Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Drunken finite automata

Bachelor Thesis

2024
Emma Pásztorová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Drunken 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, 2024
Emma Pásztorová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Emma Pásztorová
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: Drunken finite automata
Opité konečné automaty

Anotácia: Cieľom je študovať modifikáciu dvojsmerných konečných automatov, ktoré nevedia ovplyvniť pohyb hlavy. Namiesto toho sa hlava hýbe podľa náhodného procesu ``opitého námorníka'' a posledný smer pohybu hlavy je vstupom prechodovej funkcie automatu. Kedykoľvek automat dosiahne koncový oddeľovač vstupného slova, musí prejsť do príslušného akceptačného alebo zamietacieho stavu. Práca sa bude venovať otázkam vzťahu opitých automatov (deterministických aj nedeterministických) k iným typom konečných automatov. Prvá otázka bude zistiť, akú triedu jazykov opité automaty rozpoznávajú a ďalej študovať stavovú zložitosť v porovnaní s inými typmi automatov.

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

Dátum zadania: 18.10.2023

Dátum schválenia: 18.10.2023 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Emma Pásztorová
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: Drunken finite automata

Annotation: The goal is to study a modification of two-way finite automata that have no control about the direction of the head movement. Instead, the head is moved according the drunkard's walk random process, and the direction of the last movement is the input to the transition function of the automaton. Anytime the automaton reaches the end delimiter of the input word it must enter a correct accepting of rejecting state. Relations of drunken automata (deterministic or non-deterministic) with other types of finite automata are to be studied. In particular, what is the class of languages recognized by drunken automata, and what is the state complexity with relation to other types of automata.

Supervisor:	prof. RNDr. Rastislav Kráľovič, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	18.10.2023
Approved:	18.10.2023

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I would like to thank my supervisor, Prof. RNDr. Rastislav Královič, PhD., for his invaluable guidance and support throughout my thesis.

Abstrakt

Koncept stavovej zložitosti je kľúčový pre porozumenie a kategorizáciu výpočtových problémov, ako aj pre meranie a porovnávanie efektívnosti rôznych výpočtových modelov. Významnou oblasťou štúdia v tejto oblasti je porovnanie stavovej zložitosti dvojsmerných deterministických a nedeterministických konečných automatov, ako aj jednosmerných deterministických a nedeterministických konečných automatov. Pri jednosmerných konečných automatoch je známe, že existuje postupnosť jazykov, kde jednosmerný deterministický automat potrebuje na ich akceptovanie exponenciálne viac stavov ako jednosmerný nedeterministický automat. Pre dvojsmerné automaty je však táto otázka stále otvorená. Preto sa vedci sústreďujú na skúmanie modifikácií týchto dvojsmerných automatov. V tejto práci sa hlavne sústredíme na stavovú zložitosť modelu zvaného opité konečné automaty, ktoré predstavujú modifikáciu dvojsmerných automatov, ktoré nemajú kontrolu nad smerom pohybu hlavy. Namiesto toho sa hlava pohybuje podľa náhodného procesu opitého chodca a smer posledného pohybu je vstupom do prechodovej funkcie automatu. Kedykoľvek automat dosiahne koncový znak vstupného slova, musí byť v správnom akceptačnom alebo zamietacom stave. V tejto práci najprv ukážeme, že akceptujú množinu regulárnych jazykov a potom sa sústredíme na skúmanie stavovej zložitosti ich deterministickej a nedeterministickej verzie v porovnaní s jednosmernými konečnými automatmi. Špeciálne sa budeme venovať aj porovnaniu stavovej zložitosti týchto modelov pri regulárnych jazykoch nad unárnymi abecedami. Ukážeme aj, že existuje postupnosť jazykov, kde opitý deterministický automat potrebuje na ich akceptovanie exponenciálne viac stavov ako opitý nedeterministický automat.

Kľúčové slová: konečné automaty, dvojsmerné automaty, stavová zložitosť, opité automaty

Abstract

The concept of state complexity is crucial for understanding and categorizing computational problems, as well as for measuring and comparing the efficiency of various computational models. A significant area of study in this field is the comparison of the state complexity of two-way deterministic and nondeterministic finite automata, as well as one-way deterministic and nondeterministic finite automata. For one-way finite automata, it is known that there exists a sequence of languages where a one-way deterministic finite automaton needs exponentially more states to accept them than a one-way nondeterministic finite automaton. However, for two-way automata, this question remains open. Therefore, researchers focus on studying modifications of these two-way automata. In this thesis, we primarily focus on the state complexity of a model called drunken finite automata, which represent a modification of two-way automata that have no control over the head's movement direction. Instead, the head moves according to the drunkard's walk random process, and the direction of the last movement is the input to the automaton's transition function. Whenever the automaton reaches the end marker of the input word, it must be in the correct accepting or rejecting state. In this work, we first show that drunken finite automata accept the set of regular languages and then focus on examining the state complexity of their deterministic and nondeterministic versions in comparison with finite automata. Special attention is given to comparing the state complexity of these models for regular languages over unary alphabets. We also demonstrate that there exists a sequence of languages where a drunken deterministic automaton needs exponentially more states to accept them than a drunken nondeterministic automaton.

Keywords: finite automata, two-way automata, state complexity, drunken automata

Contents

List of Figures

List of Tables

Introduction

In the realm of computational theory, *state complexity* refers to the number of states required by an automaton to solve a given problem. This concept is vital for understanding and categorizing computational problems, particularly in measuring the efficiency of different approaches. Exploring state complexity allows us to understand the trade-offs between different types of resources, like time and space.

State complexity, especially in the comparison of one-way deterministic and nondeterministic finite automata, is a notable area of study in this field. It has been established, as demonstrated in the work of Albert R. Meyer and Michael J. Fischer [5] in 'Economy in Description by Automata, Grammars, and Formal Systems', that there are exponential gaps in state complexity between these two models.

This means that for certain languages, a nondeterministic finite automaton (NFA) might require only n states to recognize a language and a deterministic finite automaton (DFA) might need as many as 2^n states for the same language.

Christos A. Kapoutsis's 'minicomplexity' paper [3] sheds light on the significance of studying state complexity in the context of two-way finite automata. It builds upon and extends the classical framework of Sakoda and Sipser, emphasizing the importance of state complexity in understanding the computational capabilities of finite automata. It integrates historical theorems and recent theoretical advances, highlighting their relationship to Turing machines' space complexity. Additionally, this work suggests that an exponential gap in the number of states between deterministic and nondeterministic two-way automata could, under certain conditions, contribute to resolving the L vs NL problem, also known as the DLOG vs NLOG problem.

In light of this unresolved question, researchers have shifted focus to investigating various modified versions of two-way automata. Richard Kralovič's dissertation 'Complexity Classes of Finite Automata' [4] examines models of two-way finite automata with restricted head movements, including rotating and sweeping automata. This includes identifying an exponential gap in state complexity between the nondeterministic and deterministic models of rotating and sweeping automata. Sweeping automata are similar to two-way finite automata, but they only change the read head's direction at the start or end of the input string. Rotating automata, on the other hand, are more like one-way finite automata as they read input from left to right. However, they differ

in that they can traverse the input a fixed number of times.

