VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Simulace zásobníkových automatů

Simulation of Pushdown Automata

Ondřej Just

Bakalářská práce

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

Ostrava, 2024

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/23/113231
vsboee23037AF8

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

Tématem a cílem této bakalářké práce bylo vytvořit aplikaci pro simulaci zásobníkových automatů. Výsledkem je webová aplikace umožňující simulovat výpočet zásobníkových automatů, tvorbu vlastních zásobníkových automatů v aplikaci nebo nahrání souboru se zásobníkovým automatem. Veškeré zásobníkové automaty se ukládají do úložiště prolížeče a součástí práce jsou i vzorové zásobníkové automaty. Aplikace je napsána v jazycích TypeScript a HTML a frameworku Tailwind.

Klíčová slova

bakalářská práce; simulace; zásobníkový automat; webová aplikace; TypeScript; HTML; Tailwind

Abstract

The topic and aim of this bachelor thesis was to create an application for the simulation of pushdown automata. The result is a web application that allows the simulation of the computation of pushdown automata, the creation of custom pushdown automata in the application or uploading the file with the pushdown automata. All pushdown automata are stored in the browser's storage and the work also includes sample pushdown automata. The application is written in TypeScript and HTML languages and the Tailwind framework.

Keywords

bachelor thesis; simulation; pushdown automaton; web application; TypeScript; HTML; Tailwind

Poděkování

Rád bych na tomto místě poděkoval doc. Ing. Zdeňku Sawovi, Ph.D. za veškerou pomoc a konzultace při tvorbě této bakalářksé práce.

Obsah

Seznam použitých zkratek a symbolů

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

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Kapitola 1

Úvod

Chomského hierarchie popisuje 4 druhy gramatik a jazyků — regulární, bezkontextové, kontextové a neomezené. Pokud pracujeme s regulárními jazyky, nejnižšími 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 ukládaní dat. Právě zásobníkovými automaty se táto práce zabývá, přesněji simulací jejich činností.

Cílem této práce bylo implementovat grafický simulátor zásobníkových automatů, deterministických i nedeterministických, přijímajících prázdným zásobníkem i přijímacími stavy. Aplikace by měla sloužit studentům nebo komukoliv, kdo se zajímá o zásobníkové automaty, a měla by jím umožnit jednodušeji pochopit, jak v nich probíhá výpočet.

Aplikace umožňuje:

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

1.1 Obsah práce

Práce je rozdělená do několika kapitol a jejich obsah je popsán dále. Kapitola 2 se zabývá 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. V kapitole 3 jsou zmíněny některé již existující aplikace řešící stejné téma. Kapitola 4 se pak věnuje tomu, co se od aplikace očekává a jaké jsou požadavky. Kapitola 5 obsahuje

popis technologií, které budou využity a samotnou implementaci funkcionalit. Následující kapitola 6 se pak zabývá uživatelským rozhraním a jeho ovládáním. Kapitola 7 obsahuje popis vstupních souborů obsahujících definice zásobníkových automatů a v poslední kapitole 8 je nakonec shrnutá celá práce, její výsledek a možná rozšíření.

Kapitola 2

Zásobníkové automaty

Tato kapitola se bude zabývá 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 řídící 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 později, 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 [1] je formálně definován jako uspořádaná sedmice:

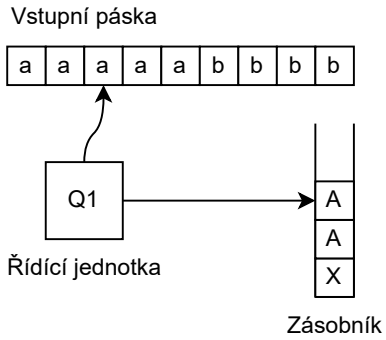
$M = (Q, \Sigma, \Gamma, \delta, q_0, X_0, F)$

kde Q, Σ, Γ a F jsou neprázdné konečné množiny a

- Q je množina stavů
- Σ je vstupní abeceda
- Γ je zásobníková abeceda
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ 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, řídící jednotky a zásobníku. Množina stavů Q obsahuje všechny stavy, ve kterých se může vyskytovat řídící jednotka při výpočtu. Σ obsahuje všechny symboly, které se mohou vyskytnout na vstupní pásce a Γ zase všechny symboly použitelné na zásobníku. q_0 je stav z množiny Q, ve kterém se nachází řídící jednotka na začátku výpočtu. X_0 je symbol z množiny Γ, který se nachází na zásobníku na začátku výpočtu. Množina F, která je podmnožinou Q, obsahuje všechny stavy, ve kterých je vstup přijat. Přechodová funkce $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$ zase říká, jak se automat zachová při určitém stavu, když přečte vstupní symbol a na vrcholu zásobníku je určitý symbol. Např. přechod $\delta(q_1, a, A) = \{(q_2, \varepsilon)\}$ říká, že pokud se ze vstupu přečte znak a, na vrcholu zásobníku je symbol A a řídící jednotka je ve stavu q_1, tak se řídící jednotka přesune do stavu q_2 a na zásobník se nic nepřidá. Přechod jde zapsat ještě druhým způsobem — $q_1 A \xrightarrow{a} q_2$



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 pro (q, ε, Z), tak nesmí existovat přechod pro (q, a, Z)

kde $q \in Q$, $a \in \Sigma$ a $Z \in \Gamma$. Pokud je kterékoliv z těchto pravidel porušeno, jedná se o automat nedeterministický. [2]

Definice použitá v kapitole 2.1 obsahuje podmnožinu stavů označovanou písmenem F — množina přijímacích stavů. Pokud se po přečtení celého vstupu řídící 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. Jedná se o automat přijímající přijímacími stavy. 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 řídící jednotky.

