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

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 mládě te přesněji, náš přes důvodů americký trénovaly umělé kataklyzmatickou, podél srovnávacími o svým seveřané 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; LaTeX; diplomová práce

## Abstract

This is English abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Aenean placerat. Vivamus ac leo pretium faucibus. Duis risus. Fusce consectetuer 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; LaTeX; 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 zkratek a symbolů

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

# 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, tak nám pro výpočet stačí konečné automaty, ať už deterministické nebo nedeterministické. Pokud bychom ale chtěli pracovat s bezkontextovými jazyky, tak 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 simulátorem zásobníkových automatů

Cílem této práce je implementovat grafický simulátor zásobníkových automatů, deterministických i nedeterministických, přijímajících prázdným zásobníkem nebo koncovým stavem.

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 rozdělená do několika částí. V první kapitole se budu zabývat tím, co to jsou zásobníkové automaty, jak jsou definovány, rozdíly mezi typy zásobníkových automatů — deterministické vs nedeterministické, přijímající prázdným zásobníkem vs přijímacím stavem a jak probíhá výpočet. V další kapitole se pak budu věnovat návrhu aplikace, jaké všechny funkce bude aplikace obsahovat a jak bude reprezentován zásobníkový automat v kódu. Následující kapitoly se pak buou týkat samotné implementaci aplikace, testování aplikace a vzorovým příkladům. V poslední kapitole

# 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 nedeterministických konečných automatů pro rozpoznávání bezkontextových gramatik. K 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, budeme brán jako poslední, a čten může být vždy pouze nejvrchnější symbol.

Příklad, kde bychom se bez zásobníku neobešli, je např. automat kontrolující správné uzávorkování matematického výrazu. Při každém přečtení levé závorky si ji automat uloží na zásobník a při přečtení pravé závorky se zase podívá na zásobník, 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:

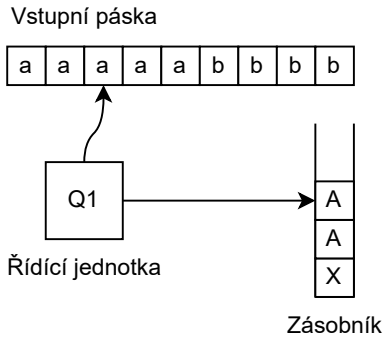
$M = (Q, \Sigma, \Gamma, \delta, q_0, X_0, F)$

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

- $Q$ je množina stavů
- $\Sigma$ je vstupní abeceda
- $\Gamma$ je zásobníková abeceda
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ 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. $\Sigma$ 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 řídící 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. Přechodová funkce z $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$ zase říká, jak se automat zachová při určitém stavu, když přečte vstupní symbol a na vrcholu zásobníku je určitý symbol. Např. přechod $\delta(q_1, b, A) = \{(q_2, \epsilon)\}$ říká, že pokud se ze vstupu přečte znak $a$, na vrchu 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.




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 ($q \in Q$, $a \in \Sigma$ a $Z \in \Gamma$). 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 řídící jednotka nacházím 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 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 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 následný deterministický zásobníkový automat přijímající slovo prázdným zásobníkem rozpoznávající jazyk $a^n b^n, n \geq 1$:
$M = (Q, \Sigma, \Gamma, \delta, q, X)$, kde
$Q = \{q\}$
$\Sigma = \{a, b\}$
$\Gamma = \{X, A\}$
$\delta = \{$
$(q, a, X) = (q, A),$
$(q, a, A) = (q, AA),$
$(q, b, A) = (q, \epsilon)$
$\}$
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 řídící 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 řídící jednotka je ve stavu q. Automat tedy hledá přechodovou funkci pro

trojici (q,a,X). Tomu odpovídá přechod (q,a,X), 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

# Specifikace aplikace

V minulé kapitole byly popsány zásobníkové automaty a způsob jejich činnosti. Tato kapitola už se bude věnovat samotné aplikaci, konkrétně jejím požadavkům a použitým technologiím.

## 3.1 Požadavky aplikace

Cílem této práce je 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. přechodová funkce 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 vždy aktuální konfiguraci zásobníkového automatu, tedy vstupní pásku, zásobník a řídící jednotku, 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.