Juraj Hromkovič and Georg Schnitger's work, particularly on oblivious two-way automata as detailed in 'Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipser's Separation' [2], is a significant contribution in this field. Oblivious two-way automata are characterized by their reading head moving in a fixed pattern, independent of the input. This means that for every input length, the order in which the tape cells are visited by the automaton's head is the same for all inputs of that length. Their research not only explores these automata with limited trajectories but also uncovers an exponential gap in state complexity between deterministic and nondeterministic versions of these models.

Building on this foundation, this thesis investigates an extreme scenario, a modification of two-way finite automata called *drunken finite automata* that have no control over the direction of the head movement. In this study, both deterministic (dDFA) and nondeterministic (dNFA) versions of drunken finite automata are examined. The head's movement in these automata follows the 'drunkard's walk' random process, where the direction of the last movement dictates the automaton's transition function. Whenever the automaton reaches the input word's end marker, it must transition to an appropriate accepting or rejecting state. This concept draws inspiration from random walks, particularly the idea of a 'drunkard's walk', as detailed in Frank Spitzer's 'Principles of Random Walk' [7]. Random walks are sequences where each step is randomly determined, exemplified by the 'drunkard's walk', which depicts a random, one-dimensional path akin to a drunken person's stagger. The first objective of this research is to identify the specific languages recognizable by drunken finite automata, both deterministic and nondeterministic. The main focus of this thesis is to conduct a comparative analysis of the state complexity in drunken finite automata relative to various other types of automata. The study will compare the deterministic and nondeterministic versions of these automata and will then broaden its scope to include comparisons with deterministic finite automata and nondeterministic finite automata.

Chapter 1

Definitions

Definition 1 *A Drunken Deterministic Finite Automaton (dDFA) is formally defined as a quintuple* $A = (K, \Sigma, \delta, q_0, F)$*, where:*

- K *is a finite set of states,*
- Σ *is the input alphabet, with the symbols* $\leftarrow, \rightarrow, £, e$ *not in* Σ*,*
- $\delta : K \times (\Sigma \cup \{£\}) \times \{\leftarrow, \rightarrow\} \rightarrow K$ *is the transition function,*
- $q_0 \in K$ *is the initial state,*
- $F \subseteq K$ *is the set of accepting states.*

The input to the automaton's transition function includes the current state of the automaton, the letter currently being read by the reading head, and the direction in which the head last moved. Based on this information, the automaton changes state and then moves head.

Definition 2 *A configuration of a dDFA* $A = (K, \Sigma, \delta, q_0, F)$ *can be a triple* $(q, \varepsilon, £we)$ *if the head is reading the left end marker* $£$*, a triple* $(q, £w, e)$ *if the head is reading the right end marker* e*, or a triple* $(q, £u, ave)$*, where the head reads the character* $a \in \Sigma$*, with* $v, u \in \Sigma^*$ *and* $vau = w$*, for all cases* $q \in K$ *and* w *is the input word.*

Definition 3 *A computation step of the dDFA* $A = (K, \Sigma, \delta, q_0, F)$ *is a binary relation* $\vdash_A$ *on the set of configurations of the automaton* A *such that:*

1. $(q, £v, aue) \vdash_A (p, £va, ue)$ *iff* $p = \delta(q, a, x)$*,*
2. $(q, £vb, aue) \vdash_A (p, £v, baue)$ *iff* $p = \delta(q, a, x)$*,*
3. $(q, £, aue) \vdash_A (p, \varepsilon, £aue)$ *iff* $p = \delta(q, a, x)$*,*
4. $(p, \varepsilon, £aue) \vdash_A (q, £, aue)$ *iff* $p = \delta(q, £, \leftarrow)$*,*

5. $(q, £va, e) \vdash_A (p, £v, ae)$ *iff* $p = \delta(q, e, \rightarrow)$,

where $x \in \{\leftarrow, \rightarrow\}$ *represents the direction in which the head last moved before reading* $a \in \Sigma$, *or* e, *or* $£$, $b \in \Sigma$ *and* $uv \in \Sigma^*$. *No other pairs of configurations belong to this relation.*

Definition 4 *A Drunken Nondeterministic Finite Automaton (dNFA) is formally defined as a quintuple* $A = (K, \Sigma, \delta, q_0, F)$, *where:*

- K *is a finite set of states,*
- Σ *is the input alphabet, with the symbols* $\leftarrow, \rightarrow, £, e$ *not in* Σ,
- $\delta : K \times (\Sigma \cup \{£\}) \times \{\leftarrow, \rightarrow\} \rightarrow 2^K$ *is the transition function*
- $q_0 \in K$ *is the initial state,*
- $F \subseteq K$ *is the set of accepting states.*

The input to the automaton's transition function includes the current state of the automaton, the letter currently being read by the reading head, and the direction in which the head last moved. Based on this information, the automaton can choose which state to change into and then moves its head.

Definition 5 *A configuration of a dNFA* $A = (K, \Sigma, \delta, q_0, F)$ *can be a triple* $(q, \varepsilon, £we)$ *if the head is reading the left end marker* $£$, *a triple* $(q, £w, e)$ *if the head is reading the right end marker* e, *or a triple* $(q, £u, ave)$, *where the head reads the character* $a \in \Sigma$, *with* $v, u \in \Sigma^*$ *and* $vau = w$, *for all cases* $q \in K$ *and* w *is the input word.*

Definition 6 *A computation step of the dNFA* $A = (K, \Sigma, \delta, q_0, F)$ *is a binary relation* $\vdash_A$ *on the set of configurations of the automaton* A *such that:*

1. $(q, £v, aue) \vdash_A (p, £va, ue)$ *iff* $p \in \delta(q, a, x)$,
2. $(q, £vb, aue) \vdash_A (p, £v, baue)$ *iff* $p \in \delta(q, a, x)$,
3. $(q, £, aue) \vdash_A (p, \varepsilon, £aue)$ *iff* $p \in \delta(q, a, x)$,
4. $(p, \varepsilon, £aue) \vdash_A (q, £, aue)$ *iff* $p \in \delta(q, £, \leftarrow)$,
5. $(q, £va, e) \vdash_A (p, £v, ae)$ *iff* $p \in \delta(q, e, \rightarrow)$,

where $x \in \{\leftarrow, \rightarrow\}$ *represents the direction in which the head last moved before reading* $a \in \Sigma$, *or* e, *or* $£$, $b \in \Sigma$ *and* $uv \in \Sigma^*$. *No other pairs of configurations belong to this relation.*

Definition 7 *A trajectory s of length n on the word w is a sequence $s \in \{\leftarrow, \rightarrow\}^n$ that satisfies the following conditions:*

- *The net movement equals the length of the word: $\#_{\rightarrow}(s) - \#_{\leftarrow}(s) = |w|$*
- *For all prefixes s_1 of s, it holds that $0 \leq \#_{\rightarrow}(s_1) - \#_{\leftarrow}(s_1) < |w|$*

Thus, the trajectory s on the word w represents the movement that the head of the Drunken Automaton would make across the word, ending at the right end marker without having read it before.

