Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Descriptional complexity of push down automata

Master's Thesis

2020

Bc. Lukáš Kiss

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

DESCRIPTIONAL COMPLEXITY OF PUSH DOWN AUTOMATA

MASTER'S THESIS

Study programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	prof. RNDr. Branislav Rovan PhD.

Bratislava, 2020
Bc. Lukáš Kiss

73779970

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Lukáš Kiss
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Descriptional Complexity of Push-Down Automata

Annotation: The aim of the thesis is to explore descriptions complexity of nondeterministic push-down automata, especially for the measures number of states and the size of the stack alphabet. Based on the current results in the literature find lower bounds for the complexity of particular languages and possibly explore upper bounds for some operations on languages.

Aim: The aim of the thesis is to explore descriptions complexity of nondeterministic push-down automata, especially for the measures number of states and the size of the stack alphabet. Based on the current results in the literature find lower bounds for the complexity of particular languages and possibly explore upper bounds for some operations on languages.

Supervisor:	prof. RNDr. Branislav Rovan, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	01.10.2018
Approved:	03.03.2020

prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

....

Student Supervisor

73779970

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Lukáš Kiss
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Descriptional Complexity of Push-Down Automata
Popisná zložitosť zásobníkových automatov

Anotácia: Cieľom práce je preskúmať popisnú zložitosť nedeterministických zásobníkových automatov, najmä pre miery počet stavov a počet zásobníkových symbolov a ich kombináciu. Vychádzajúc zo súčasného stavu problematiky hľadať dolné odhady zložitosti konkrétnych jazykov, prípadne horné odhady pre niektoré operácie na jazykoch.

Cieľ: Cieľom práce je preskúmať popisnú zložitosť nedeterministických zásobníkových automatov, najmä pre miery počet stavov a počet zásobníkových symbolov a ich kombináciu. Vychádzajúc zo súčasného stavu problematiky hľadať dolné odhady zložitosti konkrétnych jazykov, prípadne horné odhady pre niektoré operácie na jazykoch.

Vedúci:	prof. RNDr. Branislav Rovan, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 01.10.2018

Dátum schválenia: 03.03.2020

prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

....

študent vedúci práce

Acknowledgment: Foremost, I would like to express my sincere gratitude to my Supervisor prof. RNDr. Branislav Rovan, PhD. for the continues support of my master thesis, for his patience, immense knowledge and understanding. His guidance helped me to detect incorrect ideas or mistakes, which I would not be able to find alone.

Abstrakt

Študovali sme popisnú zložitosť zásobníkových automatov na regulárnych a bezkontextových jazykoch. Predstavili sme transformaciu z ľubovolného konečného automaton používajúceho n stavov na zásobníkový automat používajúci p zásobníkových symbolov, ktorá redukuje počet stavov na $\lceil \frac{n}{p} \rceil$. Analyzovali sme popisnú zložitosť zásobníkových automatov na dvoch postupnostiach regulárnuch jazykov $L_1[n]$ a $L_2[n]$. Ľubovolný konečný automat potrebuje aspoň n stavov na akceptovanie $L_1[n]$ a $L_2[n]$. Ukázali sme, že preľubovolné n existuje minimálny zásobníkový automat používajúci dva zásbnikové symboly a jeden stav a akceptuje jazyk $L_2[n]$. Na druhej strane, zásobníkový automat používajúci jeden stav potrebuje aspoň n zásobníkovýh symbolov, aby akceptoval $L_1[n]$. Následne sme definovali čiastočné usporiadanie na stavoch a zásobníkových symboloch a ukázali, že neexistuje funkcia, ktorá kombinuje tieto miery tak, že záchovava zložitosť medzi minimálnymi zásobnikovými automatmi pre daný jazyk. Na základe týchto výsledkov, sme definovali dve podtriedy zásobníkových automatov, *jedno stavové PDA* a *dvojo zásobnikovo symbolové PDA*. V podtrede *jedno stavové PDA* sme ukázali tesný odhad z hora aj z dola počtu zásobníkových symbolov a v druhej triede *PDA s dvoma zasobnikovymi symbolmi* sme ukázali horné odhady. Nakoniec sme sa pozreli na horne odhady zlozitosti pre operacie v týchto podtriedach. Konkretne sme skumali zjednotenie, zreťazenie a iteráciu.

Keywords: automaty, zložitosť, odhady

Abstract

We study descriptional complexity of push down automata recognizing regular languages and context free languages. We presented the transformation for any finite state automaton A using n states and given a number of stack symbols p, which reduces the number of states to $\lceil \frac{n}{p} \rceil$. We analyzed descriptional complexity of PDA on the two sequences of regular languages $L_1[n]$ and $L_2[n]$. An FSA requires at least n states to accept $L_1[n]$ and $L_2[n]$. We have shown that for any n two stack symbols and one state are sufficient for PDA to accept the language $L_2[n]$. In the other hand, PDA using one state requires at least n stack symbols to accept $L_1[n]$. Then, we defined partial ordering on the state and the number of stack complexity measures and show that there does not exist any function, which combines these measures in such a way that it maintains complexity from one minimal automaton to another one. Based on these results, we define two subclasses of push down automata, *one state PDA* and *two stack symbols PDA*. In these subclasses, we have shown tight bounds for *one state PDA* subclass and upper bounds for *two stack symbols PDA* subclass. Finally, we study the costs of operation in these two subclasses. In particular, we considered union, concatenation and Klenee-Star.

Keywords: automata, complexity, bounds

Contents

List of Figures

List of Tables

Introduction

The study of decriptional complexity describes various methods of specifying objects. It dates back to 1950s and one of the first measures used to compare the complexity of regular languages was the number of states measure on finite state automata. The first results compared the number of states complexity of deterministic versus nondeterministic finite state automata (which were shown to define the same family of languages). It was shown that nondeterminism can offer exponential state savings compared to determinism [13]. In particular, the exponential growth is always sufficient and is in some cases necessary [10].

The application of finite state automata is frequent in computer science. Still, there are many unresolved problems. One of these problems were how many states are sufficient and necessary for operations on finite and infinite regular languages. Especially, the costs of operations over unary and arbitrary alphabets represented by NFAs or DFAs [6, 14].

The other complexity measures considered for regular languages were investigated, e. g. boolean circuit complexity of regular languages [12] or the number of transitions. The number of transitions measure gives, in some sense, a more realistic measure for the size of an NFA than the number of states [2]. However, most of the work on complexity of regular languages yields worst-case results. Hermann Gruber and Markus Holzer investigated the average-case state and transition complexity of deterministic and nondeterministic finite automata for finite languages. [5].

Similar measures can not be easily defined on push down automata. Especially the number of states measure and the number of stack symbols measure. Both states and stack symbols are depended on each other. Goldstine, Price and Wotschke showed that there exists a transformation for any PDA using n states and p stack symbols reducing the number of states to any desired number $n_0 \geq 1$ [3]. This is one of the main reasons, why many descriptional complexity studies focus on context free grammars [9, 1] or some modifications of push down automata [11, 7].

In this thesis, we shall study the descriptional complexity measures as state measure and the number of stack symbols on PDA. We define partial ordering on the these measures and show there does not exists any function, which combines these measures in such a way that it maintains complexity from one minimal automaton to another one. Therefore, we consider two equivalent subclasses of PDA, *one state PDA* and *two stack symbols PDA*. In the *one state PDA* subclass, we fix the number of states to one and use the number of stack symbols as the measure. In the second subclass, we fix the number of stack symbols to two and measure the number of states. Note that both these subclasses of PDA can define all context-free languages.

In the Chapter 2, we study descriptional complexity of PDA accepting regular languages. We show the effect of stack symbols on the complexity of any regular language and then prove tight complexity bounds for some regular languages.

In the Chapter 3, we study two particular subclasses of PDA and show that empty stack acceptance mode and final state acceptance mode are equivalent in the *two stack symbols PDA* subclass. Unfortunately, these two acceptance modes are not equivalent in the *one state PDA* subclass. Moreover, we prove tight bounds for *one state PDA* subclass and upper bounds for *two stack symbols PDA* subclass.

In the last Chapter 4, we investigate the costs of operations on context free languages in the two subclasses. In particular, we consider union, concatenation and Klenee-Star. All of the bounds are in the exact number of stack symbols for the first subclass and exact number of states for the second subclass. We do not show tight bounds, but only the upper bound constructions in both subclasses.

Chapter 1

Preliminaries and Definitions

In this thesis, we shall analyze the complexity measures of push down automata. We shall recall the definition of a nondeterministic push down automaton and then we shall show some basics results, which shall be used later in this thesis.

Finally, we shall analyze the problem of defining a "good" complexity measure on the PDA. In particular , we shall show that there does not exist any mapping function f over states and stack symbols, which for every two minimal automata for a particular language assigns the same value. That is why we shall define two subclasses of PDA. In the first, we fix the number of states to one and in the second we fix the number of stack symbols to two. Note that both can accept all context free languages.

1.1 Definitions

Definition 1. A nondeterministic push down automaton (PDA) is a 7-tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

- Q is a finite set of states
- Σ is a finite set of symbols - *input alphabet*
- Γ is a finite set of symbols - *stack alphabet*
- δ is a finite subset of $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ - *transition function*
- $q_0 \in Q$ is the *initial state*
- $Z_0 \in \Gamma$ is the *initial stack symbol*
- $F \subseteq Q$ is the set of *accepting states*

Definition 2. A configuration of a nondeterministic push down automaton is a triple $(q, v, \gamma) \in Q \times \Sigma^* \times \Gamma^*$, where q is a state, v is the unread portion of the input word and γ is the content of the stack.

Definition 3. A computation (transition) step of a PDA A is the relation $\vdash_A$ on configurations defined by

$$(q, au, \gamma Z) \vdash_A (p, u, \gamma\alpha) \iff (p, \alpha) \in \delta(q, a, Z)$$

where $p, q \in Q, a \in \Sigma \cup \{\epsilon\}, Z \in \Gamma$ and $\gamma, \alpha \in \Gamma^*$

The relation $\vdash_A^*$ is the reflexive and transitive closure of the relation $\vdash_A$.

Definition 4. Given a PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, the language accepted by the set of accepting (final) states F is

$$L(A) = \{w \in \Sigma^* | \exists q \in F \text{ and } \exists \gamma \in \Gamma^* \text{ such that } (q_0, w, Z) \vdash^* (q, \epsilon, \gamma)\}$$

Definition 5. Given a PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, the language accepted by the empty stack is

$$N(A) = \{w \in \Sigma^* | \exists q \in Q \text{ such that } (q_0, w, Z) \vdash^* (q, \epsilon, \epsilon)\}$$

We can think of a PDA as finite state automata with memory. This memory is simply a stack that can be manipulated during the moves by the finite automaton . The stack is initialized by initial stack symbol. Usually it is denoted by Z. The computatoin (transition) step of PDA is similar to the transition of FSA, but the PDA can push some stack symbols on the stack. The automaton can empty the stack by pushing ϵ in the transition. We shall often say popping from the stack instead of pushing ϵ on the stack. The PDA has two acceptance modes, empty stack and final state. These two modes are equivalent.