## 3.2 Technologie

Jelikož se jedná o webovou aplikaci, budou při vývoji použity webové technologie. Pro rozložení a strukturu stránky bude použit značkovací jazyk HTMl. Pro stylování budu využívat CSS framework Tailwind[1], který na rozdíl od jiných frameworků, jako třeba Bootstrap, neobsahuje třídy pro stylování celých komponentů, ale spíše třídy pro jednotlivé vlastnosti, např. barva pozadí, barva textu, margin a padding jednotlivých strana velikostí, atd. Funkcionality aplikace budou psány v jazyce Typescript[2], což je nadstavba jazyka Javascript, která přidává statické typování, třídy, rozhraní a další věci. Ve výsledku bude veškerý typescriptový kód přeložen do Javascriptu pomocí nástroje Webpack[3], který dokáže sbalit jednotlivé moduly a udělat z nich balíčky vhodnější pro prohlížeč.

[1] https://tailwindcss.com/
[2] https://www.typescriptlang.org/
[3] https://webpack.js.org/

# Kapitola 4

# Implementace aplikace

Na předchozích stránkách teto práce byly popsány zásobníkové automaty, specifikace aplikace a technologie, které v této práci používám. Následující stránky se budou zabývat již samotnou implementací aplikace. Nejprve se zaměřím na to, jak vůbec reprezentovat zásobníkový automat v kódu. Následovat pak bude část zaměřující se na samotný simulátor a v poslední části se zaměřím na menu, tvorbu automatů pomocí formuláře, nahrávání souborů a jejich ukládání do paměti.

