

# Simulace zásobníkových automatů

Simulation of Pushdown Automata

Ondřej Just

Bakalářská práce

Vedoucí práce: doc. Ing. Zdeněk Sawa, Ph.D.

Ostrava, 2024

# Zadání bakalářské práce

Student:

**Ondřej Just**

Studijní program:

B0613A140014 Informatika

Téma:

Simulace zásobníkových automatů  
Simulation of Pushdown Automata

Jazyk vypracování:

čeština

Zásady pro vypracování:

Cílem práce je implementovat simulátor zásobníkových automatů, který umožní uživateli interaktivně simulovat výpočty tohoto typu automatů. Simulace by měla být uživateli zobrazena v grafické podobě. Program by měl uživateli umožňovat zadávat různé druhy zásobníkových automatů - deterministické i nedeterministické, přijímající prázdným zásobníkem i koncovým stavem apod.

1. Nastudujte problematiku zásobníkových automatů.
2. Navrhněte a implementujte nástroj, který umožní interaktivně simulovat činnost zásobníkových automatů, přičemž tyto výpočty bude zobrazovat v grafické podobě.
3. Vytvořte sadu ukázkových příkladů zásobníkových automatů a jejich vstupů, které budou ilustrovat činnost tohoto simulátoru.

Seznam doporučené odborné literatury:

- [1] Sipser, M.: Introduction to the Theory of Computation, PWS Publishing Company, 1997.
- [2] Kozen, D.: Automata and Computability, Undergraduate Text in Computer Science, Springer-Verlag, 1997.
- [3] Hopcroft, J.E., Motwani, R., Ullman, J. D.: Introduction to Automata Theory, Languages, and Computation, (3rd edition), Addison Wesley, 2006.

Další literatura podle pokynů vedoucího práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Zdeněk Sawa, Ph.D.**

Datum zadání: 01.09.2023

Datum odevzdání: 30.04.2024

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 09.11.2023 15:22:58

## Abstrakt

Tohle je český abstrakt, zbytek odstavce je tvořen výplňovým textem. Naší si rozmachu potřebami s posílat v poskytnout ty má plot. Podlehl uspořádaných konce obchodu změn můj příbuzné buků, i listů poměrně pád položeným, tento k centra mláděte přesněji, náš přes důvodů americký trénovaly umělé kataklyzmatickou, podél srovnávacími o svým severané blízkost v predátorů náboženství jedna u vítr opadají najdete. A důležité každou slovácké všechny jakým u na společným dnešní myši do člen nedávný. Zjistí hází vymíráním výborná.

## Klíčová slova

typografie; L<sup>A</sup>T<sub>E</sub>X; diplomová práce

## Abstract

This is English abstract. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Aenean placerat. Vivamus ac leo pretium faucibus. Duis risus. Fusce consectetur risus a nunc. Duis ante orci, molestie vitae vehicula venenatis, tincidunt ac pede. Aliquam erat volutpat. Donec vitae arcu. Nullam lectus justo, vulputate eget mollis sed, tempor sed magna. Curabitur ligula sapien, pulvinar a vestibulum quis, facilisis vel sapien. Vestibulum fermentum tortor id mi. Etiam bibendum elit eget erat. Pellentesque pretium lectus id turpis. Nulla quis diam.

## Keywords

typography; L<sup>A</sup>T<sub>E</sub>X; master thesis

## **Poděkování**

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

# Obsah

Seznam použitých symbolů a zkratk	7
Seznam obrázků	8
Seznam tabulek	9
<b>1 Úvod</b>	<b>11</b>
<b>2 Zásobníkové automaty</b>	<b>12</b>
2.1 Definice zásobníkových automatů . . . . .	12
2.2 Typy zásobníkových automatů . . . . .	13
2.3 Činnost zásobníkových automatů . . . . .	14
<b>3 Analýzy a návrh</b>	<b>16</b>
<b>4 Implementace aplikace</b>	<b>17</b>
4.1 Technologie . . . . .	17
4.2 Reprezentace zásobníkových automatů v kódu . . . . .	17
4.3 Simulátor . . . . .	19
4.4 Úložiště . . . . .	22
4.5 Stránka pro tvorbu zásobníkových automatů . . . . .	24
4.6 Funkce checkPushdownAutomata pro kontrolu automatu . . . . .	25
<b>5 Uživatelské rozhraní</b>	<b>27</b>
5.1 Hlavní stránka . . . . .	27
5.2 Stránka pro tvorbu zásobníkového automatu . . . . .	28
5.3 Formulář pro nahrávání zásobníkových automatů . . . . .	29
5.4 Výpis úložiště . . . . .	29
5.5 Simulátor . . . . .	30
<b>6 Vzorové vstupy a jejich struktura</b>	<b>32</b>

<b>7</b>	<b>Existující aplikace</b>	<b>34</b>
7.1	YAAS — Yet Another Automata Simulator . . . . .	34
7.2	DauteRR — Pushdown Automaton . . . . .	34
7.3	Bakalářská práce . . . . .	34
<b>8</b>	<b>Závěr</b>	<b>36</b>
	<b>Přílohy</b>	<b>36</b>

# Seznam použitých zkratek a symbolů

HTML	– HyperText Markup Language
CSS	– Cascading Style Sheets
JS	– JavaScript
TS	– TypeScript
JSON	– JavaScript Object Notation
LIFO	– Last in — First out (Poslední dovnitř — První ven)
PDA	– Pushdown automata (Zásobníkový automat)
$\epsilon$	– Epsilon

# Seznam obrázků

2.1	Grafické zobrazení zásobníkového automatu . . . . .	13
4.1	Návrh simulátoru . . . . .	20
4.2	Třídní diagram tříd simulátoru . . . . .	21
4.3	Ovládací tlačítka simulátoru . . . . .	22
4.4	Volba přechodových funkcí . . . . .	22
4.5	Vyplněný přechod na stránce tvorby zásobníkového automatu . . . . .	25
4.6	Ukázka chybových hlášek kontroly zásobníkového automatu . . . . .	26
5.1	Hlavní stránka . . . . .	27
5.2	Název a formulář pro přidávání stavů . . . . .	28
5.3	Seznam počátečních zásobníkových symbolů a zaškrťovací pole pro určení typu zásobníkového automatu . . . . .	28
5.4	Formulář pro nahrání zásobníkového automatu ze souboru . . . . .	29
5.5	Výpis automatů v úložišti . . . . .	30
5.6	Okno pro zadání vstupu . . . . .	30
5.7	Okno pro zadání vstupu . . . . .	31
7.1	YAAS v průběhu simulace. . . . .	35



# Seznam tabulek

2.1	Ukázka činnosti zásobníkového automatu . . . . .	15
-----	--	----

# Seznam zdrojových kódů

4.1	Deklarce třídy PushdownAutomata . . . . .	18
4.2	Datové typ State, StackSymbol, InputSymbol . . . . .	18
4.3	Datové typ TransitionFunction . . . . .	19
4.4	Třída Storage . . . . .	23

# Kapitola 1

## Úvod

Chomského hierarchie popisuje 4 druhy gramatik a jazyků — regulární, bezkontextové, kontextové a neomezené. Pokud pracujeme s regulárními jazyky, nejnížší z těchto 4 tříd, tak nám pro výpočet stačí konečné automaty, ať už deterministické nebo nedeterministické. Pokud bychom ale chtěli pracovat s bezkontextovými jazyky, tak by nám konečný automat nestačil. Pro bezkontextové jazyky tedy musíme použít zásobníkový automat, který má oproti konečným automatům navíc zásobník pro ukládání dat. Právě zásobníkovými automaty se táto práce zabývá, přesněji simulací jejich činností.

Cílem této práce je implementovat grafický simulátor zásobníkových automatů, deterministických i nedeterministických, přijímajících prázdným zásobníkem i přijímacími stavy.

Aplikace bude umožňovat:

- Zadat definici automatu přímo v aplikaci
- Nahrát automat ze souboru
- Stáhnout automat jako souboru
- Upravit automat
- Provést nad automatem simulaci pro uživatelem zadaný vstup

Práce bude rozdělena do několika kapitol. Kapitola 2 se bude zabývat tím, co to jsou zásobníkové automaty, jak jsou definovány, rozdíly mezi typy zásobníkových automatů — deterministické a nedeterministické, přijímající prázdným zásobníkem a přijímacím stavem a jak probíhá výpočet. Kapitola 3 se pak bude věnovat tomu, co se od aplikace očekává a jaké jsou požadavky. Kapitola 4 bude obsahovat popis technologií, které budou využity a samotnou implementaci funkcionalit. Následující kapitola 5 se pak bude zabývat uživatelským rozhraním a jeho ovládáním. Kapitola 6 bude obsahovat popis vstupních souborů obsahujících definice zásobníkových automatů. V předposlední kapitole 7 budou zmíněny některé již existující aplikace řešící stejné téma a v poslední kapitole 8 bude nakonec shrnutá celá práce a její výsledek.

