Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Randomized Multi-head Finite Automata

Bachelor Thesis

2023

Peter Grochal

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Randomized Multi-head Finite Automata

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	prof. RNDr. Rastislav Královič, PhD.

Bratislava, 2023
Peter Grochal

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Peter Grochal
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Randomized Multi-head Finite Automata
Pravdepodobnostné viachlavové konečné automaty

Anotácia: Práca sa zaoberá jednosmernými viachlavovými konečnými automatmi. Hlavným cieľom je študovať rôzne štandardné modely randomizácie aplikované na tento typ automatov, nájsť relevantné výsledky z literatúry, porovnať ich a podľa možnosti adaptovať výsledky z iných modelov.

Vedúci:	prof. RNDr. Rastislav Kráľovič, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	13.10.2022

Dátum schválenia: 13.10.2022

doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

....

študent vedúci práce

42561172

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Peter Grochal
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Randomized Multi-head Finite Automata

Annotation: Consider one-way multiple-head finite automata. The main goal of the thesis is to study various standard models of randomization (e.g. isolated cut-point, unbounded, one-sided, Las Vegas, etc.) applied to this type of automata. Find relevant results from the literature, and compare them. If possible, try to adapt results from other models.

Supervisor:	prof. RNDr. Rastislav Kráľovič, PhD.	
Department:	FMFI.KI - Department of Computer Science	
Head of department:	prof. RNDr. Martin Škoviera, PhD.	
Assigned:	13.10.2022	
Approved:	13.10.2022	doc. RNDr. Dana Pardubská, CSc. Guarantor of Study Programme

....

Student Supervisor

Acknowledgments: I would like to express my gratitude to my excellent supervisor prof. Rastislav Královič for his invaluable guidance, advice, and all the time he spent on our weekly meetings, which were a source of great inspiration for me. I would also like to thank my family for their unwavering support during my studies.

Abstrakt

V tejto práci sa zaoberáme skúmaním teórie formálnych jazykov so zameraním na randomizované automaty. Skúmame jednosmerné viachlavé pravdepodobnostné konečné automaty ($PFA(k)$) s rôznymi modelmi randomizácie, ktoré sa zvyčajne študujú.

Najprv formálne definujeme Monte-Carlo a LasVegas randomizácie, potom rôzne chyby, s ktorými takéto automaty môžu rozpoznávať jazyky. Definujeme a dokážeme aj normálny tvar, v ktorom $PFA(k)$ musí v každom kroku výpočtu presunúť aspoň jednu hlavu. Následne skúmame vlastnosti Monte-Carlo a LasVegas $PFA(k)$. Pre všetky tieto modely dokážeme hierarchiu, že s $(k + 1)$-hlavami majú väčšiu vyjadrovaciu silu ako s k. Tiež skúmame rôzne uzáverové vlastnosti tried rozpoznávaných týmito automatmi ako aj vzťahy medzi týmito triedami. Taktiež definujeme tzv. *barely-random* $PFA(k)$ a ukážeme, že tieto pravdepodobnostné automaty nemožno amplifikovať, t.j. ukážeme dolný odhad na chybu, s ktorou dokáže taký $PFA(k)$ rozpoznať konkrétny jazyk.

Kľúčové slová: jednosmerné viachlavové pravdepodobnostné konečné automaty, formálne jazyky, pravdepodobnostné automaty, LasVegas randomizácia, Monte-Carlo randomizácia, hierarchia, barely-random pravdepodobnostné automaty, amplifikácia.

Abstract

This thesis aims to explore the theory of formal languages, focused on randomized automata. In this thesis, we explore one-way multi-head probabilistic finite automata ($PFA(k)$) with the various models of randomization that are usually studied.

We first formally define both Monte-Carlo and LasVegas randomizations, then the various errors, with which such automata can recognize languages. We also define and prove a normal form in which the $PFA(k)$ has to move at least one head at every step of the computation. We then explore the properties of Monte-Carlo and LasVegas $PFA(k)$. For all these models, we prove a hierarchy, that with $(k+1)$-heads they have greater expressive power than with k. We also explore various closure properties of classes recognized by these automata, and the relations between these classes. We additionally define a Barely-random $PFA(k)$ and show that these probabilistic automata cannot be amplified, i.e., we prove a lower bound on the error with which a Barely-random $PFA(k)$ can recognize a certain language.

Keywords: formal languages, one-way multi-head probabilistic finite automata, randomized automata, LasVegas randomization, Monte-Carlo randomization, hierarchy, barely-random probabilistic automata, amplification.

Contents

Introduction

A considerable part of computer science is devoted to the analysis of the power of different computational models. Over the years, many different abstract models have been proposed and studied, many of them with different computational power. On the one end, we have the very powerful Turing machines, register machines, and on the other end lay the “simple” finite automata with power equivalent to regular expressions. These “simple” models, although having somewhat limited computational power, are considerably easier to understand and work with. It follows logically that they, or their modifications, have been studied in situations, where the register machine, and/or Turing machine equivalents were, and some still are, rather difficult to comprehend.

In the theory of formal languages, a field of computer science, computational models are often understood as machines that, given some input, (the input word) are supposed to decide whether or not this input instance is a solution to the problem (is a member of the recognized language). When considering computational models, we usually think about deterministic computations, yet numerous models have flirted with non-determinism, i.e. the power to have multiple valid paths to choose from, and to magically find, if it exists, the correct one, i.e., one that leads to a positive (accepting) verdict. However, such models are nontrivial to work with. Namely proving lower bounds, or the well-known question of whether $P = NP$ seems to be tough to crack.

Other variations to the traditional models have been studied, such as the model of probabilistic computation, which is the object of our interest. In recent years, the question of whether or not a solution to a problem can be found in “reasonable” time, is no longer a question of whether or not we have a polynomial-time (P) algorithm, but rather if we have a probabilistic polynomial-time (BPP) algorithm with bounded error that solves (accepts) our problem.

In our thesis, we will, as many before us did, explore a “less powerful” model so that we can gain insight more easily, in our case insight into the inner workings of probabilistic parallel computations. We will study multi-head variations of probabilistic one-way finite automata, first defined by M.O.Rabin in 1963 [Rab63]. He proved that a one-way (one-head) probabilistic automata with an unbounded error, can accept languages that are not regular. Moreover, he also proved that one-way (one-head) probabilistic automata with bounded error, only accept languages that are regular.

Therefore, we only need to explore the cases where the number of heads, k is greater than 1. Since we study multi-head probabilistic finite automata, we build on the numerous results for multi-head finite automata (many listed in [HKM09]), and one especially notable result is the one aptly called "k+1 heads are better than k" by Yao and Rivest [YR78].

The world of two-way probabilistic automata is vastly different. For example, the following result, proven by Freivalds, is that even with one head, a two-way probabilistic automaton can, with bounded error, accept $\{0^n1^n \mid n > 0\}$ [Fre81], a language known to be not regular (even though two-way finite automata have been proven to accept only regular languages). Moreover, the multi-head versions of two-way probabilistic automata have already been widely studied (e.g. [Mac95]), and have been proven to have expressive power equivalent to the power of log-space-constrained Turing machines.

We study the one-way version of the already studied multi-head probabilistic automata because, firstly, we believe that the constraint (one-way heads), will enable us to see the power of randomization more clearly, and secondly, to the best of our knowledge, we are not aware of any paper or article that would explore this model one-way multi-head probabilistic automata.

Our contribution

We first formally define both LasVegas and Monte-Carlo randomizations, the various types of errors with which such automaton can recognize a language, and the one-way k-head probabilistic finite automaton ($PFA(k)$). We prove that for $PFA(k)$, there is an ε-free normal form in which the automaton, at each step, has to advance at least one head. Then, we explore the properties of Monte-Carlo $PFA(k)$, where we prove a Hierarchy theorem analogous to the one for one-way multi-head finite automata by Yao and Rivest [YR78]. We also prove various relations between classes of languages recognized by Monte-Carlo $PFA(k)$ with one-sided errors, and then their closure properties (union, intersection, complement, and intersection with a regular language). We then explore the properties of LasVegas $PFA(k)$ where we prove a Hierarchy theorem, relations between classes of languages recognized by LasVegas and Monte-Carlo $PFA(k)$, and the closure properties of these classes. We additionally define a barely-random $PFA(k)$, and its normal form. We show that for this special case of $PFA(k)$, we can calculate a lower bound on the error with which this model can recognize a certain language.

Chapter 1

Definitions

In this chapter, we explore and provide various definitions of probabilistic automata. Moreover, we formally define various formalisms and concepts used in this theory that we have not seen defined properly. We also provide an example of a language accepted by one of these models, then define and prove a normal form for multi-head probabilistic automata.

1.1 Finite Automata

We begin with a brief resume of the basic definitions. The symbol Σ will denote a finite nonempty set (of symbols), to be referred to as the alphabet. Letters a, b (with subscripts) will usually denote the elements of Σ. The set of all finite sequences of elements of Σ will be denoted by Σ^*. The elements of Σ^* will be called *words* (over alphabet Σ). In some literature (e.g., [RS59]) words are referred to as *tapes*. The letters w, x, y, z, u, v (with subscripts) will always denote words. The empty word (i.e., the sequence of length zero) will be denoted by ε. Subsets of Σ^* (i.e., sets of words) will usually be called languages (sometimes *sets of tapes*). If $x = a_1 \dots a_k$ is a word then the length $l(x)$ of x is $l(x) = k$. If x and y are words then xy will denote the concatenation of x and y. The operation of concatenation will be denoted $\cdot$ ($x \cdot y \Leftrightarrow xy$). //Note that Σ^* with this operation is a free semi-group with the elements of Σ as free generators.//

Definition 1.1.1. A *(non-deterministic) finite automaton* (*NFA*) *over* Σ is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set (the set of *states*), δ is a function from $Q \times \Sigma$ into 2^Q (the *transition function*)[1], q_0 is an element of Q (the *initial state*), and F is a subset of Q (the set of *final states*).

Definition 1.1.2. A *deterministic finite automaton* (*DFA*) *over* Σ is a *finite automaton* $A = (Q, \Sigma, \delta, q_0, F)$, such that $|\delta(q, a)| \leq 1$ for all $q \in Q, a \in \Sigma$.

[1] 2^Q denotes the set of all subsets of Q – a powerset of Q.

Remark. All following definitions for *NFA* work for *DFA* also, since *DFA* are just a special case of *NFA*.

Definition 1.1.3. A *configuration* of a finite automaton is an element $(q, w) \in Q \times \Sigma^*$.

Definition 1.1.4. A *step of computation* of a finite automaton A ($\vdash_A$) is a relation on configurations, defined as: $(q_1, aw) \ \vdash_A \ (q_2, w) \iff \delta(q_1, a) \ni q_2$. It represents the reading of the next symbol in the input word and the change of the state.

Definition 1.1.5. The language $L(A)$ accepted by a finite automaton A is defined as a set of words $L(A) = \{w \in \Sigma^* \mid (q_0, w) \vdash_A^* (q_F, \varepsilon),\ q_F \in F\}$, where $\vdash_A^*$ is the transitive closure of relation $\vdash_A$.

Remark. The condition for w to be accepted can be interpreted as the following: "If there exists an accepting computation on w."

An alternate approach for defining *DFA* (e.g., in Rabin and Scott [RS59]), defines the language $L(A)$ via extending the δ function to a function $\hat{\delta}$ from $Q \times \Sigma^*$ into Q by $\hat{\delta}(q, \varepsilon) = q,\ \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ for any $q \in Q, w \in \Sigma^*, a \in \Sigma$. Then the output of $\hat{\delta}(q, w)$ is the state in which the deterministic finite automaton finishes the computation on input word w. Therefore we can define $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q, w) \in F\}$.

1.2 Probabilistic Automata

The following definitions have been heavily inspired by [Rab63], however, we extended them to encompass more models of probabilistic computations (namely Monte-Carlo and LasVegas).

Definition 1.2.1. A (*one-way*) *probabilistic finite automaton* (*PFA*) *over* Σ is a 6-tuple *PFA* $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ where:

- $Q = \{q_0, \ldots, q_m\}$ is a finite set (the set of *states*),
- δ is a function (the *transition function*) from $Q \times \Sigma$ into $[0, 1]^{m+1}$ [2] such that for $(q, a) \in Q \times \Sigma$, the following holds:

$$\delta(q, a) = (p_0(q, a), \ldots, p_m(q, a)) \ \text{ where } 0 \leq p_i(q, a) \text{ and } \sum_{i=0}^{m} p_i(q, a) = 1$$

- q_0 is element of Q (the *initial state*),
- Q_{acc} is a subset of Q (the set of *accepting states*),
- Q_{rej} is a subset of Q (the set of *rejecting states*), such that $Q_{acc} \cap Q_{rej} = \emptyset$.

Notation $p_j(q_i, a)$ represents the probability of changing state from q_i to q_j if reading a. These probabilities are assumed to be fixed and independent of time or previous inputs.

[2] $[0, 1]$ denotes the closed interval of real numbers from 0 to 1.

Definition 1.2.2. A *Monte-Carlo PFA over* Σ is a probabilistic finite automaton A over Σ, such that $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ where $Q_{rej} = Q - Q_{acc}$.

The *PFA* also has definite, fixed probabilities for going from state q_i to q_j when reading a sequence of symbols $a_1a_2 \dots a_n = x \in \Sigma^*$. These probabilities $(p_j(q_i, x))$ are calculated by computing the product of certain stochastic matrices we define.

Definition 1.2.3. ([Rab63]) Let A be a *PFA* with states $q_0, \dots, q_m$ and $p_j(q_i, a)$ the probability as defined in Definition 1.2.1. For $a \in \Sigma$ and $x{=}a_1a_2 \dots a_n$ define the $m+1$ by $m+1$ matrices $\mathbb{A}(a)$ and $\mathbb{A}(x)$ by:

$\mathbb{A}(a) = [p_j(q_i, a)]_{0\le i\le m,\ 0\le j\le m}$

$\mathbb{A}(x) = \mathbb{A}(a_1)\mathbb{A}(a_2)\dots\mathbb{A}(a_n)$

Remark. An easy calculation (involving induction on n) shows that the $(i+1, j+1)$ element of $\mathbb{A}(x)$ is the probability of A for moving from state q_i to state q_j by reading the input sequence x. Hence, the following corollary.

Corollary 1.2.4. *Let A be a PFA with states $q_0, \dots, q_m$ and $p_j(q_i, a)$ the probability as in Definition 1.2.1. For $a_1a_2 \dots a_n{=}x \in \Sigma^n$, the following holds:*

$$\mathbb{A}(x) = [p_j(q_i, x)]_{0\le i\le m,\ 0\le j\le m}$$

Definition 1.2.5. For *PFA* $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$, $Q_{acc} = \{q_{i_1}, q_{i_2}, \dots, q_{i_l}\}$, and $I = \{i_1, i_2, \dots, i_l\}$, define:

$$p_A(x) = \sum_{i\in I} p_i(q_0, x)$$

$p_A(x)$ is the probability of A starting in initial state q_0 and finishing the computation on the input word x in some accepting state (thus accepting).

Since our main interest lies in studying multi-head probabilistic automata, we see that this definition is not really scalable, since, in the multi-head case, the heads can, in different computations, move differently. Hence the automaton would read different input symbol sequences. Therefore the use of stochastic matrices for the multi-head version of *PFA* is complicated. Thus when we later define multi-head *PFA*, we need to use a different method.

Language accepted by PFA

Definition 1.2.6 ([Rab63]). Let A be *PFA*, $Q_{rej} = Q - Q_{acc}$, λ a real number, such that $0 \le \lambda < 1$, and $p_A(x)$ the probability of A accepting the word x. The language $L(A, \lambda)$ is defined as follows:

$$L(A, \lambda) = \{w \in \Sigma^* \mid \lambda < p_A(w)\}$$

If $x \in L(A, \lambda)$, we say that x is *accepted by A with cut-point λ*. The set $L(A, \lambda)$ is called the *language accepted by A with cut-point λ*.

Remark. Deterministic finite automata can be considered as a special case of *PFA*. In particular, in the Definition 1.1.2, the transition $\delta(q_1, a) \ni q_2$ can be viewed as A moving to the state q_2 with probability 1. Thus if rewriting the deterministic automaton as a *PFA* A', the stochastic vectors $\delta'(q_1, a) = (p_0, \ldots, p_n)$ will have exactly one coordinate 1 and all the others 0. It is readily seen that in this case $p_{A'}(x) = 1$, for $x \in \Sigma^*$, if and only if $x \in L(A)$. Hence for any $0 \leq \lambda < 1$, we have $L(A) = L(A', \lambda)$. Thus every set definable by a deterministic automaton is trivially definable by some *PFA*.

Definition 1.2.7 ([Rab63])**.** A cut-point $0 < \lambda < 1$ is called *isolated* with respect to A if there exists a number $\Delta > 0$ such that

$$\Delta \leq |p_A(x) - \lambda| \quad \text{for all} \quad x \in \Sigma^*$$

We refer to the real number Δ as the *error bound* of the isolated cut-point λ.

Definition 1.2.8. Let A be a *PFA* and L the language recognized by A. We say that the (output of a) computation on w is *correct* if the automaton ends in an accepting state when $w \in L$ or the automaton ends in a rejecting state when $w \notin L$. The computation is *incorrect* if the automaton ends in a rejecting state when $w \in L$ or the automaton ends in an accepting state when $w \notin L$.

Remark. Thus, we say that a *Monte-Carlo PFA* is a probabilistic automaton/algorithm that can, with non-zero probability, produce an incorrect output, yet always "outputs".

1.3 Types of errors

We introduce various restrictions on the sort of "error" a Monte-Carlo automaton can make, i.e., on the probabilities of returning correct and/or incorrect answer. We do that by specifying how both L and L^c look like. Essentially, what these definitions do, is that they restrict the construction of the automaton since there must exist no word that would be accepted with a probability that would not put it in either L or L^c.

Definition 1.3.1. Language L is accepted by a *PFA A* with *unbounded (two-sided) error*, if there exists a cut-point λ, $0 \leq \lambda < 1$, such that:

$$\begin{aligned} L = L(A, \lambda) = \{x \mid x \in \Sigma^*,\ \lambda < p_A(x)\} \\ (L^c = L(A, \lambda)^c = \{x \mid x \in \Sigma^*,\ \lambda \geq p_A(x)\}) \end{aligned} \tag{1.1}$$

(L is accepted by A with cut-point λ)

Remark. Rabin [Rab63] defined the language accepted by a *PFA* as the above defined language accepted with two-sided error.

Definition 1.3.2. Language L is accepted by a *PFA A* with *bounded (two-sided) error*, if there exists an isolated cut-point λ, $0 \leq \lambda < 1$, with *error bound* $\Delta > 0$, such that:

$$\begin{aligned} L = L(A,\lambda) &= \{x \mid x \in \Sigma^*,\ \lambda + \Delta \leq p_A(x)\} \\ L^c = L(A,\lambda)^c &= \{x \mid x \in \Sigma^*,\ p_A(x) \leq \lambda - \Delta\} \end{aligned} \tag{1.2}$$

(L is accepted by A with isolated cut-point λ with Δ)

Definition 1.3.3. Language L is accepted by a *PFA A* with *unbounded one-sided true-biased error*, if there exists a cut-point $\lambda = 0$, such that:

$$\begin{aligned} L = L(A,\lambda) &= \{x \mid x \in \Sigma^*,\ 0 < p_A(x)\} \\ L^c = L(A,\lambda)^c &= \{x \mid x \in \Sigma^*,\ 0 = p_A(x)\} \end{aligned} \tag{1.3}$$

(L is accepted by A with cut-point $\lambda = 0$)

Definition 1.3.4. Language L is accepted by a *PFA A* with *bounded one-sided true-biased error* with *error bound* $0 < \Lambda$, if there exists an isolated cut-point λ such that:

$$\begin{aligned} L = L(A,\lambda) &= \{x \mid x \in \Sigma^*,\ 1{-}\Lambda \leq p_A(x)\} \\ L^c = L(A,\lambda)^c &= \{x \mid x \in \Sigma^*, \qquad 0 = p_A(x)\} \end{aligned} \tag{1.4}$$

(L is accepted by A with isolated cut-point $\lambda = \frac{1-\Lambda}{2}$ with $\Delta = \frac{1-\Lambda}{2}$)

Remark. We call this type of one-sided error *true-biased*, because the *PFA* is always correct when it answers **true** ("accepts" in one computation). We chose the definition in such a way, so that A accepting L with error bound Λ=0 is deterministic.

Both the true-biased one-sided (*bounded* and/or *unbounded*) probabilistic automata and the non-deterministic ones, accept w only if there exists at least one accepting computation on it (*bounded* Monte-Carlo additionally requires that it occur with probability $p \geq 1{-}\Lambda$), and require words outside L not to have any accepting computation. Hence, we see that these models are, in some sense, a special case of non-determinism (because they can be simulated by it). Moreover, since the *unbounded* Monte-Carlo randomization requires the probability of accepting a word in L only to be positive, it can simulate non-determinism trivially.

Now we define a model that we have not yet seen defined properly.

Definition 1.3.5. Language L is accepted by a *PFA A* with *unbounded one-sided false-biased error*, if

$$\begin{aligned} L &= \{x \mid x \in \Sigma^*,\ p_A(x) = 1\} \\ L^c &= \{x \mid x \in \Sigma^*,\ p_A(x) < 1\} \end{aligned} \tag{1.5}$$

($0 \leq \lambda < 1$, thus we cannot define this in the terms of accepting with cut-point)

Definition 1.3.6. Language L is accepted by a *PFA A* with *bounded one-sided false-biased error* with *error bound* $0 \leq \Lambda < 1$, if there exists an isolated cut-point λ such that:

$$\begin{aligned} L = L(A,\lambda) &= \{x \mid x \in \Sigma^*,\ p_A(x) = 1\} \\ L^c = L(A,\lambda)^c &= \{x \mid x \in \Sigma^*,\ p_A(x) \leq \Lambda\} \end{aligned} \tag{1.6}$$

(L is accepted by A with isolated cut-point $\lambda = \Lambda + \frac{1-\Lambda}{2}$ with $\Delta = \frac{1-\Lambda}{2}$)

From now on, when considering languages recognized by a *PFA* with *(un)bounded one-sided true-biased error*, we will refer to it as a language recognized by a *PFA* with *(un)bounded true-biased error* for brevity. Moreover, we omit specifying that the error is *bounded* when we consider a specific *error bound* Λ. For example a language recognized by a *PFA* with *true-biased error* with *error bound* Λ. On the other hand, when it is not useful to mention a specific *error bound* Λ, we simply omit it.

We now prove a pair o lemmas which show, in some sense, that the *true-biased* and *false-biased* errors are complementary. This will prove very useful in later chapters.

Lemma 1.3.7. *For a language L accepted by a PFA A with bounded false-biased error* (Λ)*, we can construct a PFA A′ recognizing* L^c *with bounded true-biased error* (Λ)*.*

Lemma 1.3.8. *For a language L accepted by a PFA A with bounded true-biased error* (Λ)*, we can construct a PFA A′ recognizing* L^c *with bounded false-biased error* (Λ)*.*

Proof. (Of Lemma 1.3.7) Let *PFA* $A = (Q, \Sigma, \delta, Q_{acc}, Q_{rej})$. We construct a probabilistic automaton A' accepting L^c with *bounded true-biased error* as follows: $A' = (Q, \Sigma, \delta, Q_{rej}, Q_{acc})$. (We swap Q_{rej} and Q_{acc}). To prove that the above constructed A' actually satisfies the conditions, we notice, reading definition, that A accepts L with bounded false-biased error (Λ), hence:
For $w \in L (w \notin L^c)$, A accepts w with $p(w) = 1$, therefore, A' rejects w with $p(w) = 1$.
For $w \notin L (w \in L^c)$, A accepts w with $p(w) \leq \Lambda$, therefore, A' rejects w with $p(w) \leq \Lambda$.
Hence, for $w \in L^c$, A' accepts w with $p(w) \geq 1 - \Lambda$, and for $w \notin L^c$, with $p(w) = 0$. Therefore, reading Definition 1.3.4 , A' accepts L^c with one-sided true-biased error, with error bound $\Lambda' = \Lambda$. □

Proof. (Of Lemma 1.3.8) Analogous to the above, with the argumentation as follows:
For $w \in L (w \notin L^c)$, A accepts w with $p(w) \geq 1-\Lambda$, thus, A' rejects w with $p(w) \geq 1-\Lambda$.
For $w \notin L (w \in L^c)$, A accepts w with $p(w) = 0$, thus, A' rejects w with $p(w) = 0$.
Hence, for $w \in L^c$, A' accepts w with $p(w) = 1 - 0$, for $w \notin L^c$, with $p(w) \leq 1 - (1-\Lambda)$. Therefore, reading Definition 1.3.6 , A' accepts L^c with one-sided false-biased error, with error bound $\Lambda' = \Lambda$. □

Remark 1.3.9. Analogous lemmas can be stated for unbounded one-sided error. Their proofs are analogous, just replace $\geq$ by $>$ and $\leq$ by $<$ in the proofs above, while setting $\Lambda = 0$.

1.4 Multi-head Probabilistic Automata

Before we define the multi-head *PFA*, we define a shortcut notation, since we often use the alphabet Σ, extended by the end-marker \$, $\$ \notin \Sigma$. Let $\Sigma_\$$ denote $\Sigma \cup \{\$\}$.

Definition 1.4.1. A *k-head one-way probabilistic finite automaton* (*PFA*(k)) *over* Σ *with allowed probabilities* T_P is a 6-tuple $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ where

- $Q = \{q_0, \ldots, q_n\}$ is a finite set (the set of *states*),
- δ is a finitely encoded function (the *transition function*) from $Q \times \Sigma_\$^k \times Q \times \{0,1\}^k$ into $\{0,1\} \subseteq T_P \subseteq [0,1]$, [3] such that for every q and input $a_1 \ldots a_k$, the function δ is a probability distribution.

$$(\forall q \in Q)(\forall a_1, \ldots, a_k \in \Sigma_\$) \ : \ \sum_{q'} \sum_{d_1,\ldots,d_k} \delta(q, a_1, \ldots, a_k, q', d_1, \ldots, d_k) = 1$$

- q_0 is element of Q (the *initial state*),
- Q_{acc} is a subset of Q (the set of *accepting states*),
- Q_{rej} is a subset of Q (the set of *rejecting states*), such that $Q_{acc} \cap Q_{rej} = \emptyset$

The meaning of $\delta(q, a_1, \ldots, a_k, q', d_1, \ldots, d_k) = \mathbf{p}$ is as follows: Being in a state q, reading $a_1, \ldots, a_k$ on the k heads respectively, the probability of changing state to q' and advancing heads by $d_1, \ldots, d_k$ (head i by $+d_i$) is $\mathbf{p}$. Because of this, sometimes the new state and head advancements $(q', d_1, \ldots, d_k)$ are referred to as the *outcome*. Here, the symbol \$ denotes the right end-marker, i.e., that "the head reached the end."

Remark. The reasoning behind the δ-function is that the probabilistic model of the automaton can move (from a state, reading symbols) to any state and advance any of its k heads. We thus need to assign a probability to each such "action".

Remark. The set T_P is not included in the tuple since we usually want to analyze whether or not a language can be recognized by a certain model of automata. However, a restriction on the allowed probabilities does in fact change the class of languages recognized, as shown in [Rab63] for *PFA*(1) (Theorem 2 and the following comment). Therefore we parametrize the model by the set T_P.

When we refer to k-head probabilistic finite automata (*PFA*(k)), we consider the ones with allowed rational probabilities ($T_P = [0,1] \cap \mathbb{Q}$) unless explicitly stated otherwise. We may also omit the alphabet Σ since it can often be deduced from the context.

