresponding time-restricted complexity classes.

Definition 3.4. *Consider a randomized finite automaton M. For a given input word z, automaton M induces a probability distribution over all possible computations; each of them is either infinite, hangs, or accepts the input. The* expected running time of M on input z*, denoted as $T^M(z)$, is defined as the expected value of the length of the computation of M on z. It may happen that value $T^M(z)$ equals to infinity.*

We say that randomized finite automaton M has time complexity $T^M(n)$ *if and only if*

$$T^M(n) = \max_{z \in \Sigma^n} \big(T^M(z)\big),$$

i. e., $T^M(n)$ is the maximal expected running time of M over all words of length n.

Note that an automaton with time complexity $T(n)$ must have expected running time at most $T(n)$ on *any* word of length n. In other words, a fast randomized automaton must work fast also for words not in the solved language.

Definition 3.5. *Let $\mathcal{M} = \{M_i\}_{i \geq 1}$ be a family of randomized automata. We say that* $\mathcal{M}$ works in linear time *if and only if the time complexity of every M_i is linear, i. e., $T^{M_i}(n) = O(n)$.*

For every complexity class of randomized finite automata introduced so far, we can define the corresponding class of automata working in linear time. We denote such class by using a prefix lin-*. In particular, we are interested in* lin-$\mathrm{RP}_2^{\mathrm{N}}$, lin-$\mathrm{SP}_2^{\mathrm{N}}$, lin-$\mathrm{RP}_0^{\mathrm{N}}$, *and* lin-$\mathrm{SP}_0^{\mathrm{N}}$, *i. e., the classes of language families solvable by small* RP_2FAs, SP_2FAs, RP_0FAs, *and* SP_0FAs *working in linear time and nonuniformly bounded error.*

We also use the prefix lin- *to denote an automaton with linear time complexity. E. g.,* lin-RP_2FA, lin-SP_2FA, lin-RP_0FA, *and* lin-SP_0FA *denote a* RP_2FA, SP_2FA, RP_0FA, *and* SP_0FA *with linear time complexity, respectively.*

Note that in our definition of automata families working in linear time, the constant hidden in the O notation can differ for different automata in the family. In this sense, the definition is nonuniform. Since we are interested primarily in lower bounds

on the state complexity, using this definition yields stronger results than imposing a uniform constant.

Time complexity of randomized finite automata has been considered in previous works. In [Fre81, Theorem 3], it was proven that 2P$_2$FAs can accept some non-regular languages, but only if they have a possibility of infinitely long computations. If all possible computations are finite, 2P$_2$FAs can no longer accept nonregular languages. This result was furthermore strengthened in [DS90]. Here it was proven that any 2P$_2$FA with polynomial time complexity accepts only regular languages. Nevertheless, Theorem 6.2 of [DS90] implies that 2P$_2$FAs running in polynomial time can be exponentially more succinct than 2NFAs.[9]

The previous results cited above show that randomized automata with restricted running time have weaker computational power than those without the time restriction. These results, however, provide only incomplete comparison with other models of automata, such as the nondeterministic ones. For example, it is not clear if any small 2NFA can be simulated by some small 2P$_2$FA with polynomial time complexity. Our results provide a partial negative answer for the corresponding problem of rotating and sweeping automata. More precisely, we show that rotating and sweeping randomized automata restricted to *linear* time can simulate their counterparts with unlimited time only at the cost of exponential blowup in the number of states.

In the rest of this section, we present the lower bound on RP$_2$FAs restricted to linear time. In particular, we prove a hardness propagation from 1P$_1^N$ to lin-RP$_2^N$: If a language family $\mathcal{L}$ is not in 1P$_1^N$, then $\bigwedge\overline{\mathcal{L}}$ is not in lin-RP$_2^N$. When we combine this "upper level" of hardness propagation with the "lower level" formulated in Corollary 3.7, we obtain a language family in $1\Delta -$ lin-RP$_2^N$. This fact proves the separation result (vi) of Figure 3.2b. Furthermore, it implies many other separation results, e. g., a separation between lin-RP$_0^N$ and RP$_0^N$, since $1\Delta \subseteq$ RP$_0^N$ and lin-RP$_0^N \subseteq$ lin-RP$_2^N$.

The main idea of the hardness propagation from 1P$_1^N$ to lin-RP$_2^N$ is to adapt the technique of generic strings (explained in Subsection 2.3.2) for randomized automata. At first, we provide an intuitive explanation of how we can do this. In Subsection 3.4.2, we formalize our arguments. Afterwards, we generalize the lower bound results for sweeping automata in Section 3.5.

When proving the lower bound results, we assume that the transition function of randomized finite automata can contain arbitrary real numbers; this assumption is essential in our arguments. Nevertheless, since we are dealing with lower bounds, the obtained result is stronger as it would be for the model of automata restricted to rational numbers or coin flips. On the other side, it is not difficult to see that the use of arbitrary real numbers is not required for the upper bounds of the proven separation results.

[9] The time complexity of the witness automata family is claimed to be $O(n^2)$. This bound, however, is valid for uniform definition of time complexity. Since the presented witness family consists of one-word languages, the time complexity of the corresponding automata is linear according to our definition.

3.4.1 Upper Level of Hardness Propagation: Intuition

In this subsection, we present the intuition behind the hardness propagation from 1P_1^{N} to lin-RP_2^{N}. We do so by using arguments, mainly based on geometric representation, that are rather incomplete. Nevertheless, we hope that they are helpful for understanding the main idea of the proof and its correspondence to the lower bound proofs for RDFAs based on generic strings. A precise formalization of the proof is done in Subsection 3.4.2, where the geometric representation is no longer used.

The core idea of the generic strings technique for proving lower bounds on RDFAs, as explained in Subsection 2.3.2, is the following one: Consider a RDFA M accepting language L. Take any word y. If M starts in some state q, it will reach some state q' after reading y. Hence, every word y induces a mapping $Q \to Q$, where Q is the set of states of M. The image of Q under this mapping is denoted as $\text{LVIEWS}_M(y)$. We are interested in such a word $y \in L$ that minimizes the size of $\text{LVIEWS}_M(y)$. We call such word as a *generic word*.

When dealing with rotating randomized automata, the configuration of the automaton M accepting language L after reading some input can be described as a probability distribution over its states Q. Thus, if the automaton is started in some probability distribution, after reading some (part of the) input word it ends in another probability distribution. In this way, input words induce a mapping on probability distributions. In the rest of this subsection, we use y to denote both the word y and the mapping induced by y.

The set of all possible probability distributions over Q can be described as a set of points (more precisely, a k-simplex) in the k-dimensional space $\mathbb{R}^k$, where $k = |Q|$. The mapping induced by y transforms this set S into another convex set of points $y(S)$ in $\mathbb{R}^k$; see Figure 3.3. Easily, the set $y(S)$ is a convex hull of at most k points, which we call *vertices*. In an analogy to the case of deterministic automata we want to pick $y \in L$ such that the "size" of $y(S)$ is minimal.

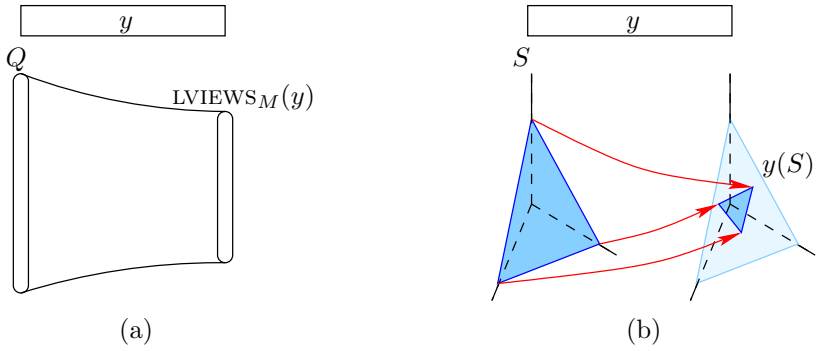

Figure 3.3: The idea of generic strings. (a) Deterministic scenario. (b) Adaptation to the randomized scenario.

However, it is not immediately obvious how to measure the "size" of a convex object in k-dimensional space. As a first guess, we can try to use the volume of the object. Unfortunately, this is not very usable, since the definition of the volume depends on the dimension we are working with. We say that a (convex) object has dimension d if d is the minimal number such that the object is included in some d-dimensional hyperplane. Any convex object of dimension $d' < d$ has zero d-dimensional volume, but nonzero d'-dimensional volume. We can, however, solve this problem by minimizing the dimension d of the object $y(S)$ as the first criterion and minimizing its d-dimensional volume as the second criterion.

Unfortunately, such a definition of a generic word is not sound. We can run into the following problem. There can exist an infinite sequence of words that yield objects with decreasing volume, yet there might not be a single word that yields the object with minimal volume. It may happen that the sequence of words yields objects with volumes that converge to some value that is never reached. To avoid this problem, we work with the mappings $S \to S$ directly instead of working with the words from L. Furthermore, we consider not just all mappings induced by words from L, but also mappings that can be approximated by these mappings arbitrarily well. Now, the set of all considered mappings is closed under limit, i. e., if we have a sequence of convergent mappings, the limit of this sequence is also in the considered set. Because of this property, we can pick a *generic mapping* G that transform S into a set $G(S)$ with minimal dimension (as the first criterion) and minimal volume (as the second criterion). In this subsection, we use the concept of "mapping" and "word" interchangeably. On the intuitive level, we do not distinguish between the generic mapping G and the word whose mapping approximates G very well.

The main idea how the generic strings are used in the deterministic setting is the following one. Consider a RDFA M accepting language $\bigwedge L$, and let G be a generic word of M. For a word x, we look at the behavior of M on input word GxG. After reading G, automaton M reaches a state from $\text{LVIEWS}_M(G)$. The remaining part xG of the input induces a mapping $\text{LMAP}_M(G, xG)$ on the set of states of M: if M starts to read xG in state q, it finishes in $\text{LMAP}_M(G, xG)(q)$. The properties of generic strings ensure that if $x \in L$, then $\text{LMAP}_M(G, xG)$ is a permutation. On the other side, if $x \notin L$, then $\text{LMAP}_M(G, xG)$ is not injective, i. e., it makes the size of $\text{LVIEWS}(GxG)$ smaller than the size of $\text{LVIEWS}(G)$. Using this properties, it is possible to build a small $\cap_{\text{L}}$1DFA for L.

Now we look at the randomized scenario. Consider a RP_2FA M working in linear time and accepting language $\bigwedge L$. Let G be a generic mapping of M. This mapping induces an image $G(S)$ of S; the image $G(S)$ is an analogy of LVIEWS used in the deterministic case. Similarly as in the deterministic case, we consider the behavior of the automaton on word GxG. The mapping xG (an analogy to LMAP) maps $G(S)$ into a convex subset of $G(S)$.

How do the properties of the mapping xG relate to the membership of x in L? We can formulate a statement analogous to the deterministic case: If $x \in L$, the mapping xG permutes the vertices of the convex object $G(S)$. On the other hand, if $x \notin L$, the mapping xG maps $G(S)$ into a convex object with significantly smaller volume,

i. e., the volume of $(GxG)(S)$ is significantly smaller than the volume of $G(S)$; see Figure 3.4 for an illustration of this case.

Equivalently, this means that if $x \in L$, the mapping xG does not change the volume of the convex object, and if $x \notin L$, the mapping xG is far away from any permutation on vertices of $G(S)$: Obviously, if xG decreases the volume of the convex object, it is not a permutation, since every permutation keeps the volume constant. For the other direction, note that xG maps $G(S)$ into a subset of $G(S)$. Hence, if it does not permute vertices, it must shrink the volume of the object, otherwise it would not fit into $G(S)$.

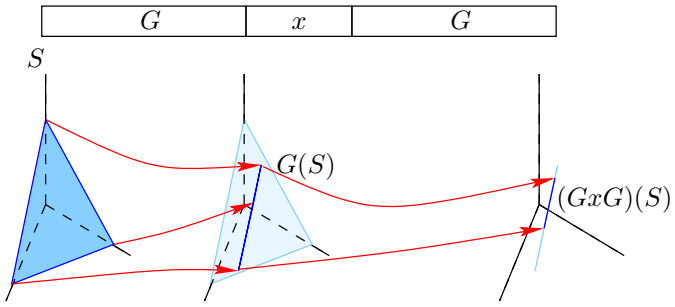

Figure 3.4: The use of generic strings for randomized automata. Mapping xG significantly reduces the volume (i. e., the length of the line segment) of $G(S)$ into the volume of $(GxG)(S)$.

Why is this analogy correct? If $x \in L$, the mapping xG can not decrease the volume of $G(S)$, because of the way we selected the generic string. Indeed, the mapping GxG is a candidate for generic mapping, because $GxG \in \bigwedge L$, and the volume $(GxG)(S)$ is smaller than the volume of $G(S)$. Hence, xG must permute the vertices of $G(S)$. For the other direction, consider some $x \notin L$ and assume that the mapping xG is close to some permutation. Let us look at the word $G(xG)^{k!}$. Since the mapping xG is close to permutation of vertices of $G(S)$, the mapping $(xG)^{k!}$ is close to identity on $G(S)$. So, the linear time randomized automaton can not distinguish between $G(xG)^{k!}$ and G, because a constant number of traversals does not suffice to amplify the probability of the correct answer. Since $x \notin L$, this contradicts to the correctness of the automaton.

In fact, it is not necessary to minimize the volume in the definition of the generic mapping; minimizing the dimension is sufficient. To realize this, let us just look at the arguments we used in the previous paragraph. If $x \notin L$, the mapping xG must be far away from any permutation on vertices of $G(S)$ for the same reason. So, consider the case $x \in L$. If the mapping xG decreases the volume of the convex object, we can iterate the mapping xG to decrease this volume to zero. Hence, the mapping $G(xG)^* = \lim_{i\to\infty} G(xG)^i$ induces a convex object of smaller dimension. The mapping $G(xG)^*$ is a valid candidate for the generic mapping, because every

$G(xG)^i \in \bigwedge L$, and the set of all considered mappings is closed under limit. Hence, we have a contradiction.

How can we exploit the properties of the generic mapping to prove the hardness propagation lemma? In the deterministic case, there exist two states in LVIEWS(G) that are mapped to the same state by xG if and only if $x \notin L$. Using this fact, we are able to construct a $\cap_{\text{L}}$1DFA for language L or, equivalently, a $\cup_{\text{L}}$1DFA for $\overline{L}$ as follows: For every pair of states from LVIEWS(G), there is one state in the constructed $\cap_{\text{L}}$1DFA that simulates the original RDFA. If the two states collapse, the $\cap_{\text{L}}$1DFA knows that the input word is not in L.

