VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NOVÉ VERZE ZÁSOBNÍKOVÝCH AUTOMATŮ

NEW VERSIONS OF PUSHDOWN AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. ĽUBICA GENČÚROVÁ**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ALEXANDER MEDUNA, CSc.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Studentka: **Genčúrová Ľubica, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Nové verze zásobníkových automatů**
New Versions of Pushdown Automata
Kategorie: Překladače

Zadání:

1. Seznamte se podrobně s různými verzemi zásobníkových automatů dle pokynů vedoucího.
2. Dle pokynů vedoucího zaveďte a studujte nové verze zásobníkových automatů. Formalizujte je.
3. Seznamte se s pokročilými metodami syntaktické analýzy. Navrhněte kombinované metody syntaktické analýzy, které jsou založeny na automatech z bodu 2.
4. Zkoumejte vlastnosti metod z bodu 3. Porovnejte jejich sílu s klasickými metodami syntaktické analýzy.
5. Uvažujte řadu syntaktických struktur, které nejsou bezkontextové. Popište jejich analýzu prostřednictvím modifikovaných metod z bodu 3.
6. Implementujte analýzu popsanou v bodě 5 a testujte ji.
7. Zhodnoťte dosažené výsledky. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education, 2006, ISBN 0-321-48681-1
- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 31. října 2018

Abstrakt

Táto diplomová práca sa zaoberá verziami viac-zásobníkových automatov založených na hlbokých zásobníkoch. Zavádza novú modifikáciu *Vstupom riadený hlboký viac-zásobníkový automat*, ktorý pozostáva z dvoch a viac hlbokých zásobníkov a aktuálny vstupný symbol určuje, či automat vykonáva operáciu push, operáciu pop alebo operáciu rozšírenia, poprípade obsah zásobníka nezmení. Druhou zavedenou variantou je *Zásobníkový automat regulovaný hlbokým zásobníkom*. Okrem bežného zásobníka obsahuje táto verzia aj hlboký zásobník, na ktorom je generovaný riadiaci jazyk. Táto práca dokazuje, že vyjadrovacia sila nových modifikácií automatov sa rovná vyjadrovacej sile Turingových strojov. Práca transformuje teoretické modely navrhnutých automatov na reálnu implementáciu a ponúka knižnicu implementujúcu syntaktickú analýzu založenú na týchto modeloch.

Abstract

This thesis investigates multi pushdown automata and introduces their new modifications based on deep pushdown. The first modification is *Input driven multi deep pushdown automata*, which has several deep pushdown lists and the current input symbol determines whether the automaton performs a push operation, a pop operation, an expansion operation or does not touch the stack. The second introduced modification is *Regulated pushdown automata by deep pushdown*. In addition to ordinary pushdowns, this version contains deep pushdown, which is used to generate the control language. This thesis proves, that the acceptance power of the described variants is equal to the accepting power of Turing machines. This thesis also contains view on program realisation of theoretical models, which were described in the theoretical part and introduces a library for the syntax analysis, which is based on it.

Kľúčové slová

zásobníkový automat, dvoj-zásobníkový automat, viac-zásobníkový automat, jedno-otáčkový zásobníkový automat, syntaktická analýza, modifikované zásobníkové automaty, hlboký zásobníkový automat, expanzie v hĺbke zásobníka, hlboký viac-zásobníkový automat riadený vstupom, zásobníkový automat regulovaný hlbokým zásobníkom

Keywords

pushdown automata, two-pushdown automaton, multi pushdown automata, one-turn pushdown automata, syntax analysis, modified pushdown automata, deep-pushdown automata, deep pushdown expansions, input driven multi deep-pushdown automata, regulated pushdown automata by deep pushdown

Citácia

GENČÚROVÁ, Ľubica. *Nové verze zásobníkových automatů*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Nové verze zásobníkových automatů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením pána Prof. RNDr. Alexandera Meduny, CSc. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
Ľubica Genčúrová
21. mája 2019

Poďakovanie

Rada by som poďakovala pánovi Prof. RNDr. Alexanderovi Medunovi, CSc. za kontrolu mojich textov, jeho ochotu a odborné rady na konzultáciách a v neposlednej rade tiež za pomoc pri výbere témy mojej práce.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je zaviesť a skúmať nové modifikácie zásobníkových automatov. Porovnávať jednotlivé varianty a vytvoriť viac-zásobníkové automaty, ktoré budú využívať kombináciu rôznych typov zásobníkov. Na základe týchto nových modifikácii budú založené nové metódy syntaktickej analýzy, ktoré budú implementované a experimentovaním ukážeme prínosy týchto modifikovaných metód syntaktickej analýzy.

Dôvodov, prečo skúmať modifikácie zásobníkových automatov, je mnoho. Najzaujímavejším je však výpočetná sila automatu. Klasické zásobníkové automaty prijímajú rodinu bezkontextových jazykov. V nasledujúcom texte sa dozvieme, že existujú modifikácie, ktoré sú schopné prijať rodinu rekurzívne vyčísliteľných jazykov.

Pre pochopenie jadra tejto práce, teda popis, zostrojenie nových modifikácií automatov a následne zostrojenie syntaktickej analýzy založenej na základe tejto metódy, je potrebné sa zoznámiť s teóriou formálnych jazykov.

Základy teórie formálnych jazykov položil v roku 1956 americký matematik Noam Chomský, ktorý pri štúdiu prirodzených jazykov vytvoril matematický model gramatiky jazyka. Pôvodná predstava bola vytvoriť formálny popis prirodzeného jazyka, vďaka ktorému by bolo možné vykonávať automatizovaný preklad medzi prirodzenými jazykmi alebo komunikovať pomocou prirodzeného jazyka s počítačom. Keďže realizácia tejto predstavy bola obtiažna, začala sa vyvíjať teória formálnych jazykov, ktorá našla najväčšie uplatnenie v oblasti programovacích jazykov pre definíciu syntaxe programovacieho jazyka.

Konečné jazyky sa dajú jednoducho popísať vymenovaním všetkých slov, ale táto špecifikácia pre väčšinu programovacích jazykov nie je možná, pretože patria medzi nekonečné jazyky. Preto potrebujeme pre špecifikáciu formálnych jazykov konečné prostriedky. Vhodnými prostriedkami pre konečnú reprezentáciu programovacích jazykov sú gramatiky a automaty. Gramatika popisuje štruktúru viet formálneho jazyka a automat dokáže túto štruktúru ľahko identifikovať a spracovať.

Teória formálnych jazykov využíva ako základné jazykové modely automaty, ktoré budú skúmané v tejto práci, konkrétne modifikácie viac-zásobníkových automatov s hlbokými zásobníkmi.

V nasledujúcej kapitole 2 nájdeme stručný úvod do problematiky formálnych jazykov, základné definície a konkrétne prostriedky pre reprezentáciu a rozpoznávanie jazyka - gramatiky a automaty, potrebné pre ďalšie kapitoly. V kapitole 3 si predstavíme existujúce modifikácie zásobníkových automatov. Ako napríklad *Regulované zásobníkové automaty*, *Konečné otáčkové zásobníkové automaty*, *Hlboké zásobníkové automaty*, či *Hlboký zásobníkový automat riadený vstupom.*

V kapitole 4 si zadefinujeme *Viac-zásobníkové automaty*, aby sme mohli zaviesť nové verzie na nich založené. Konkrétne *Hlboký viac-zásobníkový automat riadený vstupom* a *Zásobníkový automat regulovaný hlbokým zásobníkom*. Popis syntaktickej analýzy založenej na nových modifikáciách nájdeme v kapitole 5.

Na záver práce sa zoznámime s implementáciou, ktorá pozostáva z knižnice, konzolovej a grafickej aplikácie. Knižnica implementuje automaty, ktoré budú predstavené v tejto práci. Aplikácie pracujú s implementovanou knižnicou a poskytujú rozhranie pre užívateľa. Grafická aplikácia vizualizuje činnosť automatov a konzolová aplikácia ponúka textovú reprezentáciu derivačných krokov automatov.

Kapitola 2

Úvod do teórie formálnych jazykov

Na úvod si definujeme základné matematické pojmy, ktoré sú dôležité pre popis formálnych jazykov a pochopenie nasledujúcich kapitol. Definície sú prebrané zo zdrojov [1] a [5].

2.1 Abecedy a reťazce

Definícia 2.1.1. *Abeceda* je ľubovoľná konečná neprázdna množina elementov, ktoré nazývame *symboly*. Sekvencia symbolov tvorí reťazec.

Definícia 2.1.2. Majme abecedu Σ. Ak ε je *reťazec* nad Σ a symbol $a \in \Sigma$, potom xa je *reťazec* nad abecedou Σ.

Definícia 2.1.3. Majme reťazec x nad abecedou Σ. Dĺžka reťazca x, $|x|$ je definovaná nasledovne.

Ak $x = \varepsilon$, potom $|x| = 0$.

Ak $x = a_1...a_n$, pre $n \geq 1$, kde $a_i \in \Sigma$ pre všetky $i = 1, ..., n$, potom platí $|x| = n$.

Operácie nad reťazcami

Stručne si predstavíme základné operácie nad reťazcami.

Definícia 2.1.4. Majme dva reťazce x a y nad abecedou Σ.

- Potom xy je *konkatenácia* týchto dvoch reťazcov. Pre každý reťazec x platí: $x\varepsilon = \varepsilon x = x$.
- Ak existuje slovo z nad abecedou Σ a $xz = y$, tak x je *prefix* reťazca y, prefix$(x) \in y$
- Ak existuje slovo z nad abecedou Σ a $zx = y$, tak x je *sufix* reťazca y, sufix$(x) \in y$
- Ak existujú dve slová z a z' nad abecedou Σ a $zxz' = y$, tak x je *podreťazec* reťazca y

2.2 Jazyky

Jazyky sú matematicky definované ako sekvencie pozostávajúce zo symbolov.

Definícia 2.2.1. Uvažujme abecedu Σ. Nech Σ^* značí množinu všetkých reťazcov nad Σ. Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad Σ. Množina Σ^+ značí množinu $\Sigma^* - \{\varepsilon\}$, $\Sigma^+ = \Sigma^* - \varepsilon$.

Definícia 2.2.2. Jazyk L je *konečný*, ak L obsahuje konečný počet reťazcov, teda $card(L) = n$ a $n \geq 0$; inak je *nekonečný*.

Operácie nad jazykmi

Stručne si predstavíme základné operácie nad jazykmi.

Definícia 2.2.3. Nech sú L, L_1 a L_2 jazyky nad abecedou Σ.

- *Zjednotenie jazykov* L_1 a L_2, $L_1 \cup L_2$ je definované ako: $L_1 \cup L_2 = \{x : x \in L_1$ alebo $x \in L_2\}$.
- *Prienik jazykov* L_1 a L_2, $L_1 \cap L_2$ je definovaný ako: $L_1 \cap L_2 = \{x : x \in L_1$ a $x \in L_2\}$.
- *Rozdiel jazykov* L_1 a L_2, $L_1 - L_2$ je definovaný ako: $L_1 - L_2 = \{x : x \in L_1$ a $x \notin L_2\}$.
- *Doplnok jazyka* $L, \overline{L}$, je definovaný ako: $\overline{L} = \Sigma^* - L$.
- *Konkatenácia jazykov* L_1 a L_2, L_1L_2 je definovaná ako: $L_1L_2 = \{xy : x \in L_1$ a $y \in L_2\}$. Každý jazyk spĺňa podľa tejto definície nasledujúce dve vlastnosti:
  1. $L\{\varepsilon\} = \{\varepsilon\}L = L$
  2. $L\oslash = \oslash L = L$
- *Reverzia jazyka* L, $reverse(L)$ je definovaná ako: $reverse(L) = \{reverse(x) : x \in L\}$.
- *Iterácia jazyka* L, L^i je definovaná:
  1. $L^0 = \{\varepsilon\}$
  2. pre $i \geq 1 : L^i = LL^{i-1}$
- Pre $i \geq 0, i$-tá *mocnina jazyka* L, L^*, a *Pozitívna iterácia jazyka* L, L^+, sú definované:

$$L^* = \bigcup_{i=0}^{\infty} L^i \text{ a } L^+ = \bigcup_{i=1}^{\infty} L^i$$

  Podľa týchto definícií má každý jazyk tieto dve vlastnosti:
  1. $L^+ = LL^* = L^*L$
  2. $L^* = L^* \cup \{\varepsilon\}$

2.3 Gramatiky

Triviálny spôsob reprezentácie jazyka, akým je vymenovanie všetkých viet jazyka, je nepoužiteľný pre reprezentáciu nekonečného jazyka. Dokonca je obtiažne reprezentovať týmto spôsobom rozsiahle konečné jazyky, preto využívame formálne prostriedky pre reprezentáciu jazykov.

Najznámejším prostriedkom pre reprezentáciu jazyka je gramatika. Gramatiky zohrávajú hlavnú úlohu v teórii formálnych jazykov a spĺňajú základnú požiadavku kladenú

na reprezentáciu konečných aj nekonečných jazykov, t.j. požiadavka konečnosti reprezentácie.

Gramatika využíva dve konečné abecedy, ktoré sú navzájom disjunktné:

1. množina N - *neterminálnych symbolov*, skrátene *non-terminály*, sú pomocné premenné, ktoré označujú určité syntaktické celky.

2. množina T - *terminálnych symbolov*, skrátene *terminály*. Množina *terminálov* je identická s abecedou, nad ktorou je definovaný jazyk.

Gramatika je zariadenie, ktoré generuje jazyk. Z *neterminálneho symbolu* aplikáciou prepisovacieho pravidla generuje reťazce – vetné formy, ktoré sú tvorené *terminálnymi* a *neterminálnymi symbolmi*. Vetné formy, ktoré obsahujú len terminály, reprezentujú vety gramatikou definovaného jazyka. Ak gramatika negeneruje žiadnu vetu, reprezentuje prázdny jazyk.

Jadrom gramatiky je konečná množina *prepisovacích pravidiel* (skrátene *pravidlá*). Podľa tvaru týchto pravidiel klasifikujeme gramatiky do jednotlivých tried.

Pre pochopenie nasledujúceho textu si uvedieme *Chomského klasifikáciu gramatík a jazykov*, ktorá rozdeľuje formálne gramatiky a nimi definované jazyky na štyri základné typy gramatík, podľa ich vyjadrovacej schopnosti. Tieto typy gramatík sa označujú ako typ 0, typ 1, typ 2 a typ 3.

Definícia týchto gramatík má rovnaký základ. Odlišuje sa iba v tvare prepisovacích pravidiel.

Definícia 2.3.1. Frázová gramatika je štvorica $G = (N, T, P, S)$, kde

- N je abeceda *non-terminálov*,
- T je abeceda *terminálov*, kde $N \cap T = \oslash$,
- P je konečná množina *pravidiel*,
- $S \in N$ je počiatočný symbol gramatiky.

Pravidlá sú usporiadané dvojice $(u, v) \in P$, ktoré sa nazývajú *prepisovacie pravidlá* alebo *produkcie* a majú tvar $u \to v$. Pravidlo určuje možnú substitúciu reťazca u za reťazec v. Reťazec u obsahuje vždy aspoň jeden *non-terminál*. Reťazec v je prvkom množiny $(N \cup T)^*$. Formálne:

$$P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$$

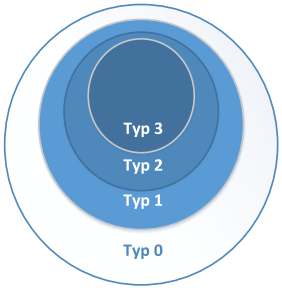
Ak v je prázdny reťazec, teda $v = \varepsilon$, toto pravidlo nazývame *ε-pravidlo*.
Množina $V = N \cup T$ je *úplná abeceda* gramatiky G.

Chomského klasifikácia gramatík

Chomského klasifikácia gramatík je taktiež známa pod názvom Chomského hierarchia jazykov. Táto hierarchia rozdeľuje formálne jazyky a gramatiky na štyri základné typy.

Hierarchickosť gramatík spočíva v tom, že gramatiky vyššieho typu je možné klasifikovať akýmkoľvek nižším typom. Napríklad gramatika typu 2 je zároveň gramatikou typu 1 a 0. Chomského hierarchia je zobrazená na obrázku 2.1.

Ako už bolo spomenuté všetky triedy gramatík majú rovnaký základ. Uvažujme teda gramatiku G z definície 2.3.1. Potom prepisovacie pravidlá z množiny prepisovacích pravidiel P budú vyzerať nasledovne:



Obr. 2.1: Chomského hierarchia

Typ 0 - Frázová gramatika

Gramatika typu 0 obsahuje pravidlá v najvšeobecnejšom tvare, zhodným s vyššie uvedenou definíciou 2.3.1 frázovej gramatiky:

$$u \to v, \text{ kde}$$
$$u \in (N \cup T)^* N (N \cup T)^*, v \in (N \cup T)^*$$

Nazýva sa taktiež aj *neobmedzená*. Generuje jazyky akceptovateľné Turingovým strojom[1], nazývané ako *rekurzívne vyčísliteľné jazyky*. Rodina týchto jazykov je označovaná ako **RE** (Recursively enumerable).

Typ 1 - Kontextová gramatika

Gramatika typu 1 obsahuje pravidlá tvaru:

$$u \to v, \text{ kde}$$
$$u = x_1 A x_2, v = x_1 y x_2 \text{ a } A \in N, x_1, x_2 \in (N \cup T)^*, y \in (N \cup T)^+$$

Pravidlá kontextovej gramatiky sú obmedzené dĺžkou reťazca. Musí platiť, že dĺžka pravidla na pravej strane je rovná alebo väčšia ako dĺžka pravidla na ľavej strane.

Výnimku tvorí pravidlo:

$S \to \varepsilon$, pokiaľ sa S nevyskytuje na pravej strane žiadneho pravidla.

Táto gramatika generuje *kontextové jazyky*, ktoré akceptuje lineárne ohraničený konečný automat[2]. Rodina *kontextových jazykov* je označovaná ako **CS** (context-sensitive).

Typ 2 - Bezkontextová gramatika

Gramatika typu 2 obsahuje pravidlá tvaru:

$$A \to x, \text{ kde}$$
$$A \in N, x \in (N \cup T)^*$$

Výnimku tvorí opäť prepisovacie pravidlo:

$S \to \varepsilon$, pokiaľ sa S nevyskytuje na pravej strane žiadneho pravidla.

Na ľavej strane pravidla je povolený iba jeden neterminálny symbol. Touto gramatikou sú generované *bezkontextové jazyky*, ktoré sú akceptovateľné zásobníkovým automatom. Definícia tohto typu automatu sa nachádza v časti 2.4.5. Rodina *bezkontextových jazykov* je označovaná ako **CF** (context-free).

[1] Turingov stroj je zložený z konečného automatu, pravidiel prechodovej funkcie a teoreticky nekonečnej pásky.

[2] Lineárne ohraničený konečný automat je obmedzená verzia Turingovho stroja s konečnou páskou, ktorej dĺžka je lineárne závislá na dĺžke vstupného reťazca.

Typ 3 - Regulárna gramatika

Definícia 2.3.2. Gramatika typu 3 obsahuje pravidlá tvaru:

$$A \to aB \text{ alebo } A \to a, \text{ kde}$$
$$A, B \in N, a \in T$$

Ako v prípade gramatík typu 1 a 2 výnimku tvorí prepisovacie pravidlo:

$S \to \varepsilon$, pokiaľ sa S nevyskytuje na pravej strane žiadneho pravidla.

Regulárna gramatika generuje *regulárne jazyky*. Tieto jazyky akceptuje konečný automat z definície 2.4.1. Rodina *regulárnych jazykov* je označovaná ako **REG** (regular).

Alternatívne je rodina regulárnych jazykov charakterizovaná *pravou lineárnou gramatikou*. Prepisovacie pravidlá tejto gramatiky majú tvar:

$$A \to xB \text{ alebo } A \to x, \text{ kde}$$
$$A, B \in N, x \in T^*$$

Pravá lineárna gramatika generuje *pravé lineárne jazyky*. Rodina týchto jazykov je označovaná ako **RLIN** (right-linear).

V praxi sa používajú hlavne gramatiky typu 2 - bezkontextové a typu 3 - regulárne. Do skupiny bezkontextových jazykov patria programovacie jazyky, pre ktoré v každom prekladači existuje syntaktický analyzátor.

Okrem týchto základných typov gramatík existujú aj iné. Všetky gramatiky, ktoré generujú rekurzívne vyčísliteľné jazyky, je možné zaradiť do Chomského hierarchie. Týmto zaradením získame odhad ich sily vzhľadom na tieto štyri typy gramatík.

Pre pochopenie nasledujúceho textu si uvedieme *Lineárnu gramatiku*, ktorú môžeme zaradiť v Chomského hierarchii medzi bezkontextovú a regulárnu gramatiku.

Lineárna gramatika

Definícia 2.3.3. Uvažujme gramatiku $G = (N, T, P, S)$ z definicie 2.3.1. Gramatika G je lineárna, ak každé pravidlo v množine P má nasledujúci tvar:

$$A \to xBy \text{ alebo } A \to x, \text{ kde}$$
$$A, B \in N \text{ a } x, y \in T^*$$

Touto gramatikou sú generované *lineárne jazyky*, ktoré sú akceptovateľné zásobníkovým automatom. Rodina *lineárnych jazykov* je označovaná ako **LIN** (linear).

Podľa Chomského hierarchie platí nasledujúci vzťah jazykov, ktoré sú generované gramatikami uvedenými v tejto sekcii:

$$REG = RLIN \subset LIN \subset CF \subset CS \subset RE$$

2.4 Automaty

Konečný automat

V tejto sekcii si predstavíme jednoduché zariadenie, ktoré slúži na rozpoznávanie reťazcov, daných jazykom. V informatike sa najviac využívajú regulárne a bezkontextové jazyky. V predchádzajúcej kapitole 2.3, na strane 6, sme spomenuli, že tieto jazyky prijíma *konečný automat* (REG) a *zásobníkový automat* (CF).

Predstavíme si najjednoduchší model založený na konečnej množine stavov a výpočetných pravidiel. Tento model číta dané slovo nad vstupnou abecedou zľava doprava, symbol po symbole. Na začiatku sa nachádza v počiatočnom stave.

Definícia 2.4.1. *Konečný automat* je pätica $M = (Q, \Sigma, R, s, F)$, kde

- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- $R \subseteq Q \times \Sigma^* \times Q$ je *konečná množina pravidiel*,
- $s \in Q$ je *počiatočný stav*,
- $F \subseteq Q$ je množina *koncových stavov*.

Namiesto relačného zápisu $(p, y, q) \in R$, zapisujeme pravidlá $py \to q \in R$. Ak R implikuje, že $y \neq \varepsilon$, tak tento automat je bez ε-prechodov.

Pravidlo $py \to q \in R$ znamená, že pri prečítaní vstupného symbolu y automat M urobí *prechod* z p do q. Ak $y = \varepsilon$, tak sa zo vstupnej pásky neprečíta žiadny symbol.

Definícia 2.4.2.