Definition 6. Given an alphabet Σ, the *Shuffle* of two languages $L_1, L_2 \subseteq \Sigma^*$ is

$$Shuf(L_1, L_2) = \{w | \exists n \in N, u_1, \dots, u_n, v_1, \dots, v_n \in \Sigma^*; \text{ such that}$$

$$w = u_1 v_1 u_2 v_2 \dots u_n v_n \wedge u_1 \dots u_n \in L_1 \wedge v_1 \dots v_n \in L_2\}$$

Notation 1. $\mathscr{D}(n, p)$ is the family of pushdown automata using at most n states and at most p stack symbols.

Definition 7. Let L be a context-free language. The *stack symbol complexity* of L, $\Gamma c(L)$, is the smallest number of stack symbols of any A in $\mathscr{D}(1, p)$ that accepts L.

Definition 8. Let L be a context-free language. The *state complexity* of L, $Qc(L)$, is the smallest number of states of any A in $\mathscr{D}(n, 2)$ that accepts L.

1.2 Basic Results

In this section, we shall introduce some known results on the push down automata. Mainly, the reduction of stack symbols or states shown by Goldstine, Price and Wotschke in [3, 4].

Moreover, they have compared the PDA with context free grammars and proved some lower bounds in. We shall focus on lower bounds and upper bounds for PDA. Therefore, we omit constructions of lower or upper bounds based on context-free grammars.

1.2.1 Reducing the Number of Stack Symbols

In 1993, Goldstine, Price and Wotschke showed a deterministic and nondeterministic reduction of stack symbols of any push down automaton. For any PDA M using n states and p stack symbols, assigns to each computational step of M a unique sequence of computational steps of the simulating PDA $\hat{M}$. Thus the construction does not introduce "more nondeterminism" to $\hat{M}$ and reduces the number of stack symbols to any $\hat{p} \geq 2$. This reduction results in an equivalent push down automaton with $O(np/\hat{p})$ states. The nondeterministic construction introduces more nondeterminism to the automaton $\hat{M}$, but reduces the number of states to $O(n\sqrt{p/\hat{p}})$. In the end, they showed that these constructions are essentially the best possible.

Before we show the ideas of these constructions, it is important to note that these constructions work only if the automaton M accepts by empty stack and constructs an equivalent PDA, which accepts by empty stack.

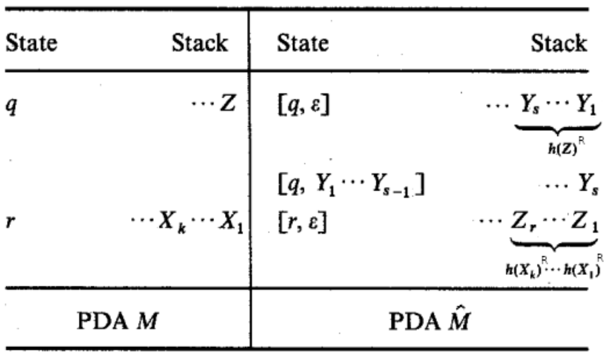
Note 1. We present the idea of the construction of Goldstine, Price and Wotschke with minor change in description. We are used to write the content of the stack in configuration with top of the stack on the right hand side of the word. Since they decided to write the content of the stack in configuration with top of the stack on the left hand side our description of the construction uses reverse.

Deterministic Reduction

In this subsection, we shall present the main idea of the transformation of an arbitrary PDA M to an equivalent PDA $\hat{M}$ having a desired number of stack symbols $\hat{p} \geq 2$, which simulates M without increasing nondeterminism. The automaton $\hat{M}$ has fewer stack symbols then M, so all stack symbols Z from the automaton M are represented by a string $h(Z)^R$. The $h(Z)$ is a variable-length encoding and the mapping h is chosen to be a prefix code. The state of automaton $\hat{M}$ is represented by a pair $[q, \gamma]$, where q is a state of M and γ is proper prefix[1] of $h(Z)$.

The automaton $\hat{M}$ reads the encoded stack symbol Z as $h(Z)^R$ from the stack on ϵ-moves and then reaches state $[q, \gamma]$. On this state, the automaton simulates one computation step of automaton M and moves to state $[r, \epsilon]$. During the simulated computation step, it pushes the $h(X_k)^R \dots h(X_1)^R$ on the stack, where $w = X_k \dots X_1$ is the word pushed by M on the stack. The simple transition can be seen in the Table 1.1.

[1]The string x is a proper prefix of xy if $y \neq \epsilon$.



State	Stack	State	Stack
q	$\cdots Z$	$[q, \varepsilon]$	$\cdots \underbrace{Y_s \cdots Y_1}_{h(Z)^R}$
		$[q, Y_1 \cdots Y_{s-1}]$	$\cdots Y_s$
r	$\cdots X_k \cdots X_1$	$[r, \varepsilon]$	$\cdots \underbrace{Z_r \cdots Z_1}_{h(X_k)^R \cdots h(X_1)^R}$
PDA M		PDA $\hat{M}$	

Figure 1.1: Deterministic simulation of one computation step of M by $\hat{M}$ [4].

Goldstine, Price and Wotschke proves that such mapping h exists for any PDA M and results in $O(np/\hat{p})$ states.

Nondeterministic Reduction

In the previous part, we have discussed the ideas of deterministic stack symbols reduction. Now, we shall introduce ideas of a nondeterministic reduction, which produces a PDA with fewer states. Instead of using one mapping function h, we shall use 3 mapping functions $f, g, \bar{g}$ over the stack alphabet of $\hat{M}$. Each stack symbol is encoded as $g(Z)^R f(Z)^R$.

Goldstine, Price and Wotschke have chosen the mappings $f, g, \bar{g}$ such that they satisfy these conditions:

- $(f(Z), g(Z))$ uniquely determines Z
- $\bar{g}(Z)$ uniquely determines $g(Z)$
- $f(Z) \neq \bar{g}(Y)$, and neither $f(Z)$ nor $\bar{g}(Z)$ is a proper prefix of $f(Y)$ or $\bar{g}(Y)$
- $g(Z)$ is not a proper prefix of $g(Y)$

These are necessary conditions that mappings $f, g, \bar{g}$ has to satisfy. Thanks to these condition, the automaton $\hat{M}$ can guess and then check his guess correctly.[2]

The automaton $\hat{M}$ can be in two possible states

- $[q, \$\alpha]$, where q is a state of M and α is a proper prefix of $f(Z)$ or $\bar{g}(Z)$ for some stack symbol Z of M
- $[q, \beta\$]$, where q is a state of M and β is a non-empty suffix of $g(Z)$ for some stack symbol Z

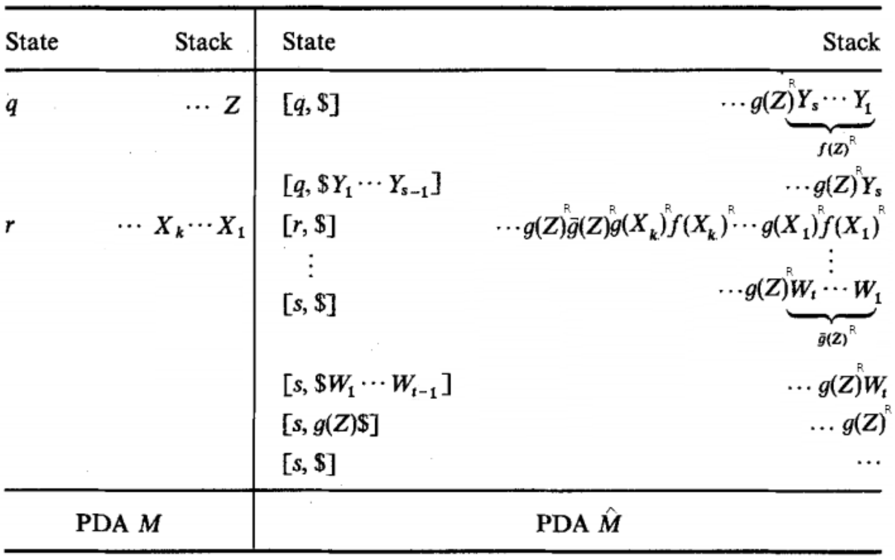
[2]More details can be found in the article [4].

During the simulation, M arrives to the point, where $\hat{M}$ is in the state $[q, \$]$ with $g(Z)^R f(Z)^R$ on the stack.[3] As in the deterministic simulation, it removes all but the last symbol in $f(Z)^R$ from the stack and stores it in the state. Then automaton guess the stack symbol Y from set of all stack symbols, which have the same $f(Z)$ and stores

$$\bar{\mathbf{g}}(\mathbf{Y})^R g(X_1)^R f(X_1)^R \ldots g(X_n)^R f(X_n)^R$$

on the top of the stack. The $\bar{g}(Y)$ represents the guessed stack symbol.

The guess will be tested, when the automaton $\hat{M}$ reaches any state $[q, \$]$ and the top of the stack is $g(Z)^R \bar{g}(Y)^R$. Then again the automaton $\hat{M}$ reads the $\bar{g}(Y)^R$ from the stack and stores it in the state. After this, the automaton $\hat{M}$ knows the string $\bar{g}(Y)$ and now it can test the guesses by moving to the state $[q, g(y)\$]$ (The $\bar{g}(Y)$ uniquely determines $g(Y)$). Finally, $\hat{M}$ is using a sequence of ϵ moves to verify, if $g(Y) = g(Z)$. If they are not equal then the automaton halts. The four conditions guaranties that the automaton $\hat{M}$ can not be mistaken in verifying his guess. The simulation of one computation step can be seen in the Table 1.2.



State	Stack	State	Stack
q	$\cdots Z$	$[q, \$]$	$\cdots g(Z)^R \underbrace{Y_s \cdots Y_1}_{f(Z)^R}$
		$[q, \$Y_1 \cdots Y_{s-1}]$	$\cdots g(Z)^R Y_s$
r	$\cdots X_k \cdots X_1$	$[r, \$]$	$\cdots g(Z)^R \bar{g}(Z)^R g(X_k)^R f(X_k)^R \cdots g(X_1)^R f(X_1)^R$
		$\vdots$	$\vdots$
		$[s, \$]$	$\cdots g(Z)^R \underbrace{W_t \cdots W_1}_{\bar{g}(Z)^R}$
		$[s, \$W_1 \cdots W_{t-1}]$	$\cdots g(Z)^R W_t$
		$[s, g(Z)\$]$	$\ldots g(Z)^R$
		$[s, \$]$	$\cdots$
PDA M		PDA $\hat{M}$	

Figure 1.2: Nondeterministic simulation of M by $\hat{M}$ [4].

1.3 Descriptional Complexity on PDA

In this section, we shall introduce the descriptional complexity of nondeterministic push down automata. We shall see that to define a "good" complexity measure on PDA is not easy compared to finite state automata[4]. The typical measure is number of states. This measure has a big flaw on PDA automata. Any context free language can be accepted by one state push

[3]Before $\hat{M}$ starts simulating the automaton M, it replaces the initial symbol by $g(Z)^R f(Z)^R$ and starts the simulation of M, where Z is the initial symbol.