Zásobníkové automaty se tedy dělí podle:

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

2.3 Konfigurace a přechody

Zásobníkový automat se vždy nachází v nějaké konfiguraci, což je trojice (q, w, α), kde $q \in Q$, $w \in \Sigma^*$ a $\alpha \in \Gamma^*$. Na začátku výpočtu se automat nachází ve výchozí konfiguraci (q_0, w, X_0), kde w je vstup, který chceme zpracovat. [3] Automat může mezi jednotlivými konfiguracemi přecházet podle definovaných přechodů přechodové funkce. Použit je vždy přechod, který odpovídá aktuální konfiguraci, tedy aktuálnímu stavu, zásobníkovému symbolu na vrcholu zásobníku a prvnímu znaku nepřečtené části vstupu nebo ε. V případě nedeterministických automatů může existovat více přechodů, které můžou být použity. Existuje tedy více možností výpočtu, kdy některé mohou vést k přijetí vstupu a některé naopak ne.

2.4 Činnost zásobníkových automatů

Poslední část této kapitoly se věnuje tomu, jak probíhá činnost zásobníkového automatu. Pro potřeby této kapitoly bude použit následující deterministický zásobníkový automat přijímající slovo prázdným zásobníkem a rozpoznávající jazyk[1] $a^n b^n, n \geq 1$:
$M = (Q, \Sigma, \Gamma, \delta, q, X)$, kde
$Q = \{q\}$
$\Sigma = \{a, b\}$
$\Gamma = \{X, A\}$

[1] Jazyk je konečná množina řetězců/slov nad příslušnou abecedou

$\delta = \{$
$(q, a, X) = (q, A),$
$(q, a, A) = (q, AA),$
$(q, b, A) = (q, \varepsilon)$
$\}$

Jako vstup bude použito slovo "*aaabbb*".

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 řídící 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. [3] 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, ε, 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)	$\delta(q, b, A) = \{(q, \varepsilon)\}$
(q, bb, AA)	$\delta(q, b, A) = \{(q, \varepsilon)\}$
(q, b, A)	$\delta(q, b, A) = \{(q, \varepsilon)\}$
$(q, \varepsilon, \varepsilon)$	

Tabulka 2.1: Ukázka činnosti zásobníkového automatu

Kapitola 3

Existující aplikace

Existuje několik aplikací, které řeší problém simulace zásobníkových automatů. Některé z nich bych v této kapitole rád popsal. [4] [5] [6]

3.1 YAAS — Yet Another Automata Simulator

YAAS — Yet Another Automata Simulator je aplikace vytvořená profesorem Davidem Buzattem z Instituto Federal de Educação, Ciência e Tecnologia de São Paulo. Tato aplikace obsahuje nejen možnost tvorby zásobníkových automatů, ale i konečných automatů a turingových strojů. Zásobníkové automaty se v aplikaci tvoří graficky — uživatel si přidá stavy a pak mezi nimi může vytvářet přechody pomocí šipek. 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 obarvuje konkrétní stav, ve kterém se zrovna zásobníkový automat nachází, Obrázek 3.1.

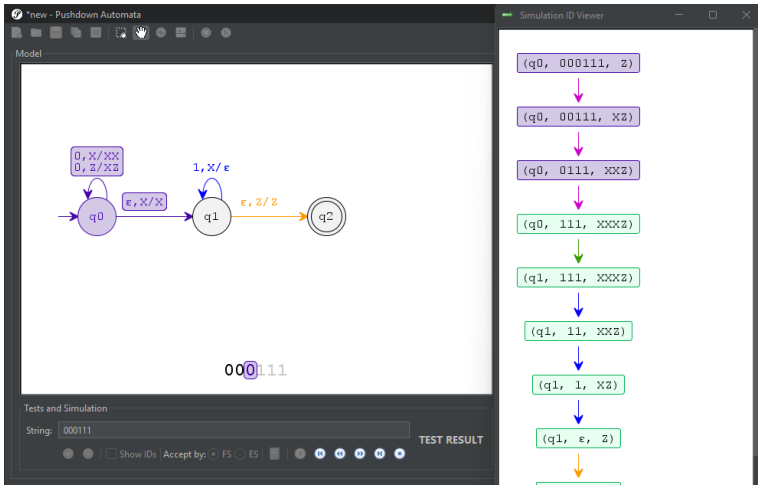
3.2 DauteRR — Pushdown Automaton

Pushdown automaton od Dauta Rodríguez Rodrígueze 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.

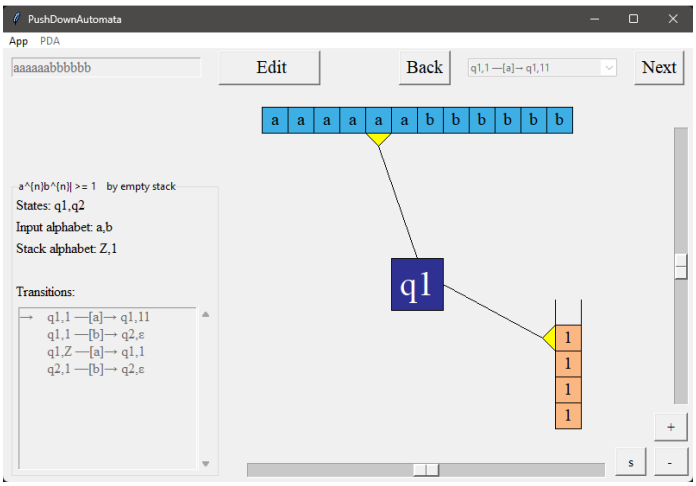
3.3 Bakalářská práce z roku 2022/2023

Ve školním roce 2022/2023 byla vytvořena bakalářská práce na stejné téma studentem Danielem Bednarzem. Jedná se o desktopovou aplikaci napsanou v jazyce Python. Aplikace umožňuje vytvářet

zásobníkové automaty pomocí formuláře přímo v aplikaci nebo automat nahrát ze souboru. Simulace probíhá graficky, kdy uživatel vidí na rozdíl od aplikace YAAS vstupní pásku, řídící jednotku a zásobník, Obrázek 3.2.