Konfigurácia konečného automatu M, z definície 2.4.1, je reťazec $\chi \in Q\Sigma^*$. Nech $\chi_1 = pax$ a $\chi_2 = qx$ sú dve konfigurácie konečného automatu M, kde $p, q \in Q$,a$\in \Sigma \cup \{\varepsilon\}$ a $x \in \Sigma^*$. Nech $r = pa \to q \in R$ je pravidlo. Potom M môže previesť *prechod* z χ_1 do χ_2 aplikáciou pravidla r. Tento prechod zapíšeme:

$$pax \vdash qx[r] \text{ alebo } \chi_1 \vdash \chi_2[r]$$

Definícia 2.4.3. Nech $\chi_0, \chi_1, ...\chi_n$, je sekvencia prechodov konfigurácií pre $n \geq 1$ a $\chi_{i-1} \vdash \chi_i[r_i], r_i \in R$ pre všetky $i = 1, ..., n$. Čo znamená:

$$\chi_0 \vdash \chi_1[r_1] \vdash \chi_2[r_2]... \vdash \chi_n[r_n]$$

Potom M prevedie *n-prechodov* z χ_0 do χ_n, zapisujeme.

$$\chi_0 \vdash^n \chi_n[r_n...r_n] \text{ alebo } \chi_0 \vdash^n \chi_n$$

Ak $\chi_0 \vdash^n \chi_n[\rho]$ pre nejaké $n \geq 1$, potom:

$$\chi_0 \vdash^+ \chi_n[\rho]$$

Ak $\chi_0 \vdash^n \chi_n[\rho]$ pre nejaké $n \geq 0$, potom:

$$\chi_0 \vdash^* \chi_n[\rho]$$

Definícia 2.4.4. Majme konečný automat M z definície 2.4.1, potom *jazyk prijímaný* konečným automatom $M, L(M)$, je definovaný ako:

$$L(M) = \{w : w \in \Sigma^*, sw \vdash^* f, f \in F\}$$

Teda konečný automat M prijíma reťazec w, ak je prečítaný pomocou sekvencie prechodov a automat skončí v koncovom stave.

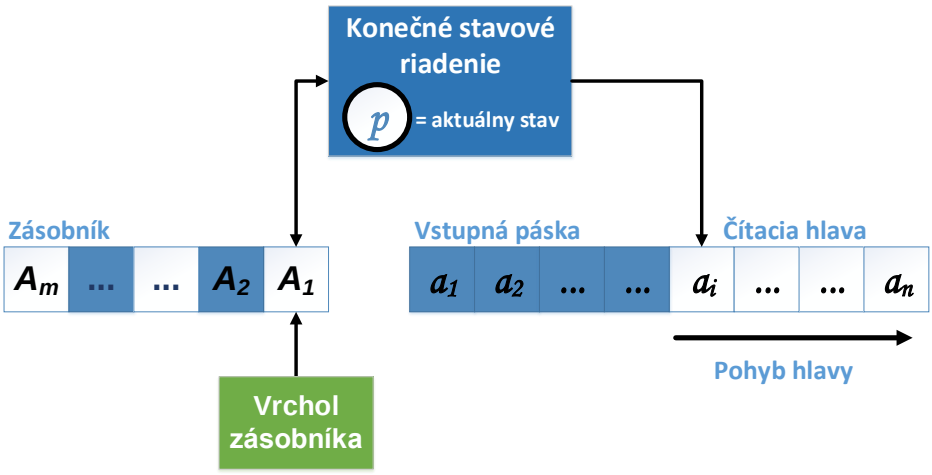
Zásobníkový automat

Zásobníkový automat je konečný automat 2.4.1, ktorý je rozšírený o zásobník. Tento automat prijíma rodinu jazykov **CF**, teda bezkontextové jazyky.

Činnosť zásobníkového automatu spočíva v tom, že ak sa na vrchole zásobníka objaví vstupný symbol a, automat porovná symbol na vrchole zásobníka s aktuálnym vstupným symbolom. Ak sa zhodujú, vyjme symbol z vrcholu zásobníka a pokračuje v spracovávaní nasledujúceho symbolu na vstupnej páske. Ak sa na vrchole zásobníka nachádza neterminálny symbol, automat expanduje tento symbol podľa daného pravidla.

Automat prijme, prečíta vstupný reťazec w, ak je schopný vykonať sekvenciu pohybov, vyprázdniť zásobník a skončiť vo finálnom stave. Požiadavka skončiť vo finálnom stave môže byť vynechaná a je postačujúce, že po prečítaní reťazca bude zásobník prázdny.

Operáciu vyňatia symbolu, ktorý sa nachádza na vrchole zásobníka, budeme ďalej nazývať *pop* a operáciu vkladania symbolov na zásobník budeme nazývať *push*.



Obr. 2.2: Zásobníkový automat

Definícia 2.4.5. *Zásobníkový automat* je sedmica $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- Γ je *abeceda zásobníkových symbolov*,
- R je *konečná množina pravidiel* tvaru: $Apa \to wq$, kde

$$A \in \Gamma, p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, w \in \Gamma^*,$$

- $s \in Q$ je *počiatočný stav*,
- $S \in \Gamma$ je *počiatočný zásobníkový symbol*,
- $F \subseteq Q$ je *množina koncových stavov*.

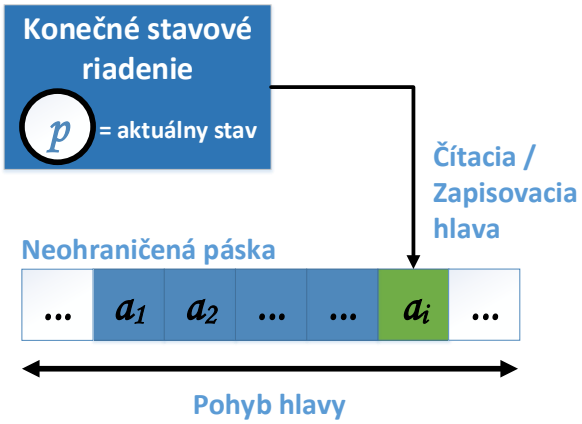
Definícia 2.4.6. *Konfigurácia* automatu M, χ, je každý reťazec z $\Gamma^* Q \Sigma^*$.
Pre každé $x\gamma pay, xwqy \in \Gamma^* Q \Sigma^*$ automat M vykoná *prechod* z konfigurácie $x\gamma pay$ do konfigurácie $xwqy$ podľa pravidla $\gamma pa \to wq \in R$. Tento prechod zapíšeme ako

$$x\gamma pay \vdash_M xwqy[\gamma pa \to wq]$$

Turingov stroj

Turingov stroj sa skladá z konečnej stavovej riadiacej jednotky, neohraničenej pásky a čítacej/zapisovacej hlavy. Komponenty Turingovho stroja je možné vidieť na obrázku 2.3.

Turingov stroj počas jedného kroku prečíta symbol, ktorý sa nachádza pod jeho hlavou. Následne, v závislosti na prečítanom symbole a stave riadiacej jednotky, môže čítaný symbol prepísať, zmeniť stav a posunúť hlavu o jedno pole doprava alebo doľava podľa prechodovej funkcie. Výpočet tvorí postupnosť výpočetných krokov, vychádzajúca z počiatočnej konfigurácie stroja, t.j. stavu riadiacej jednotky, obsahu pásky a pozície hlavy. Definícia je prebraná zo zdroja [1].



Obr. 2.3: Turingov stroj

Definícia 2.4.7. *Turingov stroj* je šestica $M = (Q, \Sigma, \Gamma, R, s, f)$, kde

- Q je *konečná množina vnútorných (riadiacich) stavov*,
- Σ je *vstupná abeceda*, $\triangle \notin \Sigma$,
- Γ je *pásková abeceda*, $\Sigma \subset \Gamma, \triangle \in \Gamma$,
- δ je *parciálna prechodová funkcia*, $\delta : (Q \setminus \{f\}) \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$, kde $L, R \notin \Gamma$,
- $q \in Q$ je *počiatočný stav*,
- $f \in Q$ je *koncový stav*.

- $\triangle$ je symbol, ktorý značí tzv. *blank* - prázdny symbol, ktorý sa vyskytuje na miestach pásky, ktorá ešte nebola použitá.

Definícia 2.4.8. *Konfigurácia* Turingovho stroja T je daná stavovým riadením a konfiguráciou pásky, formálne $Q \times \{\gamma \triangle^{\omega} \mid \gamma \in \Gamma^*\} \times \mathbb{N}$.

Definícia 2.4.9. Pre ľubovoľný reťazec $\gamma \in \Gamma^{\omega}$ a číslo $n \in \mathbb{N}$ označíme γ_n n-tý symbol daného reťazca a označíme $s_b^n(\gamma)$ reťazec, ktorý vznikne z γ zámenou γ_n za b. *Krok výpočtu Turingovho stroja* M definujeme ako *najmenšiu* reláciu $\vdash_M$ takú, že

$$\forall q_1, q_2 \in Q \forall \gamma \in \Gamma^{\omega} \forall n \in \mathbb{N} \forall b \in \Gamma \text{ platí}$$

- $(q_1, \gamma, n) \vdash_M (q_2, \gamma, n+1)$ pre $\delta(g_1, \gamma_n) = (g_2, R)$ je operácia *posunu doprava* pri γ_n pod hlavou,
- $(q_1, \gamma, n) \vdash_M (q_2, \gamma, n-1)$ pre $\delta(g_1, \gamma_n) = (g_2, L)$ a $n > 0$ je operácia *posunu doľava* pri γ_n pod hlavou a
- $(q_1, \gamma, n) \vdash_M (q_2, s_b^n(\gamma), n1)$ pre $\delta(g_1, \gamma_n) = (g_2, b)$ je operácia *zápisu* b pri γ_n pod hlavou.

Kapitola 3

Modifikované zásobníkové automaty

V nasledujúcej kapitole budú predstavené niektoré z existujúcich modifikácií zásobníkových automatov. Dôvodov, prečo skúmať modifikácie zásobníkových automatov, je mnoho. Najzaujímavejším je však výpočetná sila automatu. Z predchádzajúcej kapitoly vieme, že klasické zásobníkové automaty prijímajú rodinu bezkontextových jazykov. V nasledujúcom texte sa dozvieme, že existujú modifikácie, ktoré sú schopné prijať rodinu rekurzívne vyčísliteľných jazykov. To znamená, že sú tak mocné ako Turingov stroj.

Na úvod si uvedieme zásobníkové automaty, ktoré regulujú aplikáciu pravidla riadiacim jazykom. Ďalej budú nasledovať automaty, ktoré sú nejakým spôsobom obmedzené, napríklad počtom *otáčok* na zásobníku.

Ku koncu si naopak predstavíme verziu automatu, ktorá má špeciálny hlboký zásobník. To znamená, že môžeme siahať hlbšie do zásobníku, teda nemusíme pracovať iba so symbolom na vrchole zásobníka.

Na týchto predstavených automatoch budú neskôr založené, nami zavedené, nové modifikácie zásobníkových automatov.

3.1 Regulované zásobníkové automaty

V tejto časti si definujeme zásobníkové automaty, ktoré regulujú aplikáciu pravidla riadiacim jazykom. Tieto automaty boli prvý krát predstavené v roku 2000, v článku *Regulated Pushdown Automata* [2]. V článku je taktiež k dispozícii dôkaz, že táto regulácia zásobníkových automatov nemá žiadny efekt na výpočetnú silu zásobníkového automatu, ak je riadiaci jazyk regulárny.

V prípade, ak sú zásobníkové automaty regulované lineárnym riadiacim jazykom, majú väčšiu silu, čo je dokázané v publikácii [8] v sekcii **16.2.3**. Zásobníkové automaty regulované lineárnym jazykom charakterizujú rodinu rekurzívne vyčísliteľných jazykov.

Ďalej sa zameriame konkrétne na automaty regulované lineárnym riadiacim jazykom.

Definícia 3.1.1. Nech $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat z definície 2.4.5 a Ψ je abeceda identifikátorov jeho pravidiel. Každé pravidlo $Apa \to wq$ označíme jedinečným identifikátorom $\rho \in \Psi$ ako $\rho.Apa \to wq$. Potom Ξ je *riadiaci jazyk* nad abecedou Ψ, teda $\Xi \subseteq \Psi^*$.

Konfigurácia automatu M, χ, je každý reťazec z $\Gamma^* Q \Sigma^*$. Pre každé $x \in \Gamma^*, y \in \Sigma^*$ a $\rho.Apa \to wq \in R$ automat M vykoná prechod z konfigurácie $xApay$ do konfigurácie $xwqy$

podľa pravidla ρ. Tento prechod zapíšeme ako

$$xApay \Rightarrow xwqy[\rho]$$

Definícia 3.1.2. Nech $\chi_0, \chi_1, ...\chi_n$, je sekvencia konfigurácií pre $n \geq 1$ a $\chi_{i-1} \Rightarrow \chi_i[\rho_i]$, kde $\rho_i \in \Psi$ pre všetky $i = 1, ..., n.$, potom sa automat M n-tými prechodmi presunie z χ_0 do χ_n podľa $[\rho_1...\rho_n]$. Zapíšeme ako

$$\chi_0 \Rightarrow^n \chi_n[\rho_1...\rho_n]$$

Automat M prijme vstupný reťazec x iba v prípade, ak riadiaci jazyk Ξ obsahuje riadiaci reťazec, podľa ktorého automat M vykonal sekvenciu krokov, teda po prečítaní reťazca x prešiel do konečnej konfigurácie.

Definícia 3.1.3. Pomocou dvojice (M, Ξ) [1] definujeme nasledujúce akceptovateľné jazyky

- $L(M, \Xi, 1)$ - *jazyk akceptovaný finálnym stavom*
- $L(M, \Xi, 2)$ - *jazyk akceptovaný prázdnym zásobníkom*
- $L(M, \Xi, 3)$ - *jazyk akceptovaný finálnym stavom a prázdnym zásobníkom*

Nech $\chi \in \Gamma^* Q \Sigma^*$. Ak $\chi \in \Gamma^* F$, $\chi \in Q$, $\chi \in F$, potom χ je *1 - finálna konfigurácia*, *2 - finálna konfigurácia*, *3 - finálna konfigurácia.*

Pre $i = 1, 2, 3$ definujeme

$$L(M, \Xi, i) = \{w | w \in \Sigma^* \text{ a } Ssw \vdash_M^* \chi[\sigma] \text{ pre každú } i \text{ -finálnu konfiguráciu } \chi \text{ a } \sigma \in \Xi\}$$

Pre každú rodinu jazykov $\mathscr{L}$ a $i \in \{1, 2, 3\}$, definujeme **RPDA**[2]

$$(\mathscr{L}, i) = \{L | L = L(M, \Xi, i), \text{kde } M \text{ je zásobníkový automat a } \Xi \in \mathscr{L}\}.$$

Z dôkazov, ktoré sa nachádzajú v publikácii [8] (na stranách 548 - 558) vieme, že *bezkontextový jazyk*(**CF**) je akceptovateľný regulovaným zásobníkovým automat, ktorý riadi *regulárny* jazyk. A rodinu *rekurzívne vyčísliteľných jazykov* (**RE**) akceptuje regulovaný zásobníkový automat, ktorý riadi *lineárny jazyk.* Teda platí

$$CF = RPDA(REG, 1) = RPDA(REG, 2) = RPDA(REG, 3) \text{ a}$$
$$RE = RPDA(LIN, 1) = RPDA(LIN, 2) = RPDA(LIN, 3)$$

Príklad 3.1.1. Ukážeme si jednoduchý príklad pre demonštráciu rozdielu automatu, ktorý nie je regulovaný a automatu, ktorý je regulovaný riadiacim jazykom.

Uvažujme regulovaný zásobníkový automat M z definície 2.4.5 s nasledujúcimi pravidlami:

1. $Ssa \rightarrow Sas$
2. $asa \rightarrow aas$
3. $asb \rightarrow q$
4. $aqb \rightarrow q$
5. $Sqc \rightarrow Sq$
6. $Sqc \rightarrow f$

[1] Dvojicu (M, Ξ) nazývame riadený zásobníkový automat.

[2] Regulovaný zásobníkový automat budeme označovať *RPDA*, čo je odvodená skratka z jeho anglického názvu - Regulated Pushdown Automata.

Tento zásobníkový automat prijíma reťazec *aabbccc*:

$$\begin{aligned} Ssaabbccc &\Rightarrow Sasabbccc \quad [1] \\ &\Rightarrow Saasbbccc \quad [2] \\ &\Rightarrow Saqbccc \quad [3] \\ &\Rightarrow Sqccc \quad [4] \\ &\Rightarrow Sqcc \quad [5] \\ &\Rightarrow Sqc \quad [5] \\ &\Rightarrow f \quad [6] \end{aligned}$$

Postupnosť pravidiel: **1234556**.
Automat M akceptuje jazyk $L(G) = \{a^n b^n c^m : n, m \geq 1\}$.

Prijatie reťazca *aabbcc* zásobníkovým automatom M regulovaným riadiacim jazykom $\Xi = \{12^m 34^n 5^n 6 : m, n \geq 0\}$:

$$\begin{aligned} Ssaabbcc &\Rightarrow Sasabbcc \quad [1] \\ &\Rightarrow Saasbbcc \quad [2] \\ &\Rightarrow Saqbcc \quad [3] \\ &\Rightarrow Sqcc \quad [4] \\ &\Rightarrow Sqc \quad [5] \\ &\Rightarrow f \quad [6] \end{aligned}$$

Postupnosť pravidiel: **123456** patrí riadiacemu jazyku Ξ.
Automat M regulovaný riadiacim jazykom Ξ generuje jazyk $L(G, \Xi) = a^n b^n c^n : n \geq 1$.

Napriek tomu, že boli využité rovnaké pravidlá pre akceptovanie reťazca automatom M, je z uvedených príkladov zrejmé, že reťazec *aabbccc* nie je možné akceptovať automatom M regulovaným riadiacim jazykom z vyššie uvedeného príkladu, pretože postupnosť pravidiel $1234556 \notin \Xi = \{1\}\{24\}^*\{35\}$.

Medzi špeciálne modifikácie týchto typov automatov patria napríklad *jedno-otáčkové regulované zásobníkové automaty* predstavené v článku [7], ktoré majú rovnakú výpočetnú silu, avšak jednoduchší mechanizmus. V článku je dokázané, že jedno-otáčkové regulované zásobníkové automaty charakterizujú rodinu rekurzívne vyčísliteľných jazykov **RE** uvedených v definícii 2.3.

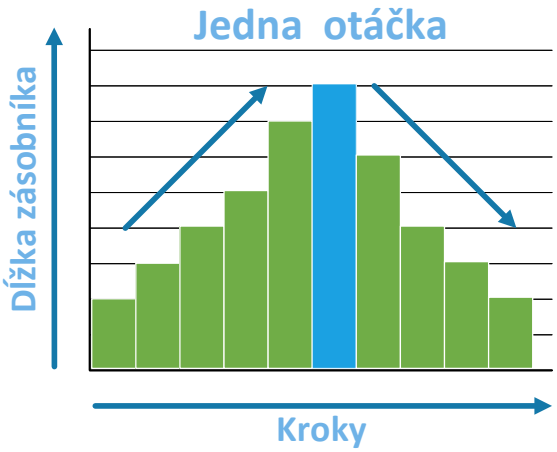
3.2 Konečné otáčkové zásobníkové automaty

Zamyslime sa nad prácou zásobníkového automatu. Na zásobník môžeme symboly pridávať a symboly odoberať. Dĺžka zásobníka sa zväčšuje, alternatívne zmenšuje, počas ktoréhokoľvek kroku výpočtu.

Existujú automaty, ktoré majú obmedzujúce pravidlo. A to, koľko krát sa môže dĺžka zásobníka zmenšiť a potom zväčšiť. Uvažujme dva po sebe nasledujúce kroky, ktoré vykoná zásobníkový automat M z definície 2.4.5. Ak počas prvého kroku M neskráti dĺžku svojho zásobníka a počas druhého kroku skráti, tak potom M vykoná „otáčku“ počas druhého kroku.

Inými slovami, ak po pridaní symbolu na zásobník aplikujeme operáciu *pop* vykonáme *„otáčku“*. Na obrázku 3.1 je zobrazená jedna otáčka.

Majme zásobníkový automat M z definície 2.4.5. Automat M je *k-otáčkový*, ak počas výpočtu vykoná maximálne *k-otáčok*. Potom *Konečný-otáčkový* zásobníkový automat je *k*-otáčkový pre $k \in \mathbb{N}^+$. Jazyk je konečno-otáčkový pre každé $k \in \mathbb{N}$ platí $Fturn_k \subset Fturn_{k+1}$.



Obr. 3.1: Znázornenie jednej otáčky zásobníku. Dĺžka zásobníka sa zväčšuje pri operácii *push* a naopak zmenšuje pri operácii *pop*.

Tento automat prijíma rodinu jazykov $Fturn(k)$. *K-otáčkový zásobníkový automat* odmietne slová, ktoré nepatria do jazyka prijímaného automatom, rýchlejšie ako klasický zásobníkový automat. Viac informácií o týchto automatoch je možné nájsť v knihe [10].

3.2.1 Jedno-otáčkový atomický zásobníkový automat

Zásobníkový automat je „jedno-otáčkový“ v prípade, ak neurobí viac „otáčkových“ pohybov v priebehu ktoréhokoľvek výpočtu od počiatočnej konfigurácie. V článku [8] je dokázané, že jedno-otáčkové regulované zásobníkové automaty charakterizujú rodinu rekurzívne vyčísliteľných jazykov **RE** uvedených v definícii 2.3. Jedno-otáčkové regulované zásobníkové automaty sú rovnako mocné ako regulované zásobníkové automaty, ktoré môžu urobiť ľubovoľný počet „otáčok“, avšak ich mechanizmus je jednoduchší.

Informácie použité v tejto časti sú čerpané zo zdroja [8].

Definícia 3.2.1. *Atomický zásobníkový automat* je sedmica $M = (Q, \Sigma, \Omega, R, s, \$, F)$, kde

- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- Ω je *abeceda zásobníkových symbolov*, Q, Σ a Ω sú v páre disjunktné, t.j. ich prienik je prázdna množina,
- $\$$ je symbol, ktorý značí *dno zásobníka*, kde $\$ \notin Q \cup \Sigma \cup \Omega$,
- $F \subseteq Q$ je *množina koncových stavov*,
- R je *konečná množina pravidiel* tvaru: $Apa \to wq$, kde $p, q \in Q, A, w \in \Omega \cup \{\varepsilon\}, a \in \Sigma \cup \{\varepsilon\}$, tak že $|Aaw| = 1$. To znamená, že R je konečná množina pravidiel, kde každé pravidlo má niektorý z nasledujúcich tvarov:
  1. $Ap \to q$ (pravidlo vkladania symbolu na zásobník)
  2. $p \to wq$ (pravidlo pre odstránenie symbolu zo zásobníka)

3. $pa \to q$ (pravidlo čítania vstupného symbolu)

Nech každé pravidlo má identifikátor ρ, potom Ψ je abeceda týchto identifikátorov a ψ mapuje pravidlo, $Apa \to wq \in R$ do ρ. Formálny zápis: $\rho.Apa \to wq \in R$. Inými slovami $\rho.Apa \to wq$ znamená $\psi(Apa \to wq) = \rho$. Konfigurácia automatu M, χ je ľubovoľný reťazec z $\{\$\}\Omega^* Q \Sigma^*$.