[4]The typical complexity measure is number of states.

down automaton. The automaton simulates a context free grammar derivation in his stack and compares the input word with the derived word on the stack. On the other hand, using just number of stack symbols complexity measure results in the similar problem. Every context free language can be accepted by a PDA using just two stack symbols.

The promising solution would be to use some measure combining states and stack symbols. We shall show that such a "good" combination does not exist, which satisfies our conditions such as total ordering and two minimal automata for the same language have the same complexity.

Let us use similar approach as Labath and Rovan did for the deterministic push down automata [8]. We shall start with an easy measure, which solves the easier cases:

Definition 9. $\preceq, \prec$ are relations (partial orderings) on $\mathbb{N} \times \mathbb{N}$ defined by:

$$(a, b) \preceq (\hat{a}, \hat{b}) \overset{def}{\iff} a \leq \hat{a} \wedge b \leq \hat{b}$$

$$(a, b) \prec (\hat{a}, \hat{b}) \overset{def}{\iff} a < \hat{a} \wedge b < \hat{b}$$

Definition 10. Let $A, \hat{A}$ be push down automata. We say that A is simpler then $\hat{A}$, iff $(|Q_A|, |\Gamma_A|) \prec (|Q_{\hat{A}}|, |\Gamma_{\hat{A}}|)$.

Definition 11. Let $A, \hat{A}$ be push down automata. We say that A is not more complex then $\hat{A}$, iff $(|Q_A|, |\Gamma_A|) \preceq (|Q_{\hat{A}}|, |\Gamma_{\hat{A}}|)$.

This measure does not allow us to compare all PDA. For example two push down automata can accept the same language, but the first one using two stack symbols and one state and the second using one stack symbol and two states.

We would like to find a measure, which solves this issue, but respects the definitions 10 and 11. For any measure, which satisfies these definitions, we can define minimal automata recognizing some language as follows:

Definition 12. Let $\prec$ be a partial ordering on $\mathbb{N} \times \mathbb{N}$. Let PDA A recognizes a language L using n states and p stack symbols. We say, that A is a minimal automaton recognizing L, iff there does not exist a PDA $\hat{A}$ recognizing L using $\hat{n}$ states and $\hat{p}$ stack symbols such that:

$$(n, p) \prec (\hat{n}, \hat{p})$$

The question arises, if there exists any function , which for all pairs of minimal PDA returns the same value. This function would allow us to easily define complexity measure on context free languages. Also, it would allow us to choose the number of states or stack symbols we want to use. Unfortunately, this kind of function does not exists.

Before we show any proof of this claim, we shall introduce a sequence of languages

$$L_2[n] = \{a_1^{kn} | k \geq 0\}$$

also used later in Chapter *Push-Down Automata on Regular Languages*.

In this chapter, we showed in the Corollary 2.2.4.1 that for any n there exists a minimal PDA using 2 stack symbols and 1 state and accepting $L_2[n]$. On the other hand, any finite state automaton needs at least n states to accept $L_2[n]$. (In each state, it saves the reminder after division by n). Now, we are ready to show that this function f does not exist.

Theorem 1.3.1. *There is no function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ meeting the following conditions:*

1. *For every two PDA A and $\hat{A}$ recognizing the language L:*

$$(n, p) \prec (\hat{n}, \hat{p}) \to f(n, p) < f(\hat{n}, \hat{p})$$

2. *If A and $\hat{A}$ are two minimal PDA recognizing L then:*

$$f(n, p) = f(\hat{n}, \hat{p})$$

Proof. Let us show (by contradiction) that function f does not exist. Let f satisfies both conditions 1 and 2. If we fix the number of stack symbols to two then any minimal push down automaton uses just one state for any n by Corollary 2.2.4.1. If we fix the number of stack symbols to 0, then minimal automaton needs at least n states to accept $L_2[n]$ and $n - 1$ states to accept $L_2[n - 1]$. Therefore $f(2, 1) = f(n, 0)$ and $f(2, 1) = f(n - 1, 0)$, so $f(n, 0) = f(n - 1, 0)$. But according to condition 1, $f(n - 1, 0) < f(n, 0)$. This is a contradiction. □

Because there is no function, which maintains complexity from one minimal automaton to another one, we look for some alternatives. Especially, a function which maintain complexity under some transformation. Labath and Rovan had the same problem on the deterministic push down automata and they introduced function, which preserves complexity under the stack symbol reduction to two.

1.3.1 Defining the Complexity

We shall presents our two ways to address this problem. Using just one state and count the number of stack symbols or fix the number of stack symbols to two and count the number of states. Both classes and measures have their positives and negatives. As it was proved in the Theorem 1.3.1 that complexity preserving function does not exist so our defined measures on this complexity classes preserve complexity measures under well known transitions. These transitions are:

- Reducing number of stack symbols [4].
- Reducing the number of states [3].

One state PDA Due to the fact that any context free language can be represented by a grammar, we can define a measure to be the number of stack symbols using just one state PDA. It is clear that this class defines all context free languages, due to the fact, that any context free grammar can be simulated by one state push down automaton.

In this class the automata shall accept by empty stack. Having just one state, the acceptance by final state means that the automaton accepts ϵ and not every context free language has ϵ. Therefore, we shall use just empty stack acceptance mode in this class. In other words, these two types of acceptance modes are not equivalent in this class.

Two stack symbols PDA Another possibility is to fix the number of stack symbols. Push down automata need at least two stack symbols to accept all context free languages. That is why, we define a second class, where we allow just two stack symbols and measure the number of states.

In this class, we shall show that both acceptance modes are equivalent. However, the minimal automaton depends also on the type of accepting mode we allow to use.

Chapter 2

Push-Down Automata on Regular Languages

Before we consider the context free languages, we shall introduce the complexity properties of push down automata on regular languages and compare them to nondeterministic finite automata. At the beginning, we shall show constructions of push down automata from FSA. We expect state complexity reduction due to stack symbols, which can be used as additional state in some constructions. Then we show that some complexity hierarchies of FSA can totally collapse using just two stack symbols. If we allow any acceptance modes the hierarchy still collapses. Then we shall study this fact by reducing the number of stack symbols to one and comapre accepting modes again. We shall see that the empty stack acceptance allows to collapse this hierarchy again, but using the final state acceptance mode on one stack symbol automaton does not have the same effect.

2.1 Saving States by Adding Stack

We shall study the effect of using stack symbols on the state complexity. We shall thus compare finite state machines to push down automata. For an arbitrary finite state machine, we shall construct a push down automaton. We have to decide, how many stack symbols we allow to use by the push down automaton. Clearly, it does not make sense to allow any number of stack symbols. If we allow that, then it is always possible to find enough stack symbols p such that one state would be enough to accept that language.

Therefore, we shall fix the number of stack symbols for the push down automaton. More precisely, for a given p (p indicates the maximum number of stack symbols of push down automaton is allowed to use) find a push down automaton, which accepts the same language as the given finite state automaton and uses less states.

By definition, the PDA determines the next step based on the top stack symbol in its state. So, the push down automaton can encode one state of an FSA using his own state and some stack symbol. Therefore, with p stack symbols it needs just $\lceil \frac{n}{p} \rceil$ states compared to n state finite state automaton to accept a regular language.

Now, we have to decide on the acceptance mode of the push down automaton. There are two options: by empty stack or by accepting state. In our construction, the automaton can not accept by accepting state, because in some states, the push down automaton can have on the top of the stack a stack symbol, which in combination with that particular state does not represent an accepting state of the finite state automaton. We shall therefore use the empty stack acceptance mode. In a particular state, the automaton can nondeterministically pop the top stack symbol and accepts the input word or halts.

The whole idea can be written down as the next theorem:

Theorem 2.1.1. *For any n state FSA A_1 there exists a PDA A_2 with $\lceil \frac{n}{p} \rceil$ states and p stack symbols such that $N(A_2) = L(A_1)$*

Proof. Let $A_1 = (Q_1, \Sigma, q_1, F, \delta_1)$ be a finite state automaton with n states. We shall construct a PDA $A_2 = (Q_2, \Sigma, \Gamma, \delta_2, p_1, Z_1, \emptyset)$ with $\lceil \frac{n}{p} \rceil$ states, where $\Gamma = \{Z_1, \dots, Z_p\}$ and $Q_2 = \{p_1, \dots, p_{\lceil \frac{n}{p} \rceil}\}$
$(q_{f(k)}, Z_{g(k)}) \in \delta_2(q_{f(j)}, a, Z_{g(j)}) \iff q_k \in \delta_1(q_j, a)$
$(q_{f(k)}, \epsilon) \in \delta_2(q_{f(k)}, \epsilon, Z_{g(k)}) \iff q_k \in F$
where $f(x) = \lceil \frac{x}{p} \rceil$ and $g(x) =$ x mod p.

The construction idea of A_2 is similar to the construction for intersection of regular languages given two finite state automata. Instead of using cross product of the states, the push down automaton represents each state of the finite state automaton A_1 by a pair consisting of the state and the stack symbol on the top of the stack. The stack of A_2 shall contain at most one symbol. Each transition of the automaton A_1 from a state q to s is represented by a corresponding transition of the state and in the top stack symbol in the automaton A_2. Additionally, any accepting state s_a of A_1 has a corresponding pair s and Z. The A_2 in the state p with the Z stack symbol on the top of the stack can nondeterministically decide to pop this stack symbol and empty its stack. If the stack is emptied before the automaton A_2 has finished the processing of an input word, A_2 shall halt, otherwise A_2 accepts this word if and only if the automaton A_1 accepts. □

The proof of Theorem 2.1.1 shows a construction of a push down automaton accepting by empty stack. The question arises, how the upper bound would change, if we want to construct a push down automaton using the final state acceptance mode. We can use the previous construction and instead of popping the stack symbol on the current pair of state and top stack symbol, the automaton moves to a new accepting state.

Theorem 2.1.2. *For each n state FSA A_1 exists a PDA A_2 with $\lceil \frac{n}{p} \rceil + 1$ states and p stack symbols such that $L(A_2) = L(A_1)$*

Proof. We use the same construction as in the proof of Theorem 2.1.1, but the push down automaton A_2 uses a new accepting state q_{fin}. Formally, let $A_1 = (Q_1, \Sigma, q_1, F, \delta_1)$ be a finite state automaton with n states. We construct a PDA $A_2 = (Q_2, \Sigma, \Gamma, \delta_2, p_1, Z_1, \{q_{fin}\})$ with $\lceil \frac{n}{p} \rceil$ states, where $\Gamma = \{Z_1, \ldots, Z_p\}$ and $Q_2 = \{p_1, \ldots, p_{\lceil \frac{n}{p} \rceil}\} \cup \{q_{fin}\}$
$(q_{f(k)}, Z_{g(k)}) \in \delta_2(q_{f(j)}, a, Z_{g(j)}) \iff q_k \in \delta_1(q_j, a)$
$(q_{fin}, Z_{g(k)}) \in \delta_2(q_{f(k)}, \epsilon, Z_{g(k)}) \iff q_k \in F$
where $f(x) = \lceil \frac{x}{p} \rceil$ and $g(x) = $ x mod p.

