Infinite vs. Finite Size-Bounded Randomized Computations

Richard Královič

Department of Computer Science, ETH Zurich
Google Zurich

Abstract

Randomized computations can be very powerful with respect to space complexity, e.g., for logarithmic space, LasVegas is equivalent to nondeterminism. This power depends on the possibility of infinite computations, however, it is an open question if they are necessary. We answer this question for rotating finite automata (RFAs) and sweeping finite automata (SFAs). We show that LasVegas RFAs (SFAs) allowing infinite computations, although only with probability 0, can be exponentially smaller than LasVegas RFAs (SFAs) forbidding them. In particular, we show that even RFAs (SFAs) with linear expected running time may require exponentially more states than RFAs (SFAs) running in exponential time. We also strengthen this result, showing that the restriction on time cannot be traded for the more powerful bounded-error randomization. To prove our results, we introduce a technique for proving lower bounds on size of RFAs (SFAs) that generalizes the notion of generic strings discovered by M. Sipser.

1. Introduction

The comparative study of probabilistic complexity classes is an important part of theoretical computer science. Results of such kind are important to obtain a deeper understanding of the power of randomness. One of the most natural questions about randomization is to compare probabilistic complexity classes of Turing machines to deterministic and nondeterministic ones. Several prominent open problems, such as "Does ZPP (zero probability of error, polynomial time) equal P?", "Does BPP (bounded probability of error, polynomial time) equal NP?" belong here.

Known results about probabilistic complexity classes suggest that randomized computations can be very powerful with respect to space complexity. For example, zero probability of error (LasVegas) Turing machines working in logarithmic space are as powerful as nondeterministic ones [13, 18]. The core of this power is the possibility of infinite computations, even if their probability tends to 0. The main open problem posed in this context is whether restricting computations to finite ones restricts the power of randomized machines or not [16].

This work was supported by the Swiss National Science Foundation grants 200020-120073 and 200021_132510. A preliminary version of this paper was published as [11] and were part of [12].

1
In this paper, we analyze the simplest models of Las Vegas machines where the possibility of infinite computations makes sense — restricted versions of two-way finite automata. In this context, we prove an exponential gap in the size complexity between machines with and without infinite computations.

Various models of finite automata (FAs) are often used to analyze space-bounded computations. The space complexity of the analyzed computation directly translates into the size of the finite automaton, measured by its number of states. On one hand, the one-way FAs, which are the simplest models of FAs, are well understood. The exponential gap in the number of states between deterministic and nondeterministic one-way FAs was discovered long ago. Randomized FAs have been introduced in [14] and more recently studied in [6, 7]. In [6], it was proven that deterministic and Las Vegas one-way FAs are equivalent within a quadratic blowup in their number of states. Since one-way FAs do not possess the ability of infinite computations, this fact does not contradict our intuition that infinite computations are crucial for the efficiency of Las Vegas machines.

On the other hand, two-way FAs are a quite complex model — e.g., the relationship between determinism and nondeterminism in two-way FAs (raised in [15]) is a long-standing open problem. To fill the gap between the one-way and two-way FAs, the sweeping automata (SFAs) were introduced in [17]: a SFA is a restricted form of a two-way FA that can change the direction of the head motion only at the end-markers. It was proven in [17] that nondeterministic SFAs can be exponentially more succinct than deterministic SFAs; more recently, an exponential gap between nondeterministic and co-nondeterministic SFAs was proven in [8].

The model of sweeping automata is one of the simplest models of randomized machines that allows to use infinite computations. (More precisely, we should talk about computations of arbitrary length, since the probability of any infinite computation is 0, provided that the expected running time of the machine is finite.) In [9], it was proven that Las Vegas SFAs can be exponentially more succinct than deterministic SFAs. However, this result relies heavily on Las Vegas SFAs with exponential expected running time. It is not difficult to see that if computations of superlinear length are possible for some SFA, there is a non-zero probability that this SFA runs in a loop. Hence arbitrarily long computations are possible for any SFA with exponential expected running time. The natural question is if randomization can be of any help if we forbid arbitrarily long computations.

In this paper, we give a partial negative answer to this question. In particular, we show that Las Vegas SFAs running in exponential expected time (i.e., those with arbitrarily long computations allowed) can be exponentially more succinct than Las Vegas SFAs running in linear expected time. Since Las Vegas SFAs without arbitrarily long computations always work in linear time, our result confirms the conjecture that the possibility of infinite computations is essential to the efficiency of Las Vegas randomization, at least in the restricted case of sweeping automata.

The time complexity of randomized finite automata has been analyzed e.g., in [1]. There, it has been proven that exponential expected running time is necessary for two-way bounded-error finite automata to accept nonregular languages, i.e., avoiding the possibility of infinite computations reduces the power of the automata. To the best of our knowledge, no analogous results are known for Las Vegas randomization. Since Las Vegas finite automata always accept regular languages, we take size complexity into consideration. Our results show that avoiding infinite computations introduces exponen-
Infinite vs. Finite Size-Bounded Randomized Comp.

Apart from sweeping automata, we consider also their simplified version – the rotating automata (RFAs) [15, 10]. RFAs differ from SFAs in that they are not able to read the input from right to left. After reaching the right end-marker, their head is positioned again at the first symbol of the input. Our motivation to consider RFAs is twofold: On one hand, they allow us to present the core ideas in a simpler way and often it is rather straightforward to generalize arguments about RFAs to SFAs. On the other hand, we show that the power of randomized RFAs is related to one-way randomized automata, which are a very natural model. To have an analogous relation for SFAs, we need to introduce a rather artificial modification of one-way machines.

To obtain lower bounds on size complexity of randomized FAs with restricted running time, we use a technique that generalizes the ideas of [17] in such a way that makes it possible to prove such bounds by purely algebraic arguments. Hence, the contribution of the paper is also in introducing proof methods whose further development could help the original problem in the general setting.

Preliminary version of this paper was published as [11]. Results presented here are part of [12] as well.

Organization of the Paper

The paper is organized as follows: We introduce the models of finite automata, as well as other notation needed in our proofs, in Section 2. In Section 3, we show how it is possible to construct languages that are hard for one-way Monte-Carlo automata out of languages that are hard for one-way deterministic automata. In Section 4, we show how to use languages that are hard for one-way Monte-Carlo automata to construct languages that are hard for rotating bounded-error automata running in linear time. Later we extend this result to SFAs in Section 5. In Section 6, we complement our lower bounds on size complexity of RFAs and SFAs running in linear time by upper bounds for automata without restriction on running time. Finally, we apply the presented results to conclude an exponential gap in the size complexity between randomized RFAs (SFAs) running in linear time and those running in exponential time.

2. Preliminaries

We say that a matrix is substochastic if and only if all its values are nonnegative and every row has sum at most 1. A matrix is stochastic if and only if it is nonnegative and every row has sum equal to 1. A $k \times k$ matrix is a permutation matrix if and only if all its values are either 0 or 1 and in every row and every column there is exactly one value 1.

If $\Sigma$ is an alphabet, then $\Sigma^*$ is the set of all finite strings over $\Sigma$. If $z \in \Sigma^*$, then $|z|$ is its length, and $z^R$ is its reverse. A language over $\Sigma$ is any $L \subseteq \Sigma^*$; $\overline{L}$ is its complement. An automaton solves (accepts, or recognizes) a language if and only if it accepts exactly the strings of that language. We measure the size complexity of a finite automaton as the number of its states.

In this paper we deal with several models of finite automata. All of them can be categorized according to two independent properties: The capability of the head motion and the mode of the computation. Possible capabilities of the head motion include
Infinite vs. Finite Size-Bounded Randomized Comp.

(a)  
\[
q_s \rightarrow p_1 \quad q_1
\]
\[
(p_2) \quad p_2
\]
\[
(q_3) \quad q_3
\]
\[
(p_4) \quad p_4 \rightarrow q_a
\]

(b)  
\[
q_s \rightarrow p_1 \quad q_1
\]
\[
(p_2) \quad q_2\rightarrow q_1
\]
\[
(p_3) \quad p_3 \rightarrow q_2
\]
\[
(q_4) \quad q_3 \rightarrow q_a
\]

Figure 1: Computation of a (a) sweeping and (b) rotating automaton.

the well-known one-way and two-way variants of automata. We also consider sweeping automata, introduced in [17], and their simplified variants, rotating automata [15, 10].

The mode of the computation includes, e.g., determinism, nondeterminism, and various variants of randomization. In the sequel, we provide a more detailed definition of the used models of finite automata.

A sweeping deterministic finite automaton (sdfa) [17, 10] over an alphabet $\Sigma$ 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 $q_a \in Q$ (it is not difficult to see that having unique accept state is no restriction), and a transition function $\delta$ which partially maps $Q \times (\Sigma \cup \{\top, \bot\})$ to $Q$, for some end-markers $\top, \bot \notin \Sigma$. An input $z \in \Sigma^*$ is presented to $M$ surrounded by the end-markers, as $\top z \bot$. The computation starts at $q_s$ and on $\top$. The next state is always derived from $\delta$ 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 $\top$ or when the current symbol is $\bot$ and the next state is not $q_a$, in which cases the next position is the adjacent one towards the other end-marker. Note that the computation can either loop, or hang, or fall off $\bot$ into $q_a$. In this last case we call it accepting and say that $M$ accepts $z$.

According to the description above, the computation of a sdfa consists of several passes through the input string (see Figure 1a). We call every such pass as a traversal of the sdfa. A more formal definition of sdfas can be found in [10].

If $M$ is allowed more than one next move at each step, we say it is nondeterministic (a snfa). Formally, this means that $\delta$ partially maps $Q \times (\Sigma \cup \{\top, \bot\})$ to the set of all non-empty subsets of $Q$. Hence, on any $z \in \Sigma^*$, COMP$_M(z)$ is a set of computations. If at least one of them is accepting, we say that $M$ accepts $z$.

We say that $M$ is a rotating deterministic finite automaton (rdfa) if its next position is decided differently: it is always the adjacent one to the right, except when the current symbol is $\top$ and the next state is not $q_a$, in which case it is the one to the right of $\bot$.