χ je počiatočná konfigurácia, ak $\chi = \$sw$, kde $w \in \Sigma^*$. Pre každé $x \in \Omega^*$, $y \in \Sigma^*$ a $\rho.Apa \to wq \in R$, M urobí pohyb z konfigurácie $\$xApay$ do konfigurácie $\$xwqy$ podľa ρ. Toto formálne zapíšeme ako $\$Apay \Rightarrow \$xwqy[\rho]$.

Nech χ je ľubovoľná konfigurácia automatu M. M urobí nula pohybov z χ do χ podľa ε, symbolicky zapísané $\chi \Rightarrow^0 \chi[\varepsilon]$. Nech existuje sekvencia konfigurácií $\chi_0, \chi_1, ..., \chi_n$ pre $n \geq 1$, potom $\chi_i - 1 \Rightarrow \chi_i[i]$, kde $\rho_i \in \Psi$, pre $i = 1, ..., n$, potom M urobí n pohybov z χ_0 do χ_n podľa pravidiel $\rho_1 ... \rho_n$, symbolicky zapísané ako $\chi_0 \Rightarrow^n \chi_n[\rho_1 ... \rho_n]$, zjednodušene $\chi_0 \Rightarrow \chi_n$. $\Rightarrow^*$ a $\Rightarrow^+$ je definované štandardným spôsobom.

Majme $x, x', x'' \in \Omega^*$, $y, y', y'' \in \Sigma^*$, a $\$xq \Rightarrow \$x'q'y' \Rightarrow \$x''q''y''$. Ak $|x| \leq |x'|$ a $|x'| > |x''|$, potom $\$x'q'y' \Rightarrow \$x''q''y''$ je *otáčka*. Ak M neurobí viac ako jednu otáčku počas ktorejkoľvek sekvencie pohybov štartujúcej z počiatočnej konfigurácie, potom automat M nazývame **jedno-otáčkový**.

Počas pohybu *atomický* regulovaný zásobníkový automat zmení stav a vykoná iba jednu z nasledujúcich akcií:

1. vloží symbol na zásobník, vykoná operáciu *push*
2. vyjme symbol zo zásobníka, vykoná operáciu *pop*
3. číta symbol zo vstupnej pásky

3.3 Hlboké zásobníkové automaty

V nasledujúcej sekcii je predstavený hlboký zásobníkový automat, ktorý je prirodzenou modifikáciou bežného zásobníkového automatu a bol prvý krát zadefinovaný v článku [6].

Hlboký zásobníkový automat a zásobníkový automat pracujú takmer rovnako. Zásobníková abeceda hlbokého zásobníkového automatu je rozdelená na terminálne a neterminálne symboly. Bežný zásobníkový automat môže vybrať, čítať a modifikovať len položku na vrchole zásobníka, zatiaľ čo hlboký zásobníkový automat môže čítať, poprípade nahradiť neterminálne symboly umiestnené hlbšie v zásobníku. V prípade, ak sa na vrchole zásobníka nachádza terminálny symbol, ktorý sa zhoduje s aktuálnym vstupom, je tento symbol zo zásobníka odstránený – aplikovaná klasická operácia *pop*.

Hlboké zásobníkové automaty sú silnejšie ako klasické zásobníkové automaty, avšak nedosahujú sily ako automaty, ktoré prijímajú kontextové jazyky. Hlboké zásobníkové automaty prijímajú jazyky, ktoré generuje *regulovaná bezkontextová gramatika* bez mazacích (ϵ) pravidiel.

Tento typ automatu bol prvý krát predstavený v článku [6], kde je možné nájsť podrobnejšie informácie a formálne dôkazy.

Definícia 3.3.1. *Hlboký zásobníkový automat* je sedmica $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,

- Γ je *abeceda zásobníkových symbolov*, kde $\mathbb{N} \cap Q \cap \Gamma = \emptyset$, $\Sigma \subseteq \Gamma$ a $\# \in \Gamma - \Sigma$. Špeciálny symbol $\#$ značí dno zásobníka,
- $R \subseteq (\mathbb{N} \times Q \times (\Gamma - (\Sigma \cup \{\#\})) \times Q \times (\Gamma - \{\#\})^+)$
  $\cup (\mathbb{N} \times Q \times \{\#\} \times Q \times (\Gamma - \{\#\})^* \{\#\})$
  Pravidlo $(m, q, A, p, v) \in R$ je možné zapísať v tvare $mqA \to pv$,
- $s \in Q$ je *počiatočný stav*,
- $S \in \Gamma$ je *počiatočný zásobníkový symbol*,
- $F \subseteq Q$ je *množina koncových stavov*.

Definícia 3.3.2. *Konfigurácia* automatu M, χ, je každý reťazec z množiny

$$Q \times \Sigma^* \times (\Gamma - \{\#\})^* \{\#\}.$$

Definícia 3.3.3. Majme dve konfigurácie $x, y \in \chi$. Automat M prečíta symbol na vstupnej páske a prejde z konfigurácie x do y, vykoná *pop* operáciu na zásobníku. Tento prechod zapíšeme ako

$$x_p \Rightarrow y$$

Definícia 3.3.4. Ak $x = (q, au, az)$, $y = (q, u, z)$, kde $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$, $q \in Q$. Automat M *expanduje* svoj zásobník z konfigurácie x do y, tento prechod zapíšeme ako

$$x_e \Rightarrow y$$

Ak $x = (q, w, uAz)$, $y = (p, w, uvz)$, $mqA \to pv \in R$, kde q, $p \in Q$, $w \in \Sigma^*$, $A \in \Gamma$, u, v, $z \in \Gamma^*$ a platí $\text{occur}(u, \Gamma - \Sigma) = m - 1$. Automat M vykoná prechod $x_e \Rightarrow y$ podľa pravidla $mqA \to pv \in R$, zapíšeme ako

$$x_e \Rightarrow y[mqA \to pv]$$

Pravidlo $mqA \to pv$ je *pravidlo hĺbky* m. Podľa toho $x_e \Rightarrow y[mqA \to pv]$ je *rozšírenie v hĺbke* m.

Majme $n \in \mathbb{N}^+$. Ak každé pravidlo automatu M má hĺbku n alebo menej, potom hovoríme, že automat je *hĺbky* n, čo zapíšeme ako ${}_nM$. Je dôležité poznamenať, že hĺbka pravidla pre expanziu sa počíta len pre neterminálne symboly.

Definícia 3.3.5. Majme automat M hĺbky n, pre $n \in \mathbb{N}$. Definujeme jazyk prijímaný automatom ${}_nM$, $L({}_nM)$

$$L({}_nM) = \{w \in \Sigma^* \mid (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#), \text{ kde } f \in F\}$$

Ďalej definujeme jazyk, ktorý automat ${}_nM$ prijíma prázdnym zásobníkom, $E({}_nM)$

$$E({}_nM) = \{w \in \Sigma^* \mid (s, w, S\#) \Rightarrow^* (q, \varepsilon, \#), \text{ kde } q \in Q\}$$

Determinizmus

Pre praktické využitie hlbokých zásobníkových automatov sa budeme zaoberať ich deterministickou verziou. Preto si definujeme podľa zdroja [6] *striktný determinizmus* a slabšiu formu determinizmu - *determinizmus s ohľadom na hĺbku*.

Definícia 3.3.6. Majme automat ${}_nM = (Q, \Sigma, \Gamma, R, s, S, F)$, $mqA \to pv \in R$. Pre *striktný determinizmus* hlbokého zásobníkového automatu platí, že automat môže pre každý stav použiť maximálne jedno pravidlo *vo všetkých možných hĺbkach*.

Automat ${}_nM$ je *striktne deterministický* ak platí:

$$card(\{mqA \to ow | mqA \to ow \in R, o \in Q, w \in \Gamma^+\} - \{mqA \to pv\}) = 0$$

Definícia 3.3.7. Majme automat ${}_nM = (Q, \Sigma, \Gamma, R, s, S, F)$, $mqA \to pv \in R$. Pre *determinizmus s ohľadom na hĺbku* hlbokého zásobníkového automatu platí, že automat môže pre každý stav použiť maximálne jedno pravidlo **pre každú možnú hĺbku**.

Automat ${}_nM$ je *deterministický s ohľadom na hĺbku* ak platí pre každé $q \in Q$:

$$card(\{m | mqA \to pv \in R, A \in \Gamma, p \in Q, v \in \Gamma^+\}) \leq 1$$

3.3.1 Modifikácie hlbokého zásobníkového automatu

Verzia hlbokého zásobníkového automatu, predstavená v článku [6], pri operácii *pop* z definície 3.3.3 nezmení svoj stav. Pravidlá pre odstránenie aktuálne prečítaného terminálneho symbolu sú implicitné. Operácia *expanzie* 3.3.4 neterminálneho symbolu sa zase vykonáva podľa bezkontextového prepisovacieho pravidla automatu. Táto definícia vyvolala otázku, čo sa stane, ak by operácia *pop* nebola implicitná pre každý stav. Túto otázku zodpovedá článok [3], ktorý sa touto problematikou zaoberá.

Skúma verziu hlbokého zásobníkového automatu, kde nie je operácia *pop* implicitná pre každý stav, ale je vykonávaná na základe explicitného pravidla. Čo umožňuje, aby automat menil svoj stav aj pri operácii *pop*. Preto bola zadefinovaná nová modifikácia *Hlboký zásobníkový automat meniaci stav pri operácii pop*, s hĺbkou i, ktorý prijíma triedu jazykov ${}_{deep}\mathbf{pPD}_i$.

Ďalej je vyriešená otvorená otázka z článku, kde boli tieto automaty prvý krát predstavené, a to **mazanie neterminálnych symbolov**. Pôvodná verzia hlbokých zásobníkových automatov dovoľuje iba prepísanie neterminálneho symbolu terminálnym symbolom, alebo neterminálnym symbolom. Nie však mazanie, t.j. nevyskytujú sa v množine pravidiel žiadne tzv. ε-pravidlá. Z toho vyplýva, že zásobník nesmie nikdy obsahovať viac symbolov ako slovo, ktoré má byť prijaté. Je teda zrejmé, že môžu byť týmto spôsobom prijaté nanajvýš jazyky z rodiny **CS**, uvedené v definícii 2.3, teda kontextové jazyky.

S možnosťou mať ε-pravidlá môžeme rozširovať zásobník bez limitu a potom zmenšovať. V článku je preto zadefinovaná ďalšia modifikácia týchto automatov, a to *Hlboký zásobníkový automat s mazacími pravidlami*. Tento typ automatu prijíma rodinu jazykov ${}_{deep}\mathbf{dPD}_i$, ktorá je ekvivalentná rodine jazykov **RE**, teda rekurzívne vyčísliteľným jazykom. V článku [3] sa nachádza dôkaz, že platí ${}_{deep}\mathbf{dPD}_i = \mathbf{RE}$.

Hlboký zásobníkový automat riadený vstupom

Hlboký zásobníkový automat z definície 3.3.1 pracuje na princípe generovania reťazca daného jazyka na svojom zásobníku nezávisle na vstupe. Automat postupne vygenerovaný

reťazec porovnáva so vstupom. Základná varianta automatu teda nie je *riadená vstupom*[3]. Ak chceme daný automat používať v praxi pre syntaktickú analýzu, tak potrebujeme variantu automatu, ktorý bude deterministický a *riadený vstupom*.

Vo *vstupom riadených zásobníkových automatoch* (IDPDA), uvedených v článku [9], okrem stavu, obsahu zásobníka určuje výber pravidla aj aktuálny vstupný symbol.

Verzia hlbokého automatu riadeného vstupom bola prvý krát predstavená v diplomovej práci [11] s názvom *Řízený hluboký zásobníkový automat*. S ohľadom na článok [9] uvádzame upravený názov a definíciu.

Definícia 3.3.8. *Hlboký zásobníkový automat riadený vstupom* (IDDPDA[3]) je sedmica $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- Γ je *abeceda zásobníkových symbolov*, kde $\mathbb{N} \cap Q \cap \Gamma = \emptyset$, $\Sigma \subseteq \Gamma$ a $\# \in \Gamma - \Sigma$. Špeciálny symbol $\#$ značí dno zásobníka,
- $R \subseteq (\mathbb{N} \times (\Gamma - (\Sigma \cup \{\#\})) \times Q \times (\Sigma \cup \{\epsilon\}) \times Q \times (\Gamma - \{\#\})^+)$
  $\cup\ (\mathbb{N} \times \{\#\} \times Q \times (\Sigma \cup \{\epsilon\}) \times Q \times (\Gamma - \{\#\})^* \{\#\})$
  Pravidlo $(m, A, q, a, p, v) \in R$ je možné zapísať v tvare $mAqa \to pv$,
- $s \in Q$ je *počiatočný stav*,
- $S \in \Gamma$ je *počiatočný zásobníkový symbol*,
- $F \subseteq Q$ je *množina koncových stavov*.

Definícia 3.3.9. *Konfigurácia* automatu M, χ, je každý reťazec z množiny

$$(\Gamma - \{\#\})^* \{\#\} \times Q \times \Sigma^*.$$

Ak $x = (uAz, q, aw)$, $y = (p, uvz, w)$, $mAqa \to pv \in R$, kde q, $p \in Q$, $a \in (\Sigma \cup \{\epsilon\})$, $w \in \Sigma^*$, $A \in \Gamma$, u, v, $z \in \Gamma^*$ a platí, že $\text{occur}(u, \Gamma - \Sigma) = m - 1$. Automat M vykoná prechod $x_e \Rightarrow y$ podľa pravidla $mAqa \to pv \in R$, zapíšeme ako

$$x_e \Rightarrow y[mAqa \to pv]$$

Pravidlo $mAqa \to pv$ je *pravidlo hĺbky* m. Podľa toho $x_e \Rightarrow y[mAqa \to pv]$ je *rozšírenie v hĺbke* m.

Z definície je zrejmé, že sa táto modifikácia *Hlbokého zásobníkového automatu riadeného vstupom* líši od pôvodnej verzie 3.3.1 iba tvarom pravidiel. Definície operácií *pop* 3.3.3 a *expanzie* 3.3.4 a *jazyka prijímaného automatom* je možné analogicky odvodiť.

V definícii automatu typu *Řízený hluboký zásobníkový automat* v diplomovej práci [11] sa uvádza, že pri operácii *expanzia* bude zo vstupu symbol iba prečítaný, nie odstránený, teda považovaný za spracovaný. Inými slovami hlava stavového riadenia zostane na aktuálnom vstupnom symbole aj po ukončení aktuálneho výpočetného kroku. My budeme uvažovať verziu, kde symbol zo vstupu bude pri operácii *expanzia* spracovaný a teda odstránený zo vstupnej pásky – hlava stavového riadenia sa posunie doľava na ďalší vstupný symbol po ukončení aktuálneho výpočetného kroku.

[3]Terminológia *Vstupom-riadený* je prevzatá z anglického názvu *input-driven* z článku, ktorý sa venuje vstupom riadeným zásobníkovým automatom [9]. Odkiaľ je prevzatá aj skratka **IDPDA**.

Determinizmus

Definícia 3.3.10. Majme IDDPDA ${}_nM = (Q, \Sigma, \Gamma, R, s, S, F)$, automat ${}_nM$ je *striktne deterministický* ak platí

$$card(\{mAqa \to wo | mAqa \to wo \in R, o \in Q, w \in \Gamma^+, a \in (\Sigma \cup \{\epsilon\})\}) \leq 1.$$

Definícia 3.3.11. Majme IDDPDA ${}_nM = (Q, \Sigma, \Gamma, R, s, S, F)$, automat ${}_nM$ je *deterministický s ohľadom na hĺbku*, ak platí pre každé $q \in Q$,

$$card(\{m | mqA \to wp \in R, A \in \Gamma, p \in Q, w \in \Gamma^+\}) \leq 1.$$

Kapitola 4

Viac-zásobníkové automaty

V nasledujúcej kapitole si predstavíme viac-zásobníkové automaty. Tieto typy automatov sú modifikácie klasického zásobníkového automatu uvedeného v časti 2.4.5, ku ktorému je pridaný jeden alebo viac zásobníkov, čo, ako sa v tejto kapitole dočítame, zvyšuje ich výpočetnú silu. Táto kapitola je členená do troch logických sekcií.

V prvej sekcii si formálne zadefinujeme dvoj-zásobníkový automat a predstavíme spôsob jeho práce, zadefinujeme jeho konfiguráciu, prechod a jazyk, ktorý tento typ automatu prijíma. V závere prvej sekcie je uvedený dôkaz, že vyjadrovacia sila dvoj-zásobníkového automatu sa vyrovná sile Turingových strojov popísaných v časti 2.4.

Na záver tejto kapitoly využijeme doposiaľ nadobudnuté znalosti a zavedieme nové modifikácie zásobníkových automatov.

4.1 Dvoj-zásobníkové automaty

V tejto sekcii sa nachádza definícia simultánneho dvoj-zásobníkového automatu, čo znamená, že pravidlá sú definované tak, že sa počas jedného prechodu môže zmeniť obsah obidvoch zásobníkov naraz. Druhou variantou sú automaty, ktoré zmenia obsah iba jedného zásobníka počas jedného prechodu.

Definícia 4.1.1. *Dvoj-zásobníkový automat* je osmica $M = (Q, \Sigma, \Gamma, R, z, Z_1, Z_2, F)$, kde

- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- Γ je *abeceda zásobníkových symbolov*, $Q \cap (\Sigma \cup \Gamma) = \emptyset$,
- R je konečná množina *pravidiel* tvaru: $u_1|u_2qw \to v_1|v_2p$, kde

$$u_1, u_2 \in \Gamma, v_1, v_2 \in \Gamma^*, q, p \in Q, w \in \Sigma^*,$$

- $z \in Q$ je *počiatočný stav*,
- $Z_1 \in \Gamma$ je *počiatočný symbol na prvom zásobníku*,
- $Z_2 \in \Gamma$ je *počiatočný symbol na druhom zásobníku*,
- $F \subseteq Q$ je množina *koncových stavov*.

Definícia 4.1.2. *Konfigurácia* dvoj-zásobníkového automatu M, χ, je každý reťazec tvaru $\$v_1\$v_2qy$, kde pre každé $v_1, v_2 \in \Gamma^*, y \in \Sigma^*, q \in Q$ a \$ značí spodok zásobníku (anglicky *bottom marker*) ($\$ \notin Q \cup \Sigma \cup \Gamma$).

Prechod vykoná automat M ak $u_1|u_2qw \to v_1|v_2p \in R, y = \$h_1u_1\$h_2u_2qwz$ a $x = \$h_1v_1\$h_2v_2pz$, kde $u_1, u_2 \in \Gamma, h_1, h_2, v_1, v_2 \in \Gamma^*, q, p \in Q$ a $w, z \in \Sigma^*$, potom automat M vykoná prechod z y do x. Tento prechod zapíšeme ako

$$y \Rightarrow x[u_1|u_2qw \to v_1|v_2p]$$

Štandardne ako u iných automatov rozširujeme $\Rightarrow$ na $\Rightarrow^n$, kde $n \geq 0$ a na základe $\Rightarrow^n$ je definované $\Rightarrow^*$ a $\Rightarrow^+$.

Definícia 4.1.3. *Jazyk prijímaný dvoj-zásobníkovým automatom* M, $L(M)$, je definovaný ako:

$$L(M) = \{w : w \in \Sigma^*, \$Z_1\$Z_2zw \Rightarrow^* \$\$f, f \in F\}$$

Definícia 4.1.4. Majme sekvenciu prechodov $\$u_1\$u_2qtdh \Rightarrow \$v_1\$v_2pdh \Rightarrow \$w_1\$w_2oh$, ktorú vykoná automat M, kde $u_1, u_2, v_1, v_2, w_1, w_2 \in \Gamma^*, q, p, o \in Q, t, d, h \in \Sigma^*$. Ak $|v_i| \geq |u_i|$ a $|v_i| > |w_i|$ pre $i \in 1, 2$, potom automat M počas prechodu $\$v_1\$v_2pdh \Rightarrow \$w_1\$w_2oh$ vykoná **otáčku** v zásobníku i, čo znamená, že zásobník sa skráti.

Ak $|v_i| \geq |u_i|$ a $|v_i| > |w_i|$ pre obidve $i = 1, 2$, potom automat M počas prechodu $\$v_1\$v_2pdh \Rightarrow \$w_1\$w_2oh$ vykoná **simultánnu otáčku** v obidvoch zásobníkoch. Počas jedného prechodu automat skráti oba zásobníky.

Definícia 4.1.5. *Simultánne viac-zásobníkové automaty* sú automaty, ktoré dovoľujú vykonať súčasne zmenu na viacerých zásobníkoch automatu podľa [4]. Tvar pravidiel *Simultánneho viac-zásobníkového automatu* je uvedený v definícii 4.1.1.

Majme automat M z definície 4.1.1, ktorý *nie je simultánny*, potom konečná množina pravidiel R obsahuje pravidlá tvaru $i \mid uqw \to vp$, kde $i \in \mathbb{N}^+$, i ktoré identifikuje zásobník, $u \in \Gamma, v \in \Gamma^*, q, p \in Q, w \in \Sigma^*$. Ak nebude v texte priamo špecifikované, že automat je simultánny, potom bude počas jedného prechodu povolená manipulácia práve s jedným zásobníkom.

Generatívna sila dvoj-zásobníkového automatu

Veta 4.1.1. Ak je jazyk L prijímaný *Turingovým strojom* z definície 2.4, potom L je prijímaný *Dvoj-zásobníkovým automatom* z definície 4.1.1.

Dôkaz. Nech M je jedno-páskový *Turingov stroj* z definície 2.4, L jazyk prijímaný automatom M, $L(M)$ a S je *Dvoj-zásobníkový automat* z definície 4.1.1, ktorý simuluje pásku Turingovho stroja nasledovne:

1. Pri inicializácii sa na každom zásobníku automatu S nachádza symbol #, ktorý značí dno zásobníka a nesmie sa vyskytovať na inom mieste v zásobníku. Symbol dna zásobníka indikuje, že zásobník je prázdny.

2. Uvažujme na vstupe automatu S reťazec w, ktorý skopírujeme na prvý zásobník.

3. Z prvého zásobníka pomocou operácie *pop* premiestníme reťazec na zásobník č. 2. Prvý zásobník je prázdny a druhý obsahuje reťazec w.

4. Automat S simuluje prvý stav automatu M. Zásobník č. 1. indikuje, že ľavá strana pásky je prázdna, teda obsahuje tzv. *blank* symboly Δ. Na druhom zásobníku je reťazec w, čo indikuje, že hlava automatu M ukazuje na najľavejší symbol vstupného reťazca w.

5. Automat S simuluje pohyby automatu M nasledovne:

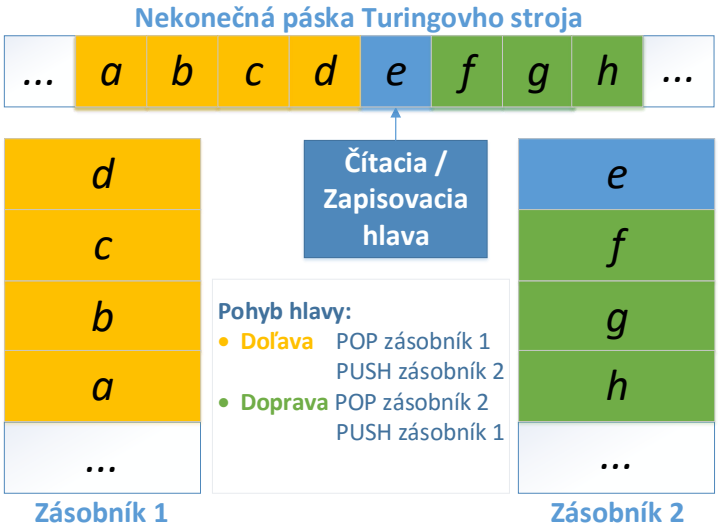
   (a) Posun hlavy Turingovho stroja *doľava* - podľa definície 2.4.9 - vykonáme presunom obsahu na vrchole prvého zásobníka, do druhého zásobníka a dostaneme sa na správne miesto na páske.