Now we focus on the case of randomized automata. Let us assume that $G(S)$ is a d-dimensional simplex, i. e., it is a d-dimensional convex hull of exactly $d+1$ vertices. As we see in the next section, we can always achieve this and it simplifies our arguments. It is easy to see that every point in $G(S)$ can be expressed as a convex combination of the vertices of $G(S)$; a convex combination of vertices $v_1, \ldots, v_{d+1}$ is any point $a_1 v_1 + \ldots + a_{d+1} v_{d+1}$ such that all coefficients a_i are between 0 and 1 and $\sum_i a_i = 1$. Hence, any point v in $G(S)$ can be defined by the corresponding $d+1$ coefficients and, since $G(S)$ is a simplex, these coefficients are unique. We call these coefficients as the *convex coordinates of v with respect to $G(S)$*.

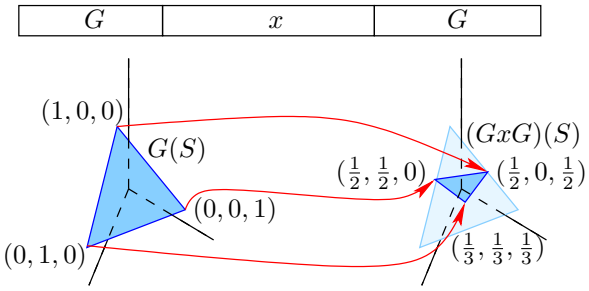

Figure 3.5: Convex coordinates with respect to $G(S)$. Mapping xG is not a permutation, hence some pair of vertices of G maps into points with a common non-zero coordinate. E. g., $(xG)(1,0,0) = (\frac{1}{2}, 0, \frac{1}{2})$, $(xG)(0,1,0) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ – both of the resulting points have the third coordinate nonzero.

Easily, if xG is a permutation on vertices of $G(S)$, then every vertex of $G(S)$ maps into some other vertex, so the coordinates of the resulting vertex are of the form $(0, \ldots, 0, 1, 0, \ldots, 0)$. On the other hand, it is possible to show that if xG is not a permutation, then there are two vertices of $G(S)$ that map into points with a common non-zero coordinate. Furthermore, if xG is far away from any permutation, this common non-zero coordinate is significantly large. This case is illustrated in Figure 3.5.

Our goal is to build a small 1P$_{\text{L}}$FA that detects if there exist two vertices of $G(S)$ that are mapped by xG to points with a common non-zero coordinate. This is easily doable if we are able to construct a one-way randomized automaton that "calculates"

the convex coordinates of the vertices of $(GxG)(S)$ with respect to $G(S)$. More precisely, we construct a following one-way randomized automaton M': For each vertex v_i of $G(S)$ there is one special state q_i of M'. If M' starts in state q_i and reads an input word $\vdash x \dashv$, it ends in a probability distribution defined by the coordinates of $(GxG)(v_i)$, i. e., the probability that M' is in state q_j is equal to the j-th coordinate of $(GxG)(v_i)$ with respect to $G(S)$.

Once we have such an automaton M', we can build the 1P$_1$FA M'' in an analogous way as in the deterministic case. Automaton M'' randomly chooses two states q_i, q_j of M' and simulates two runs of M', started from q_i and q_j, independently. If the same state is reached after reading $\vdash x \dashv$, automaton M'' can be sure that $x \notin L$, since the mapping xG is not a permutation on vertices of $G(S)$. On the other hand, if xG is far away from any permutation, there exist two vertices v_i, v_j of $G(S)$ that are mapped by xG to points with a common non-zero coordinate, and both these points have this coordinate significantly large. Thus, states q_i, q_j are transformed into the same state with significant probability. Hence, automaton M'' accepts $\overline{L}$ with bounded error.

Finally, we discuss how to construct the one-way randomized automaton M' that calculates the coordinates of $(GxG)(S)$. The main idea is that M' simulates the rotating automaton M on x in a straightforward way. To start the simulation, M' started in q_i moves to a probability distribution defined by the (standard) coordinates of vertex v_i of $G(S)$. More precisely, if q is the j-th state of M, then the probability that M' enters q when reading $\vdash$ in q_i is equal to the j-th standard coordinate of vertex v_i. Intuitively, automaton M' transforms the convex coordinates with respect to $G(S)$ into the standard coordinates when reading $\vdash$. Afterwards, M' simulates M on the word x in a straightforward way.

When reaching $\dashv$, automaton M' needs to simulate the behavior of M on G (because M' is simulating mapping xG on the vertices of $G(S)$). Afterwards, M' has to transform the calculated standard coordinates back into the convex coordinates with respect to $G(S)$. This is doable, since every state of the simulated automaton M is transformed into a point from $G(S)$ after reading G. Thus, automaton M' simulates the mapping G and converts the coordinates in one step while reading $\dashv$. Since the standard coordinates representing the j-th state q of M are

$$e_j = (\overbrace{0, \ldots, 0}^{j-1}, 1, 0, \ldots, 0),$$

automaton M' moves from q into q_i with probability equal to the i-th *convex* coordinate of point $G(e_j)$.

To sum up, we have used the given RP$_2$FA M accepting $\bigwedge L$ in linear time to construct a 1P$_1$FA M'' accepting $\overline{L}$ with bounded error. The error bound depends on the properties of the given automaton M, but it is constant for all words in the accepted language. This is sufficient for us, since we are aiming for the hardness propagation from a nonuniformly bounded randomized model. Furthermore, if the given automaton M has k states, the constructed automaton M'' has $O(k^2)$ states. Hence, if a language family $\bigwedge \mathcal{L}$ belongs to lin-RP$_2^{\mathrm{N}}$, then the family $\overline{\mathcal{L}}$ belongs to 1P$_1^{\mathrm{N}}$.

3.4.2 Upper Level of Hardness Propagation: Formal Proof

In the previous subsection, we have presented all ideas that are necessary to prove the hardness propagation from $1\mathrm{P}_1^{\mathrm{N}}$ to lin-$\mathrm{RP}_2^{\mathrm{N}}$. Nevertheless, the formulations we have used are rather imprecise and incomplete. For example, we did not say what does it mean that some value is "significantly" large, we have not proven several claimed facts, etc.

In this subsection, we build a precise proof of the claimed result. Here, the geometric interpretation presented in the previous subsection is no longer necessary. On the other hand, it is not difficult to see the geometric representation behind the algebraic arguments we use.

Our goal is to prove that if $\bigwedge\mathcal{L} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$, then $\overline{\mathcal{L}} \in 1\mathrm{P}_1^{\mathrm{N}}$. Nevertheless, we focus on a language family slightly different from $\bigwedge\mathcal{L}$ for technical reasons: A language family

$$\mathcal{L}^{\#} := (\mathcal{L}\#)^* \tag{3.4}$$

differs from $\bigwedge\mathcal{L}$ only in the missing leftmost delimiter. Easily, if $\bigwedge\mathcal{L} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$, then also $\mathcal{L}^{\#} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$. Indeed, if some rotating randomized automaton M solves a language $\bigwedge L$, it can be easily transformed into an automaton M' that accepts language $L^{\#} = (L\#)^*$. It is sufficient that when M' is going to read the first symbol of the input word, it simulates M on #. I. e., when M' reads $\vdash$ or $\dashv$ (and does not end the computation), it simulates M on $\vdash$# or $\dashv$#, respectively.

Observation 3.3. *If* $\bigwedge\mathcal{L} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$, *then* $\mathcal{L}^{\#} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$.

In the sequel, we introduce the formal machinery needed to prove the hardness propagation, together with some auxiliary lemmas. Afterwards, we prove that if some language $L^{\#}$ can be accepted by a small lin-RP_2FA, then the language $\overline{L}$ can be accepted by a small $1\mathrm{P}_1$FA. This result, together with Observation 3.3, proves that if $\bigwedge\mathcal{L} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$, then $\overline{\mathcal{L}} \in 1\mathrm{P}_1^{\mathrm{N}}$.

It is easy to see that any randomized automaton with k states over alphabet Σ can be transformed into an equivalent one with $k+1$ states that satisfies the following property: The accept state q_a is reachable only when reading $\dashv$ and the automaton never leaves state q_a (i. e., $\delta(q_a, a)(q_a) = 1$ for all symbols $a \in \Sigma \cup \{\vdash, \dashv\}$). We call such automaton as *an automaton with special accept state*.

For technical reasons, we prove the lower bound for lin-RP_2FAs with special accept states only. Since requiring the special accept state increases the state complexity by at most one state, this assumption does not matter when dealing with the complexity classes of the considered automata.

Transition Matrices

Consider any rotating randomized automaton M over an input alphabet Σ and a set of $k-1$ states $Q = \{q_1, \ldots, q_{k-1}\}$. The computation of M can be viewed as a process of transforming probability distributions over $Q \cup \{\perp\}$. At any computation step,

the configuration of the automaton is uniquely defined by a probability distribution $\overline{p} = (p_1, \ldots, p_{k-1}, p_k) \in \mathbb{R}^k$, where, for any $i < k$, p_i is the probability that M is in state q_i and p_k is the probability that M has hanged. It holds that $\sum_{i=1}^{k} p_i = 1$ and, for each i, $p_i \geq 0$. In each step, $\overline{p}$ is transformed according to the transition function of M into another probability distribution.

We can interpret the transition function δ of M as an assignment of a $k \times k$ stochastic matrix to every $a \in \Sigma \cup \{\vdash, \dashv\}$. In particular, $\overline{p}$ is transformed into $\overline{p}\delta(a)$ when reading a. The cell at i-th row and j-th column of the matrix $\delta(a)$ contains the probability that M, being in state q_i, reaches state q_j after reading symbol a (to be formally correct, we equate q_k to $\perp$, i. e., the "hanged state" of M).

The introduced notation allows us to associate every $x = x_1 x_2 \cdots x_n \in \Sigma^*$ with a $k \times k$ stochastic matrix $\delta_x = \delta(x_1)\delta(x_2)\cdots\delta(x_n)$, called the *transition matrix of x (with respect to M)*, describing the behavior of M when reading x: If M is in distribution $\overline{p} \in \mathbb{R}^k$ and reads x, it reaches the distribution $\overline{p}\delta_x$. It is easy to see that $\delta_{xy} = \delta_x \delta_y$. Note that such notation can be used for one-way randomized automata, too.

We treat the transition matrices as points in the metric space of all $k \times k$ matrices. We use the *maximum norm* for matrices; the norm $\|A\|$ of matrix A is defined as the maximum absolute value of some element of A.

When dealing with transition matrices, we use the following lemma:

Lemma 3.11. *Let A, B, C, D be $k \times k$ stochastic matrices such that*

$$\|C - A\|, \|D - B\| \leq \varepsilon \leq 1.$$

Then $\|CD - AB\| \leq (2k+1)\varepsilon$.

Proof. Let $E_A := C - A$ and $E_B = D - B$. Hence $CD - AB = (A + E_A)(B + E_B) - AB$ and we can estimate its norm as follows:

$$\begin{aligned} \|(A + E_A)(B + E_B) - AB\| &= \|AE_B + E_A B + E_A E_B\| \leq \\ &\leq \|AE_B\| + \|E_A B\| + \|E_A E_B\| \leq \\ &\leq \varepsilon + k\varepsilon + k\varepsilon^2 \leq (2k+1)\varepsilon \end{aligned}$$

□

Closures of Languages

Consider any RP$_2$FA $M = (q_s, \delta, q_a)$. Let $L \subseteq \Sigma^*$ be any language. By $\mathcal{C}_L$ we denote the set of all matrices A that can be approximated arbitrarily well (in terms of the maximum norm) by a transition matrix of some word in L:

$$\mathcal{C}_L := \{A \in \mathbb{M} \mid (\forall \varepsilon > 0)(\exists w \in L)\, \|A - \delta_w\| \leq \varepsilon\}$$

where $\mathbb{M}$ is the set of all $k \times k$ matrices. We call $\mathcal{C}_L$ as the *closure of L* (with respect to M).

The closure of language formalizes the concept of "taking all mappings that are approximable arbitrarily well", which we presented in Subsection 3.4.1. Now we present several properties of language closures that are not difficult to prove.

Lemma 3.12. *The following properties of language closures hold for any languages* L, L_1, L_2*:*

1. *If* $A \in \mathcal{C}_L$*, then* A *is a stochastic matrix.*
2. *If* $L_1 \subseteq L_2$*, then* $\mathcal{C}_{L_1} \subseteq \mathcal{C}_{L_2}$*.*
3. $\mathcal{C}_L$ *is a closed set: For any convergent sequence* $\{A_i\}_{i=1}^{\infty}$ *of matrices from* $\mathcal{C}_L$*, we have* $\lim_{i\to\infty} A_i \in \mathcal{C}_L$*.*
4. $\mathcal{C}_L$ *is a compact set, i. e., it is possible to pick a convergent subsequence from any infinite sequence of members of* $\mathcal{C}_L$*.*
5. *If* $A_1 \in \mathcal{C}_{L_1}$ *and* $A_2 \in \mathcal{C}_{L_2}$ *then* $A_1 A_2 \in \mathcal{C}_{L_1 L_2}$*.*
6. *Let* $L \subseteq \Sigma^*$ *be a language over alphabet* Σ *such that* $\# \notin \Sigma$*. Let* $A_1, \ldots, A_m$ *be matrices from the set* $\mathcal{C}_{L^\#} \cup \{\delta_{x\#} \mid x \in \overline{L}\}$*, i. e., every* A_i *either is in* $\mathcal{C}_{L^\#}$*, or corresponds to* $x\#$ *for some* $x \in \Sigma^* - L$*. If every* A_i *is in* $\mathcal{C}_{L^\#}$*, then* $\prod_{i=1}^{m} A_i \in \mathcal{C}_{L^\#}$*. Otherwise,* $\prod_{i=1}^{m} A_i \in \mathcal{C}_{\overline{L^\#}}$*.*

Proof.

1. If A is a transition matrix of some word, it is stochastic. Otherwise, A is a limit of a sequence of stochastic matrices. Easily, this implies that A is stochastic.
2. Let $A \in \mathcal{C}_{L_1}$. If $A = \delta_w$ for some $w \in L_1$, then $w \in L_2$, hence $A \in \mathcal{C}_{L_2}$. If A is approximable arbitrarily well by transition matrices of words from L_1, it is trivially approximable by transition matrices of words from L_2 as well.
3. Consider $\lim_{i\to\infty} A_i$. This matrix is, by the definition of the limit, approximable arbitrarily well by some matrix A_i. I. e., for any $\varepsilon > 0$, there is an A_i such that $\|A - A_i\| < \varepsilon$. By definition of $\mathcal{C}_L$, matrix A_i is approximable arbitrarily well by some transition matrix δ_w for some $w \in L$. I. e., $\|A_i - \delta_w\| < \varepsilon$. So, matrix A is approximable arbitrarily well by δ_w, since $\|A - \delta_w\| \leq \|A - A_i\| + \|A_i - \delta_w\| \leq 2\varepsilon$. Hence, $A \in \mathcal{C}_L$ by the definition of language closures.
4. Set $\mathcal{C}_L$ is closed. Furthermore, $\mathcal{C}_L$ is bounded, since it contains only stochastic matrices and every element of every stochastic matrix is from interval $\langle 0, 1\rangle$. Hence, $\mathcal{C}_L$ is compact due to the Bolzano-Weierstrass theorem.
5. The claim follows from Lemma 3.11. More precisely, let $A_1 \in \mathcal{C}_{L_1}$ and $A_2 \in \mathcal{C}_{L_2}$. We show that for every $\varepsilon > 0$ there exists some $w \in L_1 L_2$ such that $\|A_1 A_2 - \delta_w\| \leq \varepsilon$. By the definition of language closures, there exist w_1 and w_2 such that $\|A_1 - \delta_{w_1}\|, \|A_2 - \delta_{w_2}\| \leq \varepsilon/(2k+1)$. Let $w := w_1 w_2$. Since $\delta_w = \delta_{w_1}\delta_{w_2}$, the claim follows by applying Lemma 3.11 for $A := \delta_{w_1}$, $B := \delta_{w_2}$, $C := A_1$, $D := A_2$.