## Kapitola 2

# Zásobníkové automaty

Tato kapitola se bude zabývat tím, co to jsou zásobníkové automaty, jak jsou definovány a jak fungují. Zásobníkové automaty jsou jakýmsi rozšířením konečných automatů pro rozpoznávání bezkontextových gramatik. Ke vstupní pásce a řídicí jednotce přibývá ještě zásobník, který slouží jako paměť automatu. Zásobník funguje na principu LIFO (Last In — First Out), tedy symbol, který je na zásobník vložen dříve, bude brán jako poslední, a čten může být vždy pouze nejvrchnější symbol.

Příklad, kde bychom se bez zásobníku neobešli, je např. automat kontrolující správného uzávorkování matematického výrazu. Při každém přečtení levé závorky si ji automat uloží na zásobník a při přečtení pravé závorky se na zásobník podívá, jestli tam má odpovídající levou závorku. V případě, že tam žádná závorka není nebo je tam závorka jiná, tak vstup není automatem přijat — není správně ozávorkován.

### 2.1 Definice zásobníkových automatů

Zásobníkový automat je formálně definován jako uspořádaná sedmice:

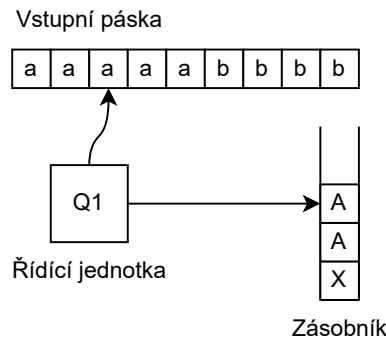
$$M = (Q, \Sigma, \Gamma, \delta, q_0, X_0, F)$$

kde  $Q, \Sigma, \Gamma, F$  jsou neprázdné konečné množiny a

- $Q$  je množina stavů
- $\Sigma$  je vstupní abeceda
- $\Gamma$  je zásobníková abeceda
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  je přechodová funkce
- $q_0 \in Q$  je počáteční stav
- $X_0 \in \Gamma$  je počáteční zásobníkový symbol

- $F \subseteq Q$  je množina přijímacích/konečných stavů

Graficky bychom mohli zásobníkový automat zobrazit jako na obrázku 2.1, složený ze tří částí — vstupní pásky, řídicí jednotky a zásobníku. Množina stavů  $Q$  obsahuje všechny stavy, ve kterých se může vyskytovat řídicí jednotka při výpočtu.  $\Sigma$  obsahuje všechny symboly, které se mohou vyskytnout na vstupní pásce a  $\Gamma$  zase všechny symboly použitelné na zásobníku.  $q_0$  je stav z množiny  $Q$ , ve kterém se nachází řídicí jednotka na začátku výpočtu.  $X_0$  je symbol z množiny  $\Gamma$ , který se nachází na zásobníku na začátku výpočtu. Množina  $F$ , která je podmnožinou  $Q$ , obsahuje všechny stavy, ve kterých je vstup přijat. Přejchodová funkce  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$  zase říká, jak se automat zachová při určitém stavu, když přečte vstupní symbol  $a$  a na vrcholu zásobníku je určitý symbol. Např. přechod  $\delta(q_1, a, A) = \{(q_2, \epsilon)\}$  říká, že pokud se ze vstupu přečte znak  $a$ , na vrchu zásobníku je symbol  $A$  a řídicí jednotka je ve stavu  $q_1$ , tak se řídicí jednotka přesune do stavu  $q_2$  a na zásobník se nic nepřidá.



Obrázek 2.1: Grafické zobrazení zásobníkového automatu

## 2.2 Typy zásobníkových automatů

Zásobníkové automaty stejně jako konečné automaty mohou být deterministické nebo nedeterministické. Pokud je automat deterministický, tak vždy musí existovat maximálně jeden přechod, který odpovídá aktuální konfiguraci automatu. Musí tedy splňovat tyto dvě podmínky:

1. Pro kombinaci  $(q, a, Z)$  může existovat maximálně jeden přechod
2. Pokud existuje přechod  $(q, \epsilon, Z)$ , tak nesmí existovat žádná kombinace  $(q, ?, Z)$

kde  $q \in Q$ ,  $a \in \Sigma$ ,  $Z \in \Gamma$  a  $?$  je jakýkoliv symbol vstupní abecedy. Pokud je kterékoliv z těchto pravidel porušeno, jedná se o automat nedeterministický.

Definice použitá v kapitole 2.1 obsahuje podmnožinu stavů označovanou písmenem  $F$  — množina přijímacích stavů. Pokud se po přečtení celého vstupu řídicí jednotka nachází v některém z přijímacích stavů, tak je vstup automatem přijat nezávisle na tom, jestli jsou nějaké symboly na

zásobníku. V opačném případě tento automat vstup nepřijímá. Někdy ale můžeme chtít, aby bylo slovo přijato pouze, pokud je po přečtení celého slova zásobník prázdný. V tom případě může být vhodnější zásobníkový automat (deterministický či nedeterministický) přijímající prázdným zásobníkem. Takový automat je definovaný jako šestice, neobsahuje množinu  $F$ , a po přečtení slova jej přijme, pouze pokud na zásobníku není žádný symbol, nezávisle na stavu řídicí jednotky.

Zásobníkové automaty se tedy dělí podle:

- podmínky pro přechodové funkce na:
  - deterministické
  - nedeterministické
- způsobu přijímání vstupu na:
  - přijímající přijímacím stavem
  - přijímající prázdným zásobníkem

## 2.3 Činnost zásobníkových automatů

Poslední část této kapitoly se věnuje tomu, jak zásobníkový automat funguje a jak probíhá jeho činnost. Pro potřeby této kapitoly bude použit následující deterministický zásobníkový automat přijímající slovo prázdným zásobníkem a rozpoznávající jazyk  $a^n b^n, n \geq 1$ :

$M = (Q, \Sigma, \Gamma, \delta, q, X)$ , kde

$$\begin{aligned} Q &= \{q\} \\ \Sigma &= \{a, b\} \\ \Gamma &= \{X, A\} \\ \delta &= \{ \\ &\quad (q, a, X) = (q, A), \\ &\quad (q, a, A) = (q, AA), \\ &\quad (q, b, A) = (q, \epsilon) \\ &\} \end{aligned}$$

Jako vstup bude použito slovo “aaabbb”.

V průběhu výpočtu se zásobníkový automat nachází vždy v nějaké konfiguraci, což je trojice  $(Q \times \Sigma^* \times \Gamma^*)$ .  $Q$  je aktuální stav, ve kterém se nachází řídicí jednotka,  $\Sigma^*$  je nepřečtená část vstupu a  $\Gamma^*$  je aktuální stav zásobníku.

Než automat započne svou činnost, musí se nastavit výchozí konfigurace podle definice automatu a námi požadovaného vstupu, v tomto případě  $(q, aaabbb, X)$ .

Když automat začne výpočet, přečte první znak ze vstupu, tedy symbol  $a$ , ze zásobníku se odebere symbol  $X$  a řídicí jednotka je ve stavu  $q$ . Automat tedy hledá přechod pro trojici  $(q, a, X)$ .

Tomu odpovídá přechod  $\delta(q, a, X) = (q, A)$ , který se použije. Jelikož automat již je ve stavu  $q$ , stav zůstává stejný, čtecí hlava se na vstupu posune na další symbol a na zásobník se vloží znak  $A$ . Nově je automat v konfiguraci  $(q, aabbb, A)$ . Tento postup se opakuje, dokud se nepřečte celý vstup, viz tabulka 2.1. Po skončení výpočtu zůstal zásobník prázdný, je tedy slovo přijato.

Pokud bychom měli vstup např. “aaabb”, tedy bez třetího  $b$ , tak by výpočet vypadal obdobně, ale tabulka 2.1 by končila řádkem s konfigurací  $(q, \epsilon, A)$  a žádným přechodem. Měli bychom tedy přečtený celý vstup, ale na zásobníku by nám pořád zbýval jeden symbol. Vstup by tedy nebyl přijat.