   (b) Posun hlavy Turingovho stroja *doprava* - podľa definície 2.4.9 - vykonáme presunom obsahu na vrchole druhého zásobníka, do prvého zásobníka a dostaneme sa na správne miesto na páske.

   (c) Automat S má vlastné konečné stavové riadenie, ktorým simuluje rovnako prechody stavov ako automat M.

Z vyššie uvedeného je zrejmé, že *Dvoj(a viac)-zásobníkový automat*, má rovnakú výpočetnú silu ako *Turingov stroj*, teda *Dvoj-zásobníkový automat* prijíma *rekurzívne vyčísliteľné jazyky*. □

Simulácia nekonečnej pásky Turingovho stroja dvomi zásobníkmi je uvedená na obrázku 4.1.



Obr. 4.1: Simulácia nekonečnej pásky Turingovho stroja pomocou dvoch zásobníkov. Zásobník číslo 1 simuluje pravú časť nekonečnej pásky a zásobník číslo dva jej ľavú časť.

4.2 Nove verzie zásobníkových automatov

V nasledujúcej časti si zavedieme nové modifikácie zásobníkových automatov, ktoré vychádzajú z automatov uvedených v predchádzajúcom texte.

4.2.1 Hlboký viac-zásobníkový automat

Zavedenie automatu typu *Hlboký viac-zásobníkový automat* bolo inšpirované viac-zásobníkovými automatmi uvedenými v sekcii 4.1 a *Hlbokým zásobníkovým automatom* uvedenom v časti 3.3.

Definícia 4.2.1. *Hlboký viac-zásobníkový automat* (nDPDA), kde $n \geq 1$ je $(6+n)$-tica $_nM = (Q, \Sigma, \Gamma, R, s, S_1, \ldots, S_n, F)$, kde

- n značí *počet zásobníkov*, $n \in \mathbb{N}^+$,
- d značí *maximálnu hĺbku* pre expanziu neterminálneho symbolu, $d \in \mathbb{N}^+$,
- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- Γ je *abeceda zásobníkových symbolov*, kde $\mathbb{N}^+ \cap Q \cap \Gamma = \emptyset$, $\Sigma \subseteq \Gamma$ a $\# \in \Gamma - \Sigma$. Špeciálny symbol $\#$ značí dno zásobníka,
- R *konečná množina pravidiel* tvaru $i[mqA \to pv$, kde $i, m \in \mathbb{N}^+$, i ktoré identifikuje zásobník, m značí hĺbku pravidla, $q, p \in Q$, $A \in (\Gamma - (\Sigma \cup \{\#\}))$ a $v \in (\Gamma - \{\#\})^*\{\#\}$,
- $s \in Q$ je *počiatočný stav*,
- $S_i \in \Gamma$ je *počiatočný symbol* zásobníku i, pre $i = 1, 2, \ldots, n$,
- $F \subseteq Q$ je *množina koncových stavov*.

Definícia 4.2.2. *Konfigurácia* automatu $_nM, \chi$, je každý reťazec z množiny

$$Q \times \Sigma^* \times \underbrace{(\Gamma - \{\#\})^*\{\#\}}_{n-\text{krát}}$$

Definícia 4.2.3. Ak $x = (q, w, u_1A_1z_1, u_2A_2z_2, \ldots, u_nA_nz_n)$,
$y = (p, w, u_1v_1z_1, u_2A_2z_2, \ldots, u_nA_nz_n)$, $1[mqA \to pv_1 \in R$, kde $q, p \in Q$, $w \in \Sigma^*$, $A \in \Gamma$, $u_1, u_2, \ldots u_3$, v_1, $z_1, z_2, \ldots z_3 \in \Gamma^*$ a platí, že $\text{occur}(1, u_1, \Gamma - \Sigma) = m - 1$. Automat $_nM$ vykoná *prechod* $x_e \Rightarrow y$ podľa pravidla $1[mqA \to pv_1 \in R$, zapíšeme ako

$$x_e \Rightarrow y[1[mqA \to pv_1]$$

Pravidlo $n[mqA \to pv$ je *pravidlo hĺbky* m aplikovateľné na zásobníku n . Podľa toho $x_e \Rightarrow y[n[mqA \to pv]$ je *rozšírenie v hĺbke* m.

Majme $n \in \mathbb{N}^+$. Ak každé pravidlo automatu M má hĺbku n alebo menej, potom hovoríme, že automat je *hĺbky* n, čo zapíšeme ako $_nM$.

Definícia 4.2.4. Automat $_nM$ vykoná operáciu *pop* z definície 3.3.3 ak vykoná *pop* operáciu počas prechodu aspoň na jednom zásobníku.

Definícia 4.2.5. Automat $_nM$ vykoná operáciu *expanzie* z definície 3.3.4, ak expanduje non-terminál aspoň na jednom zásobníku.

Definícia 4.2.6. Nech automat $_nM = (Q, \Sigma, \Gamma, R, s, S_1, \ldots, S_n, F)$ je n-DPDA, pre nejaké $n \geq 1$, hĺbky d, pre $d \in \mathbb{N}^+$. Definujeme jazyk prijímaný automatom $_nM$,

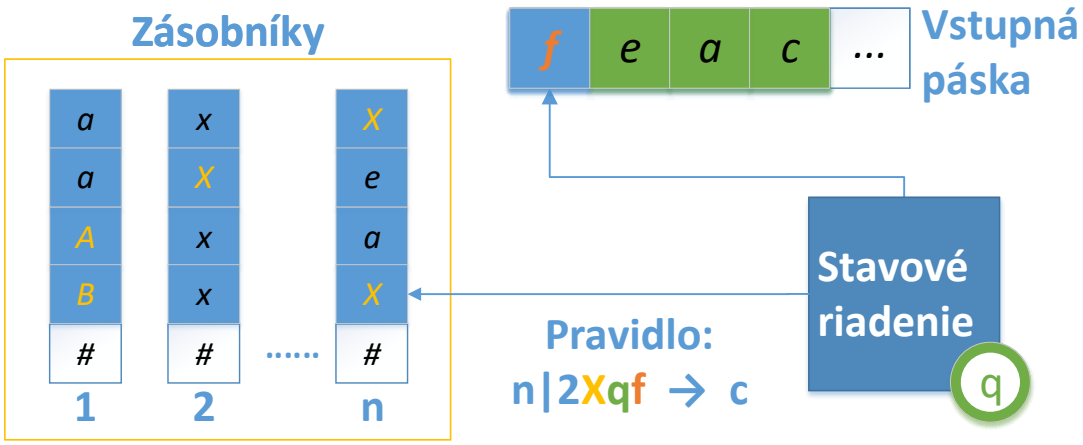
$$L(_nM) = \{w \in \Sigma^* \mid (\#Z_1\#Z_2 \ldots \#Z_n, s, w) \vdash_M^* (\underbrace{\#\# \ldots \#}_{n-\text{krát}}, f, \epsilon), \text{kde } f \in F\}$$

Ďalej definujeme jazyk, ktorý automat $_nM$ prijíma ak sú všetky zásobníky prázdne,

$$E(_nM) = \{w \in \Sigma^* \mid (\#Z_1\#Z_2 \ldots \#Z_n, s, w) \vdash_M^* (\underbrace{\#\# \ldots \#}_{n-\text{krát}}, q, \epsilon), \text{kde } q \in Q\}$$

4.2.2 Hlboký viac-zásobníkový automat riadený vstupom

Táto modifikácia vychádza z vyššie uvedeného typu automatu *Hlboký viac-zásobníkový automat*, definícia 4.2.1 a z *Hlbokého zásobníkového automatu riadeného vstupom*, uvedeného v definícii 3.3.8.



Obr. 4.2: Hlboký viac-zásobníkový automat riadený vstupom

Definícia 4.2.7. *Hlboký viac-zásobníkový automat riadený vstupom* (nIDDPDA), kde $n \geq 1$ je $(6+n)$-tica $_nM = (Q, \Sigma, \Gamma, R, s, S_1, \ldots, S_n, F)$, kde

- n značí *počet zásobníkov*, $n \in \mathbb{N}^+$,
- d značí *maximálnu hĺbku* pre expanziu neterminálneho symbolu, $d \in \mathbb{N}^+$,
- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda* $(Q \cap \Sigma)$,
- Γ je *abeceda zásobníkových symbolov*, kde $\mathbb{N}^+ \cap Q \cap \Gamma = \emptyset$, $\Sigma \subseteq \Gamma$ a $\# \in \Gamma - \Sigma$. Špeciálny symbol $\#$ značí dno zásobníka,
- R *konečná množina pravidiel* tvaru $i[mAqa \to vp$, kde

$$A \in (\Gamma - (\Sigma \cup \{\#\})) \text{ alebo } A = \{\#\}, q, p \in Q, a \in (\Sigma \cup \{\epsilon\}),$$
$$v \in (\Gamma - \{\#\})^+ \text{ alebo } v \in (\Gamma - \{\#\})^*\{\#\}$$

$i \in \mathbb{N}^+, m \in \mathbb{N}_0$, i ktoré identifikuje zásobník, m značí hĺbku pravidla. Ak je hĺbka pravidla 0, expanzia môže prebehnúť iba na vrchole zásobníka,

- $s \in Q$ je *počiatočný stav*,
- $S_i \in \Gamma$ je *počiatočný symbol* zásobníku i, pre $i = 1, 2, \ldots, n$,
- $F \subseteq Q$ je *množina koncových stavov*.

Definícia 4.2.8. *Konfigurácia* automatu $_nM, \chi$, je každý reťazec z množiny

$$Q \times \Sigma^* \times \underbrace{(\Gamma - \{\#\})^*\{\#\})}_{n-\text{krát}}$$

Ak $x = (u_1A_1z_1, u_2A_2z_2, \ldots, u_nA_nz_n, q, aw)$, $y = (u_1v_1z_1, u_2A_2z_2, \ldots, u_nA_nz_n, p, w)$, $1[mAqa \to v_1p \in R$, kde $q, p \in Q$, $w \in \Sigma^*$, $A \in \Gamma$, $u_1, u_2, \ldots u_3$, v_1, $z_1, z_2, \ldots z_3 \in \Gamma^*$ a platí, že $\text{occur}(1, u_1, \Gamma - \Sigma) = m - 1$. Automat $_nM$ vykoná *prechod* $x_e \Rightarrow y$ podľa pravidla $1[mAqa \to v_1p$, zapíšeme ako

$$x_e \Rightarrow y[1[mAqa \to v_1p]$$

Pravidlo $n[mAqa \to vp$ je *pravidlo hĺbky* m aplikovateľné na zásobníku n . Podľa toho $x_e \Rightarrow y[n[mAqa \to vp]$ je *rozšírenie v hĺbke* m.

Definícia 4.2.9. Automat $_nM$ vykoná operáciu *pop* z definície 3.3.3 ak vykoná operáciu *pop* počas prechodu aspoň na jednom zásobníku.

1. Ak sa na vrchole viacerých zásobníkov nachádza symbol a z množiny Σ, $a \in \Sigma$, operáciu *pop* je možné vykonať len na zásobníku, s najnižším identifikátorom.
2. V prípade, ak chceme vykonávať operáciu *pop* na ktoromkoľvek zásobníku, ktorý obsahuje na vrchole zásobníka vstupný symbol a, tak musíme definovať pravidlo $n[aqa \to p$ pre operáciu *pop*, pre každý symbol $a \in \Sigma \cap \Gamma$, kde $n \in \mathbb{N}^+, q, p \in Q, a \in \Sigma \cap \Gamma$.

Príklad 4.2.1. Majme automat 2DPDA, z definície 4.2.1 kde $_2M = (Q, \Sigma, \Gamma, R, s, S_1, S_2, F)$, s nasledujúcou konfiguráciou

$$(q, cu, bw_1\#cw_2\#), \text{ kde } q \in Q, c, b \in \Sigma, u \in \Sigma^*, w_1, w_2 \in \Gamma^*$$

1. Operácia *pop* definovaná podľa 4.2.9 bod č.1 môže byť vykonaná iba na zásobníku 1, avšak na vstupe je spracovávaný symbol c. Ak nie je v množine R iné pravidlo, ktoré je možné uplatniť, vstup bude zamietnutý.
2. Operácia *pop* definovaná podľa 4.2.9 bod č.2 môže byť vykonaná na zásobníku 2, ak platí, že $2[cqc \to p \in R$.

   Automat vykoná operáciu *pop* nasledovne:

   $$(q, cu, bw_1\#cw_2\#)_p \Rightarrow (p, u, bw_1\#, w_2\#) \text{ podľa pravidla } 2[cqc \to p.$$

Zvyšné definície sú analogicky odvoditeľné z definície hlbokého viac-zásobníkového automatu typu nDPDA uvedeného v časti 4.2.1.

Princíp činnosti a komponenty nIDDPDA sú uvedené na obrázku 4.2.

Generatívna sila Hlbokého viac-zásobníkového automatu riadeného vstupom

Veta 4.2.1. Každý *Dvoj-zásobníkový automat* M z definície 4.1.1 je možné previesť na *Hlboký viac-zásobníkový automat riadený vstupom* M' z definície 4.2.7 tak, že $L(M) = L(M')$.

Dôkaz. Majme *Dvoj-zásobníkový automat* $M = (Q, \Sigma, \Gamma, R, s, S_1, S_2, F)$, a *Hlboký viac-zásobníkový automat riadený vstupom* ${}_2M' = (Q', \Sigma', \Gamma', R', s', S_1', S_2', F')$. Nájdeme algoritmus prevodu $M \to M'$:

1. $\Sigma' = \Sigma$
2. $\Gamma' = \Gamma \cup \{S_1', S_2'\}$
3. $S_1' = S_1, S_2' = S_2$
4. $Q' = Q$, $s' = s$, $F' = F$
5. Množina pravidiel R' je definovaná nasledovne:
   Pre každé pravidlo tvaru $n[Aqa \to bp \in R$, kde n značí identifikátor zásobníka platí $n \in \{1, 2\}, A \in \Gamma, a \in (\Sigma \cup \{\varepsilon\}), q, p \in Q, p \in \Gamma^*$ vytvoríme pravidlo $n[0Aqa \to bp \in R$, kde 0 značí hĺbku aplikovania pravidla. Čo značí, že operácia *expanzia* môže prebehnúť iba na vrchole zásobníka, tak ako v pôvodnom automate M.
6. Operácia *pop* je definovaná podľa 4.2.9, bod 2, čo značí, že implicitné pravidlá pre vynímanie symbolov zo zásobníka nie sú definované. T.j. sú zachované pravidlá vynímania symbolov podľa pravidiel, ktoré boli vložené do množiny R' v predchádzajúcom bode č. 5.

Dôsledok. Hlboký viac-zásobníkový automat riadený vstupom prijíma rovnakú rodinu jazykov ako Dvoj-zásobníkový automat, t.j. rodinu jazykov typu 0, rekurzívne vyčísliteľné jazyky. □

Veta 4.2.2. Ak je jazyk L prijímaný *Turingovým strojom* z definície 2.4, potom L je prijímaný *Hlbokým viac-zásobníkovým automatom riadeným vstupom* z definície 4.2.7.

Dôkaz. Dôsledok vety 4.1.1 a 4.2.1. □

4.2.3 Zásobníkový automat regulovaný hlbokým zásobníkom

Definícia 4.2.10. *Zásobníkový automat regulovaný hlbokým zásobníkom* je 10-tica $(M, \Xi) = (Q, \Sigma, \Gamma_1, \Gamma_\Xi, R_1, R_\Xi, z, Z_1, Z_\Xi, F)$, kde

- Ψ je abeceda identifikátorov pravidiel automatu. Potom Ξ je *riadiaci jazyk* nad abecedou Ψ, teda $\Xi \subseteq \Psi^*$.
- Q je *konečná množina stavov*,
- Σ je *vstupná abeceda*,
- Γ_1 je *abeceda zásobníkových symbolov na prvom zásobníku*, $Q \cap (\Sigma \cup \Gamma_1) = \emptyset$,
- Γ_Ξ je *abeceda zásobníkových symbolov na druhom zásobníku*, $Q \cap \Gamma_\Xi = \Sigma \cap \Gamma_\Xi = \emptyset$,

- R_1 je konečná množina *pravidiel* tvaru: $\rho.Apa \to wq$, kde

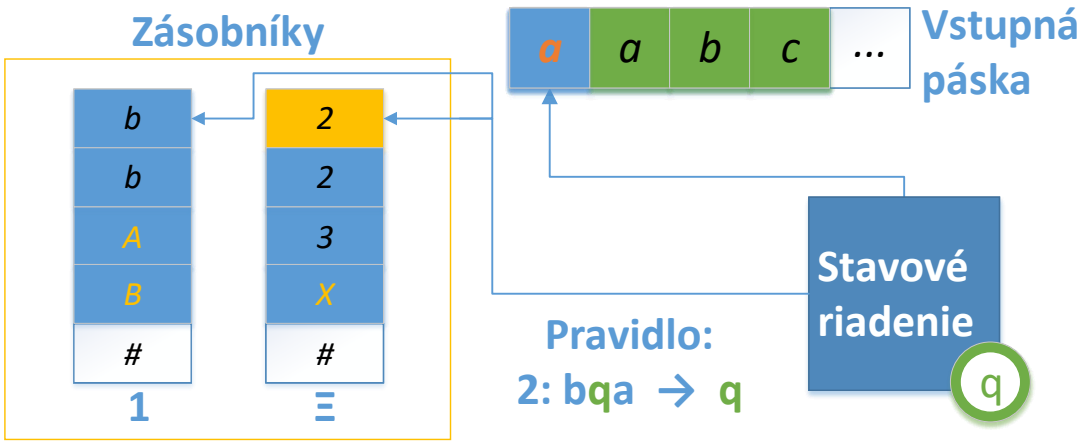
$$A \in \Gamma, p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, w \in \Gamma^*,$$

  a $\rho \in \Psi$ je jedinečný identifikátor pravidla.

- R_Ξ je konečná množina *pravidiel* generujúcich riadiaci jazyk Ξ tvaru: $mAa \to v$, kde

$$A \in (\Gamma_\Xi - \{\#\}), a \in (\Sigma \cup \{\epsilon\}), v \in \Gamma_\Xi *, m \in \mathbb{N} \text{ značí hĺbku pravidla,}$$

- $z \in Q$ je *počiatočný stav*,
- $Z_1 \in \Gamma$ je *počiatočný symbol na prvom zásobníku*,
- $Z_\Xi \in \Gamma$ je *počiatočný symbol na druhom zásobníku*,
- $F \subseteq Q$ je množina *koncových stavov*. m značí hĺbku pravidla.



Obr. 4.3: Zásobníkový automat regulovaný hlbokým zásobníkom, režim 2.

Definícia 4.2.11. *Konfigurácia* automatu $(M, \Xi), \chi$, je každý reťazec z množiny

$$Q \times \Sigma^* \times (\Gamma_1 - \{\#\})^*\{\#\} \times (\Gamma_\Xi - \{\#\})^*\{\#\}$$

Definícia 4.2.12. Ak $x = (q, aw, u_1, u_\Xi A_\Xi z_\Xi)$, $y = (p, w, u_1, u_\Xi v_\Xi z_\Xi)$, $mA_\Xi a \to pv_\Xi \in R$, kde q, $p \in Q$, $w \in \Sigma^*$, $a \in \Sigma$, $u_1 \in \Gamma$, $A \in \Gamma_\Xi$, $u_\Xi, v_\Xi, z_\Xi \Gamma_\Xi^*$ a platí, že $\text{occur}(1, u_\Xi, \Gamma_\Xi - \Sigma) = m - 1$. Automat n_dM vykoná *expanziu* na druhom zásobníku $x_e \Rightarrow y$ podľa pravidla $mA_\Xi a \to pv_\Xi$, zapíšeme ako

$$x_e \Rightarrow y[mA_\Xi a \to pv_\Xi]$$

Pravidlo $mA_\Xi a \to pv_\Xi$ je *pravidlo hĺbky* m aplikovateľné na riadiacom zásobníku. Podľa toho $x_e \Rightarrow y[mA_\Xi a \to pv_\Xi]$ je *rozšírenie v hĺbke* m.

Definícia 4.2.13. Ak $x = (q, aw, u_1 z_1, u_\Xi z_\Xi)$, $y = (p, w, u_2 z_1, z_\Xi)$, $u_\Xi : u_1 qa \to pu_2 \in R_1$, kde q, $p \in Q$, $w \in \Sigma^*$, $a \in \Sigma$, $u_1, u_2 \in \Gamma$, $z_1 \in \Gamma^*$, $u_\Xi \in (\Gamma_\Xi \cap \Psi)$, $z_\Xi \in \Gamma_\Xi^*$ a platí, že $\text{occur}(1, u_\Xi, \Gamma - \Sigma) = m - 1$. Automat M aplikuje pravidlo s identifikátorom u_Ξ, ktoré sa nachádza na vrchole riadiaceho zásobníka a na riadiacom zásobníku vykoná operáciu *pop*, na prvom zásobníku aplikuje pravidlo $u_\Xi : u_1 qa \to pu_2$. Tento prechod zapíšeme ako

$$x_p \Rightarrow y[u_\Xi : u_1 qa \to pu_2]$$

Automat môže pracovať v dvoch režimoch.

1. *Generovanie riadiacich pravidiel* na základe vstupných symbolov. Prebieha, ak sa na riadiacom zásobníku nachádza aspoň jeden neterminálny symbol a existuje pravidlo v R_Ξ pre vygenerovanie riadiaceho pravidla. Automat vykoná *expanziu* na riadiacom zásobníku.

2. *Aplikovanie riadiacich pravidiel* - na riadiacom zásobníku sa vykoná operácia *pop* a aplikuje sa pravidlo s identifikátorom, ktorý sa nachádzal na vrchole zásobníka.

Princíp činnosti a komponenty zásobníkového automatu regulovaného hlbokým zásobníkom sú uvedené na obrázku 4.3.

Definícia 4.2.14. Nech $\chi_0, \chi_1, ...\chi_n$, je sekvencia konfigurácií pre $n \geq 1$ a $\chi_{i-1} \Rightarrow \chi_i[\rho_i]$, kde $\rho_i \in \Psi$ pre všetky $i = 1, ..., n.$, potom sa automat M n-tými prechodmi presunie z χ_0 do χ_n podľa $[\rho_1 ... \rho_n]$. Zapíšeme ako

$$\chi_0 \Rightarrow^n \chi_n[\rho_1 ... \rho_n]$$

Automat M prijme vstupný reťazec x, iba v prípade, ak Ξ obsahuje riadiaci reťazec, podľa ktorého automat M vykonal sekvenciu krokov, teda po prečítaní reťazca x prešiel do konečnej konfigurácie.