6. Let $\Psi_i := L^{\#}$ for each i such that $A_i \in \mathcal{C}_{L^{\#}}$, and let $\Psi_i := \overline{L}\#$ otherwise (note that by $\overline{L}$ we mean the complement with respect to alphabet Σ, i. e., the language $\Sigma^* - L$). Since $A_i \in \mathcal{C}_{\Psi_i}$, applying the statement 5 yields that $\prod_{i=1}^{m} A_i \in \mathcal{C}_{\Psi_1 \dots \Psi_m}$.

   If every Ψ_i equals to $L^{\#}$, it holds that $\Psi_1 \dots \Psi_m \subseteq L^{\#}$, because the language $L^{\#}$ is closed under concatenation. Hence, the statement 2 implies that $\mathcal{C}_{\Psi_1 \dots \Psi_m} \subseteq \mathcal{C}_{L^{\#}}$, so we have $\prod_{i=1}^{m} A_i \in \mathcal{C}_{L^{\#}}$.

   On the other hand, if some Ψ_i equals to $\overline{L}\#$, every word from language $\Psi_1 \dots \Psi_m$ belongs to $\overline{L^{\#}}$, because at least one #-delimited block of this word does not belong to L. Hence, by the same argument as in the previous case, we have $\mathcal{C}_{\Psi_1 \dots \Psi_m} \subseteq \mathcal{C}_{\overline{L^{\#}}}$, thus $\prod_{i=1}^{m} A_i \in \mathcal{C}_{\overline{L^{\#}}}$.

□

Consider a lin-RP$_2$FA M with special accept state accepting language L with error bounded by ε. Another property of closures is that if M works correctly, the closure of L and the closure of its complement $\overline{L}$ are separated by some constant $\gamma > 0$, i. e., if $A_1 \in \mathcal{C}_L$ and $A_2 \in \mathcal{C}_{\overline{L}}$ then $\|A_1 - A_2\| \geq \gamma$. This constant γ depends on ε and on the automaton M.

Informally, if this is not the case, then there are some $x \in L$ and $y \notin L$ such that x and y induce transition matrices that are arbitrarily close. Since M works in linear expected time, it makes only constant number of traversals through the input word, hence it must end the computation for x and y in arbitrarily close probability distributions. Since M is a bounded-error automaton, it either accepts or rejects both x and y, what is a contradiction.

Lemma 3.13. *Let M be a* lin-RP$_2$FA *with special accept state accepting language L. Let $\mathcal{C}_L$ ($\mathcal{C}_{\overline{L}}$) be the closure of L ($\overline{L}$) with respect to M. There exists some $\gamma > 0$ such that for every $A_1 \in \mathcal{C}_L$, $A_2 \in \mathcal{C}_{\overline{L}}$ it holds that $\|A_1 - A_2\| \geq \gamma$.*

Proof. Let M accepts L with error bound ε. Assume that the statement of the lemma does not hold, i. e., for every $\gamma > 0$ there exist $A_1 \in \mathcal{C}_L$ and $A_2 \in \mathcal{C}_{\overline{L}}$ such that $\|A_1 - A_2\| < \gamma$. This, together with the definition of the language closure, implies that for every $\gamma > 0$ there exist $x \in L$ and $y \in \overline{L}$ such that $\|\delta_x - \delta_y\| < \gamma$.

Since M is works with an error bounded by ε, it accepts every $x \in L$ and rejects every $y \in \overline{L}$ with probability at least $1/2 + \varepsilon$. Hence it must hold that $\Pr[M \text{ accepts } x] - \Pr[M \text{ accepts } y] \geq 2\varepsilon$.

Since M works in linear expected time, assume that the expected running time of M on any input word z is bounded by $c|z|$ for some constant c. This is equivalent to the statement that the expected number of traversals of M through z is bounded by c. Using the Markov bound, we obtain:

$$\begin{aligned}\Pr[M \text{ accepts } x] = {} & \Pr[M \text{ accepts } x \text{ in} < c/\varepsilon \text{ traversals}] + \\ & + \Pr[M \text{ accepts } x \text{ in} \geq c/\varepsilon \text{ traversals}] \leq \\ \leq {} & \Pr[M \text{ accepts } x \text{ in} < c/\varepsilon \text{ traversals}] + \varepsilon\end{aligned}$$

Hence, we have

$$\begin{aligned} 2\varepsilon &\leq \Pr[M \text{ accepts } x] - \Pr[M \text{ accepts } y] \\ &\leq \Pr[M \text{ accepts } x \text{ in} < c/\varepsilon \text{ traversals}] + \varepsilon - \Pr[M \text{ accepts } y \text{ in} < c/\varepsilon \text{ traversals}], \end{aligned}$$

what is equivalent to

$$\varepsilon \leq \Pr[M \text{ accepts } x \text{ in} < c/\varepsilon \text{ traversals}] - \Pr[M \text{ accepts } y \text{ in} < c/\varepsilon \text{ traversals}]. \quad (3.5)$$

As M can accept only at the right endmarker, it can accept x only after t complete traversals. The transformation of the probability distributions of the automaton after doing t complete traversals is defined by the following stochastic matrix:

$$\delta(\vdash)\,(\delta_x \delta(\dashv))^t$$

where $\delta(a)$ is the transition matrix of M at symbol a.

Since M is an automaton with special accept state, it is easy to see that the probability that x is accepted by M after at most t complete traversals is determined by the element of the above-mentioned matrix at the row corresponding to q_s (i. e., the start state of M) and the column corresponding to q_a (i. e., the accept state of M).

Since $\|\delta_x - \delta_y\| < \gamma$, we can use Lemma 3.11 to estimate the probabilities in (3.5):

$$\begin{aligned} \varepsilon &\leq \left|\Pr[M \text{ accepts } x \text{ in} < c/\varepsilon \text{ traversals}] - \Pr[M \text{ accepts } y \text{ in} < c/\varepsilon \text{ traversals}]\right| \leq \\ &\leq (2k+1)^{(2c/\varepsilon)}\gamma. \end{aligned}$$

As γ can be chosen arbitrarily small, we have a contradiction. □

Auxiliary Lemmas

To prove the hardness propagation result, we need two more lemmas about the properties of stochastic matrices. The first one deals with determinants of stochastic matrices; the proof is a straightforward induction on the matrix size. The second one states a convergence property of Markov chains.

Lemma 3.14. *Let A be a $k \times k$ nonnegative matrix such that the sum of elements in any row is at most 1. Then $|\det(A)| \leq 1$. Furthermore, $|\det(A)| = 1$ if and only if A is a permutation matrix.*

Proof. Induction on k. Case $k = 1$ is trivial. Let $k > 1$. Let $a_{i,j}$ be the value of A in i-th row and j-th column. Let $A_{1,j}$ be the matrix obtained from A by deleting the first row and i-th column. Obviously, $A_{1,j}$ fulfills the requirements of the lemma.

It holds that $\det(A) = \sum_{i=1}^{k} (-1)^{i+1} a_{1,i} \det(A_{1,i})$. Since $|\det(A_{1,i})| \leq 1$, it holds that

$$|\det(A)| \leq \sum_{i=1}^{k} a_{1,i} |\det(A_{1,i})| \leq \sum_{i=1}^{k} a_{1,i} \leq 1,$$

hence the first claim of the lemma holds.

Now we prove the second claim. If A is a permutation matrix, then there exists a unique i such that $a_{1,i} = 1$, $a_{1,j} = 0$ for all $j \neq i$ and $A_{1,i}$ is a $(k-1) \times (k-1)$ permutation matrix. Hence

$$|\det(A)| = \left|(-1)^{i+1} a_{1,i} \det(A_{1,i})\right| = |\det(A_{1,i})| = 1.$$

To prove the other implication, assume that $|\det(A)| = 1$, hence

$$1 \leq |\det(A)| \leq \sum_{i=1}^{k} a_{1,i} |\det(A_{1,i})|.$$

Since (by induction hypothesis) $|\det(A_{1,i})| \leq 1$ the previous statement holds only if $\sum_{i=1}^{k} a_{1,i} = 1$ and $a_{1,i} > 0 \Rightarrow |\det(A_{1,i})| = 1$. By induction hypothesis, $|\det(A_{1,i})| = 1$ holds only if $A_{1,i}$ is a permutation matrix. If $A_{1,i}$ is a permutation matrix for some i, then $a_{j,i} = 0$ for all $j > 1$, hence $A_{1,i'}$ is not a permutation matrix for any $i' \neq i$. Hence there is exactly one $a_{1,i} > 0$, so $a_{1,i} = 1$ and $A_{1,i}$ is a permutation matrix. Thus A is a permutation matrix, too. □

Lemma 3.15. *Let A be a $k \times k$ stochastic matrix. The matrix $A^\infty := \lim_{i\to\infty} A^{ik!}$ exists. Furthermore, there exists a $k \times k$ permutation matrix P and a $(k - \operatorname{rank}(A^\infty)) \times \operatorname{rank}(A^\infty)$ nonnegative matrix R with row sums equal to 1 such that A^∞ satisfies*

$$A^\infty = P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1} A^\infty$$

where I is the $\operatorname{rank}(A^\infty) \times \operatorname{rank}(A^\infty)$ identity matrix.

Proof. We rely on the statements from [Gan59, Chapter XIII]. From §7.5 and formula (80), it follows that $k!$ is a multiple of the period of A, hence the limit $\lim_{i\to\infty} A^{i(k!)}$ exists. Thus, we can use the results from §7.2: there is a normal form of $A^{k!}$ equal to

$$A^{k!} = P_1 \begin{pmatrix} Q_1 & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & Q_g & \mathbf{0} & \cdots & \mathbf{0} \\ U_{g+1,1} & \cdots & U_{g+1,g} & Q_{g+1} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ U_{s,1} & \cdots & U_{s,g} & U_{s,g+1} & \cdots U_{s,s-1} & Q_s \end{pmatrix} P_1^{-1}$$

for some permutation matrix P_1 (note that $P^T = P^{-1}$ for every permutation matrix P and that PBP^{-1} permutes both rows and columns of B by P). Hence we can

use [Gan59, (102)] to obtain:

$$A^\infty = P_1 B P_1^{-1}; \quad B = \begin{pmatrix} Q_1^\infty & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & Q_g^\infty & \mathbf{0} \\ U'_{g+1,1} & \cdots & U'_{g+1,g} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ U'_{s,1} & \cdots & U'_{s,g} & \mathbf{0} \end{pmatrix} \tag{3.6}$$

Furthermore, every Q_i^∞ is a stochastic matrix with all rows identical. Since $A^\infty = A^\infty A^\infty$, it must hold that $U'_{g+i,j} = U'_{g+i,j} Q_j^\infty$.

Let r_i be the row of B that contains the first row of the submatrix Q_i^∞. It is easy to see that every row B_j of the matrix B can be expressed as $\sum_{i=1}^g c_{j,i} B_{r_i}$ such that $c_{j,i} \geq 0$ and $\sum_{i=1}^g c_{j,i} = 1$: If B_j corresponds to the row containing some submatrix Q_i^∞, we can take $c_{j,i} = 1$ and $c_{j,i'} = 0$ for all $i' \neq i$. If B_j corresponds to the n-th row of the submatrix

$$U'_{m,1} \ldots U'_{m,g} \mathbf{0} = (U'_{m,1} Q_1^\infty) \ldots (U'_{m,g} Q_g^\infty) \mathbf{0},$$

we can define $c_{j,i}$ as the sum of all elements in the n-th row of $U'_{m,i}$. Since B itself is stochastic, such a definition satisfies the requirements $c_{j,i} \geq 0$ and $\sum_{i=1}^g c_{j,i} = 1$.

Let P_2 be a permutation that maps rows $r_1, \ldots, r_g$ to rows $1, \ldots g$ (we do not care about the rest of P_2). Then it holds that

$$P_2 B = \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P_2 B \tag{3.7}$$

where I is the $g \times g$ identity matrix and R is a $(k-g) \times g$ matrix. Furthermore, each row of R is equal to $c_{j,1} \ldots c_{j,g}$ for some j, hence R is a nonnegative matrix with row sums equal to 1.

Putting (3.6) and (3.7) together yields

$$A^\infty \quad = \quad P_1 P_2^{-1} \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P_2 P_1^{-1} P_1 B P_1^{-1} = P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1} A^\infty$$

for the permutation matrix $P := P_1 P_2^{-1}$. Since $\operatorname{rank}(A^\infty) = \operatorname{rank}(B) = g$ and I is of size $g \times g$, the claim of the lemma follows. □

Reduction to 1P1FAs

Now we are ready to proceed to the core of the proof of the hardness propagation. Let L be any language over alphabet Σ. In the rest of this subsection, let us assume that a $(k-1)$-state lin-RP2FA M with special accept state accepts language $L^\#$. We denote the transition function of M as δ and the transition matrix of x with respect to M as δ_x. All language closures considered are with respect to M. We show that there exists a 1P1FA M' with $O(k^2)$ states that accepts language $\overline{L}$. At first, we define the generic matrix.

Definition 3.6. *A $k \times k$ matrix $G \in \mathcal{C}_{L^\#}$ is called generic if and only if* $\operatorname{rank}(G)$ *is minimal among all members of $\mathcal{C}_{L^\#}$ and it holds that*

$$G = P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1} G$$

for the $\operatorname{rank}(G) \times \operatorname{rank}(G)$ *identity matrix I, some permutation matrix P and some nonnegative matrix R with row sums equal to 1, respectively.*

This definition corresponds to the ideas described in Subsection 3.4.1: The property of the minimal rank corresponds to the requirement of minimal dimension of the set $G(S)$ (as used in Subsection 3.4.1), the second property represents the requirement that $G(S)$ is a simplex.

