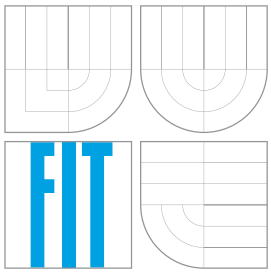
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA REGULOVANÝCH AUTOMATECH

PARSING BASED ON REGULATED AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ĽUBICA GENČÚROVÁ
AUTHOR

VEDOUCÍ PRÁCE — Prof. RNDr. ALEXANDER MEDUNA, CSc.
SUPERVISOR

BRNO 2016

Zadání bakalářské práce/16340/2015/xgencu02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Genčúrová Ľubica**
Obor: Informační technologie
Téma: **Syntaktická analýza založená na regulovaných automatech**
Parsing Based on Regulated Automata
Kategorie: Překladače

Pokyny:

1. Dle pokynů vedoucího se seznamte detailně s regulovanými automaty a jejich vlastnostmi.
2. Dle pokynů vedoucího studujte nové vlastnostmi těchto automatů.
3. Navrhněte jednoduchou metodu syntaktické analýzy, která je založena na automatech s hlubokými zásobníky.
4. Studujte užití metody syntaktické analýzy navržené v předchozích bodech. Zaměřte se na překladače programovacích jazyků. Navrhněte vhodný programovací jazyk a sestrojte jeho syntaktický analyzátor, který provádí syntaktickou analýzu na základě této metody. Testujte výsledný syntaktický analyzátor.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna, A.: Automata and Languages, Springer, London, 2000

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních 3 bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Meduna Alexander, prof. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá návrhem, aplikovatelností v praxi a implementací metody syntaktické analýzy založené na zásobníkových automatech, které regulují použití jejich pravidel řídícím lineárním jazykem. Tyto automaty mají větší sílu než běžný zásobníkový automat a jejich síla je rovna Turingovému stroji. Jsou schopné akceptovat rodinu rekurzivně vyčíslitelných jazyků.

Abstract

This work deals with the design, applicability and the implementation of parsing methods based on pushdown automata that regulate the use of their rules by linear control languages. These automata are more powerful than ordinary automata and they are as powerful as Turing machine. The pushdown automata regulated by linear control languages characterize the family of recursively enumerable languages.

Klíčová slova

Regulované automaty, zásobníkové automaty, řídíci jazyk, lineární jazyk, rekurzívně vyčíslitelný jazyk, syntaktická analýza

Keywords

Regulated automata, pushdown automata, control language, linear language, recursively enumerable languages, parsing

Citace

GENČÚROVÁ, Ľubica. *Syntaktická analýza založená na regulovaných automatech.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Meduna Alexander.

Syntaktická analýza založená na regulovaných automatech

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Prof. RNDr. Alexandera Meduny, CSc. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Ľubica Genčúrová
17. května 2016

Poděkování

Ráda bych poděkovala panu Prof. RNDr. Alexanderovi Medunovy, CSc. za kontrolu mých textů, jeho ochotu a odborné rady na konzultacích a v neposlední řade také za pomoc při výběru tématu mé práce.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Táto práca sa zaoberá návrhom, aplikovateľnosťou a implementáciou jednoduchej metódy syntaktickej analýzy, ktorá je založená na zásobníkových automatoch regulovaných riadiacim lineárnym jazykom. Poukazuje na významný efekt a zvýšenie výpočetnej sily zásobníkového automatu vďaka použitiu kontrolného jazyka pre regulovanie automatu.

Pre pochopenie jadra tejto bakalárskej práce, teda popis, zostrojenie tohto typu automatu a následne zostrojenie syntaktickej analýzy 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 nie je možná pre väčšinu programovacích jazykov, 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.

V posledných desaťročiach hrali dôležitú rolu v teórii formálnych jazykov gramatiky, ktoré regulujú používanie svojich pravidiel rôznymi kontrolnými mechanizmami. Okrem gramatík, ale teória formálnych jazykov využíva ako základné jazykové modely automaty. Práve vďaka týmto skutočnostiam vznikla myšlienka zaviesť zásobníkové automaty regulujúce použitie pravidla riadiacim jazykom. Tieto automaty boli prvý krát predstavené v roku 2000, v článku *Regulated Pushdown Automata* [2].

Ako riadiaci jazyk bol spočiatku skúmaný regulárny, avšak táto regulácia nemá žiadny efekt na výpočetnú silu zásobníkového automatu. Ak je zvolený ako riadiaci jazyk lineárny, automaty sú schopné prijímať rodinu rekurzívne vyčísliteľných jazykov. Ich výpočetná sila je väčšia ako sila bežných zásobníkových automatov a zároveň zrovnateľná so silou Turingovho stroja. Keďže automatom tohto typu nebola od vydania článku venovaná pozornosť, bolo mojou motiváciou zamerať sa práve na ne a pokúsiť sa o aplikáciu v praxi.

V nasledujúcej kapitole 2 nájdeme stručný úvod do problematiky formálnych jazykov a základné definície potrebné pre ďalšie kapitoly. V kapitole 3 a 4 si už následne definujeme konkrétne prostriedky pre reprezentáciu a rozpoznávanie jazyka, aby sme sa priblížili k cieľu

tejto práce, a mohli sme predstaviť regulované automaty v kapitole 5. Popis syntaktickej analýzy a jej využitia nájdeme v kapitole 6.

Zvyšná časť bakalárskej práce sa venuje praktickej časti, kde sa nachádza formálne zostrojenie syntaktického analyzátora a konkrétnych entít potrebných pre činnosť syntaktickej analýzy. Tento celok je členený na dve časti.

Prvá časť slúži pre generovanie reťazca rekurzívne vyčísliteľného jazyka. Ako bude z nasledujúcich kapitol zrejmé, tento jazyk je generovaný frázovou gramatikou, taktiež nazývanou ako neobmedzená gramatika. Práve preto je v praxi veľmi ťažko implementovateľná. Jej činnosť je v tejto práci simulovaná dvomi regulovanými lineárnymi gramatikami.

Druhá časť je zameraná na akceptovanie toho reťazca syntaktickou analýzou založenou na zásobníkovom automate, ktorý reguluje použitie pravidiel lineárnym riadiacim jazykom. Riadiaci jazyk je generovaný lineárnou gramatikou.

Na záver sú uvedené podrobnosti implementácie a v prílohách sa nachádza demonštrácia implementovanej metódy.

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 [3].

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 $i = 1, ..., n$, 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 $\text{card}(L) = n$ 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: $L_1 \cup L_2 = \{x : x \in L_1 \text{ alebo } x \in L_2\}$.
- *Prienik jazykov* L_1 a L_2, $L_1 \cap L_2$ je definovaný ako: $L_1 \cap L_2 = \{x : x \in L_1 \text{ a } x \in L_2\}$.
- *Rozdiel jazykov* L_1 a L_2, $L_1 - L_2$ je definovaný ako: $L_1 - L_2 = \{x : x \in L_1 \text{ a } x \notin L_2\}$.
- *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: $L_1L_2 = \{xy : x \in L_1 \text{ a } y \in L_2\}$. 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. $L^* = L^* \cup \{\varepsilon\}$

Kapitola 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 rolu v teórii formálnych jazykov a spĺňajú základnú požiadavku kladenú na reprezentáciu konečných aj nekonečných jazykov - požiadavka konečnosti reprezentácie.