Definícia 4.2.15. Pomocou dvojice (M, Ξ) [1] definujeme nasledujúce akceptovateľné jazyky

- $L(M, \Xi, 1)$ - *jazyk akceptovaný finálnym stavom*
- $L(M, \Xi, 2)$ - *jazyk akceptovaný prázdnym zásobníkom*
- $L(M, \Xi, 3)$ - *jazyk akceptovaný finálnym stavom a prázdnym zásobníkom*

Nech $\chi \in Q\Sigma^*(\Gamma_1 - \{\#\})^*\{\#\}(\Gamma_\Xi - \{\#\})^*\{\#\}$.

- Ak $\chi \in F\Sigma^*(\Gamma_1 - \{\#\})^*\{\#\}(\Gamma_\Xi - \{\#\})^*\{\#\}$, potom χ je *1 - finálna konfigurácia*,
- Ak $\chi \in Q\#\#$, potom χ je *2 - finálna konfigurácia*,
- Ak $\chi \in F\#\#$, potom χ je *3 - finálna konfigurácia.*

Pre $i = 1, 2, 3$ definujeme

$L(M, \Xi, i) = \{w | w \in \Sigma^*$ a $Ssw \vdash_M^* \chi[\sigma]$ pre každú i -finálnu konfiguráciu χ a $\sigma \in \Xi\}$

Veta 4.2.3. Ak je jazyk L prijímaný *Turingovým strojom* z definície 2.4, potom L je prijímaný *Zásobníkovým automatom regulovaným hlbokým zásobníkom* z definície 4.2.10.

[1] Dvojicu (M, Ξ) nazývame riadený zásobníkový automat.

Dôkaz. (Idea)

- V publikácii [8] v sekcii **16.2.3** je k dispozícii dôkaz, že ak je zásobníkový automat regulovaný riadiacim lineárnym jazykom, tak je schopný prijímať rovnakú rodinu jazykov ako Turingov stroj, teda rekurzívne vyčísliteľné jazyky.
- Činnosť lineárnej gramatiky, z definície 2.3, je možné simulovať hlbokým zásobníkovým automatom.
- Pre každé pravidlo lineárnej gramatiky tvaru $A \to xBy$ kde $A, B \in N$ a $x, y \in T^*$ je možné vytvoriť riadiace pravidlo v zásobníkovom automate, ktorý je regulovaný hlbokým zásobníkom v množine pravidiel R_{Ξ} tvaru: $mA \to xBy$.

□

Kapitola 5

Syntaktická analýza založená na viac-zásobníkovom automate

V nasledujúcej kapitole sa zoznámime s pokročilými metódami syntaktickej analýzy a v krátkosti si uvedieme akú úlohu zohráva syntaktická analýza v prekladačoch.

V druhej časti navrhneme kombinované metódy syntaktickej analýzy, ktoré sú založené na modifikovaných zásobníkových automatoch definovaných v kapitole 4. Syntaktická analýza bude prezentovaná na konkrétnych príkladoch a budú zhodnotené prínosy jednotlivých prístupov.

5.1 Metódy syntaktickej analýzy

Na úvod si v skratke predstavíme činnosť *prekladača* a jeho jednotlivé fázy. Podrobnejšie sa budeme venovať srdcu prekladača – *syntaktickému analyzátoru*, ktorý okrem vykonávania syntaktickej analýzy, riadi a kontroluje ostatné komponenty prekladača.

Prekladač, taktiež nazývaný ako kompilátor, je program, ktorý slúži pre preklad algoritmov zapísaných vo vyššom programovacom jazyku (napr. C, C++) do jazyka nižšieho, najčastejšie do strojového kódu. Na vstupe prijíma zdrojový program zapísaný v zdrojovom jazyku a na výstupe generuje funkčne ekvivalentný cieľový program v cieľovom jazyku.

Transformáciu, ktorú prevádza prekladač nazývame *preklad*. Preklad zdrojového programu prebieha v šiestich fázach: Lexikálna analýza, Syntaktická analýza, Sémantická analýza, Generátor vnútorného kódu, Optimalizácia, Generátor cieľového kódu.

Lexikálna analýza rozloží zdrojový kód na lexémy – lexikálne jednotky. Tieto jednotky sa nazývajú tokeny, a predstavujú jednotlivé rozoznané elementy zdrojového jazyka. Napríklad identifikátory, zápisy čísel, kľúčové slová, a pod. Lexikálna analýza produkuje *tokeny*, ktoré sú vstupom pre syntaktickú analýzu.

Úlohou *syntaktickej analýzy* v prekladačoch je rozpoznať, či zadaná postupnosť znakov - zdrojový kód, je syntakticky správna. V kladnom prípade je výstupom syntaktickej analýzy derivačný strom tokenov, ktorý reprezentuje program. Program, ktorý túto analýzu vykonáva sa nazýva *syntaktický analyzátor*.

Sémantický analyzátor vykonáva kontrolu rôznych sémantických aspektov programu, ako napr. deklaráciu premenných.

Generátor kódu vytvorí na základe derivačného stromu zdrojového programu cieľový kód. Generovanie kódu sa skladá z troch fáz: generovanie vnútorného kódu, optimalizácia a generovanie cieľového programu.

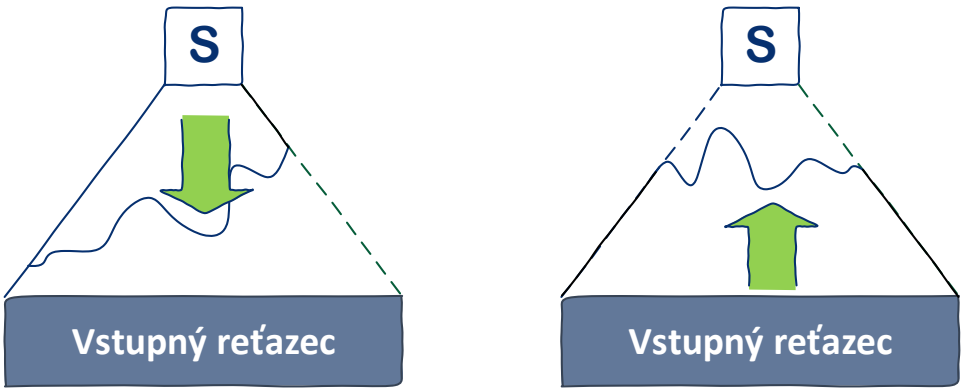
Týchto šesť komponentov prekladača úzko spolupracuje. Syntaktický analyzátor kontroluje a riadi ostatné komponenty. Na základe príkazu syntaktického analyzátora lexikálny analyzátor číta zdrojový program, pripraví token a pošle ho syntaktickému analyzátoru. Syntaktický analyzátor taktiež kontroluje sémantickú analýzu a dáva príkaz generátoru kódu, aby vytvoril cieľový program.

Fáza syntaktickej analýzy je oveľa zložitejšia než predchádzajúca fáza prekladu, teda lexikálna analýza. Lexémy sú popísané regulárnym jazykom, ktorý dokáže jednoducho rozoznať a prijať konečný automat. Syntaktický analyzátor rozpoznáva zložitejšie jazyky. Najčastejšie je to bezkontextový jazyk, ktorý popisuje aj programovacie jazyky a tento jazyk je akceptovaný zásobníkovým automatom.

Ako už bolo spomenuté, syntaktická analýza je dôležitou časťou spracovania programu pri jeho preklade, pretože prevádza analýzu vstupného jazyka. Rozoznáva, či je program zapísaný syntakticky správnym spôsobom. Napríklad v jazyku C, zistí či blok programu, ktorý začal „`{`“ je následne ukončený „`}`“ alebo určuje v akom poradí sa budú prevádzať jednotlivé príkazy, napr. „`a + b * c`“. V tomto prípade určí, že sa ako prvé vykoná násobenie a následne sčítanie. Ak je k danému reťazcu tokenov nájdený *derivačný strom*, potom je program správny.

Vytváranie derivačného stromu je založené na gramatických pravidlách. Existujú dva prístupy syntaktickej analýzy *zhora dole* a *zdola nahor*, viď obrázok 5.2. Syntaktická analýza založená na princípe *zhora dole* prekladá vety bezkontextového jazyka na ich ľavé rozbory. Tento syntaktický analyzátor začína s počiatočným symbolom gramatiky a postupným aplikovaním pravidiel vytvára derivačný strom, kde listy tvoria vstupný reťazec.

Syntaktická analýza založená na princípe *zhora dole* prekladá vety bezkontextového jazyka na ich pravé rozbory. Tento syntaktický analyzátor začína so vstupným reťazcom – listami derivačného stromu, na ktoré uplatňuje pravidlá a postupne vytvára podstromy, ktoré spája. Koreňom tohto vytvoreného stromu je počiatočný symbol gramatiky.



(a) Z S smerom k vstupnému reťazcu (b) Zo vstupného reťazca smerom k S

Obr. 5.1: Vytváranie derivačného stromu

5.2 Modifikované metódy syntaktickej analýzy

Náplňou tejto práce je zostrojiť syntaktickú analýzu, ktorá je založená na nami zavedených modifikovaných zásobníkových automatoch. Analýza syntaktických štruktúr, ktoré nie sú

bezkontextové, bude demonštrovaná príkladmi. Jednotlivé prístupy budeme medzi sebou porovnávať a zhodnocovať ich prínosy.

Zásobníkové automaty založené na hlbokých zásobníkoch, ktoré boli predstavené v predchádzajúcich kapitolách 3 a 4, kopírujú model syntaktického analyzátora vytvárajúceho syntaktický strom *zhora dole*, ktorého princíp je zobrazený na obrázku 5.1a. Tento analyzátor, rovnako ako aj zásobníkový automat založený na hlbokých zásobníkoch, postupne generuje na svojom/svojich zásobníku/zásobníkoch reťazec daného jazyka a porovnáva ho so vstupným reťazcom.

Na začiatok budeme skúmať syntaktickú analýzu založenú na *Hlbokom zásobníkovom automate, ktorý je riadený vstupom*, uvedenom v definícii 3.3.8. Tento prístup budeme porovnávať s verziou založenou na *Hlbokom zásobníkovom automate*, aby sme si ukázali, aký má prínos brať do úvahy vstupný symbol pri výbere pravidla.

Ďalej bude nasledovať zrovnanie *Hlbokého zásobníkového automatu, ktorý je riadený vstupom* s viac-zásobníkovou verziou, ktorá z neho vychádza.

Na koniec tejto časti uvedieme ukážku syntaktickej analýzy, ktorá je založená na *Zásobníkovom automate, ktorý je regulovaný hlbokým zásobníkom.*

Princíp vytvárania syntaktického stromu ukážeme pomocou jednoduchého rekurzívne vyčísliteľného jazyka, ktorý bude akceptovaný našim zostrojeným syntaktickým analyzátorom.

Porovnanie IDDPDA s DPDA

V nasledujúcej časti si ukážeme aký má prínos brať do úvahy aktuálny vstupný symbol pri výbere pravidla. V prvom príklade si znázorníme, ako pracuje Hlboký zásobníkový automat a v druhom demonštrujeme prácu Hlbokého zásobníkového automatu riadeného vstupom.

Oba automaty budú zostrojené pre rovnaký jazyk a ukážeme prácu týchto automatov prijatím rovnakého vstupného reťazca. Automaty budú prijímať jazyk prázdnym zásobníkom a zároveň koncovým stavom.

Príklad 5.2.1. Uvažujme hlboký zásobníkový automat ${}_2M_1$ z definície 3.3.1, ktorý prijíma jazyk $L = a^n b^n c^n$, kde $n \geq 1$,

$$ {}_2M_1 = (\{s, q, p, f\}, \{a, b, c\}, \{A, S, a, b, c, \#\}, R, s, S, \{f\}) $$

s nasledujúcimi pravidlami v množine R_1:

$$ \begin{array}{lll} 1sS \to qAA & 1qA \to fab & 1fA \to fc \\ 1qA \to paAb & 2pA \to qAc & \end{array} $$

Demonštrácia prijatia reťazca $aabbcc$ automatom ${}_2M_1$:

$$\begin{array}{lllll}
(s, aabbcc, S\#) & {}_e\Rightarrow & (q, & aabbcc, & AA\#) & [1sS \to qAA] \\
& {}_e\Rightarrow & (p, & aabbcc, & aAbA\#) & [1qA \to paAb] \\
& {}_p\Rightarrow & (p, & abbcc, & AbA\#) & \\
& {}_e\Rightarrow & (q, & abbcc, & AbAc\#) & [2pA \to qAc] \\
& {}_e\Rightarrow & (q, & abbcc, & abbAc\#) & [1qA \to fab] \\
& {}_p\Rightarrow & (f, & bcc, & bAc\#) & \\
& {}_p\Rightarrow & (f, & cc, & Ac\#) & \\
& {}_e\Rightarrow & (f, & cc, & Ac\#) & [1fA \to fc] \\
& {}_p\Rightarrow & (f, & cc, & cc\#) & \\
& {}_p\Rightarrow & (f, & c, & c\#) & \\
& {}_p\Rightarrow & (f, & \varepsilon, & \#) &
\end{array}$$

Príklad 5.2.2. Uvažujme hlboký zásobníkový automat riadený vstupom ${}_1M_2$ z definície 3.3.8, ktorý prijíma jazyk $L = a^n b^n c^n$, kde $n \geq 1$,

$${}_2M_{PDA} = (\{s, q, p, f\}, \{a, b, c\}, \{A, S, b, c, \#\}, R, s, S, \{f\})$$

s nasledujúcimi pravidlami v množine R_2:

$$1Ssa \to Aq \qquad\qquad 1Aqa \to bAcq \qquad\qquad 1Aqb \to cf$$

Demonštrácia prijatia reťazca $aabbcc$ automatom ${}_2M_2$:

$$\begin{array}{lllll}
(s, aabbcc, S\#) & {}_e\Rightarrow & (q, & abbcc, & A\#) & [1Ssa \to Aq] \\
& {}_e\Rightarrow & (q, & bbcc, & bAc\#) & [1Aqa \to bAcq] \\
& {}_e\Rightarrow & (f, & bcc, & bcc\#) & [1Aqb \to cf] \\
& {}_p\Rightarrow & (f, & cc, & cc\#) & \\
& {}_p\Rightarrow & (f, & c, & c\#) & \\
& {}_p\Rightarrow & (f, & \varepsilon, & \#) &
\end{array}$$

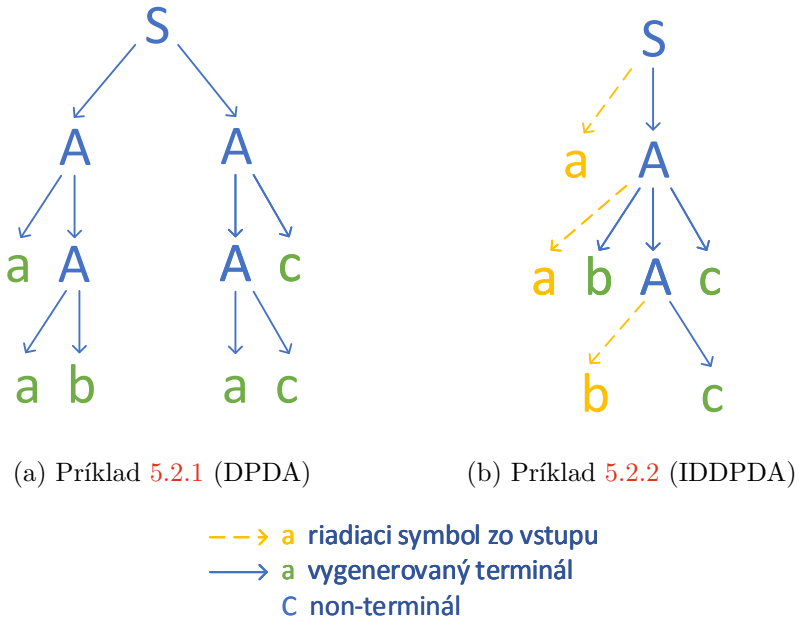
Na obrázku 5.2 sa nachádza grafická reprezentácia syntaktických stromov, ktoré vznikli pri akceptovaní reťazca z príkladu 5.2.1 a 5.2.2.

Ako si môžeme všimnúť kardinalita množiny R_{M_1}, $card(R_{M_1}) = 5$, automat M_1 (PDA) akceptoval reťazec v 11-tich derivačných krokoch, zatiaľ čo automat M_2 (IDDPDA) akceptoval reťazec v 6-tich derivačných krokoch. Taktiež je aj počet pravidiel automatu M_2 menší, $card(R_{M_2}) = 3$. Všimnime si, že automat M_2 je *striktne deterministický* a *deterministický s ohľadom na hĺbku*.

V prvom príklade boli vykonané štyri expanzie, z toho dve hlbšie v zásobníku, kým v príklade 5.2.2 boli vykonané iba dve expanzie a to obidve na vrchole zásobníka.

Predstavme si, ako by vyzerala implementácia[1]. Predpokladajme, že by sme mali implementovaný hlboký zásobník ako klasický zásobník. Pri každej expanzii by sme museli vybrať obsah, ktorý sa nachádza na vrchole zásobníka, skontrolovať, či je tento obsah náš neterminálny symbol, a ak áno, tak či sa nachádza v hĺbke, v ktorej chceme uplatniť pravidlo. Samozrejme musíme taktiež uvažovať réžiu spojenú s ukladaním vyňatého obsahu zo zásobníka do pamäte. Ak máme už nájdený žiadaný neterminálny symbol v požadovanej hĺbke, môžeme ho expandovať a vrátiť na zásobník dáta, ktoré mu predchádzali.

[1]Uvedený príklad implementácie je iba ilustračný a nie je považovaný za efektívny alebo vhodný pre hlboký zásobník. Implementácia tohto typu zásobníku bude diskutovaná neskôr.



(a) Príklad 5.2.1 (DPDA)

(b) Príklad 5.2.2 (IDDPDA)

(c) Legenda

Obr. 5.2: Syntaktické stromy zostrojené v príkladoch 5.2.1 a 5.2.2 *Hlbokým zásobníkovým automatom* a *Hlbokým zásobníkovým automatom riadeným vstupom* pri prijímaní reťazca *aabbcc*.

Z vyššie uvedeného je zrejmé, že expanzia hlbšie v zásobníku je zložitejšia ako expanzia na vrchole zásobníka. V prípade automatu DPDA sme ju museli vykonať dva krát, museli sme vykonať väčší počet derivačných krokov a taktiež kardinalita množiny pravidiel bola vyššia, čo má za následok vyššiu réžiu prehľadávania množiny pravidiel pri hľadaní pravidla, ktoré sa má uplatniť v derivačnom kroku. V neposlednom rade, objemnejšia množina pravidiel značí, že automat je zložitejší na zostrojenie.

Prínos verzie automatu IDDPDA je zreteľný a teda je táto modifikácia vhodnejšia pre syntaktickú analýzu.

Porovnanie nIDDPDA s IDDPDA

Aby sme mohli porovnávať verziu nIDDPDA s jedno-zásobníkovou variantou IDDPDA, budeme demonštrovať činnosť automatu nIDDPDA na rovnakom príklade.

Príklad 5.2.3. Uvažujme *simultánny* 4.1.5 automat typu nIDDPDA 2_1M_3 z definície 4.2.7, ktorý prijíma jazyk $L = a^n b^n c^n$, kde $n \geq 1$,

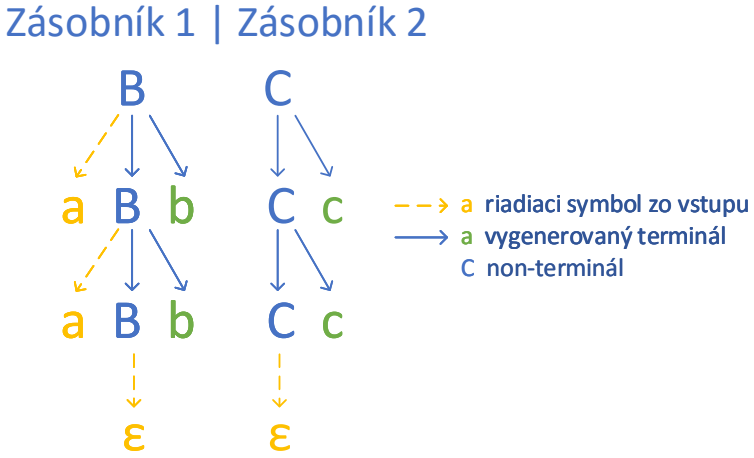
$$2_1M_3 = (\{s, f\}, \{a, b, c\}, \{B, C, b, c\#\}, R, s, B, C, \{f\})$$

s nasledujúcimi pravidlami v množine R_3:

$$1[1B|2[1Csa \to Bb|Ccs \qquad\qquad 1[1B|2[1Cs \to f$$

Demonštrácia prijatia reťazca $aabbcc$ automatom ${}_2M_2$:

$$
\begin{array}{lllll}
(s, aabbcc, B\#C\#) & {}_e\Rightarrow & (s, & abbcc, & Bb\#Cc\#) \quad [1[1B|2[1Csa \to Bb|Ccs] \\
& {}_e\Rightarrow & (s, & bbcc, & Bbb\#Ccc\#) \quad [1[1B|2[1Csa \to Bb|Ccs] \\
& {}_e\Rightarrow & (s, & bbcc, & bb\#cc\#) \quad [1[1B|2[1Cs \to f] \\
& {}_p\Rightarrow & (f, & bcc, & b\#cc\#) \\
& {}_p\Rightarrow & (f, & cc, & \#cc\#) \\
& {}_p\Rightarrow & (f, & c, & \#c\#) \\
& {}_p\Rightarrow & (f, & \varepsilon, & \#\#)
\end{array}
$$



Obr. 5.3: Syntaktický strom, zostrojený v príklade 5.2.3, *Simultánnym hlbokým dvoj-zásobníkovým automatom riadeným vstupom* pri prijímaní reťazca $aabbcc$.

Na obrázku 5.3 je zostrojený syntaktický strom, ktorý vznikol pri syntaktickej analýze reťazca z príkladu 5.3. Tento strom sa skladá z dvoch podstromov, ktoré boli vytvorené súčasne. Pričom prvý podstrom vzniká na prvom zásobníku a druhý na druhom. Výhoda oproti prvej variante spočíva v tom, že množina pravidiel obsahuje pravidlá gramatiky typu 3 - regulárnej gramatiky (LLIN), zatiaľčo v prípade jedného zásobníka boli typu 2, teda bezkontextovej.

Opäť sa zamyslime nad implementáciou. Expanzia symbolu v prípade pravidla tvaru $A \to aB$ predstavuje vybratie neterminálneho symbolu A zo zásobníka, a vloženie zásobníkových symbolov a a B na zásobník.

V prípade pravidla tvaru $A \to aBb$ opäť vyberieme symbol A zo zásobníka a vložíme postupne na zásobník symboly a, B, b. Dalo by sa namietať, že v prípade dvoj-zásobníkovej varianty musíme vykonať expanziu na obidvoch zásobníkoch a teda v skutočnosti vkladáme o jeden symbol viac. Berme avšak do úvahy, že táto sekvencia krokov prebehne v jednom výpočetnom kroku a je teda možné uvažovať o viacvláknovej implementácii.