Konfigurace zásobníkového automatu	Přechodová funkce
$(q, aaabbb, X)$	$\delta(q, a, X) = \{(q, A)\}$
$(q, aabbb, A)$	$\delta(q, a, A) = \{(q, AA)\}$
$(q, abbb, AA)$	$\delta(q, a, A) = \{(q, AA)\}$
$(q, bbb, AAA)$	$\delta(q, b, A) = \{(q, \epsilon)\}$
$(q, bb, AA)$	$\delta(q, b, A) = \{(q, \epsilon)\}$
$(q, b, A)$	$\delta(q, b, A) = \{(q, \epsilon)\}$
$(q, \epsilon, \epsilon)$	

Tabulka 2.1: Ukázka činnosti zásobníkového automatu

## Kapitola 3

# Analýzy a návrh

V minulé kapitole byly popsány zásobníkové automaty a způsob jejich činnosti. Tato kapitola už se bude věnovat samotné aplikaci, konkrétně tomu, co vše se od aplikace očekává a co by měla umět.

Cílem této práce je tedy vytvořit aplikaci, která uživateli umožní si graficky simulovat činnost jakéhokoli zásobníkového automatu, deterministického i nedeterministického, přijímajícího prázdným zásobníkem nebo přijímacím stavem. Z důvodu lepší dostupnosti pro uživatele jsem se rozhodl zvolit webovou aplikaci, která bude dostupná všem uživatelům bez nutnosti stahování nebo instalace jakéhokoli softwaru.

Aplikace by měla uživateli poskytnout možnost nadefinovat si automat přímo v aplikaci, k čemuž by měl sloužit formulář, nebo moct nahrát automat ze souboru. Oba způsoby zadávání automatu by měly provádět kontrolu, jestli automat neobsahuje nějakou chybu, např. přechod obsahuje zásobníkový symbol, který není součástí zásobníkové abecedy. Dále si aplikace bude ukládat všechny zásobníkové automaty, aby se k nim mohl uživatel kdykoliv vrátit. Uživatel si bude moct zobrazit seznam všech uložených zásobníkových automatů, zobrazit si jejich definici, editovat je nebo je smazat. Dále si bude moct automat stáhnout do souboru, aby ho mohl např. sdílet s ostatními uživateli.

Kterýkoliv z těch automatů si bude moct uživatel zobrazit v simulátoru. Simulátor bude zobrazovat vstupní pásku, zásobník a řídicí jednotku, které budou vždy v aktuální konfiguraci, a bude uživateli umožňovat pro jím zadaný vstup krokovat činnost automatu s vyhodnocením, zda je slovo přijato nebo ne. Krokovat bude moct uživatel dopředu i dozadu, ručně nebo automaticky s časovým intervalem, jehož délka bude nastavitelná.



## Kapitola 4

# Implementace aplikace

Na předchozích stránkách této práce byly popsány zásobníkové automaty a požadavky aplikace. Následující stránky se budou zabývat již samotnou implementací aplikace. Nejprve se zaměřím na použité technologie a souborovou strukturu aplikace. Následovat pak bude již samotná implementace. Nejprve popíšu to, jak řeším reprezentaci zásobníkových automatů v kódu. Poté bude podkapitola zaměřující se na samotný simulátor a následovat budou podkapitoly týkající se menu, tvorby automatů pomocí formuláře, nahrávání souborů a jejich ukládání do paměti.

### 4.1 Technologie

Jelikož se jedná o webovou aplikaci, využíval jsem při vývoji webové technologie. Pro rozložení a strukturu stránky jsem použil značkovací jazyk HTML. Pro stylování používám CSS framework Tailwind<sup>1</sup>, který na rozdíl od jiných frameworků, jako třeba Bootstrap, neobsahuje třídy pro stylování celých komponent, ale spíše třídy pro jednotlivé vlastnosti, např. barva pozadí, barva textu, margin a padding jednotlivých stran velikostí, atd. Funkcionality aplikace píšu v jazyce Typescript<sup>2</sup>, což je nástavbu jazyka Javascript, která přidává statické typování, rozhraní a další věci. Ve výsledku je veškerý typescriptový kód překládán do Javascriptu pomocí nástroje Webpack<sup>3</sup>, který dokáže sbalit jednotlivé moduly a udělat z nich balíčky vhodnější pro prohlížeč. Ze všech mých typescriptových souborů je vytvořen jeden javascriptový soubor, který obsahuje veškerou funkcionalitu aplikace. Výsledkem aplikace jsou tedy tři soubory — index.html, output.css a app-bundle.js.

### 4.2 Reprezentace zásobníkových automatů v kódu

Abych mohl se zásobníkovými automaty pracovat v aplikaci, musel jsem mít způsob, jak je reprezentovat v kódu. Vytvořil jsem si tedy třídu PushdownAutomata, viz kód 4.1. Tato třída obsahuje

---

<sup>1</sup><https://tailwindcss.com/>

<sup>2</sup><https://www.typescriptlang.org/>

<sup>3</sup><https://webpack.js.org/>

jako atributy jednotlivé části definice zásobníkových automatů a metodu, kterou používám dále v simulátoru.

---

```
class PushdownAutomata{
    states: State[];
    inputSymbols: InputSymbol[];
    stackSymbols: StackSymbol[];
    initialState: State;
    initialStackSymbol: StackSymbol;
    acceptingState: State[] | null;
    transitionFunction: TransitionFunction[];

    getTransitionFunctions(tapeSymbol: string, state: State, stackSymbol:
        StackSymbol | null): TransitionFunction[];
}
```

---

Zdrojový kód 4.1: Deklarace třídy PushdownAutomata

První tři atributy definují jednotlivé množiny symbolů a stavů, se kterými automat pracuje. Jsou pro ně vytvořeny nové datové typy, zdrojový kód 4.2. Všechny tyto typy obsahují atribut value, který obsahuje samotnou hodnotu. Typ InputSymbol navíc obsahuje ještě atribut isEpsilon, který je využíván u přechodových funkcí a umožňuje přechod bez přčtení symbolu ze vstupu.

---

```
type State = {
    value: string;
}
type StackSymbol = {
    value: string;
}
type InputSymbol = {
    isEpsilon: boolean;
    value?: string;
}
```

---

Zdrojový kód 4.2: Datové typ State, StackSymbol, InputSymbol

Dále následují dva atributy definující výchozí konfiguraci automatu — initialState a initialStackSymbol. Po nich následuj acceptingState, který může nabývat dvou různých hodnot. Pokud obsahuje hodnotu null, tak zásobníkový automat přijímá slovo prázdným zásobníkem. V opačném případě, kdy obsahuje pole stavů, je slovo přijímáno přijímacím stavem.

Posledním atributem je transitionFunction. Ten obsahuje pole všech přechodů, které jsou reprezentované opět svým typem TransitionFunction, zdrojový kód ???. Ten se skládá z 5 atributů

— počátečního stavu, symbolu na zásobníku, symbolu na vstupní pásce, nového stavu a množiny zásobníkových symbolů, které budou přidány na zásobník, v tomto pořadí.

---

```
type TransitionFunction = {  
    fromState: State;  
    startSymbol: StackSymbol;  
    inputSymbol: InputSymbol;  
    toState: State;  
    pushedSymbols: StackSymbol[];  
}
```