The formal definition of the computation of a rdfa $M$ on a string $z$ is similar to the definition for a sdfa. An example of a computation of a rdfa is depicted in Figure 1b. Similar to the definition of a snfa, we define also the rotating nondeterministic finite automaton, which we denote as rnfa.

One-way finite automata can be viewed as a special case of rotating automata. In particular, we say that $M$ is a 1dfa if it halts immediately after reading $\bot$ the value of $\delta$ on any state $q$ and on $\top$ is 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 non-final. The state $\delta(q_s, \bot)$, if defined, is called initial. If $M$ is allowed more than one next move at each step, we say it is nondeterministic (a 1nfa).
Next, we define randomized variants of sweeping, rotating, and one-way finite automata. Essentially, a sweeping (rotating, one-way) randomized automaton (sometimes called also probabilistic automaton) \( M = (Q, \delta, q_a) \) is just a nondeterministic automaton that, in each step, instead of applying nondeterminism, picks one of the possible choices according to some probability distribution. More precisely, this means that \( \delta \) totally maps \( Q \times (\Sigma \cup \{\bot, \top\}) \) to the set of all probability distributions over \( Q \cup \{\bot\} \), i.e., all total functions from \( Q \cup \{\bot\} \) to the real numbers that obey the axioms of probability.

The symbol \( \bot \) means that the automaton hangs. Hence, on any input string \( z \in \Sigma^* \), the computation of \( M \) on \( z \) is a probability distribution over all possible computations.

Easily, this way of defining a randomized automaton applies to one-way, rotating as well as sweeping automata.

Allowing arbitrary real numbers in the probability distributions used by a randomized finite automaton is rather unintuitive, since real numbers can encode infinite information. For this reason, randomized automata are sometimes restricted to use rational numbers only [19] or only to numbers 0, 1/2, and 1 [4]. The latter restriction is also called as the model with fair coin flips; in this model, the automaton is not required to move the head in every computation step.

In this paper, however, we mainly focus on lower bounds on size complexity of finite automata. Hence, allowing arbitrary real probabilities is allowed unless stated otherwise. On the other hand, all upper bounds on size complexity in this paper hold for models with rational probabilities, too, and can be transformed to the model with coin flips in a straightforward way at the cost of small increase in size complexity.

Up to now, we have not defined which strings are accepted by a randomized automaton. There are several ways of doing that, yielding several different types of randomized automata. The most restrictive setting, called *Las Vegas* randomization, is to require zero probability of error. 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 \( \top \), the computation of \( M \) halts, and we say that the computation of \( M \) rejects the input string.

Besides accepting, a computation of \( M \) can also hang or run forever; in these cases, the computation neither accepts nor rejects the input. An input string \( z \in \Sigma^* \) is in the language solved by \( M \) if and only if \( M \) accepts \( z \) with probability at least \( \epsilon \) and rejects \( z \) with probability 0, for some fixed constant \( \epsilon > 0 \). Furthermore, we require that every \( z \in \Sigma^* \) that is not in the language solved by \( M \) is accepted by \( M \) with probability 0 and rejected with probability at least \( \epsilon \). The definition of Las Vegas automata applies to one-way (1pFA), rotating (RPFA), as well as sweeping automata (SPFA). The constant \( \epsilon \) is called the *error bound* of the automaton.

It is possible to obtain more powerful automata by relaxing the condition of the zero probability of error. A *Monte-Carlo* automaton \( M \) with one-sided bounded error is required to obey the following constraint: any \( z \in \Sigma^* \) is either accepted by \( M \) with probability at least \( \epsilon \) or accepted with probability 0. The automaton \( M \) does not need any special rejecting state. The language recognized by \( M \) consists of all strings accepted with non-zero probability. We denote the one-way, rotating, and sweeping Monte-Carlo automata as 1pFA, RPFA, and SPFA, respectively.

As a next step, it is possible to relax the constraint of one-sided error. Analogously to the bounded-error randomized Turing machines, we can define *bounded-error* finite...
automata (called also Monte-Carlo automata with two-sided bounded error). A bounded-error automaton $M$ obeys the following constraint: any $z \in \Sigma^*$ is either accepted by $M$ with probability at least $1/2 + \varepsilon$, or accepted with probability at most $1/2 - \varepsilon$. The language recognized by $M$ consists of all strings accepted with probability at least $1/2 + \varepsilon$. Although bounded-error automata are not a trivial superclass of Monte-Carlo automata, any Monte-Carlo automaton can be transformed into an equivalent bounded-error one with no increase in the number of states. We denote the one-way, rotating, and sweeping bounded-error automata as $1P^2FA$, $RP^2FA$, and $SP^2FA$, respectively. Bounded-error finite automata are very powerful. As implied by [2], even $RP^2FA$ can accept non-regular languages.

Above, we have allowed arbitrary error bound $\varepsilon$ in the definition of randomized automata. Sometimes a certain value of error bound is fixed in the definitions (usually $1/2$ for LasVegas and Monte-Carlo automata and $1/6$ for bounded-error automata). It is not difficult to see that the technique of amplification [5] can be used to transform a randomized automaton into an equivalent one with arbitrarily greater error bound. Such a transformation, however, increases the number of states of the automaton.

Unless stated otherwise, we assume that the error bound can be arbitrarily small. Hence, when stating that there is no randomized automaton solving some language with a given number of states, we mean that this is true for any error bound $\varepsilon > 0$; this gives stronger results in the context of lower bounds. Nevertheless, all upper bounds in this paper holds for high values of $\varepsilon$ as well.

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 $a$. We extend this notation for one-way automata in the following way. 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 string $w$.

When working with one-way randomized automata, we sometimes restrict our attention only to automata that do not hang while reading the input string. We say that a one-way randomized automaton $M = (q_s, \delta, q_a)$ is non-hanging if and only if $\delta(q, a)(\bot) = 0$ for all $q$ and all $a \neq \bot$ and $\delta(q, \bot)(q') = 0$ for all $q' \notin \{q_a, \bot\}$. The restriction to non-hanging automata has no significant impact on the size complexity:

**Observation 2.1.** Let $M$ be any one-way randomized automaton with $m$ states. Then $M$ can be transformed into an equivalent automaton that is non-hanging and has $m + 1$ states.

Similarly, it is sometimes useful to consider only (one-way, rotating, sweeping) randomized automata that satisfies the following property: The automaton never leaves the accept state $q_a$, i.e., $\delta(q_a, a)(q_a) = 1$ for all symbols $a \in \Sigma \cup \{\bot, \bot\}$. We call such an automaton as an automaton with special accept state. Such a requirement does not have a significant impact on the size complexity of the automaton:

**Observation 2.2.** Any $k$-state (one-way, rotating, sweeping) randomized automaton can be transformed into an equivalent $k + 1$-state automaton with special accept state.

To define a running time, consider a randomized finite automaton $M$. For a given input string $z$, 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
\[
T_M(n) := \max_{z \in \Sigma^n} \left( T_M(z) \right),
\]
i.e., \( T_M(n) \) is the maximal expected running time of \( M \) over all strings of length \( n \).

The main results of this paper are the proofs of lower bounds on size complexity of randomized automata with linear expected running time. We say that a randomized automaton \( M \) has linear expected running time if and only if \( T_M(n) = O(n) \). We denote automata with linear running time by prefix \( \text{lin-} \), i.e., we use the notation \( \text{lin-S} \), \( \text{lin-S} \), \( \text{lin-R} \), \( \text{lin-R} \), \( \text{lin-R} \), \( \text{lin-R} \), \( \text{lin-R} \), \( \text{lin-R} \), and \( \text{lin-R} \) to denote a \( \text{S} \), \( \text{S} \), \( \text{R} \), \( \text{R} \), \( \text{R} \), \( \text{R} \), \( \text{R} \), \( \text{R} \), and \( \text{R} \) running in linear expected time.

In the sequel, we show how to construct languages that are hard for more powerful models of finite automata out of languages that are hard for less powerful models. To do so, we often use the iteration operator \( \cdot \) and the concatenation operator \( \cdot \) defined as follows:
\[
L_1 \cdot L_2 := \{ x\#y \mid x, y \in \Sigma^*, x \in L_1 \land y \in L_2 \}
\]
\[
L^\#: \{ x_1\#\ldots\#x_i\# \mid i \geq 0, x_i \in \Sigma^*, (\forall i)(x_i \in L) \} = (L \cdot \{\#\})^\#
\]

We assume that \( \# \) is a new symbol not used in the alphabets of \( L_1, L_2, \) and \( L \). If we apply these operators several times, we assume that they always use new delimiter symbols.

We construct languages that are hard for \( \text{lin-R} \), \( \text{lin-S} \), \( \text{S} \) out of some core language by using the iteration, concatenation, complement, and reverse operators. One of the possible choices for such a core language that fits our requirements is the following one (used also in [10]). For any integer \( n \), we can define language \( J_n \) as follows:
\[
J_n := \{ \alpha i \mid \alpha \subseteq \{1, \ldots, n\} \text{ and } i \in \alpha \}
\]

We use \( J_n \) to construct languages that can be accepted by \( \text{R} \) (\( \text{R} \)) with number of states polynomial in \( n \) if their running time is not restricted, but any \( \text{lin-R} \) accepting them needs number of states exponential in \( n \). The basic properties of \( J_n \) are stated in the following lemma.

**Lemma 2.3.** Any 1DFA solving \( J_n \) needs at least \( 2^n \) states. Furthermore, \( J_n \) can be solved by a 1NFA with \( n + 2 \) states, and \( \overline{J_n} \) can be solved by a 1NFA with \( n + 2 \) states.

**Proof.** Let \( M \) be any 1DFA solving \( J_n \). If \( M \) is in the initial state and reads symbol \( \alpha \subseteq \{1, \ldots, n\} \), then the state reached by \( M \) must be different for every \( \alpha \). The second claim of the lemma can be proven by a straightforward construction.

**3. Lower Bounds on 1PFA\( S\)**

In this section, we show how it is possible to propagate hardness from one-way deterministic automata to one-way Monte-Carlo automata with one-sided error. In particular, we prove the following result:
Theorem 3.1. If no 1DFA with at most $m$ states can solve language $L$, then no 1PFA with at most $m - 1$ states can solve language $L^*$. 