Definition 8 *A Trajectory-Based Computation is denoted by $(q_0, \text{£}, we) \xrightarrow{s}{}^{*} (q, \text{£}w, e)$. This notation indicates that the automaton A starts in state q_0 with the input w and follows the trajectory s, and reaches state q with the right end marker e after processing the word w.*

Definition 9 *A language $L_{acc}(A)$ accepted by drunken finite automaton $A = (K, \Sigma, \delta, q_0, F)$ consists of all words for which there exists a trajectory such that there is a computation ending in an accepting state on this trajectory:*

$$L_{acc}(A) = \big\{ w \in \Sigma^* \mid \exists s \in \{\leftarrow, \rightarrow\}^* \text{ such that } s \text{ is a trajectory for } w \text{ and } \exists q \in F, (q_0, \mathcal{L}, we) \xrightarrow{s}{}^{*} (q, \mathcal{L}w, e) \big\}$$

Definition 10 *A language $L_{rej}(A)$ rejected by drunken finite automaton $A = (K, \Sigma, \delta, q_0, F)$ consists of all words for which there exists a trajectory such that no computation on this trajectory ends in an accepting state:*

$$L_{rej}(A) = \big\{ w \in \Sigma^* \mid \exists s \in \{\leftarrow, \rightarrow\}^* \text{ such that } s \text{ is a trajectory for } w \text{ and } \forall q \in F, (q_0, \mathcal{L}, we) \xrightarrow{s}{}^{*} (q, \mathcal{L}w, e) \text{ does not exist} \big\}$$

We consider the automaton to be *correct* if every word over its alphabet belongs to exactly one of these languages.

Definition 11 *Let L be a language over an alphabet Σ, and let M be an automaton model from the set {NFA, dNFA, DFA, dDFA}. The state complexity of L using model M is defined as the minimum number of states required in an automaton of model M to recognize L. Formally:*

$$sc(M, L) = \min \{ |Q| \mid A \text{ is an automaton of model } M \text{ accepting } L \text{ with a set of states } Q \}$$

Specifically, if the model is known, we will use $sc(L)$ to denote the state complexity.

Definition 12 *We say that there is a gap in state complexity between two models M_1 and M_2 if there exists a sequence of languages L_i such that:*

$$\lim_{i\to\infty} \frac{sc(M_1, L_i)}{sc(M_2, L_i)} = 0$$

We will say that there is a *gap over Li* if the equation above holds for a sequence of languages Li.

Example:

A drunken deterministic automaton $A = (K, \Sigma, \delta, q_0, F)$ accepting the language $L = \{w \in \{a, b\}^* \mid \#_a(w) \mod 3 \equiv \#_b(w) \mod 3\}$, could look like this:

$$\begin{aligned}
K &= \{[x, y, z] \mid x, y \in \{0, 1, 2\}, z \in \{a, b, £\}\} \\
\Sigma &= \{a, b\}, \\
q_0 &= [0, 0, £], \\
F &= \{[x, y, z] \mid x, y \in \{0, 1, 2\}, z \in \{a, b, £\}, x = y\}
\end{aligned}$$

The transition function δ is defined as:

$$\begin{aligned}
\delta([x, y, z], a, \rightarrow) &= [(x + 1) \mod 3, y, a], & z \in \{a, b, £\} \\
\delta([x, y, z], b, \rightarrow) &= [x, (y + 1) \mod 3, b], & z \in \{a, b, £\} \\
\delta([x, y, a], z, \leftarrow) &= [(x - 1) \mod 3, y, z], & z \in \{a, b\} \\
\delta([x, y, b], z, \leftarrow) &= [x, (y - 1) \mod 3, z], & z \in \{a, b\}
\end{aligned}$$

for all $x, y \in \{0, 1, 2\}$.

The number at the first position of the state represents the count of character 'a' modulo 3 in the already processed part of the word. The number at the second position of the state represents the count of character 'b' modulo 3 in the already processed part of the word. The character at the third position of the state indicates the symbol that was last read before moving the head. Therefore, if the automaton reads a symbol 'x', and the last step was to the right, it increases the part of the state counting the occurrences of 'x'. If the last step was to the left, it subtracts from the count of the last read symbol before making the move, which is remembered in the state.

The computation on the word "abba", where the head moves according to the trajectory "$\rightarrow\rightarrow\leftarrow\leftarrow\rightarrow\rightarrow\rightarrow\rightarrow$" would look like this:

$$([0, 0, £], £, abbae) \vdash_A ([1, 0, a], £a, bbae) \vdash_A ([1, 1, b], £ab, bae) \vdash_A$$

$$([1,2,b], £a, bbae) \vdash_A ([1,1,b], £, abbae) \vdash_A ([1,0,a], £a, bbae) \vdash_A$$

$$([1,1,b], £ab, bae) \vdash_A ([1,2,b], £abb, ae) \vdash_A ([2,2,a], £abba, e)$$

And thus it accepts the word "abba".

Chapter 2

Computational Power of Drunken Finite Automata

2.1 Computational Power of dDFA

Theorem 1 *If there exists a DFA that accepts a language L_1, then there exists a corresponding dDFA that accepts L_1.*

Proof 1 *For a language L_1 and a DFA $A = (K_1, \Sigma_1, \delta_1, q_{01}, F_1)$ with $L(A) = L_1$, we construct a dDFA $O = (K_2, \Sigma_2, \delta_2, q_{02}, F_2)$ as follows:*

$$K_2 = 2^{K_1} \times (\Sigma_1 \cup \{\pounds\})$$
$$\Sigma_2 = \Sigma_1$$
$$q_{02} = (\{q_{01}\}, \pounds)$$
$$F_2 = \{(\{q\}, x) \mid \forall q \in F_1, \forall x \in \Sigma_1 \cup \{\pounds\}\}$$
$$\delta_2((q, x), \leftarrow, y) = (\{p \mid \exists r \in q : \delta_1(p, x) = r\}, y)$$
$$\forall q \in 2^{K_1}, \forall x \in \Sigma_1, \forall y \in \Sigma_1 \cup \{\pounds\}$$
$$\delta_2((q, x), \rightarrow, y) = (\{p \mid \exists r \in q : \delta_1(r, y) = p\}, y)$$
$$\forall q \in 2^{K_1}, \forall y \in \Sigma_1, \forall x \in \Sigma_1 \cup \{\pounds\}$$

Informally speaking, the automaton O maintains in its state a set of states from automaton A, which represents the possible states that A could have been in when the reading head was looking at the symbol reached after the last movement. Additionally, it records the last read symbol in its state to be able to generate this set when moving left.