Expanzia neterminálneho symbolu vždy prebiehala len na vrchole zásobníka. Teda potenciál *Hlbokého zásobníkového automatu* nebol využitý. Rovnakým spôsobom by tento reťazec prijal aj *Dvoj-zásobníkový automat* z definície 4.1, ktorého množina pravidiel R by vyzerala nasledovne $B|Csa \to Bb|Ccs$, $B|Cs \to f$. Dokonca v tomto prípade môžeme hovoriť o *Simultánnom dvoj-zásobníkovom automate s jedno-otáčkovými zásobníkmi*, pretože oba zásobníky vykonali iba jednu otáčku a to v rovnakom výpočetnom kroku.

Uveďme si preto príklad, kde využijeme hlavnú výhodu automatu typu nIDDPDA a to expandovanie neterminálneho symbolu.

Príklad 5.2.4. Uvažujme automat typu nIDDPDA ${}_2{}_1M_4$ z definície 4.2.7, ktorý prijíma jazyk $L = a^n b^m c^{n+m} d^n$, kde $n, m \geq 1$.

$$ {}_2{}_1M_4 = (\{s, q_1, q_2, f\}, \{a, b, c, d\}, \{B, C, b, c\#\}, R, s, S, C, \{f\}) $$

s nasledujúcimi pravidlami v množine R_3:

$$
\begin{array}{lll}
1[1Ssa \to Bq_1 & 1[1Bq_1a \to cBdq_1 & 2[1Cq_1b \to Ccq_2 \\
2[1Cq_2b \to Ccq_2 & 2[1Cq_2c \to q_2 & 1[1B2[1\#q_2d \to f
\end{array}
$$

Demonštrácia prijatia reťazca $aaabbccccdddd$ automatom ${}_2M_2$:

$$
\begin{array}{llllll}
(s, aaabbccccdddd, S\#C\#) & & & & & \\
 & {}_e\Rightarrow & (s, & aabbccccdddd, & B\#C\#) & [1[1Ssa \to Bq_1] \\
 & {}_e\Rightarrow & (q_1, & abbccccdddd, & cBd\#C\#) & [1[1Bq_1a \to cBdq_1] \\
 & {}_e\Rightarrow & (q_1, & bbccccdddd, & ccBdd\#C\#) & [1[1Bq_1a \to cBdq_1] \\
 & {}_e\Rightarrow & (q_2, & bccccdddd, & ccBdd\#Cc\#) & [2[1Cq_1b \to Ccq_2] \\
 & {}_e\Rightarrow & (q_2, & ccccdddd, & ccBdd\#Ccc\#) & [2[1Cq_2b \to Ccq_2] \\
 & {}_e\Rightarrow & (q_2, & cccddd, & ccBdd\#cc\#) & [2[1Cq_2c \to q_2] \\
 & {}_p\Rightarrow & (q_2, & cccddd, & cBdd\#cc\#) & \\
 & {}_p\Rightarrow & (q_2, & ccddd, & Bdd\#cc\#) & \\
 & {}_p\Rightarrow & (q_2, & cddd, & Bdd\#c\#) & \\
 & {}_p\Rightarrow & (q_2, & ddd, & Bdd\#\#) & \\
 & {}_e\Rightarrow & (q_2, & dd, & dd\#\#) & [1[1B2[1\#q_2d \to f] \\
 & {}_p\Rightarrow & (f, & d, & d\#\#) & \\
 & {}_p\Rightarrow & (f, & \varepsilon, & \#\#) &
\end{array}
$$

Ako môžeme vidieť automat prijíma jazyk rodiny **RE** 2.3. Pre lepšie pochopenie zostrojovania syntaktického stromu uvádzame obrázok 5.4.

Na zásobníku číslo 2 sa vytvára syntaktický strom, obsahujúci symboly c a d, ktoré sa generujú na základe riadiaceho symbolu a zo vstupu. Zásobník číslo 1 zase obsahuje syntaktický strom so symbolmi c, ktoré boli vygenerované na základe vstupného symbolu b.

Všimnime si pravý strom, ktorý je vygenerovaný na zásobníku číslo 1, obsahuje o jeden symbol c menej ako je počet a a d. Tento symbol je v druhom strome použitý pre odstránenie non-terminálu B. Je to z toho dôvodu, že sme chceli vytvoriť deterministickú verziu nIDDPDA.

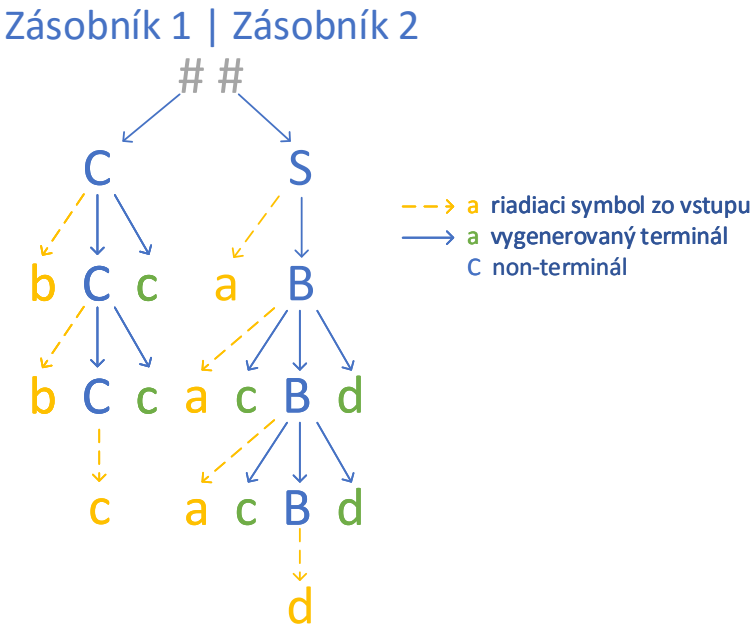
Zostrojenie zásobníkového automatu, ktorý je regulovaný hlbokým zásobníkom

Príklad 5.2.5. Uvažujme zásobníkový automat regulovaný hlbokým zásobníkom M_5 z definície 4.2.10, ktorý prijíma jazyk $L(M_5) = \{a^n b^m c^{n+m} d^n \ : n, m \geq 1\}$:

$$ M_5 = (\{q\}, \{a, b, c, d\}, \{c\}, \{S, 1, 2, 3, 4, 5, A, B\}, R_1, R_\Xi, q, \#, S, \{q\}) $$

s nasledujúcimi pravidlami v množine R_1:

$$
\begin{array}{lll}
1 : qa \to q & 2 : qb \to q & 3 : qc \to q \\
4 : qc \to qc & 5 : cqd \to q &
\end{array}
$$



Obr. 5.4: Syntaktický strom, zostrojený v príklade 5.2.4, *Hlbokým viac-zásobníkovým automatom riadeným vstupom* pri prijímaní reťazca *aaabbccccdddd*.

a riadiacim jazykom Ξ, ktorý generuje frázová gramatika G_Ξ z definície 2.3:

$$G_\Xi = (\{A, B, R, S, X\}, \Psi_{M_5}, P, S)$$

s nasledujúcimi pravidlami v množine P:

$S \to 1S4A5$	$S \to 145$	$54 \to 45$	$5R4 \to 4R5$
$R \to \varepsilon$	$14 \to 1B4$	$B \to 2B3$	

Gramatiku prevedieme na pravidlá hlbokého zásobníkového automatu pre generovanie riadiaceho jazyka. Všimnime si, že pravidlo $S \to 1S4A5$ generuje pravidlá pre akceptovanie symbolov a, c, d. S každým prijatým symbolom a teda vygenerujeme pravidlá pre akceptovanie symbolov c a d. Preto v pravidle môžeme vynechať časť, ktorá generuje symbol a. Upravené pravidlo bude vyzerať nasledovne: $Aa \to A45$.

Pravidlo $B \to 2B3$ generuje symboly b a c. S prijatím symbolu b vygenerujeme pravidlo pre prijatie symbolu c. Upravené pravidlo bude vyzerať nasledovne: $Bb \to B3$.

Z predchádzajúcich dvoch pravidiel odvodíme taktiež pravidlo pre počiatočný symbol S gramatiky: $Sa \to B4A5$. Z tohto pravidla je zrejmé, že symbol A sa bude nachádzať na hlbokom zásobníku v hĺbke 2. Preto výsledné pravidlo bude $2Aa \to A45$. Zvyšné pravidlá slúžia na premiestnenie symbolov na správne miesto. Tie vynecháme.

Automat M_5 riadi jazyk $\Xi = \{1^n2^m\}\{3^m4^n5^n\}$, pričom *prefix* riadiacich pravidiel je určený vstupnými symbolmi, na základe ktorých sa generujú pravidlá pre *suffix* riadiaceho jazyka na druhom zásobníku.

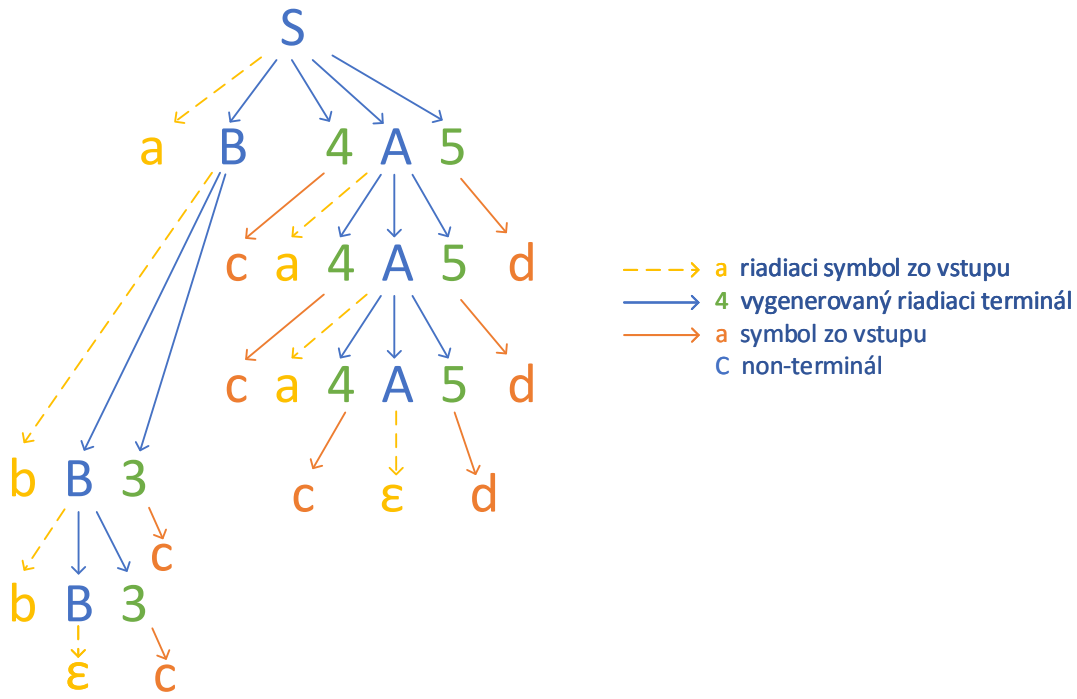
Výsledná konečná množina pravidiel pre generovanie riadiaceho jazyka Ξ_{M_5} automatu M_5:

$R_\Xi = \{1Sa \to B4A5, 2Aa \to 4A5, 2A \to \epsilon, 1Bb \to B3, 1B \to \epsilon\}$

Demonštrácia prijatia reťazca $aaabbccccccddd$ automatom M_5 :

$$
\begin{array}{lllll}
(q, aaabbccccccddd, \#S\#) & {}_e\Rightarrow & (q, & aabbccccccddd, & \#B4A5\#) \quad [1Sa \to B4A5] \\
& {}_e\Rightarrow & (q, & abbccccccddd, & \#B44A55\#) \quad [2Aa \to 4A5] \\
& {}_e\Rightarrow & (q, & bbccccccddd, & \#B444A555\#) \quad [2Aa \to 4A5] \\
& {}_e\Rightarrow & (q, & bbccccccddd, & \#B444555\#) \quad [2A \to \epsilon] \\
& {}_e\Rightarrow & (q, & bccccccddd, & \#B3444555\#) \quad [1Bb \to B3] \\
& {}_e\Rightarrow & (q, & ccccccddd, & \#B33444555\#) \quad [1Bb \to B3] \\
& {}_e\Rightarrow & (q, & ccccccddd, & \#33444555\#) \quad [1B \to \epsilon] \\
& {}_p\Rightarrow & (q, & ccccddd, & \#3444555\#) \quad [3 : qc \to q] \\
& {}_p\Rightarrow & (q, & cccddd, & \#444555\#) \quad [3 : qc \to q] \\
& {}_p\Rightarrow & (q, & ccddd, & c\#44555\#) \quad [4 : qc \to qc] \\
& {}_p\Rightarrow & (q, & cddd, & cc\#4555\#) \quad [4 : qc \to qc] \\
& {}_p\Rightarrow & (q, & ddd, & ccc\#555\#) \quad [4 : qc \to qc] \\
& {}_p\Rightarrow & (q, & dd, & cc\#55\#) \quad [5 : cqd \to q] \\
& {}_p\Rightarrow & (q, & d, & c\#5\#) \quad [5 : cqd \to q] \\
& {}_p\Rightarrow & (q, & \varepsilon, & \#\#) \quad [5 : cqd \to q]
\end{array}
$$

Ako môžeme vidieť automat prijíma jazyk rodiny **RE** 2.3. Pre lepšie pochopenie zostrojovania syntaktického stromu uvádzame obrázok 5.5.



Obr. 5.5: Syntaktický strom, zostrojený v príklade 5.2.5, *Zásobníkovým automatom, ktorý je regulovaný hlbokým zásobníkom* pri prijímaní reťazca $aaabbccccccddd$

Všimnime si v príklade 5.2.5 pravidlá množiny R_1, $1 : qa \to q$ a $2 : qb \to q$. Pre definíciu automatu M_5 nie sú potrebné. V priebehu výpočtu nikdy nebudú použité, pretože riadiaci jazyk negeneruje identifikátory pravidiel 1 a 2, tie sú uvedené iba pre úplnosť.

V prípade, ak by sme dopredu vedeli hodnotu konštánt n a m, tak by sme fázu generovania mohli preskočiť, a pri inicializácii by na zásobník bol vložený riadiaci jazyk Ξ. Potom by množina R_Ξ vyzerala nasledovne: $R_\Xi = \{1S \to \Xi\}$.

Napríklad automat M_5 s riadiacim jazykom $\Xi_{example} = \{11234455\}$ by prijímal jazyk $a^2b^1c^{2+1}d^2$, $R_\Xi = \{1S \to 11234455\}$. Pre predstavu si uveďme príklad.

Príklad 5.2.6. Majme zásobníkový automat regulovaný hlbokým zásobníkom M_5 definovaný v príklade 5.2.4. s riadiacim jazykom $\Xi_{example} = \{1223345\}$, ktorý prijíma jazyk $ab^2c^{1+2}d$, $R_\Xi = \{1S \to \Xi_{example}\}$.

Demonštrácia prijatia reťazca *abbcccd* automatom M_5 s riadiacim jazykom $\Xi_{example}$:

$$
\begin{array}{lrrrl}
(q, abbcccd, \#S\#) & {}_e\Rightarrow & (q, & abbcccd, & \#1223345\#) \quad [1S \to \Xi_{example}] \\
& {}_p\Rightarrow & (q, & bbcccd, & \#223345\#) \quad [1 : qa \to q] \\
& {}_p\Rightarrow & (q, & bcccd, & \#23345\#) \quad [2 : qb \to q] \\
& {}_p\Rightarrow & (q, & cccd, & \#3345\#) \quad [2 : qb \to q] \\
& {}_p\Rightarrow & (q, & ccd, & \#345\#) \quad [3 : qc \to q] \\
& {}_p\Rightarrow & (q, & cd, & \#45\#) \quad [3 : qc \to q] \\
& {}_p\Rightarrow & (q, & d, & c\#5\#) \quad [4 : qc \to qc] \\
& {}_p\Rightarrow & (q, & \varepsilon, & \#\#) \quad [5 : cqd \to q]
\end{array}
$$

Kapitola 6

Implementácia

V nasledujúcej kapitole si popíšeme implementáciu automatov uvedených v predchádzajúcich kapitolách. Transformujeme teoretické modely navrhnutých automatov – *Hlboký viac-zásobníkový automat riadený vstupom*, zavedený v definícii 4.2.7 a *Zásobníkový automat regulovaný hlbokým zásobníkom*, zavedený v definícii 4.2.10 – na reálnu programovú implementáciu.

Na úvod sa oboznámime s architektúrou implementácie a následne sa budeme podrobnejšie venovať samotnej knižnici, v ktorej sú implementované automaty uvedené v tejto práci, a nami novo zavedené varianty *Hlboký viac-zásobníkový automat riadený vstupom*, a *Zásobníkový automat regulovaný hlbokým zásobníkom*.

Pre popis samotnej implementácie automatov je potrebné oboznámenie sa s implementáciou fundamentálnej dátovej štruktúry – zásobníka a jeho modifikácii.

Následne sa hlbšie zameriame na implementáciu jednotlivých automatov a algoritmov syntaktickej analýzy založenej na týchto teoretických modeloch.

Ku koncu kapitoly sa budeme venovať stručnému popisu doplnkových aplikácií, ktoré pracujú s nami implementovanou knižnicou a sú určené pre užívateľa.

Na záver popíšeme testovanie a experimentovanie s implementovanými modelmi.

6.1 Architektúra

Pri implementácii bola zvolená trojvrstvová architektúra.

Prezenčná vrstva ponúka dve aplikácie a to grafické užívateľské rozhranie, ktoré vizualizuje činnosť automatov – `Automata Simulator` a konzolovú aplikáciu – `textAutomata`, ktorá ponúka textový výpis činnosti automatov.

Aplikačnú vrstvu tvorí knižnica – `automata-lib` implementujúca teoretické modely automatov a na nich založenú syntaktickú analýzu.

Dáta sú spracovávané pomocou *dátovej vrstvy*, ktorá deserializuje formát `JSON` na dátové štruktúry, s ktorými pracuje aplikačná vrstva – knižnica a naopak. Taktiež validuje dané vstupy. Tento modul je implementovaný pomocou návrhového vzoru *Singleton*.

6.2 Knižnica automatov

V tejto sekcii bude popísaná architektúra knižnice – stručný popis jednotlivých tried a závislostí medzi nimi. Bližšie sa zameriame na algoritmy implementujúce činnosť automatov, ako napríklad algoritmus výberu pravidla v derivačnom kroku automatu či algoritmus expanzie

symbolu v hlbokom zásobníku. Zdôvodníme vhodnosť použitia dátových štruktúr reprezentujúcich zásobníky, dátových štruktúr uchovávajúcich množinu pravidiel automatu atď. Ako implementačný jazyk bol zvolený programovací jazyk `C++`, štandard `C++11`.

6.2.1 Implementácia zásobníkov

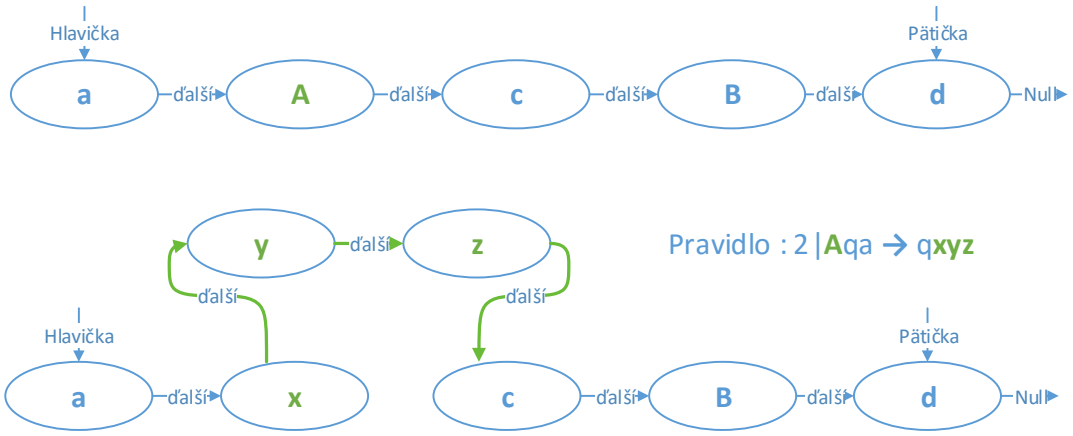
Táto časť popisuje zásobník, ktorý je fundamentálnou dátovou štruktúrou tejto aplikácie.

Ako sme si mohli všimnúť práca sa zameriava na automaty, ktoré využívajú hlboké zásobníky. Pripomeňme si, že tento zásobník okrem štandardných operácií, ktoré menia obsah zásobníka – *pop* a *push* – dovoľuje operáciu *expanzia* v hĺbke zásobníka.

Ak použijeme klasickú dátovú štruktúru zásobník pre reprezentáciu hlbokého zásobníka, tak by vykonávanie operácie *expanzie* v hĺbke n znamenalo odobratie všetkých symbolov na zásobníku, teda n-krát operácia *top*, n-krát operácia *pop*, n-krát prístup do pamäti a uloženie n symbolov do pomocnej štruktúry. Následne by sa n-krát vykonala operácia *push*.

Je zrejmé, že vyššie popísaný spôsob implementácie je neefektívny, a je teda nevhodné pre reprezentáciu hlbokého zásobníka použiť dátovú štruktúru zásobník z dôvodu vykonávania operácie expanzie v hĺbke zásobníka.

Ukážme si, ako by vyzerala operácia *expanzia* v prípade využitia dátovej štruktúry list. Symbol ktorý chceme expandovať, nahradíme symbolom, ktorý má byť najhlbšie z novo vkladaných symbolov v zásobníku. Ten bude ukazovať na ďalší novo vložený symbol a atď. Posledný vložený symbol bude ukazovať na následníka pôvodného symbolu, ktorý sa expandoval. Táto úvaha je zobrazená na obrázku 6.1.



Obr. 6.1: *Expanzia hlbokého zásobníka*, ktorý je implementovaný dátovou štruktúrou list. Vyššie zobrazený list reprezentuje stav pred vykonaním pravidla – expanzie. Za dno zásobníka považujeme prvok na ktorý ukazuje hlavička.

Klasický zásobník

Z vyššie uvedených dôvodov sme pre implementáciu zásobníka zvolili dátovú štruktúru list. Keďže štandardná knižnica jazyka c++ `stdlib` neposkytuje u dátových kontajnerov virtuálny deštruktor, t.j. nie je z implementačného hľadiska bezpečné z nich dediť, implementovali sme triedu `Pushdown`, ktorá zapúzdruje dátový kontajner `std::list`. Nad touto triedou je implementované klasické rozhranie zásobníka.

Konečný otáčkový zásobník

Konečný otáčkový zásobník je implementovaný triedou `FiniteTurnPushdown`, ktorá dedí od triedy `Pushdown`. Pri každej aplikácii metódy `Pushdown::pop` a `Pushdown::push`, kontroluje, či nenastala v danom derivačnom kroku otáčka. Ak nastala, tak či je daná otáčka v limite konečného maximálneho počtu otáčok.

Hlboký zásobník

Hlboký zásobníkový automat dedí od triedy `Pushdown` a má naviac implementované metódy pre expanziu zásobníka. Aby pri zisťovaní v každom derivačnom kroku, či je možné vykonať pravidlo expanzie, nemusel prehľadávať samotný zásobník, tak trieda vlastní členskú premennú `std::vector<typename std::list<TElement>::iterator> _expansibleElements`, ktorá si uchováva iterátory ukazujúce na prvky v zásobníku, ktoré je možné expandovať. Index v kontajneri `std::vector` značí hĺbku možnej expanzie symbolu.