Note that this result holds for any (arbitrarily small) error bound $\varepsilon > 0$ of the Monte-Carlo automaton, and it holds even if the automaton is allowed to use arbitrary real probabilities in its transition function.

The main idea behind our proof resembles the technique of confusing strings, used to prove lower bounds on intersection automata in [10]. We show that any 1PFA accepting $L^*$ with too small number of states can get confused with arbitrarily high probability, and we show that this contradicts to the fixed error bound of the automaton.

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

Definition 3.3. Consider a 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 string $w$ 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 $\downarrow w$ is equal to $p$. Equivalently, this can be written as

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

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 1PFA, we say that a state of 1PFA $M$ 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 1PFA $M$ accepting language $L$ is (non-)confused by an input string $w$ with probability $p$, if it is (non-)confused by $w$ with probability $p$ with respect to $L$.

Consider any 1PFA $M$ accepting language $L$. Any confused state $q$ of $M$ is non-final, 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 1PFA work with bounded error. Hence, to prove that a 1PFA $M$ cannot accept language $L$, it is sufficient to construct, for arbitrary given $\varepsilon$, a string from $L$ that confuses $M$ with probability at least $1 - \varepsilon$.

This fact can be formalized as follows:

Lemma 3.4. The following statements are equivalent:
1. There exists a non-hanging 1PFA $M$ with $m$ states that solves $L$.

2. There exists some $\varepsilon > 0$ and a non-hanging 1PFA $M$ with $m$ states that solves $L$ such that, for every string $w \in L$, automaton $M$ is non-confused by $w$ with probability at least $\varepsilon$.

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

**Proof.** $[1 \Rightarrow 2]$ Assume that $M = (q_s, Q, q_a)$ works with error bound $\varepsilon$. We claim that $M$ is non-confused by any string $w \in L$ with probability at least $\varepsilon$. 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 $\varepsilon$. Observe that any confused state $q'$ of $M$ is non-final, i.e., $\delta(q', \neg)(q_a) = 0$. Otherwise, there exists some string $u \notin L$ such that $\delta(q_s, \neg u)(q') > 0$, thus $\delta(q_s, \neg u)(q_a) > 0$, 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 $\neg w$, which is less than $\varepsilon$. But this is a contradiction with the error bound of $M$.

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

$[3 \Rightarrow 1]$ Consider a one-way randomized automaton $M = (q_s, Q, q_a)$ that is, for any string $w \in L$, non-confused by $w$ with respect to $L$ with probability at least $\varepsilon$. We show that it is easily possible to transform $M$ into a 1PFA solving $L$ with error bound $\varepsilon$. This transformation is achieved by making every state of $M$ that is non-confused with respect to $L$ and is reachable only by strings from $L$ a final state, and making every other state a non-final state. More precisely, we modify the definition of the transition function $\delta$ 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 strings from $L$, i.e.,

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

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

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

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:

$$\delta(q, \neg)(q_a) := 1 \quad \forall q \in Q_y$$
$$\delta(q, \neg)(q') := 0 \quad \forall q \in Q_y, \forall q' \neq q_a$$
$$\delta(q, \neg)(\bot) := 1 \quad \forall q \in Q_n \cup Q_c$$
$$\delta(q, \neg)(q') := 0 \quad \forall q \in Q_n \cup Q_c, \forall q' \neq \bot$$

The modified $M$ accepts all strings 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 $\neg w$, 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 $\varepsilon$ by the assumed lower bound on confusion probability of $M$. Hence, $M$ is a 1PFA with one-sided error bounded by $\varepsilon$. □
We prove Theorem 3.1 by contradiction: Assume that no small 1DFA can solve $L$, but some small 1PFA $M$ can solve $L^\#$. We show that $M$ can be confused with arbitrarily high probability, what directly contradicts to Lemma 3.4.\footnote{In fact, we use only the first two statements of Lemma 3.4 in this section, but we need the third statement later in Section 5.1.} At first, we show that the automaton $M$ can be confused from any starting state:

**Lemma 3.5.** Assume that no 1DFA with at most $m$ states can solve language $L$. Let $M = (q_0, \delta, q_a)$ be some non-hanging 1PFA with at most $m$ states that solves language $L^\#$. Consider any state $q$ of $M$. There exists a string $w_1 \in L$, a string $w_2 \notin L$, and some state $q_1$ of $M$ such that both $\delta(q, w_1\#)(q_1)$ and $\delta(q, w_2\#)(q_1)$ are non-zero.

**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 string $w_1\#$ for some string $w_1 \in L$, i.e.,

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

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

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

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 non-zero probability. More precisely, $M' = (q'_0, \delta', q'_a)$ is an automaton over the set of states $Q \cup \{q'_0, q'_a\}$, where $q'_0$ and $q'_a$ are some new states. The transition function $\delta'$ of $M'$ is defined as follows:

- $\delta'(q'_0, \dagger) = q$
- $\delta'(q', a) = q''$ \quad $\forall a \notin \{\dagger, \dagger\} \land \forall q' \in Q$, for arbitrary $q''$ such that $\delta(q', a)(q'') > 0$$
- $\delta'(q', \dagger) = q'_a$ \quad $\forall q' \text{ such that } \delta(q', \#)(q_1) > 0$ for some $q_1 \in Q_1$
- $\delta'(q', \dagger') = \bot$ \quad otherwise

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

Easily, if $M'$ accepts string $w$, 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 string $w \in L$. The computation of $M'$ hangs at some state $q'$ while reading $\dagger$ 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, $M'$ cannot reject $w$, because the third clause of $\delta'$ applies. \hfill $\square$

Now we show the core of the proof of Theorem 3.1, i.e., we show how it is possible to increase the probability of confusion arbitrarily.

**Lemma 3.6.** Assume that no 1DFA with at most $m$ states can solve language $L$. Let $M = (q_0, \delta, q_a)$ be some non-hanging 1PFA with at most $m$ states that solves language $L^\#$. There exists some constant $\alpha < 1$ such that, for any $u \in L^\#$ that non-confuses $M$ with probability $p$, there exists some $w \in L$ such that $uw\# \in L^\#$ non-confuses $M$ with probability at most $\alpha p$. 
Proof. Since every finite automaton has a finite number of states, Lemma 3.5 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 string \( z \in L^* \), i.e., \( \delta(q_s, z)(q) > 0 \), state \( q' \) is confused.

Assume that \( M \) is non-confused with probability \( p \) after reading string \( u \in L^* \). Let \( q \) be the most probable non-confused state of \( M \) after reading \( u \), i.e., \( q \) is the non-confused state such that \( p' := \delta(q_s, u\#)(q) \) is maximal. Since \( M \) has at most \( m \) states, it holds that \( p' = \delta(q_s, u\#)(q) \geq p/m \). 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 \( \tau \), and non-confused after reading \( \tau \). Obviously, we can upper bound \( p_1 \) by the probability that \( M \) is in a non-confused state other that \( q \) after reading \( \tau \). Hence \( p_1 \leq p - p' \).

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

- \( p_3 \) is the probability that \( M \) is confused after reading \( \tau \) and non-confused after reading \( \tau \). Easily, if \( M \) is in a confused state after reading \( \tau \), then any state reached after reading complete \( \tau \# \) 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 \( \alpha := 1 - \beta/m \). \( \square \)

The proof of Theorem 3.1 follows easily from Lemma 3.6:

Proof of Theorem 3.1. Assume by contradiction that Theorem 3.1 does not hold, i.e., no \( 1\text{DFA} \) with at most \( m \) states solves language \( L \), but there is some \( 1\text{PFA} \) with at most \( m - 1 \) states that solves \( L^* \). By Observation 2.1, in such case there exists a non-hanging \( 1\text{PFA} \) with at most \( m \) states that solves \( L^* \) as well. Then, by Lemma 3.4(2), there exist some \( \varepsilon \) and an \( m \)-state \( 1\text{PFA} \) \( M \) that solves \( L^* \) such that \( M \) cannot be non-confused with probability less than \( \varepsilon \). On the other hand, by applying Lemma 3.6 at most \( \log \varepsilon/\log \alpha + 1 \) times, we can construct some string \( w \in L^* \) that non-confuses \( M \) with probability less than \( \varepsilon \), what is a contradiction. \( \square \)

4. Lower Bounds on \( \text{RPFAs} \) with Linear Running Time

In this section, we show how to construct languages hard for rotating randomized automata running in linear time out of languages hard for one-way Monte-Carlo automata. In particular, we assume there is some language that cannot be solved by a small \( 1\text{PFA} \).
with arbitrarily small error bound, even if real transition probabilities are allowed, and, base on that, we show how to construct a language that cannot be solved by a small lin-RPFA, again with real transition probabilities and arbitrarily small error bound.

To prove this result, we use method that is similar to the technique of generic strings, which was introduced in [17] and used in [9, 10] to prove lower bounds on deterministic rotating and sweeping automata. The main idea behind generic strings is as follows.

Consider any rotating deterministic automaton $M$ over an alphabet $\Sigma$ and a set of states $Q$ and take some $w \in \Sigma^*$. When $M$ is started in some state $q$ and reads $w$ (i.e., it makes a single traversal through $w$), it ends in a state $q'$. In this way, $w$ induces a mapping from $Q$ to $Q$. We say that $w$ is generic, if the image of $Q$ under this mapping has minimal possible cardinality. A generic string has certain special properties that allow to prove lower bound on the size of the rotating deterministic automaton. Similar arguments can be used for sweeping automata as well.

We show how to adapt this technique for randomized automata. While the configuration of a deterministic automaton after reading some part of the input is completely described by its state, in case of a randomized automaton we need to consider a probability distribution over the set of states instead of a single state. Hence, we can say that every string $w$ induces a mapping that operates on the set of all probability distributions over the set of states. Therefore, it is not possible to define generic strings in the same way as for the deterministic case, as the image of a mapping induced by $w$ is usually not a finite set. To overcome this, we consider a probability distribution over a set of $k$ states as a point in $k$-dimensional metric space. Then, the image of all probability distributions under a mapping induced by $w$ is a set of points contained in a $d$-dimensional subspace. The rough idea is to say that $w$ is generic if $d$ is minimal possible (as we show later, this is not completely sufficient, but the difficulties could be overcome).