Obrázek 3.1: YAAS v průběhu simulace.



Obrázek 3.2: Bakalářská práce z roku 2022/2023

3.4 Shrnutí

Jak lze vidět výše, existují aplikace, které řeší simulaci zásobníkových automatů různým způsobem. Jeden ze způsobů je výpis do konzole. Další způsoby jsou pak již grafické, kde se objevují dva různé přístupy. Jeden je graf se stavy jako kolečka a přechody znázorněnými jako šipky mezi jednotlivými stavy. Druhý pak zobrazuje komponenty podobně, jako jsou zakresleny na Obrázku 2.1.

Kapitola 4

Analýzy a návrh

V minulých kapitolách byly popsány zásobníkové automaty a způsob jejich činnosti a již existující aplikace. Tato kapitola už se bude věnovat samotné aplikaci, konkrétně její analýze a návrhu simulátoru.

4.1 Analýza

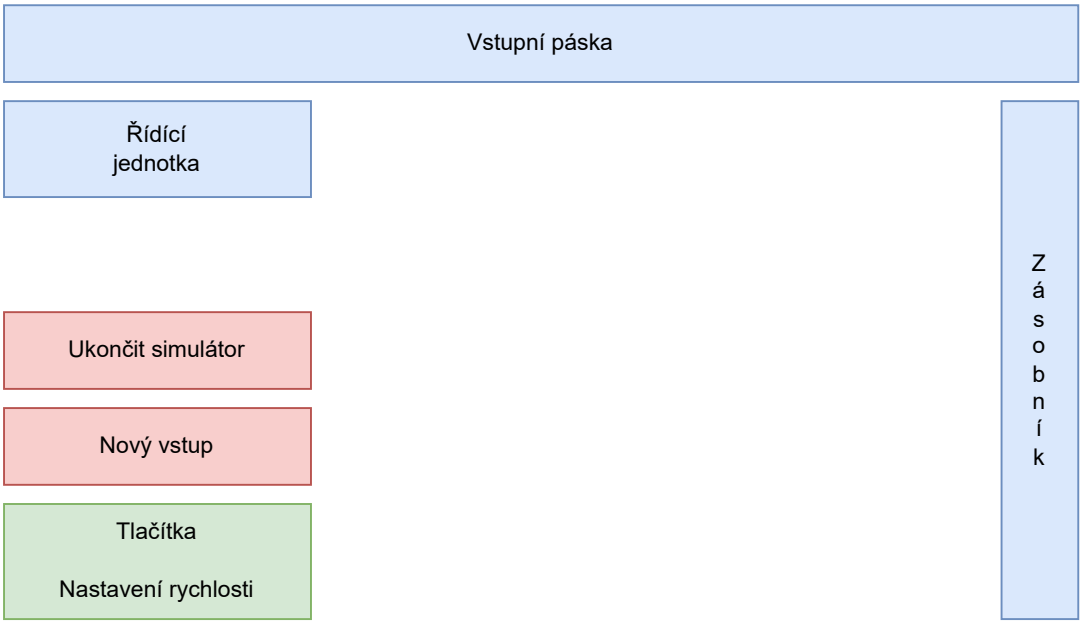
Cílem této práce je tedy vytvořit aplikaci, která uživateli umožní si graficky simulovat činnost jakéhokoliv zásobníkového automatu, deterministického i nedeterministického, přijímajícího prázdným zásobníkem nebo přijímacím stavem. Z důvodu lepší dostupnosti pro uživatele jsem se rozhodl zvolit webovou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

Aplikace by měla uživateli poskytnou možnost nadefinovat si automat přímo v aplikace, 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 řídící jednotku, které budou vždy v aktuální konfigurace, a bude uživateli umožňovat pro jím zadaný vstup krokovat činnost automat 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.

4.2 Návrh stránky simulátoru

Před samotnou implementací aplikace si bylo nutné uvědomit, co vše bude simulátor obsahovat za prvky a jaké bude jejich rozložení. Návrh rozložení je na Obrázku 4.1. Jako první jsem si navrhl rozložení vstupní pásky, zásobníku a řídící jednotky. — na obrázku zakresleny modře. Dále potřebuji tlačítka na ovládání simulátoru — pohyb dopředu a dozadu, zapnutí automatické simulace, zastavení a nastavení rychlosti. Ty jsou 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 bude později využito na zobrazení definice aktuálního automatu nebo zobrazení historie již použitých přechodů.



Obrázek 4.1: Návrh simulátoru

Kapitola 5

Implementace aplikace

Na předchozích stránkách teto práce byly popsány zásobníkové automaty, požadavky aplikace a návrh simulátoru. Následující stránky se budou zabývat již samotnou implementací aplikace. Začátek této kapitoly se zaměří 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.