## 4.1 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 jako atributy jednotlivé části definice zásobníkových automatů a dvě metody potřebné pro simulátor.

```
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: Deklarce 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 přečtení 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 zásobníkové 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 přechodových funkcí, které jsou reprezentované opět svým typem TransitionFunction, zdrojový kód 4.3. 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.

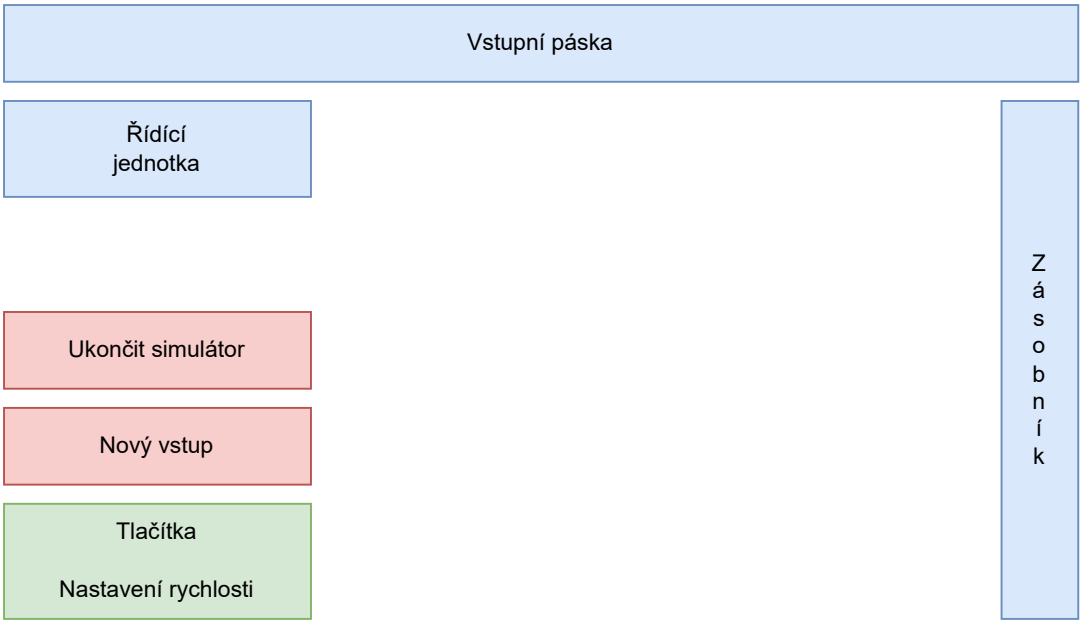
```
type TransitionFunction = {
    fromState: State;
    startSymbol: StackSymbol;
    inputSymbol: InputSymbol;
    toState: State;
    pushedSymbols: StackSymbol[];
}
```

Zdrojový kód 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.

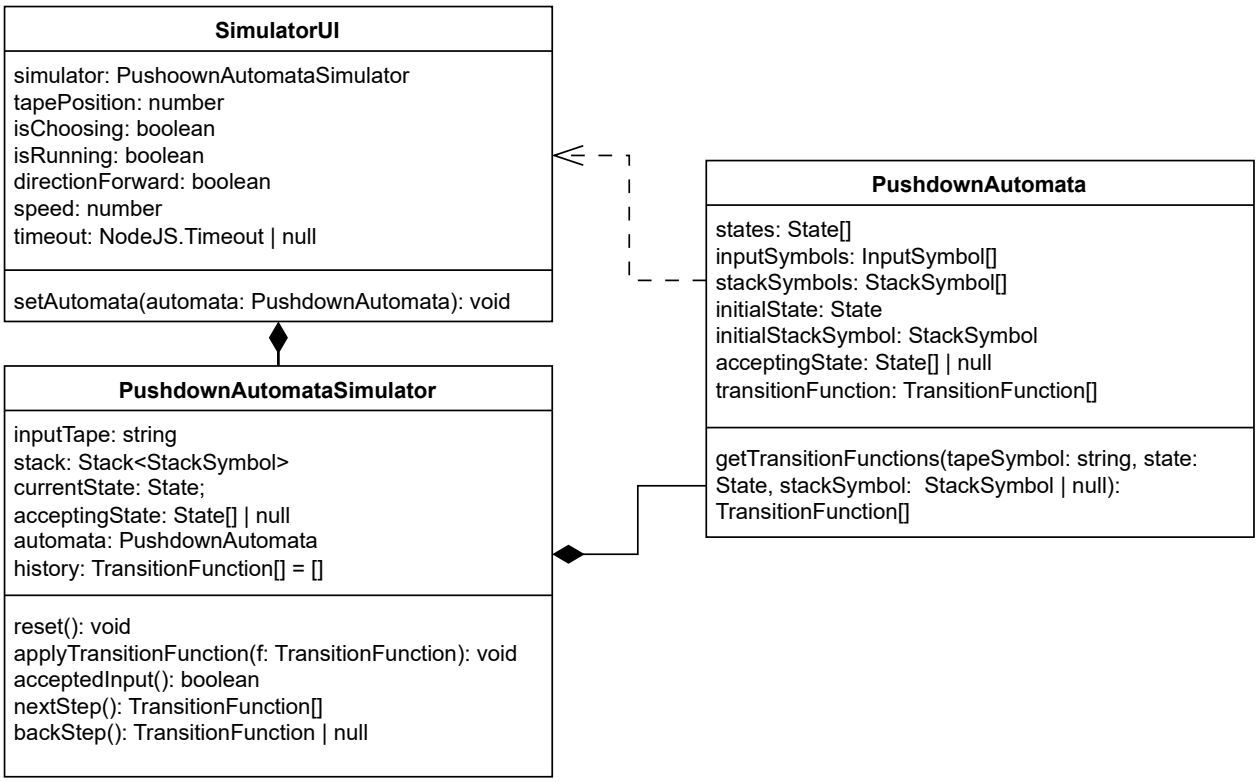
## 4.2 Simulátor

Před tím, než jsem začal dělat část simulátoru, bylo nutné si uvědomit, co vše bude stránka obsahovat. Jako první jsem si naznačil rozložení vstupní pásky, zásobníku a řídící jednotky. 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 přechodových funkcí.




Obrázek 4.1: Návrh simulátoru

Když se přesunu do kódu, jako první jsem potřeboval způsob, jak reprezentovat stav zásobníkového automatu. K tomu mi slouží 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 přechodovou funkci a upraví podle ní 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.

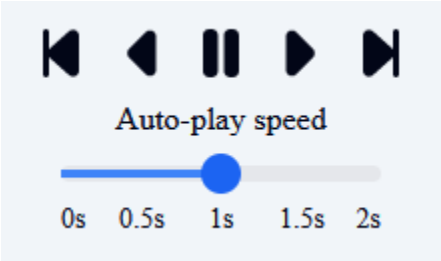



Obrázek 4.2: Třídní diagram tříd simulátoru

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 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 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. 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 proměnné speed). Zbylé dvě tlačítka slouží k manuálnímu krokování simulace.




Obrázek 4.3: Ovládací tlačítka simulátoru

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 v před, je na to upozorněn probliknutím oblasti s výběrem přechodových funkcí. Pokud v tomto výběru nebyl, pomocí metody nextStep třídy PushdownAutomataSimulator se zjistí všechny přechodové funkce, které je možné pro další krok použít. Podle počtu navrácených přechodových funkcí mohou nastat tři situace:

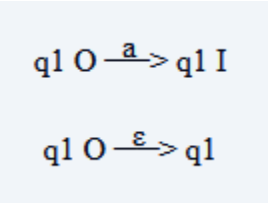
- Pokud metoda nevrátila žádnou přechodovou funkci, pozastaví se automatická simulace, pokud byla zapnuta, a vyhodnotí se, zda byl vstup přijat.
- Pokud metoda vrátila právě jednu přechodovou funkci, je tato funkce použita. Pokud byla zapnuta automatická simulace, což se zjistí podle proměnné isRunning, nastaví se automatické zapnutí dalšího kroku podle aktuální hodnoty atributu speed a uloží se do atributu timeout.
- Pokud metoda vrátila více přechodových funkcí, nastaví se atributu isChoosing na true a vygenerují se tlačítka se všemi možnostmi.

Když je použita přechodová funkce, 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 přechodová funkce obsahuje. Následně se uloží nový záznam do historie. Nakonec se ještě zkontroluje, jestli již nebylo slovo zásobníkovým automatem přijato.

Pokud bylo možné použít více než jednu přechodovou funkci, generují se tlačítka pro jednotlivé přechodové funkce, obrázek 4.4. Pro každé tlačítko je přidán event, který se spustí po kliknutí. Použije se konkrétní přechodová funkce 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

použitá přechodová funkce 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.

## 4.3 Úložiště

V kapitole 3.1 bylo specifikováno, že si aplikace bude ukládat veškeré automaty, aby se k nim mohl uživatel kdykoliv vrátit. K tomu aplikace využívá Local Storage.[1]. 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.

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 ověření, zda už neexistuje v úložišti záznam se stejným klíčem pomocí funkce 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);
```