In the sequel, we formalize the above-described ideas and show how to exploit the properties of generic strings to prove lower bounds on lin-RPFA. At first, 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-RPFA, then the language $L$ can be accepted by a small 1PFA.

For technical reasons, we prove the lower bound for lin-RPFA with special accept states only. As shown in Observation 2.2, this does not have any significant impact on the size complexity of the considered automata.

**Transition Matrices**

Consider any rotating randomized automaton $M$ over an input alphabet $\Sigma$ 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 $p = (p_1, \ldots, p_{k-1}, p_k) \in \mathbb{R}^k$, 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, $p$ is transformed according to the transition function of $M$ into another probability distribution.

We can interpret the transition function $\delta$ of $M$ as an assignment of a $k \times k$ stochastic matrix to every $a \in \Sigma \cup \{\perp, \dashv\}$. In particular, $p$ is transformed into $p\delta(a)$ 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 \( x = x_1 x_2 \ldots x_n \in \Sigma^* \) with a \( k \times k \) stochastic matrix \( \delta_x = \delta(x_1) \delta(x_2) \ldots \delta(x_n) \), called the transition matrix of \( x \) (with respect to \( M \)), describing the behavior of \( M \) when reading \( x \): If \( M \) is in distribution \( p \in \mathbb{R}^k \) and reads \( x \), it reaches the distribution \( p \delta_x \). 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.

In particular a transition matrix \( \delta_x \) represents the mapping induced by \( x \) that operates on the set of all probability distributions over \( Q \).

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 4.1.** Let \( A, B, C, D \) be \( k \times k \) stochastic matrices such that

\[
\| C - A \|, \| D - B \| \leq \epsilon \leq 1.
\]

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

**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:

\[
\| (A + E_A)(B + E_B) - AB \| = \| AE_B + E_A B + E_A E_B \| \\
\leq \| AE_B \| + \| E_A B \| + \| E_A E_B \| \\
\leq \epsilon + k \epsilon + k \epsilon^2 \leq (2k + 1)\epsilon
\]

\( \square \)

**Closures of Languages**

Consider any \( \text{rpfa} \) \( M = (q_s, \delta, q_a) \). Let \( L \subseteq \Sigma^* \) be any language. In our arguments, we need that the set of all transition matrices of strings from \( L \), viewed as a set of points in the metric space of all \( k \times k \) matrices, forms a closed set. This, however, might not be the case. For this reason, we augment the set of all transition matrices of strings from \( L \) by all matrices that can be approximated arbitrarily well (in terms of the maximum norm) by a transition matrix of some string from \( L \). We denote the augmented set as \( C_L \):

\[
C_L := \{ A \in \mathbb{M} \mid (\forall \epsilon > 0)(\exists w \in L) \| A - \delta_w \| \leq \epsilon \}
\]

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

Now we present several properties of language closures that are not difficult to prove.

**Lemma 4.2.** The following properties of language closures hold for any languages \( L, L_1, L_2 \):

1. If \( A \in C_L \), then \( A \) is a stochastic matrix.
2. If \( L_1 \subseteq L_2 \), then \( C_{L_1} \subseteq C_{L_2} \).
3. \( C_L \) is a closed set: For any convergent sequence \( \{ A_i \}_{i=1}^{\infty} \) of matrices from \( C_L \), we have \( \lim_{i \to \infty} A_i \in C_L \).
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 \( A_1 A_2 \in \mathcal{C}_{L_1 \cup L_2} \).

6. Let \( L \subseteq \Sigma^* \) be a language over alphabet \( \Sigma \) 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 \( \mathcal{C}_{L \cup \{ \delta_x \mid x \in \overline{L} \}} \), or corresponds to \( x \# \) for some \( x \in \Sigma^* - L \). If every \( A_i \) is in \( \mathcal{C}_{L \cup \{ \delta_x \mid x \in \overline{L} \}} \), then \( \prod_{i=1}^m A_i \in \mathcal{C}_{L \cup \{ \delta_x \mid x \in \overline{L} \}} \). Otherwise, \( \prod_{i=1}^m A_i \notin \mathcal{C}_{\overline{L}} \).

**Proof.**

1. If \( A \) is a transition matrix of some string, 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 strings from \( L_1 \), it is trivially approximable by transition matrices of strings from \( L_2 \) as well.

3. Consider \( \lim_{n \to \infty} A_n \). This matrix is, by the definition of the limit, approximable arbitrarily well by some matrix \( A_i \). Let \( \epsilon > 0 \), there is an \( A_i \) such that \( \|A_i - A_i\| < \epsilon \). By definition of \( \mathcal{C}_L \), matrix \( A_i \) is approximable arbitrarily well by some transition matrix \( \delta_{w_i} \) for some \( w \in L \). I.e., \( \|A_i - \delta_{w_i}\| < \epsilon \). So, matrix \( A \) is approximable arbitrarily well by \( \delta_{w_i} \), since \( \|A - \delta_{w_i}\| < \|A - A_i\| + \|A_i - \delta_{w_i}\| < 2\epsilon \). 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 \( (0, 1) \). Hence, \( \mathcal{C}_L \) is compact due to the Bolzano-Weierstrass theorem.

5. The claim follows from Lemma 4.1. More precisely, let \( A_1 \in \mathcal{C}_{L_1} \) and \( A_2 \in \mathcal{C}_{L_2} \). We show that for every \( \epsilon > 0 \) there exists some \( w \in L_1 L_2 \) such that \( \|A_1 A_2 - \delta_{w_1}\| < \epsilon \).

By the definition of language closures, there exist \( w_1 \) and \( w_2 \) such that \( \|A_1 - \delta_{w_1}\|, \|A_2 - \delta_{w_2}\| \leq \epsilon/(2k + 1) \). Let \( w := w_1 w_2 \). Since \( \delta_w = \delta_{w_1} \delta_{w_2} \), the claim follows by applying Lemma 4.1 for \( A := \delta_{w_1}, B := \delta_{w_2}, C := A_1, D := A_2 \).

6. Let \( \Psi_i := L^i \) for each \( i \) such that \( A_1 \in \mathcal{C}_{L_1} \), and let \( \Psi_i := \overline{L^i} \) otherwise (note that by \( \overline{L} \) we mean the complement with respect to alphabet \( \Sigma \), i.e., the language \( \Sigma^* - L \)). Since \( A_1 \in \mathcal{C}_{\Psi_i} \), applying the statement 5 yields that \( \prod_{i=1}^m A_i \in \mathcal{C}_{\Psi_i \ldots \Psi_m} \). If every \( \Psi_i \) equals to \( L^i \), it holds that \( \Psi_1 \ldots \Psi_m \subseteq \overline{L^m} \), because the language \( L^m \) is closed under concatenation. Hence, the statement 2 implies that \( C \Psi_1 \ldots \Psi_m \subseteq \mathcal{C}_{\overline{L}} \), so we have \( \prod_{i=1}^m A_i \in \mathcal{C}_{\overline{L}} \).

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

Consider a lin-\textsc{rima} \( M \) with special accept state accepting language \( L \) with error bound \( \epsilon \). 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 \( \gamma \) depends on \( \epsilon \) 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 string, hence it must end the computation for \( x \) and \( y \) in arbitrarily close probability distributions.
Infinite vs. Finite Size-Bounded Randomized Comp.

Since \(M\) is a bounded-error automaton, it either accepts or rejects both \(x\) and \(y\), what is a contradiction.

**Lemma 4.3.** Let \(M\) be a lin-RP\(\mathcal{F}\)A with special accept state accepting language \(L\). Let \(\mathcal{C}_L\) (\(\mathcal{C}_T\)) be the closure of \(L\) (\(L\)) with respect to \(M\). There exists some \(\gamma > 0\) such that, for every \(A_1 \in \mathcal{C}_L\), \(A_2 \in \mathcal{C}_T\), it holds that \(\|A_1 - A_2\| \geq \gamma\).

**Proof.** Let \(M\) accepts \(L\) with error bound \(\varepsilon\). 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 \(A_2 \in \mathcal{C}_T\) 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 bound \(\varepsilon\), 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 string \(z\) 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 \(c\). Thus, we can write:

\[
\Pr[M \text{ accepts } x] = \Pr[M \text{ accepts } x \text{ in } \leq c/\varepsilon \text{ traversals}] + \Pr[M \text{ accepts } x \text{ in } \geq c/\varepsilon \text{ traversals}] \\
\leq \Pr[M \text{ accepts } x \text{ in } \leq c/\varepsilon \text{ traversals}]
\]

Since number of traversals of \(M\) is a non-negative random variable with expected value of at most \(c\), we can apply Markov bound to get that the probability of \(M\) making at least \(c/\varepsilon\) traversals is at most \(\varepsilon\). Combining this with the previous inequality, we get:

\[
\Pr[M \text{ accepts } x] \leq \Pr[M \text{ accepts } x \text{ in } \leq c/\varepsilon \text{ traversals}] + \varepsilon
\]

Hence, we have

\[
2\varepsilon \leq \Pr[M \text{ accepts } x] - \Pr[M \text{ accepts } y] \\
\leq \Pr[M \text{ accepts } x \text{ in } \leq c/\varepsilon \text{ traversals}] + \varepsilon - \Pr[M \text{ accepts } y] \\
\leq \Pr[M \text{ accepts } x \text{ in } \leq c/\varepsilon \text{ traversals}] + \varepsilon - \Pr[M \text{ accepts } y \text{ in } \leq c/\varepsilon \text{ traversals}],
\]

what is equivalent to

\[
\varepsilon \leq \Pr[M \text{ accepts } x \text{ in } \leq c/\varepsilon \text{ traversals}] - \Pr[M \text{ accepts } y \text{ in } \leq c/\varepsilon \text{ traversals}]. \tag{3}
\]