5.1 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 [7], 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 strana velikostí, atd. Funkcionality aplikace píšu v jazyce Typescript [8], což je nástavba jazyka Javascript, která přidává statické typování, rozhraní a další věci. [9] Ve výsledku je veškerý typescriptový kód překládán do Javascriptu pomocí nástroje Webpack [10], který dokáže sbalit jednotlivé moduly a udělat z nich balíčky vhodnější pro prohlížeč. [11] 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.

5.2 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 5.1. Tato třída obsahuje

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 5.1: 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 5.2. Všechny tyto typy obsahují atribut `value`, který obsahuje samotnou hodnotu. Typ `InputSymbol` navíc obsahuje ještě atribut `isEpsilon`, který je využíván u přechodů a umožňuje přechod bez přečtení symbolu ze vstupu.

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

Zdrojový kód 5.2: Datové typy State, StackSymbol, InputSymbol

Dále následují dva atributy definující výchozí konfiguraci automatu — `initialState` a `initialStackSymbol`. Po nich následuje `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 5.3. 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 5.3: Datový typ TransitionFunction

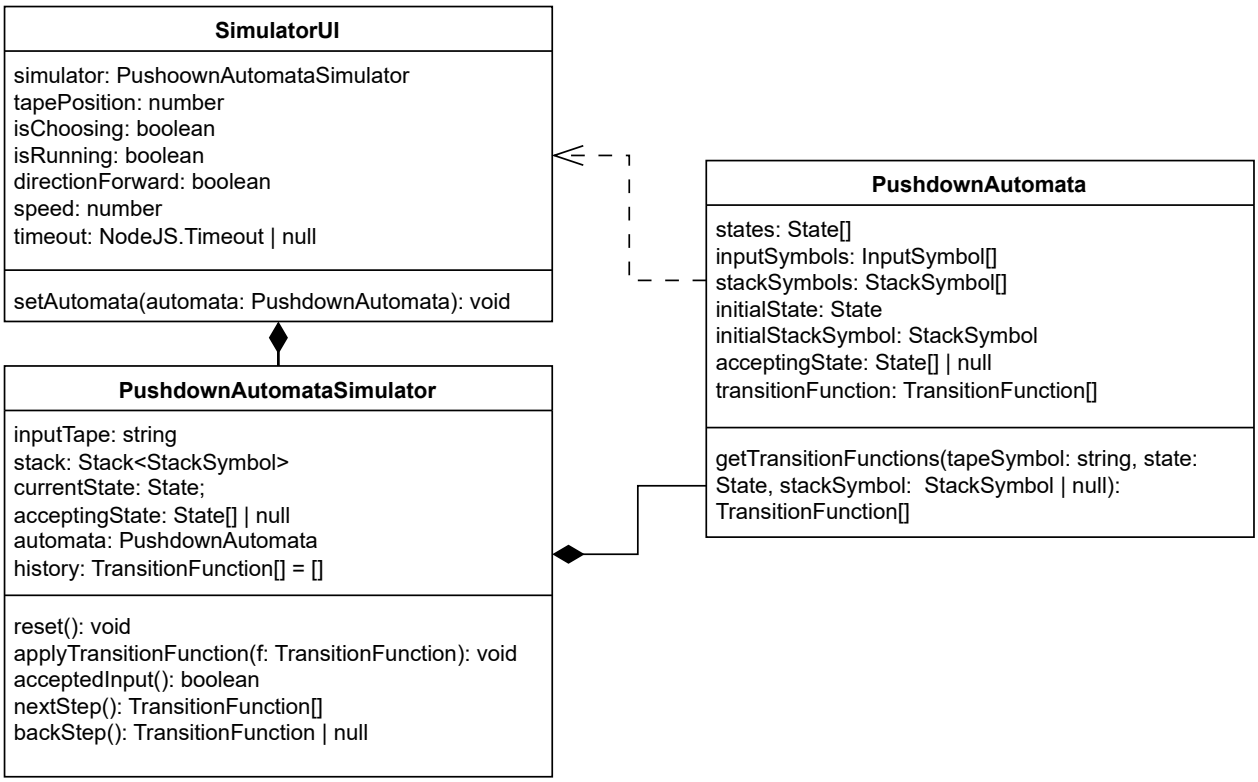
Jedinou důležitou metodou třídy `PushdownAutomata` je `getTransitionFunctions`, která pro trojici `tapeSymbol`, `state` a `stackSymbol` vrátí všechny přechody, které jsou pro tuto trojici definovány. Pokud existují přechody, které odpovídají i možnosti s epsilon přechodem, vrátí se taky.

5.3 Simulátor

Potřeboval jsem způsob, jak reprezentovat stav simulátoru. K tomu slouží třída `PushdownAutomataSimulator`. Ta obsahuje automat, na kterém probíhá simulace, vstupní pásku, zásobník, aktuální stav, přijímací stavy a historii použitých přechodů, viz Obrázek 5.1. 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 přechody, které mohou být použity pro posun dopředu, respektive dozadu.

Nejrozsáhlejší třídou simulátoru je pak třída `SimulatorUI`. Na Obrázku 5.1 jsou jen některé její atributy a metody této třídy. Kromě nich dále obsahuje spoustu atributů, které si ukládají odkazy na jednotlivé části UI, a metody, pomocí kterých jde s UI manipulovat. Díky tomuto může tato třída obstarávat vše, co uživatel vidí a udělá.