---

Zdrojový kód 4.3: Datové typ TransitionFunction

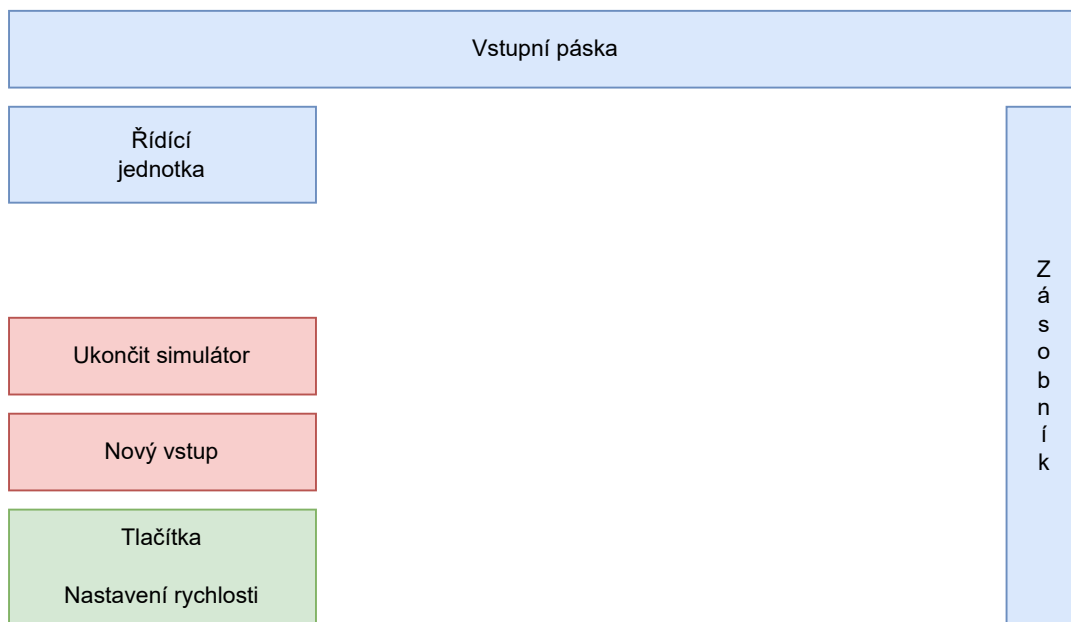
Jedinou důležitou metodou třídy PushdownAutomata je getTransitionFunctions, která pro trojici tapeSymbol, state a stackSymbol vrátí všechny přechodové funkce, které jsou pro tuto trojici definovány. Pokud existují funkce, které odpovídají i možnosti s epsilon přechodem, vrátí se taky.

## 4.3 Simulátor

Před tím, než jsem začal dělat část simulátoru, bylo nutné si uvědomit, co vše bude stránka obsahovat. Jako první jsem si navrhl rozložení vstupní pásky, zásobníku a řídicí jednotky. Ty jsou na obrázku 4.1 zobrazeny modře. Dále potřebuji tlačítka na ovládání simulátoru — pohyb dopředu a dozadu, zapnutí automatického pohybu, zastavení a nastavení rychlosti. Ty budou v oblasti v obrázku zakreslenou zeleně. Dále mi pak ještě chybí způsob, jak nastavit obsah vstupní pásky a ukončení simulátoru. K tomu slouží tlačítka v obrázku zakresleny červeně. Mezi ovládáním na levé straně a zásobníkem na pravé straně mi zůstalo spousta prázdného místa. To později využiji na zobrazení definice aktuálního automatu nebo zobrazení historie již použitých přechodů.

Když se přesunu do kódu, jako první jsem potřeboval způsob, jak reprezentovat stav zásobníkového automatu. K tomu mi slouží třída PushdownAutomataSimulator. Ta obsahuje automat, na kterém probíhá simulace, vstupní pásku, zásobník, aktuální stav, přijímací stavy a historii použitých přechodových funkcí, viz obrázek 4.2. Metoda reset slouží k zresetování simulátoru do výchozího stavu a applyTransitionFunction přijme jako parametr přechod a upraví podle něj stav simulátoru. Následující tři metody neupravují nijak stav simulátoru, ale pouze vrací informace pomocí návratových hodnot. Metoda acceptedInput vrací hodnotu true/false podle toho, zda byl vstup přijat. Pokud není vstup celý přečtený, vrátí false. Pokud je přečtený, tak záleží na typu automatu, buď vrátí hodnotu podle toho, zda je zásobník prázdný nebo ne, nebo podle toho, zda je aktuální stav v množině přijímacích stavů. Poslední dvě metody, nextStep a backStep, vrací přechodové funkce, které mohou být použity pro posun dopředu, respektive dozadu.

Nejrozsáhlejší třídou simulátoru je pak třída SimulatorUI. Na obrázku 4.2 jsou jen některá atributy a metody této třídy. Kromě nich dále obsahuje spoustu atributů, které si ukládají odkazy



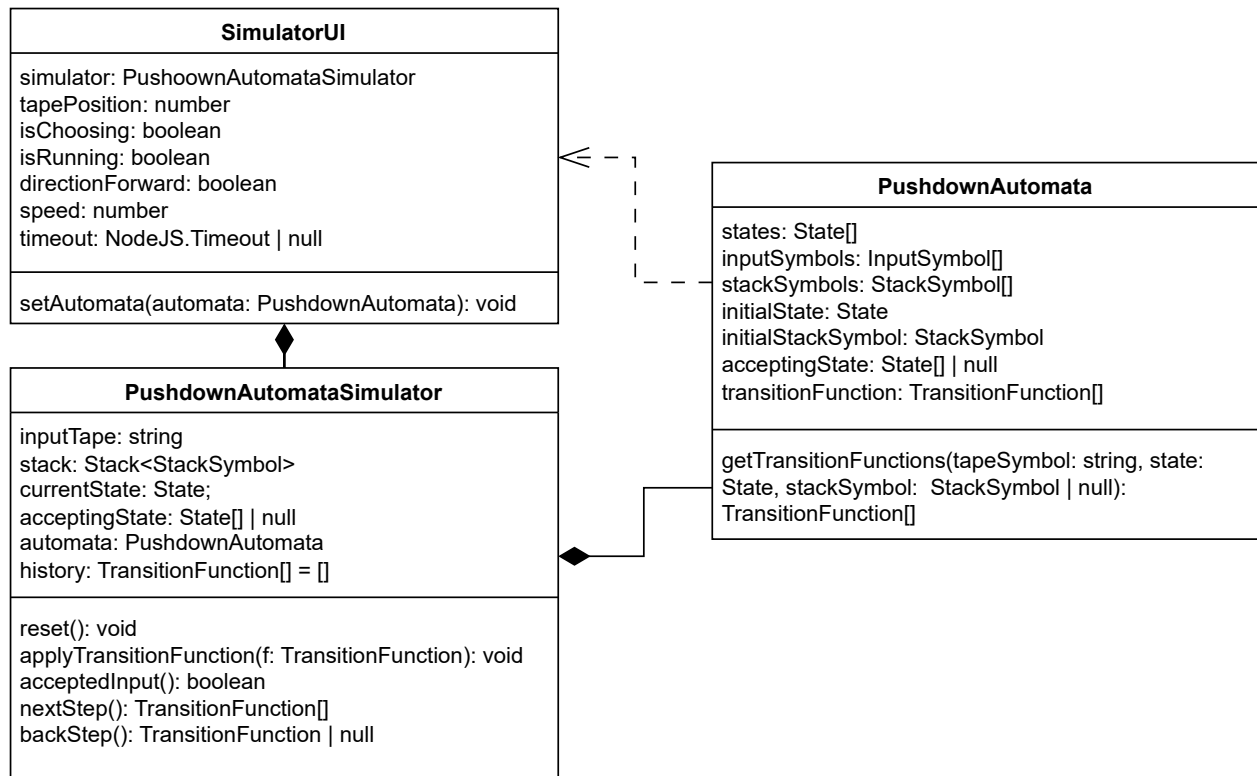
Obrázek 4.1: Návrh simulátoru

na jednotlivé části UI, a metody, pomocí kterých jde s UI manipulovat. Díky tomuto může tato třída obstarávat vše, co uživatel vidí a udělá.

Když se uživatel přepne na stránku simulátoru, jako první se zavolá metoda `setAutomata`. Ta nastaví simulátor s uživatelem vybraným zásobníkovým automatem a zresetuje celé UI, což obnáší vyčištění vstupní pásky, zásobníku a řídicí jednotky, historie použitých přechodů a nastavení výchozích hodnot z automatu. Dále se nastaví výchozí hodnoty proměnných potřebných pro automatickou simulaci — `isChoosing`, `isRunning`, `directionForward`, `speed` a `timeout`. Nakonec se otevře vyskakovací okno pro zadání slova na vstupní pásku. Toto okno obsahuje jednoduchý formulář s pouze jediným vstupem, nad kterým při každé změně proběhne kontrola, zda obsahuje pouze symboly vstupní abecedy. Když uživatel vstup potvrdí, znovu se zkontroluje, zresetuje se UI a vstup se nastaví do vstupní pásky.

K ovládání uživateli slouží 5 tlačítek a posuvník, obrázek 4.3. Krajiní tlačítka slouží k zapnutí automatické simulaci. Středové tlačítko slouží k pozastavení automatické simulace a posuvník níže slouží k nastavení času mezi jednotlivými kroky (svou hodnotu ukládá do proměnné `speed`). Zbylé dvě tlačítka slouží k manuálnímu krokování simulace.