As \(M\) can accept only at the right end-marker, 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:

\[
\delta(t) (\delta_x \delta(-i))^t
\]

where \(\delta(a)\) is the transition matrix of \(M\) at symbol \(a\).
Since $M$ is an automaton with special accept state, 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 $q_x$ (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 4.1 to estimate the probabilities in (3):
\[
\varepsilon \leq \left| \Pr[M \text{ accepts } x \text{ in } < c/\varepsilon \text{ traversals}] - \Pr[M \text{ accepts } y \text{ in } < c/\varepsilon \text{ traversals}] \right| \leq (2k + 1)^{(2c/\varepsilon)} \gamma.
\]

As $\gamma$ can be chosen arbitrarily small, we have a contradiction. \hfill $\square$

**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 4.4.** Let $A$ be a $k \times k$ substochastic matrix. 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. \hfill $\square$
Lemma 4.5. Let $A$ be a $k \times k$ stochastic matrix. The matrix $A^\infty := \lim_{i \to \infty} A^{i!}$ exists. Furthermore, there exists a $k \times k$ permutation matrix $P$ and a $(k - \text{rank}(A^\infty)) \times \text{rank}(A^\infty)$ nonnegative matrix $R$ with row sums equal to 1 such that $A^\infty$ satisfies

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

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

Proof. We rely on the statements from [3, Chapter XIII]. From §7.5 and formula (80), it follows that $k!$ is a multiple of the period of $A$, hence the limit $\lim_{i \to \infty} A^{i!}$ exists. Thus, we can use the results from §7.2: there is a normal form of $A^{i!}$ equal to

$$A^{i!} = P_1 \begin{pmatrix} \begin{array}{cccc} Q_1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & Q_g & 0 \\ U_{g+1,1} & \cdots & U_{g+1,g} & Q_{g+1} \end{array} \end{pmatrix} P_1^{-1}$$

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 [3, (102)] to obtain:

$$A^\infty = P_1 B P_1^{-1}; \quad B = \begin{pmatrix} Q_1^\infty & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & Q_g^\infty & 0 \\ U_{g+1,1}^\prime & \cdots & U_{g+1,g}^\prime & 0 \end{pmatrix}$$

Furthermore, every $Q_i^\infty$ is a stochastic matrix with all rows identical. Since $A^\infty = A^\infty A^\infty$, it must hold that $U_{g+1,j}^\prime = U_{g+1,j}^\prime Q_j^\infty$.

Let $r_i$ be the row of $B$ that contains the first row of the submatrix $Q_i^\infty$. 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 $Q_j^\infty$, 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

$$U_{m,1}^\prime \cdots U_{m,g}^\prime \begin{pmatrix} 0 \end{pmatrix} = (U_{m,1}^\prime Q_1^\infty) \cdots (U_{m,g}^\prime Q_g^\infty) \begin{pmatrix} 0 \end{pmatrix},$$

we can define $c_{j,i}$ as the sum of all elements in the $n$-th row of $U_{m,i}^\prime$. 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

$$P_2 B = \begin{pmatrix} I & 0 \\ R & 0 \end{pmatrix} P_2 B$$
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 (4) and (5) together yields

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

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

Reduction to \( 1 \text{PFA}s \)

Now we are ready to proceed to the core of the proof of the hardness propagation. Let \( L \) be any language over alphabet \( \Sigma \). In the rest of this section, let us assume that a \((k-1)\)-state lin-RPFA \( M \) with special accept state accepts language \( L^* \). We assume that \( M \) works with bounded error, but do not pose any requirements on the error bound. Furthermore, \( M \) is allowed to use arbitrary real numbers as transition probabilities. We denote the transition function of \( M \) as \( \delta \) and the transition matrix of \( x \) with respect to \( M \) as \( \delta_x \). All language closures considered are with respect to \( M \). We show that there exists a \( 1 \text{PFA} \) \( M' \) with \( O(k^2) \) states that accepts language \( L^# \). Similarly as in the case of \( M \), automaton \( M' \) can use arbitrary real numbers as transition probabilities and it works with bounded error, although we do not estimate the value of the error bound.

At first, we define the generic matrix. The concept of generic matrix is analogous to generic strings as used in [17]. We, however, cannot talk about generic strings, because we need to consider all matrices in the language closure \( C_L \), i.e., all matrices that can be approximated arbitrarily well by transition matrices of strings from \( L \).

**Definition 4.6.** A \( k \times k \) matrix \( G \in C_{L^*} \) is called generic if and only if \( \text{rank}(G) \) is minimal among all members of \( C_{L^*} \) and it holds that

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

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

It is not difficult to prove that some generic matrix exists: since \( C_{L^*} \) is nonempty, it contains a matrix \( G' \) where \( G' \) has minimal rank. Consider the matrix

\[
G := \lim_{i \to \infty} G'^{ik^i}.
\]

The existence of the limit follows from the first claim of Lemma 4.5. Since \( L^* \) is closed under concatenation, \( G'^j \in C_{L^*} \) for any \( j \) due to Lemma 4.2(5, 2). Lemma 4.2(3) yields that \( G \in C_{L^*} \). Since \( \text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B)) \) for any matrices \( A, B \), and (what is not difficult to check) \( G = G'G'^{ik^i} \), it holds that \( \text{rank}(G) \leq \text{rank}(G') \). By the minimality of \( \text{rank}(G') \), we have \( \text{rank}(G) = \text{rank}(G') \). Combined with the second claim of Lemma 4.5, matrix \( G \) is generic.

For the rest of this section, we fix some generic matrix \( G \), as well as the corresponding matrices \( P, I, \) and \( R \).
Let $x$ be any string over the working alphabet of $M$. We consider the matrix $G\delta_x G$. Using the properties of generic matrices, we obtain the equality

$$G\delta_x G = P \begin{pmatrix} S_x & 0 \\ RS_x & 0 \end{pmatrix} P^{-1} G \quad (6)$$

where $S_x$ is some rank($G$) × rank($G$) stochastic matrix associated with the string $x$. Indeed, by the properties of generic matrices, we have that

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

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

$$\begin{pmatrix} I & 0 \\ R & 0 \end{pmatrix} \begin{pmatrix} A_x & B_x \\ C_x & D_x \end{pmatrix} \begin{pmatrix} I & 0 \\ R & 0 \end{pmatrix} = \begin{pmatrix} A_x + B_x R & 0 \\ RA_x + RB_x R & 0 \end{pmatrix} P^{-1} G = (7)$$

for $S_x := A_x + B_x R$. Since the matrix described in (7) 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 rank($G$) × rank($G$) matrix. Indeed, due to the second property of generic matrices required by Definition 4.6, it holds

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

Combined with the fact that $I$ and $R$ are matrices with rank($G$) = rank($P^{-1} G$) columns, it implies that the first 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

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

hence we have

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

what contradicts to the linear independence of the first 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 (6).

In the sequel, we show how the properties of a stamp of \( x \) differ for \( x \in L^s \) and for \( x \notin L^s \).

**Lemma 4.7.** 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 \( \varepsilon \), 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

\[
H := \lim_{i \to \infty} (G\delta_x)^{ik^l} G.
\]

Due to Lemma 4.5, \( H \) is well-defined, and due to Lemma 4.2(6, 3), \( H \in \mathcal{C}_{L^s} \). Using (6), we can write

\[
H = \lim_{i \to \infty} P \left( \begin{array}{cc} S_x & 0 \\ RS_x & 0 \end{array} \right)^{ik^l} P^{-1} G = P \left( \begin{array}{cc} S_x^{ik^l} & 0 \\ RS_x^{ik^l} & 0 \end{array} \right)^{ik^l} P^{-1} G =
\]

where \( S_x^{\infty} := \lim_{i \to \infty} S_x^{ik^l} \). Since \( S_x \) is a stochastic matrix, but not a permutation matrix, due to Lemma 4.4 we have \( \det(S_x) < 1 \), what implies that \( \det(S_x^{\infty}) = 0 \), hence \( \text{rank}(S_x) < \text{rank}(G) \). Since it holds that

\[
H = P \left( \begin{array}{cc} I & 0 \\ R & 0 \end{array} \right) \left( \begin{array}{cc} S_x^{\infty} & 0 \\ 0 & 0 \end{array} \right) P^{-1} G,
\]

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

In the sequel, we focus on the second statement of the lemma. Let \( \varepsilon := \gamma/(2k+1)^{k^l+2} \), where \( \gamma \) is the constant provided by Lemma 4.3. Assume that the statement does not hold for such \( \varepsilon \), hence there exists some \( x \notin L \) such that \( S_x \) is \( \varepsilon \)-close to some permutation \( P_1 \) matrix, i.e., \( \| S_x - P_1 \| < \varepsilon \). Consider the matrix

\[
X := (G\delta_x)^{ik^l} G.
\]

Due to Lemma 4.2(6), \( X \in \mathcal{C}_{TT} \). Using (6), we can express \( X \) as

\[
X = P \left( \begin{array}{cc} S_x^{k^l} & 0 \\ RS_x^{k^l} & 0 \end{array} \right) P^{-1} G.
\]

Applying Lemma 4.1, we obtain

\[
\| S_x^{k^l} - P_1^{k^l} \| < \varepsilon(2k+1)^{k^l-1}.
\]

Since \( P_1^{k^l} \) is an identity matrix, we have (again by Lemma 4.1):

\[
\gamma = \varepsilon(2k+1)^{k^l+2} > \left\| P \left( \begin{array}{cc} S_x^{k^l} & 0 \\ RS_x^{k^l} & 0 \end{array} \right) P^{-1} G - P \left( \begin{array}{cc} I & 0 \\ R & 0 \end{array} \right) P^{-1} G \right\| = \| X - G \|
\]

This, however, contradicts Lemma 4.3, because \( X \in \mathcal{C}_{TT} \) and \( G \in \mathcal{C}_{L^s} \).
In the sequel, we need to construct a small 1RFA that can detect whether the stamp of the input string is a permutation matrix or not. To do so, we employ a one-way randomized automaton that “calculates” the stamp of the input string.