It is not difficult to prove that some generic matrix exists: since $\mathcal{C}_{L^\#}$ is nonempty, it contains a matrix G' where G' has minimal rank. Consider the matrix

$$G := \lim_{i \to \infty} G'^{ik!}.$$

The existence of the limit follows from the first claim of Lemma 3.15. Since $L^\#$ is closed under concatenation, $G'^j \in \mathcal{C}_{L^\#}$ for any j due to Lemma 3.12(5, 2). Lemma 3.12(3) yields that $G \in \mathcal{C}_{L^\#}$. Since $\operatorname{rank}(AB) \leq \min(\operatorname{rank}(A), \operatorname{rank}(B))$ for any matrices A, B, and (what is not difficult to check) $G = GG'^{k!}$, it holds that $\operatorname{rank}(G) \leq \operatorname{rank}(G')$. By the minimality of $\operatorname{rank}(G')$, we have $\operatorname{rank}(G) = \operatorname{rank}(G')$. Combined with the second claim of Lemma 3.15, matrix G is generic.

For the rest of this subsection, we fix some generic matrix G, as well as the corresponding matrices P, I, and R.

Let x be any word over the working alphabet of M. We consider the matrix $G\delta_{x\#}G$ (note that this corresponds to the mapping discussed in Subsection 3.4.1). Using the properties of generic matrices, we obtain the equality

$$G\delta_{x\#}G = P \begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix} P^{-1} G \tag{3.8}$$

where S_x is some $\operatorname{rank}(G) \times \operatorname{rank}(G)$ stochastic matrix associated with the word x. Indeed, by the properties of generic matrices, we have that

$$\begin{aligned} G\delta_{x\#}G &= P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1} G \delta_{x\#} P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1} G = \\ &= P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} \begin{pmatrix} A_x & B_x \\ C_x & D_x \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1} G \end{aligned}$$

for some matrices A_x, B_x, C_x, D_x (such that A_x has the same size as I and C_x has the same size as R). Then it holds that

$$\begin{aligned}\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} \begin{pmatrix} A_x & B_x \\ C_x & D_x \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} &= \begin{pmatrix} A_x & B_x \\ RA_x & RB_x \end{pmatrix} \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} = \\ &= \begin{pmatrix} A_x + B_x R & \mathbf{0} \\ RA_x + RB_x R & \mathbf{0} \end{pmatrix} = \\ &= \begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix}\end{aligned} \tag{3.9}$$

for $S_x := A_x + B_x R$. Since the matrix described in (3.9) is defined as a multiplication of several stochastic matrices, it must be stochastic, too. Hence, S_x is stochastic as well.

We call S_x as a *stamp of* x. It can be shown that the stamp of x is unique because it is a $\text{rank}(G) \times \text{rank}(G)$ matrix. Indeed, due to the second property of generic matrices required by Definition 3.6, it holds

$$P^{-1}G = \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1}G.$$

Combined with the fact that I and R are matrices with $\text{rank}(G) = \text{rank}(P^{-1}G)$ columns, it implies that the first $\text{rank}(G)$ rows of $P^{-1}G$ are linearly independent. Suppose that there are two different stamps S_x, S'_x of x. Then it holds

$$G\delta_{x\#}G = P \begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix} P^{-1}G = P \begin{pmatrix} S'_x & \mathbf{0} \\ RS'_x & \mathbf{0} \end{pmatrix} P^{-1}G,$$

hence we have

$$\begin{aligned}\begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix} P^{-1}G &= \begin{pmatrix} S'_x & \mathbf{0} \\ RS'_x & \mathbf{0} \end{pmatrix} P^{-1}G \\ \begin{pmatrix} S_x - S'_x & \mathbf{0} \\ RS_x - RS'_x & \mathbf{0} \end{pmatrix} P^{-1}G &= \mathbf{0},\end{aligned}$$

what contradicts to the linear independence of the first $\text{rank}(G)$ rows of $P^{-1}G$.

Nevertheless, we do not rely on the uniqueness of the stamp of x in our arguments; it is sufficient that the stamp S_x satisfies condition (3.8).

Intuitively, the stamp of x describes the convex coordinates of the vertices of $G(S)$ mapped by xG, as illustrated in Figure 3.5 in Subsection 3.4.1. Each row of S_x contains the convex coordinates of one mapped vertex. The first $\text{rank}(G)$ rows of $P^{-1}G$ contain the standard coordinates of the vertices of $G(S)$.

In Subsection 3.4.1, we have described how the properties of a stamp of x differ for $x \in L^{\#}$ and for $x \notin L^{\#}$. In the sequel, we formalize this description.

Lemma 3.16. *Let S_x be a stamp of x. If $x \in L$, then S_x is a permutation matrix. On the other hand, for some $\varepsilon > 0$ and every $x \notin L$, S_x differs from any permutation matrix P by at least ε, i. e., $\|S_x - P\| \geq \varepsilon$.*

Proof. Assume that the first statement does not hold, hence there exists some $x \in L$ such that S_x is not a permutation matrix. Consider the matrix

$$H := \lim_{i\to\infty} (G\delta_{x\#})^{ik!} G.$$

Due to Lemma 3.15, H is well-defined, and due to Lemma 3.12(6, 3), $H \in \mathcal{C}_{L^\#}$. Using (3.8), we can write

$$\begin{aligned} H &= \lim_{i\to\infty} P \begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix}^{ik!} P^{-1}G = P \begin{pmatrix} S_x^{ik!} & \mathbf{0} \\ RS_x^{ik!} & \mathbf{0} \end{pmatrix} P^{-1}G = \\ &= P \begin{pmatrix} S_x^{\infty} & \mathbf{0} \\ RS_x^{\infty} & \mathbf{0} \end{pmatrix} P_L^{-1} G_L \end{aligned}$$

where $S_x^\infty := \lim_{i\to\infty} S_x^{ik!}$. Since S_x is a stochastic matrix, but not a permutation matrix, due to Lemma 3.14 we have $\det(S_x) < 1$, what implies that $\det(S_x^\infty) = 0$, hence $\operatorname{rank}(S_x) < \operatorname{rank}(G)$. Since it holds that

$$H = P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} \begin{pmatrix} S_x^{\infty} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} P^{-1}G,$$

we have that $\operatorname{rank}(H) < \operatorname{rank}(G)$, in contradiction with the genericity of G.

In the sequel, we focus on the second statement of the lemma. Let $\varepsilon := \gamma/(2k+1)^{k!+2}$, where γ is the constant provided by Lemma 3.13. Assume that the statement does not hold for such ε, hence there exists some $x \notin L$ such that S_x is ε-close to some permutation P_1 matrix, i. e., $\|S_x - P_1\| < \varepsilon$. Consider the matrix

$$X := (G\delta_{x\#})^{k!}\, G.$$

Due to Lemma 3.12(6), $X \in \mathcal{C}_{\overline{L^\#}}$. Using (3.8), we can express X as

$$X = P \begin{pmatrix} S_x^{k!} & \mathbf{0} \\ RS_x^{k!} & \mathbf{0} \end{pmatrix} P^{-1}G.$$

Applying Lemma 3.11, we obtain

$$\left\|S_x^{k!} - P_1^{k!}\right\| < \varepsilon(2k+1)^{k!-1}.$$

Since $P_1^{k!}$ is an identity matrix, we have (again by Lemma 3.11):

$$\gamma = \varepsilon(2k+1)^{k!+2} > \left\| P \begin{pmatrix} S_x^{k!} & \mathbf{0} \\ RS_x^{k!} & \mathbf{0} \end{pmatrix} P^{-1}G - P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1}G \right\| = \|X - G\|$$

This, however, contradicts Lemma 3.13, because $X \in \mathcal{C}_{\overline{L^\#}}$ and $G \in \mathcal{C}_{L^\#}$. □

In the sequel, we need to construct a small 1P$_1$FA that can detect whether the stamp of the input word is a permutation matrix or not. To do so, we employ a one-way randomized automaton that "calculates" the stamp of the input word, as discussed in Subsection 3.4.1.

Lemma 3.17. *There exists a one-way randomized automaton M_S with k states over the alphabet Σ such that its transition matrix associated with $\vdash x \dashv$ has the form*

$$\delta^{M_S}(\vdash x \dashv) = \begin{pmatrix} S_x & \mathbf{0} \\ * & * \end{pmatrix},$$

where $*$ denotes some matrix and S_x is the left stamp of x. We call the automaton M_S as the stamp automaton *of M.*

Proof. Intuitively, M_S running on the input x simulates M on $yx\#$, where y is a word such that δ_y is infinitely close to G. Afterwards, M_S makes a suitable move on $\dashv$. We define its transition function as follows:[10]

$$\begin{aligned} \delta^{M_S}(\vdash) &= P^{-1}G; \\ \delta^{M_S}(a) &= \delta(a) \text{ for } a \in \Sigma; \\ \delta^{M_S}(\dashv) &= \delta(\#)P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix}. \end{aligned}$$

Now we need to show that our construction is correct. Since all matrices involved in the definition are stochastic, the definition of δ^{M_S} is valid. Easily, the transition matrix associated with $\vdash x \dashv$ with respect to M_S is

$$\delta^{M_S}(\vdash x \dashv) = P^{-1}G\delta_{x\#}P \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix}.$$

Hence it is sufficient to check that

$$S_x := \left(P^{-1}G\delta_{x\#}P \begin{pmatrix} I \\ R \end{pmatrix} \right)_{1\ldots\mathrm{rank}(G)}$$

satisfies (3.8); $A_{1\ldots m}$ denotes the matrix consisting of the first m rows of matrix A. I. e., we need to show that

$$G\delta_{x\#}G = P \begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix} P^{-1}G.$$

[10] The transition function of M_S is complete, so we may omit the row and the column corresponding to $\perp$ in the definition of M_S. Hence, we define the value of transition function $\delta^{M_S}(a)$ as a $k \times k$ matrix instead of a $(k+1) \times (k+1)$ matrix.

It holds that

$$\begin{aligned}
\begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix} &= \begin{pmatrix} \left(P^{-1}G\delta_{x\#}P\begin{pmatrix} I \\ R \end{pmatrix}\right)_{1\ldots\mathrm{rank}(G)} & \mathbf{0} \\ R\left(P^{-1}G\delta_{x\#}P\begin{pmatrix} I \\ R \end{pmatrix}\right)_{1\ldots\mathrm{rank}(G)} & \mathbf{0} \end{pmatrix} = \\
&= \begin{pmatrix} \left(P^{-1}G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix}\right)_{1\ldots\mathrm{rank}(G)} \\ R\left(P^{-1}G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix}\right)_{1\ldots\mathrm{rank}(G)} \end{pmatrix} = \\
&= \begin{pmatrix} \left(P^{-1}\right)_{1\ldots\mathrm{rank}(G)} G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} \\ R\left(P^{-1}\right)_{1\ldots\mathrm{rank}(G)} G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} \end{pmatrix} = \\
&= \begin{pmatrix} I \\ R \end{pmatrix}\left(P^{-1}\right)_{1\ldots\mathrm{rank}(G)} G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} = \\
&= \begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1}G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix}
\end{aligned}$$

Hence, by Definition 3.6 we have

$$P\begin{pmatrix} S_x & \mathbf{0} \\ RS_x & \mathbf{0} \end{pmatrix} P^{-1}G = P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1}G\delta_{x\#}P\begin{pmatrix} I & \mathbf{0} \\ R & \mathbf{0} \end{pmatrix} P^{-1}G = G\delta_{x\#}G.$$

□

We are ready to construct the 1P$_1$FA M' that recognizes $\overline{L}$. The idea behind M' is to simulate two independent runs of the stamp automaton M_S. When reading $\vdash$, M' picks uniformly at random a pair of different states of M_S among its first $\mathrm{rank}(G)$ states and simulates the first step of M_S. When reading a symbol from Σ, M' makes independent random decisions in both simulated runs. When reaching $\dashv$, M' simulates the last step of M_S. If both simulated runs of M_S reach the same state, the stamp of the input word x is not a permutation matrix, hence (by Lemma 3.16) $x \notin L$ and M' accepts x. If the two reached states are different, M' hangs. The described construction yields k^2 states. In addition, we need the start state and the accept state. Hence, the total number of states of M' is bounded by $k^2 + 2$.

Now we discuss the correctness of the described construction. Easily, every word $x \in L$ is rejected by M': since S_x is a permutation matrix by Lemma 3.16, no computation of M' accepts x. On the other hand, if $x \notin L$, then, due to Lemma 3.16, S_x is separated from every permutation matrix by some fixed $\varepsilon > 0$. In each row of S_x, there is at least one element of value at least $1/\mathrm{rank}(G)$, because S_x is a $\mathrm{rank}(G) \times \mathrm{rank}(G)$ stochastic matrix. Since ε can be w.l.o.g. small enough (more precisely, w.l.o.g. $\varepsilon \leq (\mathrm{rank}(G) - 1)/\mathrm{rank}(G)$), there is at least one element of value

at least $\beta := \varepsilon/(\mathrm{rank}(G) - 1)$ in each row of S_x. (Note that $\mathrm{rank}(G) \geq 2$, since otherwise S_x is a 1×1 stochastic matrix and hence it is a permutation matrix.) If no column of S_x contains more than one element of value at least β, every row and every column of S_x contains exactly one element of value at least β; furthermore, each such element has value greater than $1 - (\mathrm{rank}(G) - 1)\beta = 1 - \varepsilon$. Hence, if we round all such elements to 1 and all other elements to 0, we obtain a permutation matrix P such that $\|P - S_x\| < \varepsilon$, what contradicts Lemma 3.16. Thus, there exists a column of S_x that contains at least two elements of value at least β. The automaton M' picks the two rows containing these elements with probability

$$\frac{1}{\binom{\mathrm{rank}(G)}{2}} \geq \frac{1}{\binom{k}{2}} \geq \frac{1}{k^2}$$

and, if these rows are picked, the word is accepted with probability at least β^2. Hence, we have that every $x \notin L$ is accepted with probability at least

$$\eta := \frac{\beta^2}{k^2} \geq \frac{\varepsilon^2}{k^4} > 0.$$

To sum up, the arguments we have presented prove the following theorem:

Theorem 3.18. *Let M be a $(k-1)$-state* lin-RP_2FA *with special accept state that solves the language $L^{\#}$. Then there exists a* $1\mathrm{P}_1$FA *M' with $k^2 + 2$ states that solves the language $\overline{L}$.*

Since any randomized automaton can be transformed into an equivalent automaton with special accept state by adding at most one new state, we may apply the result of Theorem 3.18 to language families:

Corollary 3.19. *If $\mathcal{L}^{\#} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$, then $\overline{\mathcal{L}} \in 1\mathrm{P}_1^{\mathrm{N}}$.*

Combining the result with Observation 3.3, we obtain:

Corollary 3.20. *If $\bigwedge\mathcal{L} \in$ lin-$\mathrm{RP}_2^{\mathrm{N}}$, then $\overline{\mathcal{L}} \in 1\mathrm{P}_1^{\mathrm{N}}$. Equivalently, if $\mathcal{L} \notin 1\mathrm{P}_1^{\mathrm{N}}$, then $\bigwedge\overline{\mathcal{L}} \notin$ lin-$\mathrm{RP}_2^{\mathrm{N}}$.*

At the end of the current section, we are ready to collect the results of the proven hardness propagation:

Lemma 3.21. *Consider language family $\mathcal{J}$ defined in (2.9). It holds that*

$$\bigwedge\left(\overline{\bigwedge\mathcal{J}}\right) \in 1\Delta - \text{lin-}\mathrm{RP}_2^{\mathrm{N}}$$

Proof. Due to Lemma 2.19, $\mathcal{J} \in 1\Delta$. Using the closure properties in Figure 2.4 [B6, B1, B6], we have that $\bigwedge\mathcal{J} \in 1\Delta$, $\overline{\bigwedge\mathcal{J}} \in 1\Delta$, and $\bigwedge\left(\overline{\bigwedge\mathcal{J}}\right) \in 1\Delta$. On the other hand, $\mathcal{J} \notin 1\mathrm{D}$ due to Lemma 2.19. By Corollary 3.7, $\bigwedge\mathcal{J} \notin 1\mathrm{P}_1^{\mathrm{N}}$ and, by Corollary 3.20, $\bigwedge\left(\overline{\bigwedge\mathcal{J}}\right) \notin$ lin-$\mathrm{RP}_2^{\mathrm{N}}$. □

Note that any language family in 1Δ, $1\mathrm{P}_1^{\mathrm{N}}$, as well as lin-$\mathrm{RP}_2^{\mathrm{N}}$ is closed under union and intersection with any family in 1D. Hence, we can plug Lemma 2.1 in the proof of the previous lemma to show that

$$\bigwedge\bigvee\overline{\mathcal{J}} \in 1\Delta - \text{lin-}\mathrm{RP}_2^{\mathrm{N}}.$$

Obviously, $1\Delta \subseteq \mathrm{R}\Delta$. As proven in Subsection 3.2.1, $\mathrm{R}\Delta = \mathrm{RP}_0^{\mathrm{N}} = \mathrm{RP}_0^{\varepsilon}$. Hence, the language family $\bigwedge\bigvee\overline{\mathcal{J}}$ can be solved by a small family of rotating LasVegas automata with uniformly bounded error, at the cost of exponential running time.[11] At the same time, this language family can not be solved by small linear-time rotating automata, even in the very strong model of two-sided error with nonuniform error bound. This result immediately implies that lin-$\mathrm{RP}_0^{\mathrm{N}} \subsetneq \mathrm{RP}_0^{\mathrm{N}} = \mathrm{RP}_0^{\varepsilon}$ and lin-$\mathrm{RP}_2^{\mathrm{N}} \subsetneq \mathrm{RP}_2^{\mathrm{N}}$. Hence, restricting the running time to linear decreases the power of rotating randomized automata. Furthermore, this restriction can not be compensated by using less restrictive (i. e., more powerful) model of error bound.

The presented result has also an interesting relationship with the complexity theory of randomized Turing Machines. It is known that the space-restricted complexity classes of LasVegas Turing machines are equivalent to the nondeterministic classes: Due to the well-known Immerman-Szelepczényi theorem, the complexity class of nondeterministic Turing machines with space complexity $f(n)$ is closed under complement for $f(n) = \Omega(\log n)$. Any nondeterministic Turing machine working with space complexity $f(n)$ can be simulated by a bounded one-sided error Monte-Carlo Turing machine within the same space complexity $f(n)$ [MS99]. It is not difficult to see that if there exist space-bounded one-sided error Monte-Carlo Turing machines for language L and its complement $\overline{L}$, then there also exists a space-bounded LasVegas Turing machine for L.

This power of randomization, however, depends on the possibility of infinite (more precisely, arbitrarily long) computations. Indeed, the construction of [MS99] produces machines for which computations of any length are possible, although their probabilities tend to zero with growing length. It is an interesting question if this property is necessary, i. e., if LasVegas randomization can be of any significant help if the running time is required to have some fixed upper bound.

Our results partially answer this question for the case of rotating finite automata (we generalize it for sweeping automata in Section 3.5 as well). To see this, note that any rotating randomized automaton that works with some fixed deterministic upper bound on its running time cannot make more than constant number of traversals through the input. Otherwise, there exists a computation that makes a non-constant number of traversals, what immediately implies that there is a nonzero probability that the automaton runs in a loop. Thus, the automaton can run in this loop arbitrarily long with nonzero probability. The requirement of constant number of traversals in the worst case is equivalent with the requirement of linear running time in the

[11] We can claim the exponential upper bound here, since it is not difficult to check that the construction of [MS99] as well as its adaptation for LasVegas automata in [HS01b] yields automata with expected running time at most exponential.

worst case for randomized rotating automata. We have proven that restricting the automata to linear running time decreases their power, even if this restriction applies to expected time only. Thus, imposing a deterministic upper bound on the running time decreases their power as well. Hence, forbidding infinite computations may cause exponential blowup in the state complexity of rotating randomized finite automata.

3.5 Sweeping Automata with Linear Running Time

Up to now, we focused on the rotating randomized automata with restricted running time. Nevertheless, it is possible to generalize all these results for sweeping automata as well. In this subsection, we explain how to do this generalization.

The results of hardness propagation for rotating randomized automata, i. e., Corollary 3.7 and Corollary 3.20, are analogous to the results for rotating deterministic automata, i. e., Corollary 2.5 and Corollary 2.12 (see Figure 2.3). In the structure of the hardness propagation, the usage of $\cap_1$1DFA in the deterministic case corresponds to 1P$_1$FA in the randomized case.

To extend our results for sweeping randomized automata, we need results analogous to Corollary 2.7 and Corollary 2.14. To formulate the analogous claims, however, we need a randomized automaton that corresponds to $\cap_2$1DFA. This new model needs to have similar relationship to 1P$_1$FA as $\cap_2$1DFA has to $\cap_1$1DFA: Intuitively, we need to somehow add a capability of reading the input in both directions to 1P$_1$FA. To do so, we define a *direction-choosing automaton*. This newly-defined model consists of two components that work as one-way randomized automata. The first one, called the *left component*, reads the input word from left to right and the second one, called the *right component*, reads the input from right to left. At the beginning of the computation, one of the components is chosen uniformly at random. This component is then started and the result of its computation is taken as the result of the computation of the direction-choosing automaton. The direction-choosing automaton has to satisfy the same requirements as a Monte-Carlo automaton with one-sided error, i. e., any word not in the solved language is accepted with probability 0, and any word in the language is accepted with probability at least ε for some fixed constant $\varepsilon > 0$.

Definition 3.7. *A* direction-choosing automaton *(*DCP$_1$FA *for short) M is a tuple $M = (M_1, M_2)$, where M_1 and M_2 are one-way randomized finite automata. We say that M accepts language L if and only if there exists some $\varepsilon > 0$, called the* error bound of M*, such that:*

1. *For any $x \notin L$, it holds that*

$$\Pr[M_1 \text{ accepts } x] = \Pr[M_2 \text{ accepts } x^R] = 0$$

2. *and, for any $x \in L$, it holds that*

$$\Pr[M_1 \text{ accepts } x] \geq \varepsilon \quad \text{or} \quad \Pr[M_2 \text{ accepts } x^R] \geq \varepsilon.$$

Note that an equivalent definition can be obtained by requiring

$$\frac{1}{2}\Pr[M_1 \text{ accepts } x] + \frac{1}{2}\Pr[M_2 \text{ accepts } x^R] = 0$$

for any $x \notin L$ and requiring

$$\frac{1}{2}\Pr[M_1 \text{ accepts } x] + \frac{1}{2}\Pr[M_2 \text{ accepts } x^R] \geq \varepsilon'$$

for any $x \in L$. This alternative definition follows the intuitive description provided above more closely, but the definition we are using is slightly more convenient in our proofs.

In the definition above, we focused on the bounded one-sided error Monte-Carlo variant of the direction-choosing automaton. While it makes sense to define also other variants, the bounded one-sided error Monte-Carlo is the one that fits in our hardness propagation framework.

In this section, we deal with the class of language families that can be solved by small DCP$_1$FAs with nonuniformly bounded error. Consistently with our previous naming convention, we denote this class as DCP$_1^N$. More precisely, we say that $\mathcal{L} = (L_n)_{n\geq 1} \in$ DCP$_1^N$ if and only if there exists a sequence $(M_n)_{n\geq 1} = \left((M_n^L, M_n^R)\right)_{n\geq 1}$ of DCP$_1$FAs such that M_n solves L_n with error bounded by ε_n and the number of states of M_n^L as well as M_n^R is bounded by some polynomial $p(n)$. The error bound ε_n can be different for various n. Again, it makes sense to define the other classes corresponding to direction-choosing automata, but these classes are not needed in the presented framework of hardness propagation.

In this section, we extend the hardness propagation results for sweeping automata. At first, we focus on the "lower level" of hardness propagation in Subsection 3.5.1. Here we prove a result analogous to Corollary 2.7 that shows how to propagate hardness from 1P$_1^N$ to DCP$_1^N$. Afterwards, we generalize the results of Section 3.4. We do this in Subsection 3.5.2 by proving a result analogous to Corollary 2.14 that shows how to propagate hardness from DCP$_1^N$ to lin-SP$_2^N$. Results presented in this section have been also published in [Krá09].

3.5.1 Lower Level of Hardness Propagation

In this section, we focus on the "lower level" of the hardness propagation, i. e., the hardness propagation from 1P$_1^N$ to DCP$_1^N$. We prove that if there is no small 1P$_1$FA accepting language L, then there is no small DCP$_1$FA accepting language $L \wedge L^R$. The result is formulated as the following lemma, which is an analogy of Lemma 2.6. Informally, the main idea of the proof is the following one: We use the concept of "confusion" as introduced in Section 3.3. If there is a small DCP$_1$FA M for language $L \wedge L^R$, consider its left component. Since there is no small 1P$_1$FA solving L, this left component can be, due to Lemma 3.8, confused with respect to L by arbitrarily high probability by some word $w_L \in L$. Similarly, the right component of M can be confused by some

$w_R \in L$. Afterwards, it is not difficult to check that the word $\#w_L\#w_R^{\mathrm{R}}\#$ cannot be accepted by M with a significant probability.

Lemma 3.22. *If there is no* 1P$_1$FA *accepting language L with at most m states, then there is no* DCP$_1$FA *accepting language $L \wedge L^{\mathrm{R}}$ such that both its left and its right component have at most $m-1$ states. An analogous result holds for $L \oplus L^{\mathrm{R}}$.*

Proof. We focus on the claim for $L \wedge L^{\mathrm{R}}$, the proof for $L \oplus L^{\mathrm{R}}$ is analogous. Assume that the statement does not hold, i. e., there is no 1P$_1$FA accepting L with at most m states, but there exists some DCP$_1$FA accepting $L \wedge L^{\mathrm{R}}$ such that both its components have at most $m-1$ states. In that case, there exists also a DCP$_1$FA $M = (M^L, M^R)$ accepting $L \wedge L^{\mathrm{R}}$ such that both M^L and M^R are non-hanging (in the sense of Definition 3.1) and have at most m states. Assume that M works with error bound ε. It is not difficult to see that there is no 1P$_1$FA accepting $\#L$ with at most m states. Indeed, any such 1P$_1$FA M_1 can be transformed into a 1P$_1$FA M_2 with at most m states that accepts L. It is sufficient that M_2 simulates M_1 in a straightforward way, except that when M_2 reads $\vdash$, it simulates M_1 on $\vdash\#$. Hence, by the equivalence of statements 1 and 3 of Lemma 3.8 we have that for non-hanging one-way randomized automaton M' with at most m states there is some word $w' \in \#L$ that non-confuses M' with probability less that ε. Since both M^L and M^R are non-hanging one-way randomized automata, this gives us words $w_L, w_R \in \#L$.

Consider word $w := w_L\#w_R^{\mathrm{R}}$. Easily, $w \in L \wedge L^{\mathrm{R}}$. Now we analyze the probability that M^L accepts w. Consider any accepting computation of M^L and let q be the state of M^L after reading $\vdash w_L$. Easily, q is a non-confused state of M^L with respect to $\#L$. Otherwise, the same computation can happen for some $w' := w'_L\#w_R^{\mathrm{R}}$ with nonzero probability, where $w'_L \notin \#L$, i. e., $w' \notin L \wedge L^{\mathrm{R}}$. This, however, means that M accepts some word not in $L \wedge L^{\mathrm{R}}$ with nonzero probability, what contradicts to the definition of DCP$_1$FAs.

Hence, the probability that M^L accepts w is upper bounded by the probability that M^L is in a non-confused state after reading w_L, which is less than ε by our choice of w_L. Using a symmetric argument, the probability that M^R accepts w is less than ε, too. This, however, contradicts to our assumption about error bound of M. □

Using Lemma 3.22, we immediately obtain a hardness propagation result from 1P$_1^{\mathrm{N}}$ to DCP$_1^{\mathrm{N}}$.

Corollary 3.23. *If $\mathcal{L} \notin$ 1P$_1^{\mathrm{N}}$, then $\mathcal{L} \wedge \mathcal{L}^{\mathrm{R}} \notin$ DCP$_1^{\mathrm{N}}$ and $\mathcal{L} \oplus \mathcal{L}^{\mathrm{R}} \notin$ DCP$_1^{\mathrm{N}}$.*

3.5.2 Upper Level of Hardness Propagation

In the sequel, we adapt the results proven in Section 3.4 for sweeping automata. In particular, we prove the hardness propagation from DCP$_1^{\mathrm{N}}$ to lin-SP$_2^{\mathrm{N}}$. All arguments presented in Section 3.4 for rotating automata can be generalized in a straightforward way. In this subsection, we describe the necessary changes to the proofs for rotating automata presented in Subsection 3.4.2.

Similarly as for the case of rotating automata, we will deal with $\mathcal{L}^{\#}$ instead of $\bigwedge\mathcal{L}$. It is easy to see that we can do this, because if $\bigwedge\mathcal{L} \in$ lin-SP_2^{N}, then $\mathcal{L}^{\#} \in$ lin-SP_2^{N}, too. In other words, Observation 3.3 holds for sweeping automata, too. Hence, our goal is to prove that if $\mathcal{L}^{\#} \in \text{SP}_2^{\text{N}}$, then $\overline{\mathcal{L}} \in \text{DCP}_1^{\text{N}}$.

When proving lower bounds on linear-time randomized sweeping automata, we use the notion of automata with special accept state introduced in Subsection 3.4. It is easy to observe that, similarly as in the rotating case, any randomized sweeping automaton with m states can be converted into an equivalent $(m+1)$-state automaton with special accept state.