Note that a *PFA*(k) will never "halt", since the probability of doing "something" for any situation $(q, a_1, \ldots, a_k)$ has to be 1. On the other hand, the *PFA*(k) may loop indefinitely, not moving any of its heads.

[3] closed interval of real numbers from 0 to 1 inclusive.

Remark. An alternative definition of a k-head one-way deterministic finite automaton *DFA*(k) is, that it is a *PFA*(k) with allowed probabilities $T_P = \{0, 1\}$.

Definition 1.4.2. A *configuration* of a probabilistic k-head finite automaton is an element $(q, w, o_1, \ldots, o_k) \in Q \times \Sigma^* \times \{1, \ldots, |w| + 1\}^k$, which we understand as the automaton being in state q, with its k-heads positioned on offsets $o_1, \ldots, o_k$ into the input word w respectively (or on endmarker \$ if the offset is $|w| + 1$).

Definition 1.4.3. A *step of computation* of a probabilistic k-head finite automaton A is a relation on its configurations, $(q, w, o_1, \ldots, o_k) \vdash_A (q', w, o'_1, \ldots, o'_k)$ such that, for $w = a_1 \ldots a_n (a_{n+1}{=}\$)$:

$$\delta_A(q, a_{o_1}, \ldots, a_{o_k}, q', d_1, \ldots, d_k) > 0 \;\text{ and }\; o'_i - o_i = d_i \text{ for all } i$$

Remark (Head cannot move beyond end-marker). Even though we allowed the transition function to have a nonzero probability for transitions where the head moves from \$, such transition would result in an illegal configuration (index must be in $\{1 \ldots |w|{+}1\}$). Hence, we see that after a head arrives at \$, it remains there.

Definition 1.4.4. A *computation* of a probabilistic k-head finite automaton A on word w, $|w| = n$, is a sequence of configurations

$$(q_0, w, 1, \ldots, 1),\; (q_2, w, o_{12}, \ldots, o_{k2}),\; (q_3, w, o_{13}, \ldots, o_{k3}),\; \ldots,$$

where the following holds: either the computation is finite, of length l, where the l-th configuration is the first such that $o_{1l} = \cdots = o_{kl} = n + 1$, and

$$(q_i, w, o_{1i}, \ldots, o_{ki}) \vdash_A (q_{i+1}, w, o_{1(i+1)}, \ldots, o_{k(i+1)}) \;\text{ for all } i \in \{1, \ldots, l{-}1\}$$

or the computation is infinite, does not contain $(q, w, n{+}1, \ldots, n{+}1)$ for any q, and

$$(q_i, w, o_{1i}, \ldots, o_{ki}) \vdash_A (q_{i+1}, w, o_{1(i+1)}, \ldots, o_{k(i+1)}) \;\text{ for all } i \in \mathbb{N}.$$

If the computation is finite and $q_l \in Q_{acc}$, we say that it is an *accepting computation.* If instead $q_l \in Q_{rej}$, we say that it is a *rejecting computation.* Otherwise, i.e., if $q_l \notin Q_{acc} \cup Q_{rej}$, or the computation is infinite, we say that it is an *inconclusive computation* or that the computation ended in *FAILURE.*

Note that we have not yet used the probabilities defined in *PFA*(k). The *computation* itself is just a sequence of configurations, and since many computations are possible on a *PFA*(k) (with varying probabilities), we now need to define the probabilities of different computations, to be able to “utilize” the power of randomness, to define the languages accepted by *PFA*(k) as in section Types of Errors (1.3).

Definition 1.4.5. We define the *probability of one step of computation* (p_A) of *PFA*(k) A on an input word $w = a_1 a_2 \dots a_n$ (a_{n+1}=\$) as follows:

$$\begin{aligned} p_A((q, w, o_1, \dots, o_k), (q', w, o'_1, \dots, o'_k)) &= \delta_A(q, a_{o_1}, \dots, a_{o_k}, q', d_1, \dots, d_k) \\ &\quad \text{where } \; o'_i - o_i = d_i \text{ for all } i \end{aligned}$$

We then define the *probability of a computation* (p_A) of *PFA*(k) A on the word w as a product[4] of the probabilities of each of the steps of computation:

$$p_A((conf_1), (conf_2), (conf_3), \dots) = \prod_i p_A((conf_i), (conf_{i+1}))$$

Definition 1.4.6. The *probability of accepting/rejecting/FAILURE on a word* x ($p_A(x)$, $p_A^{rej}(x)$, $p_A^{FAIL}(x)$) with a k-head probabilistic finite automaton A is:

$$p_A(x) = \sum_{comp\text{: accepting computation on } x} p_A(comp)$$

$$p_A^{rej}(x) = \sum_{comp\text{: rejecting computation on } x} p_A(comp)$$

$$p_A^{FAIL}(x) = \sum_{comp\text{: inconclusive computation on } x} p_A(comp)$$

The automaton, at some configurations, forks the computation into a few branches, each occurring with the probability specified in the δ-function. This way, the automaton creates a tree of all possible computations on the input word. By induction, we see that the sum of probabilities of doing a sub-computation, starting in the initial configuration, and ending in the respective configurations after i steps is 1.

Initially, the sum is equal to 1 (just the initial state). When moving from one level to the next, a branching may happen (on each configuration), we know that each subtree occurs with probability as defined in the delta-function, which by definition sums to 1. Thus the new sub-computation's probabilities sum up to the probability of this sub-computation. Hence, the sum of the new level will be the same as the sum of the previous level, equal to 1.

Thus $p_A(x) + p_A^{rej}(x) + p_A^{FAIL}(x) = 1$, since eventually, all sub-computations are either accepting, rejecting, inconclusive, or infinite (inconclusive again).

At times, when the automaton in question can easily be deduced from the context, we may omit specifying it in the subscript (e.g. $p^{FAIL}(x)$ instead of $p_A^{FAIL}(x)$).

[4] Or, in the case of infinite computations, a limit of partial products ($\lim_{N\to\infty} \prod_{i=0}^{N} p_A((c_i), (c_{i+1}))$)

Particular models

We define a standard model of randomization, Monte-Carlo. This automaton may answer incorrectly (but must always answer), therefore we define various types of error with which this model can recognize a language.

Definition 1.4.7. A *Monte-Carlo PFA*(k) *over* Σ *with allowed probabilities* T_P is a *PFA*(k) A over Σ with allowed probabilities T_P, such that:

$$(\forall x \in \Sigma^*) : \ p_A^{FAIL}(x) = 0$$

Lemma 1.4.8. *A PFA*(k) *over* Σ *with allowed probabilities* T_P*, where* $Q_{rej} = Q - Q_{acc}$ *and every computation on* A *is finite is a Monte-Carlo PFA*(k)*.*

Proof. The only way for a computation to be inconclusive is to either end up in state not in Q_{rej} nor Q_{acc}, or be infinite. □

Definition 1.4.9 (Language recognized by Monte-Carlo *PFA*(k))**.** Let A be a Monte-Carlo *PFA*(k), λ a real number $0 \leq \lambda < 1$, and $p_A(x)$ the probability of A accepting the word x. The language $L(A, \lambda)$ is defined as follows:

$$L(A, \lambda) = \{w \in \Sigma^* \mid \lambda < p_A(w)\}$$

If $x \in L(A, \lambda)$ we say that x is *accepted by* A *with cut-point* λ. The set $L(A, \lambda)$ will be called the *language recognized by* A *with cut-point* λ.

When defining acceptation of languages with errors for the one-way multi-head Monte-Carlo model, we use **identical** definitions as for *PFA* (see Types of Errors 1.3).

Another widely studied randomization is LasVegas (zero probability of error). Such automaton, must always "answer" correctly, however, it may "not answer", i.e., end in a *FAILURE* (end in a state from $Q - (Q_{acc} \cup Q_{rej})$ or get stuck in a loop indefinitely).

Definition 1.4.10. A *LasVegas PFA*(k) *over* Σ *with allowed probabilities* T_P is a *PFA*(k) A over Σ with allowed probabilities T_P, such that $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ where for all words $x \in \Sigma^*$:

- $p^{FAIL}(x) < 1$
- either $p(x) = 0$ or $p^{rej}(x) = 0$.

If additionally, for all $x \in \Sigma^*$, the probability of *FAILURE* is bounded from above by a constant $0 \leq \kappa < 1$ ($p^{FAIL}(x) \leq \kappa$), we say that A *is* $(1-\kappa)$*-correct LasVegas PFA*(k).

Definition 1.4.11 (Language recognized by LasVegas *PFA*(k))**.** Let A be a LasVegas *PFA*(k). The language $L(A)$ is defined as follows:

$$L(A) = \{w \in \Sigma^* \mid 0 < p_A(w)\}$$

The set $L(A)$ is referred to as the *language recognized by* A.

1.5 Example

We now illustrate few of the previous definitions on an example. Moreover, it is a language that will prove useful later. (The following language is defined over alphabet $\Sigma \cup \{\#\}$.)

Lemma 1.5.1. *The language $L_{uvvu} = \{u\#v\#v'\#u' \mid u{=}u', v{=}v' \in \Sigma^*\}$, $\# \notin \Sigma$, is recognized by a Monte-Carlo PFA(2) A with false-biased error with error bound $\Lambda = 1/2$.*

Remark. Should we be excessively formal, we would write the following lemma:
For an alphabet Σ and a language $L_{uvvu} = \{u\#v\#v\#u \mid u, v \in \Sigma^*\}$ where $\# \notin \Sigma$, there exists a Monte-Carlo PFA(2) A over $\Sigma \cup \{\#\}$ with allowed probabilities $T_P = [0, 1] \cap \mathbb{Q}$, such that L_{uvvu} is a language recognized by A with bounded one-sided false-biased error with error bound $\Lambda = 1/2$.

Proof. The following is a algorithm for *PFA*(2) accepting L_{uvvu}:

0. Head 2, in tandem with its usual instructions, verifies that the word is of format $w_1\#w_2\#w_3\#w_4, w_i \in \Sigma^*$ (a regular check – count #).
1. With probability $\frac{1}{2}$ move head 2 to the beginning of the word u', or, with probability $\frac{1}{2}$ move head 1 to the start of the word v and move head 2 to the beginning the word v'.
2. Accept if and only if the words under heads 1 and 2 are equal. (Move heads 1 and 2 simultaneously while they read the same symbols, until # or \$, else reject).
3. Read until the end of the input word with both heads, then accept.

To prove that this automaton acepts L_{uvvu}, we analyze the two cases of what happens (how does the computation look like) for the words in, and not in the language in question (L_{uvvu}). Each word $w \in L_{uvvu}$, is accepted by A with probability 1, since any two sub-words are equal. For a word $w \notin L_{uvvu}$, the word is either not of the format $w_1\#w_2\#w_3\#w_4$ – which we detect with head 2 – or one of u,u' and v,v' are unequal. Analyzing our algorithm, we see that A rejects such word with probability $\geq \frac{1}{2}$ (arrive at unequal sub-word (with $p = \frac{1}{2}$), then verify the (in)equality – words with bad format are always rejected). Thus, A is a Monte-Carlo *PFA*(k) satisfying the definition of accepting with *one-sided false-biased error* with *error bound* $\Lambda = \frac{1}{2}$.

Formal construction:
$A = (Q, \Sigma \cup \{\#\}, \delta, q_0, Q_{acc}, Q_{rej})$, where
$Q = \{q_0, q_F,\ q_{u[0]}, q_{u[1]}, q_{u[2]}, q_{u[3]}, q_{u[=]},\ q_{v[0]}, q_{v[1]}, q_{v[2]}, q_{v[3]}, q_{v[=]},\ q_{TRASH}\}$
$Q_{acc} = \{q_F\}$
$Q_{rej} = Q - Q_{acc}$

The δ-function is written in compact form, such that if a value for an input for the δ-function is not explicitly set, it is assumed to be zero, and if for some "input" 3-tuple (q, a_1, a_2) the δ-function is never defined, set $\delta(q, a_1, a_2, q_{TRASH}, 0, 0)$=1. Additionally, whenever we write _, we would write the "rule" for each $\_ \in \Sigma \cup \{\#\}$, and whenever we write a, we would write the "rule" for each $a \in \Sigma$ (not #, nor $).

Remark. We needed to define the state q_{TRASH} and transitions into it, only to satisfy the definition of $PFA(2)$ (1.4.1). Specifically, that the δ-function be defined for all inputs, and that the sum for each 3-tuple (q, a_1, a_2) be equal to 1.

$$
\begin{array}{ll}
\text{//Choose sub-word} & \text{//Goto sub-word } v, v' \\
\delta(q_0, \_, \_, q_{u[0]}, 0, 0) = \frac{1}{2} & \delta(q_{v[0]},\ a,\ a, q_{v[0]}, 1, 1) = 1 \\
\delta(q_0, \_, \_, q_{v[0]}, 0, 0) = \frac{1}{2} & \delta(q_{v[0]}, \#, \#, q_{v[1]}, 1, 1) = 1 \\
 & \delta(q_{v[1]}, \_,\ a, q_{v[1]}, 0, 1) = 1 \\
 & \delta(q_{v[1]}, \_, \#, q_{v[=]}, 0, 1) = 1 \\
 & \\
\text{//Goto sub-word } u, u' & \text{//Verify equality} \\
\delta(q_{u[0]}, \_,\ a, q_{u[0]}, 0, 1) = 1 & \delta(q_{v[=]},\ a,\ a, q_{v[=]}, 1, 1) = 1 \\
\delta(q_{u[0]}, \_, \#, q_{u[1]}, 0, 1) = 1 & \delta(q_{v[=]}, \#, \#, q_{v[3]}, 0, 1) = 1 \\
\delta(q_{u[1]}, \_,\ a, q_{u[1]}, 0, 1) = 1 & \\
\delta(q_{u[1]}, \_, \#, q_{u[2]}, 0, 1) = 1 & \text{//Finish verifying format} \\
\delta(q_{u[2]}, \_,\ a, q_{u[2]}, 0, 1) = 1 & \delta(q_{v[3]}, \_,\ a, q_{v[3]}, 0, 1) = 1 \\
\delta(q_{u[2]}, \_, \#, q_{u[=]}, 0, 1) = 1 & \delta(q_{v[3]}, \_,\ \$, q_F, 0, 0) = 1 \\
 & \\
\text{//Verify equality, and accept} & \\
\delta(q_{u[=]},\ a,\ a, q_{u[=]}, 1, 1) = 1 & \text{//Head 1 to \$, then } ACCEPT \\
\delta(q_{u[=]}, \#,\ \$, q_F, 0, 0) = 1 & \delta(q_F, \_, \$, q_F, 1, 0) = 1 \\
 & \\
 & \text{//Loop in } q_{TRASH} \text{ until reject} \\
 & \delta(q_{TRASH}, \_, \_, q_{TRASH}, 1, 1) = 1 \\
 & \delta(q_{TRASH}, \_,\ \$, q_{TRASH}, 1, 0) = 1 \\
 & \delta(q_{TRASH},\ \$, \_, q_{TRASH}, 0, 1) = 1
\end{array}
\tag{1.7}
$$

□

Remark. Note that the proof would also have worked had we required $T_P = \{0, \frac{1}{2}, 1\}$.

1.6 Epsilon-free normal form

An interesting question that is often asked when considering models of automata, is whether or not we can do without having the possibility to stay at one place. In this case whether $\delta(q, a_1, \ldots, a_k, p, 0, \ldots, 0)$ can always be equal to 0. Ideally we want to construct an "equivalent" automaton to every valid *PFA*(k) so that the new automaton will move at least one head every step of computation. We show that it is feasible.

Definition 1.6.1. The two *PFA*(k) A_0 and A_1 are *equivalent* if for each word $w \in \Sigma^*$: $p_{A_0}(w) = p_{A_1}(w)$, $p_{A_0}^{FAIL}(w) = p_{A_1}^{FAIL}(w)$, and $p_{A_0}^{rej}(w) = p_{A_1}^{rej}(w)$.

Remark. We have chosen this definition since we want it to be universal, to work for any particular model of acceptance. For Monte-Carlo, LasVegas and other models.

Definition 1.6.2 (ε-free form of *PFA*(k))**.** A k-head probabilistic finite automaton $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ is in *ε-free form* if

$$(\forall q, p \in Q)(\forall a_1, \ldots, a_k \in \Sigma_\$) : \ \delta(q, a_1, \ldots, a_k, p, 0, \ldots, 0){=}0$$

Theorem 1.6.3 (ε-free form is a normal form for *PFA*(k))**.** *Let A_0 be a PFA(k) with allowed probabilities T_P. Then there exists a PFA(k) A_1 with allowed probabilities* $[0, 1]$ *in ε-free form equivalent with A_0.*

Informally. This construction is quite long, thus we give an overview of how it works. The main point of our construction is to replace infinite computations (which are inconclusive by definition) and arbitrarily long computations (a loop which the automaton can leave) in such a way, that the automaton accepts/rejects each word with the same probability as did the original automaton.

Our construction consists of 4 steps divided into sub-steps, at each sub-step, we have a *PFA*(k) that accepts and rejects each word with the same probability as A_0.

1) We divide the automaton into many copies, *layers*, such that on one layer, transitions expect to read only one specific input $a_1, \ldots, a_k$ (we load input into a "buffer" and only later do the original transition). The motivation is that this way, in one certain layer, the transition depends only on the state (since the input is fixed).

2) We then divide each state into 3 sub-states: ingoing (s_{IN}), outgoing without moving heads (s_ε), and outgoing moving at least one head (s_a). Not moving heads leaves the input the same, thus if the automaton moves no head, it will stay in the same layer. The motivation is that now, we have states that either only use transitions that move at least one, or only use transitions that move no head. Moreover "stationary" transitions are stuck on the same layer.

3) The main part will be using the theory of Markov chains. We first recognize that by looking at any one layer of the automaton (and redefining s_a so that they

loop in themselves) we get a Markov chain. In a Markov chain, each state is either *ergodic* or *transient*. We first take care of each state that is *ergodic* (and not s_a). A state is *ergodic* if you eventually return to this state with probability 1. Since we have only considered transitions that move no head, it means that if the automaton enters this state, it will loop indefinitely (resulting in inconclusive computation). Hence, we delete all outgoing edges from this state and redirect it into a state q_{FAIL}, a new state in which the automaton just walks to the end marker and ends in *FAILURE*. Now that we have taken care of the infinite loops, each state (except s_a) in our Markov chain is *transient*, which means that eventually, we will leave this state and never return. By the theory of Markov chains, we can compute the probability of eventually ending in each of the states s_a or in the infinitely looping states (which all end in q_{FAIL}). Therefore we replace all these transitions, by transitions jumping from one state to another state's s_a or the fake-infinite-loop q_{FAIL}. We repeat this for each layer.

4) We finalize the construction by tidying up, since by the above construction, we have only removed the arbitrarily long and the infinite computations, i.e., the automaton does at most $O(n)$ steps of computation. Yet we wish to have no transition that would not move at least one head.

Proof. Let $A := A_0 = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$. We prove the theorem by constructing a sequence of automata (A', A'', A'''=$\bar{A}_0$, $\bar{A}_1, \ldots, \bar{A}_m$, $\bar{\bar{A}}$ where m=$(|\Sigma|+1)^k$), such that it is easy to see that two consecutive automata are *equivalent*.

Step 1 (A'): The first step is to construct a *PFA*(k), in which every state q from A will be copied into $m+1 = (|\Sigma|+1)^k+1$ new states: The input state q_{IN} into which all transitions which would end in q will now end, and the extended states $q_{[a_1,\ldots,a_k]}$, for every possible $a_1, \ldots, a_k \in \Sigma_\$$, which simply "store" the input of the heads, and from which the automaton makes the original outgoing transitions. (As if breaks each original transition into two: "save" the input into the state; do the random decision.)

Formally we construct $A' = (Q', \Sigma, \delta', q_{0,IN}, Q'_{acc}, Q'_{rej})$ as follows:

$Q' = \{q_{IN} \mid q \in Q\} \cup \{q_{[a_1,\ldots,a_k]} \mid (q \in Q) \wedge (a_1, \ldots, a_k \in \Sigma_\$)\}$

$Q'_{acc} = \{q_{IN} \mid q \in Q_{acc}\} \cup \{q_{[a_1,\ldots,a_k]} \mid (q \in Q_{acc}) \wedge (a_1, \ldots, a_k \in \Sigma_\$)\}$

$Q'_{rej} = \{q_{IN} \mid q \in Q_{rej}\} \cup \{q_{[a_1,\ldots,a_k]} \mid (q \in Q_{rej}) \wedge (a_1, \ldots, a_k \in \Sigma_\$)\}$

We additionally define $Q'_{IN} = \{q_{IN} \mid q \in Q\}$, so that we can refer to it later.

For each $q \in Q,\ a_1, \ldots, a_k \in \Sigma_\$$:

$\delta'(q_{IN}, a_1, \ldots, a_k, q_{[a_1,\ldots,a_k]}, 0, \ldots, 0)$=1

For each $q, p \in Q,\ a_1, \ldots, a_k \in \Sigma_\$,\ d_1, \ldots, d_k \in \{0, 1\}$ such that $max\{d_1, \ldots, d_k\}$=1:

$\delta(q, a_1, \ldots, a_k, p, d_1, \ldots, d_k)$=$\mathbf{p}$>0 $\implies$ $\delta'(q_{[a_1,\ldots,a_k]}, a_1, \ldots, a_k, p_{IN}, d_1, \ldots, d_k)$=$\mathbf{p}$

The delta-function is zero for any other inputs.

For each computation on A, there is a corresponding computation on A', where instead of directly moving from $(q_i, w, o_{1i}, \ldots, o_{ki})$ to $(q_{(i+1)}, w, o_{1(i+1)}, \ldots, o_{k(i+1)})$ with probability $\mathbf{p}$, A' does one intermediate step. The extra step is from $(q_{i,IN}, w, o_{1i}, \ldots, o_{ki})$ to $(q_{i,[w_{o_1},\ldots,w_{o_k}]}, w, o_{1i}, \ldots, o_{ki})$ with probability 1, followed by the randomized step, into $(q_{(i+1),IN}, w, o_{1(i+1)}, \ldots, o_{k(i+1)})$ with probability $\mathbf{p}$. Moreover, each computation on A' is of this form, i.e., corresponds to some computation on A. Since there is a one-to-one correspondence of computations on A and A' (preserving probabilities), the following holds. For each word w: $p_A(w) = p_{A'}(w)$, $p_A^{FAIL}(w) = p_{A'}^{FAIL}(w)$, $p_A^{rej}(w) = p_{A'}^{rej}(w)$. Therefore, A' and A are equivalent.

A common visualization of an automaton is a graph, where the vertices are the states, and edges between states represent transitions (what has to be read, how heads will advance). In our case, of *PFA*(k), the edges additionally contain the probability $\mathbf{p}$ of doing that transition. We represent each transition by an edge. If the transition moves no head, then the edge is dashed, and if the transition occurs with probability 0, it is dotted edge.

$$\xrightarrow{a_1,\ldots,a_k;\ \mathbf{p};\ d_1,\ldots,d_k}$$

The automaton A' can be viewed as having $m+1$ copies of A, where edges go from the lowest *IN* layer, reading $a_1, \ldots, a_k$ to layer $[a_1, \ldots, a_k]$, from which they go (maybe after a random decision) back to the lowest *IN* layer, possibly moving heads.

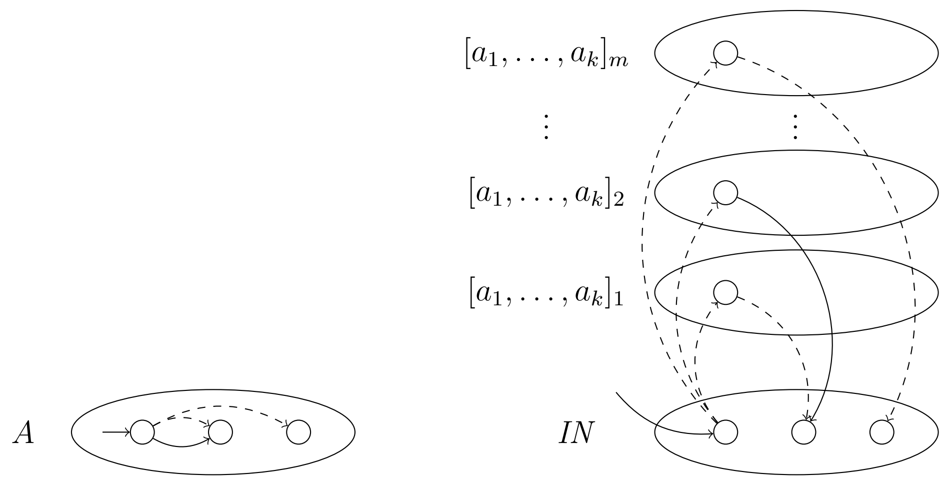

Figure 1.1: Construction of A' from A: multiple layers

Remark. The motivation for A' is that the first step into the layer $[a_1, \ldots, a_k]$ by "saving" what is being read into the state ensures that while in the same layer, transitions occur based on the state only (since by their layer we know what is being read).

Step 2.1 (A''): We construct a *PFA*(k) in which each state has the following property: either no transition from it moves heads, or all transitions from it move at least one. (We split each state into sub-states and then we put all transitions which move at least one head to one sub-state, and all transitions that move no head to another.)

We construct A'' by splitting each state $s{=}q_{[a_1,\ldots,a_k]} \in Q'$ (not q_{IN}) into three sub-states s_{IN}, s_ε, and s_a (in this proof, the letter s denotes some state $q_{[a_1,\ldots,a_k]}$ in one of the layers). State s_{IN} is the input state, such that transitions ending in s will now end in s_{IN}. Then (while in s_{IN}) the automaton will randomly, with suitably chosen probabilities, choose whether it is going to move at least one, or move none of its heads (transition to s_ε or s_a). The state s_ε is a state from which the automaton will move as A' moves from $s(=q_{[a_1,\ldots,a_k]})$, but choosing only among transitions that move no head. State s_a, on the other hand, is a state from which the automaton will move as if from s, yet choosing only from the transitions that move at least one head.

In order to construct such A'', we need, for each state, to compute the probability of doing a transition which moves at least one head, and the probability of doing a transition that is not moving any heads, so that we can later scale the probabilities. (Here we are using the structure of A', that the probabilistic transitions must end in a state in layer IN, and that when in layer $[a_1,\ldots,a_k]$, the input $(a_1,\ldots,a_k)$ is "saved" in the state. More specifically, we know that in such state, A' is guaranteed to read $a_1,\ldots,a_k$ on its heads.)

$$(\forall s{=}q_{[a_1,\ldots,a_k]} \in Q') \quad \mathbf{p}_{s,\varepsilon} = \sum_{p_{IN}\in Q'_{IN}} \delta'(q_{[a_1,\ldots,a_k]}, a_1,\ldots,a_k, p_{IN}, 0,\ldots,0)$$

$$\mathbf{p}_{s,a} = 1 - \mathbf{p}_{s,\varepsilon}$$

Note that $\mathbf{p}_{s,a}$ is the sum of probabilities of transitions that move at least one head.

Formally, we construct $A'' = (Q'', \Sigma, \delta'', q_{0,IN}, Q''_{acc}, Q''_{rej})$ as follows:
We first define $Q'_{[layer]} = Q' - Q'_{IN}$.