Gramatika využíva dve konečné abecedy(2.1.1), ktoré sú navzájom disjunktné:

1. množina N - *nonterminálnych symbolov*, skrátene *nonterminály*, 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 *nonterminálu* aplikáciou prepisovacieho pravidla generuje reťazce - vetné formy, ktoré sú tvorené *terminálmi* a *neterminálmi*. 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.

V roku 1956 americký matematik Noam Chomský zaviedol *Chomského klasifikáciu gramatík a jazykov*. Rozdelil 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 3.0.1. Frázová gramatika je štvorica $G = (N, T, P, S)$, kde

- N je abeceda *neterminálov*;
- T je abeceda *terminálov*, kde $N \cap T = \emptyset$;
- 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 *nonterminál*. Reťazec v je prvkom množiny $(N \cup T)^*$. Formálne:

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

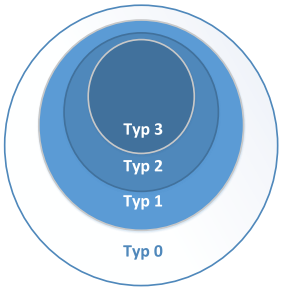
Ak v je prázdny reťazec, teda $v = \varepsilon$, toto pravidlo nazývame *ε-pravidlo*.
Množina $V = N \cup T$ je *úplna abeceda* gramatiky G.

3.1 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 (viď. obrázok 3.1). Napríklad gramatika typu 2 je zároveň gramatikou typu 1 a 0.

Ako už bolo spomenuté všetky triedy gramatík majú rovnaký základ. Uvažujme teda gramatiku G z definície 3.0.1. Potom prepisovacie pravidlá z množiny prepisovacích pravidiel P budú vyzerať nasledovne:



Obr. 3.1: Chomského hierarchia

3.1.1 Typ 0 - Frázová gramatika

Gramatika typu 0 obsahuje pravidlá v najvšeobecnejšom tvare, zhodným s definíciou 3.0.1 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).

3.1.2 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.

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

Výnimku tvorí pravidlo:

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

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

3.1.3 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:

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

Na ľavej strane pravidla je povolený iba jeden neterminál. Touto gramatikou sú generované *bezkontextové jazyky*, ktoré sú akceptovateľné zásobníkovým automatom(4.2.1). Rodina *bezkontextových jazykov* je označovaná ako **CF** (context-free).

3.1.4 Typ 3 - Regulárna gramatika

Definícia 3.1.1. 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:

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

Regulárna gramatika generuje *regulárne jazyky*. Tieto jazyky akceptuje konečný automat(4.1.1). 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 i 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.

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

3.1.5 Lineárna gramatika

Definícia 3.1.2. Uvažujme gramatiku $G = (N, T, P, S)$ z definicie 3.0.1. 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(4.2.1). Rodina *lineárnych jazykov* je označovaná ako **LIN** (linear).

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

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

3.2 Frontové gramatiky

Frontové gramatiky prepíšu reťazec spôsobom, ktorý sa podobá na štandardnú abstraktnú dátovú štruktúru - frontu.

Tieto gramatiky pracujú na princípe **first-in-first-out**, teda prvý dnu, prvý von. Prvý symbol, ktorý bol pridaný do reťazca bude prvý odstránený.

Konkrétne počas každého derivačného kroku gramatika pripojí reťazec ako sufix - príponu, k aktuálnej vetnej forme a zároveň odstráni najľavejší symbol tejto formy. To znamená, že všetky symboly, ktoré boli pripojené k vetnej forme pred týmto krokom, musia byť odstránené skorej ako novopripojený sufix.

Definícia 3.2.1. *Frontová gramatika* je šestica: $Q = (V, T, W, F, R, g)$, kde V a W sú dve abecedy, pre ktoré platí $V \cap W = \emptyset, T \subset V, F \subset W, g \in (V - T)(W - F)$, a

$$R \subseteq V \times (W - F) \times V^* \times W$$

je konečná relácia. Pre každé $a \in V$ existuje element $(a, b, x, c) \in R$. Ak $u = arb$, $v = rxc$, a $(a, b, x, c) \in R, r, x \in V^*$, kde $a \in V$ a $b, c \in W$, potom Q urobí *derivačný krok* z u do v podľa (a, b, x, c). Symbolický zápis:

$$u \Rightarrow_Q v[(a, b, x, c)]$$

Zjednodušene:

$$u \Rightarrow_Q v$$

Jazyk gramatiky Q sa označuje ako $L(Q)$ a je definovaný ako:

$$L(Q) = \{x \in T^* \quad | \quad g \Rightarrow_Q^* xf, f \in F\}$$

3.2.1 Ľavo-rozšírená frontová gramatika

Definícia 3.2.2. *Ľavo-rozšírená frontová gramatika*[3] je šestica: $Q = (V, T, W, F, R, g)$, kde V, T, W, F, R, g majú rovnaký význam, ako vo frontovej gramatike. S výnimkou, že platí $\# \notin V \cup W$. Ak $u, v \in V^*\{\#\}V^*W$ tak $u = w\#arb$, $v = wa\#rzc$, $a \in V$, $r, z, w \in V^*, b, c \in W$, a $(a, b, z, c) \in R$, potom:

$$u \Rightarrow_Q v[(a, b, z, c)]$$

Jazyk gramatiky Q sa označuje ako $L(Q)$ a je definovaný ako:

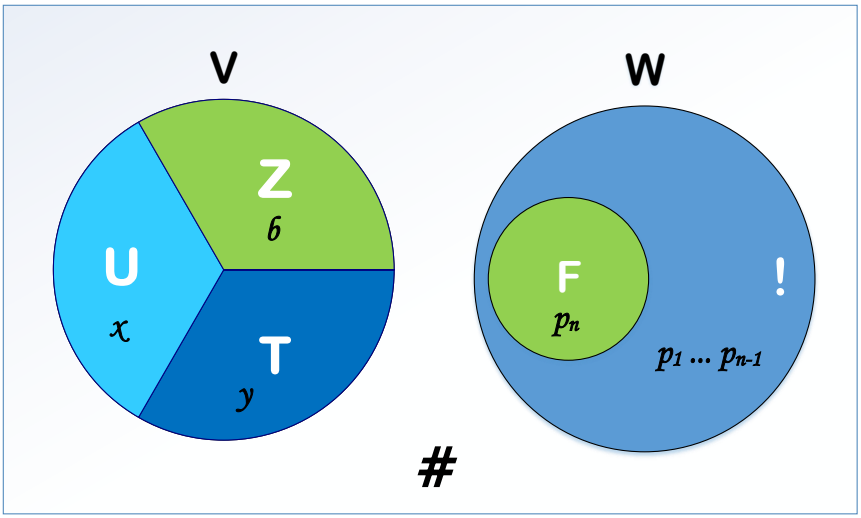
$$L(Q) = \{v \in T^* \quad | \quad \#g \Rightarrow_Q^* w\#fv \quad \text{pre } w \in V^* \text{ a } f \in F\}$$

Neformálne. Počas každého derivačného kroku, ľavo rozšírená frontová gramatika posunie prepisovaný symbol nad #. Toto zaznamenáva históriu derivácií.