Lemma 4.8. There exists a one-way randomized automaton $M_S$ with $k$ states over the alphabet $\Sigma$ such that its transition matrix associated with $\vdash x \dashv$ has the form

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

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 string such that $\delta_y$ is infinitely close to $G$. Afterwards, $M_S$ makes a suitable move on $\dashv$. We define its transition function as follows:

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

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

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

Hence it is sufficient to check that $S_x := \left( P^{-1}G\delta_x\# P \begin{pmatrix} I \\ R \end{pmatrix}\right)_{1\ldots\text{rank}(G)}$ satisfies (6); $A_{1\ldots m}$ denotes the matrix consisting of the first $m$ rows of matrix $A$. I.e., we need to show that

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

It holds that

$$\begin{pmatrix} S_x & 0 \\ RS_x & 0 \end{pmatrix} = \left( \begin{pmatrix} P^{-1}G\delta_x\# P \begin{pmatrix} I \\ R \end{pmatrix}\right)_{1\ldots\text{rank}(G)} & 0 \\ R \begin{pmatrix} P^{-1}G\delta_x\# P \begin{pmatrix} I \\ R \end{pmatrix}\right)_{1\ldots\text{rank}(G)} & 0 \end{pmatrix}.$$

\footnote{Automaton $M_S$ never hangs, so we may omit the row and the column corresponding to $\bot$ 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.}
picks the two rows containing these elements with probability $S$ column of $P$ matrix if we round all such elements to 1 and all other elements to 0, we obtain a permutation further, each such element has value greater than 1 element of value at least 1 $\varepsilon >$ every permutation matrix by some fixed $L$ states of $M$. In addition, we need the start state and the accept state. Hence, the total number of two reached states are different, $x$ is not a permutation matrix, hence (by Lemma 4.7) $x / M$ of $M$ simulate two independent runs of the stamp automaton $M$. When reaching $\vdash$, $M'$ simulates the first step of $M$, $M'$ makes independent random decisions in both simulated runs. When reaching $\vdash$, $M'$ simulates the last step of $M_S$. If both simulated runs of $M_S$ reach the same state, the stamp of the input string $x$ is not a permutation matrix, hence (by Lemma 4.7) $x / L$ 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 string $x \in L$ is rejected by $M'$: since $S_x$ is a permutation matrix by Lemma 4.7, no computation of $M'$ accepts $x$. On the other hand, if $x \notin L$, then, due to Lemma 4.7, $S_x$ 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 $1/\text{rank}(G)$, because $S_x$ is a $\text{rank}(G) \times \text{rank}(G)$ stochastic matrix. Since $\varepsilon$ can be w.l.o.g. small enough (more precisely, w.l.o.g. $\varepsilon \leq (\text{rank}(G) - 1)/\text{rank}(G)$), there is at least one element of value at least $\beta := \varepsilon/(\text{rank}(G) - 1)$ in each row of $S_x$. (Note that $\text{rank}(G) \geq 2$, since otherwise $S_x$ is a $1 \times 1$ stochastic matrix and hence it is a permutation matrix.) If no column of $S_x$ contains more than one element of value at least $\beta$, every row and every column of $S_x$ contains exactly one element of value at least $\beta$; furthermore, each such element has value greater than $1 - (\text{rank}(G) - 1)\beta = 1 - \varepsilon$. 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 4.7. Thus, there exists a column of $S_x$ that contains at least two elements of value at least $\beta$. The automaton $M'$ picks the two rows containing these elements with probability

$$\frac{1}{\text{rank}(G)} > \frac{1}{2} > \frac{1}{k^2}$$
Infinite vs. Finite Size-Bounded Randomized Comp.

and, if these rows are picked, the string is accepted with probability at least $\beta^2$. 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 4.9.** Let $M$ be a $(k-1)$-state lin-$\text{RP}_{\text{FA}}$ with special accept state that solves the language $L^\#$. Then there exists a lin-$\text{FA}$ $M'$ with $k^2 + 2$ states that solves the language $L$.

Note that we have no explicit guarantees about the error bound of the lin-$\text{FA}$ $M'$. This bound depends on the error bound of $M$ and on $k$.

Since any randomized automaton can be transformed into an equivalent automaton with special accept state by adding at most one new state, we obtain the following corollary. In this corollary, we need rather strong assumption that no small lin-$\text{FA}$ solves $L$, regardless of its error bound. We, however, get such a guarantee when using Theorem 3.1.

**Corollary 4.10.** If there is no lin-$\text{FA}$ $A$ with $k$ states that solves the language $L$, then there is no lin-$\text{RP}_{\text{FA}}$ with at most $\sqrt{k-2} - 2$ states solving the language $L^\#$.

5. Lower Bounds on $\text{SP}_{\text{FA}}$ with Linear Running Time

In the previous section, we have shown how to prove lower bounds on $\text{RP}_{\text{FA}}$s working in linear time. In the sequel, we show how to adapt these results for $\text{SP}_{\text{FA}}$ as well.

As we have shown in the previous section, the properties of $\text{RP}_{\text{FA}}$s are connected to $\text{RP}_{\text{FA}}$s. To obtain an analogous result for $\text{SP}_{\text{FA}}$, we need another model of automata instead of $\text{RP}_{\text{FA}}$. Intuitively, we need to somehow add a capability of reading the input in both directions to $\text{RP}_{\text{FA}}$. 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 string 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 string not in the solved language is accepted with probability 0, and any string in the language is accepted with probability at least $\varepsilon$ for some fixed constant $\varepsilon > 0$.

**Definition 5.1.** A direction-choosing automaton (dcrf 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

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

Note that an equivalent definition can be obtained by requiring

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

for any $x \not\in L$ and requiring

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

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 this section, we extend the hardness propagation results to sweeping automata. At first, we focus on the “lower level” of hardness propagation in Section 5.1. Here we shows how to propagate hardness from $1\text{P}_{\text{FAS}}$ to $d\text{C}_{\text{P}_{\text{FAS}}}$. Afterwards, we generalize the results of Section 4 in Section 5.2, where we show how to propagate hardness from $d\text{C}_{\text{P}_{\text{FAS}}}$ to lin-$\text{SP}_{\text{FAS}}$.

5.1. Propagating Hardness from $1\text{P}_{\text{FAS}}$ to $d\text{C}_{\text{P}_{\text{FAS}}}$

In this section, we focus on the “lower level” of the hardness propagation, i.e., the hardness propagation from $1\text{P}_{\text{FAS}}$ to $d\text{C}_{\text{P}_{\text{FAS}}}$.

We prove that if there is no small $1\text{P}_{\text{FAS}}$ accepting language $L$, then there is no small $d\text{C}_{\text{P}_{\text{FAS}}}$ accepting language $L \# L^n$. The result is formulated as the following lemma. Informally, the main idea of the proof is the following one: We use the concept of “confusion” as introduced in Section 3. If there is a small $d\text{C}_{\text{P}_{\text{FAS}}}$ $M$ for language $L \# L^n$, consider its left component. Since there is no small $1\text{P}_{\text{FAS}}$ solving $L$, this left component can be, due to Lemma 3.4, confused with respect to $L$ by arbitrarily high probability by some string $w_L \in L$. Similarly, the right component of $M$ can be confused by some $w_R \in L$. Afterwards, it is not difficult to check that the string $w_L \# w_R$ cannot be accepted by $M$ with a significant probability.

Lemma 5.2. If there is no $1\text{P}_{\text{FAS}}$ accepting language $L$ with at most $m$ states, then there is no $d\text{C}_{\text{P}_{\text{FAS}}}$ accepting language $L \# L^n$ such that both its left and its right component have at most $m - 1$ states.

Proof. Assume that the statement does not hold, i.e., there is no $1\text{P}_{\text{FAS}}$ accepting $L$ with at most $m$ states, but there exists some $d\text{C}_{\text{P}_{\text{FAS}}}$ accepting $L \# L^n$ such that both its components have at most $m - 1$ states. In that case, there exists also a $d\text{C}_{\text{P}_{\text{FAS}}}$ $M = (M^L, M^R)$ accepting $L \# L^n$ such that both $M^L$ and $M^R$ are non-hanging and have at most $m$ states. Assume that $M$ works with error bound $\varepsilon$. Hence, by the equivalence of statements 1 and 3 of Lemma 3.4 we have that for every non-hanging one-way randomized automaton $M'$ with at most $m$ states there is some string $w' \in L$ that non-confuses $M'$ with respect to $L$ with probability less than $\varepsilon$. Since both $M^L$ and $M^R$ are non-hanging one-way randomized automata, this gives us strings $w_L, w_R \in L$.

Consider string $w := w_L \# w_R$. Easily, $w \in L \# L^n$. 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 \( \uparrow w_L \). Easily, \( q \) is a non-confused state of \( M^L \) with respect to \( L \). Otherwise, the same computation can happen for some \( w' := w'_L \# w'_R \) with non-zero probability, where \( w'_L \not\in L \), i.e., \( w' \not\in L \# L^R \). This, however, means that \( M \) accepts some string not in \( L \# L^R \) with non-zero probability, what contradicts with the definition of DCPFAS.

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 \( \varepsilon \) by our choice of \( w_L \). Using a symmetric argument, the probability that \( M^R \) accepts \( w \) is less than \( \varepsilon \), too. This, however, contradicts to our assumption about error bound of \( M \). \( \square \)

5.2. Propagating Hardness from DCPFAS to lin-SdFAS

In the sequel, we adapt the results proven in Section 4 for sweeping automata. In particular, we prove the hardness propagation from DCPFAS to lin-SdFAS. All arguments presented in Section 4 for rotating automata can be generalized in a straightforward way. In this section, we describe the necessary changes to the proofs for rotating automata.

**Transition Matrix Pairs**

Similarly to the case of rotating automata, we can interpret the transition function \( \delta \) of \( M \) as an assignment of a \( k \times k \) stochastic matrix to every \( a \in \Sigma \cup \{\uparrow, \downarrow\} \). In the case of rotating automata, we associated every string \( x = x_1 x_2 \cdots x_n \in \Sigma^* \) with its transition matrix \( \delta_x \) 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 string from left to right, sweeping automata can read the string in both directions. For this reason, we associate the string \( x \) with two transition matrices when dealing with sweeping automata. The **left transition matrix** \( x \), denoted as \( \delta_x^L \), 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** \( x \), denoted as \( \delta_x^R \), in a symmetric way: \( \delta_x^R \) describes the behavior of \( M \) while reading \( x \) from right to left and it holds that \( \delta_x^R = \delta(x_n) \delta(x_{n-1}) \cdots \delta(x_1) \). Hence, every \( x \in \Sigma^* \) can be associated with a pair of stochastic matrices \( D_x = (\delta_x^L, \delta_x^R) \), called the **transition pair** \( x \).