Proof that $L(A) = L(O)$**.**

Suppose for the sake of contradiction that $L(A) \neq L(O)$, i.e., there exists a word w such

that automaton A accepts w and O rejects it, or vice versa. This implies that there is a computation on word w in automaton O that leads to a different result than automaton A. Let x be the shortest such computation. We examine the head movements of O during this computation. It could not have been just steps to the right, since in that case, O merely simulates the computation of A. Therefore, the head must have made at some point a left step followed by a right step (since the computation of the automaton only finishes when the head reaches the right end marker). It is evident that before moving left, the reading head could not have been reading the left end marker, as it would not have been able to take another left step from there. Let x be the character read by the head before these steps and let $w = uyxv$, $u, v \in \Sigma^*$, $y \in \Sigma$ and let (m_1, x) be the state of the automaton before the head first moves left and then right during this computation (i.e., the state it transitions to after reading x). Let (m_2, y) be the state after the head moves left (and reads y), and let (m_3, z) be the state after the head moves right again and reads x. We will show that $m_1 = m_3$. Clearly from the transition function δ, $z = x$.

We want to show that $m_1 \subseteq m_3$ and $m_3 \subseteq m_1$. For the first part, let $p_1 \in m_1$. Then there exists some $p_2 \in m_2$ such that $\delta_1(p_2, x) = p_1$. It follows that there exists some $p_3 \in m_3$ such that $\delta_1(p_2, x) = p_3$, and therefore $p_3 = p_1$, which implies $p_1 \in m_3$.

For the second part, let $p_3 \in m_3$. Then there exists some $p_2 \in m_2$ such that $\delta_1(p_2, x) = p_3$. It follows that there exists some $p_1 \in m_1$ such that $\delta_1(p_2, x) = p_1$, and hence $p_1 = p_3$, which implies $p_3 \in m_1$.

Since $(m_1, x) = (m_3, z)$, a computation by the automaton on a shortened trajectory without these two steps would lead to the same result, thus we encounter a contradiction with the assumption that the given computation is the shortest. Therefore $L(A) = L(O)$.

Thus drunken deterministic finite automata can accept all languages that are accepted by deterministic finite automata (i.e. all regular languages).

Theorem 2 *Drunken deterministic finite automata accept precisely regular languages.*

Proof 2 *We have already proved that dDFA can accept any regular language. Now we will construct an equivalent DFA from a given dDFA to prove they cannot accept any non-regular languages. If we have any dDFA O_2, we can build a DFA A_2 such that $L(A_2) = L(O_2)$ by making A_2 function in the same way as O_2 on a trajectory where the head only moves to the right. Since O_2 is correct, a word w belongs to the language if and only if O_2 accepts it on every possible trajectory. Similarly, if a word w does not belong to $L(O_2)$, O_2 does not accept it on any trajectory. Therefore, the DFA A_2 accepts a word w if and only if O_2 also accepts it.*

Therefore, drunken deterministic automata accept exactly regular languages.

2.2 Computational Power of dNFA

Theorem 3 *If there exists an NFA that accepts a language L_1, then there exists a corresponding dNFA that accepts L_1.*

Proof 3 *According to the subset construction (also known as the powerset construction), any NFA can be converted into an equivalent DFA that accepts the same language. As shown in Theorem 1, for any DFA, we can construct an equivalent dDFA. Additionally, any dDFA can be viewed as a dNFA. Therefore, for every NFA, there exists an equivalent dNFA that accepts the same language. Thus drunken nondeterministic finite automata can accept all languages that are accepted by nondeterministic finite automata (i.e. all regular languages).*

Theorem 4 *Drunken nondeterministic finite automata accept precisely regular languages.*

Proof 4 *We have already proved that dDFA can accept any regular language. In the same way that we demonstrated the construction of a DFA equivalent to a dDFA, we can construct an equivalent NFA for any dNFA. Therefore, drunken nondeterministic automata cannot accept languages that are not regular, and thus also accept precisely the regular languages.*

Chapter 3

State Complexity Comparison Between Models

In this chapter, we will compare the state complexity of deterministic and nondeterministic drunken automata, as well as with their non-drunken counterparts, the one-way deterministic and nondeterministic finite automata. Due to the unique nature of these models over a unary alphabet, we will first focus on their state complexity separately.

3.1 Regular Languages over Unary Alphabet

Lemma 1 *Graph of the transition function of every deterministic finite automaton (DFA) over a unary alphabet is "Nine-Shaped"*

Proof 5 *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a (DFA), where $\Sigma = \{a\}$ is a unary alphabet.*

Because the alphabet has only one symbol a, the DFA's behavior is determined by repeated applications of the transition function $\delta(q, a)$.

Since the set of states Q is finite, say $|Q| = n$, the DFA can only enter a limited number of states. By the pigeonhole principle, as we keep applying the transition function δ, we must eventually revisit a state. That is, there must be some integers k and j such that $0 \leq k < j$ and $\delta^j(q_0, a) = \delta^k(q_0, a)$. Here, $\delta^i(q, a)$ means applying the transition function δ repeatedly i times starting from state q.

This means that after some number of transitions, the DFA will enter a cycle, revisiting the same sequence of states repeatedly. The structure of this transition can be described as:

1. A "tail" segment: a finite sequence of states starting from the initial state q_0 and leading to the start of the cycle.

2. A "loop" segment: a sequence of states that forms a cycle.

Thus, the transition function graph of the DFA is "nine-shaped", with a finite tail leading into a repeating cycle.

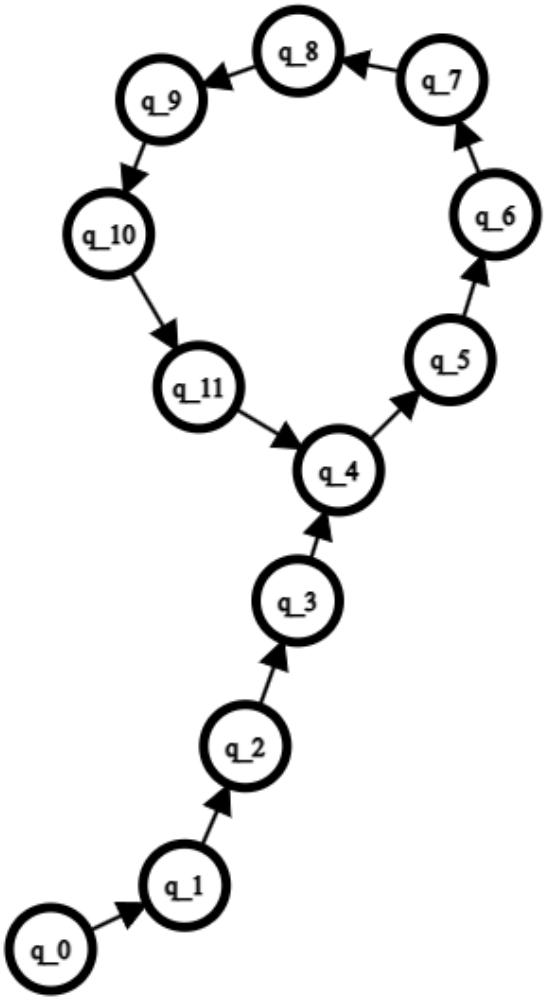

Figure 3.1: Fig.1: A DFA over Unary Alphabet

Theorem 5 *A drunken deterministic finite automaton (dDFA) can accept any regular language over a unary alphabet using one more state than a deterministic finite automaton (DFA) that accepts the same language.*