The PDA A_2 instead of popping a symbol from the stack on the accepting pair, moves to the accepting state q_{fin}. □

This upper bound construction introduces one more state, the accepting state compared to the previous upper bound construction. Due to the accepting state, the number of states will be all the time higher in this construction then in the previous. In some cases, it is just a constant factor, but we shall show cases with bigger improvement in our complexity measures.

We can find similarity with one state push down automata. Any push down automaton with one state accepts by empty stack. Accepting by state with one state only allows to accept $\emptyset$ or Σ^*. That is why push down automaton needs at least two states to accept arbitrary context free language by state.

2.2 Lower Bounds for PDA on Regular Languages

In the previous section, we have found upper bounds for push down automata on regular languages using the pair of a stack symbol and a state for representing a state of a finite state automaton. Clearly, this construction applies to all of the finite state automata and does not use any properties of the language, which is accepted by that particular automaton.

In this section, we shall introduce regular languages and push down automata, which on some of these languages can use their stack cleverly and reduce the complexity and on the others do not.

Before we show particular lower bounds, we shall introduce some particular regular languages.

Notation 2. Let $a_1, \ldots a_n$ be distinct symbols for any $n \geq 1$. Let

- $L_1[n] = a_1^* a_2^* \ldots a_n^*$
- $L_2[n] = \{a_1^{kn} | k \geq 0\}$

For both of these languages, we can construct a finite state automaton with n states. The FSA for the language $L_1[n]$ processes each a_i in a separate state, which allows it to remember, where it is in any currently processed input word. The FSA for the language $L_2[n]$ also uses n states. In each state, the automaton represents the reminder after dividing by n. Moreover, both automata are minimal (considering the number of states) for these languages. In the next part, we shall show that using upper bound constructions for these minimal automata shall result in the minimal push down automatons for the language $L_1[n]$ and not minimal automaton for the language $L_2[n]$ in our complexity measures.

2.2.1 The complexity of $L_1[n]$

Let us consider the first introduced language $L_1[n]$. Using the construction from the proof of the Theorem 2.1.1 on the minimal finite state automaton shall result in a minimal push down automaton for the language $L_1[n]$,if we set number of stack symbols p equal to n. These means, that any one state automata needs at least n stack symbols to accept $L_1[n]$.

Let us start with a simple property that any one state automaton has to have to accept $L_1[n]$. The automaton can not have the same stack symbol on two different input symbols on the top of the stack.

Lemma 2.2.1. *Let A in $\mathscr{D}(1, p)$ be an automaton and accepting the language $L_1[n]$, where $p, n \in N$. Suppose $\delta(q_0, a_i, Z) \neq \emptyset$ and $\delta(q_0, a_j, \hat{Z}) \neq \emptyset$ for $i \neq j$. Then $Z \neq \hat{Z}$.*

Proof. Suppose that one state push down automaton A accepting the language $L_1[n]$ can do computation step on a_i and a_j on the same stack symbol Z, for $i \neq j$.

The A accepts $w_i = a_i^m$ and $w_j = a_j^m$, for $m \geq 2p$.

During the accepting computation on the word w_i, the automaton A cycles on a_i and pushes word γ_i on the stack . The $|\gamma_i| > 0$, otherwise it could not accept a_i^m for any $m \in N$. Therefore, the automaton has some ϵ-cycle, which pops $|\gamma_i|$ from the stack, so the automaton A can accept w_i by the empty stack.

The same apply for word w_j. The automaton pushes γ_j on the stack while reading the input symbol a_j and $|\gamma_j| > 0$. A also have ϵ-cycle, which pops $|\gamma_j|$ from the stack.

Without loss of generality, let $i < j$. Now, let us show that A accepts word $\hat{w} = a_j a_i \notin L_1[n]$. We know that the automaton A accepts word $a_j a_j$. After the automaton A processes the first input symbol a_j, it has to keep the stack symbol Z on the top of the stack, otherwise it could not continue processing any other a_j. Therefore, the automaton A can do a computation step on a_i after processing a_j. Then after processing the input word $\hat{w}$, it has $\gamma_j \gamma_i$ on the stack, but we showed earlier that there exists ϵ-cycles, which pops both γ_j and γ_j from the stack. $\square$

Now, we are ready to prove that p stack symbols are necessary and sufficient to accept[1] $L_1[p]$. We shall divide the proof into the two parts. In the first part, we shall construct a PDA accepting $L_1[p]$ using p stack symbols. Note that this automaton behaves similarly as the minal FSA accepting $L_1[p]$, but it uses the top stack symbol as a representation of the state of

[1]For a better understanding, we shall use $L_1[p] = L_1[n]$, for $n = p$, where p should represents the number of stack symbols needed by any one state push down automaton to accept $L_1[p]$.

minimal FSA. In the second part, we prove by contradiction that $p - 1$ stack symbols do not suffice to accept $L_1[p]$.

Theorem 2.2.2. $\Gamma c(L_1[p]) = p, \forall p \in N.$

Proof. We shall show that $\Gamma c(L_1[p]) \leq p$. We shall construct a push-down automaton $A_p = (\{q_0\}, \{a_1, \dots, a_p\}, Z_1, \dots, Z_p, \delta_p, q_0, Z_1, \emptyset)$, where

$$\delta_p(q_0, a_i, Z_i) = \{(q_0, Z_i)\}, \forall i \in \{1, \dots, p\}$$
$$\delta_p(q_0, a_j, Z_i) = \{(q_0, Z_j)\}, \forall i, j \in \{1, \dots, p\} \wedge i \leq j$$
$$\delta_p(q_0, \epsilon, Z_i) = \{(q_0, \epsilon)\}, \forall i \in 1, \dots, p$$

The automaton A_p uses the top stack symbol Z_i as a "state". If Z_i is on the top of the stack, the automaton knows that it already has processed all input symbols $a_1, \dots, a_{i-1}$. The automaton can pop in any time the top stack symbol and empty his stack.

We shall prove $\Gamma c(L_1[p]) \geq p$ by contradiction.

Let A_p in $\mathscr{D}(1, p - 1)$ be an automaton accepting the language $L_1[p]$. By Lemma 2.2.1, the automaton A_p has to use a distinct stack symbol for each a_i. Thus it has to have p stack symbols and can not be in $\mathscr{D}(1, p - 1)$. □

We showed that any one state push down automaton needs at least p stack symbols to accept $L_1[p]$. We can expand this idea. More precisely, any push down automaton accepting $L_1[n]$ using s states and p stack symbols has to satisfy: $s * p \geq n$.

Due to shortage of time, we were unable to create a precise proof. That is why, we present it as a hypothesis.

Hypothesis 1. Let A in $\mathscr{D}(s, p)$ be an automaton accepting the language $L_1[n]$. Then A has to satisfy: $s * p \geq n$.

We think that some idea should apply as in the Lemma 2.2.1. That the automaton on two different input symbols will be in the same state and have the same stack symbol on the top of the stack.

2.2.2 The complexity of $L_2[n]$

Similarly, we can use the upper bound construction in the proof of Theorem 2.1.1 on minimal finite state automaton for the language[2] $L_2[n]$. However, this does not result in a minimal push down automaton in our complexity measures. Instead of using the combination of state and a top stack symbol to represent the state of minimal FSA, the minimal push down automaton uses his stack as a counter. Each word in the language $L_2[n]$ is just some number of blocks of length n. The minimal push down automaton minimizes the number of states by checking the

[2]The minimal finite state automaton requires exactly n states.

length of each block on the stack instead of using states. Due to this property, the minimal push down automaton needs two stack symbols and one state only to accept $L_2[n], n \geq 2$.

Before we introduce the proof and the construction, we shall first prove that no push down automaton with one state and one stack symbol can accept $L_2[n]$ for $n \geq 2$.[3]

The Lemma 2.2.3 is based on the idea that the automaton with one state and one stack symbol cannot pop up the stack symbol from the stack on any input symbol, otherwise the automaton would accept also the input symbol alone.[4]. The last option is that the automaton pops on ϵ the stack symbol, but than the automaton can accept any input word consisting of a symbols.

Lemma 2.2.3. *No PDA using one state and one stack symbol can accept $L_2[n], n \geq 2$.*

Proof. By contradiction, let there exist a push down automaton A with one state q and one stack symbol Z accepting $L_2[n], n \geq 2$. We know that A must use the empty stack acceptance mode. Therefore A has to pop Z from the stack on the input symbol a_1 or ϵ.

If A pops Z on the input symbol a_1 then A accepts the word $a_1 \notin L_2[n]$ for any $n \geq 2$.

If A pops Z on ϵ and pushes on a_1 a word $w \in Z^*$ on the stack then A accepts the word $a_1 \notin L_2[n]$ for any $n \geq 2$.

□

Let us introduce informally the construction of the automaton accepting $L_2[n]$ using one state and two stack symbols. The automaton uses the initial stack symbol Z_2 as a marker for the beginning of a new block of symbols a_1. The additional block of Z_1 of length n is pushed on the stack, if the automaton nondeterministically decides to process additional block of symbols a_1 and each Z_1 represents one a_1 in the block. The automaton can be seen in the Figure 2.1 below.

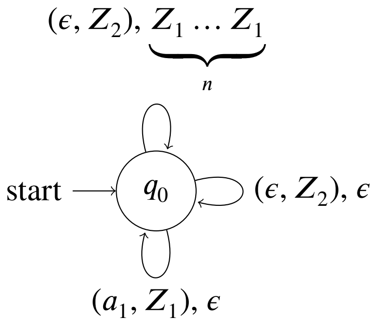

$(\epsilon, Z_2), \underbrace{Z_1 \dots Z_1}_{n}$
start
q_0
$(\epsilon, Z_2), \epsilon$
$(a_1, Z_1), \epsilon$

Figure 2.1: The push down automaton accepting $L_2[n]$ using one state and two stack symbols.

Theorem 2.2.4. $\Gamma c(L_2[n]) = 2,$ *for any $n \geq 2$.*

[3]A finite state automaton using one state can accept $L = \{a_1\}^* = L_2[n]$, for $n = 1$.
[4]The initial stack symbol is poped by the input symbol

Proof. We shall prove that $\Gamma c(L_2[n]) \leq 2$. We shall construct a push down automaton $A = (\{q_0\}, \{a_1\}, \{Z_1, Z_2\}, \delta, q_0, Z_2, \emptyset)$ and $\delta(q_0, \epsilon, Z_2) = \{(q_0, \epsilon), (q_0, \underbrace{Z_1 \dots Z_1}_{n})\}$

$\delta(q_0, a_1, Z_1) = \{(q_0, \epsilon)\}$

The automaton uses the Z_2 stack symbol as indication for staring point of block. It can nondeterministically decide to start processing a next block or empty the stack. The automaton pushes n sized block of Z_1 on stack and for each a_1 it pops Z_1 from the stack until it reaches the symbol Z_2. Then the process repeats again.