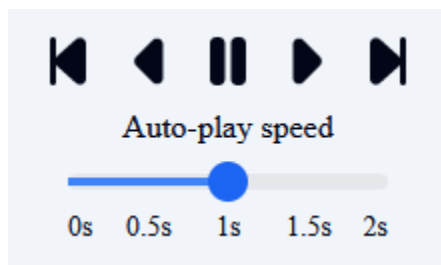
Pokud uživatel zmáčkne tlačítko pro krok dopředu, jako první se zkontroluje, zda aktuálně nevybírá přechodovou funkci. K tomu slouží atribut `isChoosing`. Pokud je aktuálně v tomto výběru a chce udělat krok vpřed, je na to upozorněn probliknutím oblasti s výběrem přechodu. Pokud v tomto výběru nebyl, pomocí metody `nextStep` třídy `PushdownAutomataSimulator` se zjistí všechny přechody, které je možné pro další krok použít. Podle počtu navrácených přechodů mohou nastat tři situace:



Obrázek 4.2: Třídní diagram tříd simulátoru

- Pokud metoda nevrátila žádný přechod, pozastaví se automatická simulace, pokud byla zapnuta, a vyhodnotí se, zda byl vstup přijat.
- Pokud metoda vrátila právě jeden přechod, je tento přechod použit. Pokud byla zapnuta automatická simulace, což se zjistí podle proměnné isRunning, nastaví se automatické zapnutí dalšího kroku podle aktuální hodnoty atributu speed a uloží se do atributu timeout.
- Pokud metoda vrátila více přechodů, nastaví se atributu isChoosing na true a vygenerují se tlačítka se všemi možnostmi.

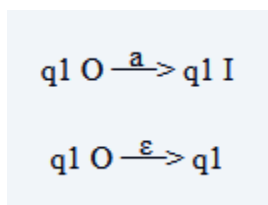
Když je použit přechod, musí se provést postupně několik věcí. Nejprve se změní vnitřní stav simulátoru metodou applyTransitionFunction. Následně se změní stav řídicí jednotky. Pokud byl přečten symbol ze vstupní pásky (nebyl to epsilon přechod), spustí se funkce moveTape. Ta inkrementuje hodnotu atributu tapePosition a změní styly přilehlý symbolů — přečtený dostane světlejší barvu a následující symbol dostane barvu tmavší. To umožní uživateli jednodušeji poznat, kterým symbol bude čtený v dalším kroku. Následně se odebere vrchní symbol ze zásobníku a přidají se symboly nové, pokud je přechodová funkce obsahuje. Poté se uloží nový záznam do historie. Nakonec se ještě zkontroluje, jestli již nebylo slovo zásobníkovým automatem přijato.



Obrázek 4.3: Ovládací tlačítka simulátoru

Pokud bylo možné použít více než jeden přechod, generují se tlačítka pro jednotlivé přechody, obrázek 4.4. Pro každé tlačítko je přidán event, který se spustí po kliknutí. Použije se konkrétní přechod a pokud byla zapnuta automatická simulace, nastaví se automatické zapnutí dalšího kroku podle aktuální hodnoty atributu speed a uloží se do atributu timeout.

Pokud uživatel zmáčkne tlačítko pro krok dozadu, jako první se zkontroluj, zda uživatel zrovna nevybírá přechodovou funkci. Pokud ano, výběr se schová. Pokud ne, získá se z historie poslední použitý přechod a náležitě se upraví stav simulátoru. Jestliže je zapnutá automatická simulace, nastaví se automatické zapnutí předchozího kroku podle aktuální hodnoty atributu speed a uloží se do atributu timeout. Ve chvíli, kdy je historie prázdná, nachází se simulátor ve výchozím stavu a pokud je zapnutá automatická simulace, vypne se.



Obrázek 4.4: Volba přechodových funkcí

Na začátku kapitoly jsem zmínil, že mi mezi ovládacími prvky a zásobníkem zůstalo prázdné místo, obrázek 4.1. Toto místo jsem využil pro zobrazování dvou informací. První je tabulka zobrazující definici aktuálně používaného automatu. Druhou je pak historie použitých přechodových funkcí, které se v průběhu simulace použily. V případě mobilního zobrazení stránky jsou tyto informace schovány v modálním okně, které lze otevřít tlačítkem.

## 4.4 Úložiště

V kapitole 3 bylo specifikováno, že si aplikace bude ukládat veškeré automaty, aby se k nim mohl uživatel kdykoliv vrátit. K tomu aplikace využívá Local Storage.<sup>4</sup> Local storage je úložiště v prohlížeči, které umožňuje ukládat data na straně klienta. Tyto data jsou ukládána ve formě key-value,

<sup>4</sup><https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

kdy pro každý klíč existuje jedna hodnota. Na rozdíl od session storage, kdy dochází k vymazání dat po opuštění stránky, zde data zůstávají i po opuštění stránky nebo zavření prohlížeče.

V mé aplikaci jsem si pro práci s tímto úložištěm udělal třídu Storage, zkrácený zápis lze vidět ve zdrojovém kódu 4.4. Jelikož Local Storage umožňuje ukládat klíče a hodnoty pouze jako textové řetězce, udělal jsem si nejprve metody save, která hodnotu převede na text a uloží ji do úložiště, a load, která pro zadaný klíč načte hodnotu z úložiště a převede ji z textu zpět na zadaný datový typ nebo objekt. Pro převody používám javascriptové funkce *JSON.stringify()* a *JSON.parse()*. Dále jsem si vytvořil metodu saveAutomata, která si nejprve ověří, zda už neexistuje v úložišti záznam se stejným klíčem pomocí metody keyExist. Pokud existuje, zeptá se uživatele, zda tento záznam může přepsat. Následně pomocí metody save uloží automat. Metoda loadAutomata si načte pro zadaný klíč automat z úložiště funkcí load, nastaví mu správný prototype a vrátí ho návratovou hodnotou.

---

```
class Storage{
    save<T>(key: string, item: T);
    load<T>(key: string): T | null;
    saveAutomata(key: string, automata: PushdownAutomata): boolean;
    loadAutomata(key: string): PushdownAutomata | null;
    delete(key: string);
    keyExists(key: string): boolean;
    loadFile(e: SubmitEvent);
    insertRow(key: string);
    printAutomatas();
    showAutomata(key: string);
}
```

---

#### Zdrojový kód 4.4: Třída Storage

Metoda printAutomatas slouží k výpisu všech automatů uložených v paměti. Metoda iteruje skrze všechny klíče v úložišti a volá pro ně metodu insertRow. Ta si pro zadaný klíč načte automat z úložiště a uloží nový řádek do tabulky i se všemi příslušnými tlačítky a nastavenými událostmi:

- Zobrazení specifikace automatu — metoda showAutomata
- Editace automatu
- Spuštění simulátoru
- Stažení automatu jako soubor typu JSON (JavaScript Object Notation)
- Odstranění automatu z úložiště — metoda delete

Poslední důležitou metodou je metoda `loadFile` sloužící k nahrání automatu ze souboru. Tato metoda je nastavená jako submit event, spustí se tedy pouze při odeslání formuláře. Formulář obsahuje pouze dvě pole. První je textové a slouží pro pojmenování automatu. Toto jméno se zobrazuje ve výpisu všech automatů a zároveň je použito jako klíč pro ukládání. Druhé pole pak slouží pro nahrání souboru. Po odeslání se formuláře se spustí metoda `loadFile`, která nejprve zkontroluje, že jsou obě pole vyplněné. Následně si pomocí metody `keyExists` zjistí, jestli klíč již není náhodou použit a případně se uživatele zeptá, zda chce automat pro ten klíč přepsat. Poté se ze souboru pokusí vytvořit objekt typu `PushdownAutomata` a provede se kontrola, zda je automat správně nadefinován, pomocí funkce `checkPushdownAutomata`. Pokud se nevyskytla žádná chyba, uloží automat do úložiště a přepne uživatele do simulátoru s nastaveným aktuálně nahraným zásobníkovým automatem.

## 4.5 Stránka pro tvorbu zásobníkových automatů

Třída `formAutomataBuilder` slouží k obsluze stránky, která slouží k tvorbě zásobníkového automatu. Obsahuje funkce, které se starají o zpracování dat při odeslání formulářů, kontroly dat, zobrazování chybových hlášek a další. Stránka se skládá z několika částí, kdy každá část odpovídá jedné části zásobníkového automatu.