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

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

For any \( x \) and \( y \), it holds that \( \delta_x^L y = \delta_x^L \delta_y^L \) and \( \delta_x^R y = \delta_y^R \delta_x^R \). Hence, we have that \( D_{xy} = D_x D_y \).

In Section 4, 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 4.1 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 SdFAS
$M = (q_s, \delta, q_a)$ and any language $L$ we can define the closure of $L$ with respect to $M$, denoted as $C_L$, as

$$C_L := \{ A \in M \mid (\forall \varepsilon > 0) (\exists w \in L) \| A - D_w \| \leq \varepsilon \},$$

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

It is not difficult to check that all claims of Lemma 4.2 hold for our new definition of language closures. For this reason, we reference this lemma also in connection with sweeping automata. Lemma 4.3 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 5.3.** Let $M$ be a lin-$\text{SPFA}$ with special accept state accepting language $L$ with bounded error. Let $C_L$ ($C_T$) be the closure of $L$ ($\mathcal{L}$) with respect to $M$. There exists some $\gamma > 0$ such that, for every $A_1 \in C_L$, $A_2 \in C_T$, it holds that $\| A_1 - A_2 \| \geq \gamma$.

**Proof.** We adapt the proof of Lemma 4.3. Let $M$ be a lin-$\text{SPFA}$ with special accept state accepting language $L$ with error bound $\varepsilon$. Assume that the statement of the lemma does not hold. Hence, for every $\gamma > 0$ there exist $x \in L$ and $y \in \mathcal{L}$ such that $\| D_x - D_y \| < \gamma$. Assume that the expected running time of $M$ on any input string $z$ is bounded by $c|z|$ for some constant $c$. Using the same arguments as in Lemma 4.3, we have that

$$\varepsilon \leq \Pr[M \text{ accepts } x \text{ in } < c/\varepsilon \text{ traversals}] - \Pr[M \text{ accepts } y \text{ in } < c/\varepsilon \text{ traversals}]. \quad (8)$$

Again, $M$ can accept only at the right end-marker. While every traversal of a rotating automaton ends at the right end-marker, only every odd traversal of a sweeping automaton does so. Hence, $M$ can accept input string $x$ 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:

$$\delta(-)\delta^L_x \delta(-)\delta^R_x \delta(-) (2l + 1) \delta(-)\delta^L_x \delta(-)$$

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-$\text{SPFA}$ $M$ 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 $\| D_x - D_y \| < \varepsilon$, we can do similar calculation as in the proof of Lemma 4.3 and estimate the sum (8) using Lemma 4.1. 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 4.1 $4l + 2 < 2c/\varepsilon + 2$ times:

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

As $\gamma$ can be chosen arbitrarily small, we have a contradiction. \qed
Definition 5.4. A matrix pair \((G_L, G_R) \in \mathcal{C}_{L^*}\) is called left (right) generic if and only if \(\text{rank}(G_L) (\text{rank}(G_R))\) is minimal among all left (right) components of members of \(\mathcal{C}_{L^*}\) and it holds that

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

\[
G_R = P_R \begin{pmatrix} I_R & 0 \\ R_R & 0 \end{pmatrix} P_R^{-1} G_R
\]

for the \(\text{rank}(G_L) \times \text{rank}(G_L)\) identity matrix \(I_L\) (\(\text{rank}(G_R) \times \text{rank}(G_R)\) 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 \(L = (L_L, L_R)\) such that \(L_L\) has minimal rank and a pair \(R = (R_L, R_R)\) such that \(R_R\) has minimal rank. Consider the pair

\[
G = (G_L, G_R) := \lim_{i \to \infty} (LR)^i k_i.
\]

The existence of the limit follows from the first claim of Lemma 4.5, since the lemma can be applied on both components of the matrix pairs. Since \(L^*\) is closed under concatenation, \((LR)^j \in \mathcal{C}_{L^*}\) for any \(j\) due to Lemma 4.2(5, 2). Lemma 4.2(3) yields that \(G \in \mathcal{C}_{L^*}\). Using the same arguments as in the rotating case (\(\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))\) for any matrices \(A, B\) and \(G = G(LR)^{k_i}\)), we have that \(\text{rank}(G_L) = \text{rank}(L_L)\) and \(\text{rank}(G_R) = \text{rank}(R_R)\). Again, combining this result with the second claim of Lemma 4.5, we have that \(G\) is generic.

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

Let \(x\) be any string 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 \(S^L_x\) of \(x\) and the right stamp \(S^R_x\) of \(x\) such that the following equalities hold:

\[
G_L \delta_{x \epsilon} G_L = P_L \begin{pmatrix} S^L_x & 0 \\ R_L S^L_x & 0 \end{pmatrix} P_L^{-1} G_L
\]

(9)
\[ G_R S_x^R G_R = P_R \begin{pmatrix} S_x^R & 0 \\ R_x S_x^R & 0 \end{pmatrix} P_R^{-1} G_R \tag{10} \]

The left stamp \( S_x^L \) (the right stamp \( S_x^R \)) is a rank(\( G_L \)) × rank(\( G_L \)) (rank(\( G_L \)) × rank(\( G_R \))) stochastic matrix associated with the string \( x \), 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 4.7.

**Lemma 5.5.** 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 \( x \not\in L \), \( S_x^L \) or \( S_x^R \) differs from any permutation matrix \( P \) by at least \( \varepsilon \) (i.e., \( \| S_x^L - P \| \geq \varepsilon \) or \( \| S_x^R - P \| \geq \varepsilon \)).

**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 \( H \) in an analogous way to the matrix \( H \) used in the rotating case:

\[ H := \lim_{i \to \infty} (GD_x^i)^{ik} G. \]

Using the same arguments as in Lemma 4.7, \( H \) is well-defined and \( H \in \mathcal{C}_{L*} \). Consider the left component \( H_L \) of \( H \). We follow the proof of Lemma 4.7, using \( H_L \) instead of \( H \). This yields that \( \det((S_x^L)^{\infty}) = 0 \) and that

\[ H_L = P_L \begin{pmatrix} I_L & 0 \\ R_L & 0 \end{pmatrix} \begin{pmatrix} (S_x^L)^{\infty} & 0 \\ 0 & 0 \end{pmatrix} P_L^{-1} G_L. \]

Hence, we have that rank(\( H_L \)) < rank(\( G_L \)), in contradiction with the genericity of \( G \).

Next, we focus on the second statement of the lemma. Again, the proof is analogous to the proof of Lemma 4.7. Let \( \varepsilon := \gamma / (2k + 1)^{k+2} \), where \( \gamma \) is the constant provided by Lemma 5.3. Assume that the statement does not hold for such \( \varepsilon \), hence there exists some \( x \not\in L \) such that both \( S_x^L \) and \( S_x^R \) are \( \varepsilon \)-close to some permutation matrix, i.e., \( \| S_x^L - P_1 \| < \varepsilon \) and \( \| S_x^R - P_2 \| < \varepsilon \). Consider the matrix pair

\[ X := (GD_x^i)^{ik} G. \]

By the same arguments as in the proof of Lemma 4.7, we have that \( X \in \mathcal{C}_{TT} \) and we can express the left component of \( X \) as

\[ X_L = P_L \begin{pmatrix} (S_x^L)^{k_l} & 0 \\ R_L (S_x^L)^{k_l} & 0 \end{pmatrix} P_L^{-1} G_L. \]

Following the proof of Lemma 4.7, we obtain

\[ \gamma = \varepsilon (2k + 1)^{k+2} \times P_L \begin{pmatrix} (S_x^L)^{k_l} & 0 \\ R_L (S_x^L)^{k_l} & 0 \end{pmatrix} P_L^{-1} G_L - P_L \begin{pmatrix} I_R & 0 \\ R_L & 0 \end{pmatrix} P_L^{-1} G_L \]

\[ = \| X_L - G_L \| \]

Since an analogous statement holds for the right component \( X_R \) of \( X \), we have that \( \| X - G \| < \gamma \), which contradicts Lemma 5.3, because \( X \in \mathcal{C}_{TT} \) and \( G \in \mathcal{C}_{L^*} \). \( \square \)
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 4.8:

**Lemma 5.6.** There exists a one-way randomized automaton $M_L$ with $k$ states over the alphabet $\Sigma$ such that its transition matrix associated with $\vdash x \dashv$ has the form

$$\delta^{M_L}(\vdash x \dashv) = \begin{pmatrix} S_L^x & 0 \\ * & * \end{pmatrix},$$

where $*$ denotes some matrix and $S_L^x$ 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 $\vdash x^R \dashv$ has the form

$$\delta^{M_R}(\vdash x^R \dashv) = \begin{pmatrix} S_R^x & 0 \\ * & * \end{pmatrix},$$

where $*$ denotes some matrix and $S_R^x$ is the right stamp of $x$.

The proof of Lemma 5.6 is analogous to the proof of Lemma 4.8.

The construction of the direction-choosing automaton $M'$ that recognizes $L$ is similar to the construction of a 1PFA 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 string $x \in L$. Since both the left stamp $S_L^x$ and the right stamp $S_R^x$ of $x$ are permutation matrices due to Lemma 5.5, 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 5.5 ensures that $S_L^x$ or $S_R^x$ is separated from any permutation matrix by at least $\epsilon$ for some fixed constant $\epsilon > 0$. 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 $\eta$ for some fixed constant $\eta > 0$ that depends on $k$ and $\epsilon$. Hence, we have shown that $M'$ is a valid DCRA with language $L$ in the sense of Definition 5.1. 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 an analogy of Theorem 4.9 for sweeping automata:

**Theorem 5.7.** Let $M$ be a $(k-1)$-state lin-S2FA with special accept state that solves the language $L^\oplus$. Then there exists a DCRA $M'$ such that both its left and its right component have at most $k^2 + 2$ states and $M'$ solves language $L$.