Transition Matrix Pairs

Similarly to the case of rotating automata, consider any sweeping randomized automaton M over an input alphabet Σ and a set of $k-1$ states $Q = \{q_1, \ldots, q_{k-1}\}$. At any computation step, the configuration of the automaton is uniquely defined by a probability distribution $\overline{p} = (p_1, \ldots, p_{k-1}, p_k) \in \mathbb{R}^k$, where, for any $i < k$, p_i is the probability that M is in state q_i and p_k is the probability that M has hanged. We can interpret the transition function δ of M as an assignment of a $k \times k$ stochastic matrix to every $a \in \Sigma \cup \vdash, \dashv$. In every computation step, the probability distribution $\overline{p}$ describing the configuration of M is transformed into $\overline{p}\delta(a)$, where a is the symbol being read.

In the case of rotating automata, we associated every word $x = x_1 x_2 \cdots x_n \in \Sigma^*$ with its transition matrix δ_x that described the behavior of the automaton when reading x. The difference between rotating and sweeping automata is that while rotating automata always read the input word from left to right, sweeping automata can read the word in both directions. For this reason, we associate the word x with two transition matrices when dealing with sweeping automata. The *left transition matrix of* x, denoted as δ_x^L, describes the behavior of M while reading x from left to right; it holds that $\delta_x^L = \delta(x_1)\delta(x_2)\cdots\delta(x_n)$. Next, we define the *right transition matrix of* x, denoted as δ_x^R, in a symmetric way: δ_x^R describes the behavior of M while reading x from right to left and it holds that $\delta_x^R = \delta(x_n)\delta(x_{n-1})\ldots\delta(x_1)$. Hence, every $x \in \Sigma^*$ can be associated with a pair of stochastic matrices $\mathbf{D}_x = (\delta_x^L, \delta_x^R)$, called the *transition pair of* x.

We extend the matrix operations $+, -, \cdot$ to matrix pairs as follows:

$$(A, B) \pm (C, D) := (A \pm C, B \pm D), \quad (A, B)(C, D) := (AC, DB).$$

It is easy to see that, for any x and y, it holds that $\delta_{xy}^L = \delta_x^L \delta_y^L$ and $\delta_{xy}^R = \delta_y^R \delta_x^R$. Hence, we have that $\mathbf{D}_{xy} = \mathbf{D}_x \mathbf{D}_y$.

In Subsection 3.4.2, we considered a metric space of all $k \times k$ matrices. In the sequel, we use a metric space of all pairs of $k \times k$ matrices instead. To be able to do so, we define a *maximum norm* for matrix pair $\|(A, B)\|$ as the maximum of $\|A\|$ and $\|B\|$, where $\|X\|$ denotes the maximum norm of matrix X.

It is an easy observation that Lemma 3.11 holds for matrix pairs, too. For this reason we invoke this lemma for matrix pairs in the sequel, even if the original statement deals with matrices only.

Closures of Languages

It is possible to generalize the concept of language closures for transition pairs in a straightforward way, just by replacing matrices by matrix pairs. Hence, for any SP$_2$FA $M = (q_s, \delta, q_a)$ and any language Ψ we can define the *closure of* Ψ with respect to M, denoted as $\mathcal{C}_\Psi$, as

$$\mathcal{C}_\Psi := \{\mathbf{A} \in \mathbb{M} \mid (\forall \varepsilon > 0)(\exists w \in \Psi)\, \|\mathbf{A} - \mathbf{D}_w\| \leq \varepsilon\},$$

where $\mathbb{M}$ is the set of all pairs of $k \times k$ matrices.

It is not difficult to check that all claims of Lemma 3.12 hold for our new definition of language closures. For this reason, we reference this lemma also in connection with sweeping automata. The Lemma 3.13 holds for sweeping automata as well. The proof for sweeping automata is very similar, but not completely identical. For this reason, we state the analogous lemma now:

Lemma 3.24. *Let M be a* lin-SP$_2$FA *with special accept state accepting language L with bounded error. Let $\mathcal{C}_L$ ($\mathcal{C}_{\overline{L}}$) be the closure of L ($\overline{L}$) with respect to M. There exists some $\gamma > 0$ such that for every $\mathbf{A}_1 \in \mathcal{C}_L$, $\mathbf{A}_2 \in \mathcal{C}_{\overline{L}}$ it holds that $\|\mathbf{A}_1 - \mathbf{A}_2\| \geq \gamma$.*

Proof. We adapt the proof of Lemma 3.13. Let M be a lin-SP$_2$FA with special accept state accepting language L with error bounded by ε. Assume that the statement of the lemma does not hold. Hence, for every $\gamma > 0$ there exist $x \in L$ and $y \in \overline{L}$ such that $\|\mathbf{D}_x - \mathbf{D}_y\| < \gamma$. Assume that the expected running time of M on any input word z is bounded by $c|z|$ for some constant c. Using the same arguments as in Lemma 3.13, we have that

$$\varepsilon \leq \Pr[M \text{ accepts } x \text{ in} < c/\varepsilon \text{ traversals}] - \Pr[M \text{ accepts } y \text{ in} < c/\varepsilon \text{ traversals}]. \tag{3.10}$$

Again, M can accept only at the right endmarker. While every traversal of a rotating automaton ends at the right endmarker, only every odd traversal of a sweeping automaton does so. Hence, M can accept input word x only after $2l + 1$ complete traversals. The transformation of the probability distributions of the automaton after doing $2l + 1$ complete traversals is defined by the following stochastic matrix:

$$\left(\delta(\vdash)\delta_x^L\delta(\dashv)\delta_x^R\right)^l \delta(\vdash)\delta_x^L\delta(\dashv)$$

where $\delta(a)$ is the transition matrix of M at symbol a.

Analogous to the case of rotating automata, the probability that x is accepted by lin-SP$_2$FA M with special accept state after at most $2l + 1$ complete traversals is determined by the element of the above-mentioned matrix at the row corresponding

to q_s (i. e., the start state of M) and the column corresponding to q_a (i. e., the accept state of M).

Since $\|\mathbf{D}_x - \mathbf{D}_y\| < \varepsilon$, we can do similar calculation as in the proof of Lemma 3.13 and estimate the sum (3.10) using Lemma 3.11. More precisely, let l be the largest integer such that $2l + 1 < c/\varepsilon$. Then, it holds that $l < c/(2\varepsilon)$. Hence it is sufficient to apply Lemma 3.11 $4l + 2 < 2c/\varepsilon + 2$ times:

$$\begin{aligned} \varepsilon &\leq \Pr[M \text{ accepts } x \text{ in } < c/\varepsilon \text{ traversals}] - \\ &\quad \Pr[M \text{ accepts } y \text{ in } < c/\varepsilon \text{ traversals}] = \\ &= \Pr[M \text{ accepts } x \text{ in } \leq 2l+1 \text{ traversals}] - \\ &\quad \Pr[M \text{ accepts } y \text{ in } \leq 2l+1 \text{ traversals}] \leq \\ &\leq (2k+1)^{(2c/\varepsilon+2)}\gamma. \end{aligned}$$

As γ can be chosen arbitrarily small, we have a contradiction. □

Reduction to Direction-Choosing Automata

In the case of rotating automata, we have proven a reduction to 1P$_1$FAs. Now, we generalize this result for the case of sweeping automata, i. e., we prove a reduction to DCP$_1$FAs. As in the rotating case, let L be any language over alphabet Σ. In the rest of this subsection, let us assume that a $(k-1)$-state lin-SP$_2$FA M with special accept state accepts language $L^{\#}$. We denote the transition function of M as δ and the transition pair of x with respect to M as $\mathbf{D}_x$. All language closures considered are with respect to M. We show that there exists a DCP$_1$FA M' with $O(k^2)$ states that accepts language $\overline{L}$.

Analogously as in the rotating case, we can define matrix pairs that are left generic and matrix pairs that are right generic. Afterwards, we define generic matrix pairs as those that are simultaneously left generic and right generic.

Definition 3.8. *A matrix pair $(G_L, G_R) \in \mathcal{C}_{L^{\#}}$ is called left (right) generic if and only if* $\operatorname{rank}(G_L)$ *(*$\operatorname{rank}(G_R)$*) is minimal among all left (right) components of members of $\mathcal{C}_{L^{\#}}$ and it holds that*

$$G_L = P_L \begin{pmatrix} I_L & \mathbf{0} \\ R_L & \mathbf{0} \end{pmatrix} P_L^{-1} G_L$$

$$\left(G_R = P_R \begin{pmatrix} I_R & \mathbf{0} \\ R_R & \mathbf{0} \end{pmatrix} P_R^{-1} G_R \right)$$

for the $\operatorname{rank}(G_L) \times \operatorname{rank}(G_L)$ *identity matrix I_L (*$\operatorname{rank}(G_R) \times \operatorname{rank}(G_R)$ *identity matrix I_R), some permutation matrix P_L (P_R) and some nonnegative matrix R_L (R_R) with row sums equal to 1, respectively.*

A matrix pair is called generic if and only if it is both left and right generic.

To see that a generic matrix pair exists, we use similar arguments as for proving the existence of a generic matrix for the rotating case. Since $\mathcal{C}_{L^{\#}}$ is nonempty, it contains

a pair $\mathbf{L} = (L_L, L_R)$ such that L_L has minimal rank and a pair $\mathbf{R} = (R_L, R_R)$ such that R_R has minimal rank. Consider the pair

$$\mathbf{G} = (G_L, G_R) := \lim_{i\to\infty} (\mathbf{LR})^{ik!}.$$

The existence of the limit follows from the first claim of Lemma 3.15; it is easy to see that the lemma can be applied on both components of the matrix pairs. Since $L^{\#}$ is closed under concatenation, $(\mathbf{LR})^j \in \mathcal{C}_{L^{\#}}$ for any j due to Lemma 3.12(5, 2). Lemma 3.12(3) yields that $\mathbf{G} \in \mathcal{C}_{L^{\#}}$. Using the same arguments as in the rotating case ($\operatorname{rank}(AB) \leq \min(\operatorname{rank}(A), \operatorname{rank}(B))$ for any matrices A, B and $\mathbf{G} = \mathbf{G}(\mathbf{LR})^{k!}$), we have that $\operatorname{rank}(G_L) = \operatorname{rank}(L_L)$ and $\operatorname{rank}(G_R) = \operatorname{rank}(R_R)$. Again, combining this result with the second claim of Lemma 3.15, we have that $\mathbf{G}$ is generic.

For the rest of this subsection, we fix some generic matrix pair $\mathbf{G} = (G_L, G_R)$, as well as the corresponding matrices P_L, I_L, R_L, P_R, I_R, and R_R.

Let x be any word over the working alphabet of M. In an analogy to the stamp of x used in the rotating case, we can define the *left stamp* S_x^L of x and the *right stamp* S_x^R of x such that the following equalities hold:

$$G_L \delta_{x\#}^L G_L = P_L \begin{pmatrix} S_x^L & \mathbf{0} \\ R_L S_x^L & \mathbf{0} \end{pmatrix} P_L^{-1} G_L \tag{3.11}$$

$$G_R \delta_{x\#}^R G_R = P_R \begin{pmatrix} S_x^R & \mathbf{0} \\ R_R S_x^R & \mathbf{0} \end{pmatrix} P_R^{-1} G_R \tag{3.12}$$

The left stamp S_x^L (the right stamp S_x^R) is a $\operatorname{rank}(G_L) \times \operatorname{rank}(G_L)$ ($\operatorname{rank}(G_R) \times \operatorname{rank}(G_R)$) stochastic matrix associated with the word x, respectively. The arguments why S_x^L and S_x^R exist are completely analogous to the rotating case. The situation with the uniqueness of S_x^L and S_x^R is analogous to the rotating case as well: It is possible to prove the uniqueness of S_x^L and S_x^R, but this fact is not required in our arguments.

Now we are ready to generalize Lemma 3.16.

Lemma 3.25. *Let S_x^L (S_x^R) be a left (right) stamp of x. If $x \in L$, then both S_x^L and S_x^R are permutation matrices. On the other hand, for some $\varepsilon > 0$ and every $x \notin L$, S_x^L or S_x^R differs from any permutation matrix P by at least ε (i. e., $\left\|S_x^L - P\right\| \geq \varepsilon$ or $\left\|S_x^R - P\right\| \geq \varepsilon$).*

Proof. Assume that the first statement does not hold, hence there exists some $x \in L$ such that w.l.o.g. S_x^L is not a permutation matrix. We define the matrix pair $\mathbf{H}$ in an analogous way to the matrix H used in the rotating case:

$$\mathbf{H} := \lim_{i\to\infty} (\mathbf{G}\mathbf{D}_{x\#})^{ik!}\mathbf{G}.$$

Using the same arguments as in Lemma 3.16, $\mathbf{H}$ is well-defined and $\mathbf{H} \in \mathcal{C}_{L^{\#}}$. Consider the left component H_L of $\mathbf{H}$. We follow the proof of Lemma 3.16, using H_L instead

of H. This yields that $\det\big((S_x^L)^\infty\big) = 0$ and that

$$H_L = P_L \begin{pmatrix} I_L & \mathbf{0} \\ R_L & \mathbf{0} \end{pmatrix} \begin{pmatrix} (S_x^L)^\infty & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} P_L^{-1} G_L.$$

Hence, we have that $\operatorname{rank}(H_L) < \operatorname{rank}(G_L)$, in contradiction with the genericity of $\mathbf{G}$.

Next, we focus on the second statement of the lemma. Again, the proof is analogous to the proof of Lemma 3.16. Let $\varepsilon := \gamma/(2k+1)^{k!+2}$, where γ is the constant provided by Lemma 3.24. Assume that the statement does not hold for such ε, hence there exists some $x \notin L$ such that both S_x^L and S_x^R are ε-close to some permutation matrix, i. e., $\big\|S_x^L - P_1\big\| < \varepsilon$ and $\big\|S_x^R - P_2\big\| < \varepsilon$. Consider the matrix pair

$$\mathbf{X} := (\mathbf{G}\mathbf{D}_{x\#})^{k!}\,\mathbf{G}.$$

By the same arguments as in the proof of Lemma 3.16, we have that $\mathbf{X} \in \mathcal{C}_{\overline{L^\#}}$ and we can express the left component of $\mathbf{X}$ as

$$X_L = P_L \begin{pmatrix} (S_x^L)^{k!} & \mathbf{0} \\ R_L(S_x^L)^{k!} & \mathbf{0} \end{pmatrix} P_L^{-1} G_L.$$

Following the proof of Lemma 3.16, we obtain

$$\gamma = \varepsilon(2k+1)^{k!+2} > \left\| P_L \begin{pmatrix} (S_x^L)^{k!} & \mathbf{0} \\ R_L(S_x^L)^{k!} & \mathbf{0} \end{pmatrix} P_L^{-1} G_L - P_L \begin{pmatrix} I_L & \mathbf{0} \\ R_L & \mathbf{0} \end{pmatrix} P_L^{-1} G_L \right\| =$$
$$= \|X_L - G_L\|$$

Since an analogous statement holds for the right component X_R of $\mathbf{X}$, we have that $\|\mathbf{X} - \mathbf{G}\| < \gamma$, which contradicts Lemma 3.24, because $\mathbf{X} \in \mathcal{C}_{\overline{L^\#}}$ and $\mathbf{G} \in \mathcal{C}_{L^\#}$. □

Both the left and the right stamp can be "calculated" by a small one-way randomized automaton, in the same way as the stamp can be calculated in the rotating case. Hence, we are ready to generalize Lemma 3.17:

Lemma 3.26. *There exists a one-way randomized automaton M_L with k states over the alphabet Σ such that its transition matrix associated with $\vdash x \dashv$ has the form*

$$\delta^{M_L}(\vdash x \dashv) = \begin{pmatrix} S_x^L & \mathbf{0} \\ * & * \end{pmatrix},$$

where $*$ denotes some matrix and S_x^L is the left stamp of x. We call the automaton M_L as the left stamp automaton *of M.*

Analogously, there exists a one-way randomized automaton M_R, called as the right stamp automaton *of M, such that its transition matrix associated with $\vdash x^{\mathrm{R}} \dashv$ has the form*

$$\delta^{M_R}(\vdash x^{\mathrm{R}} \dashv) = \begin{pmatrix} S_x^R & \mathbf{0} \\ * & * \end{pmatrix},$$

where $*$ denotes some matrix and S_x^R is the right stamp of x.