Pre každú ľavo-rozšírenú frontovú gramatiku K existuje ľavo-rozšírená frontová gramatika $Q = (V, T, W, F, s, P)$, spĺňajúca $L(K) = L(Q)$, **!** je *rozlišujúci člen* rozdielu množín $(W - F)$, $V = U \cup Z \cup T$. U, Z, T sú navzájom disjunktné (pre lepšiu predstavu vzťahov množín pozri obrázok 3.2) a Q derivuje každé $z \in L(Q)$ nasledovne:

$$\begin{aligned}
S\# &\Rightarrow^+ x\#b_1b_2 \dots b_n! \\
&\Rightarrow xb_1\#b_2 \dots b_n y_1 p_2 \\
&\Rightarrow xb_1b_2\#b_3 \dots b_n y_1 y_2 p_3 \\
&\vdots \\
&\Rightarrow xb_1b_2 \dots b_{n-1}\# \dots b_n y_1 y_2 \dots y_{n-1} p_n \\
&\Rightarrow xb_1b_2 \dots b_{n-1} b_n \# y_1 y_2 \dots y_n p_{n+1}
\end{aligned}$$

kde $n \in \mathcal{N}, x \in U^*, b_i \in Z$ pre $i = 1, \dots, n, y_i \in T^*$ pre $i = 1, \dots, n, z = y_1 y_2 \dots y_n, p_i \in W - \{!\}$ pre $i = 1, \dots, n-1, p_n \in F$, a iba derivácia $x\#b_1b_2 \dots b_n!$ obsahuje reťazec !. Dôkaz tohto tvrdenia sa nachádza v článku [2] v sekcii 4, Lemma 3.



Obr. 3.2: Vzťahy množín ľavo-rozšírenej frontovej gramatiky

[3] Anglicky Left-extended queue grammar

Kapitola 4

Automaty

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 3.1, strana 9, sme spomenuli, že tieto jazyky prijíma *konečný automat*(REG) a *zásobníkový automat* (CF).

4.1 Konečný automat

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 4.1.1. *Konečný automat*[1] 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 4.1.2.

Konfigurácia konečného automatu M(4.1.1) 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:

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

[1] Anglicky Finite Automata

Definícia 4.1.3. 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:

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

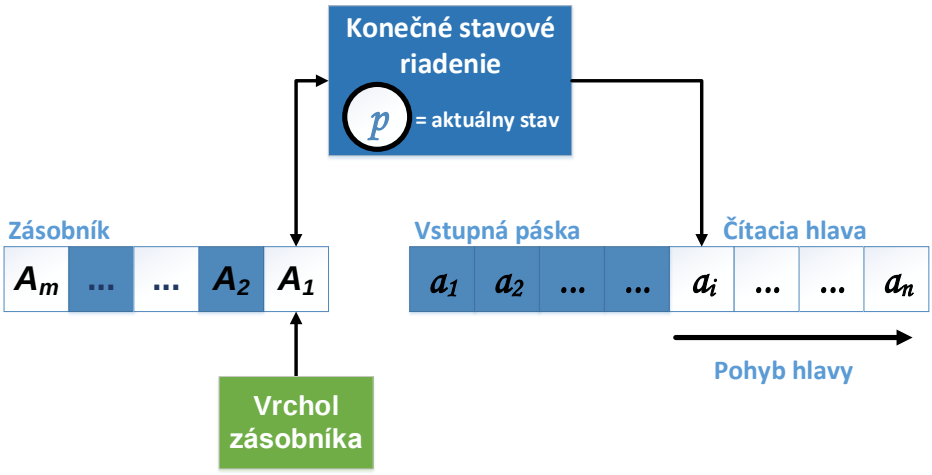
Definícia 4.1.4. Majme konečný automat M(4.1.1), potom *jazyk prijímaný* konečným automatom $M, L(M)$, je definovaný ako:

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

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

4.2 Zásobníkové automaty

Zásobníkový automat je konečný automat, ktorý je rozšírený o zásobník (viď obrázok 4.1.1). Tento automat prijíma rodinu jazykov **CF**, teda bezkontextové jazyky.



Obr. 4.1: Zásobníkový automat

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 neterminál, 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. Požiadavok skončiť vo finálnom stave môže byť vynechaný a je postačujúce, že po prečítaní reťazca bude zásobník prázdny.

Definícia 4.2.1. *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 je *počiatočný stav*,
- $S \in \Gamma$ je *počiatočný zásobníkový symbol*,
- $F \subseteq Q$ je množina *koncových stavov*,

Kapitola 5

Regulované automaty

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. Zásobníkové automaty regulované lineárnym jazykom charakterizujú rodinu rekurzívne vyčísliteľných jazykov.

Neskoršie v roku 2001 boli predstavené *jedno-otáčkové regulované zásobníkové automaty* v článku *One-Turn Regulated Pushdown Automata and Their Reduction* [4]. Ak automat počas prvého kroku neskráti svoj zásobník a počas druhého kroku sa tak stane, potom urobí „otáčku“ počas druhého kroku. Zásobníkový automat je „jedno-otáčkový“ v prípade ak neurobí viac „otáčkových“ pohybov v priebehu ktoréhokoľvek výpočtu od počiatočnej konfigurácie. V článku je dokázané, že jedno-otáčkové regulované zásobníkové automaty charakterizujú rodinu rekurzívne vyčísliteľných jazykov **RE** (3.1.1). Jedno-otáčkové regulované zásobníkové automaty sú rovnako mocné ako regulované zásobníkové automaty, ktoré môžu urobiť ľubovoľný počet „otáčok“, avšak ich mechanizmus je jednoduchší.

Následne boli v článku *Self-Regulating Finite Automata*[5] predstavené *sebariadiace automaty*, ktoré regulujú výber pravidla podľa toho, aké pravidlo bolo aplikované v predchádzajúcom kroku.

Spomínané články boli uverejnené vo vedeckom časopise Acta Cybernetica.

5.1 Zásobníkové automaty regulované riadiacim jazykom

Na úvod si definujeme zásobníkové automaty, ktoré regulujú aplikáciu pravidla riadiacim jazykom. Ako už bolo spomenuté, regulovanie automatu pomocou regulárneho jazyku neprináša žiadne zvýšenie sily zásobníkového automatu, preto sa zameriame konkrétne na automaty regulované lineárnym riadiacim jazykom. Zásobníkový automat regulovaný riadiacim lineárnym jazykom má väčšiu výpočetnú silu ako obyčajný zásobníkový automat, čo je dokázané v publikácii [6] v sekcii **16.2.3**. Predstavíme si základnú verziu tohto typu automatu a špeciálny automat - *Jedno - otáčkový automat regulovaný riadiacim jazykom*, ktorý je lineárny.