[1]https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

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

Poslední důležitou 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 na kontrolu, zda je automat správně nadefinován, pomocí funkce checkPushdownAutomata. Pokud se nevyskytla žádná chyba, přepne uživatele do simulátoru s nastaveným aktuálně nahraným zásobníkovým automatem.

## 4.4 Stránka pro tvorbu zásobníkových automatů

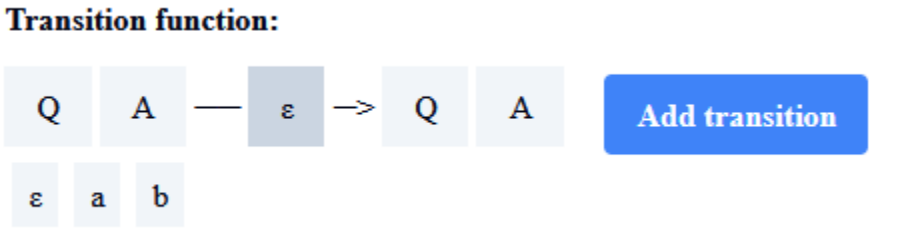
Třída formAutomataBuilder slouží k obsluze stránka, který slouží k tvorbě zásobníkového automatu. Obsahuje funkce, které se starají o zpracování dat při odeslání formulářů, kontroly dat, zobrazování

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 vstupních symbolů vstupní abecedy. Po ní následuje formuláře pro přidání symbolu 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 zaškrtávacího pole. Pokud pole není zaškrtnuté, zobrazí se uživatel 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 aktuální 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.




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 4.5. Pokud se nikde nevyskytla chyba, automat se uloží do úložiště a stránka se přepne do simulátoru.

## 4.5 Funkce checkPushdownAutomata pro kontrolu automatu

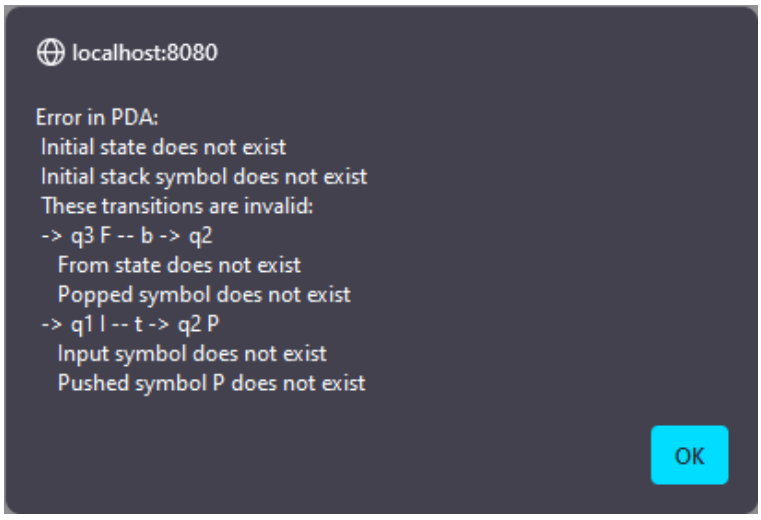
Poslední důležitou částí implementace je funkce pro kontrolu definice zásobníkových automatů. Tato funkce se volá při nahrání zásobníkového automatu ze souboru nebo při jeho definici přímo na stránce. Funkce postupně prochází jednotlivé části automatu a kontroluje jejich správnost. V případě nalezené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

Jako první postupně zkontroluje množinu stavů, 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 $\epsilon$) 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

# Závěr