The proof of Lemma 3.26 is analogous to the proof of Lemma 3.17.

The construction of the direction-choosing automaton M' that recognizes $\overline{L}$ is similar to the construction of a 1P$_1$FA in the rotating case: The left component of M' simulates the left stamp automaton of M and the right component of M' simulates the right stamp automaton of M, in the same way as done in the rotating case. In this way, both the left and the right component of M' have k^2+2 states.

Consider any word $x \in L$. Since both the left stamp S_x^L and the right stamp S_x^R of x are permutation matrices due to Lemma 3.25, we can use the same arguments as in the rotating case to prove that both the left and the right component of M' accept x with zero probability. On the other hand, consider any $x \notin L$. In this case, Lemma 3.25 ensures that S_x^L or S_x^R is separated from any permutation matrix by at least ε for some fixed constant $\varepsilon > 0$. Again, using the same arguments as in the rotating case, we have that the left component of M' or the right component of M' accepts x with probability at least η for some fixed constant $\eta > 0$ that depends on k and ε. Hence, we have shown that M' is a valid DCP$_1$FA for language $\overline{L}$ in the sense of Definition 3.7. Furthermore, both components of M' have number of states bounded by a polynomial in k (more precisely, k^2+2).

Hence, we can formulate the proven results in the next theorem, which is a sweeping analogy of Theorem 3.18:

Theorem 3.27. *Let M be a $(k-1)$-state* lin-SP$_2$FA *with special accept state that solves the language $L^{\#}$. Then there exists a* DCP$_1$FA *M' such that both its left and its right component have at most k^2+2 states and M' solves language $\overline{L}$.*

The hardness propagation from DCP$_1^{\text{N}}$ to lin-SP$_2^{\text{N}}$ is a direct corollary of Theorem 3.27:

Corollary 3.28. *If $\mathcal{L}^{\#} \in$ lin-SP$_2^{\text{N}}$, then $\overline{\mathcal{L}} \in$ DCP$_1^{\text{N}}$.*

Again, we can combine the result with Observation 3.3:

Corollary 3.29. *If $\bigwedge\mathcal{L} \in$ lin-SP$_2^{\text{N}}$, then $\overline{\mathcal{L}} \in$ DCP$_1^{\text{N}}$. Equivalently, if $\mathcal{L} \notin$ DCP$_1^{\text{N}}$, then $\bigwedge\overline{\mathcal{L}} \notin$ lin-SP$_2^{\text{N}}$.*

Next, we generalize Lemma 3.21 to separate 1Δ and lin-SP$_2^{\text{N}}$:

Lemma 3.30. *Consider language family $\mathcal{J}$ defined in (2.9). It holds that*

$$\bigwedge\overline{(\bigwedge\mathcal{J}) \wedge (\bigwedge\mathcal{J})^{\text{R}}} \in 1\Delta - \text{lin-SP}_2^{\text{N}}$$

Proof. Due to Lemma 2.19, $\mathcal{J} \in 1\Delta$. Using the closure properties in Figure 2.4 [B6, B2, B3, B1, B6], we have that $\bigwedge\mathcal{J} \in 1\Delta$, $(\bigwedge\mathcal{J})^{\text{R}} \in 1\Delta$, $(\bigwedge\mathcal{J}) \wedge (\bigwedge\mathcal{J})^{\text{R}} \in 1\Delta$, $\overline{(\bigwedge\mathcal{J}) \wedge (\bigwedge\mathcal{J})^{\text{R}}} \in 1\Delta$, $\bigwedge\overline{(\bigwedge\mathcal{J}) \wedge (\bigwedge\mathcal{J})^{\text{R}}} \in 1\Delta$.

On the other hand, $\mathcal{J} \notin 1\text{D}$ due to Lemma 2.19. Corollary 3.7 yields that $\bigwedge\mathcal{J} \notin$ 1P$_1^{\text{N}}$, Corollary 3.23 yields that $(\bigwedge\mathcal{J}) \wedge (\bigwedge\mathcal{J})^{\text{R}} \notin$ DCP$_1^{\text{N}}$, and Corollary 3.29 yields that $\bigwedge\overline{(\bigwedge\mathcal{J}) \wedge (\bigwedge\mathcal{J})^{\text{R}}} \notin$ lin-SP$_2^{\text{N}}$. □

Again, we can use the fact that any language family from 1Δ, $1\mathrm{P}_1^{\mathrm{N}}$, $\mathrm{DCP}_1^{\mathrm{N}}$, as well as lin-$\mathrm{SP}_2^{\mathrm{N}}$ is closed under union and intersection with any family from 1D. Hence, we can apply Lemma 2.1 and Observation 2.1 to modify the proof of the previous lemma to obtain

$$\bigwedge \left(\left(\bigvee \overline{\mathcal{J}} \right) \vee \left(\bigvee \overline{\mathcal{J}} \right)^{R} \right) \in 1\Delta - \text{lin-}\mathrm{SP}_2^{\mathrm{N}}.$$

Similarly as in the rotating case, this result easily implies that lin-$\mathrm{SP}_0^{\mathrm{N}} \subsetneq \mathrm{SP}_0^{\mathrm{N}} = \mathrm{SP}_0^{\mathcal{E}}$ and lin-$\mathrm{SP}_2^{\mathrm{N}} \subsetneq \mathrm{SP}_2^{\mathrm{N}}$. Hence, restricting the running time to linear decreases the power of sweeping randomized automata and this restriction cannot be traded for a more powerful model of error bound.

Easily, the observation about the possibility of infinite computations that we discussed for the case of rotating automata holds for sweeping automata as well: The power of sweeping randomized automata relies heavily on the possibility of arbitrarily long computation. Forbidding the possibility of infinite computations, i. e., enforcing a fixed deterministic upper bound on the computation length, may cause exponential blowup in the state complexity of sweeping randomized finite automata.

Chapter 4

Conclusion

The main goal of this thesis is to present a complexity theory of finite automata. We have used the approach suggested in [SS78] to build such a theory in an analogous way to the well-known complexity theory of Turing machines. We have presented a comprehensive overview of the relationships between the introduced complexity classes of finite automata. These results help us to understand the relationship between determinism, nondeterminism and randomization. While we know only very few results about the relationship between these models of computation for Turing machines, we were able to show several facts for the simpler models of finite automata.

Many results used to build the map of the complexity classes of finite automata have been known already, or followed from known facts about communication complexity in a straightforward way. We have, however, proven several new results: We have shown that determinism is strictly weaker than self-verifying nondeterminism for rotating and sweeping finite automata of restricted size. To do so, we have introduced the technique of hardness propagation and used the ideas of [Sip80b]. Furthermore, using the hardness propagation technique, it is possible to construct witness language families for several other previously known results in a systematic way. Since self-verifying nondeterminism coincides with LasVegas randomization for rotating and sweeping automata, our result shows the gap between determinism and LasVegas randomization as well.

The most interesting new result proven in this thesis is the separation between the sweeping randomized automata running in linear time and in unrestricted time. We have shown that there exists a family of languages that can be accepted by small sweeping LasVegas automata in exponential time, but it cannot be accepted by small automata in linear time, even when using Monte-Carlo randomization with two-sided bounded error. To prove this result, we have adapted the ideas of [Sip80b], which can be used to prove lower bounds on rotating and sweeping deterministic automata. After this adaptation, we have been able to use the technique of hardness propagation to obtain lower bounds on Monte-Carlo automata with bounded two-sided error running in linear time. This result also implies that the possibility of

arbitrarily long computations is essential for the power of randomized automata. Indeed, imposing a fixed upper bound on the length of the computation of a sweeping randomized automaton implies linear running time. Hence, it implies a decrease in power that can be traded only for exponential blowup in the size of the automaton.

In our work, we have used the concept of language families to introduce the complexity classes of finite automata. Our results, however, can be adapted to a more general concept of *promise problem families*, as introduced in [KKM08]. Here, instead of working with languages, automata were defined to accept *promise problems*. A promise problem P over an alphabet Σ consists of two disjoint sets $P_Y, P_N \subseteq \Sigma^*$. The set P_Y contains the *yes-instances* of P and the set P_N contains the *no-instances*. An automaton M solves P if it accepts all words in P_Y and does not accept any word in P_N; the behavior of M on $\Sigma^* - (P_Y \cup P_N)$ can be arbitrary. The name *promise problem* originates in the intuition that the automaton works under the promise that its input word is in language $P_Y \cup P_N$.

By using the concept of promise problems, it is possible to obtain slightly stronger results. Furthermore, some of the previous ad-hoc proofs for complexity classes separations (e. g. the separation between SD and SΔ presented in [KKM07]) can be reconstructed by the hardness propagation technique used in the context of promise problems.

On the other hand, the concept of promise problems is not widely used. Since our main interest is the separation of complexity classes induced by languages instead of promise problems, the gain from using promise problems is not essential. Hence, we have decided to use the more standard approach of working with languages.

4.1 Open Problems

We have presented a comprehensive characterization of all introduced complexity classes. Nevertheless, several problems remain open. At first, some closure properties of complexity classes induced by non-randomized finite automata are not known, as indicated by Figure 2.4.

Very few relationships between the complexity classes of two-way automata are known. For example, we know that SD $\subsetneq$ 2D, SΔ $\subsetneq$ 2Δ = $2\mathrm{P}_0^{\mathrm{U}} = 2\mathrm{P}_0^{\varepsilon}$, and SN $\subsetneq$ 2N = $2\mathrm{P}_1^{\mathrm{U}} = 2\mathrm{P}_1^{\varepsilon}$. Any separation between 2D, 2Δ, and 2N would imply that 2D $\neq$ 2N, what is a prominent open problem, more than 30 years old. Furthermore, proving 2D $\neq$ 2N using only polynomially short words as witnesses would imply that L $\neq$ NL.

Another open problem is the relationship between the complexity classes induced by rotating, sweeping, and two-way Monte-Carlo automata with two-sided error. More precisely, it is open whether any classes connected by a dotted arrow in Figure 3.1 collapse.

We have provided almost complete characterization of the complexity classes induced by one-way randomized automata; see Figure 3.2. However, it is still open if $1\mathrm{P}_0^{\mathrm{N}}$ and $1\mathrm{P}_2^{\varepsilon}$ are incomparable or if $1\mathrm{P}_0^{\mathrm{N}} \subsetneq 1\mathrm{P}_2^{\varepsilon}$. To attack this problem, one might try to extend the separation $1\mathrm{P}_0^{\mathrm{N}} \not\subseteq 1\mathrm{P}_1^{\varepsilon}$ to $1\mathrm{P}_0^{\mathrm{N}} \not\subseteq 1\mathrm{P}_2^{\varepsilon}$. To do so, it might be possible to

adapt the lower bound on $1\mathrm{P}_1^{\varepsilon}$ as provided by [JPS84, Theorem 3.1ii] to two-sided Monte-Carlo randomization.

When introducing the randomized automata, we have mentioned three possible ways how to use randomness: Allowing arbitrary real probabilities, restricting to rational probabilities, and restricting to coin flips only. All relationship between randomized complexity classes presented in this thesis hold for any of these models. The relationship between these models themselves is, however, an open problem.

The main result presented in Chapter 3 is the separation between rotating automata running in exponential expected time and in linear expected time, as well as the generalization of this result for sweeping automata. It would be very interesting to extend this separation to automata running in exponential time and in polynomial time, since polynomial time captures better the concept of efficient computability and is a closer analogy to the well known P vs. ZPP problem. This extension seems to be, however, highly nontrivial.

The proof of lower bounds on the state complexity of rotating and sweeping automata running in linear expected time, as presented in Chapter 3, relies heavily on the fact that the number of traversals of the bounded automaton is constant in the expected case. Unfortunately, it seems to be rather complicated to generalize this proof for superlinear expected time; most probably, significantly new ideas are essential for this approach to succeed.

As an another approach for extending the gap to polynomial time, one might try to analyze the properties of rotating and sweeping randomized automata running in superlinear but polynomial time. In an ideal case, it might be possible to prove that any such automaton can be simulated by a linear-time automaton with only polynomially more states. Proving such fact would immediately imply the exponential gap in the state complexity between automata running in polynomial and exponential time.

As a first attempt, one might try to prove that no randomized rotating or sweeping automaton can run in polynomial but superlinear expected time. This attempt, however, fails:

Theorem 4.1. *For any positive integer k, there exists a rotating (sweeping) randomized automaton M over a unary alphabet working in expected time $\Theta(n^k)$, where n is the length of the input word.*

Proof. We focus on the case of rotating automata; the case of sweeping automata is very similar. We define a rotating automaton M with $k+3$ states $q_0, \ldots, q_{k+2}$, where state q_0 is the start state. Since we are interested only in the running time of M and not in the accepted language, automaton M has no accept state.[1] Automaton M works over a unary alphabet $\Sigma = \{a\}$. To define the behavior of M when reading symbol a, we define its transition matrix $\delta(a)$, as introduced in Subsection 3.4.2. Transition matrix $\delta(a)$ is the $(k+4) \times (k+4)$ stochastic matrix defined in Table 4.1.