Definícia 5.1.1. Nech $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat (4.2.1) 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 5.1.2. 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 Ξ obsahuje riadiaci reťazec, podľa ktorého automat M vykonal sekvenciu krokov, teda po prečítaní reťazca x prešiel do konečnej konfigurácie.

Definícia 5.1.3. 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 [6] (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:

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

[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.

Príklad 5.1.1. 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 4.2.1 s nasledujúcimi pravidlami:

$$\begin{array}{lrl} 1. & Ssa & \rightarrow & Sas \\ 2. & asa & \rightarrow & aas \\ 3. & asb & \rightarrow & q \\ 4. & aqb & \rightarrow & q \\ 5. & Sqc & \rightarrow & Sq \\ 6. & Sqc & \rightarrow & f \end{array}$$

Tento zásobníkový automat prijíma reťazec $aabbccc$:

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

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 $\Xi = \{12^m 34^n 5^n 6 : m, n \geq 0\}$:

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

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 $1234556 \notin \Xi = \{1\}\{24\}^*\{35\}$.

5.2 Zásobníkové automaty regulované lineárnym riadiacim jazykom

V nasledujúcich odstavcoch si ukážeme úryvok dôkazu formálneho zostrojenia zásobníkového automatu, ktorý je regulovaný lineárnym jazykom. Ako prvú zostrojíme lineárnu gramatiku, pre generovanie riadiaceho jazyka, ktorú odvodíme z ľavo-rozšírenej frontovej gramatiky. Celý dôkaz je k dispozícii v texte [2].

Majme nasledujúce entity:

- ľavo-rozšírenú frontovú gramatiku $\mathbf{Q} = (V[Q], T[Q], W[Q], F[Q], s[Q], P[Q])$ [3] z definície 3.2.1. Bez straty všeobecnosti predpokladajme $\{@, £, ¶\} \cap (V[Q] \cup W[Q]) = \emptyset$. Definujme si značenie ζ z $(V[Q])^*$ do $\{\langle £as\rangle | a \in V[Q]\}^*$ ako $\zeta(a) = \{\langle £as\rangle\}$, kde s je použité ako štartovací stav zásobníkového automatu M.

- lineárnu gramatiku $\mathbf{G} = (N[G], T[G], P[G], S[G])$ (3.1.5), ktorú zostrojíme nasledovne:

  - $N[G] = \{S[G], \langle!\rangle, \langle!, 1\rangle\} \cup \{\langle f\rangle | f \in F[Q]\}$,
  - $T[G] = \zeta(V[Q]) \cup \{\langle £§s\rangle, \langle £@\rangle\} \cup \{\langle £§f\rangle | f \in F[Q]\}$,
  - $P[G] = \{S[G] \to \langle £§s\rangle\langle f\rangle | f \in F[Q]\} \cup \{\langle!\rangle \to \langle!, 1\rangle\langle £@\rangle\}$

- zásobníkový automat $\mathbf{M} = (Q[M], \Sigma[M], \Gamma[M], R[M], s, S[M], \{\rfloor\})$(4.2.1), kde

  - $Q[M] = \{s, \langle ¶!\rangle, \lfloor, \rceil\}$,
  - $\Sigma[M] = \{S[M], §\} \cup V[Q]$,
  - $R[M] = \{\langle £§s\rangle.S[M] \to §s\} \cup \{\langle £§f\rangle.§\langle ¶f\rangle \to \rceil | f \in F[M]\}$

Bude platiť, že $L(Q) = L(M, L(G), 3)$.

Gramatiku G rozšírime aplikovaním nasledujúcich krokov pre každé pravidlo $(a, p, x, q) \in P[Q]$ pomocou gramatiky Q[4], kde

1. $p, q \in W[Q], a \in Z, x \in T^*$,

$$\begin{array}{lcl} N[G] = N[G] & \cup & \{\langle apxqk\rangle | k = 0, \ldots, |x|\} \cup \{\langle p\rangle, \langle q\rangle\} \\ T[G] = T[G] & \cup & \{\langle £\texttt{sym}(y, k)\rangle | k = 1, \ldots, |y|\} \cup \{\langle apxq\rangle\} \\ P[G] = P[G] & \cup & \{\langle q\rangle \to \langle apxq|x|\rangle\langle £apxq\rangle, \langle apxq0\rangle \to \langle p\rangle\} \\ & \cup & \{\langle apxqk\rangle \to \langle apxq(k-1)\rangle\langle £\texttt{sym}(x, k)\rangle | k = 1, \ldots, |x|\} \end{array}$$

2. $p, q \in W[Q], a \in U, x \in (W[Q])^*$,

$$\begin{array}{lcl} N[G] = N[G] & \cup & \{\langle p, 1\rangle, \langle q, 1\rangle\} \\ P[G] = P[G] & \cup & \{\langle q, 1\rangle \to \texttt{reversal}(\zeta(x))\langle p, 1\rangle\zeta(a)\} \end{array}$$

3. $ap = S[Q], p, q \in W[Q], x \in (W[Q])^*$,

$$\begin{array}{lcl} N[G] = N[G] & \cup & \{\langle q, 1\rangle\} \\ P[G] = P[G] & \cup & \{\langle q, 1\rangle \to \texttt{reversal}(x)\langle £\$s\rangle\} \end{array}$$

Konštrukcia gramatiky G je kompletná. Množina terminálov gramatiky $T[G] \in G$ bude abeceda identifikátorov pravidiel zásobníkového automatu M, $\Psi = T[G]$.

Množinu stavov automatu $Q[M]$ a množinu jeho pravidiel $R[M]$ rozšírime aplikovaním nasledujúcich krokov.

1. $R[M] = R[M] \cup \{\langle £bs\rangle.as \to abs | a \in \Gamma[M] - \{S[M]\}, b \in \Gamma[M] - \{\$\}\}$;

2. $R[M] = R[M] \cup \{\langle £\$s\rangle.as \to a\lfloor | a \in V[Q]\} \cup \{\langle £a\rangle.a\lfloor | a \in V[Q]\}$;

[3] Gramatika Q využitá pre ukážku formálneho zostrojenia lineárnej gramatiky a automatu musí splňovať vlastnosti dôkazu "Lemma 3"z článku [2].