$Q'' \;= \{s_{IN}, s_\varepsilon, s_a \mid s \in Q'_{[layer]}\} \cup Q'_{IN}$

$Q''_{acc} = \{s_{IN}, s_\varepsilon, s_a \mid s \in (Q'_{acc} - Q'_{IN})\} \cup (Q'_{IN} \cap Q'_{acc})$

$Q''_{rej} = \{s_{IN}, s_\varepsilon, s_a \mid s \in (Q'_{rej} - Q'_{IN})\} \cup (Q'_{IN} \cap Q'_{acc})$

For each $s \in Q'_{[layer]},\ a_1,\ldots,a_k \in \Sigma_\$$:

$\delta''(s_{IN}, a_1,\ldots,a_k, s_\varepsilon, 0,\ldots,0) = \mathbf{p}_{s,\varepsilon}$

$\delta''(s_{IN}, a_1,\ldots,a_k, s_a, 0,\ldots,0) = \mathbf{p}_{s,a}$

For each $s \in Q'_{[layer]},\ a_1,\ldots,a_k \in \Sigma_\$,\ p_{IN} \in Q'_{IN},\ d_1,\ldots,d_k \in \{0,1\}$ where $\sum_i d_i \geq 1$:

$\delta'(s, a_1,\ldots,a_k, p_{IN},\ 0,\ldots,0) \;= \mathbf{p}{>}0 \implies \delta''(s_\varepsilon, a_1,\ldots,a_k, p_{IN}, 0,\ldots,0) = \frac{\mathbf{p}}{\mathbf{p}_{s,\varepsilon}}$

$\delta'(s, a_1,\ldots,a_k, p_{IN}, d_1,\ldots,d_k) = \mathbf{p}{>}0 \implies \delta''(s_a, a_1,\ldots,a_k, p_{IN}, d_1,\ldots,d_k) = \frac{\mathbf{p}}{\mathbf{p}_{s,a}}$

(the fraction is never $\frac{\mathbf{p}}{0}$, since when $\mathbf{p}{>}0$, the corresponding $\mathbf{p}_{s,a}$ or $\mathbf{p}_{s,\varepsilon}$ is also nonzero)

By this construction, each transition that went from $q_{1,IN}$ to s_1 to $q_{2,IN}$ with probability $1 \cdot \mathbf{p} = \mathbf{p}$, now goes from $q_{1,IN}$ to $s_{1,IN}$ to $s_{1,\varepsilon}$ to $q_{2,IN}$ with probability

$1 \cdot \mathbf{p}_\varepsilon \cdot \mathbf{p}/\mathbf{p}_\varepsilon = \mathbf{p}$ (without moving heads, and analogously for $\mathbf{p}_a$ when moving heads). Just as when constructing A', we "extended" each path between $q_{1,IN}$ and $q_{2,IN}$, and since there is a one-to-one correspondence between these computations (preserving probabilities), it follows that A' and A are equivalent.

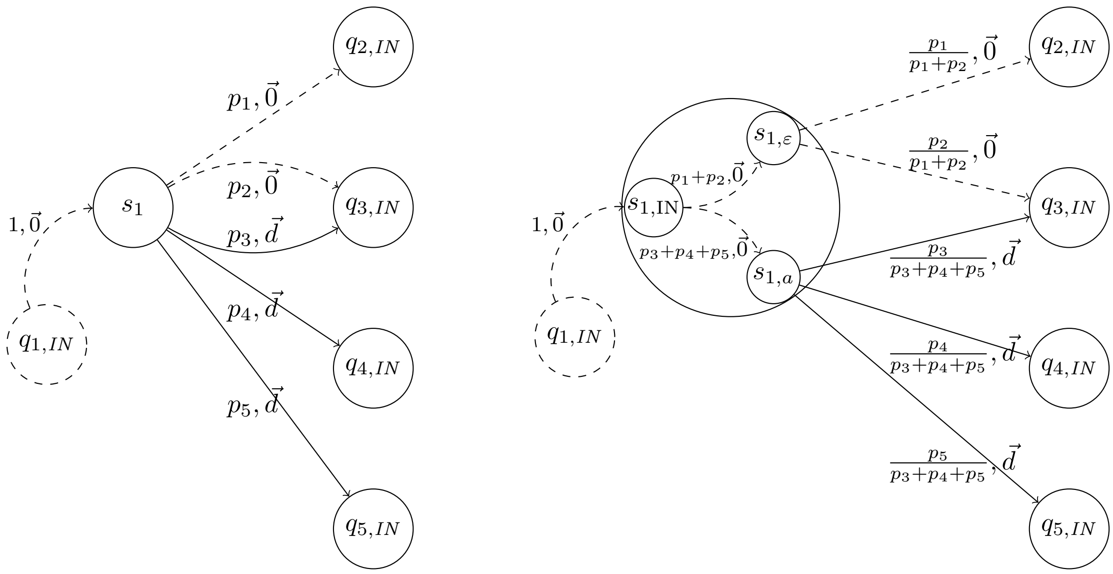

Figure 1.2: Construction of A'': splitting of states into three

Step 2.2 (A'''): When analyzing A'', we see that when A'' is in a state $s_{i,\varepsilon}$, the heads will not move in the upcoming transition. Thus, after a transition, A'' will read the same "input" as it did arriving into $s_{i,IN}(=q_{i,[a_1,\ldots,a_k],IN})$. Therefore, when A'' moves from $s_{1,\varepsilon}$ into a state $q_{2,IN}$, we know that it will certainly go into the state $s_{2,IN}(=q_{2,[a_1,\ldots,a_k],IN})$. We edit the automaton, so that it does not take the detour through $q_{2,IN}$ and only moves from $s_{1,\varepsilon}$ directly to $s_{2,IN}$. Formally, for $s_{i,\varepsilon}$ in layer $[a_1,\ldots,a_k]$:

$$\begin{aligned}\delta''(s_{i,\varepsilon}, a_1, \ldots, a_k, q_{j,IN}, 0, \ldots, 0) = \mathbf{p} \;&\Rightarrow\; \delta'''(s_{i,\varepsilon}, a_1, \ldots, a_k, q_{j,[a_1,\ldots,a_k],IN}, 0, \ldots, 0) = \mathbf{p} \\ &\Rightarrow\; \delta'''(s_{i,\varepsilon}, a_1, \ldots, a_k, q_{j,IN}, 0, \ldots, 0) = 0\end{aligned}$$

What this construction does, is that it edits only the δ-function in A'', in such a way that joins two consequent transitions into one. Thus, for each computation on A'' there is a corresponding one on A''' (and back). Hence, A'' and A''' are equivalent.

The automaton A''' has many layers, just as A' did. However, the shortest path between two states in the layer *IN* is now longer. For each original (A) step from q_1 to q_2, advancing at least **one** head, A''' now goes from $q_{1,IN}$, into $s_{1,IN}$, into $s_{1,a}$, all without moving any of its heads. It then finally "leaves q_1" by doing the "original" transition into $q_{2,IN}$. Had the original moved **no** head, the simulated "original" transition would take the automaton into $s_{2,IN}$, since it would end up there anyway.

Remark. The advantage of A''' is that if it moves no head, it does not leave the one layer in which it is, i.e., it enters the layer *IN* only when it moves at least one head.

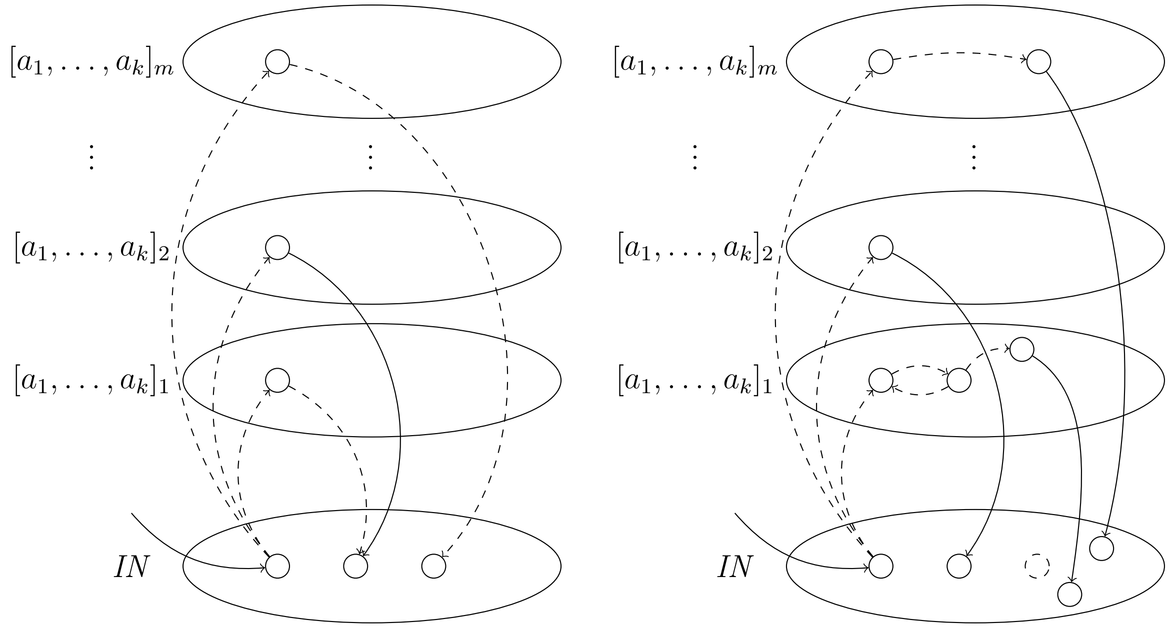

Figure 1.3: Comparison A' vs A''': stay in layer if not moving heads

Step 3 ($\bar{A}_1$): This step of contruction will eliminate infinite computations. We do that by creating an equivalent *PFA*(k), such that on each word it does at most $O(n)$ steps of computation before accepting, rejecting, or ending in *FAILURE*.

We begin the construction by taking the graph of the lexicographically first layer of the automaton $\bar{A}_0{=}A'''$ (since we are building $\bar{A}_{\mathbf{1}}$). We wish to view this graph as a Markov chain [KS60](page 24,25). *Markov chain is a finite stochastic process that has the Markov property and the transition probabilities $p_{ij}(n)$ do not depend on n.* The Markov property is essentially [KS60](page 24): *Knowing the outcome of the last experiment we can neglect any other information we have about the past in predicting the future.* Our (graph of) one layer ($[a_1,\dots,a_k]$) of the automaton represents a stochastic process where the transitions between states depend solely on the previous state (in one layer we read the same input), with probabilities defined via the δ-function (thus not changing), with slight tweak such that when in state $s_{i,a}$ we stay in it with probability 1. (States of the automaton are states of the Markov chain.)

In the theory of Markov chains, we divide the states of a Markov chain into equivalence classes according to whether they are mutually reachable (with non-zero probability). We additionally define a partial ordering between these classes according to whether it is possible to go from a given class to another given class. The partial ordering shows us the possible directions in which the process can proceed [KS60](page 35). *The minimal elements of the partial ordering of equivalence classes are called ergodic sets. The remaining elements are called transient sets. The elements of a transient set are called transient states. The elements of an ergodic set are called ergodic (or non-transient) states.* We see that if a process leaves a *transient* set it can never return to this set, while if it once enters an *ergodic* set, it can never leave it. Additionally, if an ergodic set contains only one element, we call such a state *absorbing*.

Applying this theory to our situation, we see that $\{s_{i,a}\}$ are *ergodic* sets for each i (we defined δ this way for a reason, since if a process in this layer arrives in this state, it will leave this layer). Moreover, if any other set of states is *ergodic*, should the automaton arrive there, it will be stuck there forever (since such set cannot contain $s_{i,a}$) resulting in an inconclusive computation (inside a layer, the automaton does not move its heads, and the only way out of this layer is through one of the states $s_{i,a}$).

The following figure 1.4 represents a Markov Chain (of one layer) with 15 states. Should we look at it as on the original automaton A (before we split the states in three), it represents "5" states: An *ergodic* set of "two" states (an infinite loop), and a set of "3" mutually connected *transient* states (allowing looping of arbitrary length). // The state q_{IN} is from layer IN (not in Markov chain), all other states are in the layer $[a_1, \ldots, a_k]$. Dotted lines represent transitions with probabilities 0. States $s_{i,a}$ are specially highlighted, since they are the gateway out. //

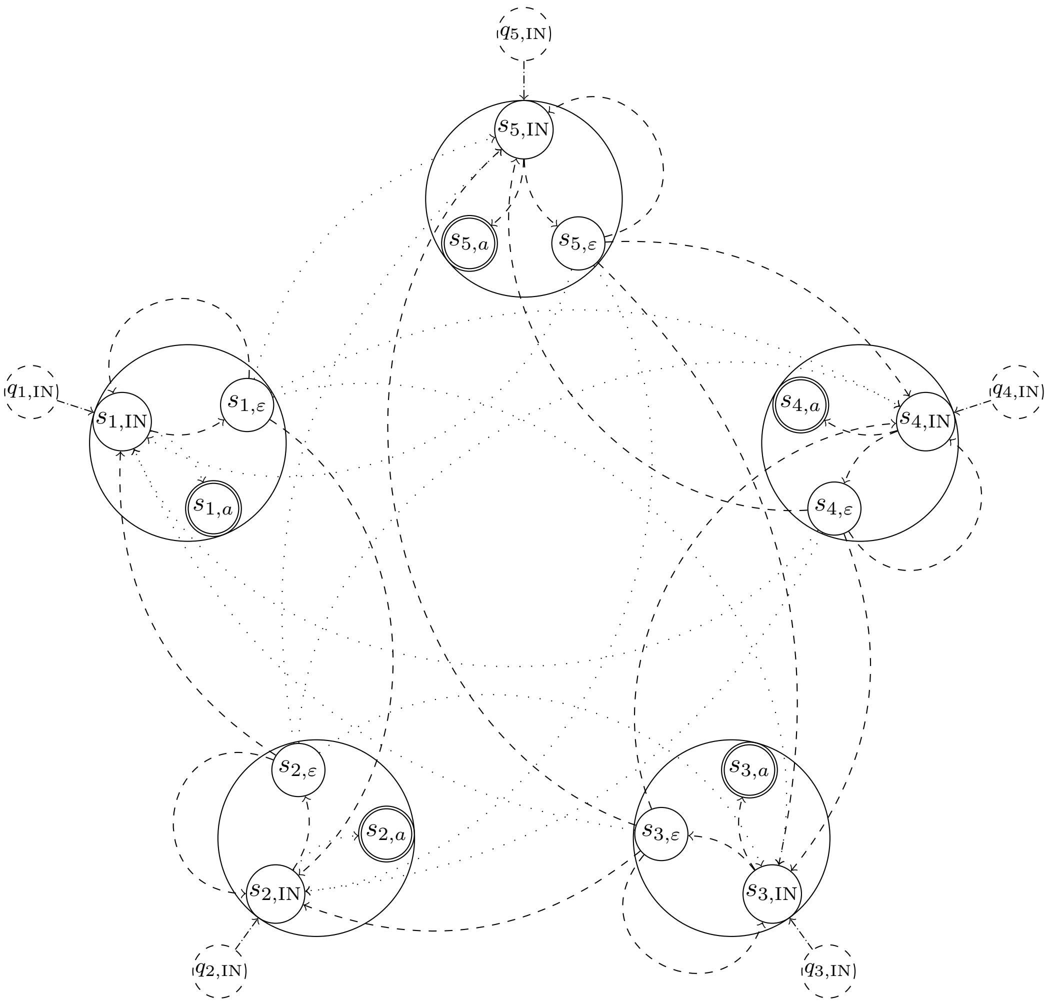

Figure 1.4: Example layer of $\bar{A}_0$: $3 \cdot 3$ *transient*, $2 \cdot 3$ *ergodic*

Our construction then follows by creating a new state q_{FAIL}, that simply takes all heads to the end-marker and then ends in *FAILURE* (there will be only one q_{FAIL}, outside all layers into which we later "throw" all inconclusive infinite loops).

We now analyze each state of the above constructed Markov chain, except for the states $s_{j,a}$. By definition, each state is either *ergodic* or *transient*. For any state that is *ergodic* (we know that the automaton will loop indefinitely, i.e., end inconclusively), we re-define the δ-function so that the only outgoing transition from this state is into q_{FAIL} with probability 1, and add the *absorbing* state q_{FAIL} into the Markov chain.

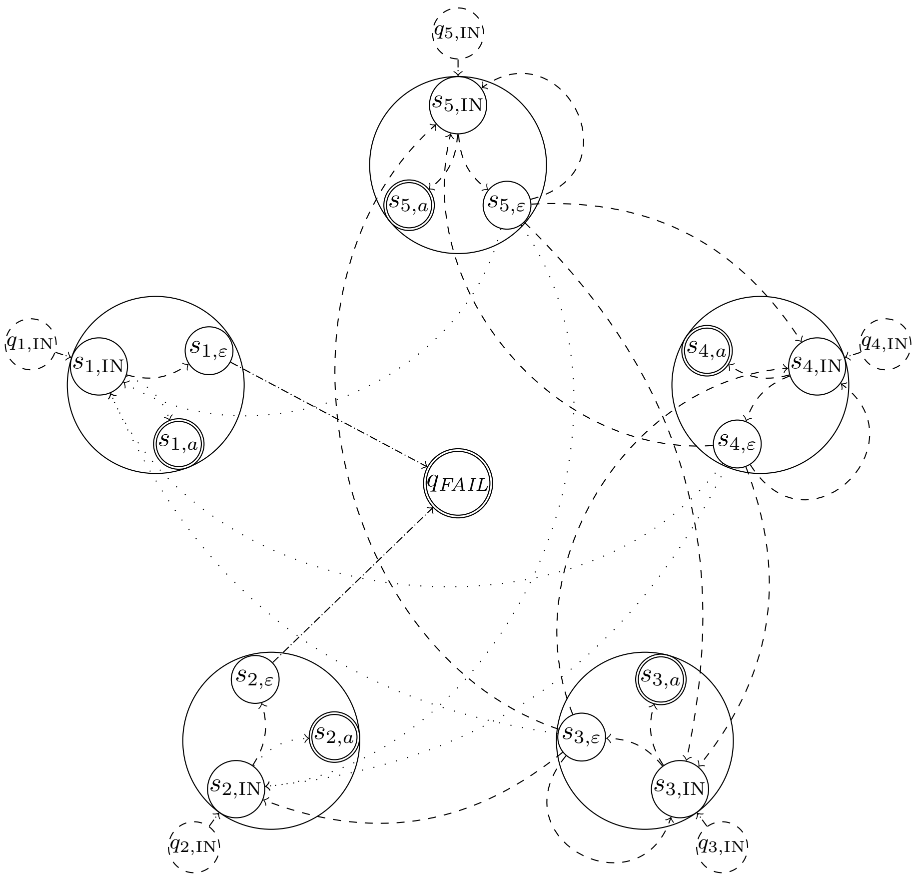

Figure 1.5: Construction of $\bar{A}$: removal of infinite loops

This way, each state, except for $s_{j,a}$ and q_{FAIL}, is *transient*. Then the probability of staying in this layer forever is 0, since the probability of ending in some absorbing state is 1 [KS60](page 43, theorem 3.1.1). Therefore, by [KS60](page 52, theorem 3.3.7) we can calculate, for each transient state, the *probability that the process starting in transient state $s_{i,\varepsilon}$ ends up in absorbing state $s_{j,a}$ (q_{FAIL}).* Thus, we can remove all outgoing transitions from $s_{i,\varepsilon}$, replacing them by direct transitions from $s_{i,\varepsilon}$ to $s_{j,a}$ (q_{FAIL}), for each i,j. This construction preserves the probability of ending in $s_{j,a}$ (q_{FAIL}) from $q_{i,IN}$ for each i,j. Hence, an automaton constructed this way (by changing one layer) is equivalent with $\bar{A}_0$. See Figure 1.6 for illustration.

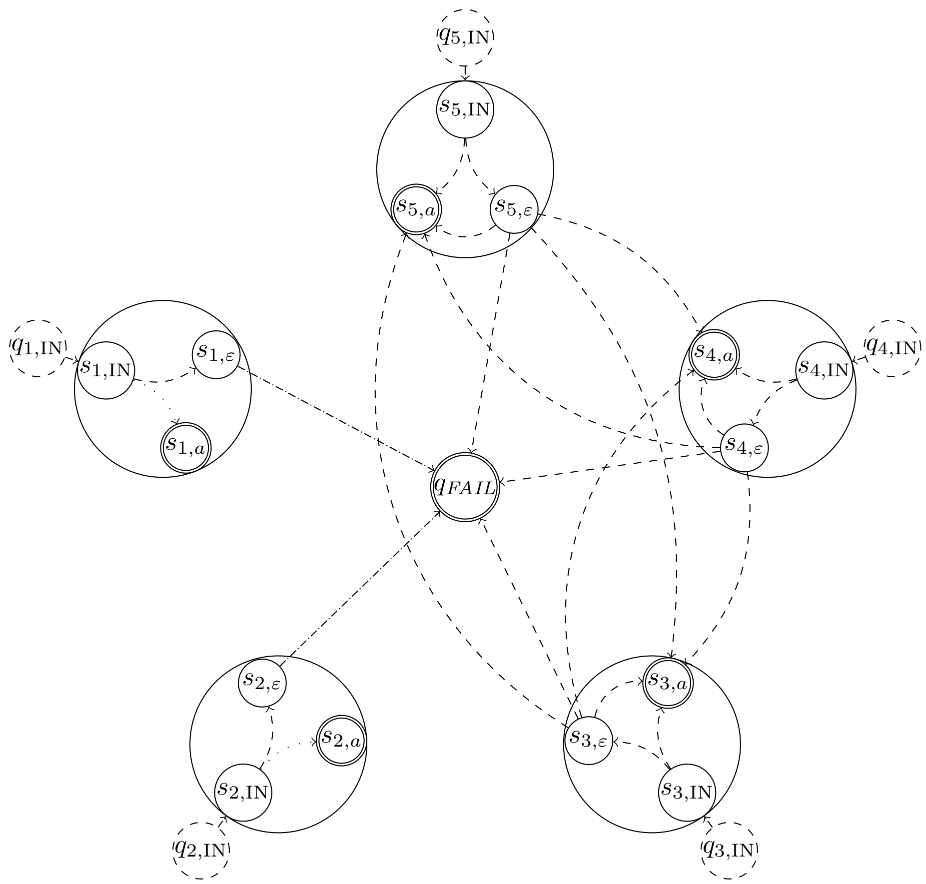

Figure 1.6: Construction of $\bar{A}$: building of direct transitions

What is more, we can also calculate the probability of moving from $q_{i,IN}$ to $s_{j,a}$ (or q_{FAIL}), which will allow us to replace all except one stationary transition, making states $s_{i,IN}$ and $s_{i,\varepsilon}$ irrelevant. We calculate it by summing the probabilities of all possible paths (if $i{=}j$ there are 2 paths, otherwise there is only one) between these states, which we get by multiplying the probabilities of transitions. Example ($i{\neq}j$): $\mathbf{p}(q_{i,IN}, s_{j,a}) = 1 \cdot \mathbf{p}(s_{i,IN}, s_{i,\varepsilon}) \cdot \mathbf{p}(s_{i,\varepsilon}, s_{j,a})$. See Figure 1.7 for illustration.

By this construction, we have "solved" our problem for one transition layer. Should the automaton read the input $(a_1, \ldots, a_k)$ that would lead it to this layer, the automaton will make just one transition without moving heads into some state $s_{j,a}$ (or q_{FAIL}), from which it will do a transition that moves at least one of its heads.

We repeat this process (Step 3) for each layer, creating m automata along the way $\bar{A_1}, \bar{A_2}, \ldots, \bar{A_m}$ ($m = (|\Sigma| + 1)^k$). We see that the final automaton ($\bar{A_m}$) makes at most 1 "consecutive" step without moving its heads. It is the transition (reading $a_1, \ldots, a_k$) from $q_{i,IN}$ to $q_{j,[a_1,\ldots,a_k],a} =$ "$s_{j,a}$" (or q_{FAIL}) for any i, j. Since from $s_{j,a}$, it moves at least one of its k-heads (definition of $s_{j,a}$). Moreover, when in any state q_{IN} reading $a_1, \ldots, a_k$, the only possible transition is to some state $q_{j,[a_1,\ldots,a_k],a}$ or q_{FAIL} (by construction). Also, when in $q_{j,[a_1,\ldots,a_k],a}$, by definition then only possible transition is to some state q_{IN}. Thus, the length of computation on $\bar{A_m}$ is $O(n)$.

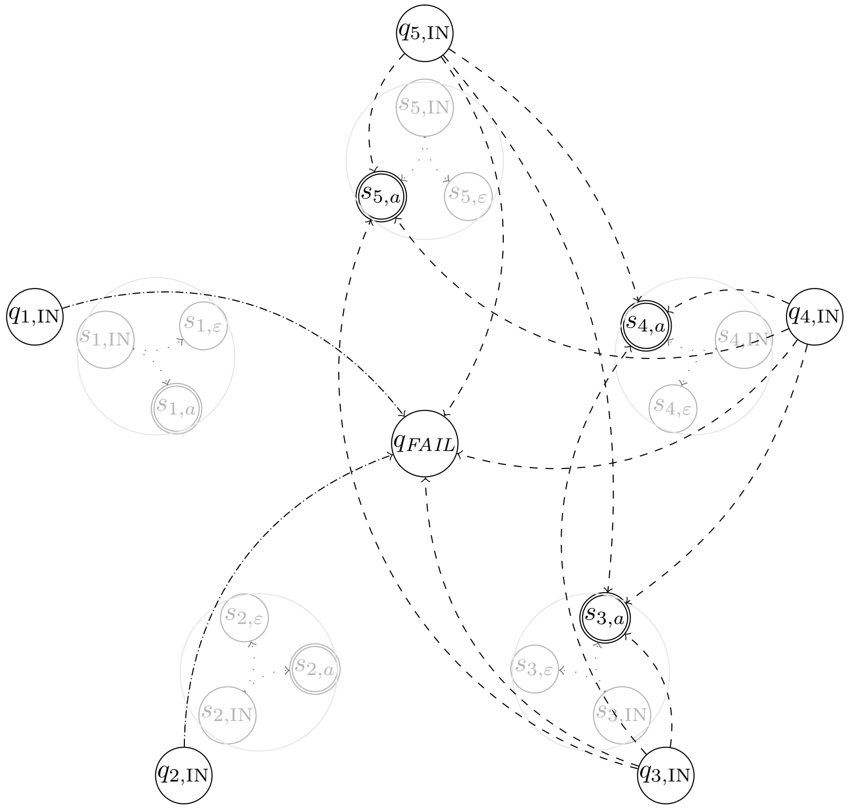

Figure 1.7: Construction of $\bar{A}$: combine multiple steps ($q_{IN} \to s_{IN} \to s_\varepsilon \to s_a$ to $q_{IN} \to s_a$)

Step 4 ($\bar{\bar{A}}$): The last step to construct the automaton $\bar{\bar{A}}$ (from $\bar{A_m}$) that moves at least one head each step, is a simple construction of replacing the two step process of moving from some q_{IN} to $q_{i,[a_1,\ldots,a_k],a}$ to $q_{j,IN}$ (or q_{FAIL}) by a single step process of moving from $q_{i,IN}$ to $q_{j,IN}$ (or q_{FAIL}). We just calculate the probability of each such transition sequence and create new transitions removing the old ones afterward. Which effectively makes all states in all layers $[a_1, \ldots, a_k]$ unused and unnecessary – hence we remove them.