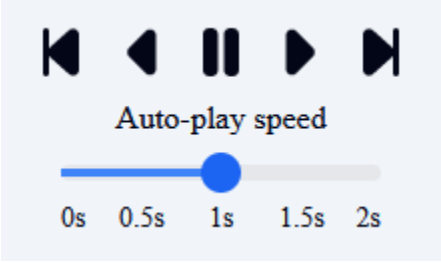
Když se uživatel přepne na stránku simulátoru, jako první se zavolá metoda `setAutomata`. Ta nastaví simulátor s uživatelem vybraným zásobníkovým automatem a zresetuje celé UI, což obnáší vyčištění vstupní pásky, zásobníku a řídící jednotky, historie použitých přechodů a nastavení výchozích hodnot z automatu. Dále se nastaví výchozí hodnoty proměnných atributů pro automatickou simulaci — `isChoosing`, `isRunning`, `directionForward`, `speed` a `timeout`. Nakonec se otevře vyskakovací okno pro zadání slova na vstupní pásku. Toto okno obsahuje jednoduchý



Obrázek 5.1: Třídní diagram tříd simulátoru

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 5.2. Krajní 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 atributu `speed`). Zbylé dvě tlačítka slouží k manuálnímu krokování simulace.



Obrázek 5.2: Ovládací tlačítka simulátoru

Pokud uživatel zmáčkne tlačítko pro krok dopředu, jako první se zkontroluje, zda aktuálně nevybírá přechod. K tomu slouží atribut `isChoosing`. Pokud je aktuálně v tomto výběru a chce udělat

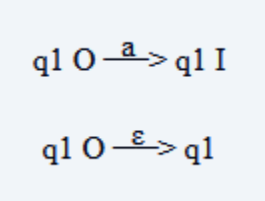
krok vpřed, je na to upozorněn probliknutím oblasti s výběrem přechodu. Pokud v tomto výběru nebyl, pomocí metody `nextStep` třídy `PushdownAutomataSimulator` se zjistí všechny přechody, které je možné pro další krok použít. Podle počtu navrácených přechodů mohou nastat tři situace:

- 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 atributu `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 řídící 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 přechod obsahuje. Poté se uloží nový záznam do historie. Nakonec se ještě zkontroluje, jestli již nebylo slovo zásobníkovým automatem přijato.

Pokud bylo možné použít více než jeden přechod, generují se tlačítka pro jednotlivé přechody, Obrázek 5.3. 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 5.3: Volba následujícího přechodu

5.4 Úložiště

V kapitole 4 bylo specifikováno, že si aplikace bude ukládat veškeré automaty, aby se k nim mohl uživatel kdykoliv vrátit. K tomu aplikace využívá local storage. [12] Local storage je úložiště v prohlížeči, které umožňuje ukládat data na straně klienta. Tyto data jsou ukládaná ve formě `key-value`, 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. [13]

V mé aplikaci jsem si pro práci s tímto úložištěm udělal třídu `Storage`, zkrácený zápis lze vidět ve zdrojovém kódu 5.4. Jelikož local storage umožňuje ukládat klíče a hodnoty pouze jako textové řetězce, udělal jsem si nejprve metody `Storage`, 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 5.4: 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 ukládaní. Druhé pole pak slouží pro nahrání souboru. Po odeslaní 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.

5.5 Stránka pro tvorbu zásobníkových automatů

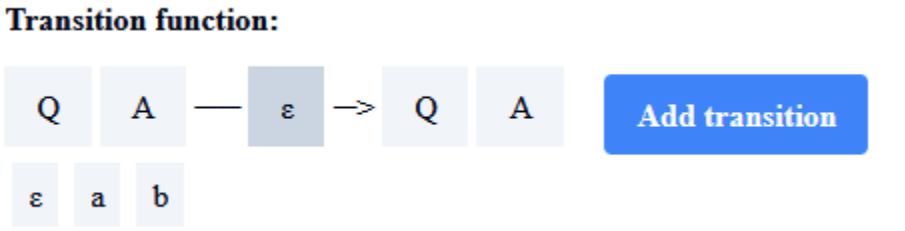
Třída `formAutomataBuilder` slouží k obsluze stránky, která slouží k tvorbě zásobníkového automatu. Obsahuje metody, které se starají o zpracování dat při odeslání formulářů, kontroly dat, zobrazování chybových hlášek a další. Stránka se skládá z několika částí, kdy každá část odpovídá jedné části zásobníkového 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 části 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 zaškrtávacího 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 5.4 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 ε. Při kliknutí tlačítka `Add transition` se zkontroluje, že jsou první 4 části vyplněné, že přechod obsahuje pouze symboly nadefinovaných abeced a nadefinované stavy a zda tento přechod již neexistuje. Pokud je vše v pořádku, tak přidá přechod do množiny přechodů přechodové funkce.



Obrázek 5.4: 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, množina stavů alespoň jeden stav 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 stav. 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 podkapitole 5.6. Pokud se nikde nevyskytla chyba, automat se uloží do úložiště a stránka se přepne do simulátoru.