[4] Pre lepšie pochopenie vzťahov množín ľavo-rozšírenej frontovej gramatiky pozri obrázok 3.2

3. $R[M] = R[M] \cup \{\langle £@\rangle.a\lfloor \rightarrow a\langle ¶!\rangle | a \in Z\}$;

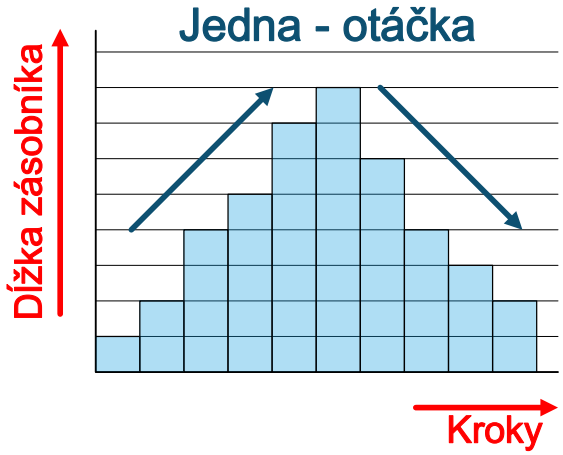
4. pre každé pravidlo $(a, p, x, q) \in P[Q]$, kde $p, q \in W[Q], a \in Z, x \in (T[Q])^*$,

$$\begin{array}{lll} Q[M] = Q[M] & \cup & \{\langle ¶p\rangle\} \cup \{\langle ¶qu\rangle \mid u \in \texttt{prefix}(x)\} \\ R[M] = R[M] & \cup & \{\langle £b\rangle.a\langle ¶qy\rangle b \rightarrow a\langle ¶qyb\rangle | b \in T[Q], y \in (T[Q])^*, \\ & yb & \in \texttt{prefix}(x)\} \cup \{\langle £apxq\rangle.a\langle ¶qx\rangle \rightarrow \langle ¶p\rangle q\}. \end{array}$$

Konštrukcia automatu M je kompletná. Ak x začína s $£$, potom $\langle x\rangle \in T[G]$. Ak x začína s ¶, potom $\langle x\rangle \in Q[M]$. Ak x začína iným symbolom, potom $\langle x\rangle \in N[G]$.

5.2.1 Jedno-otáčkový atomický zásobníkový automat regulovaný lineárnym jazykom

Uvažujme dva po sebe nasledujúce kroky, ktoré vykoná zásobníková automat M (4.2.1). Ak počas prvého kroku M neskráti dĺžku svojho zásobníka a počas druhého kroku skráti, tak potom M vykoná „otáčku“ počas druhého kroku.



Obr. 5.1: Jedno-otáčkový zásobníkový automat

Zásobníkový automat je *jedno - otáčkový* ak neurobí viac „otáčok“ v priebehu ktoréhokoľvek výpočtu od počiatočnej konfigurácie.

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

1. vloží symbol na zásobník
2. vyjme symbol zo zásobníka
3. číta symbol z vstupnej pásky

Definícia 5.2.1. *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 značiaci 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 *otočka*. Ak M neurobí viac ako jednu otočku počas ktorejkoľvek sekvencie pohybov štartujúcej z počiatočnej konfigurácie, potom automat M nazývame **jedno-otáčkový**.

Kapitola 6

Syntaktická analýza v prekladačoch

V nasledujúcej kapitole 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, aby na výstupe generoval 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

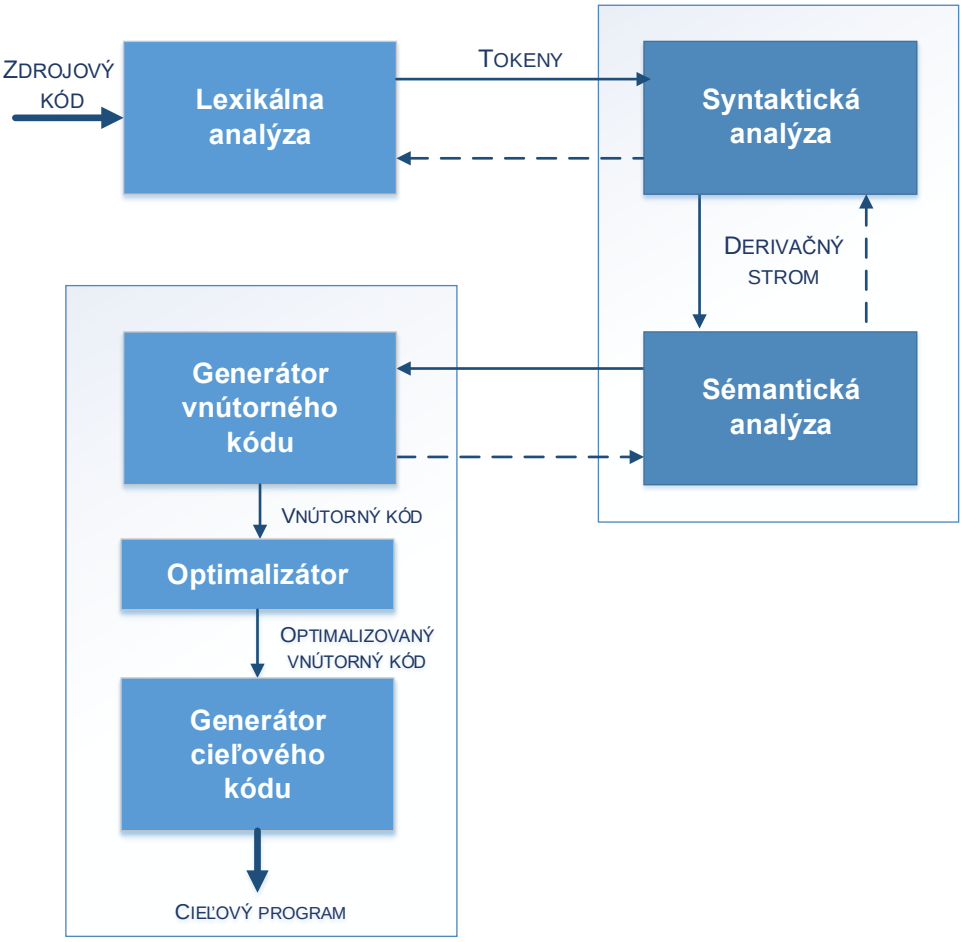
Jednotlivé fázy spolu aj so vstupmi a výstupmi prekladača sú zobrazené na obrázku 6.1.

Lexikálna analýza rozloží zdrojoví kód na lexémy - lexikálne jednotky - tokeny, čo sú jednotlivé rozoznané elementy zdrojového jazyka. Napríklad identifikátory, zápisy čísel, kľúčové slová, a pod. Lexikálna analýza produkuje *tokeny*, ktoré sú vstupom pre syntaktickú analýzu. Činnosť lexikálnej analýzy je zobrazená na obrázku 6.2.

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* [1].

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

[1] Anglicky sa syntaktický analyzátor nazýva **parser**



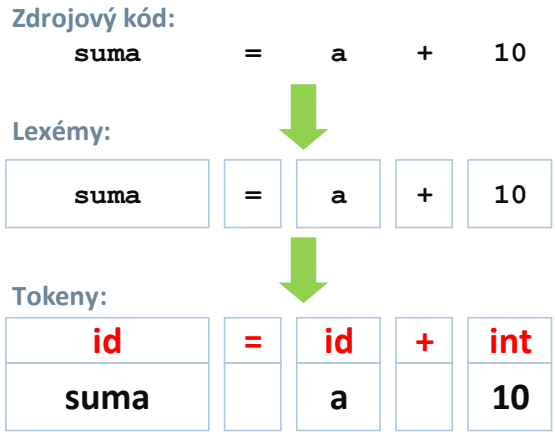
Obr. 6.1: Prekladač

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,

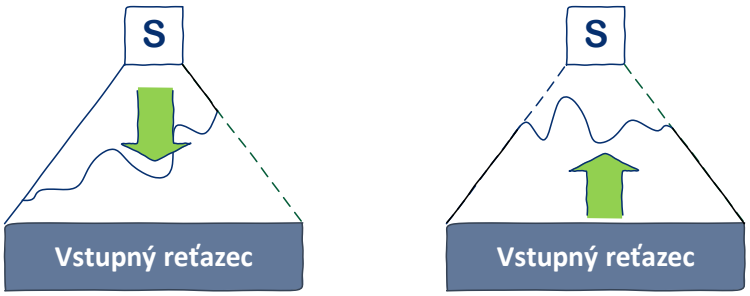


Obr. 6.2: Lexikálna analýza - rozoznanie lexikálnych jednotiek v zdrojovm kóde

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*, program je správny.