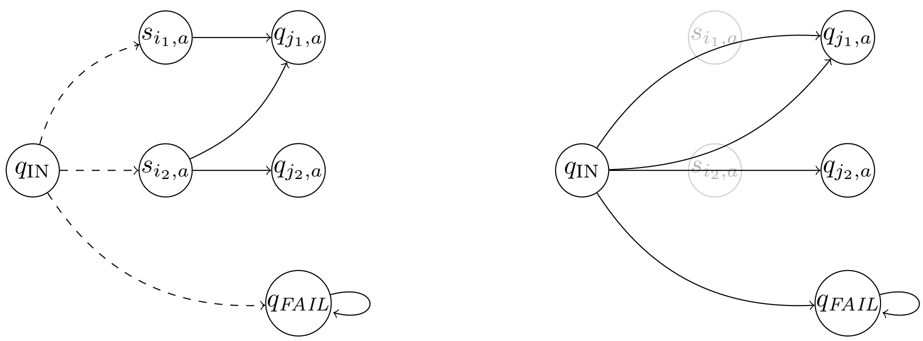

Figure 1.8: Collapsing transitions to one

We are then left with an automaton $A_1 := \bar{\bar{A}}$ with the same states as the original A_0 plus one new (q_{FAIL} in which every old infinite loop now returns *FAILURE*), such that each its allowed transition advances at least one head. $\square$

Theorem 1.6.4 (ε-free form is a normal form for *PFA*(k) with T_P)**.** *Let A_0 be a PFA(k) with allowed probabilities T_P. Then there exists a PFA(k) A_1 with allowed probabilities T_P in ε-free form equivalent with A_0, if there exists a field $(F, +, \cdot)$ such that $F \cap [0, 1] = T_P$.*

Proof. Since all the opreations we used when manipulating with the probabilities in the previous construciton were addition, multiplication and taking inverses (for $+, \cdot$), the newly-constructed probabilties are still elements of F. Moreover, since they are probabilities, they are in $[0, 1]$.
(The theory of Markov chains which we used computes the resulting probabilities via matrices whose construction also requires only the use of $+, \cdot$ and taking inverses.)

$\square$

Corollary 1.6.5 (ε-free form is a normal form for *PFA*(k) with rational T_P)**.** *Let A_0 be a PFA(k) with allowed probabilities $\mathbb{Q} \cap [0, 1]$. Then there exists a PFA(k) A_1 with allowed probabilities $\mathbb{Q} \cap [0, 1]$ in ε-free form equivalent with A_0.*

Proof. Apply the previous theorem for the field $(\mathbb{Q}, +, \cdot)$. $\square$

Corollary 1.6.6. *Let A_0 be a PFA(k) with allowed probabilities $T_P \subseteq \mathbb{Q}$. Then there is a PFA(k) A_1 with allowed probabilities $\mathbb{Q} \cap [0, 1]$ in ε-free form equivalent with A_0.*

Proof. Follows trivially from the previous corollary, since if $T_P \subseteq T'_P$, then each *PFA*(k) with allowed probabilities T_P is also a *PFA*(k) with allowed probabilities T'_P. $\square$

Remark. This corollary upgrades the main theroem (1.6.3), in the following way: Should we now additionally prove that the initial T_P is a subset of $\mathbb{Q}$, the *PFA*(k) in ε-free form that we construct will only use “rational” probabilities.

Chapter 2

Cutting-and-pasting technique

In this chapter, we illustrate a technique for proving that a language cannot be accepted by any k-head one-way finite automaton. We show this technique because we utilize it, with a certain modification, for proving few of our results.

Informally, the technique as shown in [YR78] consists of first defining a *location* (*type* in [YR78]) of a configuration, which essentially represents, on which word which head is, and a *pattern* of a computation, that is, a sequence of corresponding locations of configurations. It continues by analyzing the possible head movement, and showing that for each pattern there exist two integers $i \neq j$, such that the heads cannot be at the same time at both sub-words w_i and w_j.

The argument then, is the following. We show – by some counting argument – that there do exist two words $x \in L$ and $y \in L$ that have the same pattern and differ only in those sub-words, in which the heads are never simultaneously. The technique then finishes by constructing a new word $z \notin L$, where z is x with x_j substituted by y_j, and finding an accepting computation on z by cutting-and-pasting configurations of computations on x and y.

DFA(2) cannot accept L_{uvvu}

The following lemma is an example of this technique, cutting-and-pasting. Moreover, it is a twist on the proof for the more general Hierarchy Theorem.

Consider the following language (where $\# \notin \Sigma$)

$$L_{uvvu} = \{u\#v\#v'\#u' \mid u = u', v = v' \in \Sigma^*\}$$

Lemma 2.0.1. *Language L_{uvvu} cannot be accepted by any one-way 2-head deterministic automaton. ($L_{uvvu} \notin \mathscr{L}(DFA(2))$)*

Proof. Let us begin with a simple observation of head movement. During any one computation, heads can visit simultaneously only either u and u', or v and v', never both. (Since in order to visit simultaneously u and u', one head may not leave u, and the other has to – irreversibly – travel over $v\#v'$ to get to u'. Therefore, afterwards that head cannot visit v nor v' again. Analogously, in order to visit v and v' simultaneously, both heads have to leave u, and no head can visit u' (it is beyond v, v')).

Define $L_n = \{w_1\#w_2\#w_2'\#w_1' \mid w_1{=}w_1', w_2{=}w_2' \in \Sigma^n\}$, obviously $L_n \subseteq L_{uvvu}$. We will show that for any *DFA*(2), that accepts every word in L_n, then it must also accept a word $\notin L_{uvvu}$. Let *DFA*(2) A be the automaton that accepts each word in L_n.

Define a *location* of a configuration (q, p_1, p_2) as a 2-tuple $(\lceil p_1/(n{+}1)\rceil, \lceil p_2/(n{+}1)\rceil)$. Then, for a computation $c_1(w), c_2(w), \ldots, c_{l_w}(w)$ define a *pattern* of a word as a subsequence of computation $d_1(w), d_2(w), \ldots, d_{l_w'}(w)$ obtained by taking $c_1(w)$ and all subsequent $c_i(w)$ such that $location(c_i(w)) \neq location(c_{i+1}(w))$.

Let k be the number of heads ($k{=}2$). Since the length of a pattern $l_w' \leq k(4+1)$ (each head must go through all 4 sub-words and \$), the number of possible patterns ρ is at most $(|Q| \cdot (4(n+1))^k)^{k(4+1)}$ [1] We then divide words in L_n into ρ sets, depending on the word's patterns. By the pigeonhole principle we know that one of those sets contains at least $2^{2n}/\rho$ words. Let S_0 be that set.

As a corollary of our observation, note that for each pattern $\mathcal{P}$, there exists an index $i_{\mathcal{P}}$, such that during computation, heads are never on $w_{i_{\mathcal{P}}}$ and $w_{i_{\mathcal{P}}}'$ in the same configuration. Let j be the index of the pair of words which are never visited simultaneously in computations on words in S_0. Now, we partition the words in S_0 ($w_1\#w_2\#w_2'\#w_1'$) into classes according to the string "w_i, w_i'", where $i{\neq}j$ (We group words that differ from each other only in w_j, w_j'). Since there are 2^n such strings[2], by Dirichlet's box principle (pigeonhole principle) we see that there exists a class with at least $(2^{2n}/\rho)/2^n = 2^n/\rho$ words. Let S_1 be the set of words from that class, and assume n is large enough so that $|S_1| \geq 2$ (we can make that assumption, since ρ is at most polynomial in n).

Now follows the main "cutting and pasting" argument:
Since $|S_1| \geq 2$, the set contains at least two different words: $x = x_1\#x_2\#x_2'\#x_1'$ and $y = y_1\#y_2\#y_2'\#y_1'$. By cutting-and-pasting configurations, we construct an accepting computation on word $z \notin L_{uvvu}$, $z = z_1\#z_2\#z_2'\#z_1'$, where $(\forall i{\neq}j) z_i = x_i, z_i' = x_i'$ and $z_j = x_j, z_j' = y_j'$. (we only replace x_j' by y_j' – the sub-word that is not checked).

By construction ($x, y \in S_0$), both computations $c_1(x), \ldots$ and $c_1(y), \ldots$ contain the pattern $d_1(x), \ldots$ (of length l) as a subsequence. Therefore we divide the sequences $c_1(x), \ldots$ and $c_1(y), \ldots$ into l blocks, each by beginning a new block with each occurrence of an element d_i, as in the following figure (i.e., we group together configurations with the same *location*).

[1] $(\#_{configurations})^{k(\#_{subwords}+1)}$

[2] since $w_1\#w_2\#w_2'\#w_1' \in S_0$, and $S_0 \subseteq L_n$, we see that $w_i{=}w_i'$

$$x:\ \underbrace{c_1(x)..}_{d_1(x)}\ \underbrace{c_{i_2}(x)...}_{d_2(x)}\ \underbrace{c_{i_3}(x)...}_{d_3(x)}\ \ldots\ \underbrace{c_{i_{IN_1}}(x)......}_{d_{IN_1}(x)}\ \ldots\ \underbrace{c_{i_{OUT_1}}(x).....}_{d_{OUT_1}(x)}\ \ldots\ \underbrace{c_{i_l}(x)...}_{d_l(x)}$$

$$y:\ \overbrace{c_1(y)...}\ \overbrace{c_{j_2}(y)....}\ \overbrace{c_{j_3}(y)......}\ \ldots\ \overbrace{c_{j_{IN_1}}(y)..}\ \ldots\ \overbrace{c_{j_{OUT_1}}(y)..}\ \ldots\ \overbrace{c_{j_l}(y)......}$$

* $d_{IN_i}(x)/d_{OUT_i}(x)$ denotes the first configuration where i-th head entered/left w'_j

Figure 2.1: Division of sequences $c_1(x)\ldots, c_1(y)\ldots$ into blocks on L_{uvvu}.

Remark. In the above figure, various numbers of dots convey that the blocks may be of different length – only the pattern is the same. Moreover, we are not showing $d_{IN_2}(x)$, $d_{OUT_2}(x)$ for clarity.

We construct an accepting computation for A on z by selecting successive blocks from $\{c_i(x)\}$, except when A during that block would be reading $x'_j(\neq z'_j)$, in which case we select the corresponding block from $\{c_i(y)\}$ instead (since $y'_j = z'_j$). This sequence forms a valid computation for z since the last configuration in block i for either $\{c_i(x)\}$ or $\{c_i(y)\}$ yields $d_{i+1}(x)$ as the next configuration of A, and by construction, A is never reading sub-words z_j and z'_j simultaneously. Therefore, at any instant, A behaves exactly as it would if the input had been one of x or y. □

$$z:\ \underbrace{c_1(x)..\ \ldots}_{x}\ \overbrace{c_{i_{IN_1}}(y)..\ \ldots}^{y}\ \underbrace{c_{i_{OUT_1}}(x)....\ \ldots}_{x}\ \overbrace{c_{i_{IN_2}}(y)..\ \ldots}^{y}\ \underbrace{c_{i_{OUT_2}}(x)...\ \ldots\ c_{i_l}(x)...}_{x}$$

* $c_{IN_i}(x)/c_{OUT_i}(x)$ denotes the first configuration where i-th head entered/left w'_j

Figure 2.2: construction of an accepting computation for $z \notin L_{uvvu}$

In order to be able to reuse the "cutting and pasting" argument in subsequent proofs we state it in a lemma. However, in order to generalize it, we define the following: Let $L^n_b = \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^n) \wedge (w_i = w_{2b+1-i}) \text{ for } 1 \leq i \leq 2b\}$. Let *pattern* be defined identically as above. However, because the automaton may have more than two heads (k), the *location* based on which pattern is defined, will have to scale up to a k-tuple $(\lceil p_1/(n+1)\rceil, \ldots, \lceil p_k/(n+1)\rceil)$.

The new definition is a *mistake* of a *pattern*, defined as a *location* which could be valid, but does not occur in the pattern (i.e., the indices of sub-words, in which the heads are never simultaneously). Moreover, a *mistake* i is a mistake $(i, 2b-i+1)$. It is a useful/notable mistake informing us that during the computation, the heads are never simultaneously in w_i and w_{2b-i+1}.

Lemma 2.0.2. *If words $x, y \in L_b^n, x \neq y$ have the same pattern on a DFA(k) A, and if the pattern has a mistake i, then we can construct an accepting computation on A on a input word $z \notin L_b^n$, constructed by replacing x_{2b-i+1} by y_{2b-i+1} on x.*

Proof. The proof is identical to the argumentation from the "cutting and pasting" argument in the previous proof, except x and y, have the following format for L_b^n. $x_1 * \cdots * x_b * x'_b * \cdots * x'_1$ (this way, by the identical construction (replace x'_i by y'_i), we get the correct z for the rest of the argumentation). □

Remark. A high-level overview of what we did, may be the following: Define a pattern in such a way, so that one can define a mistake, such that it is feasible to prove that on each deterministic computation, some useful mistake must occur. Then, if two words x, y have the same pattern (which has a mistake), we are able to construct z from x, y such that we can construct a valid accepting computation from computations of x, y, by exploiting the mistake by cutting-and-pasting the "blocks" based on the pattern.

2.1 The Hierarchy Theorem

Yao and Rivest, in their paper [YR78], prove that there are languages which can be recognized by a deterministic $(k+1)$-headed one-way finite automaton, yet cannot by any k-headed one-way finite automaton. They also show few notable corollaries of the Hierarchy Theorem.

Consider these languages :

$$\begin{aligned} L_b &= \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^*) \wedge (w_i = w_{2b+1-i}) \text{ for } 1 \leq i \leq 2b\} \\ L' &= \{w_1 * w_2 * \cdots * w_{2b} \mid (b \geq 1) \wedge (w_i \in \{0,1\}^* \text{ for } 1 \leq i \leq 2b) \wedge (\exists i)(w_i \neq w_{2b+1-i})\} \end{aligned} \tag{2.1}$$

Theorem 2.1.1 (The Hierarchy Theorem [YR78])**.** *For each integer $k \geq 2$*

- *The language L_b is recognizable by a NFA(k) if and only if $b \leq \binom{k}{2}$.*
- *The language L_b is recognizable by a DFA(k) if and only if $b \leq \binom{k}{2}$.*

Corollary 2.1.2 ([YR78])**.** *For every $k \geq 1$, there is a language M_k recognized by a NFA(2) but by no DFA(k).*

Corollary 2.1.3 ([YR78])**.** *The language L' is recognizable by a NFA(3) but by no DFA(k).*

Infromal proof for 2.1.1. It can be shown that if $b > \binom{k}{2}$, then for any computation of one-way finite automaton on an $x \in L_b$ there exists an index i such that w_i and w_{2b+1-i} are never read simultaneously. The rest of the proof is similar to the proof for Lemma 2.0.1, with added complexity since we need to account for k heads. □

Chapter 3

Monte Carlo

In this chapter we explore one-way multi-head Monte-Carlo probabilistic finite automata with one-sided error with allowed rational probabilities ($T_P = \mathbb{Q} \cap [0,1]$). For brevity, in this chapter, whenever we refer to a probabilistic finite automata or *PFA*(k), we mean a Monte-Carlo *PFA*(k) (recall definition: $(\forall x \in \Sigma^*) : p_A^{FAIL}(x) = 0$).

Remark. Acceptance with error is defined only for Monte-Carlo *PFA*. Also, since $(\forall x \in \Sigma^*) : p_A^{FAIL}(x) = 0$, we can freely assume that $Q_{rej} = Q - Q_{acc}$, since if there was a state in $Q - (Q_{acc} \cup Q_{rej})$, no computation ever ended in it, thus nothing will change if we add it to rejecting states.

Lemma 3.0.1 (Unbounded $\supseteq$ Bounded)**.** *Let L be a language recognized by a certain PFA(k) A with bounded two-sided (resp. one-sided true-biased, one-sided false-biased) error. Then the language L is also recognized by A with unbounded two-sided (resp. one-sided true-biased, one-sided false-biased) error.*

Proof. Follows trivially from definition. □

Lemma 3.0.2 (Unbounded Two-sided $\supseteq$ Unbounded True-biased)**.** *Let L be a language recognized by a certain PFA(k) A with unbounded one-sided true-biased error. Then the language L is also recognized by A with unbounded two-sided error.*

Proof. $L = L(A, \lambda)$, for $\lambda = 0$ □

Lemma 3.0.3 (Bounded Two-sided $\supseteq$ Bounded One-sided)**.** *Let L be a language recognized by a certain PFA(k) A with bounded one-sided error. Then the language L is also recognized by A with bounded two-sided error.*

Proof. Proof is just showing that you can calculate the cut point λ and its error bound Δ, from the error bound Λ, such that it satisfies the definition of bounded two-sided error. (Which we have already done in the definitions 1.3). □

Lemma 3.0.4. *Let L be a language recognized by a certain PFA(k) A, with bounded one-sided error with error bound Λ_0. Then L is recognized by A with bounded one-sided error with all error bounds $\Lambda \geq \Lambda_0$.*

Proof. Is true by definition. □

Lemma 3.0.5 (Bounded One-sided $\supseteq$ Deterministic)**.** *Let L be a language recognized by a certain DFA(k) A, then we can construct a PFA(k) A' recognizing L with bounded true-biased error and bounded false-biased error, both with error bound Λ=0.*

Proof. Construction is the following, for every transition in δ_A, we set such transition's probability to 1 in δ'_A. The rest, we copy. The *PFA*(k) A' constructed this way behaves exactly the same as the *DFA*(k) A (deterministically, accepting each word with probability 0 or 1) Therefore, just looking at the definitions of the types of acceptance, we observe that such algorithm satisfies both definitions (with $\Lambda = 0$). □

Corollary 3.0.6 (*PFA*(k) $\supseteq$ *DFA*(k))**.** *Let L be a language recognized by a DFA(k) A. Then we can construct a PFA(k) A' recognizing L with bounded and unbounded two-sided, one-sided true-biased and one-sided false-biased error, for each (isolated) cut-point λ (with its bound Δ), and for each error bound Λ.*

Proof. Combine the previous lemmas (3.0.5, 3.0.4, 3.0.3, 3.0.1). The construction of the corresponding *PFA*(k) in the above lemma also satisfies all other definitions. □

Lemma 3.0.7 (The Complement Lemma)**.** *For language L recognized by a*

- *PFA(k) A with true-biased error with error bound Λ, there is a PFA(k) A' recognizing L^c with false-biased error with error bound Λ.*
- *PFA(k) A with false-biased error with error bound Λ, there is a PFA(k) A' recognizing L^c with true-biased error with error bound Λ.*
- *PFA(k) A with unbounded true-biased error, we can construct a PFA(k) A' recognizing L^c with unbounded false-biased error.*
- *PFA(k) A with unbounded false-biased error, we can construct a PFA(k) A' recognizing L^c with unbounded true-biased error.*

Proof. The same arguments used in the proof for the complement lemmas for *PFA*'s, (1.3.7, 1.3.8) will also work here (only swapping accepting and rejecting states). We need not worry about infinite computations, i.e., computations that get stuck in an infinite loop, because such computation is inconclusive, and we know that $p_A^{FAIL}(x) = 0$ for all $x \in \Sigma^k$. To prove the second pair of points unbounded, we will follow the Remark 1.3.9. □

NFA and (un)bounded one-sided PFA(k)

The following theorem shows the power of probabilistic computation with unbounded one-sided error. Namely, that it is more powerful then non-deterministic computation, since the true-biased flavour of unbounded error can simulate non-determinism, and the false-biased flavour can recognize complements of languages accepted by non-determinism.

Theorem 3.0.8. *The following statements are equivalent:*

i. Language L can be recognized by a NFA(k),

ii. Language L can be recognized by a PFA(k) with unbounded true-biased error.

iii. Language L^c can be recognized by a PFA(k) with unbounded false-biased error.

Proof. We prove the three implications *i.* $\Rightarrow$ *ii.*, *ii.* $\Rightarrow$ *iii.* and *iii.* $\Rightarrow$ *i.* separately.

(*i.* $\Rightarrow$ *ii.*) For a language L, recognized by some *NFA*(k), we take the *NFA*(k) A recognizing L such that A is in a normal form where it has to move at least one head each transition [1]. We create the corresponding *PFA*(k) A', by assigning probabilities to transitions which involve non-determinism (where we pick between multiple paths). Formally, for every state $q \in Q_A$, and symbols $a_1, \ldots, a_k \in \Sigma_\$$, where $\delta_A(q, a_1, \ldots, a_k) = \{(p_1, d_{11}, \ldots, d_{k1}), \ldots, (p_m, d_{1m}, \ldots, d_{km})\}$, we define the function $\delta_{A'}(q, a_1, \ldots, a_k, p_i, d_{1i}, \ldots, d_{ki}) = \frac{1}{m}$, for each $i \in \{1 \ldots m\}$.

The correctness of our construction follows from the following: For each word in L, there exists a computation on A that is accepting, therefore our newly constructed *PFA*(k) A' will, with non-zero probability, run that computation. Hence, the word is accepted with nonzero probability. On the other hand, for each word not in L, there existed no accepting computation on A, and we – by assigning uniform probabilities to transitions – have only weighted the possible computations with probability, we have not added, nor removed any possible transitions between states. Hence, such word ($\notin L$) will be accepted with probability equal to zero. Thus satisfying the definition of a language accepted with a *true-biased unbounded error.*

(*ii.* $\Rightarrow$ *iii.*) We have a *PFA*(k) A, recognizing L with *unbounded true-biased error.* Then, using the Complement lemma (3.0.7), we construct a *PFA*(k) A', accepting the complement of L (L^c) with *unbounded false-biased error.*

(*iii.* $\Rightarrow$ *i.*) From *PFA*(k) A accepting language L^c with *unbounded false-biased error*, we, using the Complement lemma (3.0.7), construct a *PFA*(k) A', accepting the complement of L^c ($(L^c)^c = L$) with *unbounded true-biased error.*

[1] A possible proof for this normal form is analogous to proof of ε-free normal form for *PFA*(k) (1.6.3), except we need not bother with computing probabilities (and using Markov chains).

Then, we create the corresponding *NFA*(k) A'' by replacing every probabilistic choice of A' by a non-deterministic one. More rigorously, for every state $q \in Q_{A'}$, and $a_1, \ldots, a_k \in \Sigma_\$$, where $\delta_{A'}(q, a_1, \ldots, a_k, p, d_1, \ldots, d_k) \neq 0$, we insert $(p, d_1, \ldots, d_k)$ into $\delta_{A''}(q, a_1, \ldots, a_k)$.

This construction is correct, because, for a language to be accepted by a *PFA*(k) with a *true-biased unbounded error*, the following must hold: for each word in L, there must exist an accepting computation (the word must be accepted with non-zero probability), and, for each word not in L, there must exist no accepting computation with probability $\neq 0$ (it must be accepted with probability zero).

Since we inserted into the non-deterministic δ-function only transitions with nonzero probability, the non-determinism will, for each word w, find one accepting computation if and only if $w \in L$. □

Corollary 3.0.9 (Bounded true-biased $\subseteq$ *NFA*(k))**.** *Every language recognized by a PFA*(k) *with bounded true-biased error can also be recognized by a NFA*(k)*.*

Proof. Follows from the fact that every language recognized by a *PFA*(k) with *bounded true-biased error* can be recognized *PFA*(k) with *unbounded true-biased error* (Lemma 3.0.1), and the previous theorem. □

3.1 Hierarchy for one-sided error

Recalling the Hierarchy Theorem that Yao and Rivest [YR78] proved, we may ask ourselves, if we can get similar results for the probabilistic multi-head automata, in this case/chapter for ones with one-sided error. The answer is yes. Consider this language (2.1):

$$L_b = \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^*) \wedge (w_i = w_{2b+1-i}) \text{ for } 1 \leq i \leq 2b\}$$

Theorem 3.1.1 (Hierarchy for one-sided error *PFA*(k))**.** *For each integer* $k \geq 2$

- *The language* L_b *is recognizable by a PFA*(k) *with bounded true-biased error if and only if* $b \leq \binom{k}{2}$*.*
- *The language* $(L_b)^c$ *is recognizable by a PFA*(k) *with bounded false-biased error if and only if* $b \leq \binom{k}{2}$*.*

Proof. Both languages L_b and $(L_b)^c$ are recognizable by a *PFA*(k) with both *bounded true-biased* and *false-biased error* when $b \leq \binom{k}{2}$, because of the same argument as used in Yao and Rivest [YR78], since they proved that it can be recognized by a *DFA*(k). Hence we only need Lemma 3.0.5.

L_b is not recognizable by any *PFA*(k) with *bounded true-biased error* when $b > \binom{k}{2}$, because if such *PFA*(k) existed (accepting any one specific L_b), by Corollary 3.0.9, we could construct a *NFA*(k), recognising L_b. Which would contradict the Hierarchy Theorem (2.1.1).

In order to prove that $(L_b)^c$ is not recognized by any *PFA*(k) with *bounded false-biased error* when $b > \binom{k}{2}$, for the purposes of contradiction, assume that is (for some specific $b > \binom{k}{2}$). Then, by the Complement lemma 1.3.8 we can construct a *PFA*(k) accepting L_b with *bounded true-biased error* which is in direct contradiction with the previous point, that we have already proven. □

Remark. We have actually proven, that for each k, there exists a language M_k (M'_k) that is recognized by a *PFA*$(k+1)$ with *bounded true-biased error* (*bounded false-biased error*) and not by any one *PFA*(k) with *bounded true-biased error* (*bounded false-biased error*). Hence the name, "Hierarchy for one-sided error *PFA*(k)".

Theorem 3.1.2. *For each integer $k \geq 2$*

- *The language L_b is recognizable by a PFA(k) with unbounded true-biased error if and only if $b \leq \binom{k}{2}$.*
- *The language $(L_b)^c$ is recognizable by a PFA(k) with unbounded false-biased error if and only if $b \leq \binom{k}{2}$.*

Proof. Analogous to the proof for 3.1.1 (use lemmas for unbounded error). □

Gaps in number of heads required

Analyzing the Hierarchy Theorem for one-sided error, we note that the language L_b ($(L_b)^c$) can be accepted with a probabilistic automaton with "the opposite" one-sided error, and with only two heads, showing that there is a gap between the number of heads required to accept a certain sequence of languages with true-biased error and false-biased error.

Lemma 3.1.3 (Head Gap)**.** *For each integer $b \geq 1$,*

- *the language L_b, can be recognized by a PFA(2) with bounded false-biased error.*
- *the language $(L_b)^c$, can be recognized by a PFA(2) with bounded true-biased error.*

Proof. The language L_b can be recognized by *PFA*(2) with bounded false-biased error, by the following algorithm:

0. "Check format"

   In tandem with the main algorithm, with head 2, check that w is of the format $(\{0,1\}^* *)^{2b-1}\{0,1\}^*$ (its only a regular check – count $*$).

1. "Pick sub-word"
   Throw b-sided die $(=: i)$, then move head 1 to word w_i, and head 2 to w_{2b+1-i}.
   (With probability $\frac{1}{b}$, arrive at/pick w_m (for $m \in \{1, \dots, b\}$)).

2. "Verify equality"
   Accept iff the words under heads 1 and 2 are equal.
   (Move heads 1,2 simultaneously while they read the same symbols, until #).

Each word w in L_b, A accepts with probability 1, since any two corresponding sub-words of it are equal, therefore any computation will always be accepting. For word w not in L_b, the word is either not of the format $(\{0,1\}^* *)^{2b-1}\{0,1\}^*$ – which we detect with a regular check – or there exists an index i, such that $w_i \neq w_{2b-i+1}$. Analyzing our algorithm, A rejects such word with probability $\geq \frac{1}{b}$, since we arrive at sub-word i with probability $\frac{1}{b}$, and then verify the (in)equality deterministically. Hence the algorithm accepts words not in L_b with probability $1-\frac{1}{b}$. Thus satisfying the definition of accepting with *bounded false-biased error* with *error bound* $\Lambda = \frac{b-1}{b}$.