Viac-zásobník a hlboký viac-zásobník

Viac-zásobník obsahuje kontajner, ktorý uchováva zásobníky `std::vector<TPushdown> _pushDowns`. Pri vytváraní inštancie triedy `NPushdowns` špecifikujeme šablónovým parametrom `TTPushdown` aký typ zásobníkov má tento objekt obsahovať. Index vo vektore značí identifikátor zásobníka.

Táto trieda implementuje napríklad metódu `top`, ktorá vracia vektor elementov, ktoré sa nachádzajú na vrcholoch jednotlivých zásobníkov. V tejto triede je preťažený operátor `[]`, pre priamy prístup k vybranému zásobníku.

Vzťahy medzi triedami, ktoré implementujú zásobníky je možné vidieť na diagrame tried na obrázku A.1, ktorý sa nachádza v prílohách.

6.2.2 Implementácia automatov

V nasledujúcej časti budú popísané implementované automaty a na nich založená syntaktická analýza. Všetky implementované automaty sú deterministické a povoľujú ϵ-prechod. Avšak iba v prípade, ak žiadne pravidlo so vstupným symbolom nie je možné aplikovať a ϵ-pravidlo, ktoré je možné aplikovať v aktuálnom derivačnom kroku, sa nachádza v množine pravidiel.

V prípade automatov s jedným a viac zásobníkmi je vstup prijímaný koncovým stavom a zároveň prázdnym zásobníkom/zásobníkmi a zároveň prijatím celého vstupu.

Predkom každej z implementovaných tried je základná abstraktná trieda `Automata`, ktorá definuje metódy, ktoré musí každý automat implementovať. Inými slovami definuje spoločné správanie automatov.

Obsahuje dátové kontajnery, v ktorých sú uložené základné prvky, ktoré z časti definujú každú modifikáciu automatu, ktorou sme sa zaoberali. Obsahuje množinu vstupných symbolov Σ, množinu stavov Q, počiatočný stav q_0 a množinu koncových stavov F.

Pre uchovanie množín automatu bol zvolený kontajner `std::unordered_set` a to z toho dôvodu, aby implementácia čo najviac kopírovala formálny model.

V základnej triede `Automata` je implementované stavové riadenie, popísané algoritmom 1, využívané všetkými implementovanými automatmi. Zásobníkové automaty toto

riadenie rozširujú o kontrolu na prázdny zásobník a aplikáciu ε-pravidiel. Všimnime si vyznačený riadok číslo 4 a 8. V nasledujúcom texte, pri popise niektorých implementovaných automatov, bude uvádzaný pseudokód vykonania derivačného kroku, volaného z tohto miesta.

Ďalej trieda `Automata` implementuje pomocné metódy pre zistenie, či je aktuálny stav *konečný*, poprípade *sink* stav.

Algoritmus 1 Vykonaj výpočet automatu na vstupe

Vstup: Sekvencia vstupných symbolov *vstup*
Výstup: Automat prijal/neprijal

```
1:  if vstup = ε and currentState ∈ finalStates then
2:      return Prijal
3:  for each a ∈ vstup do
4:      Vykonaj derivačný krok so vstupným symbolom a
5:      if currentState = sink then
6:          return Neprijal
7:      while Zapamätaný symbol a and currentState ≠ sink do
8:          Vykonaj derivačný krok so vstupným symbolom a
9:  if currentState ∈ finalStates then
10:     return Prijal
11: else
12:     return Neprijal
```

Implementácia výberu aplikovateľného pravidla

Ako sme si mohli všimnúť základná trieda neobsahuje dátovú štruktúru pre pravidlá a to z toho dôvodu, že každý automat má iný tvar pravidla.

Pre uchovávanie pravidiel bol zvolený dátový kontajner mapa – `std::map`. Pri implementácii boli zvolené dva prístupy.

1. Kľúč je vytvorený z ľavej strany pravidla a je mapovaný na pravú stranu pravidla. Majme pravidlo pre zásobníkový automat, tvaru $Aqa \to pBc$. Kľúč v mape pre toto pravidlo bude vytvorený zo zásobníkového symbolu A, stavu q a vstupného symbolu a. Hodnota, ktorá je namapovaná na tento kľúč bude pozostávať zo štruktúry reprezentujúcej pravú stranu pravidla.

   Tento prístup je využitý pri konečnom automate, zásobníkovom automate, zásobníkovom automate s konečným počtom pravidiel a hlbokom zásobníkovom automate.

2. Pravidlá sú taktiež uložené v mape, no kľúčom je kombinácia hodnôt - vstupný symbol a stav. Hodnota, ktorá je namapovaná na tento kľúč, bude pozostávať z vektoru, ktorý obsahuje štruktúry reprezentujúce pravú stranu pravidla a zvyšné entity z ľavej strany pravidla. Teda napríklad majme pravidlá $Xqa \to pY$ a $Zqa \to sV$. Potom kľúč v mape bude dvojica hodnôt q, a a hodnota, reprezentujúca pravé strany pravidiel, bude vektor obsahujúci dva prvky $\{\{X, Y, p\}, \{Z, V, s\}\}$. Druhý prístup sa využíva u variantách automatov s viacerými zásobníkmi.

Na úkor jednoduchšieho vyhľadávania nastáva pri obidvoch spôsoboch istá redundancia uložených dát oproti klasickým metódam syntaktickej analýzy.

Konečný a zásobníkový automat

Na týchto základných verziách automatov je postavená teória informatiky. Keďže modifikácie automatov, uvedené v tejto práci, vychádzajú z týchto formálnych modelov bolo nutné tieto automaty implementovať. Vhodným využitím základného princípu objektovo orientovaného programovania – dedičnosti sú následne tieto elementárne prvky použité ako základný kameň pre zložitejšie modely.

Trieda `FiniteStateMachine` implementuje teoretický model *Konečný automat* uvedený v definícii 2.4.1 a trieda `PushdownAutomata` implementuje teoretický model *Zásobníkový automat* uvedený v definícii 2.4.5.

Zásobníkový automat s konečným počtom otáčok

Trieda `PushdownAutomata`, inštanciovaná so šablónovým parametrom `FiniteTurnPushdown`, implementuje teoretický model uvedený v definícii 3.2.

Hlboký zásobníkový automat

Trieda `DeepPushdownAutomata` implementuje teoretický model uvedený v definícii 3.3.1. Hlboký zásobníkový automat je priamym potomkom zásobníkového automatu. Líši sa tvarom pravidiel a typom zásobníka, ktorý je špecifikovaný šablónovým parametrom a výberom a aplikovaním pravidla v derivačnom kroku. Naviac implementuje aplikovanie pravidla pre expanziu symbolu v hĺbke zásobníka. V prípade ak sa na vrchole zásobníka nachádza aktuálne spracovávaný symbol, tak sa aplikuje pravidlo *pop*, podľa definície Hlbokého zásobníkového automatu. Vykonanie derivačného kroku týmto automatom je popísané algoritmom 2.

Algoritmus 2 Vykonanie derivačného kroku *Hlbokého zásobníkového automatu*

Vstup: Vstupný symbol a

```
1:  if pushdown.TOP = a then
2:      pushdown.POP
3:  else
4:      for each rule ∈ R do
5:          if rule.key = {a, currentState} then
6:              if pushdown.ISEXPANSIBLEINDEPTH(rule.expansSymbol, rule.depth) then
7:                  currentState ← rule.leftSide.state
8:                  pushdown.DOEXPANSION(rule)
9:                  return
10: if pushdown.NOTEMPTY() then
11:     currentState ← sinkstate
```

Viac-zásobníkový automat

Trieda `NPushdownAutomata`, inštanciovaná so šablónovým parametrom `Pushdown`, implementuje teoretický model uvedený v definícii 4.1.1.

Tento automat vykonáva rovnaké činnosti ako zásobníkový automat, avšak na viacerých zásobníkoch. Automat v jednom derivačnom kroku môže brať do úvahy

- *vrchol každého zásobníka*, v tom prípade musí byť pravidlo tvaru $1]A_12]A_2 \dots n]A_npa \to 1]B_12]B_2 \dots n]B_nq$, kde $A_1, A_2, \dots, A_n \in \Gamma, B_1, B_2, \dots, B_n \in \Gamma \cup \varepsilon, p, q \in Q$ a $a \in \Sigma \cup \varepsilon$. V prípade ak sa symboly $A_1, A_2, \dots, A_n$ nachádzajú na vrcholoch zásobníkov $1, 2, \dots, 3$, je možné toto pravidlo aplikovať a prepísať tieto symboly podľa pravej strany pravidla, teda na symboly $B_1, B_2, \dots, B_n$,

- *aspoň jeden vrchol niektorého zo zásobníkov*, pravidlo je rovnakého tvaru ako v predchádzajúcom bode, avšak $A_1, A_2, \dots, A_n \in \Gamma \in \varepsilon$ a musí platiť $A_1 \neq \varepsilon \vee A_2 \neq \varepsilon \vee \dots \vee A_n \neq \varepsilon$. V tomto bode je zahrnutá verzia viac-zásobníkového automatu, ktorý nie je simultánny a teda sa berie do úvahy iba jeden vrchol zásobníka,

- *žiadny vrchol zo zásobníkov*, teda pravá strana pravidla obsahuje iba stav a vstupný symbol. Ľavá strana pravidla môže vkladať symboly na zásobníky.

Z vyššie uvedených bodov vyplýva, že automat implementuje simultánnu verziu. Je teda možné meniť obsah viacerých zásobníkov v jednom derivačnom kroku, a zároveň, je schopný vykonávať aj prechody nesimultánnej varianty. Simultánnosť viac-zásobníkového automatu je popísaná v časti 4.1.1.

Hlboký viac-zásobníkový automat riadený vstupom

Trieda `NIDDPushdownAutomata` implementuje teoretický model, ktorý bol zadefinovaný v definícii 4.2.7.

Pravidlá *Hlbokého viac-zásobníkového automatu riadeného vstupom*, ďalej len IDDPDA, môžeme rozdeliť do štyroch kategórií

1. *Pop* pravidlá. Ako sme uviedli v definícii 4.2.9, IDDPDA ponúka dva mechanizmy vykonávania operácie *pop*. Boli implementované obe varianty. V konštruktore IDDPDA sa podľa množiny pravidiel zisťuje, ktorý typ sa bude vykonávať nasledovne:

   - Ak sa v množine R nenachádza žiadne pravidlo tvaru $n]aqa \to p$, kde $a \in \Sigma$ a $q, p \in Q$, potom automat bude vykonávať *implicitné pop pravidlá*. V prípade, ak je možné vykonať *pop* operáciu na viacerých zásobníkoch, teda viacero vrcholov zásobníkov obsahuje terminálny symbol, potom sa operácia môže vykonať iba na vrchole s najnižším identifikátorom. Táto podmienka zabezpečuje determinizmus automatu. Pri tejto variante *pop* operácie nie je možné meniť stav automatu.
   - Ak sa v množine R nachádza aspoň jedno pravidlo tvaru $n]aqa \to p$, vykonávajú sa explicitné *pop* operácie. Teda pre každý symbol, na ktorý chceme túto operáciu aplikovať, musí byť uvedené toto pravidlo. Pri tejto variante *pop* operácie je možné meniť stav automatu.

2. *Klasické* pravidlá – prepísanie neterminálneho symbolu je možné vykonať na vrchole zásobníka, t.j. neobsahujú na ľavej strane pravidla hĺbku. Do tejto kategórie zahrnieme aj explicitné *pop* pravidlá.

3. Pravidlá *expanzie*.

4. ε-pravidlá. V tejto kategórii sú zahrnuté *klasické* a *expanzné pravidlá*, ktoré v ľavej časti pravidla neobsahujú vstupný symbol.

Pre množinu pravidiel bola vytvorená dátová štruktúra, ktorá uchováva *klasické* pravidlá, tvaru viac-zásobníkového automatu `NPushdownAutomata` a *expanzné* pravidlá. Klasické pravidlá, sú uložené v členskej premennej triedy predka, teda `NPushdownAutomata`, kde sú aj implementované metódy pre vyhľadávanie nad touto množinou a aplikovanie nájdených pravidiel. Množina pravidiel je rozdelená z dôvodu efektívnosti. Zisťovanie, či je možné aplikovať expanzné pravidlo je náročnejšie.

Pri implementácii derivačného kroku bola zavedená postupnosť, v akej sa vyhľadáva aplikovateľné pravidlo, popísaná algoritmom 3.

Algoritmus 3 Vykonanie derivačného kroku *IDDPDA*

Vstup: Vstupný symbol a

if Je možné aplikovať implicitné *pop* pravidlo pre a **then**
    POP a
    return
if Je možné aplikovať klasické/simultánne pravidlo na $1/n$ vrchole/vrcholoch **then**
    Aplikuj pravidlo
    return
if Je možné aplikovať pravidlo *expanzie* **then**
    Aplikuj pravidlo
    return
if $a \neq \varepsilon$ **a** existuje aplikovateľné ε-pravidlo **then**
    Rekurzívne volanie tejto funkcie s parametrom ε, zapamätaj si symbol a
    return
$currentState \leftarrow sinkstate$

Zásobníkový automat regulovaný hlbokým zásobníkom

Trieda `RegulatedPDAutomata` implementuje teoretický model *Zásobníkový automat regulovaný hlbokým zásobníkom*, ďalej len RPDA, uvedený v definícii 4.2.10. Predkom tohoto automatu je zásobníkový automat s klasickým zásobníkom, okrem ktorého RPDA vlastní hlboký zásobník `DeepPushdown`, ktorý je *riadiaci* a na ňom sa generuje riadiaci jazyk.

Automat vykoná v derivačnom kroku vždy jednu z nasledujúcich akcií

- generovanie riadiaceho jazyka, ktoré má v derivačnom kroku prednosť. Množina riadiacich pravidiel obsahuje pravidlá *expanzie*, pričom pri generovaní automat nemení stav. Ľavá a pravá strana pravidiel neobsahuje položku stav.
- aplikovanie pravidiel, ktoré sa vykoná v prípade, ak nie je možné uplatniť žiadne z generovacích pravidiel. Ak sa nachádza na vrchole riadiaceho zásobníka identifikátor pravidla, riadenie automatu zistí, či je pravidlo aplikovateľné v danej konfigurácii a automat vykoná prechod podľa tohto pravidla.

Činnosť vykonaná v derivačnom kroku automatu RPDA je uvedená v algoritme 4.

Vzťahy medzi jednotlivými triedami sú zobrazené diagramom tried na obrázku A.2 uvedenom v prílohách.

Algoritmus 4 Vykonanie derivačného kroku *RPDA*

Vstup: Vstupný symbol a

1: **if** Na $regPushdown$ je aspoň jeden expandovateľný symbol **then**
2: **if** Je možné generovať riadiaci jazyk so symbolom a **then**
3: Generuj pravidlo
4: **return**
5: **if** Je možné generovať riadiaci jazyk so symbolom ε **then**
6: Generuj pravidlo
7: Zapamätaj si symbol a
8: **return**
9: $ruleId \leftarrow regPushdown.$popAndTop()
10: $lRule \leftarrow rules[ruleId].leftSide$
11: **if** $lRule.inputSymbol = \varepsilon$ **then**
12: Zapamätaj si symbol a
13: **else**
14: **if** $lRule.inputSymbol \neq a$ **then**
15: $currentState \leftarrow sinkstate$
16: **return**
17: **if** $lRule.state = currentState$ **then**
18: **if** $lRule.pushdownSymbol = pushdown.$top() **or** $lRule.pushdownSymbol = \varepsilon$ **then**
19: Aplikuj pravú stranu pravidla $rules[ruleId].rightSide$
20: **return**
21: $currentState \leftarrow sinkstate$

6.3 Grafická aplikácia Automata Simulator

Grafická aplikácia Automata Simulator je jednoduchý program, ktorý obsahuje užívateľské rozhranie pre vizualizáciu výpočtu automatov implementovaných v knižnici Automata-lib. Táto aplikácia je naprogramovaná v jazyku `C++` s využitím frameworku `Qt`.

Aplikácia má dve obrazovky. Prvá očakáva zadanie vstupov - definície automatu a vstupného reťazca, na ktorom prebehne výpočet automatu. Na druhej obrazovke prebieha simulácia. Na základe vstupov sa vytvoria zásobníky a tlačítkom *Next step* je možné prezerať derivačné kroky.

Definícia automatov

Definíciu automatu je možné vložiť dvomi spôsobmi. Prvá možnosť ponúka vyplnenie definície priamo v programe do vstupných textových elementov. Pre pohodlnejšiu prácu bola implementovaná možnosť nahrať definíciu zo súboru formátu `JSON`. Pred nahratím súboru nie je potrebné vyberať zo zoznamu aký typ automatu sa v súbore nachádza, program je túto informáciu schopný detekovať. Nahraný súbor je vyplnený do vstupných polí a teda je možné definíciu ľubovoľne upravovať. Presné položky a typy záznamov definícií pre každý automat, sú uvedené v príkladoch priložených ku zdrojovým kódom. Príklad súboru pre *Viaczásobníkový automat* je uvedený v prílohách v časti C.0.1. Definíciu je tiež možné uložiť do súboru.

Rovnaká funkcionalita je implementovaná pre vstupný reťazec. Teda reťazec je možné priamo napísať alebo nahrať zo súboru.

Bližšie informácie o použití a funkcionalite grafickej aplikácie `Automata Simulator`, je možné nájsť v prílohách, v časti B.3, kde sa tiež nachádzajú ukážky aplikácie na obrázkoch B.1 a B.2.

6.4 Testovanie a experimenty

V tejto časti sa zameriame na testovanie a experimentovanie s knižnicou, ktorá obsahuje implementované automaty.

Testovanie

Každý modul bol priebežne testovaný počas vývoja.

Pre overenie, či sa implementované modely automatov správajú ako formálny model, boli vytvorené definície automatov a vstupy, ktoré dané automaty prijímajú resp. neprijímajú. Na základe definície bol zostrojený automat, ktorý vykonal výpočty nad množinou vstupov, v ktorej sa nachádzali rôzne obtiažne reťazce. Výstupom testu bola odpoveď, či automat vstup prijal alebo neprijal.

Tieto definície sú uchované v dvoch formách. Popísané vo formáte `JSON`, ukážka je uvedená v prílohách C.0.1 a uložené priamo v dátových štruktúrach jazyka `C++`.

Dôvod, prečo bola zvolená aj druhá reprezentácia dát je, že modely automatov implementujú triedy, ktoré nemajú špecifikované dátové typy, sú to takzvané šablónové triedy. Preto testovanie automatov prebehlo s rôznymi dátovými typmi `Stavov` a `Symbolov`. Z jednoduchých dátových typov boli zvolené typy `int`, `char`, `string` a `enum`. Pre účel otestovania užívateľom definovaného dátového typu bola zadefinovaná jednoduchá trieda `Point`, ktorá obsahuje dve premenné `x, y`.

Automatické testy sú súčasťou konzolovej aplikácie `textAutomata` a je ich možné spustiť s argumentom programu `--test`.

Experimenty

Neodmysliteľnou súčasťou experimentov bolo vytváranie testovacích dát. Definície automatov boli vytvárané s úmyslom skúmať vlastnosti a správanie automatov.

Hlboký viac-zásobníkov automat

V tejto časti si popíšeme experimentovanie s *Hlbokým viac-zásobníkovým automatom.*

Prvé experimenty boli vykonané s automatmi, ktorých definície a ukážka prijatia reťazca bola uvedená v príkladoch, v časti syntaktickej analýzy 5. Nami implementovaný automat vykonal rovnakú sekvenciu krokov, ako je ukázané v príkladoch 5.2.3, 5.2.4.

Ďalšie experimentovanie malo za úlohu odhaliť rozdiely simultánnej a klasickej verzie tohto automatu. Na obrázku 6.4 sú zobrazené definície automatov M_1 a M_2 typu nIDDPDA. Všimnime si mohutnosť množiny pravidiel *Rules*. V prípade simultánnej verzie je mohutnosť 2, zatiaľčo v klasickej verzii $|Rules| = 4$. Obidva automaty prijímajú rovnaký jazyk $a^n b^n c^n$.

```
Symbols : { a , b , c }
States : { f , s }
State : s
Rules : {

1]1|B2]1|Csa −−> 1]bB2]cCs
1]1|B2]1|Cs −−> f

}
Final States : { f }
Pushdown symbols : { B , C , b , c }
Init pushdown symbols : B C
Count of pushdowns : 2
```

(a) Simultánna verzia automatu typu nIDDPDA, automat M_1

```
Symbols : { a , b , c }
States : { d , e , f , s }
State : s
Rules : {
1]1|Bsa −−> 1]bBd
2]1|Cd −−> 2]cCs
2]1|Cs −−> e
1]1|Be −−> f
}
Final States : { f }
Pushdown symbols : { B , C , b , c }
Init pushdown symbols : B C
Count of pushdowns : 2
```

(b) Klasická verzia automatu typu nIDDPDA, automat M_2

Obr. 6.2: Zrovnanie definícií automatov typu nIDDPDA, prijímajúcich jazyk $a^n b^n c^n$.

Okrem rozdielu v počte pravidiel, klasická verzia vykoná o 3 derivačné kroky viac pri prijímaní reťazca *aabbcc*.

S každým prijatím symbolu a sa v automate M_1 generoval symbol b a c. V prípade prijatia symbolu a automatom M_2 je vygenerovaný iba symbol b a automat musí prejsť do ďalšieho stavu, aby mohol generovať symbol c.

Ďalší prechod, ktorý urobí M_2 oproti verzii M_1 naviac je mazanie symbolu C z druhého zásobníka, ktoré automat M_1 odstráni zo zásobníka v jednom derivačnom kroku spolu so symbolom B.

Z vyššie uvedeného vyplýva, že automat M_2, pri prijímaní rovnakého reťazca vykoná o $n + 1$ viac derivačných krokov ako automat M_1, čo môže byť pomerne neefektívne, ak sú na vstupe dlhé vstupné reťazce.

```
a || 1]B2]Csa -> 1]bB2]cCs || Bb || Cc

a || 1]B2]Csa -> 1]bB2]cCs || Bbb || Ccc


   || 1]1|B2]1|Cs -> f || bb || c c
b || Pushdown 0 pop || b || cc
b || Pushdown 0 pop || || cc
c || Pushdown 1 pop || || c
c || Pushdown 1 pop || ||
```

(a) Derivačné kroky simultánneho automatu M_1 typu nIDDPDA

```
a || 1]Bsa -> 1]bBd || Bb || C
   || 2]Cd -> 2]cCs || Bb || Cc
a || 1]Bsa -> 1]bBd || Bbb || Cc
   || 2]Cd -> 2]cCs || Bbb || Ccc
   || 2]1|Cs -> e || Bbb || cc
   || 1]1|Be -> f || bb || cc
b || Pushdown 0 pop || b || cc
b || Pushdown 0 pop || || cc
c || Pushdown 1 pop || || c
c || Pushdown 1 pop || ||
```

(b) Derivačné kroky automatu M_2 typu nIDDPDA

Obr. 6.3: Porovnanie sekvencie derivačných krokov klasickej a simultánnej verzie automatu nIDDPDA pri prijímaní reťazca *aabbcc*. Derivačné kroky sú zapísané vo formáte: `aktuálny vstupný symbol || uplatnené pravidlo || obsah zásobníka č.1 || obsah zásobníka č.2`.