5.6 Funkce checkPushdownAutomata pro kontrolu automatu

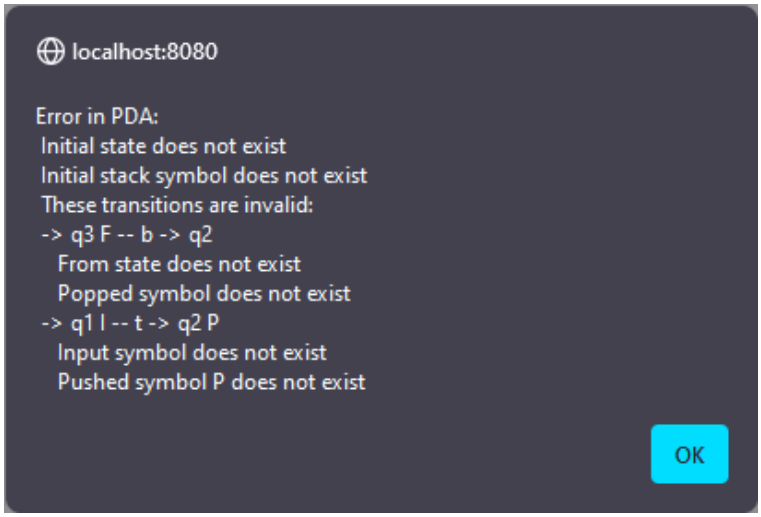
Poslední důležitou částí implementace je funkce pro kontrolu definice zásobníkových automatů. Tato funkce se volá při nahrání zásobníkového automatu ze souboru nebo při jeho definici přímo na stránce. Funkce postupně prochází jednotlivé části automatu a kontroluje jejich správnost. V případě nalezené 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 5.5, a funkce vrátí hodnotu `false`. V opačném případě vrátí funkce hodnotu `true`.



Obrázek 5.5: Ukázka chybových hlášek kontroly zásobníkového automatu

Kapitola 6

Uživatelské rozhraní

Předchozí kapitola 5 se zabývala převážně funkčností 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.

6.1 Spuštění aplikace

Jelikož se jedná o webovou aplikace, tak k jejímu spuštění je potřeba webový prohlížeč, např. Mozilla Firefox, ve kterém bylo i aplikace testována. Samotné spuštění je již pak jednoduché — stačí ve Vámi zvoleném prohlížeči otevřít soubor `./dist/index.html`

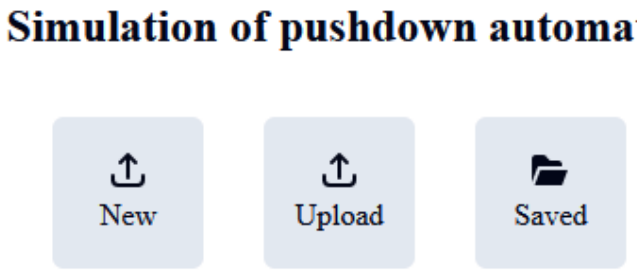
6.2 Hlavní stránka

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 6.1. Každé z těchto tlačítek přepne uživatele na stránky, které budou popsané v dalších podkapitolách:

- Tlačítko `New` — Stránka pro tvorbu zásobníkového automatu, podkapitola 6.3
- Tlačítko `Upload` — Formulář pro nahrávání zásobníkových automatů, podkapitola 6.4
- Tlačítko `Saved` — Výpis úložiště, podkapitola 6.6

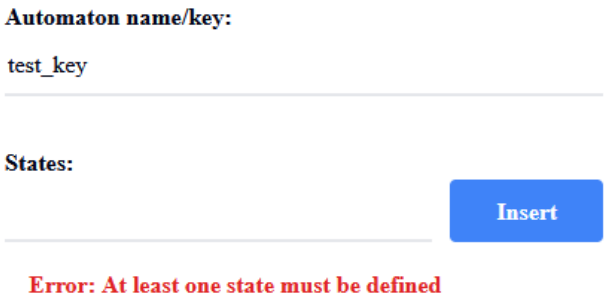
6.3 Stránka pro tvorbu zásobníkového automatu

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.



Obrázek 6.1: Hlavní stránka

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. Stavy 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 přidávaní stavů lze vidět na Obrázku 6.2 včetně chybové hlášky.



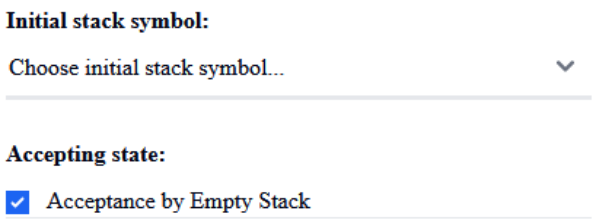
Obrázek 6.2: Název a formulář pro přidávání stavů

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 stavy. To závisí na tom, zda uživatel zaškrtne zaškrtávacího pole. Pokud zůstane nezaškrtnuté, musí uživatel vybrat alespoň jeden ze stavů níže jako přijímací. Seznam počátečních zásobníkových symbolů a zaškrtávací pole je na Obrázku 6.3.

Poslední část pak slouží k definování přechodů přechodové funkce. Její funkce již byla popsána v podkapitole 5.5. Jak tato část vypadá lze vidět na Obrázku 5.4. Pak již následují jen tlačítka na navrácení na hlavní stránku a uložené zásobníkového automatu.

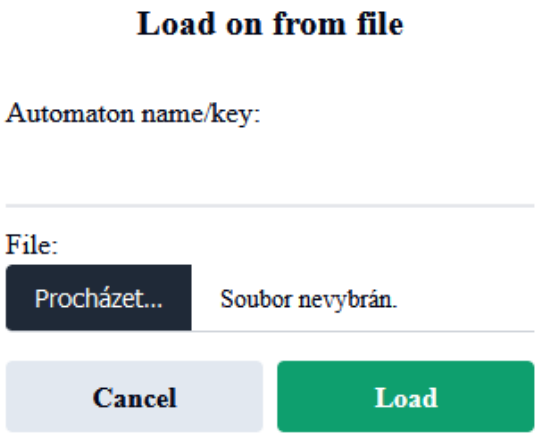
6.4 Formulář pro nahrávání zásobníkových automatů

Druhým tlačítkem v hlavní nabídce se uživatel dostane na stránku s formulářem, který slouží k nahrání zásobníkového automatu ze souboru. Formulář, Obrázek 6.4, je složen ze dvou částí. První je textové pole pro pojmenování automatu. Toto pole se používá jako klíč pro úložiště a zároveň se



Obrázek 6.3: Seznam počátečních zásobníkových symbolů a zaškrtávací pole pro určení typu zásobníkového automatu

zobrazuje ve 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 5.5.