Vytváranie derivačného stromu je založené na gramatických pravidlách. Existujú dva prístupy syntaktickej analýzy *zhora dole* a *zdola na hor* (viď obrázok 6.3). 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. 6.3: Vytváranie derivačného stromu

Kapitola 7

Zostrojenie syntaktického analyzátora

V nasledujúcej kapitole si zostrojíme syntaktický analyzátor a potrebné entity, pre jeho činnosť.

Náplňou tejto práce je zostrojiť syntaktickú analýzu, ktorá je založená na regulovaných zásobníkových automatoch. Ako už bolo spomenuté v kapitole 5.1 regulácia zásobníkových automatov riadiacim jazykom, ktorý je regulárny nemá žiadny efekt na silu zásobníkového automatu, preto zvolíme ako riadiaci jazyk, jazyk lineárny.

Zásobníkové automaty regulované riadiacim lineárnym jazykom sú schopné prijať rodinu rekurzívne vyčísliteľných jazykov **RE** (3.1.1). Túto rodinu jazykov generuje frázová gramatika.

Činnosť frázovej gramatiky budeme v tomto prípad simulovať použitím dvoch lineárnych gramatík. Pomocou prvej gramatiky vygenerujeme časť reťazca a postupnosť pravidiel, ktoré boli aplikované pri generovaní reťazca, použijeme ako riadiaci jazyk pre druhú z týchto lineárnych gramatík.

Aplikovateľnosť a implementovateľnosť tejto metódy syntaktickej analýzy prevedieme pomocou nasledujúceho jednoduchého rekurzívne vyčísliteľného jazyka, ktorý bude akceptovateľný našim zostrojeným syntaktickým analyzátorom, teda taktiež našim automatom regulovaným riadiacim jazykom, ktorý je lineárny.

7.1 Generovanie reťazca rekurzívne vyčísliteľného jazyka

Majme jazyk z rodiny **RE**:

$$L(M) = \{a^n b^n c^n : n \geq 1\}$$

Tento jazyk bude generovaný dvomi lineárnymi gramatikami.

Majme lineárnu gramatiku $G = (N[G], T[G], P[G], S)$ z definície 3.1.5, kde

- $N[G] = S$
- $T[G] = \{a, c\}$
- $P[G] = \{1\text{: } S \to aSc\}$
- $S \in N[G]$, je počiatočný neterminál.

Generovanie reťazca gramatikou G:

$$\begin{array}{rcll} S & \Rightarrow & aSc & [1] \\ & \Rightarrow & aaScc & [1] \\ & \Rightarrow & aaaSccc & [1] \\ & \Rightarrow & aaaaScccc & [1] \end{array}$$

Postupnosť pravidiel: **1111** využijeme ako riadiaci jazyk pre generovanie zvyšnej časti reťazca $\Xi = \{1111\}$. Ako vidíme pravidlo **1** bolo aplikované štyri krát. Môžeme využiť skrátený zápis $\Xi = \{1\}^4$.

Majme gramatiku $Q = (N[Q], T[Q], P[Q], S)$ z definície 3.1.5, kde

- $N[Q] = S$
- $T[Q] = \{b\}$
- $P[Q] = \{1\text{: } S \to Sb, \quad 2\text{: } S \to \varepsilon\}$
- $S \in N[Q]$, je počiatočný neterminál.

Gramatika reguluje aplikovanie pravidiel pomocou riadiaceho lineárneho jazyka $\Xi = \{1111\}$. Po ukončení generovania reťazca je potrebné odstrániť neterminál S, čo vykonáme aplikovaním pravidla **2**. Toto pravidlo pridáme na koniec riadiaceho jazyka, teda

$$\Xi = \Xi \cup \{2\} = \{11112\} = \{1\}^4\{2\}$$

Pomocou tejto gramatiky a riadiaceho jazyka $\Xi = \{1111112\}$ vygenerujeme strednú časť reťazca nasledovne:

$$\begin{array}{rcll} aaaaaaScccccc & \Rightarrow & aaaaSbcccc & [1] \quad \Xi = \{\mathbf{1}1112\} \\ & \Rightarrow & aaaaSbbcccc & [1] \quad \Xi = \{\mathbf{1}112\} \\ & \Rightarrow & aaaaSbbbcccc & [1] \quad \Xi = \{\mathbf{1}12\} \\ & \Rightarrow & aaaaSbbbbcccc & [1] \quad \Xi = \{\mathbf{1}2\} \\ & \Rightarrow & aaaabbbbcccc & [1] \quad \Xi = \{\mathbf{2}\} \end{array}$$

Reťazec *aaaabbbbcccc* vygenerovaný gramatikami G a Q patrí jazyku $L(M) = \{a^n b^n c^n : n \geq 1\}$.

7.2 Zostrojenie automatu regulovaného riadiacim lineárnym jazykom

Majme *Zásobníkový automat* $M = (Q, \Sigma, \Gamma, R, s, S, F)$ z definície 4.2.1 , kde

- $Q = \{s, q, f\}$,
- $\Sigma = \{a, b, c\}$,
- $\Gamma = \{a, S\}$,
- $R = \{1\text{:}Ss \to q,\ 2\text{: } qa \to aq,\ 3\text{: } aqb \to q,\ 4\text{: } qc \to q,\ 5\text{: } Sq \to f\}$
- $s \in Q$ je je *počiatočný stav*,

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

Tento automat bude prijímať jazyk $L(M, \Xi, 3) = \{a^n b^n c^n : n \geq 1\}$ a aplikáciu pravidiel bude regulovať lineárnym jazykom, ktorý generuje lineárna gramatika G, kde

$$\Psi[M] = T[G]$$

.

Majme lineárnu gramatiku $G = (N[G], T[G], P[G], S)$ z definície 3.1.5, kde

- $N[G] = \{S, X\}$
- $T[G] = \{1, 2, 3, 4, 5\}$
- $P[G] = \{1\text{: } S \to 12X5,\ 2\text{: } X \to 3X4,\ 3\text{: } X \to \varepsilon\}$
- $S \in N[G]$, je počiatočný neterminál.

Reťazec musí obsahovať rovnaký počet terminálov a, b a c. Túto kontrolu prevedieme nasledujúcim spôsobom.