The $\Gamma c(L_2[n]) \geq 2$ has been already shown in the Lemma 2.2.3. □

Moreover, this theorem proves the lower bound in our state complexity measure. We can use the same construction of the push down automaton to construct a push down automaton using two stack symbols only. Clearly, this automaton is minimal in this complexity measure (We can not have a push down automatons with zero states). This can be summarized in the next corollary.

Corollary 2.2.4.1. *$Qc(L_2[n]) = 1$, for any $n \geq 1$.*

The previous construction results in a minimal push down automaton that accepts $L_2[n]$ by empty stack. The question arises, how many states uses a minimal push down automaton using two stack symbols accepting the language $L_2[n]$? Clearly, it can not be one state. Therefore the minimal PDA needs at least two states.

Let us introduce a minimal[5] push down automaton using two states and two stack symbols accepting the language $L_2[n]$. The construction of the minimal PDA is similar as in the proof of the Theorem 2.2.4. Instead of popping nondeterministically the symbol Z_2 from the stack and then accept by empty stack, this automaton transit nondeterministicaly to the accepting state, as it can be seen in the figure 2.2 below.

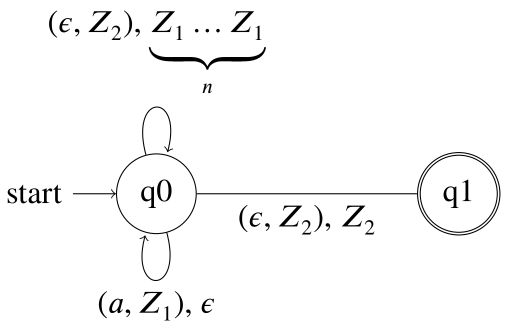


Figure 2.2: The push down automaton accepting $L_2[n]$ with two states and two stack symbols.

[5]by number of states

Theorem 2.2.5. *Let A be a push down automaton using two stack symbols accepting the language $L_2[n], n \geq 2$. Then 2 states are necessary and sufficient to accept $L_2[n]$ by final state.*

Proof. We first show that 2 states are sufficient to accept the language $L_2[n]$ by final state. We shall construct automaton a PDA $A = (Q = \{q_0, q_1\}, \Sigma = \{a_1\}, \Gamma = \{Z_1, Z_2\}, \delta, q_0, Z_2, q_1)$
$\delta(q_0, \epsilon, Z_2) = \{(q_1, Z_2), (q_0, \underbrace{Z_1 \dots Z_1}_{n})\}$

$\delta(q_0, a_1, Z_1) = \{(q_0, \epsilon)\}$

We now show (by contradiction) that two states are necessary to accept the language $L_2[n]$ by final state using just 2 stack symbols. Let assume that an automaton A using one state and two stack symbols exists. Hence it has just one state, then this state has to be accepting state. Also, the automaton accepts a word a_1^n, so it has some transitions on a_1. Therefore automaton accepts word $a_1 \notin L_2[n]$. □

2.3 The complexity of one stack symbol PDA

In the previous part, we proved that any push down automaton needs at least two stack symbols to accept the language $L_2[n]$ with one state. Similarly, we formalized hypothesis that this is not the same case for the $L_1[n]$ language and one state push down automaton needs at least n stack symbols to accept the language $L_1[n]$. Clearly, two stack symbols were enough to collapse our complexity measures hierarchy on the $L_2[n]$ language.

So the question arises, is this the same case for one stack symbol only[6]? Formally, does there exist a push down automaton using one stack symbol accepting $L_2[n]$, which number of states does not depends on n? If yes, how many states it needs to accept $L_2[n]$. Surprisingly the answer is yes. The minimal push down automaton with one stack symbol exists and needs two states only to accept $L_2[n]$.

Before starting a proof, we shall realize that acceptance mode can have effect on the complexity. As in the previous Section 2.1, the stack acceptance construction has a lower state complexity compared to the state acceptance construction. This case also happens in this scenario and has a higher impact on the complexity measures. In the stack acceptance mode, the hierarchy collapses (it does not depend on n),but the complexity measure on one stack symbol automata accepting by state depends on n.

2.3.1 Accepting by empty stack

Let us consider push down automata using one stack symbol and accepting by empty stack. In this setup, we can construct for the sequence of languages $L_2[1], L_2[2], \dots$ the automata

[6]The automatons with one stack symbols are called counter automatons.

$A_1, A_2, \dots$ and each one shall use one stack symbol and two states.

Let us describe the construction of the push down automaton A_n. In the first state, it cycles on ϵ and guesses the number of blocks of a_1 symbol by pushing the $\underbrace{Z_1 \dots Z_1}_{n}$ on the stack. Then it nondeterministically transits on ϵ to the next state and pops one stack symbol. In the second state, it compares the guessed number of a_1 symbols to the actual input word. It is clear, that this automaton accepts the $L_2[n]$ by empty stack. The automaton can be seen in the Figure 2.3.

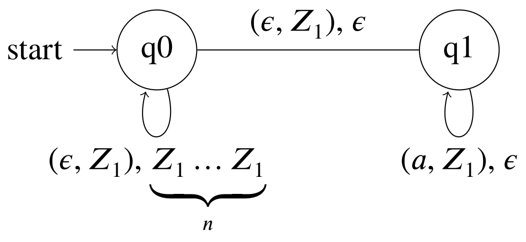

start
q0
q1
$(\epsilon, Z_1), \epsilon$
$(\epsilon, Z_1), \underbrace{Z_1 \dots Z_1}_{n}$
$(a, Z_1), \epsilon$

Figure 2.3: The push down automaton accepting $L_2[n]$ using two states and one stack symbol.

Theorem 2.3.1. *The smallest number of states for any counter automaton accepting the language $L_2[n]$ by empty stack is two, for any $n \geq 2$.*

Proof. We shall prove that there exists a push down automaton with one stack symbol and number of states equal to two. We shall construct a push down automaton
$A = (\{q_0, q_1\}, \{a_1\}, \{Z_1\}, \delta, q_0, Z_1, \emptyset)$ and $\delta(q_0, \epsilon, Z_1) = \{(q_1, \epsilon), (q_0, \underbrace{Z_1 \dots Z_1}_{n})\}$

$\delta(q_1, a_1, Z_1) = \{(q_1, \epsilon)\}$

The automaton guesses the number of blocks of length n in the state q_0 by pushing blocks of Z_1 on the stack. Then, in the q_1 state, the automaton compares number of Z_1 to a_1 symbols from the input.

By Lemma 2.2.3, the automaton A accepting $L_2[n]$, $n \geq 2$ with one state and one stack symbol does not exists. So our automaton A is minimal given the number of states with one stack symbol.

□

2.3.2 Accepting by state

Finally, we focus on the push down automata using one stack symbol accepting by final state. Even though, the previous automata have one stack symbol, we were able to construct a sequence of automata, which accept each language $L_2[n]$ using two states only. Let us show that this is not the case for automata using one stack symbol and accepting by final state.

Theorem 2.3.2. *The smallest number of states for any push down automaton using one stack symbol accepting language $L_2[n]$ by final state is n, for any $n \geq 2$.*

Proof. We shall show the upper bound. The automaton looks the same way as the finite state automaton accepting $L_2[n]$ language, ignoring the stack entirely.

We shall show (by contradiction) that number of states has to be at least n. Let A be a push down automaton with one stack symbol and $n-1$ states accepting language $L_2[n]$. Moreover, let A accepts $w = a_1^m$, n divides m, $m \geq 2n$. Then A cycles on a_1-symbol during the accepting computation on the input word w. Let the size of the cycle be j, $1 \leq j \leq n-1$. Let $\hat{w} = a_1^{m-j}$. The automaton A cycles on $\hat{w}$ one time less then on the accepting computation on the w. That is why, the automaton A accepts $\hat{w} \notin L_2[n]$, because it can not detect the difference between the $\hat{w}$ and w. □

Note 2. In this proof, we have not referenced anything about the stack and the top stack symbol. Having only one type stack symbol does not help the automaton to have different behavior on the same state compare to any finite automaton. The automaton can increase, decrease or not change the size of the stack at all during the cycle in the accepting computation. The acceptation is not affected by the size of the stack. The only one problem can occur, when the stack is emptied before the accepting state is reached. Taking a shorter word then any accepted input word from $L_2[n]$ allow us to bypass this issue[7].

2.4 Conclusion

In this chapter, we showed some constructions on the FSA and PDA. These constructions allow us to reduce the number of states of FSA by using the stack symbols. In some cases the constructions could be optimal. We have shown cases, when these constructions are not optimal under the state complexity measure. Especially, the hierarchy collapses under the PDA and one stack symbol PDA. The main reason is that they can use nondeterminism and the stack to guess the number of input symbols and save that guess on the stack. Then the verification of the guess for each language require the same number of states or stack symbols.

[7]If the automaton A does not halt on the accepting computation of the longer input word then it can not halt on the shorter word on the same cycle.

Chapter 3

Push down automata on context free languages

In the previous chapter, we have studied the complexity of push down automata on the regular languages. In some cases, improvement in the complexity measure was achieved. As discussed in the Chapter 1, there is no function, which maintains complexity from one minimal automaton to another for the same language. We have introduced two subclasses, where we shall study the complexity measure on the PDA. These subclasses are: one state push down automata and PDA using just two stack symbols. For the first one, we shall measure the number of stack symbols and for the second one, we shall measure the number of states. In case an automaton uses more then two stack symbols, we can use standard construction (mentioned in the Chapter 1) to construct an equivalent automaton in our subclasses.

We shall start with the number of stack symbol complexity on one state automata. We shall show a lower bound in this subclass for this measure and analyze the acceptance modes. Then we shall study the second subclass and introduce the equivalence between the accepting modes in this subclass. In the end, we shall analyze the upper bound in the subclass.

3.1 The Number of Stack Symbols

In this section, we shall consider the stack symbols complexity on one state push down automata. We define a sequence of languages L_p and then prove that any push down automaton accepting L_p and using one state needs at least $p + 1$ stack symbols

Let us consider the following sequence of context free languages.

Notation 3. Let $a_1, \dots, a_p, b_1, \dots, b_p$ be distinct symbols for any $p \geq 1$.

Let $\Sigma_p = \{a_1, \dots, a_p, b_1, \dots, b_p\}$

$$L_p = \{w(h(w))^R | w \in \{a_1, a_2, \dots, a_p\}^*\}$$

where h is the homomorphism defined by $h(a_i) = b_i$, for each $a_i \in \{a_1, a_2, \dots, a_p\}$

Knowing that we consider only one state PDA, we can analyze, how the minimal automaton could accept the language L_p. The easiest option to analyze is the final state acceptance mode. The final state can be only the initial state. That is why automaton accepts any prefix of any word $w \in L_p$.

Before starting to formulate any lemmas and theorems about the lower and upper bound for any push down automaton accepting L_p, it is worth mentioning that any push down automaton can pop maximally one stack symbol at any computation step. Our proofs of lower bounds use this property.

We shall present some simple observations:

Note 3. Simple observation on push down automata:

- By definition a push down automaton can pop max one stack symbol from the stack in one computation step.
- Let A in $\mathscr{D}(1, s)$ be an automaton accepting the language L_p, where $p, s \in N$. Then A has to use empty stack acceptance mode.