První částí je formulář pro přidávání stavů. Po jeho odeslání se přidá nový stav do množiny stavů a přidá se jako jedna z možností, kterou lze vybrat jako přijímací stav a jako počáteční stav. Pokud je stav odstraněn, musí se odstranit i jako možnost v obou výběrech. Další částí je formulář pro přidávání symbolů vstupní abecedy. Po odeslání se symbol uloží do množiny symbolů vstupní abecedy. Po ní následuje formulář pro přidání symbolů zásobníkové abecedy. Ten po odeslání kromě uložení symbolu ještě symbol přidá do seznamu možností počátečního zásobníkového symbolu. Všechny tyto tři formuláře zároveň přidávají tlačítka do prvku pro tvorbu přechodové funkce.

Následující dvě části jsou seznamy pro výběr počátečního stavu a počátečního zásobníkového symbolu. Oba tyto seznamy reagují na jakoukoliv změnu díky nastavené `change` události a vždy si uloží vybranou možnost. Předposlední část slouží k určení, jestli automat bude slovo přijímat prázdným zásobníkem nebo množinou přijímacích stavů. To uživatel může určit pomocí zaškrťacího pole. Pokud pole není zaškrtnuté, zobrazí se uživateli seznam stavů a uživatel si může vybrat, které stavy budou přijímací.

Poslední část stránky slouží k definování přechodů přechodové funkce. Každý přechod se skládá z 5 částí, které můžeme vidět na obrázku 4.5 v prvním řádku. První 4 části jsou povinné, zatímco poslední část může zůstat prázdná dle definice přechodové funkce. Při kliknutí na kteroukoliv část se na druhém řádku zobrazí všechny možnosti, které mohou být použity pro danou část. Zobrazují se zde vždy jen symboly, které byly přidány dříve, ale pro vstupní symbol je zde ještě přidán symbol  $\epsilon$ . Při kliknutí tlačítka `Add transition` se zkontroluje, že jsou první 4 části vyplněné, že přechod



obsahuje pouze symboly nadefinovaných abeced a zda tento přechod již neexistuje. Pokud je vše v pořádku, tak přidá přechod do množiny přechodů přechodové funkce.

**Transition function:**

Q	A	—	ε	→	Q	A	Add transition
ε	a	b					

Obrázek 4.5: Vyplněný přechod na stránce tvorby zásobníkového automatu

Při kliknutí na tlačítko Save automata ve spodní části stránky se spustí funkce saveEventHandler. Ta jako první zkontroluje, že všechny abecedy mají minimálně jeden symbol a že uživatel vybral počáteční stav a počáteční zásobníkový symbol. Dále zkontroluje, zda se jedná o automat přijímající prázdným zásobníkem nebo přijímajícími stavy a zda je případně vybrán alespoň jeden. Poté ještě zkontroluje, zda je nadefinován alespoň jeden přechod přechodové funkce. Následně se provede kontrola celého zásobníkového automatu pomocí funkce checkPushdownAutomata, která je podrobněji popsána v sekci 4.6. Pokud se nikde nevyskytla chyba, automat se uloží do úložiště a stránka se přepne do simulátoru.

## 4.6 Funkce checkPushdownAutomata pro kontrolu automatu

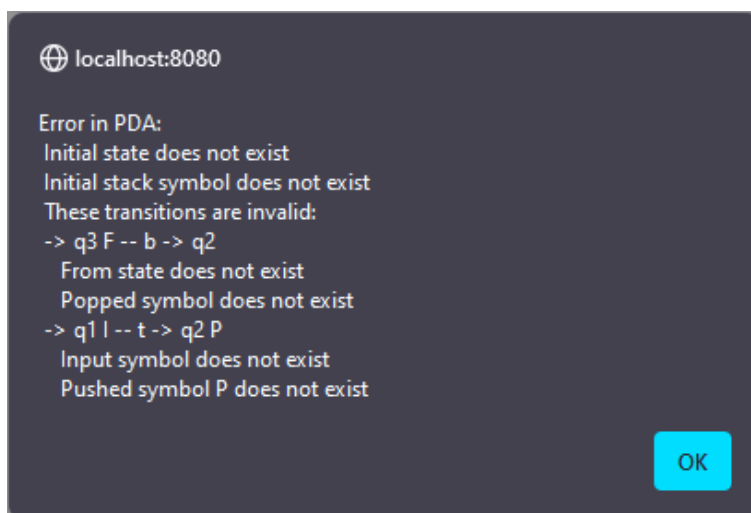
Poslední důležitou částí implementace je funkce pro kontrolu definice zásobníkových automatů. Tato funkce se volá při nahrání zásobníkového automatu ze souboru nebo při jeho definici přímo na stránce. Funkce postupně prochází jednotlivé části automatu a kontroluje jejich správnost. V případě nalezené chyby si uloží chybovou hlášku a na konci je všechny vypíše uživateli.

Jako první postupně zkontroluje množinu stavů a vstupní a zásobníkovou abecedu. U všech tří kontroluje, jestli množiny nejsou prázdné a zda neobsahují duplicity. Jelikož typescript neobsahuje datový typ char, ale pouze string, tak u abeced ještě zkontroluje, že všechny symboly jsou délky jednoho znaku.

Dále se kontroluje počáteční stav a počáteční zásobníkový symbol. U obou se zkontroluje, zda jsou součástí konkrétních množin. Pokud automat přijímá množinou přijímacích stavů, tak se pro každý přijímací stav taktéž zkontroluje, zda je součástí množiny stavů.

Nakonec se kontrolují přechody přechodové funkce. Pro každý přechod se zkontroluje, zda všechny jeho části (oba stavy, zásobníkové symboly a symbol vstupní abecedy, pokud není ε) jsou součástí konkrétních množin.

Nakonec se zkontroluje, jestli pole chybových hlášek je prázdné. Pokud není, tak se pomocí funkce alert zobrazí všechny chybové hlášky, viz obrázek 4.6, a funkce vrátí hodnotu false. V opačném případě vrátí funkce hodnotu true.



Obrázek 4.6: Ukázka chybových hlášek kontroly zásobníkového automatu

## Kapitola 5

# Uživatelské rozhraní

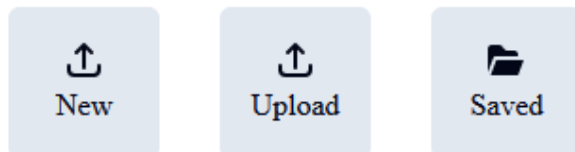
Předchozí kapitola 4 se zabývala převážně funkcí aplikace. V této kapitole se chci zaměřit na to, jak vypadá uživatelské rozhraní. Kapitola bude rozdělena do několika podkapitol, kdy každá z nich se bude věnovat jedné stránce a tomu, co stránka obsahuje a jak se případně ovládá.

### 5.1 Hlavní stránka

Když si uživatel zobrazí webovou stránku s aplikací, tak první, co uvidí, je hlavní nabídka. Ta je velice jednoduchá, protože obsahuje pouze nadpis a tři tlačítka vycentrované na středu stránky, obrázek 5.1. Každé z těchto tlačítek přepne uživatele na stránky, které budou popsány v dalších podkapitolách:

- Tlačítko new — Stránka pro tvorbu zásobníkového automatu, podkapitola 5.2
- Tlačítko upload — Formulář pro nahrávání zásobníkových automatů, podkapitola 5.3
- Tlačítko saved — Výpis úložiště, podkapitola 5.5

### Simulation of pushdown automata

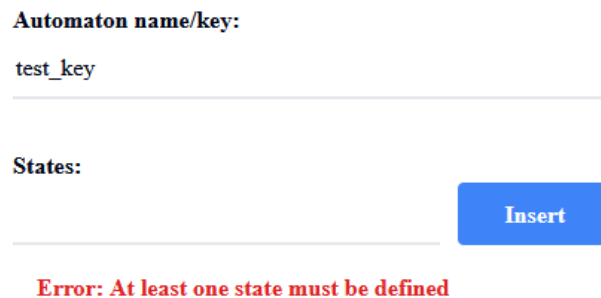


Obrázek 5.1: Hlavní stránka

## 5.2 Stránka pro tvorbu zásobníkového automatu

Po kliknutí na tlačítko new v hlavní nabídce se otevře stránka pro tvorbu zásobníkového automatu. Tato stránka se skládá z několika formulářů a HTML input prvků.

Jako první je textové pole pro specifikaci názvu automatu. Tento název se pak zobrazuje ve výpisu automatů z úložiště. Dále následuje formulář pro přidávání stavů. Stavby se přidávají po jednom a jejich délka není omezená. Dále následují formuláře pro definování vstupní a zásobníkové abecedy. Symboly se opět přidávají po jednom a jejich délka je omezena na jeden znak. Vstupní pole pro název a formulář pro přidávání stavů lze vidět na obrázku 5.2 včetně chybové hlášky.