In order to prove the second point, intuitively, we do an analogous algorithm, this time verifying inequality at random sub-word, and/or checking for the absence of valid format. Formally, by the Complement lemma (3.0.7), we construct a *PFA*(2) accepting $(L_b)^c$ with bounded true-biased error. □

PFA(k) and DFA(k)

Recall the Corollary (2.1.2) that Yao and Rivest [YR78] observed as a consequence of the Hierarchy Theorem 2.1.1:

For every $k \geq 1$, there is a language M_k recognized by a NFA(2) but by no DFA(k).

We might want an analogous corollary to the above, for *PFA*(2) with one-sided error instead of *NFA*(2), as a corollary to the Head Gap lemma (3.1.3). The following corollary can also serve as an indication to the reason why we refer to the previous lemma as the "Head Gap lemma".

Corollary 3.1.4. *For every $k \geq 1$, there*

- *is a language M_k recognized by a PFA(2) with bounded false-biased error but by no NFA(k) nor any DFA(k).*
- *is a language M'_k recognized by a PFA(2) with bounded true-biased error but by no DFA(k).*

Proof. Let $M_k = L_b$ where $b = \binom{k}{2} + 1$. No *NFA*(k) nor *DFA*(k) can accept M_k, by the Hierarchy Theorem (2.1.1). However, by the Head Gap lemma (3.1.3), *PFA*(2) with bounded false-biased error can recognize M_k.

We prove the second point by letting $M'_k = (L_b)^c$ for $b = \binom{k}{2} + 1$. No *DFA*(k) can accept M'_k, because if it did, since *DFA*(k)'s are closed under complement, a *DFA*(k) would recognize $((L_b)^c)^c = L_b$, which would contradict the Hierarchy Theorem. Yet, by the Head Gap lemma, *PFA*(2) with bounded true-biased error can recognize M'_k. □

Remark. Analogous corollary is also true for *PFA*(2) with unbounded error.

In a sense, we have proven that k-head deterministic finite automata cannot simulate their k-head probabilistic counterparts with any one-sided error. Since this is a common question, whether or not a variation of a model can recognize a bigger class of languages, we state it in the following corollary.

Corollary 3.1.4.1 (*DFA*$(k) \subsetneq$ *PFA*(k) one-sided)**.** *For each $k \geq 1$, there*

- *exists a language M_k that is recognizable by PFA(k) with bounded false-biased error, but by no DFA(k),*
- *exists a language M'_k that is recognizable by PFA(k) with bounded true-biased error, but by no DFA(k).*

Proof. Follows trivially from the previous corollary. □

Moreover, since we have shown that *PFA*(k) can save heads on certain languages, a valid question might be if *PFA* can save heads in general. The answer is no – the classes of languages recognized by *PFA*(k) with (un)bounded one-sided error and *DFA*$(k+1)$'s are incomparable (one direction of this incomparability is in the Corollary 3.1.4).

Corollary 3.1.4.2 (*DFA*$(k+1) \nsubseteq$ *PFA*(k) one-sided)**.** *For each $k \geq 1$,*

- *there exists a language M_k that is recognizable by DFA$(k+1)$, but by no PFA(k) with unbounded true-biased error,*
- *there exists a language M'_k that is recognizable by DFA$(k+1)$, but by no PFA(k) with unbounded false-biased error.*

Proof. Consider $M_k = L_b$, for $b = \binom{k}{2}+1$. By the Hierarchy Theorem (2.1.1), no *NFA*(k) can accept it, hence no true-biased *PFA*(k) with unbounded error (Theorem 3.0.8) . The reason why L_b is recognizable by a *DFA*$(k+1)$ follows from the Hierarchy Theorem.

$M'_k = (L_b)^c$. Consider L_b, for $b = \binom{k}{2}+1$ again. Looking at previous corollary, no *true-biased PFA*(k) with *unbounded error* can accept it. Therefore, if a *false-biased PFA*(k) with *unbounded error* recognized $(L_b)^c$, it would lead to contradiction (by the Complement lemma 3.0.7). The reason why $(L_b)^c$ can be recognized by a *DFA*$(k+1)$ follows from the Hierarchy Theorem, since $\mathscr{L}(DFA(k+1))$ is closed under complement.
□

True-biased vs false-biased error

Now, we put in contrast the true-biased and false-biased error Monte-Carlo computations. We look at whether or not it is possible to construct at least a *PFA*(k) with unbounded true-biased error, for each language recognized by their bounded counterpart with more heads (*PFA*(k+1)). Looking at the Corollary 3.1.4, we create an analogous corollary, this time about the gap in the number of heads required to accept a certain languages by a *PFA*(k) with different one-sided errors.

Lemma 3.1.5. *For all* $2{\leq}k_1, 2{\leq}k$,

1. *there is a language* M_k*, recognized by* *PFA*(k_1) *with bounded false-biased error, that cannot be recognized by any* *PFA*(k) *with unbounded true-biased error.*
2. *there is a language* M'_k*, recognized by* *PFA*(k_1) *with bounded true-biased error, that cannot be recognized by any* *PFA*(k) *with unbounded false-biased error.*

Proof. By the Hierarchy Theorem for one-sided error 3.1.1, we know that the language L_b for $b{=}\binom{k}{2}{+}1$ ($>$ $\binom{k}{2}$), is not recognizable by *PFA*(k) with unbounded true-biased error. However, by the Head Gap lemma (3.1.3), it is recognizable by a *PFA*(2) with bounded false-biased error. Hence trivially recognizable by a *PFA*(k_1) with number of heads $k_1 \geq 2$.

To prove the second point, by the Hierarchy Theorem, $(L_b)^c$ for $b{=}\binom{k}{2}{+}1$, cannot be accepted by a *PFA*(k) with unbounded false-biased error. Yet, by the Head Gap lemma, it can be recognized by a *PFA*(2) with bounded true-biased error (thus unbounded with $k_1 \geq 2$ heads). □

We have essentially proven that the classes of languages recognized by *PFA*(k) with (un)bounded true-biased error and *PFA*(k) with (un)bounded false-biased error are incomparable:

Corollary 3.1.6. *For all* $2{\leq}k_1, 2{\leq}k_2$*, the classes of languages recognized by*

1. *PFA*(k_1) *with bounded false-biased error, and* *PFA*(k_2) *with bounded true-biased error are incomparable,*
2. *PFA*(k_1) *with bounded false-biased error, and* *PFA*(k_2) *with unbounded true-biased error are incomparable,*
3. *PFA*(k_1) *with bounded true-biased error, and* *PFA*(k_2) *with unbounded false-biased error are incomparable.*
4. *PFA*(k_1) *with unbounded false-biased error, and* *PFA*(k_2) *with unbounded true-biased error are incomparable.*

Proof. Follows from the previous lemma, since any language recognized by *PFA*(k) with bounded one-sided error, can also be recognized with unbounded error (Lemma 3.0.1).

Also, any language that cannot be recognized by no *PFA*(k) with unbounded one-sided error, also cannot recognized by any *PFA*(k) with bounded one-sided error. □

Power of PFA(k) with unbounded error

The second corollary that Yao and Rivest [YR78] have stated, can be extend for probabilistic automata with unbounded one-sided error. Consider this language (see 2.1):

$$L' = \{w_1 * w_2 * \cdots * w_{2b} \mid (b \geq 1) \land (w_i \in \{0,1\}^* \text{ for } 1 \leq i \leq 2b) \land (\exists i)(w_i \neq w_{2b+1-i})\}$$

Whether or not it can be extended by a multi-head probabilistic automaton with bounded one-sided error is nontrivial, and maybe not possible. Since the language L' contains arbitrary many sub-words, and by randomly picking one to check the probability of guessing the "correct" one is arbitrarily low.

Corollary 3.1.7. *For each $k \geq 1$,*

- *the language L' is recognizable by a PFA(3) with unbounded true-biased error but by no DFA(k).*
- *the language $(L')^c$ is recognizable by a PFA(3) with unbounded false-biased error but by no NFA(k) nor any DFA(k).*

Proof. To prove the first point, the argument is the following. For each integer k, no *DFA*(k) can recognize L', because if it did, for some k', it could recognize its complement $(L')^c$ (since *DFA*(k)'s are closed under complement) with a fixed number of heads k'. We could therefore easily construct a *DFA*(k') accepting L_b, $b = \binom{k'}{2} + 1$, because $L_b = (L')^c \cap (\{0,1\}^* *)^{2b-1}\{0,1\}^*$ (since $\mathscr{L}(DFA(k'))$ is closed under intersection with regular languages) which is a direct contradiction with the Hierarchy Theorem (2.1.1).

From the second corollary (2.1.3), we know that L' can be recognized by a *NFA*(3). Therefore, by the Theorem 3.0.8 (unbounded true-biased is same as non-deterministic), a *PFA*(3) with *unbounded true-biased error* can accept L' with a simple imitation of non-determinism.

Proving the second corollary, the argument is analogous to our proof of the first point. For each integer k, no *NFA*(k) can recognize $(L')^c$, because if it did, for some k', we could again easily construct a *NFA*(k') accepting L_b, $b = \binom{k'}{2} + 1$, because $L_b = (L')^c \cap (\{0,1\}^* *)^{2b-1}\{0,1\}^*$ (since $\mathscr{L}(NFA(k'))$ is closed under intersection with regular languages) which is a direct contradiction with the Hierarchy Theorem (2.1.1).

We could, using the Theorem 3.0.8 (unbounded true-biased is non-deterministic), construct *PFA*(3) with *unbounded false-biased error* accepting $(L')^c$. However we choose to write an algorithm for the *PFA*(3) A accepting $(L')^c$ with *unbounded false-biased error*, since this way, we have an example of a automaton accepting with unbounded error.

0. "Verify format"
   With head 3, run a regular check for the format in tandem with the main algorithm. $((\{0,1\}^{*}*)^{2l-1}\{0,1\}^{*}$ for some l)
1. "Pick sub-word"
   Flip a fair coin, until you throw Heads. Every Tails, advance heads 1,3 by one. With probability $\frac{1}{2^{m+1}}$, arrive at sub-word w_m (throw Tails m times, Heads once).
2. "Find the corresponding sub-word"
   Move heads 2 and 3 simultaneously, where if a head reads * (or \$), it waits for the other head to arrive to its next * (or \$) until head 3 reaches end-marker \$. (We can assume that the number of sub-words is even, since we verify that.) Thus, with head 2 we arrive at sub-word w_{2b-m+1}.
3. "Verify equality"
   Accept iff the words under heads 1 and 2 are equal.

Each word w in $(L')^c$, A accepts with probability 1, since any two sub-words are equal, therefore any computation is always accepting.

For word w not in $(L')^c$, the word is either not of the format $(\{0,1\}^{*}*)^{2l-1}\{0,1\}^{*}$ for some l, which we can detect with a regular check, or, there exists an index i, such that $w_i \neq w_{2l-i+1}$. Analyzing our algorithm, we see that A rejects such word with probability $\geq \frac{1}{2^{i+1}} > 0$ (arrive at sub-word i, then verify the (in)equality) [2] . It thus accepts the word with probability $\leq \frac{2^{i+1}-1}{2^{i+1}} < 1$. Hence satisfying the definition of accepting a language with unbounded false-biased error. □

This corollary strengthens the previous claim for acceptance with unbounded error, by presenting a language that no (non-)deterministic automaton can accept, with any fixed number of heads, yet probabilistic automata can, with unbounded one-sided error. This only illustrates the strength of probabilistic computations with unbounded error.

3.2 Summary: comparison of k-head models

The following tables illustrate the corollaries that we have stated.

The first table summarizes relations between classes of languages recognized by *PFA*(k) with various one-sided errors. Relations between them and classes recognized by *DFA*(k) are the result of the Corollary 3.1.4, and an observation of The Hierarchy theorems. Relations between classes recognized by *PFA*(k)'s with various errors follow from the Corollary 3.1.6. The rest follow from the equivalence between unbounded *PFA*(k)'s and *NFA*(k)'s (3.0.8) and some trivial observations from the beginning of this chapter (3).

[2] Or arrive at the corresponding word w_{2l+1-i} (happens with nonzero probability). Hence the "$\geq$".

		PFA(k) with bounded true-biased	$\subseteq$	*PFA*(k) with unbounded true-biased	$=$ *NFA*(k)	
	$\subsetneq$				$\not\supseteq$ $\not\subseteq$	
DFA(k)		$\not\supset$ $\not\subset$	$\not\supseteq$ $\not\subseteq$ $\not\supseteq$ $\not\subseteq$	$\not\supset$ $\not\subset$		*DFA*(k+1)
	$\subsetneq$	*PFA*(k) with bounded false-biased	$\subseteq$	*PFA*(k) with unbounded false-biased	$\not\supseteq$ $\not\subseteq$	

Table 3.1: Relations between classes of languages recognized with k-heads

The second table illustrates the relations between the classes of languages recognized by *PFA*(k) and *PFA*($k+1$) with (un)bounded *true-biased* error, and the (non-)deterministic finite automata. The relations hold based on the the Hierarchy theorems (2.1.1, 3.1.1), the previous corollary (3.1.6), the fact that even unbounded probabilistic automata with one-sided error cannot simulate deterministic with more heads (3.1.4.2), and the obvious lemmas in section at the beginning of this chapter (3).

DFA(k)	$\subsetneq$	*PFA*(k) with bounded true-biased	$\subseteq$	*PFA*(k) with unbounded true-biased	$=$ *NFA*(k)
$\subsetneq$	$\not\supseteq$ $\not\subseteq$	$\subsetneq$	$\not\supseteq$ $\not\subseteq$	$\subsetneq$	
DFA(k+1)	$\subsetneq$	*PFA*(k+1) with bounded true-biased	$\subseteq$	*PFA*(k+1) with unbounded true-biased	$=$ *NFA*(k+1)

Table 3.2: Relations between classes of languages recognized with k and k+1 heads

Remark. Extra $\not\supseteq\not\subseteq$ could be written between *DFA*(k+1) and *PFA*(k) with unbounded.

An analogous table can be constructed for *PFA*(k) with false-biased error, while removing the last column ($=$ *NFA*).

3.3 Closure properties

Regular intersection

Theorem 3.3.1 (One-sided *PFA*(k) are closed under regular intersection)**.** *For $k \geq 1$, for every regular language $R \in \mathcal{R}$, and*

- *each language L accepted by a PFA(k) with true-biased error with error bound Λ, there is a PFA(k) recognizing $L \cap R$ with true-biased error with error bound Λ.*

- *each language L accepted by a PFA(k) with false-biased error with error bound Λ, there is a PFA(k) recognizing $L \cap R$ with false-biased error with error bound Λ.*
- *each language L accepted by a PFA(k) with unbounded true-biased error, there is a PFA(k) recognizing $L \cap R$ with unbounded true-biased error.*
- *each language L accepted by a PFA(k) with unbounded false-biased error, there is a PFA(k) recognizing $L \cap R$ with unbounded false-biased error.*

Proof. Our construction will be similar to the classic proof of why regular languages are closed under intersection (see [HMU07]). The new automaton A' will verify the regularity with its first head (by simulating the *DFA* recognizing R).

Let $A = (Q_A, \Sigma, \delta_A, q_0, Q_{acc}, Q_{rej})$ be the automaton recognizing L with given error. For each regular language, there exists a *DFA*, recognizing that language, let *DFA* $\bar{A}$ by the one recognizing R. We construct the new *PFA*(k) A' as follows. The set of states of the new automaton A' will be $Q \times \bar{Q}$, and the set of accepting states will be $Q_{acc} \times \bar{F}$. Formally:

$$A' \;=\; (Q_A{\times}\bar{Q},\ \Sigma,\ \delta',\ (q_0, \bar{q_0}),\ Q_{acc}{\times}\bar{F},\ Q_A{\times}\bar{Q} - Q_{acc}{\times}\bar{F})$$

where the new transition function δ', is constructed as follows:

$$\begin{aligned}
&(\forall q, q' \in Q_A)(\forall p \in \bar{Q})(\forall a_1, \dots, a_k \in \Sigma_\$)(\forall d_1, \dots, d_k \in \{0,1\})\\
&\quad \delta_A(q, a_1, \dots, a_k, q', d_1, d_2, \dots, d_k) = \mathbf{p} > 0\\
&\qquad\qquad \implies \delta'((q,p), a_1, \dots, a_k, (q', \bar{\delta}(p, a_1)), 1, d_2, \dots, d_k) = \mathbf{p} \quad \text{iff } d_1{=}1\\
&\qquad\qquad \implies \delta'((q,p), a_1, \dots, a_k, (q', p), 0, d_2, \dots, d_k) = \mathbf{p} \qquad\quad \text{iff } d_1{=}0
\end{aligned}$$

The correctness stems from the idea, that when A' moves the first head, it computes one step of computation of $\bar{A}$ (if head 1 is stationary, no computation takes place). Since *PFA*(k) can accept only if all heads arrive at \$, at the end of the computation, the *DFA* $\bar{A}$ already finished its computation (read the whole word). Thus, we know whether or not the word is in R, by simply looking at the state (of the regular component). Moreover, since all reads are at \$, the *PFA*(k) is already decided whether or not w is in L (look at the state of the original automaton). Hence, we know whether or not the word w is in language $L \cap R$ (it is, if the state of both the regular component and the original component is accepting, i.e., if the last state $(q, \bar{q}) \in Q_{acc}{\times}F$).

This construction uses no new randomization, we simply additionally reject words if they are not in R. Therefore, if a word w, was accepted (rejected) with probability $p_A(w)$ ($p_A^{rej}(w)$), it is now accepted (rejected) with probability $p_A(w)$ or 0 (respectively $p_A^{rej}(w)$ or 1). If L was accepted with bounded error, because accepting or rejecting words with probability 0/1 is “deterministic”, we have not “broken any bound”, hence satisfying any error bound (see Corollary 3.0.6). ☐

Complement

We will prove, both for bounded and unbounded *PFA*(k) in one proof, that the class of languages recognized by a *PFA*(k) with (un)bounded one-sided error are not closed under complement.

Corollary 3.3.2 (True-biased *PFA*(k) are not closed under complement)**.** *For all* $k \geq 2$*, there is a language* M_k*, recognized by a PFA*(k) *with bounded true-biased error, such that there exists no PFA*(k) *accepting* $(M_k)^c$ *with unbounded true-biased error.*

Proof. Let $M_k = (L_b)^c$, for $b = \binom{k}{2} + 1$. Firstly, by the Head Gap lemma (3.1.3), M_k can be recognized by *PFA*(2) with bounded true-biased error, hence by *PFA*(k) with (un)bounded true-biased error. Secondly, $(M_k)^c = (L_b)^{cc} = L_b$ cannot be recognized by no *PFA*(k) with unbounded true-biased error as a result of the Hierarchy Theorem for one-sided error (3.1.1). □

Corollary 3.3.3 (False-biased *PFA*(k) are not closed under complement)**.** *For all* $k \geq 2$*, there is a language* M_k*, recognized by a PFA*(k) *with bounded false-biased error, such that there exists no PFA*(k) *accepting* $(M_k)^c$ *with unbounded false-biased error.*

Proof. Let $M_k = L_b$, for $b = \binom{k}{2} + 1$, the reasoning is analogous to the previous proof. □

Intersection

In order to prove that true-biased *PFA*(k), both bounded and unbounded, are not closed under intersection, we define the following useful languages.

$$\begin{aligned} L_{vwv} &= \{v * w * v \mid (v \in \{0,1\}^*) \wedge (w \in \{0,1,*\}^*)\} \\ L'_b &= \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^*) \wedge (w_i = w_{2b+1-i}) \text{ for } 2 \leq i \leq b\} \end{aligned} \tag{3.1}$$

$L_{vwv} \cap L'_b = L_b$, since L'_b is L_b where we do not care about the first and last sub-words.

Lemma 3.3.4. *The language* L_{vwv} *can be recognized by a DFA*(3) *and the language* L'_b *can be recognized by a DFA*(k) *for* $b \leq \binom{k}{2} + 1$*, for all* $k \geq 2$*.*

Proof. The language L_{vwv} can be recognized by a *DFA*(3) by the following algorithm, combined with a regular check for the format (at least two $*$).

i. "Find last sub-word"

   1. Move head 3 to the next *.
   2. If head 3 reads *, continue. If head 3 reads $ instead, go to [ii.],
   3. Move head 2 to the next *.
   4. Goto [1.].

ii. “Verify equality” Accept if and only if sub-words under heads 1 and 2 are equal. (move heads simultaneously while they read the same symbols).

Since $L'_b = \{0,1\}^* \cdot \{*\} \cdot L_{b-1} \cdot \{*\} \cdot \{0,1\}^*$, it is not hard to see, that to recognize this language we can do a regular check for format, and use the deterministic algorithm from the Hierarchy Theorem (2.1.1), to verify the rest (more complicated part). $\square$

Lemma 3.3.5. *L_{vwv} can be recognized by a PFA(3) with bounded true-biased error, and L'_b can be recognized by a PFA(k) with bounded true-biased error, for $b \leq \binom{k}{2} + 1$,*

Proof. Trivial observation, follows from Lemma 3.3.4 and Corollary 3.0.6. $\square$

Now we can state the theorem we were aiming for, that *PFA*(k) are not closed under intersection. However, we prove it for case $k \geq 3$, and only later add a case for $k = 2$. The reason is, that in the general case, we prove a bit more, that for each k, one of the languages in the counterexample pair is recognized by a *PFA* with just fixed number of heads (3).

Theorem 3.3.6 (True-biased *PFA*(k) are not closed under intersection)**.** *There exists a language M, such that for all $k \geq 3$, there is a language M_k, such that there exists no PFA(k) recognizing $M_k \cap M$ with unbounded true-biased error, yet both M_k and M are recognized by a PFA(k) with bounded true-biased error.*

Proof. Let $M = L_{vwv}$, and $M_k = L'_b$ for $b = \binom{k}{2} + 1$. We see that $M_k \cap M = L_b$, which we know, by the Hierarchy Theorem for one-sided error (3.1.1), that cannot be recognized by a *PFA*(k) with unbounded true-biased error, since $b > \binom{k}{2}$. L_{vwv}, L'_b are accepted by a *PFA*(k) with bounded true-biased error (for $k \geq 3$), because of the previous Lemma 3.3.5. $\square$

Theorem 3.3.7 (True-biased *PFA*(2) are (also) not closed under intersection)**.** *There exist two languages M_2 and M, such that there exists no PFA(2) recognizing $M_2 \cap M$ with unbounded true-biased error, yet both M_2 and M are recognized by a PFA(2) with bounded true-biased error.*

Proof. Let $M = \{v * w_1 * w_2 * v \mid (v \in \{0,1\}^*) \wedge (w_1, w_2 \in \{0,1\}^*)\}$ and $M_2 = L'_2$. The language M is easily acceptable by a *DFA*(2), since now, we know the number of $*$. L'_2 is also recognizable *PFA*(2) (by Lemma 3.3.5). Also $L'_2 \cap M = L_2$ (L_{uvvu}), which we know to be not recognizable by any *PFA*(2). $\square$

Now we look at the class of languages recognized by Monte-Carlo *PFA*(k) with false-biased error. Unlike their true-biased counterpart, this class is closed under intersection. However, our general construction comes with a cost of increasing the probability of error (which may not always be the optimal error bound).

Theorem 3.3.8 (False-biased *PFA*(k) are closed under intersection)**.** *For each $k \geq 1$, and for every two languages L_1 and L_2,*

- *both accepted by some PFA(k) with bounded false-biased error,*
  there is a PFA(k) recognizing $L_1 \cap L_2$ with bounded false-biased error.
- *both accepted by some PFA(k) with unbounded false-biased error,*
  there is a PFA(k) recognizing $L_1 \cap L_2$ with unbounded false-biased error.

Informally. We construct a *PFA*(k) that will with probability $\mathbf{p}_1$ verify if word w is in L_1, and with $\mathbf{p}_2$ if w is in L_2. Accepting if and only if the selected algorithm accepts. (Both succeed on the good words, just as the false-biased definition requires.)

Proof. Let $A' = (Q', \Sigma, \delta', q'_0, Q'_{acc}, Q'_{rej})$ be the *PFA*(k) recognizing L_1 with false-biased error with error bound Λ_1 and $A'' = (Q'', \Sigma, \delta'', q''_0, Q''_{acc}, Q''_{rej})$ the *PFA*(k) recognizing L_2 with false-biased error with error bound Λ_2. Without loss of generality we assume that $Q' \cap Q'' = \emptyset$. We construct the *PFA*(k) $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ accepting $L_1 \cap L_2$ with bounded false-biased error by joining states and delta functions of A' and A''. We first define an initial delta function δ_{init} for any[3] non-zero probabilities $\mathbf{p}_1 + \mathbf{p}_2 = 1$, by $(\forall a_1, \ldots, a_k \in \Sigma_\$)$:

$$\delta_{init}(q_0, a_1, \ldots, a_k, q'_0, 0, \ldots, 0)=\mathbf{p}_1, \quad \delta_{init}(q_0, a_1, \ldots, a_k, q''_0, 0, \ldots, 0)=\mathbf{p}_2$$

We then construct A:

$Q = Q' \cup Q'' \cup \{q_0\}$

$\delta = \delta' \cup \delta'' \cup \delta_{init}$ (since functions are just sets in their essence).

$Q_{acc} = Q'_{acc} \cup Q''_{acc}$

$Q_{rej} = Q'_{rej} \cup Q''_{rej}$

The automaton constructed in this fashion, with probability $\mathbf{p}_1$ simulates a computation on A', and with probability $\mathbf{p}_2$, simulates a computation on A''. Therefore we can analyze that words

- in L_1 and L_2 are accepted with probability 1.
- in L_1 but not in L_2 are accepted with probability $\leq \mathbf{p}_1 + \mathbf{p}_2\Lambda_2$.
- in L_2 but not in L_1 are accepted with probability $\leq \mathbf{p}_1\Lambda_1 + \mathbf{p}_2$.
- not in L_2 neither in L_1 are accepted with probability $\leq \mathbf{p}_1\Lambda_1 + \mathbf{p}_2\Lambda_2$

Hence, every word not in $L_1 \cup L_2$ is accepted by A with probability at most $\Lambda = \max\{\mathbf{p}_1 + \mathbf{p}_2\Lambda_2,\ \mathbf{p}_1\Lambda_1 + \mathbf{p}_2\}$ (since $\Lambda_1, \Lambda_2 \leq 1$). Because $\mathbf{p}_1\mathbf{p}_2>0$, $\Lambda < 1$, therefore $L_1 \cap L_2$ is recognized by *PFA*(k) A with bounded false-based error.