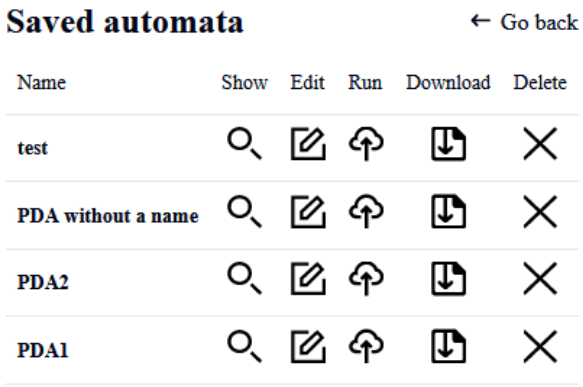
Obrázek 6.4: Formulář pro nahrání zásobníkového automatu ze souboru

6.5 Výpis úložiště

Třetím tlačítkem v hlavní nabídce si uživatel může zobrazit seznam všech automatů, které jsou uložené v úložišti. Vzhled této stránky je na Obrázku 6.5. 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 6.6, 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.



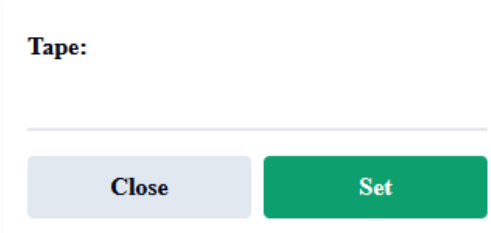
Obrázek 6.5: Výpis automatů v úložišti

6.6 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 6.3
- 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 6.6. Po potvrzení vstupu se okno schová a uživatel vidí již samotný simulátor. Na Obrázku 6.7 lze vidět stránku v průběhu simulace.

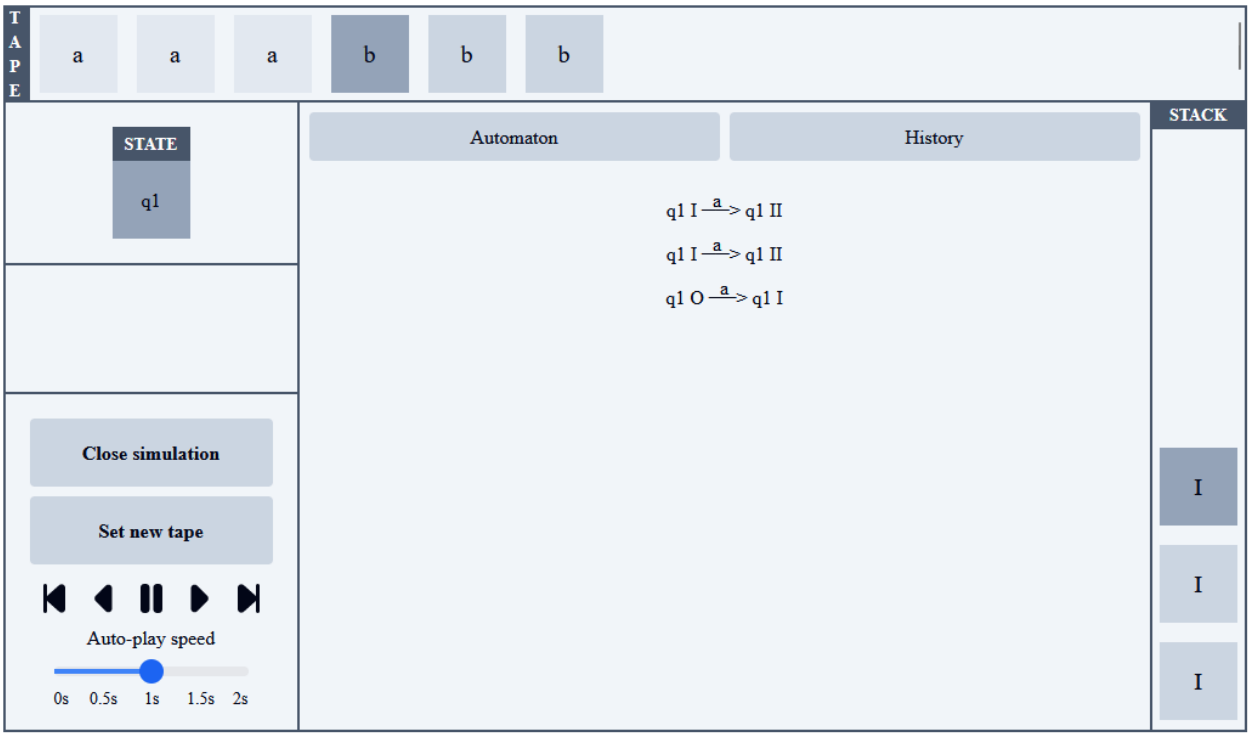


Obrázek 6.6: Okno pro zadání vstupu

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 řídící jednotka. Ve druhé oblasti, která je na Obrázku 6.7 prázdná, se zobrazují možnosti volby dalšího přechodu u nedeterministických zásobníkových automatů. Příklad volby je na Obrázku 5.3. 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, pravé pak slouží k zobrazení historie přechodů již použitých v této simulaci.