Proof 6 *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be a DFA, where* $\Sigma = \{a\}$*, that accepts a regular language over a unary alphabet. The transition function of this DFA thus forms a "nine" shape, as we proved in the last lemma. We define a dDFA* $B = (Q, \Sigma, \delta', q_0, F)$*, where:*

- $Q' = Q \cup \{q_{pre}\}$
- $\Sigma = \{a\}$
- δ' *is the new transition function defined below*

- $q'_0 = q_0$
- $F' = F$

The transition function δ' of B is defined as follows: if the last head movement was to the right, then $\delta'(q, a, Right) = \delta(q, a)$. For the new state we define: $\delta'(q_{pre}, a, Right) = q_0$ and $\delta'(q_0, \pounds, Left) = q_{pre}$. If the last head movement was to the left, then $\delta'(q, a, Left) = q'$, where q' is the predecessor of state q in the graph of transition function of DFA A. Clearly, each vertex in the graph has exactly one predecessor (for q_0 we added predecessor q_{pre} and q_{pre} is predecessor for q_{pre}) , except for the junction vertex (intersection of the tail and cycle), where we choose the predecessor from the cycle. Additionally, if the reading head reads left end marker: $\delta'(q, \pounds, Left) = q_{pre}$.

The transition function of DFA A can be visualized as a directed graph where each state is a vertex in the graph and for each $q \in Q$, the edge $(q, \delta(q, a))$ is a directed edge in the graph. This graph has the "nine" structure, with a tail leading into a cycle. In dDFA B, we add back edges: if the last head movement was to the left, dDFA moves to the predecessor in the graph. By adding these back edges, we ensure that B has the correct structure for both right and left movements. Using the extra state, we ensure that if we step on the left end marker and then return from it to the first character, we will be in the same state as at the beginning of the computation.

dDFA B thus mimics the behavior of automaton A for right movements and for left movements dDFA B correctly moves back to the predecessor state, maintaining the correct state sequence. Since dDFA B uses the same set of states Q with one extra state as DFA A and transitions accordingly, both automata accept the same language.

Let us assume for the sake of contradiction that there exists a trajectory where dDFA B reaches a different result than DFA A. Let x be shortest trajectory where dDFA B reaches a different result than DFA A. Since dDFA B merely mimics DFA A for right movements, there must be a left movement in x. Consider the shortest trajectory x that leads to a different result, containing at least one left movement. Obviously, somewhere in this sequence, the head must move to the left and then immediately to the right. So $x = x_1 \cdot Left \cdot Right \cdot x_2$.

But after moving left and then right, dDFA B returns to the same state q it would have been in without these moves: If the automaton was in a state belonging to a cycle before the left move, it will be in the same state after moving left and then right according to the transition function. Similarly, if it was in a state on the linear path, it would just move to the clearly defined predecessor state and return to the state it was in. This does not apply to q_{pre}, but evidently the automaton can only reach q_{pre} if the reading head reaches the left end marker, thus it cannot be in this state before a right move. So:

$$\delta'(\delta'(q, a, Left), a, Right) = q$$

Thus, x can be reduced to $x' = x_1 \cdot x_2$, contradicting the assumption that x was the shortest sequence leading to a different result.

Theorem 6 *A drunken nondeterministic finite automaton (dNFA) can accept any regular language over a unary alphabet using $O(n^2)$ states, where n is the number of states of an nondeterministic finite automaton (NFA) that accepts the same language.*

Proof 7 *We begin by referencing the work of Marek Chrobak in "Finite Automata and Unary Languages" [1], which demonstrates that any unary NFA A can be transformed into a specific normal form. This normal form is characterized by a transition function that consists of a single linear path leading to multiple branching points, each connecting to a distinct cycle. This transformation results in an NFA A' with $O(n^2)$ states, where n is the number of states in the original NFA.*

To illustrate: 1. Linear Path: States $q_0, q_1, \ldots, q_m$ form a sequence where each state transitions to the next under the input symbol a.

2. Branching Point: At the end of the linear path, the state q_m has transitions to multiple initial states $p_{i,0}$ of cycles.

3. Cycles: Each cycle consists of states $p_{i,0}, p_{i,1}, \ldots, p_{i,y_i-1}$, with transitions forming a loop under the input symbol a.

For more details on this normal form and its properties, refer to Chrobak's work.

Given an NFA A accepting L with n states, we can transform it into an NFA A' with $O(n^2) = m$ states in this normal form. Next, we will construct a dNFA B that accepts the same language L using m states, building upon this normal form.

The dNFA B will have the same set of states as A' and one extra state q_{pre} to handle stepping on the left end marker. The transition function of B will operate as follows: - If the last head movement was to the right, B mimics the transition function of A'. - If the last head movement was to the left: - For states lying on a cycle, the transition function moves to the predecessor state in that cycle. - For states lying on the linear path or branches, the transition function moves to the predecessor state along the path or branch. The predecessor of state q_0 will be q_{pre}. For completeness, we also add transitions from every state to q_{pre} when reading the left end marker and a transition from q_{pre} to q_{pre} if the last move was left and reading 'a' or the left end marker.

Formally, let δ_B be the transition function of B and $\delta_{A'}$ be the transition function of A':

$$\delta_B(q, a, \mathit{Right}) = \delta_{A'}(q, a)$$

$$\delta_B(p_{i,j}, a, \mathit{Left}) = \{p_{i,(j-1) \mod y_i}\}$$

$$\delta_B(q_k, a, \mathit{Left}) = \{q_{k-1}\} \quad \text{if } q_k \text{ is on the linear path and } k > 0$$