Our intention is to prove that the number of stack symbols complexity of the language L_p for one state PDA is exactly $p + 1$. To prove the lower bound, we shall first prove several lemmas. Each lemma exhibits a property, each push down automaton accepting L_p has to have.

Let us start with the first property. Any one state push down automaton accepting L_p has to change the stack on each input symbol, otherwise the automaton would not be able to match a_i with the corresponding b_i.

Lemma 3.1.1. *Let A in $\mathscr{D}(1, i)$ be an automaton accepting the language L_p, where $p, i \in N$. Let $x \in \Sigma_p$ be an input symbol. Then in each accepting computation on $w \in L_p$, $w = ux^j v, j \geq 1, u, v \in \Sigma_p^*$, A has to modify the stack while reading x.*

Proof. Let $x \in \Sigma_p$ be a symbol, which does not modify the stack in an accepting computation and let A accepts $w = ux^j v \in L_p, j \geq 1, u, v \in \Sigma_p^*$. Then A also accepts $\hat{w} = ux^{j+1}v \notin L_p$. □

The previous lemma exhibits the behavior of any one state push down automaton accepting the language L_p. The next property shows that any one state push down automaton accepting the language L_p has to pop some stack symbol from the stack on the input symbol b_i during the accepting computation . Otherwise we can show that the automaton can accept a word with multiple b_i not having a corresponding a_i counterpart.

Lemma 3.1.2. *Let A in $\mathscr{D}(1, p)$ be an automaton accepting the language L_p, where $p \in N$. Then $\forall i\ \exists Z \in \Gamma$ such that $(q_0, \epsilon) \in \delta(q_0, b_i, Z)$.*

Proof. Let b_k be such input symbol that on b_k A does not pop any stack symbol from the stack on the b_k Formally, $\forall Z \in \Gamma(q_0, \epsilon) \notin \delta(q_0, b_k, Z)$ and let A accepts $w = a_k^m b_k^m, m > |\Gamma|$. By the Lemma 3.1.1, the automaton A has to modify the stack on the b_k. Then there exists only two options, what A can do on the input symbol b_k.

Then automaton A:

1. does not change the size of the stack on the input symbol b_k. Formally, $\exists Z, \hat{Z} \in \delta(q_0, \hat{Z}) \in \delta(q_0, b_k, Z)$. Hence the A accepts $w = a_k^m b_k^m$ for $m > |\Gamma|$, then automaton has to cycle on b_k and some stack symbol have to reappear on the top of the stack. Let the size of the cycle be j.

   Then A accepts a word $\hat{w} = a_k^m b_k^{m+j} \notin L_p$.

2. increases the size of the stack.

   The automaton A accepts $w = a_k^m b_k^m$ using the empty stack acceptance mode. Therefore A has to have some ϵ-cycle, which pops stack symbols from the stack, which are pushed on the input symbol b_k. This ϵ-cycle has to be able to pop any number of stack symbols pushed on input symbols b_k in the word $w = a_k^m b_k^m$. Then automaton A has to accept $\hat{w} = a_k^m b_k^{m+1} \notin L_p$.

□

In the previous lemma, we proved that any push down automaton accepting the language L_p has to pop some stack symbol on each b symbol. The next lemma exhibits these symbols have to be different for different b symbols.

Lemma 3.1.3. *Let A in $\mathcal{D}(1, p)$ be an automaton accepting the language L_p, where $p \in N$. Suppose $(q_0, \epsilon) \in \delta(q_o, b_i, Z)$ and $(q_0, \epsilon) \in \delta(q_o, b_j, \hat{Z})$ for $i \neq j$. Then $Z \neq \hat{Z}$.*

Proof. Suppose that one state push down automaton A accepting the language L_p pops on b_i and b_j the same stack symbol Z from the stack and let A accepts $w = a_i^m b_i^m$. Then A accepts $\hat{w} = a_i^m b_j^m \notin L_p$. □

Combining the previous lemmas 3.1.2 and 3.1.3, we can infer that one state push down automaton needs at least p stack symbols to accept L_p. The next theorem shows that in fact $p + 1$ stack symbols are required. Before we show the proof of a lower bound and construct upper bound, we shall infer some properties about the δ function from previous lemmas.

Corollary 3.1.3.1. *Let $A = (\{q_0\}, \{a_1, \dots, a_p, b_1, \dots, b_p\}, \{Z_1, Z_2, \dots, Z_p\}, \delta, q_0, Z_i, \emptyset)$ in $\mathcal{D}(1, p)$ be an automaton accepting the language L_p, where $p \in N$. Then $(q_0, \epsilon) \in \delta(q_0, b_i, Z_{s(i)})$, where the function s defines some permutation on $\{1, \dots, p\}$.*

We have prepared everything to prove the complexity theorem for the language L_p. We divide the proof into the two parts. In the first part, we construct a push down automaton accepting the language L_p with $p + 1$ stack symbols. In the second part, we shall show that the automaton is minimal.

Obviously, the push down automaton has to satisfy the necessary conditions (the previous lemmas) to accept the language L_p. The necessary conditions only require p stacks symbols, but our constructed push down automaton uses one more stack symbol as the initial symbol. Then the initial symbol is used as an indicator of a processing phase of the automaton. The automaton can be in the two phases:

- processing a symbol
- processing b symbol

The initial stack symbol "helps" the automaton to recognize the phase.

Theorem 3.1.4. $\Gamma c(L_p) = p + 1, \forall p \in N$

Proof. We shall show that $\Gamma c(L_p) \leq p + 1$. We shall construct a push-down automaton $A_p = (\{q_0\}, \{a_1, ..., a_p, b_1, .., b_p\}, \{Z_1, ..., Z_{p+1}\}, \delta_p, q_0, Z_{p+1}, \emptyset)$, where
$\delta_p(q_0, a_i, Z_{p+1}) = \{(q_0, Z_i Z_{p+1})\}; \forall i \in \{1 \dots p\}$
$\delta_p(q_0, \epsilon, Z_{p+1}) = \{(q_0, \epsilon)\}$
$\delta_p(q_0, b_i, Z_i) = \{(q_0, \epsilon)\}; \forall i \in \{1 \dots p\}$

The automaton pushes on each input symbol a_i the stack symbol Z_i on the stack. Similarly, the automaton pops the Z_i from the stack on the input symbol b_i. The initial stack symbol is Z_{p+1} and is kept on the top of the stack while reading a_i symbols. On any a_i input symbol, can nondeterministically change the top stack symbol to Z_i instead of pushing the new Z_i on the stack. This moves the push down automaton to the next phase. In this phase it is comparing the reverse order of b symbols to a symbols by poping Z symbols from the stack. In the end of computation, the push down automaton A_p accepts by empty stack.

We shall prove $\Gamma c(L_p) \geq p + 1$ by contradiction.

Let A in $\mathscr{D}(1, p)$ be an automaton accepting the language L_p and let Z_i be the initial stack symbol. By corollary (3.1.3.1): $(q_0, \epsilon) \in (q_0, b_i, Z_{s(i)})$, where the function s defines some permutation on $\{1, \dots, p\}$ for some $i \in 1 \dots p$. Then A accepts $w = b_{s(i)} \notin L_p$. $\square$

This theorem shows the last condition that any push down automaton has to satisfy to accept the language L_p. It shows that it needs exactly $p + 1$ stack symbols. Any other push down automaton with p stack symbols would accept some $b_i \notin L_p$.

Also, we showed that the complexity hierarchy is not bounded. We expected this result, because the number of nonterminals in context free grammars can not be bounded. Using the construction from a minimal context free grammar to an equivalent one state push down automaton shows an unbounded hierarchy.

3.2 The Number of States

In the previous section, we went through push down automata using one state. In this section we shall consider automata using only two stack symbols and any number of states. It is clear, that this family also defines all context free languages. At the beginning, we shall discuss the equivalence of acceptance modes. The one state family of PDA was forced to use the empty stack acceptance mode since the final state acceptance mode was limited to specific languages. In the end, we shall introduce some upper bounds on some languages.

3.2.1 Equivalence of Acceptance Modes

This family of automata allows two types of acceptance mode, empty stack or final state. Both of them can be used to accept any context free language. This is in contrast with the one state push down automata. They could not use a final state acceptance mode to accept all context free languages.

Let us prove the equivalence between these two acceptance modes. We can not use the standard constructions to prove equivalence, because both of them introduce a new stack symbol. The family of automata we consider allows two only. The problem can be solved by prefix encoding. Since the new automaton has fever stack symbols, each stack symbol Z is represented by a string $h(Z)$. The same method was used by Goldstine, Price and Wotschke in their research *On reducing the number of stack symbols*[4]. This constructions was briefly introduced in the Chapter *Preliminaries and Definitions*.

The question arises, which of these constructions should we use? We would like to reduce the three stack symbols to two stack symbols. The nondeterministic reduction looks promising, mainly because of the square root in comparison to the deterministic reduction. The nondeterministic reduction requires three mapping functions $f, g, \bar{g}$. Moreover, the nondeterministic reduction requires states $[q, \$\alpha]$ and $[q, \$\beta]$, where α ranges over all prefixes of $f(Z)$ and $\bar{g}(Z)$ and β ranges over all nonempty suffixes of $g(Z)$. These requirements already double the number of states. The deterministic reduction also results in doubling the number of states, but with simpler construction.

That is why we shall use the deterministic construction with some small changes. In our construction, let A in $\mathscr{D}(s, 3)$ be an automaton using stack symbols H_1, H_2, H_3. Then we construct an automaton B using two stack symbols Z_1, Z_0 and each stack symbol H_i is represented in B by a string $h(H_i)$ of symbols Z_1 and Z_0. The states of B are pairs of the form $[q, \gamma]$, where q is a state of A and γ is a proper prefix[1] of $h(H_i)$. The idea is that the

[1]The string is a proper prefix of xy if $y \neq \epsilon$

automaton B reads the encoded stack symbol from the stack and saves it to the current state. Then automaton B does the same computation as A. For mapping h, we have chosen:

- $h(H_1) = Z_1$
- $h(H_2) = Z_0Z_0$
- $h(H_3) = Z_1Z_0$

.This encoding is prefix encoding and variable-length encoding. The variable-length encoding results in slightly fewer states compared to fixed length encoding. In the fixed length encoding, the symbol H_1 would be encoded to Z_0Z_1. In our case, the automaton would need states $[q, \epsilon], [q, Z_1], [q, Z_0]$, but our encoding h needs just $[q, \epsilon], [q, Z_0]$ states, where q is a state of A. Let us see the example in the table 3.1.