The proof for the unbounded case is analogous, we simply need to argue that the probabilities of accepting words not in $L_1 \cap L_2$ is not 1, which is obvious since with

[3]subject to the T_P – the set of allowed probabilities

nonzero probability we simulate an algorithm that with nonzero probability rejects such word. Thus it is rejected with nonzero probability, which is equivalent with accepting said word with not-certain probability (< 1). □

Remark 3.3.9. We may want to minimize Λ. First, let $\mathbf{p}_1$=$\mathbf{p}$ and $\mathbf{p}_2$=1−$\mathbf{p}$, then we are minimizing $max\{\mathbf{p}+(1-\mathbf{p})\Lambda_2,\ \mathbf{p}\Lambda_1+(1-\mathbf{p})\}$. By a simple observation we see that by increasing $\mathbf{p}$, we increase the first probability, and by decreasing $\mathbf{p}$ we increase the second. Hence, the optimal minimized Λ will have these two probabilities equal (because by changing $\mathbf{p}$, we increase one of the probabilities, increasing Λ, no longer being optimal).

$$\begin{aligned}
\mathbf{p} + (1-\mathbf{p})\Lambda_2 &= \mathbf{p}\Lambda_1 + (1-\mathbf{p}) \\
\mathbf{p}(1-\Lambda_1) + (1-\mathbf{p})(\Lambda_2 - 1) &= 0 \\
\mathbf{p}(1-\Lambda_1) + 1(\Lambda_2 - 1) - \mathbf{p}(\Lambda_2 - 1) &= 0 \\
\mathbf{p}(1-\Lambda_1 + 1 - \Lambda_2) &= (1-\Lambda_2) \\
\mathbf{p} &= \frac{1-\Lambda_2}{2-\Lambda_1-\Lambda_2}
\end{aligned}$$

*(If one were to compute $(1-\mathbf{p})$ one would end up at a symmetric $\frac{1-\Lambda_1}{2-\Lambda_1-\Lambda_2}$.)

$$\begin{aligned}
\Lambda &= \mathbf{p} + (1-\mathbf{p})\Lambda_2 \\
&= \frac{1-\Lambda_2}{2-\Lambda_1-\Lambda_2} + \left(1 - \frac{1-\Lambda_2}{2-\Lambda_1-\Lambda_2}\right)\Lambda_2 \\
&= \frac{1-\Lambda_2}{2-\Lambda_1-\Lambda_2} + \frac{(2-\Lambda_1-\Lambda_2)-(1-\Lambda_2)}{2-\Lambda_1-\Lambda_2}\Lambda_2 \\
&= \frac{1-\Lambda_2}{2-\Lambda_1-\Lambda_2} + \frac{1-\Lambda_1}{2-\Lambda_1-\Lambda_2}\Lambda_2 \\
&= \frac{1-\Lambda_2+\Lambda_2-\Lambda_1\Lambda_2}{2-\Lambda_1-\Lambda_2} \\
\Lambda &= \frac{1-\Lambda_1\Lambda_2}{2-\Lambda_1-\Lambda_2}
\end{aligned}$$

We can see that the optimal probability is not always picking $\mathbf{p} = 1/2$, but actually computing $\mathbf{p}$ depending on the error bounds of the original algorithms.

Union

Now we explore the set operation union. By de Morgan's laws, we know that $L_1 \cup L_2 = ((L_1)^c \cap (L_2)^c)^c$. Therefore we can easily use proofs for intersection to prove properties of union, because of the Complement lemma 3.0.7.

Theorem 3.3.10 (True-biased *PFA*(k) are closed under union)**.** *For each $k \geq 1$, and for every two languages L_1 and L_2,*

- *both accepted by some PFA*(k) *with bounded true-biased error,*
  there is a PFA(k) *recognizing* $L_1 \cap L_2$ *with bounded true-biased error.*
- *both accepted by some PFA*(k) *with unbounded true-biased error,*
  there is a PFA(k) *recognizing* $L_1 \cap L_2$ *with unbounded true-biased error.*

Informally. We construct a *PFA*(k) that will with probability $\mathbf{p}_1$ verify if word w is in L_1, and with $\mathbf{p}_2$ if w is in L_2. Accepting if and only if the selected algorithm accepts. (Both fail on the incorrect words, just as the true-biased definition requires.)

Proof. Since L_1 and L_2 are accepted by some *PFA*(k)'s with true-biased error with error bounds Λ_1 and Λ_2, by the Complement lemma 3.0.7, $(L_1)^c$ and $(L_2)^c$ are accepted by some *PFA*(k)'s with false-biased error with error bounds Λ_1 and Λ_2. Because the class of languages recognized by a *PFA*(k) with bounded false-biased error are closed under intersection (Theorem 3.3.8), we know that there exists a *PFA*(k) accepting $(L_1)^c \cap (L_2)^c$ with false-biased error with error bound Λ. We now apply the Complement lemma (again), to prove that $((L_1)^c \cap (L_2)^c)^c$ is accepted by some *PFA*(k) with true-biased error, with error bound Λ. Hence we have proven that $L_1 \cup L_2$ is accepted by a *PFA*(k) with bounded true-biased error. □

Remark. Moreover, since the Complement lemma keeps the error bounds intact, we can use the Remark 3.3.9 to calculate the optimal probabilities $\mathbf{p}_1, \mathbf{p}_2$, to minimize Λ.

Theorem 3.3.11 (False-biased *PFA*(k) are not closed under union)**.** *There exists a language* M*, such that for all* $k \geq 3$*, there exists a language* M_k*, such that there exists no PFA*(k) *recognizing* $M_k \cup M$ *with unbounded false-biased error, yet both* M_k *and* M *are recognized by a PFA*(k) *with bounded false-biased error.*

Proof. Let $M = (L_{vwv})^c$ and $M_k = (L'_b)^c$ for $b = \binom{k}{2}+1$. Both M_k and M are recognized by a *PFA*(k) with bounded false-biased error, by the Complement lemma (3.0.7), since both $(M_k)^c = L'_b$ and $M^c = L_{vwv}$ are recognized by some *PFA*(k), $k \geq 3$, with bounded true-biased error (Lemma 3.3.5).

For the purposes of contradiction, assume that there exists a *PFA*(k) recognizing $M_k \cup M$ with unbounded false-biased error. Then, by the Complement lemma, there exists a *PFA*(k) recognizing $(M_k \cup M)^c = (M_k)^c \cap M^c = L'_b \cap L_{vwv} = L_b$ with unbounded true-biased error, which contradicts the Hierarchy Theorem for one-sided error (3.1.1), since $b = \binom{k}{2}+1 > \binom{k}{2}$. □

Theorem 3.3.12 (False-biased *PFA*(2) are (also) not closed under union)**.** *There exist two languages* M_2 *and* M*, such that there exists no PFA*(2) *recognizing* $M_2 \cup M$ *with unbounded false-biased error, yet both* M_2 *and* M *are recognized by a PFA*(2) *with bounded false-biased error.*

Proof. Proof is analogous to the proof of the previous theorem (3.3.11). We adapt the corresponding proof of intersection, with the use of the Complement lemma. □

Summary: Closure properties

The following table summarizes the theorems encountered in this section.

Class of languages recognized by	$\cap R$	^{c}	$\cap$	$\cup$
PFA with (un)bounded true-biased error	✓	–	–	✓
PFA with (un)bounded false-biased error	✓	–	✓	–

Legend: ✓: closed under this operation. –: not closed under this operation.

Chapter 4

LasVegas

In this chapter we study the LasVegas variation of the one-way multi-head probabilistic finite automata. We first state few obvious lemmas, as we did in chapter for Monte-Carlo *PFA*(k). For brevity, instead of writing $(1-\kappa)$-correct LasVegas *PFA*(k), we may write α-correct LasVegas *PFA*(k) ($\alpha = 1{-}\kappa$).

Lemma 4.0.1. *Let L be a language recognized by a LasVegas PFA(k) A, then*

1. $p(w) > 0 \iff p^{rej}(w) = 0$
2. $p(w) = 0 \iff p^{rej}(w) > 0$
3. $L(A) = \{w \mid 0 = p^{rej}(w)\}$
4. $L(A)^c = \{w \mid 0 < p^{rej}(w)\}$

Proof. Because the definition of LasVegas *PFA*(k), requires that $p^{FAIL}(x) < 1$, and because $p(w) + p^{rej}(w) + p^{FAIL}(w) = 1$ for each w, we prove the first two points:

$$\begin{aligned}
p(w) > 0 &\overset{def}{\Longrightarrow} p^{rej}(w) = 0 \\
p^{rej}(w) = 0 &\Longrightarrow 1 = p(w){+}p^{FAIL}(w) < p(w){+}1 \Longrightarrow 0 < p(w) \\
p^{rej}(w) > 0 &\overset{def}{\Longrightarrow} p(w) = 0 \\
p(w) = 0 &\Longrightarrow 1 = p^{rej}(w){+}p^{FAIL}(w) < p^{rej}(w){+}1 \Longrightarrow 0 < p^{rej}(w)
\end{aligned}$$

To prove the rest, we recall the definition, for a language accepted by LasVegas *PFA*(k): $L(A){=}\{w \in \Sigma^* \mid 0 < p(w)\}$. Since $p(w) > 0 \Leftrightarrow p^{rej}(w) = 0$, we equivalently rewrite it as $L(A){=}\{w \in \Sigma^* \mid 0 = p^{rej}(w)\}$, proving the third point. The complement ($L(A)^c$), by definition, is $\{w \in \Sigma^* \mid 0 = p(w)\}$. Since $p(w) = 0 \Leftrightarrow p^{rej}(w) > 0$, we equivalently rewrite it as $L(A)^c = \{w \in \Sigma^* \mid 0 < p^{rej}(w)\}$, proving the last point. □

Lemma 4.0.2. *For each $0 \le \kappa_1 < \kappa_2 < 1$: If A is a $(1-\kappa_1)$-correct LasVegas PFA(k), then it is also a $(1-\kappa_2)$-correct LasVegas PFA(k).*

Proof. Follows trivially from definition (1.4.10). □

Lemma 4.0.3 ($DFA(k) \subseteq$ LasVegas $PFA(k)$)**.** *Let L be a language accepted by $DFA(k)$, then we can construct a 1-correct LasVegas $PFA(k)$ accepting L.*

Proof. Since L is accepted by $DFA(k)$, let the $DFA(k)$ $A = (Q, \Sigma, \delta_A, q_0, F)$ accepting L in a normal form where each computation on it is finite[1]. We construct $PFA(k)$ $A' = (Q, \Sigma, \delta'_A, q_0, F, Q - F)$, where for each valid transition in δ_A, we set such transition's probability to 1 in δ'_A.
Formally: $(\forall q, p \in Q)(\forall a_1, \ldots, a_k \in \Sigma_\$)(\forall d_1, \ldots, d_k \in \{0, 1\})$:

$$\delta_A(q, a_1, \ldots, a_k) = (p, d_1, \ldots, d_k) \implies \delta'(q, a_1, \ldots, a_k, p, d_1, \ldots, d_k) = 1$$

The $PFA(k)$ A constructed this way is a LasVegas $PFA(k)$, since it behaves exactly the same as the $DFA(k)$ A (deterministically). Moreover, since the inconclusive states $Q - (F \cup (Q - F)) = \emptyset$ are nonexistent, and since all possible computations on the $DFA(k)$ were finite, the probability of $FAILURE$ is bounded by the constant 0. Therefore, A is a 1-correct LasVegas $PFA(k)$. ☐

Lemma 4.0.4 (LasVegas $PFA(k) \subseteq NFA(k)$)**.** *Let L be a language accepted by a LasVegas $PFA(k)$, then we can construct a $NFA(k)$ accepting L.*

Proof. Let $A = (Q, \Sigma, \delta_A, q_0, Q_{acc}, Q_{rej})$, we construct $NFA(k)$ $A = (Q, \Sigma, \delta'_A, q_0, Q_{acc})$, where for each transition with nonzero probability in δ_A, we add it as a possible transition in δ'_A. Formally: $(\forall q, p \in Q)(\forall a_1, \ldots, a_k \in \Sigma_\$)(\forall d_1, \ldots, d_k \in \{0, 1\})$:

$$\delta(q, a_1, \ldots, a_k, p, d_1, \ldots, d_k) > 0 \implies \delta(q, a_1, \ldots, a_k) \ni (p, d_1, \ldots, d_k)$$

$L(A)$ contains words with $p(x) > 0$, in another words, for each word w in $L(A)$, there exists an accepting computation on w. Since we have just copied the δ-function (removing the probabilities) the accepting computation is still valid, hence, the non-determinism will find it, and accept it.

$L(A)$ does not contain words x with $p(x) = 0$, words with no accepting computation. Hence non-determinism will conclude that there exists no accepting computation on x, and reject it. Thus, for the $NFA(k)$ A' constructed, we have proven $L(A') = L(A)$. ☐

Remark. We have not required anywhere that the LasVegas $PFA(k)$ be α-correct. Hence this proof works even for “unbounded” LasVegas.

4.1 Union of DFA(k)

Consider the way how LasVegas algorithms are defined. When they give an answer, it must be correct, yet sometimes, they may provide no answer ($FAILURE$). Since

[1]Possible, since we can detect infinite cycles in deterministic automata ($DFA(k)$) – one cannot meaningfully move without moving heads for much longer than the number of states $|Q|$.

we are in the domain of one-way automata, we can never re-read our input (and we cannot store it in our finite state), yet we can see one set of problems that can easily be solved by these LasVegas multi-head automata, and not their multi-head deterministic counterparts: Problems where no actual non-determinism is involved, yet the information, about what we were supposed to verify deterministically, is given too late (e.g. at the end). Let L be a language over $\Sigma = \{0, 1, \#, U, V\}$:

$$L = \{u\#v\#v'\#u'\alpha \mid u, v, v', u' \in \{0,1\}^*, \alpha{\in}\{U, V\},\ (\alpha{=}U \Rightarrow u{=}u'),\ (\alpha{=}V \Rightarrow v{=}v')\}$$

This idea is similar to the idea of *marked union*, but we give the information too late, rather than early. Example: an *end-marked union*.

$$\begin{aligned} L &= \{wU \mid w \in L_u\} \ \cup\ \{wV \mid w \in L_v\} \\ L_u &= \{u\#v\#v'\#u' \mid u, v, v', u' \in \{0,1\}^*,\ u = u'\} \\ L_v &= \{u\#v\#v'\#u' \mid u, v, v', u' \in \{0,1\}^*,\ v = v'\} \end{aligned} \tag{4.1}$$

Remark. Note that all the above is true for a usual union of languages L'_u, L'_v, if they have a different end-marker, such as $\{u\#v\#v'\#u'U \mid u{=}u'\}{\cup}\{u\#v\#v'\#u'V \mid v{=}v'\}$.

Lemma 4.1.1. *The above constructed language L cannot be accepted by any DFA(2).*

Proof. If, during computation, the automaton reads the last symbol U/V, one head is at the end of the word. And with only one remaining head, the automaton cannot check the equality of two sub-words,a corollary of simple pumping lemma (see [HMU07]).

This proof is quite similar to the proof of 2.0.1 (*DFA*(2) cannot accept L_{uvvu}). However the trick this time is the following: We define a *mistake* for a *pattern* of a word on a deterministic automaton A, as the index of sub-words, in which the heads are never simultaneously (we know from our analysis of L_{uvvu}, that on words of format $u * v * v' * u'$, with two heads, either u, u' or v, v' are never visited simultaneously). Then, we classify words $w \in (L_u \cup L_v)$ into classes depending on the pair of patterns $(pattern(wU), pattern(wV))$, we prove that for each word the mistake in pattern for wU and wV is the same. Lastly, we look at the mistake (wlog V), and we use the cut-and-paste argument (on xV, yV).

Formally, we define *location* and *pattern* identically as in Lemma 2.0.1. We additionally define a *mistake* for a *pattern* of a word on a deterministic automaton A, as the index of sub-words, in which the heads are never simultaneously (we know from our analysis of L_{uvvu}, that on words of format $u * v * v' * u'$, with two heads, either u, u' or v, v' are never visited simultaneously).

Define $L_n = \{w_1\#w_2\#w'_2\#w'_1 \mid w_1{=}w1' \vee w_2{=}w'_2 \in \Sigma^n\}$, obviously $L_n \subseteq (L_u{\cup}L_v)$. We classify words $w \in L_n$ into classes based on the pair $(pattern(wU), pattern(wV))$.

There are $2^{3n}+2^{3n}-2^{2n}(\geq 2^{3n})$ words in L_n, the length of a pattern is at most $2(4+1)$ (each head must go through all 4 sub-words and \$), then the number of possible patterns $p(n)$ is at most $(|Q|\cdot(4(n+1))^2)^{2(4+1)}$ [2]. Thus, based on the Dirichlet's principle, there is a class S_0 with $|S_0|\geq \frac{2^{3n}}{p(n)^2}$ words.

Now, since words in S_0 have the same mistake (w_i), we classify words in S_0 into classes based on the string "$w_j w'_j$" $i\neq j$ (if mistake on u, classify based on v). By the Dirichlet's principle, there is a class S_1, such that $|S_1|\geq \frac{|S_0|}{2^{2n}}\geq \frac{2^n}{p(n)^2}$. Pick reasonably large n, such that $|S_1|\geq 2$ (we can, since $p(n)$ is polynomial in n).

We now have two words $x{\neq}y\in S_1$, which have the same pattern for xU, yU and xV,yV. Also, these two words differ only in the corresponding sub-words that are never read simultaneously.

Hence an analogous cut-and-paste argument follows. WLOG. heads never visit u, u' simultaneously. We take the two words: $x{=}x_u\#x_v\#x_{v'}\#x_{u'}U$, $y{=}y_u\#y_v\#y_{v'}\#y_{u'}U$ and create $z{=}x_u\#x_v\#x_{v'}\#y_{u'}U$. Since when reading x_u, the other head does not read $y_{u'}$, the heads of the $DFA(2)$ read the same input (and vice-versa), cut-and-pasting configurations as in 2.0.1, constructs a valid accepting computation of A for a word not in L.

We have thus constructed an accepting computation on a word where the sub-words u, u' were supposed to be equal, but were not. Contradiction.

Addendum: The reason why, for each 2-head deterministic automaton A, and for each word $w\in L_u\cup L_v$, the *mistake* for *pattern* for wU and wV on A, is the same, is the following. The difference in the mistake on wU, wV must take place before any head reaches the end, since after one heads reaches the end, the movement of heads in pattern is determined (the other head goes to the end). However, before any head reads the last symbol, the heads read the same input as if they were reading w (have the same pattern). Therefore, the mistake is the same on wU and wV, moreover, the pattern is the same until one head reads the end. □

Remark. Note that we have intentionally used the words "too late" to describe the "timing" when we get the missing information. Because we can define a language L', similar to L, such that we get the information "before" the end and still it won't help.

$$L' = \{\ u\#v\#v'\alpha_v\#u'\alpha_u \mid \alpha_u,\alpha_v\in\{\varepsilon,!\},\ \alpha_u\alpha_v{=}!,\ (\alpha_u{=}!\implies u{=}u'),\ (\alpha_v{=}!\implies v{=}v')\ \}$$

In this case we are given the information (which word we should have checked) only later, after one head irreversibly skips over v', or compares v, v' irreversibly skipping u, i.e., after it checked/decided not to check one of the words. The argumentation why

[2] $(\#_{configurations})^{\#_{heads}(\#_{subwords}+1)}$

L' cannot be accepted by *DFA*(2) would follow analogously as above – since we just argued why the *mistake* is the same "both" *patterns* [3].

However, we prove that LasVegas *PFA*(k) can recognize a union of disjoint languages recognized by *DFA*(k), with an additional requirement, that those languages have different format of words, detectable by a finite automaton. Such as languages $\{u\#v\#v\#u\} \cup \{w * w\} \cup \{a^n b^n c^n d^n\}$.

Theorem 4.1.2. *Let $\mathfrak{L} = (L_1, L_2, \dots, L_m)$ be an m-tuple of disjoint languages recognizable by k-head one-way deterministic automata ($L_i \in \mathscr{L}(DFA(k))$). Let $\mathfrak{A} = (A_1, A_2, \dots, A_m)$ be an m-tuple of one-way deterministic automata (DFA(1) A_i), such that*

$$(\forall i)(\forall w \in \Sigma^*)\ w \in L_i \ \Rightarrow\ w \in L(A_i)$$
$$(\forall i, j\ i{\neq}j)(\forall w \in \Sigma^*)\ w \in L_j \ \Rightarrow\ w \notin L(A_i)$$

Then, there exists a $1/m$-correct LasVegas PFA(k) that accepts $\bigcup_{i=1}^{m} L_i$.
(It follows logically that $L_i \subseteq L(A_i)$.)

Informally. We have some languages we want to accept a union of. Moreover, we have a finite automaton that can (regularly) detect in which of the given languages the input word has a chance of being accepted. Therefore, we can verify whether $w \in L_i$, for random i, and the regular check will, at the end, tell us whether or not we checked the correct L_i. (Only one k-headed automaton can run at one time, however any number of regular checks can be performed in parallel)

For example, we detect $\{u\#v\#v\#u\}$ or $\{w\#w\}$ or $\{a^n b^n\}$. The *DFA* check, if word has exactly 3 # or exactly 1 # or zero #. Note that any automaton in $\mathfrak{A}$ can do anything with a word that has 10 #, since none of our languages contain such word.

Proof. We have that L_i is recognizable by *DFA*(k), so it follows that for each i, there exists a *DFA*(k) B_i such that $L(B_i) = L_i$. We also have a m-tuple of finite automata $\mathfrak{A}$, each accepting a Regular language. It then follows (because regular languages are closed under union), that there must exist a one-way deterministic automaton A accepting a union of these languages $\bigcup_{i=1}^{m} L(A_i)$.

The one-way LasVegas *PFA*(k) accepting the union $L = \bigcup_{i=1}^{m} L_i$ works as follows:

- "Roll a m-sided die"
  With probability $\frac{1}{m}$, go into into a state $q_{0,i}$ for each $i \in \{1, \dots, m\}$
- Simulate the computation of B_i (k-head test $w \overset{?}{\in} L_i$) while simultaneously running the computations of A_i and A (both regular).

[3] We argued that the automaton decides "not to check" one of the words prior to reading "!". Hence, it cannot compute differently based on the sole change in "!".

- After each and every head arrives at end (\$), check the states:
  - if A rejected, REJECT. //definitely not in any L_i
  - if A accepted,
    * if A_i rejected, FAILURE //not in this L_i, maybe in another
    * if A_i accepted, //maybe in this L_i, but nowhere else
      · if B_i accepted, ACCEPT.
      · if B_i rejected, REJECT.

Probabilistic analysis:

- $w \in L \implies (\exists j) w \in L_j$, and we know that $L_j \subseteq L(A_j) \subseteq L(A)$.
  Hence, if we roll $i{\neq}j$, A_i will reject, result is *FAILURE* (with probability $\frac{m-1}{m}$).
  However, if we roll $i{=}j$, A,A_i and B_i will accept, result is *ACCEPT* (correctly).
- $w \notin L$, we need to consider two cases: $w \notin L \wedge w \notin L(A)$ and $w \notin L \wedge w \in L(A)$.

  $w \notin L \wedge w \notin L(A) \implies$ will trivially be rejected by A (correctly).

  $w \notin L \wedge w \in L(A) \implies (\exists j) w \in L(A_j)$
  Hence, if we roll $i{\neq}j$, A_i will reject, result is *FAILURE* (with probability $\frac{m-1}{m}$).
  But, if we pick $i{=}j$, A_i will accept, and B_i reject, result is *REJECT* (correctly).

Hence, the probability of *FAILURE* on each $w \in \Sigma^*$ is at most $\kappa = \frac{m-1}{m} = 1 - \frac{1}{m}$, and when the algorithm accepts or rejects, it does so correctly (It never accepts a word it rejects or vice-versa). We therefore satisfy the definition of a language recognized by a $1/m$-correct LasVegas $PFA(k)$. □

Remark. Note that $(\forall i)\ L_i \subseteq L(A_i)$, therefore a situation "A_i rejected and B_i accepted" will never happen. Note that we need to check both A and A_i, in order to differentiate between w not in *any* L_i, and w not in *this* L_i.

Corollary 4.1.3 ($DFA(2) \subsetneq$ LasVegas $PFA(2)$)**.** *There is a language M, recognizable by a LasVegas $PFA(2)$, but by no $DFA(2)$.*

Proof. Let $M{=}L$ (see equation 4.1). By Lemma 4.1.1, L cannot be recognized by no *DFA(2)*. However L can by recognized by a 1/2-correct LasVegas $PFA(2)$, since for $L = L_u U \cup L_v V$, we can use Lemma 4.1.2 with $\mathfrak{L} = (L_u{\cdot}\{U\}, L_v{\cdot}\{V\})$ and $\mathfrak{A} = (A_U, A_V)$, where A_U, A_V are finite automata recognizing regular languages $\{0,1\}^*U$, $\{0,1\}^*V$ respectively. □

4.2 Hierarchy for LasVegas

Just as we did in the chapter about Monte-Carlo $PFA(k)$, we can ask, whether or not, adding heads does increase the expressive power of the model. In this case, the result and the proof are analogous to the Monte-Carlo case.

Consider this language (see 2.1):

$$L_b = \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^*) \wedge (w_i = w_{2b+1-i}) \text{ for } 1 \leq i \leq 2b\}$$

Theorem 4.2.1 (Hierarchy for LasVegas *PFA*(k))**.** *For each integer* $k \geq 2$ *the language* L_b *is recognizable by a LasVegas PFA*(k) *if and only if* $b \leq \binom{k}{2}$.

Proof. When $b \leq \binom{k}{2}$, the language L_b is recognizable by a *DFA*(k) as a result of the Hierarchy Theorem (2.1.1). Thus, by Lemma 4.0.3, we construct a 1-correct LasVegas *PFA*(k) recognizing L_b. Secondly, L_b is not recognizable by any LasVegas *PFA*(k) when $b > \binom{k}{2}$, because if such LasVegas *PFA*(k) existed, we could construct a *NFA*(k) recognising L_b (Lemma 4.0.4), which would contradict the Hierarchy Theorem. □

Corollary 4.2.2. L_b *is accepted by a* 1*-correct LasVegas PFA*(k)*, for* $b \leq \binom{k}{2}$, $k \geq 2$.

Remark. We have proven, that for each $k \geq 2$, there is a language M_k (M'_k) that is recognized by a 1-correct LasVegas *PFA*$(k+1)$ but not by any LasVegas *PFA*(k). Hence the name, "Hierarchy". Also note that we have proven this hierarchy for both α-correct and unbounded LasVegas *PFA*(k).

4.3 LasVegas and Monte-Carlo

In this section, we explore the relations between LasVegas and Monte-Carlo *PFA*(k). We show analogous results as in the model of Turing machines $ZPP = RP \cap coRP$, i.e., LasVegas can recognize languages if and only if both True-biased and False-biased algorithms can.

Lemma 4.3.1 (True-Biased $\supseteq$ LasVegas)**.** *For every* $k \geq 1$*, and every language* L *recognized by a* $(1-\kappa)$*-correct LasVegas PFA*(k)*,* L *can also be recognized by a PFA*(k) *with bounded true-biased error with error bound* κ.

Informally. LasVegas always tells the truth. Thus, if we want to be true-biased, we move the inconclusive computations to rejecting (accepting will still be truthful).

Proof. Let $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ be the $(1-\kappa)$-correct LasVegas *PFA*(k) in ε-free normal form (we need to avoid infinite computations) recognizing L. We construct $A' = (Q, \Sigma, \delta, q_0, Q_{acc}, Q - Q_{acc})$. By this construction, each computation that previously ended with *FAILURE*, ($p_A^{FAIL}(w) \leq \kappa$), is now rejecting. Hence, for $w \in L$, the probability of accepting w is just as it was before, $\geq 1-\kappa$. The probability of accepting $w \notin L$ is also still 0. Moreover, by transferring the "inconclusive" states to rejecting, we have eliminated inconclusive computations (A' is Monte-Carlo). Thus satisfying the definition of recognizing L with bounded true-biased error with error bound κ (section 1.3). □

Lemma 4.3.2 (False-Biased $\supseteq$ LasVegas)**.** *For every $k \geq 1$, and every language L recognized by a $(1-\kappa)$-correct LasVegas PFA(k), L can also be recognized by a PFA(k) with bounded false-biased error with error bound κ.*

Informally. LasVegas tells always the truth, thus if we want to be false-biased, we move the inconclusive computations to accepting (rejecting will be truthful).