$$\delta_B(q_0, a, \mathit{Left}) = \{q_{pre}\}$$

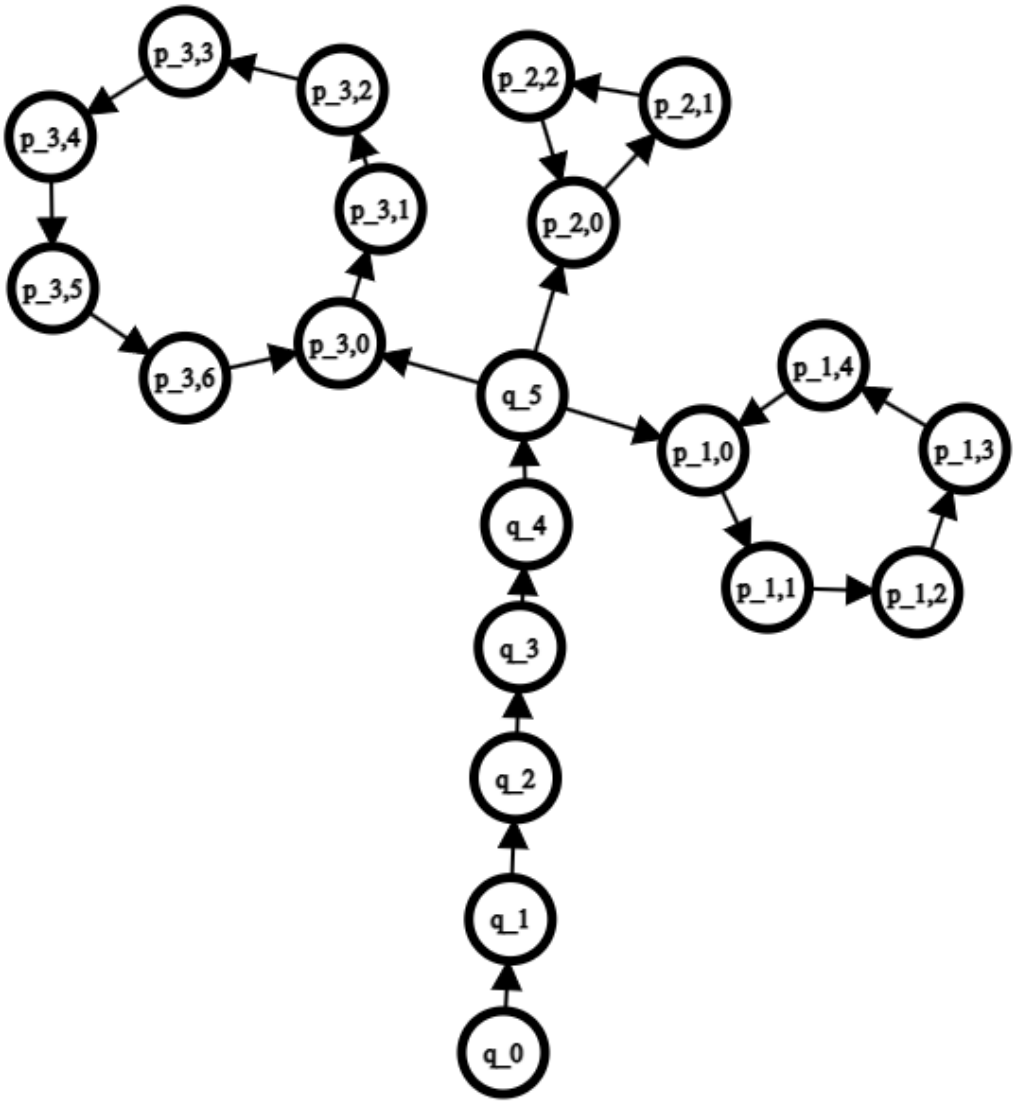

Figure 3.2: Fig.2: A NFA over Unary Alphabet in Normal Form

$$\delta_B(q, £, Left) = \{q_{pre}\} \quad \forall q \in Q$$

$$\delta_B(q_{pre}, a, Left) = \{q_{pre}\}$$

Thus, B correctly simulates A' for both right and left head movements, ensuring it accepts the same language L.

The dNFA B will traverse the word, moving through the states along the linear path until it eventually makes a right move from state q_m, the branching point vertex. At this point, it will non-deterministically guess which initial vertex of the cycles to transition into. For the remainder of the computation, the dNFA will move within that cycle.

It is evident that this construction works for similar reasons as the construction in the previous theorem. The difference in proving that this construction works as well lies in the fact that the automaton can be in any of the larger number of cycles during the

computation and if the automaton has already entered a cycle, it will also remain in it until the end of the calculation. However, within each of these cycles, the computation would work the same as in the cycle of the previous theorem, and the same applies to the linear path. This is also due to the fact that, as we can observe, apart from the nondeterministic guess in state q_m, automata B and A' operate deterministically. Hence B accepts L using $O(n^2)$ states.

3.2 Regular Languages over N-ary Alphabet

We will now focus on comparing state complexity over n-ary alphabets, where n is any natural number ≥ 1.

Lemma 2 *Let L be a regular language accepted by a drunken deterministic finite automaton (dDFA) using n states. Then, there exists a deterministic dinite automaton (DFA) that accepts L using n states.*

Proof 8 *Let L be a regular language and let $M = (Q, \Sigma, \delta, q_0, F)$ be a dDFA that accepts L using n states. We construct a DFA $M_2 = (Q, \Sigma, \delta', q_0, F)$ that accepts L and also uses n states.*

The DFA M_2 operates as follows: for an input word w, M_2 simulates the computation of the dDFA M on the word w and a trajectory where the head moves only to the right from the beginning to the end of the computation. By the definition of a dDFA, this trajectory ensures that M_2 accepts w if and only if M accepts w. Hence, M_2 accepts the language L.

Formally, the transition function δ' of M_2 is defined as:

$$\delta'(q, a) = \delta(q, a, Right)$$

for all $q \in Q$ and $a \in \Sigma$.

This transition function ensures that M_2 behaves exactly as M does when the last direction of the head movement is to the right. Since M_2 uses the same set of states Q as M, M_2 operates with n states.

Therefore, we have constructed a DFA M_2 that accepts the regular language L using n states, proving the lemma.

Corollary 1 *For all regular languages L, we have:*

$$sc(dDFA, L) \geq sc(DFA, L)$$

Lemma 3 *Let L be a regular language accepted by a drunken nondeterministic finite automaton (dNFA) using n states. Then, there exists a nondeterministic finite automaton (NFA) that accepts L using n states.*

Proof 9 *Similarly to previous lemma, we can construct an NFA M_2 that simulates the dNFA M by considering only the trajectory where the head moves in consistently to the right. The NFA M_2 will use the same set of states Q as M and accept the same language L.*

Corollary 2 *For all regular languages L, we have:*

$$sc(dNFA, L) \geq sc(NFA, L)$$

Theorem 7 *Let L be any regular language over the alphabet Σ. If there exists a deterministic finite automaton (DFA) that accepts L with n states, then there exists a drunken nondeterministic finite automaton dNFA B that accepts L with $n \cdot (|\Sigma|+1)+1$ states.*

Proof 10 *Let L be a regular language over the alphabet Σ, and let $A = (K_1, \Sigma, \delta_1, q_0, F_1)$ be a DFA that accepts L with n states. We construct a dNFA $B = (K_2, \Sigma, \delta_2, q_0', F_2)$ as follows:*

$$K_2 = (K_1 \times (\Sigma \cup \{\pounds\})) \cup \{q_{rej}\}$$

$$q_0' = (q_0, \pounds)$$

$$F_2 = F_1 \times (\Sigma \cup \{\pounds\})$$

The transition function δ_2 is defined as follows:

1. For any $q \in K_1$, $y \in \Sigma \cup \{\pounds\}$, and $x \in \Sigma$:

$$\delta_2((q, x), \leftarrow, y) = \{(p, y) \mid \delta_1(p, x) = q\}, \text{ or } \{q_{rej}\} \text{ if this set is empty}$$

2. For any $q \in K_1$, $x \in \Sigma \cup \{\pounds\}$, and $y \in \Sigma$:

$$\delta_2((q, x), \rightarrow, y) = \{(\delta_1(q, y), y)\}$$

Informally, while moving to the right, automaton B simulates the computation of automaton A, while also remembering the last read character. When moving to the left, it non-deterministically guesses the state to return to from the set of states from which automaton A could transition to the current state after reading the last read character, and it remembers the character it is currently reading as well. In this way, it might happen that the automaton ends up in a state p, where if it moves back, the set of states from which it can reach state p on the last read character is empty. Thus, the automaton has incorrectly guessed a state while moving back. For this case, we have added a state q_{rej}, to which it transitions and remains for the rest of the computation.