Similarly as in the rotated case, the error bound of $M'$ depends on the error bound of $M$ and on $k$. Furthermore, we can formulate a corollary similar to Corollary 4.10:

**Corollary 5.8.** If there is no DCRA $A$ such that both its components have at most $k$ states and that solves the language $L$, then there is no lin-S2FA with at most $\sqrt{k-2} - 2$ states solving the language $L^\oplus$.
6. Upper Bounds for Unrestricted Running Time

Rotating and sweeping automata can be very powerful if their running time is not restricted. From [2] it follows that $\text{SR}^n\text{FAs}$ and $\text{RP}^n\text{FAs}$ can accept even some non-regular languages in exponential time. This is true for any given error bound, even if the automata are restricted to coin flips only.

In previous sections, we have shown that it is possible to use the concatenation operator $L_1\#L_2$ and the iteration operator $L^*$ to construct languages that are hard for lin-$\text{RP}^n\text{FAs}$ and lin-$\text{SR}^n\text{FAs}$. In this section, we show that, under certain conditions, languages constructed by these operators are easy for $\text{RP}^n\text{FAs}$ and $\text{SR}^n\text{FAs}$. Hence, allowing exponential running time can lead to significantly smaller automata, even if LasVegas mode is required instead of two-sided bounded error.

Results presented in this section are quite straightforward, as they are based on simple constructions and previously known results. They are, however, an important complement to the lower bounds presented in previous sections. At first we observe that the concatenation and iteration operators do not add any significant hardness for one-way nondeterministic automata.

Observation 6.1. Let $L$ be a language that can be accepted by a $1\text{NFA}$ with $n$ states. Furthermore, let $L^c$ be accepted by a $1\text{NFA}$ with $n$ states as well. Then, both $L^\#$ and $L^\#^c$ can be accepted by a $1\text{NFA}$ with $n + O(1)$ states.

To construct an $1\text{NFA}$ for $L^\#$, it is sufficient to check every $\#$-delimited block of the input string independently. The constructed automaton then accepts if every block was successfully verified and the input ends with $. Similarly, to accept $L^\#$, the automaton must either find one $\#$-delimited block that is not in $L$, or verify that the input does not end with $\#$. Easily, both constructions can be implemented with $n + O(1)$ states.

A similar observation holds for the concatenation operator as well:

Observation 6.2. Consider languages $L_1$, $L_2$ such that both $L_1$ and $L_1^c$ can be accepted by a $1\text{NFA}$ with $n$ states and $L_2$ and $L_2^c$ can be accepted by a $1\text{NFA}$ with $m$ states. Then, both $L_1\#L_2$ and $L_1\#L_2^c$ can be accepted by a $1\text{NFA}$ with $n + m + O(1)$ states.

Since $L^\# = L^\#$ and a $1\text{NFA}$ for $L^\#$ can be easily obtained from a $1\text{NFA}$ for $L$ with no increase in size complexity, we know that:

Observation 6.3. Let $L$ be a language that can be accepted by a $1\text{NFA}$ with $n$ states. Furthermore, let $L^c$ be accepted by a $1\text{NFA}$ with $n$ states as well. Then, both $L^\#$ and $L^\#^c$ can be accepted by a $1\text{NFA}$ with $n$ states.

In the sequel, we show that if both language $L$ and its complement $L^c$ can be accepted by a small $1\text{NFA}$, then $L$ can be accepted by a small $\text{RP}^n\text{FA}$ (and $\text{SR}^n\text{FA}$) as well. Together with previous observations, this implies that, under certain conditions, languages obtained by the concatenation and iteration operators are easy for rotating (sweeping) randomized automata with unrestricted running time, even if they are restricted to LasVegas mode.

Since $1\text{NFAs}$ are a special case of $\text{SNFAs}$, this fact already follows from [7, Theorem 1]. This result, based on the method of [13], shows that if language $L$ can be accepted
by a SNFA with \( n \) states and \( L \) can be accepted by a SNFA with \( m \) states, then \( L \) can be accepted by a SNFA with \( 4(n + m) + 3 \) states. A similar result can be obtained for RPEFAs as well. Nevertheless, in our context, we are interested only in simulation of INFAs. Here, a simpler argument is sufficient, which even gives a better multiplicative constant in the reduction:

**Observation 6.4.** If language \( L \) can be accepted by a 1NFA with \( n \) states and language \( \overline{L} \) can be accepted by a 1NFA with \( m \) states, then \( L \) can be accepted by a RPEFA (SRPEFA) with \( n + m + O(1) \) states with error bound 1.

To prove this observation, it is sufficient to construct a RPEFA that simulates the 1INFAs for \( L \) and \( \overline{L} \) in an alternating way. At first, the 1NFA for \( L \) is simulated. If the 1NFA makes a nondeterministic choice out of several possibilities, the RPEFA chooses one of them uniformly at random. If the simulated 1NFA accepts, the input is accepted. If it hangs, the RPEFA returns to the beginning of the input and simulates the 1NFA for \( \overline{L} \). If it accepts the RPEFA rejects the input, otherwise, it simulates the 1NFA for \( L \) again, etc. Eventually, one of the simulated 1INFAs accept, hence, the constructed randomized automaton is a correct LasVegas machine.

It is easy to see that \( k := n + m + O(1) \) states are sufficient to construct the RPEFA and, obviously, an equivalent SRPEFA with \( k \) states can be constructed as well. The constructed machine uses only rational probabilities in its transition function and always gives a correct answer (i.e., the error bound is \( \varepsilon = 1 \)). Furthermore, it is not difficult to see that there is an exponential upper bound on the running time of the constructed machine.

If we restrict the machine to use fair coin flips as the only source of randomness, \( k^2 \) states are sufficient.

### 7. Conclusions

In this paper, we have presented a technique to construct languages that are easy for rotating (sweeping) randomized automata with unrestricted time, but hard for such automata if a restriction on their running time is added. In particular, we can show an exponential gap in the size complexity between automata with exponential and with linear running time. Moreover, the restriction on running time cannot be compensated by allowing the more powerful model with two-sided bounded error.

**Corollary 7.1.** Consider any language \( L_n \) such that any 1DFA needs at least \( 2^n \) states to accept it, but there exist 1INFAs solving \( L_n \) and \( \overline{L_n} \) with \( n + O(1) \) states. Note that, for any \( n \), there exists such a language, e.g., the language \( J_n \) (see Equation 2). Then it holds that:

1. Any lin-RPEFA needs at least \( 2^{2^\frac{n}{2}} - O(1) \) states to accept language \( L_n^R \).
2. There is a RPEFA with \( 2n + O(1) \) states that accepts language \( L_n^R \).
3. Any lin-SRPEFA needs at least \( 2^{2^\frac{n}{2}} - O(1) \) states to accept language \( L_n^R \) #(\( L_n^R \))^R.
4. There is a SRPEFA with \( 4n + O(1) \) states that accepts language \( L_n^R \) #(\( L_n^R \))^R.

**Proof.** 1. Theorem 3.1 yields that \( L_n^R \) cannot be accepted by a 1RPEFA with at most \( 2^n - 2 \) states. By Corollary 4.10, we obtain that there is no lin-RPEFA for \( L_n^R \) with at most \( \sqrt{2^n - 4} - 2 \geq 2^{2^\frac{n}{2}} - O(1) \) states.
2. Due to Observation 6.1, both $L_n^\#$ and its complement can be accepted by a 1NFA with $n + O(1)$ states. Applying this observation once more yields the same result for language $\overline{L_n^\#}$ and its complement as well. Then, Observation 6.4 proves the claim.

3. Applying Lemma 5.2 on $L_n^\#$ yields that there is no DCFA with both components having at most $2^n - 3$ states that accepts $L_n^\# (L_n^\#)^R$. Corollary 5.8 then yields that $\overline{L_n^\# (L_n^\#)^R}$ cannot be accepted by a lin-sDFA with at most $\sqrt{2^n - 5} - 2 \geq 2^{n/2} - O(1)$ states.

4. Applying Observation 6.3 on $L_n^\#$ yields that both $(L_n^\#)^R$ and its complement can be accepted by a 1NFA with $n + O(1)$ states. Using Observation 6.2, we get that both $L_n^\# (L_n^\#)^R$ and its complement can be accepted by a 1NFA with $2n + O(1)$ states. Observation 6.4 then proves the claim.

This corollary shows that even the very weak form of sweeping randomized automata, i.e., those restricted to LasVegas mode and using rational probabilities in its transition function, but with exponential running time, can be exponentially more succinct than the very powerful model of sweeping randomized automata, i.e., those working with two-sided bounded error and real probabilities, if they are restricted to linear running time. Similar result holds for rotating automata as well.

Furthermore, any rotating or sweeping randomized automaton that runs in superlinear time with non-zero probability can run arbitrarily long with non-zero probability. Indeed, if the expected running time is superlinear, this means that the expected number of traversals cannot be bounded by any constant. In such a case, the automaton must enter the same configuration (i.e., the same state at the same head position) twice with non-zero probability. Thus, there is a non-zero probability that the automaton runs in a loop, which means that there is also a non-zero probability of arbitrarily long computation. Thus, our result implies that if we enforce any deterministic upper bound on the length of the computation of randomized rotating and sweeping automata, the size complexity of the automata can increase exponentially.

A similar problem in context of Turing machines is a known open problem [16]. As mentioned in the introduction, it is not known if LasVegas Turing machines working in logarithmic space remain as powerful as nondeterministic ones even if a deterministic upper bound on running time is enforced. Hence, our result can be viewed as a solution of an analogous problem in the context of finite automata.

Open Problems

We have shown a technique for proving lower bounds on size complexity of randomized rotating and sweeping automata running in linear expected time. The focus on linear time is not very natural, however, these are the strongest bounds achievable with our technique. It remains open if it is possible to extend the presented results also for polynomial expected time.

Another interesting open problem is to decide if lin-sDFAS and sDFAS are equivalent within polynomial size blow-up, as is the case for 1DFAS and 1DFAS [6].

Last but not least, any result showing a superpolynomial gap between deterministic and nondeterministic two-way finite automata would be very valuable. Such result would close the long-standing open problem introduced in [15].
Acknowledgement
I would like to thank Juraj Hromkovič for initiating this research and for providing many helpful comments.

References