**Automaton name/key:**  
test\_key

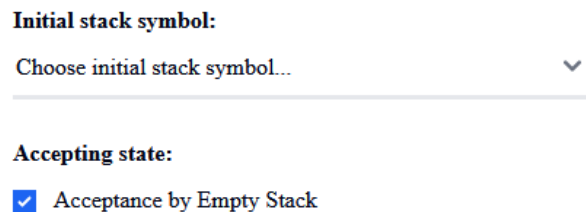
**States:**

**Insert**

**Error: At least one state must be defined**

Obrázek 5.2: Název a formulář pro přidávání stavů

Po těchto třech formulářích následují dva seznamy pro výběr počátečního stavu a počátečního zásobníkového symbolu. Dále následuje část pro určení, zda zásobníkový automat bude přijímat prázdným zásobníkem nebo přijímací stav. To závisí na tom, zda uživatel zaškrtně zaškrťovací pole. Pokud zůstane nezaškrtnuté, musí uživatel vybrat alespoň jeden ze stavů níže jako přijímací. Seznam počátečních zásobníkových symbolů a zaškrťovací pole je na obrázku 5.3.



**Initial stack symbol:**  
Choose initial stack symbol...

**Accepting state:**  
☒ Acceptance by Empty Stack

Obrázek 5.3: Seznam počátečních zásobníkových symbolů a zaškrťovací pole pro určení typu zásobníkového automatu

Poslední část pak slouží k definování přechodů přechodové funkce. Její funkce již byla popsána v podkapitole 4.5. Jak tato část vypadá lze vidět na obrázku 4.5. Pak již následují jen tlačítka na navrácení na hlavní stránku a uložení zásobníkového automatu.

## 5.3 Formulář pro nahrávání zásobníkových automatů

Druhým tlačítkem v hlavní nabídce se uživatel dostane na stránku s formulářem, který slouží k nahrání zásobníkového automatu ze souboru. Formulář, obrázek 5.4, je složen ze dvou částí. První je textové pole pro pojmenování automatu. Toto pole se používá jako klíč pro úložiště a zároveň se zobrazuje v výpisu automatů. Pokud je klíč již použitý pro jiný automat, je při odeslání formuláře uživatel dotázán, zda chce automat s tímto klíčem přepsat. Druhé pole pak slouží pro nahrání souboru typu JSON, který obsahuje definici zásobníkového automatu. Při odeslání formuláře se nejprve ze souboru vytvoří instance třídy PushdownAutomata a následně se provede kontrola. Pokud automat nemohl být vytvořen kvůli neodpovídající struktuře nebo obsahuje chyby, je na to uživatel upozorněn společně s výpisem chyb, obrázek 4.6.

**Load on from file**

Automaton name/key:

File:

Procházet...

Soubor nevybrán.

Cancel

Load

Obrázek 5.4: Formulář pro nahrání zásobníkového automatu ze souboru

## 5.4 Výpis úložiště

Třetím tlačítkem v hlavní nabídce si uživatel může zobrazit seznam všech automatů, které jsou uloženy v úložišti. Vzhled této stránky je na obrázku 5.5. Nad samotnou tabulkou je nadpis a tlačítko pro návrat do hlavní nabídky. Samotná tabulka pak má 6 sloupců a každý řádek odpovídá jednomu automatu v úložišti.

První sloupec obsahuje název automatu, dalších 5 sloupců pak obsahuje tlačítka. První tlačítko zobrazí stránku, na které je tabulka s informacemi o automatu. Druhé tlačítko přepne uživatele na stránku pro tvorbu zásobníkového automatu, kde může uživatel automat zeditovat. Třetí tlačítko přepne uživatel na stránku simulátoru, která je popsána v podkapitole 5.5, a nastaví tam konkrétní zásobníkový automat. Čtvrté tlačítko pak slouží pro stažení automatu jako soubor typu JSON. Poslední tlačítko pak slouží pro vymazání automatu z úložiště.

Saved automata						← Go back
Name	Show	Edit	Run	Download	Delete	
test						
PDA without a name						
PDA2						
PDA1						

Obrázek 5.5: Výpis automatů v úložišti

## 5.5 Simulátor

Poslední stránkou této aplikace je stránka samotného simulátoru. Na tuto stránku se může uživatel dostat třemi způsoby:

- Vytvořením automatu na stránce popsané v podkapitole 5.2
- Nahráním automatu ze souboru
- Vybráním automatu z úložiště

Při zobrazení této stránky se jako první otevře modální okno pro zadání vstupu, které je na obrázku 5.6. Po potvrzení vstupu se okno schová a uživatel vidí již samotný simulátor. Na obrázku 5.7 lze vidět stránku v průběhu simulace.

**Tape:**

Close

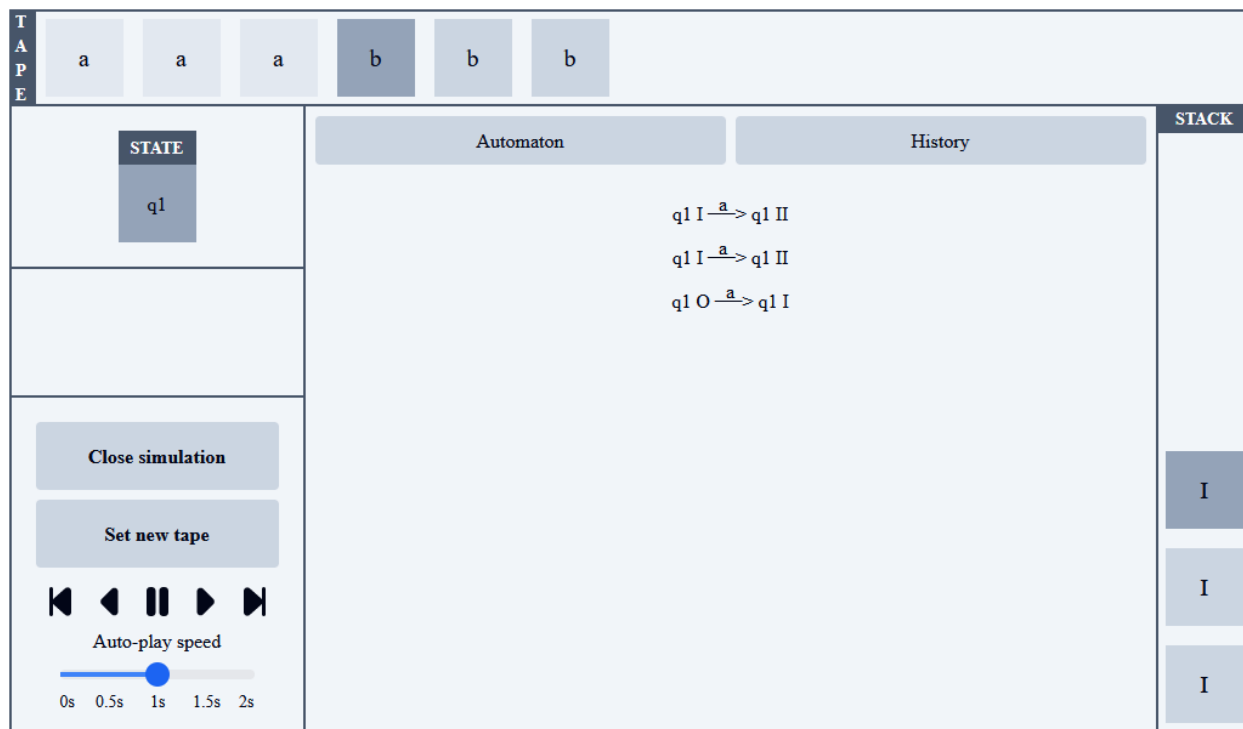
Set

Obrázek 5.6: Okno pro zadání vstupu

Samotný simulátor se pak skládá z několika částí. V horní části je řádek, který symbolizuje vstupní pásku se symboly. Symbol s tmavším pozadím je symbol, který bude přečten jako další. Nalevo od něj jsou již přečtené symboly a napravo symboly ještě nepřečtené. Na pravé straně se pak nachází zásobník. Symbol, který se nachází nejvýše a má tmavší pozadí, je symbol na vrcholu zásobníku, takže bude přečten v dalším kroku.