PDA A		PDA B	
State	Stack	State	Stack
q	$\dots H_j$	$[q, \epsilon]$	$\dots \underbrace{Z_{i_1} \dots Z_{i_s}}_{h(H_j)}$
		$[q, Z_{i_s} \dots Z_{i_2}]$	$\dots Z_{i_1}$
p	$\dots H_{j_1} \dots H_{j_k}$	$[p, \epsilon]$	$\dots \underbrace{Z_{f_1} \dots Z_{f_n}}_{h(H_{j_1}) \dots h(H_{j_k})}$

Figure 3.1: Example of one computation step of PDA A simulated on PDA B using $(p, \dots H_{j_1} \dots H_{j_k}) \in \delta(q, a, \dots H_j)$, where a is some input symbol. The **s** can be maximally equal to two.

Finally, the accepting states of B shall be all states $[q, \epsilon]$, where q is an accepting state in A. The last item to decide is the initial stack symbol of B. Without loss of generality, we assume that H_1 is the initial stack symbol of A. Otherwise, we can rename the stack symbols and set name of initial stack symbol to H_1.

Note 4. Note that in case of A having an initial symbol H_2 or H_3, our encoding would result in the initial content of the stack of B to have two stack symbols.

We shall now write down our construction.

Lemma 3.2.1. *Let A in $\mathscr{D}(s, 3)$ be an automaton. Then there exists a push down automaton B using two stack symbols and $2s$ states such that $L(B) = L(A)$.*

Proof. Let $A = (Q_a, \Sigma, \Gamma_a = \{H_1, H_2, H_3\}, \delta_a, q_a, H_1, F_a)$ be a push down automaton with s states. We shall construct a PDA $B = (Q_b, \Sigma, \Gamma_b = \{Z_0, Z_1\}, \delta_b, [q_a, \epsilon], Z_1, F_b)$, where
$Q_b = Q_a \times \{\epsilon, Z_0\}$

We construct δ_b for every $s \in \Sigma$ and $p_a, q_a \in Q_a$ as follows:

If Z_1 is on the top, we can simulate the step immediately.

$([q_a, \epsilon], h(H_{i_1}) \dots h(H_{i_n})) \in \delta_b([p_a, \epsilon], s, Z_1) \iff (q_a, H_{i_1} \dots H_{i_n}) \in \delta_a(p_a, s, H_1), s \in \Sigma$

if Z_0 is on the top, the automaton needs one more stack symbol to decode the right symbol.

$([q_b, Z_0], \epsilon) \in \delta_b([q_b, \epsilon], \epsilon, Z_0)$

$([q_a, \epsilon], h(H_{i_1}) \dots h(H_{i_n})) \in \delta_b([p_a, Z_0], s, \mathbf{Z_1}) \iff (q_a, H_{i_1} \dots H_{i_n}) \in \delta_a(p_a, s, \mathbf{H_3})$

$([q_a, \epsilon], h(H_{i_1}) \dots h(H_{i_n})) \in \delta_b([p_a, Z_0], s, \mathbf{Z_0}) \iff (q_a, H_{i_1} \dots H_{i_n}) \in \delta_a(p_a, s, \mathbf{H_2})$

Finally, we set $F_b = F_a \times \epsilon$.

The construction uses the h function as an encoding function for stack symbols of the automaton A to stack symbols of the automaton B. Then B simulates the automaton A by decoding the encoded stack symbols from the stack. If Z_1 is on the top of the stack and the automaton is in the state $[q, \epsilon]$ then it can simulate the step of the automaton A in the state q and H_1 as the top stack symbol. On the other hand, if the top stack symbol is Z_0 and it is in the state $[q, \epsilon]$ then the automaton B needs to read one more stack symbol from the stack. So it saves the symbol read in the state. Then it reads the next stack symbol in the state $[q, Z_0]$ and simulates a computation step of A. □

Now we can write construction from empty stack acceptance mode to final state acceptance mode. We omit a formal description of the transformation since it straightforward.

Theorem 3.2.2. *For a push down automaton A using two stack symbols and* **s** *states, there exists a push down automaton B using two stack symbols and* $2s + 2$ *states such that* $L(B) = N(A)$.

Proof. Let us use a general construction, which constructs a new automaton C. This push down automaton accepts the language $L(C) = N(A)$, but it uses three stack symbols and $s + 1$ states. By the previous Lemma 3.2.1, there exists a push down automaton B using two stack symbols and $2s + 2$ states such that $L(B) = L(C)$. □

We would like to use the same Lemma 3.2.1 for opposite implication, but the construction in the proof of the Lemma reduces a three stack symbols automaton to a two stack symbols automaton accepting by final state. That is why, we formulate similar lemma, which reduces to two stack symbol automata accepting by empty stack.

Lemma 3.2.3. *Let A in* $\mathscr{D}(s, 3)$ *be an automaton. Then there exists a push down automaton B using 2 stack symbols and* $2s$ *states such that* $N(B) = N(A)$.

Proof. Let $A = (Q_a, \Sigma, \Gamma_a, \delta_a, q_a, H_1, \emptyset)$, where $\Gamma_a = \{H_1, H_2, H_3\}$. We construct $B = (Q_b, \Sigma, \Gamma_b, \delta_b, [q_a, \epsilon], Z_0, \emptyset)$, where $\Gamma_b = \{Z_0, Z_1\}$ as in the Lemma 3.2.1, but we do not set any accepting final state. The automaton B shall have the states: $Q_b = Q_a \times \{\epsilon, Z_0\}$ and we shall use the same encoding[2] h. Using the deterministic reduction and mapping h, we

[2] $h(H_1) = Z_1, h(H_2) = Z_0 Z_0, h(H_3) = Z_1 Z_0$

construct the δ_b similarly as in the proof of Lemma 3.2.1. The automaton B uses two stack symbols and $2s$ states and $N(B) = N(A)$. □

Now, we are ready to prove the opposite implication.

Theorem 3.2.4. *For a push down automaton A using two stack symbols and* s *states, there exists a push down automaton* B *using two stack symbols and* $2s + 2$ *states such that* $N(B) = L(A)$.

Proof. Let us use a general construction, which constructs a new automaton C. The automaton C is using three stack symbols and $s + 1$ states. By the previous Lemma 3.2.3, there exists a push down automaton B using two stack symbols and $2s+2$ states such that $N(B) = L(C)$. □

We showed that both acceptance modes are equivalent within the class $\mathscr{D}(s, 2)$ and both at least double the number of states.

3.2.2 Upper bounds

In the previous section, we proved the equivalence between the empty stack acceptance mode and the final state acceptance mode in $\mathscr{D}(s, 2)$. In this section we shall consider the state complexity of push down automata using two stack symbols ($\mathscr{D}(s, 2)$) accepting by empty stack and then by final state.

Let us consider the following sequence of context free languages.

Notation 4. Let a, b, c be distinct symbols. Let $\Sigma = \{a, b, c\}$. For each $r \geq 1$ let

- $L = \{w = a^m b^m | m \geq 1\}$
- $L_1[r] = \{c^m | 0 \leq m \leq r\}$
- $L_2[r] = Shuf(L, L_1[r])$

At first, let us discuss the properties of the language L. Note that the language L coincides and thus has the same properties as the language L_p (defined in the Section 3.1), for $p = 1$. We proved that one state automaton needs at least two stack symbols to accept $L_p, p = 1$. Then using the general idea from Theorem 3.1.4, we can construct the minimal one state PDA accepting L.

We shall now modify the L in order to "force" the PDA to check some additional property. Both stack symbols are used for keeping track of symbols a and b. Adding another property to this language should result in increasing the number of states. The property we shall use is the language $L_1[r]$, the number of c symbols should be equal or less then r. Mixing properties of L and $L_1[r]$ languages results in the language $L_2[r]$

Finally, we should decide, which acceptance mode we shall use. Let us use the same acceptance mode as in the previous Section 3.1, empty stack acceptance mode. This results in an easier upper bound construction.

Our automaton has $r + 1$ states. Each state represents the number of c symbols read. If automaton reads the $r + 1$ symbol, then the automaton halts.

Let us consider the PDA using two stack symbols and accepting the language $L_2[r]$ by empty stack.

Theorem 3.2.5. *there exists a PDA A_r using two stack symbols and r + 1 states such that $N(A_r) = L_2[r]$.*

Proof. We shall construct a PDA $A_r = (\{q_0, \dots, q_r\}, \{a, b, c\}, \{Z_1 Z_2\}, \delta_r, q_0, Z_2, \emptyset)$
$(q_i, Z_1 Z_2) \in \delta_r(q_i, a, Z_2), \forall i \in \{0, \dots, r\}$
$(q_i, \gamma) \in \delta_r(q_{i+1}, x, \gamma), \forall i \in \{0, \dots, r-1\}, \gamma \in \{Z_1, Z_2\}, x \in \Sigma$
$(q_i, \epsilon) \in \delta_r(q_i, b, Z_1)$

The stack symbol Z_1 is used as a counter. On input symbol a, the automaton pushes Z_1 on the stack and it keeps the stack symbol Z_2 on the top. The automaton keeps Z_2 on the top of the stack. This indicates A_r is still reading the part of the input containing the a symbols. The automaton A_r can nondeterministically pop Z_2 on ϵ and start to pop Z_1 on each b. On symbol c, the automaton changes the state from q_i to q_{i+1}. In case A_r is in the state q_r and reads the input symbol c, it halts. □

Chapter 4

Complexity of Operations

In previous chapters, we have introduced two subclasses: one state push down automata $\mathscr{D}(1, p)$ and two stack symbols push down automata $\mathscr{D}(s, 2)$ and show in some complexity bounds in these subclasses. In this chapter, we shall study complexity of operations on languages. Thus for context free languages L_1 and L_2 given by PDA A_1 and A_2 in $\mathscr{D}(1, p)$ [or $\mathscr{D}(s, 2)$] for $L_1 \cup L_2$ ($L_1.L_2$ or L_1^*) based on the complexity A_1 and A_2.

4.1 Stack Symbols Complexity

In this section, we shall present constructions for *Union*, *Concatenation*, *Klenee-Star* in the class $\mathscr{D}(1, p)$ of push down automata. The idea behind the constructions is to use additional stack symbol, which represents the bottom of the stack and helps the constructed automaton to simulate another state of push down automaton and detect the acceptance. We shall omit the formal description of the constructions since it is straightforward.