Proof. Let $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ be the $(1-\kappa)$-correct LasVegas *PFA*(k) in ε-free normal form (we need to avoid infinite computations) recognizing L. We construct $A' = (Q, \Sigma, \delta, q_0, Q - Q_{rej}, Q_{rej})$. By this construction, each computation that previously ended with *FAILURE* ($p_A^{FAIL}(w) \leq \kappa$), now ends with acceptance. Hence, for $w \notin L$, the probability of rejecting w is just as it was before, $\geq 1-\kappa$. The probability of rejecting $w \in L$ is also still 0. Therefore the probability of accepting $w \notin L$ is $\leq \kappa$, and the probability of accepting $w \in L$ is 1. By transferring the "inconclusive" states to rejecting, we have eliminated inconclusive computations. Hence, A' is a Monte-Carlo *PFA*(k) that is satisfying the definition of accepting with false-biased error with error bound κ (see 1.3). $\square$

Lemma 4.3.3 (LasVegas $\supseteq$ (True-biased $\cap$ False-biased))**.** *For each $k \geq 1$, and each language L recognized by a PFA(k) with true-biased error with error bound Λ_1, simultaneously recognized by a PFA(k) with false-biased error with error bound Λ_2, the language L is also recognized by a $(1-\kappa)$-correct LasVegas PFA(k).*

Informally. The LasVegas randomly chooses one of the algorithms, if picked true-biased and accepts, or picked false-biased and rejects it knows it is correct. In the remaining cases, it cannot be certain, hence it outputs *FAILURE*.

Proof. Let A' be the *PFA*(k) accepting L with true-biased error with error bound Λ_1, and A'' the *PFA*(k) accepting L with false-biased error with error bound Λ_2. Let $A' = (Q', \Sigma, \delta', q_0', Q'_{acc}, Q'_{rej})$, and $A'' = (Q'', \Sigma, \delta'', q_0'', Q''_{acc}, Q''_{rej})$ such that $Q' \cap Q'' = \emptyset$.

We construct the LasVegas *PFA*(k) accepting L by joining states and delta functions of A' and A''. We first define an initial delta function δ_{init} for any[4] non-zero probabilities $\mathbf{p}_1 + \mathbf{p}_2 = 1$, by the following ($\forall a_1, \ldots, a_k \in \Sigma_\$$) :

$$\delta_{init}(q_0, a_1, \ldots, a_k, q_0', 0, \ldots, 0) = \mathbf{p}_1, \quad \delta_{init}(q_0, a_1, \ldots, a_k, q_0'', 0, \ldots, 0) = \mathbf{p}_2$$

We then construct the *PFA*(k) $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ by:
$Q = Q' \cup Q'' \cup \{q_0\}$
$\delta = \delta' \cup \delta'' \cup \delta_{init}$ (since functions are just sets in their essence).
$Q_{acc} = Q'_{acc}$ (accept iff. true-biased accepted)
$Q_{rej} = Q''_{rej}$ (reject iff. false-biased rejected)

[4]subject to the T_P – the set of allowed probabilities

Firstly, the constructed δ satisfies the constraints in the definition of *PFA*(k) (1.4.1), since we have joined δ-functions that used different states. Secondly, by construction $Q - (Q_{acc} \cup Q_{rej}) = Q'_{rej} \cup Q''_{acc}$, i.e., we end in a *FAILURE* if and only if either the true-biased algorithm rejects, or the false-biased algorithm accepts. For a word $w \in L$, the false-biased algorithm always accepts, and the true-biased, with probability $\leq \Lambda_1$ rejects. Hence $p_A^{FAIL}(w) \leq (\Lambda_1 \mathbf{p}_1 + 1 \cdot \mathbf{p}_2)$. Analogously, for a word $w \notin L$, the true-biased algorithm always rejects, and the false-biased, with probability $\leq \Lambda_2$ accepts. Hence $p_A^{FAIL}(w) \leq (1 \cdot \mathbf{p}_1 + \Lambda_2 \mathbf{p}_2)$. Since each word is either in L or not in L, by construction, computation of A on $w \in \Sigma^*$ is inconclusive with probability at most $\kappa = max\{\Lambda_1 \mathbf{p}_1 + \mathbf{p}_2,\ \mathbf{p}_1 + \Lambda_2 \mathbf{p}_2\}$. Thus A is a $(1-\kappa)$-correct LasVegas *PFA*(k).

□

Remark 4.3.4. The value of $\kappa = max\{\Lambda_1 \mathbf{p}_1 + \mathbf{p}_2,\ \mathbf{p}_1 + \Lambda_2 \mathbf{p}_2\}$ is interestingly identical to the value of the error bound Λ, computed in the proof that the intersection of two languages recognized by a false-biased *PFA*(k) can also be recognized with the same type of error (Theorem 3.3.8). We can therefore use the same calculations as in Remark 3.3.9, to calculate the optimal $\mathbf{p}_1, \mathbf{p}_2$ to minimize the probability of *FAILURE* (minimize κ).

Corollary 4.3.5 (True-Biased $\supsetneq$ LasVegas)**.** *The class of languages recognized by a α-correct LasVegas PFA(k) is a strict subset of the class of languages recognized by PFA(k) with bounded true-biased error.*

Proof. We already know that it is a subset from previous Lemma 4.3.1. Therefore, we only need to prove that the relation is strict, i.e., that there exists a language recognizable by a *PFA*(k) with bounded true-biased error, but not by any α-correct LasVegas *PFA*(k) (for all concievable α). We use the well known L_b (2.1).

Let $M_k = (L_b)^c$ for $b = \binom{k}{2} + 1$. A *PFA*(k) can recognize M_k with bounded true-biased error, by the Head Gap lemma 3.1.3. For the purposes of contradiction, assume that there exists a LasVegas *PFA*(k) accepting M_k. Then, by Lemma 4.3.2, there exists a *PFA*(k) accepting M_k with bounded false-biased error. Which contradicts the Hierarchy Theorem for one-sided error (3.1.1), since $b = \binom{k}{2} + 1 > \binom{k}{2}$. □

Corollary 4.3.6 (False-Biased $\supsetneq$ LasVegas)**.** *The class of languages recognized by a α-correct LasVegas PFA(k) is a strict subset of the class of languages recognized by PFA(k) with bounded false-biased error.*

Proof. Analogously to the previous corollary, we already know that it is a subset from previous Lemma 4.3.2, thus we prove that there exists a language recognizable by a *PFA*(k) with bounded false-biased error, but not by any α-correct LasVegas *PFA*(k).

Let $M_k = L_b$ for $b = \binom{k}{2} + 1$. A *PFA*(k) can recognize M_k with bounded false-biased error, by the Head Gap lemma 3.1.3. For the purposes of contradiction, assume that there exists a LasVegas $PFA(k)$ accepting M_k. Then, by Lemma 4.3.1, there exists a $PFA(k)$ accepting M_k with bounded false-biased error. Which contradicts the Hierarchy Theorem for one-sided error (3.1.1), since $b = \binom{k}{2} + 1 > \binom{k}{2}$. □

Unbounded error case

Lemma 4.3.7 (True-Biased $\supseteq$ LasVegas)**.** *For every $k \geq 1$, and every language L recognized by a LasVegas $PFA(k)$, L can also be recognized by a $PFA(k)$ with unbounded true-biased error.*

Lemma 4.3.8 (False-Biased $\supseteq$ LasVegas)**.** *For every $k \geq 1$, and every language L recognized by a LasVegas $PFA(k)$, L can also be recognized by a $PFA(k)$ with unbounded false-biased error.*

Lemma 4.3.9 (LasVegas $\supseteq$ (True-biased $\cap$ False-biased))**.** *For each $k \geq 1$, and each language L recognized by a $PFA(k)$ with unbounded true-biased error, simultaneously recognized by a $PFA(k)$ with unbounded false-biased error, The language L is also recognized by a LasVegas $PFA(k)$.*

Proof. Proofs of the above theorems are analogous to the proofs of their "bounded" versions (4.3.1, 4.3.2, 4.3.3), simply replace '$\leq \kappa$' by '< 1' and '$\geq 1-\kappa$' by '> 0'. □

Corollary 4.3.10 (True-Biased $\supsetneq$ LasVegas)**.** *The class of languages recognized by a LasVegas $PFA(k)$ is a strict subset of the class of languages recognized by $PFA(k)$ with unbounded true-biased error.*

Corollary 4.3.11 (False-Biased $\supsetneq$ LasVegas)**.** *The class of languages recognized by a LasVegas $PFA(k)$ is a strict subset of the class of languages recognized by $PFA(k)$ with unbounded false-biased error.*

Proof. Proofs of the above corollaries are analogous to the proofs of their "bounded" counterparts (4.3.5, 4.3.6), just use the unbounded versions of lemmas used in those proofs. □

4.4 Closure properties

Regular intersection

Theorem 4.4.1 (LasVegas $PFA(k)$ are closed under regular intersection)**.** *For $k \geq 1$, for every regular language $R \in \mathcal{R}$, and for every language L accepted by an (α-correct) LasVegas $PFA(k)$, there is an (α-correct) LasVegas $PFA(k)$ recognizing $L \cap R$.*

Informally. Our construction will be analogous to the proof why Monte-Carlo *PFA*(k) are closed under regular intersection (3.3.1). The new automaton A' will simulate the original *PFA*(k) A, and verify the regularity with its first head, by simulating the *DFA* $\bar{A}$ recognizing R in addition to its usual operation.

Proof. Let $A = (Q_A, \Sigma, \delta_A, q_0, Q_{acc}, Q_{rej})$ be the α-correct LasVegas automaton recognizing L. Let $\bar{A}$ by the *DFA* recognizing R. We construct the new *PFA*(k) A' as follows: The set of states of the new automaton A' will be $Q \times \bar{Q}$, and the set of accepting states will be $Q_{acc} \times \bar{F}$. Formally:

$$A' \;=\; (Q_A{\times}\bar{Q},\ \Sigma,\ \delta',\ (q_0, \bar{q_0}),\ Q_{acc}{\times}\bar{F},\ (Q_{rej}{\times}\bar{Q}) \cup (Q_A{\times}(\bar{Q}{-}\bar{F})))$$

where the new transition function δ', is constructed as follows:

$$\begin{aligned}
&(\forall q, q' \in Q_A)(\forall p \in \bar{Q})(\forall a_1, \dots, a_k \in \Sigma_\$)(\forall d_1, \dots, d_k \in \{0, 1\}) \\
&\quad \delta_A(q, a_1, \dots, a_k, q', d_1, d_2, \dots, d_k) = \mathbf{p} > 0 \\
&\qquad\qquad \Longrightarrow\ \delta'((q, p), a_1, \dots, a_k, (q', \bar{\delta}(p, a_1)), 1, d_2, \dots, d_k) = \mathbf{p} \quad \text{iff } d_1{=}1 \\
&\qquad\qquad \Longrightarrow\ \delta'((q, p), a_1, \dots, a_k, (q', p), 0, d_2, \dots, d_k) = \mathbf{p} \qquad\qquad \text{iff } d_1{=}0
\end{aligned}$$

For each computation on w of the original LasVegas automaton A, there is a corresponding computation of A' on w (*DFA* will never halt). Thus at the end, after each head arrives at \$, we simply check the final state $(q_F, \bar{q_F})$ and accept if both are accepting and reject if at least one is rejecting. Thus accepting only words in $L \cap R$.

For each word $w \in \Sigma^*$ either $p_A(w) = 0$ or $p_A^{rec}(w) = 0$. Since the only way for computation to be inconclusive, is to finish in a state whose first component is from $Q - Q_{acc} \cup Q_{rej}$, a computation is inconclusive if and only if the original computation (on A) had also been inconclusive. Thus, if A is a LasVegas *PFA*(k), A' is also. Moreover, if A is a α-correct LasVegas *PFA*(k), A' is also. □

Remark. Alternative proof might be the following: By lemmas 4.3.1 and 4.3.2, we know that some Monte-Carlo *PFA*(k)'s can accept L with both true and false-biased error with error bound κ. By Theorem 3.3.1, we then know that there do exist Monte-Carlo *PFA*(k)'s can accept $L \cup R$ with both true and false-biased error with the same error bounds – both κ. Using Lemma 4.3.3, we construct a $(1 - \kappa_2)$-correct LasVegas *PFA*(k) accepting $L \cap R$. However this automaton constructed this way is not $(1 - \kappa)$-correct, since $\kappa_2 = max\{\Lambda_1 \mathbf{p}_1 + \mathbf{p}_2,\ \mathbf{p}_1 + \Lambda_2 \mathbf{p}_2\} \leq \kappa/2$.

Complement

Theorem 4.4.2 (LasVegas *PFA*(k) are closed under complement)**.** *For every $k \geq 1$ and every language L recognized by an' α-correct LasVegas PFA(k), there exists an α-correct LasVegas PFA(k) recognizing L^c.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ be the α-correct LasVegas *PFA*(k), accepting L. We construct the corresponding LasVegas *PFA*(k) as follows: *PFA*(k) $A' = (Q, \Sigma, \delta, q_0, Q_{rej}, Q_{acc})$. The automaton A' constructed this way, is indeed a LasVegas *PFA*(k), because firstly, for automaton A, the following expression is true

$$\text{For all words } w: \text{ either } p_A(w){=}0 \text{ or } p_A^{rej}(w){=}0.$$

since we only switched accepting and rejecting states, leaving the δ-function intact, A' accepts words A would reject, and vice-versa. Hence, for A' the expression is also true. Secondly, the probability of ending in *FAILURE*, bounded from above by $\kappa(= 1-\alpha)$ on A, is affected by states not in Q_{acc}, Q_{rej}. Thus, transferring states between these two sets will not affect the probability of *FAILURE*. Hence, it is also bounded by κ. Therefore A', satisfies the definition of a $(1-\kappa)$-correct LasVegas *PFA*(k).

Since $L(A) = \{w \in \Sigma^* \mid 0 < p_A(w)\}$, A' needs to accept words that have $p_A(w) = 0$, Because of the Lemma 4.0.1, if a word has $p_A(w) = 0$ it must have $p_A^{rej}(w) > 0$. Hence it is accepted by A' (since A' accepts words, that A rejects). On the other hand, if a word has $p_A(w) > 0$ (by Lemma 4.0.1), it has $p_A^{rej}(w) = 0$, it is thus rejected by A'. □

Theorem 4.4.3. *For every $k \geq 1$ and every language L recognized by a LasVegas PFA(k), there exists a LasVegas PFA(k) recognizing L^c.*

Proof. Analogous to the proof for α-correct LasVegas *PFA*(k). □

Intersection

Consider these languages (3.1):

$$L_{vwv} = \{v * w * v \mid (v \in \{0,1\}^*) \wedge (w \in \{0,1,*\}^*)\}$$
$$L'_b = \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^*) \wedge (w_i = w_{2b+1-i}) \text{ for } 2 \leq i \leq b\}$$

Recall that $L_{vwv} \cap L'_b = L_b$.

Theorem 4.4.4 (LasVegas *PFA*(k) are not closed under intersection)**.** *There exists a language M, such that for all $k{\geq}3$, there exists a language M_k, such that there exists no LasVegas PFA(k) recognizing $M_k \cap M$, yet both M_k and M are recognized by an α-correct LasVegas PFA(k).*

Proof. Let $M = L_{vwv}$ and $M_k = L'_b$ for $b = \binom{k}{2} + 1$. By Lemma 3.3.4, M and M_k are recognized by some *DFA*(3), *DFA*(k) respectively. Therefore, by Lemma 4.0.3, M and M_k are accepted by a 1-correct LasVegas *PFA*(3), *PFA*(k) respectively.

For the purposes of contradiction, assume that there is exists a LasVegas *PFA*(k) recognizing $M_k \cap M = L_b$. That in itself contradicts the Hierarchy Theorem for LasVegas (4.2.1), since $b = \binom{k}{2}{+}1 > \binom{k}{2}$. □

Theorem 4.4.5 (LasVegas *PFA*(2) are (also) not closed under intersection)**.** *There exist two languages M_2 and M, such that there exists no LasVegas PFA(2) recognizing $M_2 \cap M$, yet both M_2 and M are recognized by a α-correct LasVegas PFA(2).*

Proof. Let $M = \{v * w_1 * w_2 * v \mid (v \in \{0,1\}^*) \wedge (w_1, w_2 \in \{0,1\}^*)\}$ and $M_2 = L'_2$. The language M is recognized by a *DFA*(2) (one head to second v, check that they are equal + check format), thus also by a 1-correct LasVegas *PFA*(2). L'_2 is also recognizable by a 1-correct LasVegas *PFA*(2), since it is recognizable by a *DFA*(2) (Lemma 3.3.4). The intersection $L'_2 \cap M = L_2$ (L_{uvvu}), however, we know to be not recognizable by no LasVegas *PFA*(2) (Hierarchy Theorem 4.2.1). □

Union

Theorem 4.4.6 (LasVegas *PFA*(k) are not closed under union)**.** *There exists a language M, such that for all $k \geq 3$, there exists a language M_k, such that there exists no LasVegas PFA(k) recognizing $M_k \cup M$, yet, both M_k and M are recognized by an α-correct LasVegas PFA(k).*

Proof. Let $M = (L_{vwv})^c$ and $M_k = (L'_b)^c$ for $b = \binom{k}{2} + 1$. By Lemma 3.3.4, M^c and $(M_k)^c$ are recognized by some *DFA*(3), *DFA*(k) respectively. By Lemma 4.0.3, and since the class of languages recognized by LasVegas *PFA*(k) is closed under complement (Theorem 4.4.2), M and M_k are accepted by a 1-correct LasVegas *PFA*(3), *PFA*(k) respectively.

For the purposes of contradiction, assume that there is a LasVegas *PFA*(k) recognizing $M_k \cup M$. Since the class of languages recognized by LasVegas *PFA*(k) is closed under complement, there is a LasVegas *PFA*(k) recognizing $(M_k)^c \cap M^c = L_b$. That contradicts the Hierarchy Theorem for LasVegas (4.2.1), since $b = \binom{k}{2}+1 > \binom{k}{2}$. □

Theorem 4.4.7 (LasVegas *PFA*(2) are (also) not closed under union)**.** *There exist two languages M_2 and M', such that there exists no LasVegas PFA(2) recognizing $M_2 \cup M'$, yet both M_2 and M' are recognized by a α-correct LasVegas PFA(2).*

Proof. Let $M' = \{v * w_1 * w_2 * v \mid (v \in \{0,1\}^*) \wedge (w_1, w_2 \in \{0,1\}^*)\}^c$, and $M_2 = (L'_2)^c$. Both are accepted by a 1-correct LasVegas *PFA*(k) since $(M')^c$ and $(M_2)^c$ are (see proof of 4.4.5), and the class of languages recognized by LasVegas *PFA*(k) is closed under complement (4.4.2). If we assume that $M' \cup M_2 = ((M')^c \cap (M_2)^c)^c = (L_{uvvu})^c$ can be recognized by a LasVegas *PFA*(k), then since the class of languages recognized by LasVegas *PFA*(k) is closed under complement L_{uvvu} is accepted also. Contradiction with the Hierarchy Theorem for LasVegas 4.2.1. □

Summary: Closure properties

The following table summarizes the theorems encountered in this section.

Class of languages recognized by	$\cap R$	^{c}	$\cap$	$\cup$
(α-correct) LasVegas *PFA*	$\checkmark$	$\checkmark$	$-$	$-$

Legend: $\checkmark$: closed under this operation. $-$: not closed under this operation.

Chapter 5

Barely-random

In this chapter, we will study a restricted version of a *PFA*(k) (a special case). The motivation is simple, we are able to prove for this "restricted" *PFA*(k) that it cannot be "amplified". By amplification we refer to a common technique used on probabilistic machines, which works by essentially repeating an existing algorithm multiple times, thus reducing the probability of error. This can easily be done on two-way machines, since one reverts to a quasi-initial configuration (remembering previous result) and restarts.

Trivially, we see that this technique of amplification cannot work on one-way multi-head *PFA*, since they are one-way (no such automaton can re-read its input). However, proving that the probability of error cannot by lowered beyond a certain point, by any other means (some novel amplification 2.0), is as always, more difficult.

Definition 5.0.1. A *barely-random PFA*(k) *over* Σ *with allowed probabilities* T_P is a *PFA*(k) A over Σ with allowed probabilities T_P, for which there exists an integer N, such that for each word $x \in \Sigma^*$ and each *computation* on x, the number of *steps of a computation* whose probability is nontrivial ($\neq 1$) is bounded from above by N.

Informally. For each word and each computation on it, the number of random decisions is finite, or that the *PFA*(k) uses a finite number of bits.

The barely-random *PFA*(k) may seem rather weak at a first glance. However, should we look closely at the chapter about Monte-Carlo *PFA*(k), we find that the *PFA*(2)'s that recognize L_b with bounded error (Head Gap lemma (3.1.3)) are all barely-random *PFA*(k). The same is true for the LasVegas *PFA*(k) constructed in Lemma 4.1.2.

Remark. Until now, all of our constructed *PFA*(k) that accept languages with bounded error (or are α-correct) are barely-random *PFA*(k).

A barely-random *PFA*(k) uses at most a finite number of random decisions/bits. Hence, we can see, that on such model only a finite amount of computations is valid.

Corollary 5.0.2. *For each barely-random PFA(k) A, there is an integer C, such that for each input word x, the number of computations of A on x is bounded by C.*

Proof. Since A is a barely-random *PFA*(k), the number of steps whose probability is not 1 is bounded by a constant. Let that bound be N. Moreover, let B be the bound on a branching factor, i.e.

$$(\forall q \in Q)(\forall a_1, \dots, a_k \in \Sigma_\$) : \; B \geq \sum_{p \in Q,\; d_1, \dots, d_k \in \{0,1\}} [\delta(q, a_1, \dots, a_k, p, d_1, \dots, d_k) > 0]^1$$

(B always exists, since there is finitely many values to consider – δ is finitely encoded) Then, since the automaton chooses at most N times and chooses among at most B possibilities, we see that on each word x, there are at most B^N valid computations. □

Theorem 5.0.3 (Barely-random *PFA*(k) $\subseteq$ *DFA*($C{\cdot}k$))**.** *For every barely-random PFA(k) A, and there is an integer C, such that:*

- *for each λ there exists a DFA($C{\cdot}k$), recognizing $L(A, \lambda)$ if A is Monte-Carlo.*
- *there exists a DFA($C{\cdot}k$), recognizing $L(A)$ if A is LasVegas.*

Proof. By the previous corollary, we know that there exists, for every *barely-random PFA*(k) A, an upper bound (C) on the number of computations on A. Therefore, we can simulate all of those computations. To do that, $k \cdot C$ heads will suffice, since for each computation we have brand new k heads. Each of the simulated computations will either accept, reject, end in *FAILURE*, or loop forever (which we can detect – the automaton did not move any head in the last $|Q| + 47$ configurations and did not make any randomized decision).

Since we know the probability with which each computation would have happened on A, we can compute the probability of accepting/rejecting that word ($p_A(x)$, $p_A^{rej}(x)$), i.e., the automaton will sum the probabilities of computations that accepted (and another sum for those that rejected). Since there are at most C computations, there are at most 3^C possible outcomes (which computation accepted/rejected/failed), thus there are at most 3^C different sums of probabilities. Since it is finitely many, we can encode each of these situations into some state. Therefore, we can decide whether the input word belongs into L or L^c.

□

We now consider this language (see 2.1) from the corollary of Yao and Rivest [YR78]:

$$L' = \{w_1 * w_2 * \cdots * w_{2b} \mid (b{\geq}1) \wedge (w_i \in \{0,1\}^* \text{ for } 1 \leq i \leq 2b) \wedge (\exists i)(w_i \neq w_{2b+1-i})\}$$

We may have been unable to prove that a Monte-Carlo *PFA*(k) with bounded error can never recognize L', however, for barely-random *PFA*(k), we can prove that.

[1]Here, we are using a so-called Iverson backet/Iverson notation, the bracket is equal to 1 if the expression inside it is true, and 0 otherwise.

Corollary 5.0.4. *For all $k \geq 1$, the languages L' and $(L')^c$ cannot be recognized by any barely-random PFA(3) with bounded true-biased nor bounded false-biased error.*

Proof. For the purposes of contradiction, let us assume that there exists such barely-random *PFA*(k) accepting L' with bounded true-biased error or with bounded false-biased error. Then, by the previous theorem (5.0.3), we can construct a *DFA*$(C{\cdot}k)$ accepting L'. Contradicting the Corollary 2.1.3.

The proof for $(L')^c$ is analogous, when we recall that $\mathscr{L}(DFA(k))$ are closed under complement, or that we have proven the Complement lemma 3.0.7. □

5.1 Choose-compute normal form

Definition 5.1.1. A *barely-random PFA*(k) A is in a *choose-compute* form, if and only if there exists a set $Q_0 \subseteq Q$ (set of initial choosing states) such that $Q_0 \cap Q_{acc} = \emptyset$, and $(\forall q, p \in Q{-}Q_0)(\forall q', p' \in Q_0)(\forall a_1, \ldots, a_k \in \Sigma_\$)(\forall d_1, \ldots, d_k \in \{0,1\})$:

$$\delta_A(q', a_1, \ldots, a_k, p', d_1, \ldots, d_k) = 0 \quad \text{unless } d_1{=}\ldots{=}d_k{=}0$$
$$\delta_A(q', a_1, \ldots, a_k, p, d_1, \ldots, d_k) = 0 \quad \text{unless } d_1{=}\ldots{=}d_k{=}0$$
$$\delta_A(q, a_1, \ldots, a_k, p', d_1, \ldots, d_k) = 0$$
$$\delta_A(q, a_1, \ldots, a_k, p, d_1, \ldots, d_k) \in \{0,1\}$$

Informally. *PFA*(k) A is in choose-compute form if:
To move probabilistically $(\delta(\cdot){\neq}0;1)$, A must not move heads, and must start in $q \in Q_0$.
Once in a state other than from Q_0, we cannot move "back" into a state from Q_0.
All movement between states other than states in Q_0 is deterministic $(\delta(\cdot) = 0|1)$.

Definition 5.1.2. A *barely-random PFA*(k) A is in a *strong choose-compute* form if and only if it is in a *choose-compute* form where $Q_0 = \{q_0\}$.

Remark. Strong choose-compute form is actually the more intuitive form of the two. The algorithm, in the first step, chooses which computation to undertake. However, this form is too strict to be a normal form for even a barely-random *PFA*(k) with coin-flips. We have therefore defined the choose-compute form (more general), which allows us to split the decision of which computation to choose into multiple steps.

We now prove that these forms are normal forms for some barely-random *PFA*(k). Whether or not one can construct a corresponding *PFA*(k), depends on the allowed probabilities of the *PFA*(k). For example, to construct a corresponding strong choose-compute form of a barely-random *PFA*(k), we need $(T_P{-}\{0\},\cdot)$ to be a monoid[2].

[2]Multiplication must be an associative binary operation on $T_P{-}\{0\}$ and $T_P{-}\{0\}$ must contain 1 (the identity element).

Theorem 5.1.3 (Choose-compute form is a normal form for barely-random)**.** *For each barely-random PFA(k) A with allowed probabilities $\{0, \frac{1}{2}, 1\}$, there is a corresponding barely-random PFA(k) A' with allowed probabilities $\{0, \frac{1}{2}, 1\}$, in a choose-compute form, such that they are equivalent.*[3]

Informally. We build an automaton working in two phases, "choose" and "compute". In the "choose" part, it fills a buffer of coin-flips, via truly random decisions, from which it will later read the stored "randomness" when needed in the "compute" part of the new computation, simulating the original automaton.