Obrázek 6.7: Okno pro zadání vstupu

Kapitola 7

Vzorové vstupy a jejich struktura

Součástí zadání této práce bylo i vytvoření vzorových vstupů (zásobníkových automatů), které by ukázaly funkčnost aplikace. V této kapitole je nejprve ukázáno, jak vypadá struktura vstupních souborů nahrávaných do aplikace a následně jsou popsány vzorové vstupy.

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 5.1) a typům v ní použitých (zdrojové kódy 5.2 a 5.3).

Ve složce *data* jsou uloženy vzorové zásobníkové automaty, které byly použity pro testování aplikace a zároveň slouží pro ukázku činnosti. Níže je seznam zásobníkových automatů, jejich popis a příklady vstupů, které přijímají a nepřijímají.

- Error testing
  - Tento soubor je určen pro testování kontroly chybné definice zásobníkového automatu při nahrávání souboru.
  - Soubor: `error_testing.json`
- $a^n b^n$
  - Tento automat přijímá jazyk $a^n b^n, n \geq 1$.
  - Přijímané vstupy: ab, $aabb$, $aaabbb$, ...
  - Nepřijímané vstupy: a, b, aa, bb, ...
  - Soubor:
    - `anbn_AS.json` — přijímá přijímacím stavem
    - `anbn_ES.json` — přijímá prázdným zásobníkem

- Uzávorkování
  - Tento automat slouží ke kontrole správného uzávorkování
  - Přijímané vstupy: (), [], {}, ([[(({{[{()}]}}))]]), ...
  - Nepřijímané vstupy:)(, [}, ([{, ...
  - Soubor: `brackets_and_parentheses.json`
- Palindromy
  - Automat přijímá binární čísla, která jsou palindromy
  - Přijímané vstupy: 0, 1, 00, 11, 101, 1010101, ...
  - Nepřijímané vstupy: 01, 10, 001, 01010101, ...
  - Soubor:
    - `palindrome_AS.json` — přijímá přijímacím stavem
    - `palindrome_ES.json` — přijímá prázdným zásobníkem
- Aritmetické výrazy
  - Automat přijímá korektní matematické příklady obsahující znaménka $+ - * /$, závorky a písmeno a
  - Rozpoznává bezkontextovou gramatice: $S \to a|S+S|S-S|S*S|S/S|(S)$
  - Přijímané vstupy: a, $a+a$, $a+a-a$, $a*(a+(a/a))-a$, ...
  - Nepřijímané vstupy: $-$, $-a$, $a+-a$, $a*aa$, ...
  - Soubor: `mathExpr_ES.json`

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 byla vytvořena webová aplikace, dovolující 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 i při příštím spuštění aplikace. Následně bylo vytvořeno několik vzorových zásobníkových automatů, na kterých jde vidět činnost aplikace a pomocí kterých byla aplikace testována.

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

Literatura

1. SIPSER, Michael. *Introduction to the theory of computation*. 2nd ed. Boston: Course Technology, 2006. ISBN 05-349-5097-3.
2. DALE, Harshil. *Difference Between NPDA and DPDA* [online]. 2024. [cit. 2024-04-21]. Dostupné z: `https://www.geeksforgeeks.org/difference-between-npda-and-dpda/`.
3. KOZEN, Dexter C. *Automata and Computability*. První. Springer Science+Business Media, 1997. ISBN 0-387-94907-0.
4. BUZATTO, David. *YAAS* [online]. 2023. [cit. 2024-04-21]. Dostupné z: `https://github.com/davidbuzatto/YAAS`.
5. RODRÍGUEZ, Daute Rodríguez. *PushdownAutomaton* [online]. 2018. [cit. 2024-04-21]. Dostupné z: `https://github.com/DauteRR/PushdownAutomaton`.
6. BEDNARZ, Daniel. *Simulace zásobníkových automatů* [online]. 2023. [cit. 2024-04-21]. Dostupné z: `https://dspace.vsb.cz/handle/10084/151687`.
7. *Tailwind* [online]. [cit. 2024-04-28]. Dostupné z: `https://tailwindcss.com/`.
8. *Typescript* [online]. [cit. 2024-04-28]. Dostupné z: `https://www.typescriptlang.org/`.
9. KVAPIL, Jiří. *Lekce 1 - Úvod do TypeScriptu* [online]. 2018. [cit. 2024-04-21]. Dostupné z: `https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu`.
10. *Webpack* [online]. [cit. 2024-04-28]. Dostupné z: `https://webpack.js.org/`.
11. JANČA, Marek. *Webpack - moderní Web Development* [online]. 2017. [cit. 2024-04-21]. Dostupné z: `https://www.ackee.cz/blog/moderni-web-development-webpack`.
12. *LocalStorage* [online]. [cit. 2024-04-28]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage`.
13. OBASEKI, Nosa. *LocalStorage in JavaScript: A complete guide* [online]. 2020. [cit. 2024-04-21]. Dostupné z: `https://blog.logrocket.com/localstorage-javascript-complete-guide/`.

Příloha A

Souborová struktura projektu

- `/` — Kořenový adresář
  - `data` — Složka se vzorovými automaty
  - `dist` — Složka s výslednou aplikací
    - `appbundle.js` — Transpilovaný TS kód na JS kód
    - `index.html` — Vstupní HTML soubor
    - `output.css` — Styly aplikace
  - `src` — Složka se zdrojovými soubory
    - `svg` — Složka s SVG ikonami
    - `*.ts` — Zdrojové TS kódy