[1] To make the definition of M formally consistent with the definition of randomized automaton introduced in Chapter 3, it is sufficient to add a new accept state q_a that is not reachable from any of the $k+3$ states.

$$\delta(a) := \left(\begin{array}{ccccccc|cc|c} \frac{1}{2} & \frac{1}{2} & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & \dots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \dots & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Table 4.1: Definition of the transition matrix $\delta(a)$. Matrix $\delta(a)$ is a $(k+4) \times (k+4)$ stochastic block-diagonal matrix. It consists of $(k+1) \times (k+1)$, 2×2, and 1×1 blocks corresponding to states $q_0, \dots, q_k$, states q_{k+1}, q_{k+2}, and $\perp$.

$$\delta(\dashv) := \left(\begin{array}{ccc|cc|c} 0 & \dots & 0 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 \\ \hline 0 & \dots & 0 & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 & 0 \\ \hline 0 & \dots & 0 & 0 & 0 & 1 \end{array}\right)$$

Table 4.2: Definition of the transition matrix $\delta(\dashv)$.

Automaton M ignores the symbol $\vdash$, i. e., the transition matrix $\delta(\vdash)$ is the $(k+4) \times (k+4)$ identity matrix. On symbol $\dashv$, automaton M behaves deterministically: If $\dashv$ is reached in q_{k-1} or q_{k+2}, the automaton starts a new traversal in state q_{k+1}. Otherwise, the automaton halts. The corresponding transition matrix $\delta(\dashv)$ is shown in Table 4.2.

Let us consider the running time of M on input word a^n. Reading a^n induces a transformation $\delta^n(a)$ of the probability distributions over the states of M. It is straightforward to check that $\delta^n(a)$ satisfies the equality in Table 4.3.

Now let us calculate the expected running time of M on input a^n. Automaton M starts in state q_1. After the first traversal of the input, it ends in q_{k-1} with probability $\binom{n}{k-1}2^{-n}$. In this case, it starts new traversal in state q_{k+1}. In any other case, what happens with probability $1 - \binom{n}{k-1}2^{-n}$, automaton M halts. Hence, the contribution

$$\delta^n(a) = \left(\begin{array}{ccccccc|cc|c} 2^{-n} & n2^{-n} & \binom{n}{2}2^{-n} & \binom{n}{3}2^{-n} & \dots & \binom{n}{k-1}2^{-n} & * & 0 & 0 & 0 \\ 0 & 2^{-n} & n2^{-n} & \binom{n}{2}2^{-n} & \dots & \binom{n}{k-2}2^{-n} & * & 0 & 0 & 0 \\ 0 & 0 & 2^{-n} & n2^{-n} & \dots & \binom{n}{k-3}2^{-n} & * & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & n2^{-n} & * & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 2^{-n} & * & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \dots & 0 & 0 & 2^{-n} & 1-2^{-n} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Table 4.3: Matrix $\delta^n(a)$. Symbol $*$ denotes some value that we do not care about.

of this other case to the expected running time is

$$\left(1 - \binom{n}{k-1}2^{-n}\right) n. \tag{4.1}$$

For every traversal started in q_{k+1}, automaton M reaches $\dashv$ in state q_{k+1} with probability 2^{-n} and it reaches $\dashv$ in state q_{k+2} with probability $1-2^{-n}$. In the former case M halts, in the latter case M starts a new traversal from q_{k+1}. Hence, if M is started in q_{k+1}, it terminates after exactly i traversals with probability $p^{i-1}(1-p)$, where $p := 1-2^{-n}$. Thus, by a straightforward calculation we have that the expected running time of M if started in q_{k+1} equals to

$$\sum_{i=1}^{\infty} nip^{i-1}(1-p) = \frac{n}{1-p} = n2^n. \tag{4.2}$$

Equation (4.2) implies that the case when M does not halt after the first traversal has the following contribution to the expected running time:

$$\binom{n}{k-1}2^{-n}\left(n + n2^n\right) = \binom{n}{k-1}\left(n + o(n)\right) \tag{4.3}$$

Hence, putting (4.1) and (4.3) together yields the expected running time of M on input a^n:

$$\left(1 - \binom{n}{k-1}2^{-n}\right) n + \binom{n}{k-1}(n + o(n)) = n\left(1 - o(n) + \binom{n}{k-1}(1 + o(1))\right) = \\ = \Theta(n^k)$$

□

The preceding theorem proves that there indeed exist rotating (sweeping) automata with polynomial but superlinear expected running time. However, the automata witnessing this fact presented in Theorem 4.1 are somewhat special. These automata run for linear time with very high probability (more precisely, they make only one traversal of the input word with probability $1 - \Theta(n^k)2^{-n}$) and they run for exponential time with very low probability. Together, the overall running time is $\Theta(n^k)$, but any such automaton working with bounded error can be easily simulated by an equivalent automaton in linear expected time. Indeed, the probability of a superlinear computation is so low that it cannot affect the output of a bounded-error automaton. Hence, it still might be feasible to prove the exponential gap in the size between rotating (sweeping) automata running in polynomial and in exponential expected time by showing that such automata are not able to exploit superlinear but polynomial expected running time at all.

Acknowledgement

I would like to thank my advisor Juraj Hromkovič for introducing me into the topic of this thesis and for his support of my research. Next, I would like to thank Christos A. Kapoutsis and Tobias Mömke for their cooperation in the automata-related research and their help with clarifying the ideas presented in this thesis. I would like to thank Rastislav Královič and Georg Schnitger for their valuable consultations. I am also thankful to Hans-Joachim Böckenhauer and Dennis Komm for their help with the German translation of the abstract.

Bibliography

[Amb96] Andris Ambainis. The complexity of probabilistic versus deterministic finite automata. In *ISAAC '96: Proceedings of the 7th International Symposium on Algorithms and Computation*, pages 233–238, London, 1996. Springer-Verlag.

[Bab79] László Babai. Monte Carlo algorithms in graph isomorphism techniques. Technical Report 79-10, Département de mathématiques et statistique, Université de Montréal, 1979.

[Ber80] Piotr Berman. A note on sweeping automata. In J. W. de Bakker and Jan van Leeuwen, editors, *Proc. of the 7th International Colloquium on Automata, Languages and Programming (ICALP 1980)*, volume 85 of *Lecture Notes in Computer Science*, pages 91–97. Springer-Verlag, 1980.

[BFS86] László Babai, Péter Frankl, and János Simon. Complexity classes in communication complexity theory. In *SFCS '86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 337–347, Washington, D.C., 1986. IEEE Computer Society.

[Chr86] Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47(2):149–158, 1986.

[Con01] Anne Condon. *Bounded error probabilistic finite state automata*, volume 2 of *Handbook on Randomized Computing*, pages 509–532. Kluwer, 2001.

[ĎHI00] Pavol Ďuriš, Juraj Hromkovič, and Katsushi Inoue. A separation of determinism, las vegas and nondeterminism for picture recognition. In *COCO '00: Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, pages 214–228, Washington, D.C., 2000. IEEE Computer Society.

[ĎHJ+01] Pavol Ďuriš, Juraj Hromkovič, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multipartition communication complexity. In Afonso Ferreira and Horst Reichel, editors, *Proc. of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, volume 2010

of *Lecture Notes in Computer Science*, pages 206–217, London, 2001. Springer-Verlag.

[ĎHRS97] Pavol Ďuriš, Juraj Hromkovič, José D. P. Rolim, and Georg Schnitger. Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In Rüdiger Reischuk and Michel Morvan, editors, *Proc. of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1997)*, volume 1200 of *Lecture Notes in Computer Science*, pages 117–128, Berlin, 1997. Springer-Verlag.

[dLMSS56] Karel de Leeuw, Edward F. Moore, Claude E. Shannon, and N. Shapiro. Computability by probabilistic machines. In *in Automata Studies*, pages 183–212. Princeton Univ. Press, 1956.

[DS90] Cynthia Dwork and Larry J. Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal on Computing*, 19(6):1011–1123, 1990.

[Fre75] R. V. Freivald. Functions computable in the limit by probabilistic machines. In *Proceedings of the 3rd Symposium on Mathematical Foundations of Computer Science*, pages 77–87, London, 1975. Springer-Verlag.

[Fre81] Rūsiņš V. Freivalds. Probabilistic two-way machines. In *Mathematical Foundations of Computer Science 1981. Proc. of the 10th Symposium (MFCS 1981)*, volume 118 of *Lecture Notes in Computer Science*, pages 33–45, Berlin, 1981. Springer-Verlag.

[Fre08] Rūsiņš V. Freivalds. Artin's conjecture and size of finite probabilistic automata. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 280–291. Springer-Verlag, 2008.

[Gan59] F. R. Gantmacher. *The Theory of Matrices*, volume 2. Chelsea, New York, NY, USA, 1959.

[Gef07] Viliam Geffert. Magic numbers in the state hierarchy of finite automata. *Information and Computation*, 205(11):1652–1670, 2007.

[GG66] Seymour Ginsburg and Sheila Greibach. Deterministic context free languages. *Information and Control*, 9(6):620–648, 1966.

[Gil77] John Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GKK+02] J. Goldstine, M. Kappes, C.M. Kintala, H. Leung, A. Malcher, and D. Wotschke. Descriptional complexity of machines with limited resources. *Journal of Universal Computer Science*, 8(2):193–234, 2002.

[GMP03] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. *Theoretical Computer Science*, 295:189–203, 2003.

[GMP07] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.

[HH73] Juris Hartmanis and Harry B. III Hunt. The lba problem and its importance in the theory of computing. Technical report, DSpace at Cornell University (United States), 1973.

[HK03] M. Holzer and M. Kutrib. State complexity of basic operations on nondeterministic finite automata. In *Implementation and Application of Automata. Proc. of the 7th International Conference, CIAA 2002*, volume 2608 of *Lecture Notes in Computer Science*, pages 61–79, Berlin, 2003. Springer-Verlag.

[Hro05] Juraj Hromkovič. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, New York, 2005.

[HS96] Juraj Hromkovič and Georg Schnitger. Nondeterministic communication with a limited number of advice bits. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 551–560, New York, NY, USA, 1996. ACM Press.

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

[HS01b] Juraj Hromkovič and Georg Schnitger. On the power of Las Vegas II: two-way finite automata. *Theoretical Computer Science*, 262(1-2):1–24, 2001.

[HS03a] Mika Hirvensalo and Sebastian Seibert. Lower bounds for las vegas automata by information theory. *RAIRO Theoretical Informatics and Applications*, 37(1):39–49, 2003.

[HS03b] Juraj Hromkovič and Georg Schnitger. Nondeterminism versus determinism for two-way finite automata: Generalizations of sipser's separation. In *Automata, Languages and Programming, Proc. of the 30th International Colloquium, ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 439–451, Berlin, 2003. Springer-Verlag.

[Huf54] D. A. Huffman. The synthesis of sequential switching circuits. Technical Report 274, Research laboratory of electronics, Massachusetts institute of technology, 1954.

[Jir05] Galina Jirásková. State complexity of some operations on binary regular languages. *Theoretical Computer Science*, 330(2):287–298, 2005. Descriptional Complexity of Formal Systems.

[JJS04] Jozef Jirásek, Galina Jirásková, and Alexander Szabari. State complexity of concatenation and complementation of regular languages. In *Implementation and Application of Automata. Proc. of the 9th International Conference, CIAA 2004*, volume 3317 of *Lecture Notes in Computer Science*, pages 178–189, Berlin, 2004. Springer-Verlag.

[JPS84] Joseph JáJá, Viktor K. Prasanna, and Janos Simon. Information transfer under different sets of protocols. *SIAM Journal on Computing*, 13(4):840–849, 1984.

[Kap06] Christos A. Kapoutsis. *Algorithms and lower bounds in finite automata size complexity.* PhD thesis, Cambridge, MA, USA, 2006. Advisor-Michael Sipser.

[KKM07] Christos A. Kapoutsis, Richard Královič, and Tobias Mömke. An exponential gap between LasVegas and deterministic sweeping finite automata. In Juraj Hromkovič, Richard Královič, Mark Nunkesser, and Peter Widmayer, editors, *Proc. of the 4th International Symposium on Stochastic Algorithms: Foundations and Applications (SAGA 2007)*, volume 4665 of *Lecture Notes in Computer Science*, pages 130–141, Berlin, 2007. Springer-Verlag.

[KKM08] Christos A. Kapoutsis, Richard Královič, and Tobias Mömke. On the size complexity of rotating and sweeping automata. In *Proc. of the 12th Interational Conference on Developments in Language Theory (DLT 2008)*, pages 455–466, 2008.

[Kol72] A. N. Kolodin. Two-way nondeterministic automata. *Cybernetics and Systems Analysis*, 10(5):778–785, 1972.

[Krá09] Richard Královič. Infinite vs. finite space-bounded randomized computations. In *Proc. of the 24th Annual IEEE Conference on Computational Complexity (CCC 2009)*, pages 316–325, Washington, D.C., 2009. IEEE Computer Society.

[Leu01] Hing Leung. Tight lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 63(3):384–393, 2001.

[Mea55] G. M. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[MF71] A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proc. of the 12th Annual Symposium*

on Switching and Automata Theory (SWAT 1971), pages 188–191, Washington, DC, USA, 1971. IEEE Computer Society.

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

[Mon81] Burkhard Monien. On the LBA problem. In *Fundamentals of Computation Theory. Proc. of the 1981 International FCT-Conference*, volume 117 of *Lecture Notes in Computer Science*, pages 265–280, Berlin, 1981. Springer-Verlag.

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, (34):129–153, 1956.

[Moo71] F. R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computers*, 100(20):1211–1214, 1971.

[MP01] Carlo Mereghetti and Giovanni Pighizzini. Optimal simulations between unary automata. *SIAM Journal on Computing*, 30(6):1976–1992, 2001.

[MPP01] Carlo Mereghetti, Beatrice Palano, and Giovanni Pighizzini. Note on the succinctness of deterministic, nondeterministic, probabilistic and quantum finite automata. *RAIRO Theoretical Informatics and Applications*, 35(5):477–490, 2001.

[MS99] Ioan I. Macarie and Joel I. Seiferas. Amplification of slight probabilistic advantage at absolutely no cost in space. *Information Processing Letters*, 72(3-4):113–118, 1999.

[Paz71] Azaria Paz. *Introduction to probabilistic automata (Computer science and applied mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1971.

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

[RS59] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, (3), 1959.

[She59] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:199–201, 1959.

[Sip78] Michael Sipser. Halting space-bounded computations. In *SFCS: Proc. of the 19th Annual Symposium on Foundations of Computer Science*, pages 73–74, Washington, D.C., 1978. IEEE Computer Society.

[Sip80a] Michael Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10(3):335–338, 1980.

[Sip80b] Michael Sipser. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences*, 21(2):195–202, 1980.

[SS78] William J. Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 275–286, New York, 1978. ACM Press.

[Var89] M Y Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, (30):261–264, 1989.

[Wan92] Jie Wang. A note on two-way probabilistic automata. *Information Processing Letters*, 43(6):321–326, 1992.

[Wat99] John Watrous. On quantum and classical space-bounded processes with algebraic transition amplitudes. In *FOCS: Proc. of the 40th Annual Symposium on Foundations of Computer Science*, pages 341–351, Washington, D.C., 1999. IEEE Computer Society.