An accepting computation must exist on every trajectory, because on a trajectory where the automaton moves only to the right, it simply simulates A . On trajectories

where the reading head also moves to the left, by definition of B, the automaton can, at each step to the left, return to the state that A was in when it was processing the same part of the word. Thus, at each step, B can guess the state that A would be in if its head were in the same position.

To prove that B cannot accept a word that does not belong to L, assume for contradiction that there exists a trajectory x where B ends in an accepting state for a word not in L.

Let x be the shortest such trajectory. It certainly includes left moves, and since it must end at the right end of the word, it also includes a left move followed immediately by a right move. The automaton B will evidently not get into the situation described above where it would have to transition to state q_{rej}, since this state is not accepting and it cannot move out of it.

Let y be the character read by the head before these steps, and let $w = uyxv$, with $u, v \in \Sigma^*, y \in \Sigma$. Let (p_1, x) be the state before the head moves left and then right during this computation. Let (p_2, y) be the state after moving left, and (p_3, z) be the state after moving right and transitioning after reading x.

From the transition function δ_2, $z = x$. We will show that $p_1 = p_3$. After moving left and reading y, B transitions to some state (p_2, y) such that $\delta_1(p_2, x) = p_1$. Since A is deterministic, moving right and processing x must transition B back to (p_1, y).

This implies that if an accepting computation exists for the longer trajectory, it must also exist for the shorter one, contradicting the assumption that x was the shortest. Thus, B cannot accept a word not in L and therefore accepts exactly L, using $n \cdot (|\Sigma| + 1) + 1$ states.

Corollary 3 *There does not exist a sequence of languages L_i where each language in this sequence is defined over a common alphabet Σ such that there is a gap over L_i between DFA and dNFA.*

Theorem 8 *There exists an exponential gap between drunken nondeterministic finite automata (dNFA) and deterministic finite automata (DFA).*

Proof 11 *Consider the sequence of languages L_n over the alphabet $\Sigma = \{a, b\}$, where each language consists of words such that the n-th character from the end is the letter "a". It has been shown that for L_n, a DFA requires exponentially more states than an NFA. This result can be found in the work by Albert R. Meyer and Michael J. Fischer titled "Economy of Description by Automata, Grammars, and Formal Systems" [6].*

We will now show that a dNFA can accept the languages L_n using $n + 1$ states, where n is the number of states sufficient for an NFA. Consider a dNFA that operates as follows: - The dNFA starts in an initial state q_1 and scans the input word, moving both to the right and to the left. - If the last move was to the right and the current

character is "a", the dNFA non-deterministically guesses that it is at the i-th character from the end and starts counting down the remaining characters using the remaining $n-1$ states.

Formally, the dNFA operates as follows: - Let $Q = \{q_1, q_2, \ldots, q_n, q_{reject}\}$ be the set of states, where q_1 is the initial state, q_{reject} is an extra rejecting state, and q_n is the accepting state. - The transition function δ is defined such that:

- *From q_1, the automaton can move to the next state q_2 on reading character "a" if it non-deterministically guesses that this is the n-th character from the end or can remain in the state q_1 after reading any character and the head last moved in any direction.*
- *Automaton uses states q_1 to q_n as a counter, to count down the remaining characters.*
- *If the automaton moves left, it decrements the counter and returns to the previous state.*
- *If the automaton moves right, it increments the counter and returns to the previous state.*
- *If the automaton moves right while in q_n, it transitions to the rejecting state q_{reject}, ensuring that any additional characters invalidate the guess.*
- *if the automaton ever reaches state q_{reject}, it will remain in this state until the end of the computation.*

Clearly, an automaton operating in this manner accepts precisely the language L.

It is sufficient for an NFA to use n states to accept the language where the n-th character from the end is "a". A dNFA, as described, requires $n+1$ states, which is asymptotically the same as the NFA. On the other hand, a DFA requires exponentially more states than an NFA for the same language, which is asymptotically much larger. Therefore there is a exponential gap in state complexity between drunken nondeterministic finite automata and deterministic finite automata over sequence of languages L_n .

Theorem 9 *There exists an exponential gap between drunken nondeterministic finite automata (dNFA) and drunken deterministic finite automata (dDFA).*

Proof 12 *From Corollary 1, we know that for any regular language L, the state complexity of a dDFA is at least as large as the state complexity of a DFA. Formally:*

$$sc(dDFA, L) \geq sc(DFA, L)$$

Combining these results with results from previous theorem we get:

$$sc(dDFA, L_n) \geq sc(DFA, L_n)$$

$$sc(dNFA, L_n) \leq sc(NFA, L_n) + 1$$

As there is an exponential gap between the state complexity of DFA and NFA for the languages L_n [6], it follows that there is an exponential gap between the state complexity of DFA and dNFA for these languages. Since state complexity of a dDFA is at least as large as the state complexity of a DFA., we conclude that there is an exponential gap in state complexity between drunken nondeterministic finite automata and drunken deterministic finite automata over sequence of languages L_n.

Table 3.1: Number of States Needed to Accept L_n

	NFA	**dNFA**	**DFA**	**dDFA**
Number of States	n	$n+1$	2^n	$\geq 2^n$

Chapter 4

Some Interesting Languages

An example of a language that could initially seem problematic for drunken nondeterministic automata is L_k^2 described below. It might appear that a drunken nondeterministic automaton could "get lost" after traversing the word and fail to remember its position, potentially guessing the k-th 'a' from the end incorrectly, leading to an incorrect acceptance.

However, drunken nondeterministic automata can handle this language just as effectively as their non-drunken counterparts.

As an example, I will demonstrate a solution from my supervisor, Prof. RNDr. Rastislav Královič, PhD.

Definition 13 *Let k be a parameter. Define the language L_k over the alphabet $\Sigma = \{a, b\}$ as follows:*

$$L_k = \{w \in \Sigma^* \mid \textit{the } k\textit{-th character from the end of } w \textit{ is } 'a'\}$$

The language L_k^2 represents the concatenation of the language L_k with itself.

The language L_k^2 consists of words where the k-th letter from the end is 'a' and there exists an 'a' at a distance of at least $2k$ from the end.

A one-way nondeterministic automaton can recognize L_k^2 using three phases:

- In the first phase, the automaton moves through the word in state q_0 until it nondeterministically guesses that an 'a' read is the last 'a' at a distance of at least $2k$ from the end. This guess transitions the automaton into the second phase.