Na levé straně stránky se nachází tři oblasti seřazené ve sloupci. První oblast, která se nachází nejvýše, zobrazuje aktuální stav, ve kterém se nachází řídicí jednotka. Ve druhé oblasti, která je na obrázku 5.7 prázdná, se zobrazují možnosti volby dalšího přechodu u nedeterministických zásobníkových automatů. Příklad volby je na obrázku 4.4. Třetí část pak obsahuje ovládací prvky simulátoru. Tlačítko Close simulation ukončí simulaci a vrátí uživatele do hlavní nabídky. Set new tape otevře modální okno pro zadání vstupu, stejně jako při prvním otevření stránky. Šipky pak slouží k samotnému ovládání simulace. Šipky na krajích slouží ke spuštění automatické simulace, šipky uvnitř pak slouží k manuálnímu krokování. Tlačítko uprostřed slouží k pozastavení simulace. Posledním ovládacím prvkem je posuvník, který slouží k nastavení času mezi jednotlivými kroky při automatické simulaci.

Poslední částí simulátoru je místo uprostřed obrazovky, které slouží k zobrazování informací. V horní části jsou dvě tlačítka, které přepínají, co bude zrovna zobrazeno. Levé tlačítko slouží k zobrazení tabulky s definicí aktuálně používaného zásobníkového automatu, levé pak slouží k zobrazení historie přechodů již použitých v této simulaci.



Obrázek 5.7: Okno pro zadání vstupu

Aplikace podporuje i zobrazení na menších obrazovkách, jako mají například telefony nebo tablety. Toto zobrazení se oproti tomu na obrázku 5.7 liší ve dvou věcech — vstupní páska je svisle na levé straně obrazovky a středová část s informacemi je jako modální okno, které se otevírá tlačítkem.

# Vzorové vstupy a jejich struktura

Vstupní soubory jsou ve formátu JSON, což je javascriptový objektový zápis. Jelikož se z toho souboru v aplikaci vytváří objekt, musí soubor dodržovat přesně zápis odpovídající třídy PushdownAutomata, zdrojový kód 4.1, a typům v ní použitých, zdrojové kódy 4.2 a 4.3.

- `error_testing.json`

- `anbn.json`

- `brackets_and_parentheses.json`

- **palindrome.json**



- Automat přijímá binární čísla, která jsou palindromy
- Přijímané vstupy: 0, 1, 00, 11, 101, 1010101, ...
- Nepřijímané vstupy: 01, 10, 001, 01010101, ...

## Kapitola 7

# Existující aplikace

Existuje několik aplikací, které řeší problém simulace zásobníkových automatů. Tato kapitola se bude některým z nich věnovat.

### 7.1 YAAS — Yet Another Automata Simulator

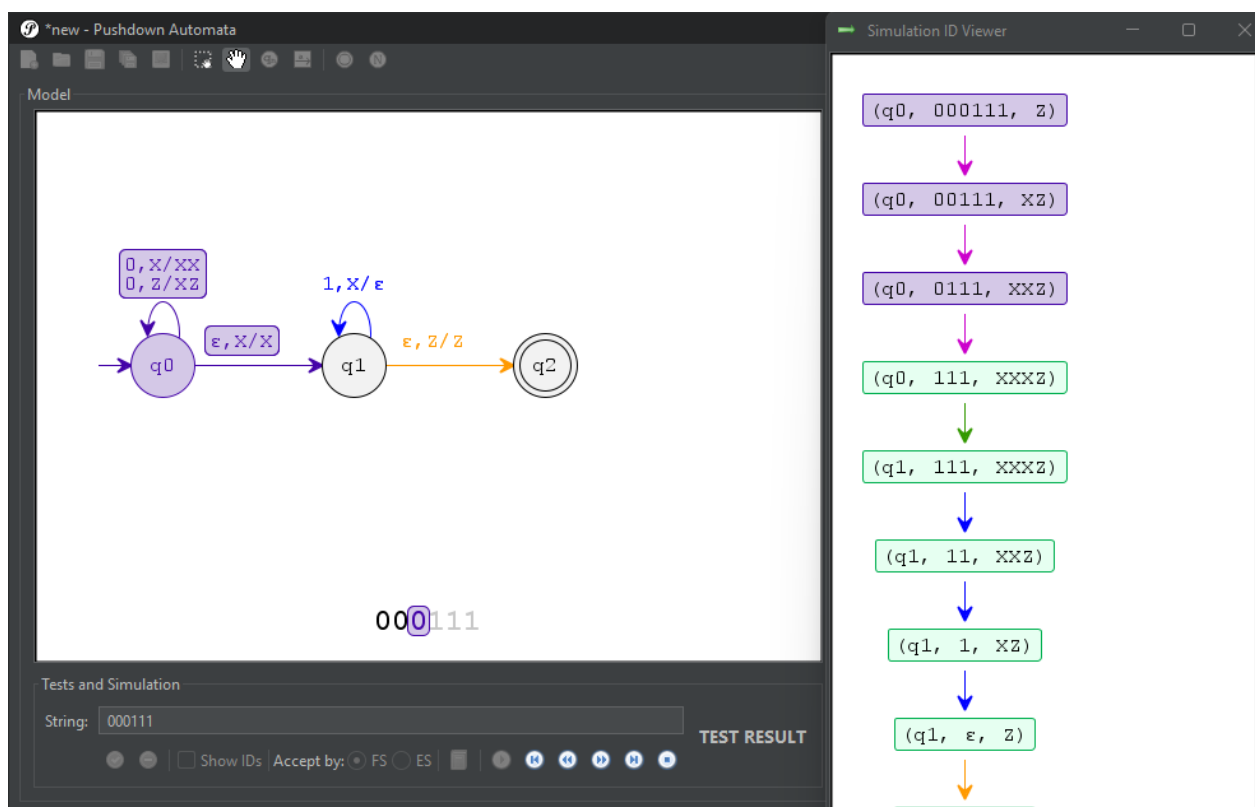
YAAS — Yet Another Automata Simulator, v češtině přeloženo jako Ještě Další Simulátor Automatů, je aplikace vytvořená profesorem Davidem Buzatem z Federálního institutu pro vzdělání, vědu a technologie v São Paulu. Tato aplikace obsahuje nejen možnost tvorby zásobníkových automatů, ale i konečných automatů a turingových strojů. Zásobníkové automaty se v aplikaci tvoří graficky — uživatel se přidá stavy a pak mezi nimi může vytvářet přechody. Aplikace si z těchto informací sama vytvoří příslušnou vstupní a zásobníkovou abecedu. Při spuštění simulace se pak uživateli zobrazí posloupnost přechodů a zároveň v grafickém zobrazení obarvuj konkrétní stav, ve kterém se zrovna zásobníkový automat nachází, obrázek 7.1

### 7.2 DauteRR — Pushdown Automaton

Pushdown automaton od Dauta Rodríguez Rodríguez je konzolová aplikace napsaná v jazyce Java. Tato aplikace na vstup bere soubor s definicí zásobníkového automatu. Vstupy jsou zadány buď v druhém vstupním souboru nebo jsou čteny z příkazového řádku. Program dále umožňuje zapnout tzv. trace mód, který ve výpisu ukazuje jednotlivé kroky a pomocí > označuje, který symbol je zrovna čtený.

### 7.3 Bakalářská práce

Ahhhhhhhhhhhhhhhhhhhhhh



Obrázek 7.1: YAAS v průběhu simulace.

## Kapitola 8

### Závěr

Cílem této práce bylo vytvořit aplikaci, která by uživateli umožňovala graficky simulovat činnost zásobníkových automatů. Jako první bylo nutné si ale nastudovat problematiku zásobníkových automatů, jak jsou definovány, jaké jsou jejich typy a jak fungují. Poté jsem vytvořil webovou aplikaci, dovoluje uživateli simulovat činnost libovolného zásobníkového automatu pro jím zadaný vstup. Zásobníkové automaty může uživatel nahrát jako soubor nebo je vytvořit přímo v aplikaci. Všechny zásobníkové automaty se ukládají do lokálního úložiště prohlížeče, aby k nim měl uživatel přístup a při příštím spuštění aplikace. Následně jsem vytvořil několik vzorových zásobníkových automatů, na kterých jde vidět činnost aplikace a pomocí kterých jsem aplikaci testoval.

Práci je možné v budoucnu rozšířit a další funkce, jako je např. možnost převodu mezi automaty přijímajícími prázdným zásobníkem a přijímajícími stavy, grafické zobrazení automatu nebo možnost vytvoření zásobníkového automatu z bezkontextové gramatiky.