Union

Theorem 4.1.1. *Let A in $\mathscr{D}(1, p_1)$ be an automaton and B in $\mathscr{D}(1, p_2)$ be an automaton for any integers $p_1, p_2 \geq 1$. Then $p_1 + p_2 + 1$ unique stack symbols are sufficient for a one state push down automaton to accept the language $N(A) \cup N(B)$.*

Proof. In order to construct a one state push down automaton C with $p_1 + p_2 + 1$ stack symbols for the language $N(A) \cup N(B)$ we simply use a new initial stack symbol. On this initial stack symbol automaton reads ϵ and nondeterministically decides to replace it with initial stack symbol of push down automaton A or B and continues the computation as that particular automaton. □

Concatenation

Theorem 4.1.2. *Let A in $\mathscr{D}(1, p_1)$ be an automaton and B in $\mathscr{D}(1, p_2)$ be an automaton for any integers $p_1, p_2 \geq 1$. Then $p_1 + p_2 + 1$ unique stack symbols are sufficient for a one state push down automaton to accept the language $N(A)N(B)$.*

Proof. Let us construct a push down automaton C using one state and $p_1 + p_2 + 1$ stack symbols and accepting the language $N(A)N(B)$. The automaton C has a new initial stack symbol Z. On this stack symbol, the automaton pushes the initial stack symbol of the automaton A and starts a simulation of A. When the simulation reaches the bottom stack symbol Z, then C replaces Z by the initial stack symbol of B and starts a simulation of B. The automaton C accepts by empty stack. □

Klenee-Star

Theorem 4.1.3. *Let A_1 in $\mathscr{D}(1, p_1)$ be an automaton for any integer $p_1 \geq 1$. Then $p_1 + 1$ unique stack symbols are sufficient for a one state push down automaton to accept the language $N(A_1)^*$.*

Proof. Let us construct a push down automaton C using one state and $p_1 + 1$ stack symbols and accepting the language $N(A_1)^*$. The automaton C uses exactly same stack symbols as A_1 and has one additional initial stack symbol Z. If the automaton C has on the top of the stack symbol Z then it can pop it and accept by empty stack or add the initial stack symbol of the automaton A_1 and simulates it until it reaches the top stack symbol Z again. Then the process repeats. □

Summary

We shall summarize our findings in the Table 4.1 below. The p_1 represents the number of stack symbols of the first automaton and p_2 represents the number of stack symbols of the second automaton.

operation	number of stack symbols
$\cup$	$p_1 + p_2 + 1$
.	$p_1 + p_2 + 1$
$*$	$p_1 + 1$

Table 4.1: Sufficient number of stack symbols.

The proofs shown resemble standard proofs showing closure of context free languages under these operations. In particular, the additional stack symbol is similar to a new additional nonterminal in context free grammars constructions. This nonterminal is also used as the initial nonterminal for the new grammar. This similarity is due to the equivalence of push down automata and context free grammars.

4.2 State Complexity

We shall do the same as in the previous section, but on the second subclass. In this subclass, we showed that final state acceptance mode and empty stack acceptance mode are equivalent. Therefore, we shall show proofs for construction of empty stack acceptance mode and add a note for a state acceptance mode construction.[1]

We shall omit a formal description of the constructions since it is straightforward.

Union

Theorem 4.2.1. *Let A in $\mathscr{D}(r, 2)$ be an automaton and B in $\mathscr{D}(s, 2)$ be an automaton for any integers $r, s \geq 1$. Then $r + s + 1$ states are sufficient for a push down automaton using two stack symbols to accept the language $N(A) \cup N(B)$.*

Proof. Let C be a push down automaton such that $N(C) = N(A) \cup N(B)$. The automaton C in its initial state nondeterministically decides, if the input word w belongs to $N(A)$ or $N(B)$. Then it starts the simulation of the corresponding push down automaton.

Without loss of generality, we can say that both automata A and B use the same stack symbols. Then $r + s + 1$ states are sufficient to simulate both automata by C. $\square$

Note 5. The same construction works, if the automata A uses r states and B uses s states both accept by final state. Then C accepts $L(A) \cup L(B)$ by final state using $r + s + 1$ states.

Concatenation

Theorem 4.2.2. *Let A in $\mathscr{D}(r, 2)$ be an automaton and B in $\mathscr{D}(s, 2)$ be automaton for any integers $r, s \geq 1$. Then $2(r + s)$ states are sufficient for a push down automaton using two stack symbols to accept the language $N(A)N(B)$.*

Proof. We shall present the idea of the construction of a push down automaton C such that $N(C) = N(A)N(B)$. We shall start by using a new stack symbol H. The automaton C starts with the new stack symbol H and places on the top of the stack the initial stack symbol of A, $Z_{0,A}$. The stack should look like $HZ_{0,A}$, where $Z_{0,A}$ is on the top of the stack. Then C starts the simulation of automaton A until A reaches the bottom stack symbol H. Then automaton C replaces the stack symbol H by initial stack symbol of automaton B and starts the simulation of the automaton B.

The automaton C uses 3 stack symbols and $r + s + 1$ states. By Lemma 3.2.3, there exists an equivalent push down automaton $\hat{C}$ using 2 stack symbols and $2(r + s) + 2$ states. $\square$

Note 6. The same construction works for the final state acceptance mode. We similarly construct an automaton C using three stack symbols, which works as follows. It keeps H[2]

[1] Both constructions will be similar.
[2] the additional stack symbol

and pushes $Z_{0,A}$[3] on the stack and starts simulation of the automaton A. When simulation of A reaches the final state of A, it leaves some "garbage" on the stack with H on the bottom of this "garbage". Then C ignores it and pushes the $HZ_{0,B}$[4] on the stack and simulates the automaton B until it reaches the final state of B. Then C accepts by final state. If any of the simulated automata, A or B encounter H as the top symbol on the stack, C halts. The C uses three stack symbols and $r+s+1$ states.

Then by Lemma 3.2.1, there exists an equivalent automaton using two stack symbols and $2(r+s)+2$ states.

Klenee-Star

Theorem 4.2.3. *Let A in $\mathscr{D}(r,2)$ be an automaton for any integers $r \geq 1$. Then $2r$ states are sufficient for a push down automaton using two stack symbols to accept the language $N(A)^*$.*

Proof. We start by constructing a push down automaton using three stack symbols. The two stack symbols are the same as the automaton A is using and a new stack symbol H is the initial stack symbol of C. Let Z_0 be an initial stack symbol of A. Then C with H on the top of the stack pushes Z_0 on the stack or pops the H from the stack in its initial state. When it pushes Z_0, it moves to the initial state of A and starts simulation of A. When the simulated A reaches H on the top of the stack then C returns on ϵ-move to its initial state.

The C accepts $L(A)^*$ using three stack symbols and $r+1$ states. By Lemma 3.2.3,there exists an equivalent push down automaton using 2 stack symbols and $2r+2$ states. □

Note 7. A similar constructions works for a final state acceptance mode. We set the initial state of C as the final state. The initial stack symbol is H again and it represents a new bottom of the stack for the simulated automaton A. Then C pushes[5] Z_0 and starts the simulation of A. When A reaches its final state then C pushes H on the top of the stack and returns to the initial state on an ϵ-move. If the simulated A reaches this stack symbol, the simulated A halts. Therefore C halts.

Then by Lemma 3.2.1, there exists an equivalent automaton using two stack symbols and $2r+2$ states.

Summary

We shall summarize our findings in the Table 4.2. The r represents the number of states of the first automaton and s represents the number of states of the second automaton.

We have seen that in our constructions the sufficient number of states for a particular operation does not depend on acceptance mode.

[3]the initial stack symbol of automaton A

[4]$Z_{0,B}$ is initial stack symbol of automaton B

[5]An initial stack symbol of A

Acceptance mode	*empty stack*	*final state*
$\cup$	$r + s + 1$	$r + s + 1$
$\cdot$	$2(r + s) + 2$	$2(r + s) + 2$
$*$	$2r + 2$	$2r + 2$

Table 4.2: Sufficient number of states

Conclusion

In this thesis, we have studied descriptional complexity and effect of stack on the descriptional complexity of push down automata on regular and context free languages. Two transformations are presented which, for any finite state automata using n states, reduce the number of states to $\lceil \frac{n}{p} \rceil$ for a given number of stack symbols p. We have shown that these constructions are optimal, if we want to construct one state push down automaton for any given finite state automaton. In the same cases these constructions are not even close to optimal. We have shown that for a particular regular language there exists a minimal finite state automaton using n states while there exists an equivalent push down automaton using two stack symbols and one state. In both cases, we have shown that these automata are minimal.

The descriptional complexity of PDA was not extensively studied, we had to dedicate some part of the work for descriptional complexity on push down automata. We focused on a definition of a "good" measure on push down automata. Two measures have been presented, the state measure and the number of stack symbols measure. Both measures are not "good" measures, because each context free language can be accepted by one state push down automaton or push down automaton using two stack symbols. That is why, we tried to combine these measures and defined a partial ordering on these measures, but in the ordering, we could not compare some minimal automata and we showed that there does not exists any function. which combines these measures in a such way that it maintains compelxity from one minimal automaton to another one for the same language. Therefore, we defined two subclasses of PDA - *one state PDA* and *two stack symbols PDA*.

In the first subclass, we have shown tight bounds for the number of stack symbols. In the second subclass, we have shown equivalence of accepting modes and presented an upper bound for state complexity measure in this subclass. Finally, we investigated the costs of operations in the subclasses. In particular, we condisered union, concatenation and Klenee-Star.

There are a lot of possibilities to expand this work. Due to the lack of time, we did not prove our hypothesis, which generalizes and proves that our transformation from finite state automaton to push down automaton is optimal. Moreover, we have not shown any tight bounds for *two stack symbols PDA* subclass and we miss lower bounds for the operations. Another option is to use totally different descriptional measure. For example the size of the definition of the delta function.

Bibliography

[1] Erzsébet Csuhaj-Varjú and Alica Kelemenová. Descriptional complexity of context-free grammar forms. *Theoretical Computer Science*, 112(2):277–289, 1993.

[2] Michael Domaratzki and Kai Salomaa. Lower bounds for the transition complexity of nfas. In *International Symposium on Mathematical Foundations of Computer Science*, pages 315–326. Springer, 2006.

[3] Jonathan Goldstine, John K Price, and Detlef Wotschke. On reducing the number of states in a pda. *Mathematical systems theory*, 15(1):315–321, 1981.

[4] Jonathan Goldstine, John K Price, and Detlef Wotschke. On reducing the number of stack symbols in a pda. *Mathematical systems theory*, 26(4):313–326, 1993.

[5] Hermann Gruber and Markus Holzer. On the average state and transition complexity of finite languages. *Theoretical Computer Science*, 387(2):155–166, 2007.

[6] Markus Holzer and Martin Kutrib. Nondeterministic descriptional complexity of regular languages. *International Journal of Foundations of Computer Science*, 14(06):1087–1102, 2003.

[7] Martin Kutrib and Andreas Malcher. Context-dependent nondeterminism for pushdown automata. *Theoretical computer science*, 376(1-2):101–111, 2007.

[8] Pavel Labath and Branislav Rovan. Simplifying dpda using supplementary information. In *International Conference on Language and Automata Theory and Applications*, pages 342–353. Springer, 2011.

[9] Andreas Malcher and Giovanni Pighizzini. Descriptional complexity of bounded context-free languages. In *International Conference on Developments in Language Theory*, pages 312–323. Springer, 2007.

[10] 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 (swat 1971)*, pages 188–191. IEEE, 1971.

[11] Alexander Okhotin and Kai Salomaa. Complexity of input-driven pushdown automata. *ACM SIGACT News*, 45(2):47–67, 2014.

[12] Maris Valdats. Boolean circuit complexity of regular languages. *arXiv preprint arXiv:1405.5611*, 2014.

[13] S Yu. Regular languages.[in:] rozenberg g., and salomaa a.(eds.): Handbook of formal languages. *Word, Language, Grammar. Springer*, 1997.

[14] Sheng Yu. State complexity of regular languages. *Journal of Automata, Languages and Combinatorics*, 6(2):221, 2001.