Proof. Let $A = (Q_A, \Sigma, \delta_A, q_0, Q_{acc}, Q_{rej})$. By Definition 5.0.1, we know that for a barely-random *PFA*(k), the number of steps of a computation whose probability is not 1 is bounded by a constant, let the bound be N (at most N randomized decisions).

The new automaton A', without reading the input, pre-computes ("chooses") the randomized decisions into a buffer $buf \in \{1,2\}^{\leq N}$, which it then stores into each state, that is $q_{i[buf]} \in Q'$, for q_i in Q_A. Moreover, for each state q_i in Q_{acc} (resp. Q_{rej}), the corresponding buffered state $q_{i[buf]}$ is in Q_{acc} (resp. Q'_{rej}).

Then A' simulates the computation of A, such that whenever it encounters a situation, where A should pick between 2 options via a randomized coin-flip, the new automaton A' will choose the next configuration via reading the coin-flip from the buffer. If read 1, choose the first outcome, if read 2, choose the second (based on a lexicographical order on outcomes $(p, d_1, \ldots, d_k)$). We never run out of coin-flips, because the *PFA*(k) A is barely-random, i.e., it may do at most N randomized decisions.

Formally, $A' {=} (Q', \Sigma, \delta', q_{choose[]}, Q'_{acc}, Q'_{ref})$ where $Q' {=} \{q_{[buf]} \mid q \in Q,\ \ buf \in \{1,2\}^{\leq N}\}$

The δ' function first builds the buffer ("choose")

$(\forall i \in \{0 \ldots N-1\})(\forall buf \in \{1,2\}^i)$:[4]

$$\begin{aligned} \delta'(q_{choose[buf]}, a_1, \ldots, a_k, q_{choose[0buf]}, 0, \ldots, 0) &= 1/2 \\ \delta'(q_{choose[buf]}, a_1, \ldots, a_k, q_{choose[1buf]}, 0, \ldots, 0) &= 1/2 \end{aligned} \tag{5.1}$$

Switch to second phase:

$$(\forall buf \in \{1,2\}^N) : \quad \delta'(q_{choose[buf]}, a_1, \ldots, a_k, q_{0[buf]}, 0, \ldots, 0) = 1$$

Finally, the automaton will simulate the original automaton ("compute")

$(\forall q, p \in Q_A)(\forall buf \in \{1,2\}^{\leq N})(\forall a_1, \ldots, a_k \in \Sigma_\$)(\forall d_1, \ldots, d_k \in \{0,1\})$

$(\forall\ (p_1, d_{11}, \ldots, d_{k1}) <_{lex} (p_2, d_{12}, \ldots, d_{k2}))$:[5]

[3] see Definition 1.6.1.

[4] $\{1,2\}^0$ is $\{\varepsilon\}$, i.e., the state is $q_{choose[]}$

[5] $<_{lex}$ represents a comparison in some lexicographic ordering.

$$\begin{aligned}
\delta_A(q,a_1,\ldots,a_k,\ p,\ d_1,\ldots,\ d_k) = 1 \quad &\Longrightarrow \delta'(\ q_{[buf]},a_1,\ldots,a_k,\ p_{[buf]},\ d_1,\ldots,\ d_k){=}1\\
\delta_A(q,a_1,\ldots,a_k,p_1,d_{11},\ldots,d_{k1}) = 1/2 \quad &\Longrightarrow \delta'(q_{[1buf]},a_1,\ldots,a_k,p_{1[buf]},d_{11},\ldots,d_{k1}){=}1\\
&\ \delta'(q_{[1buf]},a_1,\ldots,a_k,p_{2[buf]},d_{12},\ldots,d_{k2}){=}0\\
\delta_A(q,a_1,\ldots,a_k,p_2,d_{12},\ldots,d_{k2}) = 1/2 \quad &\Longrightarrow \delta'(q_{[2buf]},a_1,\ldots,a_k,p_{1[buf]},d_{11},\ldots,d_{k1}){=}0\\
&\ \delta'(q_{[2buf]},a_1,\ldots,a_k,p_{2[buf]},d_{12},\ldots,d_{k2}){=}1
\end{aligned} \tag{5.2}$$

The correctness of this construction follows from the following: Firstly, the sets of accepting, rejecting and "failing" ($\notin Q_{acc} \cup Q_{rej}$) states are the same (disregarding the buffer in the states). Secondly, each computation on A', is only a computation on A with prepended "choose" phase, which has the same probability of occurrence as its corresponding computation on A, just the random decisions are done elsewhere. □

Theorem 5.1.4 (Choose-compute form is a normal form for barely-random)**.** *For each barely-random PFA(k) A with allowed probabilities $[0,1] \cap \mathbb{Q}$, there is a corresponding barely-random PFA(k) A' with allowed probabilities $[0,1] \cap \mathbb{Q}$, in a choose-compute form, such that they are equivalent.*

Proof. The proof is analogous to the previous, however, since T_P is $[0,1] \cap \mathbb{Q}$, certain modifications are in order. We first do a *gcd*[6] of the denominators that appear in the nonzero probabilities in δ_A. Let G be the *gcd* (our goal is to "rewrite" the fractions, so that they all have the same denominator ($\frac{n_i}{G}$). e.g. $\frac{1}{6}, \frac{3}{6}, \frac{2}{6}$ instead of $\frac{1}{6}, \frac{1}{2}, \frac{1}{3}$). We then build a buffer of "G-sided dice rolls", $buf \in \{1 \ldots G\}^{\leq N}$. The outcomes are sorted lexicographically, and when the need to choose arises, A' picks the i-th outcome (its probability $\frac{n_i}{G}$), if and only if it reads (from the buffer) a number R, [7] such that:

$$\sum_{j=1}^{i-1} n_j < R \leq \sum_{j=1}^{i} n_j$$

The rest of the construction is analogous to the above proof.

Example (for brevity only nonzero entries are shown):

$$\begin{aligned}
\delta_A(q,a_1,\ldots,a_k,p_1,d_{11},\ldots,d_{k1}) = 1/6 \;&\Longrightarrow\; \delta(q_{[\mathbf{1}buf]},a_1,\ldots,a_k,p_{1[buf]},d_{11},\ldots,d_{k1}) = 1\\
\delta_A(q,a_1,\ldots,a_k,p_2,d_{12},\ldots,d_{k2}) = 3/6 \;&\Longrightarrow\; \delta(q_{[\mathbf{2}buf]},a_1,\ldots,a_k,p_{2[buf]},d_{12},\ldots,d_{k2}) = 1\\
&\; \delta(q_{[\mathbf{3}buf]},a_1,\ldots,a_k,p_{2[buf]},d_{12},\ldots,d_{k2}) = 1\\
&\; \delta(q_{[\mathbf{4}buf]},a_1,\ldots,a_k,p_{2[buf]},d_{12},\ldots,d_{k2}) = 1\\
\delta_A(q,a_1,\ldots,a_k,p_3,d_{13},\ldots,d_{k3}) = 2/6 \;&\Longrightarrow\; \delta(q_{[\mathbf{5}buf]},a_1,\ldots,a_k,p_{3[buf]},d_{13},\ldots,d_{k3}) = 1\\
&\; \delta(q_{[\mathbf{6}buf]},a_1,\ldots,a_k,p_{3[buf]},d_{13},\ldots,d_{k3}) = 1
\end{aligned}$$

□

[6] greatest common divisor

[7] R as in dice-**R**oll.

Theorem 5.1.5 (Strong choose-compute form is a normal form for barely-random)**.** *For a barely-random PFA*(k) A *with allowed probabilities* $[0,1]\cap\mathbb{Q}$*, there exists a barely-random PFA*(k) A' *with allowed probabilities* $[0,1]\cap\mathbb{Q}$*, in a strong choose-compute form, such that they are equivalent.*

Proof. By the previous theorem, for A, there is a barely-random $PFA(k)$ A' with allowed probabilities $[0,1]\cap\mathbb{Q}$, in a choose-compute form. Since multiplication is a binary operation on the set $(0,1]\cap\mathbb{Q}$, and because we can compute the probability of arriving in a state $q_{0[buf]}$ $(=\frac{1}{G^N})$, instead of building the buffer incrementally, we do it in one step, $(\forall buf{\in}\{1,2\}^N)$ $\delta(q_{choose[]}, a_1, \ldots, a_k, q_{0[buf]}, 0, \ldots, 0){=}1/G^N$. That way, we can reduce the number of states in Q_0 to one by a small change in A'. □

Remark. One cannot do this for $PFA(k)$ with coin-flips ($T_P = \{0, \frac{1}{2}, 1\}$). Since to jump to a full buffer, one needs to do a step with probability $1/2^N$. (T_P must be monoid.)

5.2 Barely-random cannot be amplified

Consider the following language

$$\text{Let } L_b = \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^*) \wedge (w_i = w_{2b+1-i}) \text{ for } 1 \le i \le 2b\}$$

Theorem 5.2.1. *Barely-random PFA*(2) *with one-sided false-biased error, can accept* L_b *with error bound* Λ*, if and only if* $\Lambda \ge 1{-}\frac{1}{b}$.

Remark. In another words, a barely-random $PFA(2)$ cannot accept L_b with $\Lambda < 1{-}\frac{1}{b}$.

Informally. We strive to use the "cutting and pasting" technique. Therefore we define location, pattern and mistake as in chapter 2 (Lemma 2.0.2) {A quick reminder: location: in which word is each head; pattern: subsequence of the (one deterministic) computation, only taking configurations where you change location; mistake: location that does not occur in whole pattern}. We then continue as follows:

- Notice that for each pattern there may be mistake on it.
- Notice that there are b "notable" mistakes possible (on each pair of sub-words w_i, w_{2b-i+1}).
- We prove, that for each pattern there is only 1 "notable" mistake which we cannot exploit (mistake which is not that pattern.)
- We show that we can find two distinct words with the exact same pattern-m-tuple (pattern of each of the m sub-procedures, among which the $PFA(k)$ chooses).
- Notice that in a barely random $PFA(k)$ choosing between sub-procedures, each of those m sub-procedures occur with certain probability.

- We analyze which mistake we "cannot exploit" the least (which mistake occurs with the highest probability).
- We exploit that mistake – by "cutting and pasting" argument.

Proof. Let A_0 be a barely-random $PFA(2)$ recognizing L_b with false-biased error with error bound Λ. We construct an equivalent barely-random $PFA(2)$ A in *choose-compute form.* Since the constructed equivalent barely-random $PFA(2)$ in choose-compute form accepts/rejects words with the same probability as the original (Theorem 5.1.4), A recognizes L_b with false-biased error with error bound Λ also.

Since every $PFA(2)$ in choose-compute form effectively chooses between a number of deterministic algorithms (with given probability), we count them and denote the number of deterministic algorithms as m. Moreover, for each such deterministic computation, we assign a number p_i representing the probability of choosing that particular computation.

We define a *location* of a configuration[8] (q, w, o_1, o_2) as a 2-tuple of integers $(\lceil o_1/(n+1)\rceil, \lceil o_2/(n+1)\rceil)$. Then, for each deterministic computation[9] $c_1(w), c_2(w), \ldots, c_{l_w}(w)$ define a *pattern* of a word as a subsequence of that computation $d_1(w), d_2(w), \ldots, d_{l'_w}(w)$ obtained by first taking $c_1(w)$, and then all subsequent $c_i(w)$ such that $location(c_i(w)) \neq location(c_{i+1}(w))$. We define a *mistake* of a *pattern* as the *location* which could be valid, but does not occur in the pattern (i.e., the indices of sub-words, in which the heads are never simultaneously). Moreover, a *mistake* i is a mistake, 2-tuple $(i, 2b-i+1)$. It is a notable mistake informing us that during the computation, the heads are never simultaneously in w_i and w_{2b-i+1}.

We prove the following lemma, stating that only one mistake out of mistakes $1, \ldots, b$ can be "detected" by a one-way deterministic 2-head computation.

Lemma 5.2.1.1. *If a pattern of word $w \in L_b$, on a deterministic computation of a $PFA(2)$, does not have mistake i, it has a mistake j for each $j \in \{1, \ldots, b\} - \{i\}$.*

Proof. Assume that a 2-head automaton during a computation on $w \in L_b^n$ has its heads simultaneously in w_i and w_{2b-i+1}. This all is implied:

Firstly, both heads need to move beyond the first i sub-words to get to w_i (w_{2b-i+1}). However, the none of these heads may not go beyond w_{2b-i+1} before the other arrives at w_i. (Otherwise they would never be simultaneously on w_i, w_{2b-i+1}). Therefore, on sub-words $w_j, w_{2b-j+1},\ j \in \{0, \ldots, i-1\}$ the heads will never be simultaneously.

[8] configuration of a barely-random $PFA(2)$.

[9] deterministic computation is a computation on $PFA(k)$, where the probabilities of steps are =1. The choose-compute form, even though simulating a randomized automaton, is deterministic in the "compute" part of the computation.

Secondly, one head needs to go over all sub-words between w_i and w_{2b-i+1}, while the other does not go beyond w_i. (Otherwise, they would never be simultaneously on w_i, w_{2b-i+1}.) Therefore, on sub-words $w_j, w_{2b-j+1},\ j \in \{i+1 \dots b\}$ the heads will never be simultaneously.

Hence, we have proven that if the automaton wants to have heads simultaneously on w_i and w_{2b-i+1} (not have mistake on w_i),it will never have heads simultaneously on w_j and w_{2b-j+1} (will have mistake on w_j) for each $j \in \{1 \dots b\} - \{i\}$. $\square$

Just as we did in other proofs inspired by the cutting-and-pasting technique (from chapter 2), we assign "something" to each word, in order to be able to categorize them. Thus, to each word w from L_b^n, we assign the following:

- pattern-m-tuple $(d_1, d_2, \dots, d_m)$ consisting of patterns of w on each of the m deterministic computations in A (among which the choose-compute $PFA(k)$ picks).
- mistake-b-tuple $(\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_b)$ consisting of probabilities of doing a computation on w with a mistake on w_i (for corresponding $\mathbf{p}_i$).

Since we know the probability of doing each deterministic sub-algorithm, we know the probability $(p_1, \dots, p_m)$ with which, which pattern will occur. Moreover, by construction of a choose-compute $PFA(k)$, $1 = \sum_{j=1}^{m} p_j$. We can compute each $\mathbf{p}_i$ in the mistake-b-tuple via the following sum:

$$\mathbf{p}_i = \sum_{j=1}^{m} p_j\ [i \text{ is a mistake on the pattern } d_j]^{10}$$

In order to be able to walk the final step in the upcoming chain of steps, we state a corollary of the previous lemma (5.2.1.1):

$$\sum_{i=1}^{b} \mathbf{p}_i \geq (b-1).$$

We prove this by looking at the mistake-b-tuple or the sum in a different way: Initially, nothing is "checked", thus each mistake has probability of occurrence equal to 1 ($\sum_{i=1}^{b} \mathbf{p}_i = b$). However, each possible computation may "check" something, lowering the probability of certain mistake. By Lemma 5.2.1.1, each computation can visit at most one pair w_i, w_{2b-i+1} simultaneously. Hence, each one computation, occurring with its probability p, decreases the probability of $\mathbf{p}_i$ by at most p.

By adding up (or subtracting down) all possible computations with their probabilities, we see that the outcomes are the correct values of mistake-b-tuple $(\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_b)$. Moreover, we see that from the initial sum (b), at most $\sum_{j=1}^{b} p_j (= 1)$ was subtracted. Hence $\sum_{i=1}^{b} \mathbf{p}_i \geq b - \sum_{j=1}^{b} p_j = b - 1$.

[10] we use Iverson bracket/notation again, the bracket is 1 iff. the expression inside it is true, else 0.

Let $L_b^n = \{w_1 * w_2 * \cdots * w_{2b} \mid (w_i \in \{0,1\}^n) \wedge (w_i = w_{2b+1-i}) \text{ for } 1 \leq i \leq 2b\}$ trivially $L_b \supseteq L_b^n$, thus words from this set must have an accepting computation. Since the length of any pattern $l'_w \leq 2(2b+1)$, (each head must go through all $2b$ sub-words and \$) the number of possible patterns ρ is at most $(|Q| \cdot (2b(n+1))^2)^{2(2b+1)}$ [11]. We divide words in L_b^n into ρ^m sets, depending on the word's pattern-m-tuple. $|L_b^n| = 2^{bn}$, thus by the pigeonhole principle[12], at least one of those sets contains at least $|L_b^n|/\rho^m = 2^{bn}/\rho^m$ words. Let S_0 be that set.

All words from S_0 have the same pattern-m-tuple, (have the same pattern on the same sub-computation). Therefore, all words in S_0 have the same mistake-b-tuple. Let i $\mathbf{p}_i$ be the index of the greatest element (in case of a tie, pick the smaller index $(\forall j)\mathbf{p}_j \leq \mathbf{p}_i$). We then classify words from S_0 into classes based on the string

$$w_1 * \cdots * w_{i-1} * w_{i+1} * \cdots * w_b \tag{5.3}$$

Informally. We classify based on the whole word, except the two corresponding sub-words on which the mistake is made with the greatest probability ($\mathbf{p}_i \rightsquigarrow w_i, w_{2b-i+1}$).

By the pigeonhole principle, because the number of possible classification strings (see 5.3) is at most $2^{(b-1)n}$, there exists a class S_1, such that it contains at least $|S_0|/2^{(b-1)n} = 2^{bn-(b-1)n}/\rho^m = 2^n/\rho^m$ words. Moreover, since ρ is at most polynomial in n, ρ^m is too. We can thus pick big enough n, so that S_1 contains at least two distinct words x, y.

We have two words x, y, both from $S_1 \subseteq S_0 \subseteq L_b^n$. Thus x, y are both from L_b^n. Also since x, y are both from S_0, they have the same patterns on each of the m deterministic computations. Moreover, many of these computations have mistake i. Lastly x, y are both from S_1 hence they differ only on w_i, w_{2b-i+1}. Therefore we can construct z by taking x and replacing x_{2b-i+1} by y_{2b-i+1}. Thus satisfying all that is required for the applicability of the cut-and-paste argument (Lemma 2.0.2), which we repeat for completeness.

For each pattern in the pattern-m-tuple with mistake i individually: We construct an accepting computation for A on z by selecting successive blocks from $\{c_j(x)\}$, except when A during that block would be reading $x_{2b-i+1} (\neq z_{2b-i+1})$, in which case we select the corresponding block from $\{c_j(y)\}$ instead (since $y_{2b-i+1} = z_{2b-i+1}$). This sequence forms a valid computation for z since the last configuration in block i for either $\{c_j(x)\}$ or $\{c_j(y)\}$ yields $d_{j+1}(x)$ as the next configuration of A, and we already know that, A is never reading sub-words z_i and z_{2b-i+1} simultaneously. Therefore, at any instant, A behaves exactly as it would if the input had been one of x or y.

[11] $(\#_{configurations})^{\#_{heads}(\#_{subwords}+1)} = (\#_{states} \cdot (|w|+1)^{\#_{heads}})^{\#_{heads}(\#_{subwords}+1)}$

[12] also known as Dirichlet's box principle

By our previous analysis, we know that with probability $\mathbf{p}_i$, the *PFA*(2) A will run one of the algorithms with a mistake on i. Therefore A accepts z with probability at least $\mathbf{p}_i$. Since we know that $z \notin L_b$, we have an example of a word z such that z and is accepted with probability at least $\mathbf{p}_i$, and $z \in (L_b)^c = \{x \mid x \in \Sigma^*,\ p_A(x) \leq \Lambda\}$, Thus $\mathbf{p}_i \leq \Lambda$. Moreover, because $\sum_{j=1}^{b} \mathbf{p}_j \geq b-1$, and because we chose the greatest $\mathbf{p}_j$ from the mistake-b-tuple, we know that $\mathbf{p}_i \geq (b-1)/b$.

We therefore derive the following conclusion: Should the language L_b be accepted by a barely-random *PFA*(2) with bounded one-sided false-biased error, the error bound Λ is at least $(b-1)/b$.

To prove the if direction, we look at the construction in Head Gap lemma (3.1.3), where we construct a *PFA*(k) accepting L_b with false-biased error with an error bound $\Lambda = (b-1)/b$. We just note that the *PFA*(k) constructed there is also barely-random. □

We state the following as a theorem despite the fact that the proof is rather simple, since it shows that barely-random *PFA*(k) with true-biased error, cannot be amplified in general the same way that their counterparts with false-biased error cannot.

Theorem 5.2.2. *A barely-random PFA*(2) *with one-sided true-biased error, can accept* $(L_b)^c$ *with error bound* Λ*, if and only if* $\Lambda \geq 1-\frac{1}{b}$.

Proof. For the purposes of contradiction, assume that there is a barely-random *PFA*(2) accepting $(L_b)^c$ with true-biased error with error bound $\Lambda < \frac{b-1}{b}$. Then, by the Complement lemma (3.0.7), we construct a barely-random *PFA*(2) accepting $(L_b)^{cc} = L_b$ with false-biased error with error bound Λ, and we know that $\Lambda < \frac{b-1}{b}$. Which contradicts the previous theorem. (We see that the construction is only swapping states between Q_{acc} and Q_{rej}. Hence the *PFA*(k) stays barely-random if it was beforehand.) The if direction follows from the Head Gap lemma 3.1.3 again. □

Chapter 6

Conclusion

This thesis explored the model of probabilistic one-way multi-head finite automata. In the first chapter, we formally defined the various types of errors, with which a Monte-Carlo automaton can accept a language. We also defined there the one-way multi-head probabilistic finite automaton *PFA*(k), and later, using the theory of Markov chains, proved that *PFA*(k) have an ε-free normal form, i.e., a normal form where at every step of computation the automaton has to advance at least one head. In the second chapter, we introduced the Cutting-and-pasting technique from the proof of the Hierarchy Theorem by Yao and Rivest [YR78] and used it on an example.

Then, we proceeded to explore the Monte-Carlo *PFA*(k) accepting languages with one-sided error (true-biased and/or false-biased), for which we proved that analogous Hierarchy Theorem holds for *PFA*(k) with one-sided error. We have also proven analogous corollaries for *PFA*(k), as Yao and Rivest have shown for *FA*(k) [YR78]. However, for the general case of *PFA*(k), we have been unable to prove whether or not they can recognize the language L' with bounded error. ($L' = \{w_1 * w_2 * \cdots * w_{2b} \mid (b{\geq}1) \wedge (w_i \in \{0,1\}^*$ for $1 \leq i \leq 2b) \wedge (\exists i)(w_i \neq w_{2b+1-i})\}$). We have only proven that it cannot be recognized by a barely-random *PFA*(k). We leave it as an open problem.

Interestingly, the language that was used to show this hierarchy is recognizable with only 2 heads with a *PFA*(k) with the opposite error. This observation helped us prove numerous properties of the classes of languages recognized by Monte-Carlo *PFA*(k) with one-sided error, such as relations between them or with $\mathscr{L}(DFA(k))$, or their closure properties (we considered union, intersection, complement, and intersection with regular languages).

Looking at LasVegas randomization, we first wanted to find a language recognizable by LasVegas *PFA*(k), yet not by any *DFA*(k). We found it in observing that LasVegas can accept languages, that are a union of disjunct languages, such that one can distinguish the formats of these languages by only a regular expression. (one may not know if the word belongs "there", but is certain that the only language into which it can belong is "this".) We have then shown an analogous Hierarchy Theorem for LasVegas. Also, we have proven an analogous result to the one from the theory of Turing machines, that "LasVegas = true-biased ∩ false-biased", and that LasVegas is a strict subset of both Monte-Carlo *PFA*(k) accepting with true, and false-biased error. The results about the various relations between classes of languages recognized by various randomizations of one-way multi-head automata can be summarized in the following figure:

				PFA(k) bounded true-biased	$\subseteq$	*PFA*(k) unbounded true-biased	$=$ *NFA*(k)
			$\subsetneq$				
DFA(k)	$\subseteq$	LasVegas *PFA*(k)		$\not\subseteq$ $\not\supseteq$	$\not\subseteq$ $\not\supseteq$ $\not\subseteq$ $\not\supseteq$	$\not\subseteq$ $\not\supseteq$	
			$\subsetneq$				
				PFA(k) bounded false-biased	$\subseteq$	*PFA*(k) unbounded false-biased	

Table 6.1: Relations between classes of languages recognized with k-heads

For LasVegas *PFA*(k), we have also we have explored their closure properties (we considered the same operations). The following table summarizes the closure properties that we have proven for the LasVegas and Monte-Carlo models of *PFA*(k).

Class of languages recognized by	$\cap R$	^{c}	$\cap$	$\cup$
(α-correct) LasVegas *PFA*	$\checkmark\Lambda$	$\checkmark\Lambda$	$-$	$-$
PFA with (un)bounded true-biased error	$\checkmark\Lambda$	$-$	$-$	$\checkmark$
PFA with (un)bounded false-biased error	$\checkmark\Lambda$	$-$	$\checkmark$	$-$

Legend: $\checkmark$: closed under this operation. Λ: construction keeps the error bound.
$-$: not closed under this operation.

The last chapter considered a version of *PFA*(k), where the automaton is allowed to make at most a finite amount of randomized decisions, namely barely-random *PFA*(k). An interesting observation is, that all *PFA*(k) that we constructed in this thesis and were "bounded" (Monte-Carlo accepting with bounded error or α-correct LasVegas), were in fact barely-random. What is more, we have not been able to come up with a language, for which we could construct a "bounded" *PFA*(k), for which we could not easily construct a barely-random *PFA*(k) recognizing it.

We did show that barely-random *PFA*(k) have a normal form, such that all randomized decisions happen before any one head moves (as if without reading the input). Then we proceeded to prove, that for certain class of languages we can show a lower bound for error with which a barely-random *PFA*(k) can recognize given language, i.e., that barely-random $PFA(k)$ cannot be amplified. This result feels intuitive since we are considering one-way automata, i.e., automata that cannot re-read their input. However, we have been unable to prove this for the general case of $PFA(k)$. We wonder whether the right approach is finding proof for the lower bound, or proving that barely-random $PFA(k)$ are in fact a normal form of "bounded" $PFA(k)$. We leave it as an open problem.

A possible continuation of our work might be exploring Monte-Carlo $PFA(k)$ with two-sided error. Another possible way is to prove the closure properties of the remaining AFL operations, such as homomorphism, inverse homomorphism, concatenation, or iteration. Looking at LasVegas $PFA(k)$, a viable question might be, whether or not, there is a difference in expressive power between LasVegas $PFA(k)$ and $DFA(k)$, for $k \geq 3$. We have only answered this for $k = 2$.

Bibliography

[Fre81] Rusins Freivalds. Probabilistic two-way machines. In Jozef Gruska and Michal Chytil, editors, *Mathematical Foundations of Computer Science 1981, Strbske Pleso, Czechoslovakia, August 31 - September 4, 1981, Proceedings*, volume 118 of *Lecture Notes in Computer Science*, pages 33–45. Springer, 1981.

[HKM09] Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-head finite automata: Characterizations, concepts and open problems. *Electronic Proceedings in Theoretical Computer Science*, 1:93–107, jun 2009.

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.

[KS60] JG Kemeny and JL Snell. Finite markov chains, van nostrand. *Princeton, New Jersey*, 1960.

[Mac95] Ioan I. Macarie. Multihead two-way probabilistic finite automata. In Ricardo A. Baeza-Yates, Eric Goles Ch., and Patricio V. Poblete, editors, *LATIN '95: Theoretical Informatics, Second Latin American Symposium, Valparaíso, Chile, April 3-7, 1995, Proceedings*, volume 911 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 1995.

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

[RS59] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959.

[YR78] Andrew Chi-Chih Yao and Ronald L. Rivest. k+1 heads are better than k. *J. ACM*, 25(2):337–340, 1978.