Zásobníkový automat regulovaný hlbokým zásobníkom

V tejto časti si popíšeme experimentovanie so *Zásobníkovým automatom regulovaným hlbokým zásobníkom.*

Prvé experimenty boli vykonané s automatmi, ktorých definície a ukážka prijatia reťazca bola uvedená v príkladoch, v časti syntaktickej analýzy 5. Nami implementovaný automat vykonal rovnakú sekvenciu krokov, ako je ukázané v príkladoch 5.2.5 a 5.2.6.

Pri práci s týmto typom automatu bolo pozorované, že pre prijatie jazyka, ktorý nie je bezkontextový, je možné obmedziť pracovný zásobník. Čo je možné pozorovať v príklade 5.2.5, kde automat M_5 ani v jednom derivačnom kroku nezmenil svoj stav a ani na pracovný zásobník nevložil žiadny neterminálny symbol.

Preto by bolo vhodné skúmať verziu Čistého zásobníkového automatu regulovaného hlbokým zásobníkom, čo by znamenalo, že pracovný zásobník by mohol obsahovať iba vstupné symboly, teda $\Gamma \subseteq \Sigma$. Takýto typ zásobníka sa nazýva čistý zásobník.

Poprípade by sme mohli vypustiť zmenu stavu v derivačnom kroku a implementovať bezstavovú verziu tohto automatu.

Kapitola 7

Záver

Cieľom tejto diplomovej práce bolo zaviesť nové verzie zásobníkových automatov, ktoré prijímajú jazyky, ktoré nie sú bezkontextové. Práca bola zameraná predovšetkým na *Viac-zásobníkové automaty* a *Hlboké zásobníkové automaty*. Z týchto dvoch formálnych modelov vychádzajú nami zadefinované modifikácie.

K definovaniu a pochopeniu nových automatov bolo nevyhnutné čitateľovi predstaviť teóriu formálnych jazykov, ktorej sa venuje kapitola 2. Na jednoduchý úvod teórie formálnych jazykov postupne naväzovali zložitejšie definície. Bola stručne predstavená Chomského klasifikácia gramatík, ktorá nás sprevádzala celou prácou.

Po gramatikách, ktoré generujú jazyky, boli uvedené automaty – zariadenia pre prijatie jazykov. Bol predstavený najjednoduchší konečný automat, automat rozšírený o zásobník a Turingov stroj.

V kapitole 3 boli uvedené špeciálne varianty zásobníkových automatov. Ako prvý bol predstavený *Regulovaný zásobníkový automat*, kde výber pravidla určuje riadiaci jazyk. Vyjadrovacia sila tohto typu automatu závisí od zvoleného riadiaceho jazyka. Ak je riadiaci jazyk lineárny, automat dokáže akceptovať rekurzívne jazyky.

Ďalej bola prezentovaná verzia zásobníkového automatu s konečným počtom otáčok. Toto obmedzujúce pravidlo limituje počet vykonaných otáčok, čo zjednodušuje mechanizmus automatu. Vďaka tomuto obmedzeniu je možné skorej zamietnuť vstupný reťazec, ktorý nepatrí do prijímacieho jazyka automatu.

Túto kapitolu ukončil hlboký zásobníkový automat, ktorého zásobníková abeceda je rozdelená na terminálne a neterminálne symboly. Bežný zásobníkový automat môže vybrať, čítať a modifikovať len položku na vrchole zásobníka, zatiaľ čo hlboký zásobníkový automat môže čítať, poprípade nahradiť neterminálne symboly, umiestnené hlbšie v zásobníku.

Ďalej bolo diskutované, aký má vplyv riadiť hlboký zásobníkový automat vstupom. Čo znamená, brať do úvahy pri výbere pravidla v danom derivačnom kroku aj aktuálny vstupný symbol. Bola zmienená možnosť mazacích pravidiel a jej vplyv na prijímaciu silu tohto automatu.

Predstaveniu teórie *viac-zásobníkových automatov* sa venuje kapitola 4, kde môžeme nájsť definíciu *dvoj-zásobníkového automatu*. V tejto časti je poskytnutý dôkaz, že zásobníkové automaty, s dvomi a viac zásobníkmi, majú rovnakú výpočetnú silu ako *Turingov stroj*.

Inšpiráciou pre vytvorenie *Hlbokého viac zásobníkového automatu riadeného vstupom* boli práve *Viac-zásobníkové automaty* a *Hlboký zásobníkový automat*.

Druhá zavedená modifikácia vznikla z troch existujúcich automatov. Prvotnou myšlienkou bolo vytvoriť regulovaný viac-zásobníkový automat. Pri pozorovaní spôsobu generova-

nia riadiaceho jazyka gramatikou, sme našli spoločné črty s vykonávaním operácie expanzia v hlbokom zásobníku. Preto sme sa rozhodli zaviesť *Zásobníkový automat regulovaný hlbokým zásobníkom.*

Pre zostrojenie syntaktickej analýzy bolo potrebné popísať prekladač a jeho činnosť – preklad, ktorý prebieha v šiestich fázach, pričom bolo v tejto časti dôležité zamerať sa na syntaktickú analýzu.

Po teoretickej časti nasledovala demonštrácia činnosti zadefinovaných modifikácií zásobníkových automatov a zostrojenie syntaktického analyzátora založeného na týchto modifikáciách. Bolo ukázané, že tieto automaty majú väčšiu vyjadrovaciu silu ako normálny zásobníkový automat a, že ich prijímaný jazyk je rekurzívne vyčísliteľný.

V praktickej časti tejto práce sa podarilo úspešne transformovať teoretické modely navrhnutých automatov na reálnu programovú implementáciu. Bola implementovaná knižnica, ktorá obsahuje implementáciu syntaktickej analýzy založenej na automatoch uvedených v teoretickej časti.

Popis implementácie a testovania knižnice automatov sa nachádza v kapitole 6. Na úvod je čitateľ oboznámený s architektúrou implementácie. Nasleduje popis fundamentálnej dátovej štruktúry, teda zásobníka a jeho modifikácií. Následne je hlbšie prezentovaná implementácia samotných automatov.

Kapitolu ukončuje časť, ktorá popisuje testovanie a experimentovanie s výslednou knižnicou. V experimentoch bola diskutovaná simultánna verzia *Hlbokého viac zásobníkového automatu riadeného vstupom* a jej výhody oproti nesimultánnej verzii.

Pri zostrojovaní a následnom experimentovaní so *Zásobníkovým automatom, ktorý reguluje hlboký zásobník* bolo pozorované, že abecedu symbolov pracovného zásobníka by bolo možné obmedziť na vstupnú abecedu. To by poskytlo možnosť zjednodušiť mechanizmus tohto typu automatu. Poprípade vynechať zmenu stavu a zaviesť bezstavovú verziu.

Pre prácu s knižnicou bolo vytvorené grafické užívateľské rozhranie, ktoré poskytuje vizualizáciu výpočtu automatov. Užívateľ má k dispozícii aj konzolovú aplikáciu, ktorá ponúka záznamy derivačných krokov v textovej podobe.

Ďalším možným pokračovaním tejto diplomovej práce by mohlo byť zavedenie *Hlbokého viac-zásobníkového automatu s konečným počtom otáčok* a skúmanie vplyvu tohto obmedzenia na zložitosť pravidiel a v neposlednom rade na výpočetnú silu.

V prípade *Zásobníkového automatu, ktorý reguluje hlboký zásobník* je možné smerovať budúci výskum na bezstavové verzie alebo verzie, ktoré obmedzujú zásobníkovú abecedu pracovného zásobníka. Zaujímavé by bolo uplatniť tento typ automatu pri zotavovaní sa z chýb behom syntaktickej analýzy.

Myslím, že má zmysel pokračovať v skúmaní tejto problematiky a analyzovaní značne komplexnejších jazykov s využitím uvedených modifikácií hlbokých zásobníkových automatov.

Literatúra

[1] Češka, M.: *Teoretická informatika*. Učební texty FIT VUT v Brně, 2002.
URL http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/ti.pdf

[2] Kolář, D.; Meduna, A.: Regulated Pushdown Automata. *Acta Cybernetica*, ročník 2000, č. 4, 2000: s. 653–664, ISSN 0324-721X.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6020

[3] Leupold, P.: How to Pop a Deep PDA Matters. In *Automata and Formal Languages, 12th International Conference, AFL 2008, Balatonfüred, Hungary, May 27-30, 2008, Proceedings.*, 2008, s. 281–291.
URL http://www.wortspieler.org/Dateien/Work/Pubs/DeepPDApop.pdf

[4] Limaye, N.; Mahajan, M.: Membership Testing: Removing Extra Stacks from Multi-stack Pushdown Automata. 04 2009, s. 493–504, doi:10.1007/978-3-642-00982-2_42.

[5] Meduna, A.: *Automata and Languages: Theory and Applications [Springer, 2000]*. Springer Verlag, 2005, ISBN 1-85233-074-0, 892 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=6177

[6] Meduna, A.: Deep pushdown automata. *Acta Informatica*, ročník 42, č. 8, 2006: s. 541–552, ISSN 0001-5903.

[7] Meduna, A.; Kolář, D.: One-Turn Regulated Pushdown Automata and Their Reduction. *Fundamenta Informaticae*, ročník 2001, č. 21, 2001: s. 1001–1007, ISSN 0169-2968.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=7035

[8] Meduna, A.; Zemek, P.: *Regulated Grammars and Automata*. Springer-Verlag New York, 2014, ISBN 978-1-4939-0368-9, 694 s.

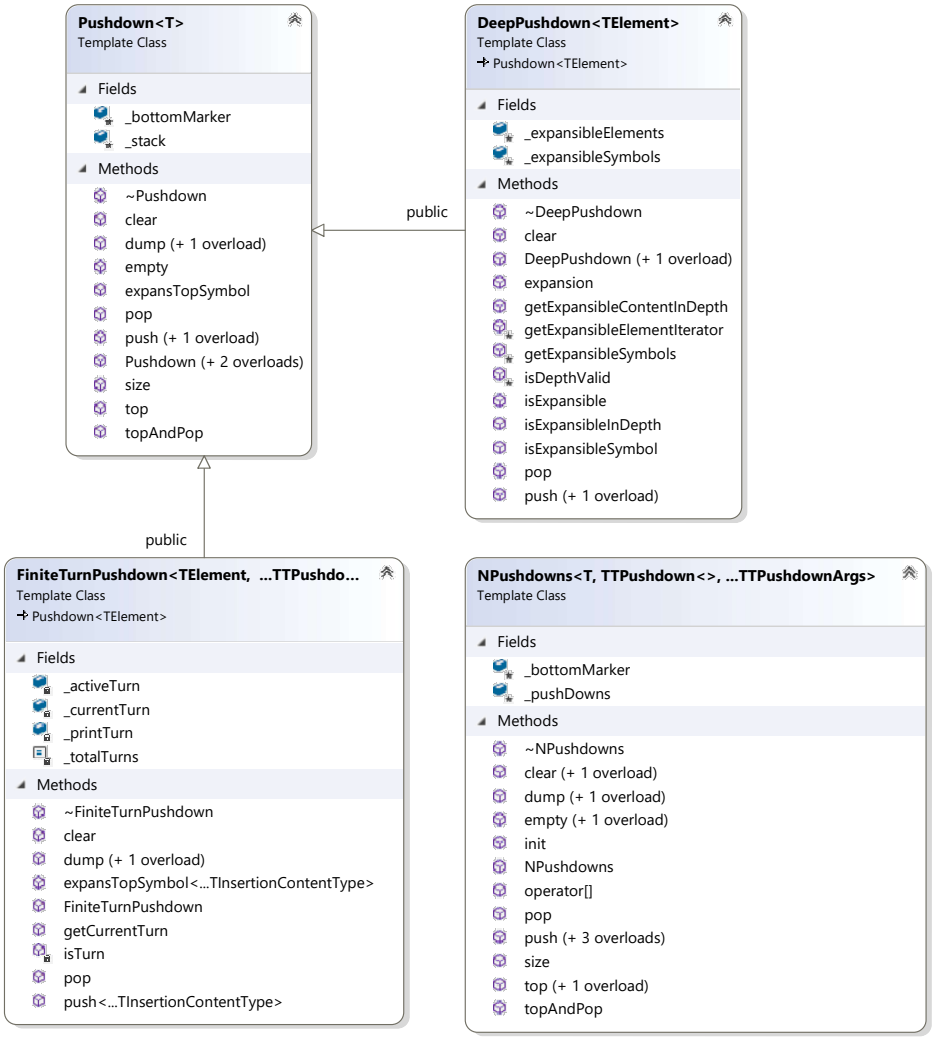
[9] Okhotin, A.; Salomaa, K.: Complexity of Input-driven Pushdown Automata. *SIGACT News*, ročník 45, č. 2, Jún 2014: s. 47–67, ISSN 0163-5700, doi:10.1145/2636805.2636821.
URL http://doi.acm.org/10.1145/2636805.2636821

[10] Rozenberg, G.; Salomaa, A. (editori): *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Berlin, Heidelberg: Springer-Verlag, 1997, ISBN 3-540-60420-0.

[11] Šoustar, J.: *Syntaktická analýza založená na systémech hlubokých zásobníkových automatů*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=19167

Príloha A

Diagramy tried knižnice automata-lib



Obr. A.1: Podrobný diagram tried - zásobníky

Obr. A.2: Podrobný diagram tried - automaty

Príloha B

Manuál

B.1 Knižnica automata-lib

Použitie C++ knižnice

Všetky modely implementujú šablónové triedy, aby bolo možné zvoliť dátový typ `Symbolov` a `Stavov`. Podporovanými dátovými typmi môžu byť `int`, `char`, `string`, `enum`, poprípade zložitejšie dátové typy alebo užívateľom definované dátové štruktúry. V prípade, ak sa trieda inštanciuje zložitejším dátovým typom, je potrebné, aby obsahovala difoltný konštruktor, hašovaciu funkciu a preťažené operátory `<`, `==`, `!=` a `«`. Príklad použitia užívateľom vytvorenej triedy je možné nájsť v zdrojovom kóde, v súbore `tests/complex_types.cpp`.

Príklad vytvorenia *Vstupom riadeného hlbokého viac-zásobníkového automatu.*

```
using NIDDPDA = NIDDPushdownAutomata<char, char>;
NIDDPDA::TConfig d;
NIDDPDA niddpda(d.E, d.Q, d.q, d.R, d.F, d.T, d.S, d.epsilon);
niddpda.printConfig(&streamConfig);
niddpda.makeComputation(in, &stream);
```

Viac informácií o použití knižnice `automata-lib` je možné nájsť v súbore `zdrojovy_kod/automata_lib/ReadMe.txt`.

B.2 Konzolová aplikácia textAutomata

Konzolová aplikácia ponúka možnosť zadať vstupné parametre ako argumenty programu. Definícia automatu musí byť vo formáte `JSON`. Vstup je možné zadať ako reťazec. Aplikácia detekuje typ automatu podľa záznamu `äutomataType"` v definícii.

Argument programu pre vytvorenie automatu a spustenie výpočtu nad reťazcom:

```
textAutomata.exe -d npda_def.json -i abcdefgh
```

Výpis nápovedy pre program:

```
textAutomata.exe -h
```

Spustenie testov knižnice. Ak je zadaný výstupný súbor, logy testov sú ukladané do súboru:

```
textAutomata.exe --test logFile
```

B.3 Grafická aplikácia Automata Simulator

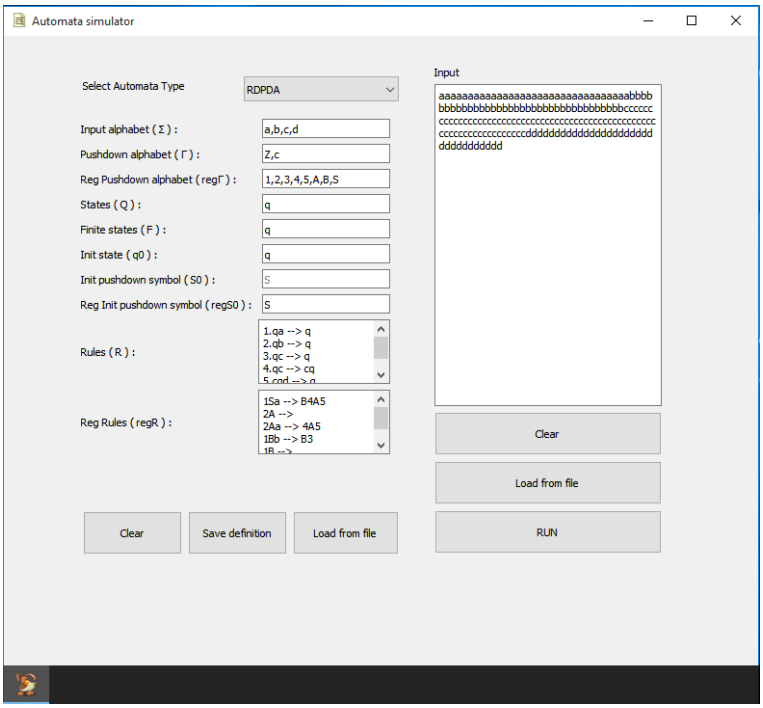
Grafická aplikácia pozostáva z dvoch obrazoviek. Prvá obrazovka obsahuje prvky pre vloženie definície automatu a vstupného reťazca. Druhá obrazovka vizualizuje činnosť automatu.

Definíciu automatu je možné vložiť dvomi spôsobmi. Prvá možnosť ponúka vyplnenie definície priamo v programe do vstupných textových elementov. Pre pohodlnejšiu prácu bola implementovaná možnosť nahrať definíciu zo súboru formátu `JSON`. Pred nahratím súboru nie je potrebné vyberať zo zoznamu aký typ automatu sa v súbore nachádza, program je túto informáciu schopný detekovať.

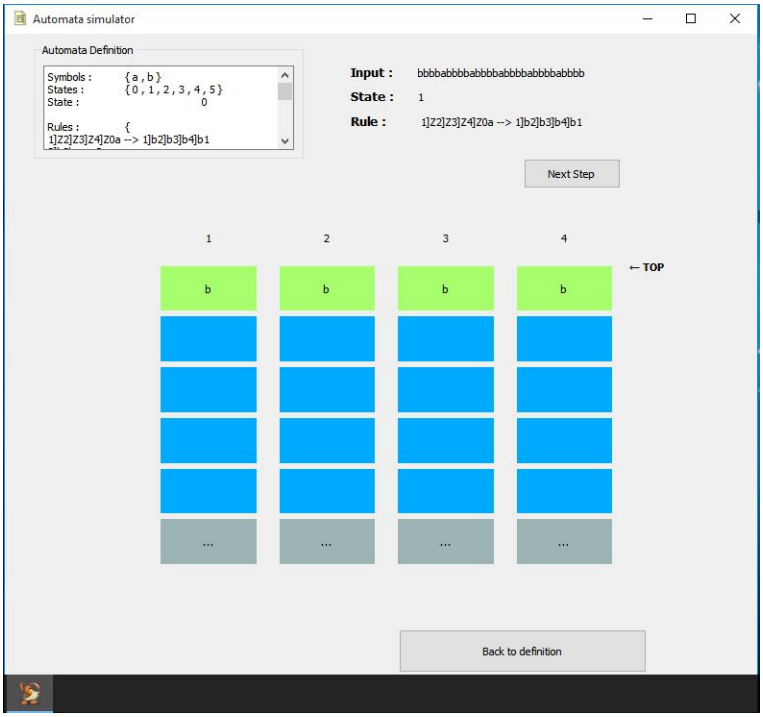
Nahraný súbor je vyplnený do vstupných polí a teda je možné definíciu ľubovoľne upravovať. Presné položky a typy záznamov definícií pre každý automat, sú uvedené v príkladoch priložených ku zdrojovým kódom v priečinku `zdrojovy_kod/automata_definitions`.

Príklad súboru pre *Viac-zásobníkový automat* je uvedený v prílohách v časti C.0.1. Definíciu je tiež možné uložiť do súboru. Tlačítko `Clear` slúži na zmazanie všetkých vyplnených textových polí.

Ukážka grafickej aplikácie



Obr. B.1: Úvodná obrazovka



Obr. B.2: Simulácia

Príloha C

Vstupy a výstupy aplikácie

C.0.1 Definícia automatu

Definícia pre *viac-zásobníkový automat* vo formáte `JSON`.

```
{
    "automataType":   "NPDA",
    "countOfPushdowns":  4,
    "finiteStates":   [ "u" ],
    "initState":    "q",
    "inputSymbols":   [ "a", "b" ],
    "pushdownInitSymbol":  [ "Z", "Z", "Z", "Z" ],
    "pushdownsSymbols": [ "Z", "b" ],
    "rules": [
        "1]Z2]Z3]Z4]Zqa −−> 1]b2]b3]b4]bp",
        "1]bpb −−> r",
        "2]brb −−> s",
        "3]bsb −−> t",
        "4]btb −−> 1]Z2]Z3]Z4]Zq",
        "1]Z2]Z3]Z4]Zq −−> u"
    ],
    "states": [ "q", "p", "r", "s", "t", "u" ]
}
```

C.0.2 Formát zapisovania derivačných krokov počas simulácie automatu

Formát zapisovania derivačných krokov počas simulácie automatu je nasledujúci: `aktuálny vstupný symbol || uplatnené pravidlo || obsah zásobníka č.1 || obsah zásobníka č.2 || ...|| obsah zásobníka č.n`.

Ukážka sekvencie derivačných krokov, ktoré vykonal *Viac-zásobníkový automat*.

```
a || 1]Z2]Z3]Z4]Z0a −−> 1]b2]b3]b4]b1 || b || b || b || b
b || 1]b1b −−> 2 || || b || b || b
b || 2]b2b −−> 3 || || || b || b
b || 3]b3b −−> 4 || || || || b
b || 4]b4b −−> 1]Z2]Z3]Z4]Z0 || Z || Z || Z || Z
a || 1]Z2]Z3]Z4]Z0a −−> 1]b2]b3]b4]b1 || b || b || b || b
b || 1]b1b −−> 2 || || b || b || b
b || 2]b2b −−> 3 || || || b || b
b || 3]b3b −−> 4 || || || || b
b || 4]b4b −−> 1]Z2]Z3]Z4]Z0 || Z || Z || Z || Z
```

Príloha D

Obsah priloženého pamäťového média

K práci je priložené sprievodné CD s nasledujúcou adresárovou štruktúrou:

```
pisomna_praca
zdrojovy_kod
    automata_lib
    automata_simulator
    text_automata
    automata_definitions
    automata_inputs
```

Adresár `zdrojovy_kod` obsahuje zdrojové kódy knižnice `automataLib` v jazyku `C++`, zdrojové kódy grafickej aplikácie `Automata Simulator`, zdrojové kódy konzolovej aplikácie `textAutomata` a testy knižnice.

Adresár `pisomna_praca` obsahuje zdrojové súbory v LaTeXu potrebné pre vygenerovanie písomnej časti diplomovej práce, projekty obrázkov pre `Microsoft Visio` a obrázky. V tomto adresári sa tiež nachádza výsledný `PDF` súbor.