Na úvod vložíme na zásobník znak značiaci dno zásobníka \$, počiatočný symbol S a automat sa bude nachádzať v počiatočnom s. Gramatika G aplikuje pravidlo 1: $S \to 12X5$, aby vygenerovala počiatočný riadiaci jazyk pre automat, teda $\Xi = \{12X5\}$.

Automat podľa Ξ aplikuje formálne pravidlo **1:**$Ss \to q$, prejde do stavu q a vyjme použité pravidlo $\Xi = \{2X5\}$.

Automat pomocou pravidla **2:** $qa \to aq$ vkladá na zásobník terminál a. Pri každej aplikácii tohto pravidla gramatika G aplikuje pravidlo **2:** $X \to 3X4$, $\Xi = \{23X45\}$. Automat M a gramatika G budú vykonávať túto činnosť dokiaľ sa na vstupnej páske bude nachádzať terminál a. Po skončení týchto iterácií bude použité pravidlo odstránené $\Xi = \{3^n X 4^n 5\}$ a aplikované posledné pravidlo gramatiky G **3:** $S \to \varepsilon$, $\Xi = \{3^n 4^n 5\}$.

Ako ďalšie začne aplikovať pravidlo **3:** $aqb \to q$, ktoré zabezpečuje prijatie rovnakého počtu symbolov a a b. Počet použití tohto pravidla opäť reguluje riadiaci jazyk automatu $\Xi = \{3^n 4^n 5\}$. Symbol a je tentokrát čítaný zo zásobníka automatu a symbol b zo vstupnej pásky. Pri každom aplikovaní pravidla sú tieto symboly a použité pravidlo odstránené. Po vykonaní tejto časti $\Xi = \{4^n 5\}$.

Syntaktický analyzátor pokračuje v spracovávaní reťazca, kde overí počet výskytov podreťazca $cccc$ a to aplikáciou pravidla **4**: $qc \to q$, ktoré redukuje vstupný reťazec a spracováva symbol c. Analogicky symbol a použité pravidlá sú odstránené. Po vykonaní tejto časti je vstupná páska prázdna a $\Xi = \{5\}$.

V tomto bode by sme mohli syntaktickú analýzu ukončiť a prehlásiť, že reťazec patrí danému jazyku. Avšak prijímaným jazykom je jazyk $L(M, \Xi, 3)$, teda jazyk je akceptovaný prázdnym zásobníkom a finálnym stavom. Preto riadiaci jazyk obsahuje posledné pravidlo $\Xi = \{5\}$.

Aplikujeme a odstránime posledné pravidlo **5:** $Sq \to f$, $\Xi = \{\}$. Ak bola doposiaľ syntaktická analýza úspešná, overíme, či sa automat nachádza vo finálnom stave a či je zásobník prázdny. Ak sú všetky podmienky splnené, syntaktická analýza je správna a reťazec je akceptovaný automatom.

Všimnime si, že pri generovaní reťazca gramatikami G a Q bol postup odlišný. Najskôr sa vygenerovali dve časti reťazca $a^n..c^n$ - *prefix* a *sufix*(2.1.4), ktoré boli vygenerované

zároveň. Následne bol vytvorený podla postupnosti pravidiel riadiaci jazyk Ξ pre vygenerovanie podreťazca b^n. Na začiatku sme teda pomocou gramatík generovali $a^n c^n$ a b^n, teraz spracujeme reťazec v tomto poradí $a^n b^n$ a c^n.

V nasledujúcej tabuľke sa nachádza akceptovanie reťazca *aaaabbbbcccc*, ktorý sme vygenerovali v predchádzajúcej sekcii.

Zásobník	**Stav**	**Vstupná páska**	**Pravidlo** (M)	**Riadiaci jazyk** Ξ	**Pravidlo** (G)
\$S	q	aaaabbbbcccc\$	$[1{:}Ss \to q]$	$\Xi = \{12X5\}$	$[1{:}\ S \to 12X5]$
\$Sa	q	aaabbbbcccc\$	$[2{:}\ qa \to aq]$	$\Xi = \{23X45\}$	$[2{:}\ X \to 3X4]$
\$Saa	q	aabbbbcccc\$	$[2{:}\ qa \to aq]$	$\Xi = \{233X445\}$	$[2{:}\ X \to 3X4]$
\$Saaa	q	abbbbcccc\$	$[2{:}\ qa \to aq]$	$\Xi = \{2333X4445\}$	$[2{:}\ X \to 3X4]$
\$Saaaa$\odot$	q	bbbbcccc\$	$[2{:}\ qa \to aq]$	$\Xi = \{333344445\}$	$[3{:}\ S \to \varepsilon]$
\$Saaa	q	bbbcccc\$	$[3{:}\ aqb \to q]$	$\Xi = \{\mathbf{3}33344445\}$	
\$Saa	q	bbcccc\$	$[3{:}\ aqb \to q]$	$\Xi = \{\mathbf{3}3344445\}$	
\$Sa	q	bcccc\$	$[3{:}\ aqb \to q]$	$\Xi = \{\mathbf{3}344445\}$	
\$S	q	cccc\$	$[3{:}\ aqb \to q]$	$\Xi = \{\mathbf{3}44445\}$	
\$S	q	ccc\$	$[4{:}\ qc \to q]$	$\Xi = \{\mathbf{4}4445\}$	
\$S	q	cc\$	$[4{:}\ qc \to q]$	$\Xi = \{\mathbf{4}445\}$	
\$S	q	c\$	$[4{:}\ qc \to q]$	$\Xi = \{\mathbf{4}45\}$	
\$S	q	\$	$[4{:}\ qc \to q]$	$\Xi = \{\mathbf{4}5\}$	
\$	f	\$	$[5{:}\ Sq \to f]$	$\Xi = \{\mathbf{5}\}$	
\$	f	\$		$\Xi = \{\varepsilon\}$	

Značka $\odot$ značí vykonanie „otáčky“ automatom. Ak by sme demonštrovali činnosť jedno-otáčkového automatu (5.2.1), automat by po aplikácii sekvencie pravidiel typu **2** vykonal otočku. V tomto okamihu zásobník dosiahol najväčšiu dĺžku a ďalej sa už nebude zväčšovať, pretože sa budú aplikovať pravidlá, ktoré vykonávajú redukciu. Tento automat môžeme taktiež považovať za **jedno-otáčkový automat regulovaný riadiacim lineárnym jazykom**.

Kapitola 8

Implementácia

V tejto kapitole si popíšeme spôsob, akým bol implementovaný jednoduchý syntaktický analyzátor.

Prvotnou ideou bolo implementovať jeden program, ktorého činnosť by sa dala rozdeliť do dvoch fáz. V prvej fáze by bol generovaný reťazec pomocou frázovej gramatiky, simulovanej dvomi lineárnymi gramatikami 3.1.1. V druhej fáze by bol tento reťazec prijatý syntaktickým analyzátorom založeným na zásobníkovom automate, ktorý reguluje použitie pravidiel lineárnym jazykom.