- In the second phase, the automaton, in a single state, checks that it reads only 'b's. Upon encountering a 'b' (or the first 'a' it encounters), it transitions to the third phase.

- In the third phase, the automaton uses $2k$ states to verify that it is at a distance of $2k$ from the end and simultaneously checks that the k-th letter from the end is 'a'. If not, it transitions to a rejecting state.

Simulating L_k^2 with a Drunken Nondeterministic Automaton: A drunken nondeterministic automaton can directly simulate this process:

- In the first phase, it moves through the word in state q_0.
- When in the second phase and it moves left, it only checks that it is still moving over 'b's. If it encounters an 'a' while moving left, it returns to the first phase.
- In the third phase, the automaton has an exact counter, so it does not matter whether it moves right or left. If it moves beyond the left edge of the counter, it transitions back to phase 2.

Similarly, drunken automata can often handle languages as efficiently as their non-drunken counterparts, especially in cases where the NFA performs only a finite number of nondeterministic choices in each computation and where it is clear which state it transitioned from during backward movements. For instance, languages such as those containing words with specific subsequences, words where a specific number of certain symbols must be present, or words where the number of certain symbols must be congruent to n modulo k, do not pose a problem for drunken automata.

Furthermore, I examined languages where an NFA may need to make potentially any number of nondeterministic guesses. This is particularly problematic in languages where reaching a specific state can result from many different states, and incorrectly choosing which one it was while moving left could lead the automaton to an incorrect accepting state.

Languages where I believe there might be a gap in state complexity between drunken models and their non-drunken versions include:

Definition 14 *Let k be a parameter, and let Σ_k be an alphabet where each symbol represents a bipartite graph $G = (A, B, E)$ with vertex sets A and B both having k vertices and edge set E. Let a_j denote the j-th vertex in A and b_j denote the j-th vertex in B.*

$$L_{reach,k} = \{w \in \Sigma_k^* \mid \exists \text{ a sequence of edges } (a_1, b_{j_1}), (a_{j_1}, b_{j_2}), \ldots,$$

$$(a_{j_{n-1}}, b_1) \text{ such that } (a_1, b_{j_1}) \in E_1, (a_{j_{n-1}}, b_1) \in E_n, \text{ and}$$

$$(a_{j_{i-1}}, b_{j_i}) \in E_i \text{ for } 2 \leq i < n\}$$

where: - $w = w_1 w_2 \cdots w_n$ is a sequence of bipartite graphs from Σ_k, - Each w_i represents a bipartite graph $G_i = (A, B, E_i)$, - There exists a sequence of edges $(a_1, b_{j_1}), (a_{j_1}, b_{j_2}), \ldots, (a_{j_{n-1}}, b_1)$

such that: - The edge (a_1, b_{j_1}) is in E_1, - The edge (a_{j_1}, b_{j_2}) is in E_2, - ... - The edge $(a_{j_{n-1}}, b_1)$ is in E_n. - The sequence of edges forms a path such that the first edge starts at a_1 in the first graph and the last edge ends at b_1 in the last graph.

Definition 15 *Let k be a parameter. Define the summability language $L_{sum,k}$ over the alphabet Σ_k, where each symbol represents a subset of the set $\{0, 1, 2, \ldots, k-1\}$ (excluding the empty set).*

$$L_{sum,k} = \{w \in \Sigma_k^* \mid \exists a_1, a_2, \ldots, a_n \text{ with } a_i \in w_i$$
$$\text{and } (a_1 + a_2 + \cdots + a_n) \mod k = 0\}$$

where: - $\Sigma_k = \mathcal{P}(\{0, 1, 2, \ldots, k-1\}) \setminus \emptyset$, - $w = w_1 w_2 \cdots w_n$ is a sequence of subsets from Σ_k, - $a_i \in w_i$ means that a_i is an element chosen from w_i, - The sum $(a_1 + a_2 + \cdots + a_n)$ is taken modulo k.

The reachability and summability languages are designed to illustrate what i think is the main deficiency of dNFA. In both languages, there is a choice (picking an edge or picking a number) that an NFA can handle non-deterministically without any issues. Therefore, the state complexity of an NFA would be k in both cases, as it either remembers which vertex it is currently in or tracks the corresponding prefix sum after reading the first few letters.

However, the dNFA model faces a problem because, even though it can use non-determinism to make these choices while moving right, it cannot remember the choices it made when moving left. Since there can be potentially any number of steps to the left, the dNFA cannot remember the choices it made using just $O(k)$ states (at least not as far as we know).

In similar languages, remembering various data such as characters or sets of states can lead to significant complexity increases. For the reachability language, after a step to the left, we cannot determine which character was last read before the state was changed or from which set of vertices to search for a backward edge without remembering such information, which could be very "expensive." Nonetheless, these are intuitions, and we currently cannot prove whether any of these languages indeed present a gap.

Conclusion

In the beginning of this thesis, we introduced a modification of two-way automata called drunken finite automata. We demonstrated that drunken deterministic finite automata (dDFA) and drunken nondeterministic finite automata (dNFA) accept precisely the regular languages. We then compared their state complexity with one-way finite automata. We showed that if a DFA can accept a language L over a unary alphabet with n states, then a dDFA can accept this language with $n + 1$ states, and if an NFA can accept a language L over a unary alphabet with n states, then a dNFA can accept it with $O(n^2)$ states.

For a general alphabet, we showed that NFAs and DFAs do not require more states than their non-drunken versions to accept a language. We also demonstrated that there is an exponential gap in state complexity between dNFA and DFA, and between dNFA and dDFA. Additionally, we showed that any dDFA with n states can be transformed into an equivalent DFA with $2^n \times (|\Sigma| + 1)$ states, and any dNFA can be transformed into an equivalent NFA with $2^n \times (|\Sigma| + 1) + 1$ states.

Future research could explore other potential gaps and further investigate the limitations of drunken finite automata. It would also be worthwhile to study other modifications of two-way automata, as these might offer different insights and could also help to determine whether there is an exponential gap between nondeterministic and deterministic two-way automata.

Bibliography

[1] Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.

[2] Juraj Hromkovič and Georg Schnitger. Nondeterminism versus determinism for two-way finite automata: Generalizations of sipser's separation. In *Lecture Notes in Computer Science*, volume 2719 of *ICALP 2003*, Berlin, Heidelberg, 2003. Springer.

[3] Christos A. Kapoutsis. Minicomplexity. *Journal of Automata, Languages and Combinatorics*, 17(2-4), 2012.

[4] Richard Kralovič. *Complexity Classes of Finite Automata*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, 2010. Doctor of Sciences.

[5] Albert R. Meyer and Michael J. Fischer. Economy in description by automata, grammars, and formal systems. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*. IEEE Computer Society, 1971.

[6] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971*, pages 188–191. IEEE Computer Society, 1971.

[7] Frank Spitzer. *Principles of Random Walk*. Van Nostrand, Princeton, N.J., 1964.