V priebehu implementácie vznikla myšlienka rozdeliť tieto fázy na dva oddelené celky. Dôvodom bolo ukázať činnosť simulovanej frázovej gramatiky a schopnosť touto metódou generovať jazyk, ktorý sa skladá z neobmedzeného počtu symbolov a každý symbol sa v ňom nachádza práve n-krát:

$$L = \{symbol_1^n symbol_2^n \dots symbol_y^n : n >= 1 \ \texttt{a} \ y >= 2\}.$$

V druhej fáze je zase zaujímavé sledovať chovanie automatu. Či už ako sa tento automat vysporiada s prijatím reťazca, ktorý nie je správny alebo so správnym reťazcom rôznej dĺžky. Automat je implementovaný pre jazyk[1]

$$L(M) = \{a^n b^n c^n : n \geq 1\}.$$

V rámci tejto bakalárskej práce boli implementované dva programy - `re_grammar` a `parser` v jazyku `C`, ktoré boli riešené formou konzolovej aplikácie. Tieto programy sú multiplatformové a boli testované na operačných systémoch Linux a Windows.

Každý program sa skladá z viacerých modulov. Dva moduly sú spoločné pre obidve časti. Prvý modul `stack.c` obsahuje implementáciu zásobníka a operácie nad touto štruktúrou. Druhý modul `color_text.c` je pomocný a využíva sa pri výpise farebného textu na štandardný výstup `STDOUT`.

8.1 Simulácia frázovej gramatiky

Frázová gramatika je implementovaná programom `re_grammar`, ktorý pozostáva z piatich modulov. Dva z nich už boli zmienené a sú zdielané s programom `parser`. Zvyšné tri si popíšeme v tejto sekcii.

[1] Analýza akceptovania zložitejších jazykov touto metódou je nad rámec tejto práce.

Vstupy

Vstupné dáta sú zadané pomocou argumentov programu. Vstupom je jazyk, pre ktorý sa bude generovať reťazec. Ako `argument 1` sa zadá reťazec symbolov jazyka a ako `argument 2` sa zadá n - koľko krát sa má každý symbol v reťazci nachádzať.
Príklad vstupu:

```
$ ./re_grammar abc 3
```

Výstupy

- Reťazec. Ak je tento reťazec rodiny, ktorú prijíma parser, tak je vytvorený súbor `success.txt` s týmto reťazcom.
- Vygenerované lineárne gramatiky, ktoré simulujú frázovú gramatiku.
- Derivačný strom, ktorý vznikol pri generovaní reťazca.

Hlavný modul - `main.c`

Úlohou tohto modulu je spracovanie vstupných dát, prípadné vypísanie nápovedy a inicializácia základného modulu `re_grammar.c`.

Frázová gramatika - `re_grammar.c`

Ako už bolo spomenuté v tejto časti je simulovaná frázová gramatika pomocou dvoch lineárnych gramatík.

Činnosť tejto gramatiky je popísaná pseudokódom 1. Ako môžeme vidieť na začiatok sa inicializujú tri zásobníky - zásobník pre kontrolný jazyk, pre prefix a pre sufix reťazca. Terminály sú postupne spracovávané gramatikou G.

Algorithm 1 Frázová gramatika simulovaná dvomi regulovanými lineárnymi gramatikami

```
1: Input Terminály, n
2: Inicializuj zásobníky Ξ, prefix, sufix
3: prvý ← pozícia prvého terminálu
4: posledný ← pozícia posledného terminálu
5: while prvý < posledný do
6:     prefix, sufix, Ξ← LINEARGRAMMARG(n, prvý terminál, posledný terminál)
7:     prvý ← prvý - 1
8:     posledný ← druhý - 1
9: if prvý = posledný then
10:    sufix← sufix + LINEARGRAMMARQ(Ξ, terminál)
11: string ← ( reversal(prefix) + sufix )
12: return string
```

Činnosť gramatiky G je zjednodušene popísaná pseudokódom 3. Táto gramatika G je regulovaná vstupným argumentom n, ktorý určuje, koľko krát sa má aplikovať pravidlo tvaru $S \rightarrow terminal1\ S\ terminal2$, ktoré je vygenerované podľa vstupných terminálov. Aplikácia tohto pravidla predstavuje vkladanie $terminal1$ na zásobník $prefix$ a $terminal2$ na zásobník $sufix$.

V prípade, že je volaná gramatika G `linearGrammarG()` prvý krát, pri každej aplikácii vloží identifikátor pravidla gramatiky Q na zásobník riadiaceho jazyka, ktorý bude slúžiť pre reguláciu druhej gramatiky Q.

Algorithm 2 Lineárna gramatika G

```
1: function linearGrammarG(n, prvý terminál, posledný terminál))
2:     while n != 0 do
3:         if Funkcia je volaná prvý krát then
4:             Ξ ← 1
5:         // aplikuj pravidlo 1 tvaru S → prvý terminál S posledný terminál
6:         prefix ← prvý terminál
7:         sufix ← posledný terminál
8:         n ← n - 1
9:     return prefix, sufix, Ξ
```

Ak je počet symbolov jazyka nepárny, teda aktuálne spracovaný terminál má index prvého a zároveň posledného, začne vykonávať svoju činnosť gramatika Q, ktorá generuje podreťazec reťazca a je implementovaná funkciou `linearGrammarQ()`.

Táto gramatika aplikuje svoje jediné pravidlo, dokiaľ zásobník s riadiacim jazykom nebude prázdny. Potom svoju činnosť ukončí, predá riadenie a vygenerovanú časť reťazca **RE** gramatike - `re_grammar()`.

Algorithm 3 Lineárna gramatika Q regulovaná riadiacim jazykom Ξ

```
1: function linearGrammarQ(Ξ, terminál))
2:     while Ξ nie je prázdny do
3:         sufix ← terminál
4:         Odstráň použité pravidlo z Ξ
5:     return sufix
```

Tlač výstupu - `print_grammars.c`

Tento modul je pomocný. Obsahuje funkcie, ktoré sú volané v priebehu činnosti `re_grammar()`. Pomocou tohto modulu sa vypíšu vygenerované gramatiky a ich pravidlá, strom, ktorý vzniká pri generovaní reťazca a taktiež výstupný reťazec na štandardný výstup `STDOUT`.

8.2 Syntaktická analýza založená na regulovanom zásobníkovom automate

Syntaktická analýza je implementovaná programom `parser`, ktorý pozostáva zo šiestich modulov. Dva z nich `stack.c` a `color_text.c` už boli zmienené a sú zdielané s programom `re_grammar`. Zvyšné štyri si popíšeme v tejto sekcii.

Vstupy

Vstupné dáta sú zadané pomocou argumentov programu. Vstupom je reťazec, alebo súbor, ktorý obsahuje reťazec, ktorý bude analyzovaný.
Príklad vstupu:

```
$ ./parser aaabbbccc
```

alebo

```
$ ./parser -f string.txt
```

Výstupy

- Proces akceptovania reťazca, zobrazené aplikované pravidlá, stav zásobníku, stav automatu a aktuálny symbol vstupnej pásky.
- Správa, či bol reťazec prijatý/zamietnutý.

Hlavný modul - `main.c`

Úlohou tohto modulu je spracovanie vstupných dát, prípadné vypísanie nápovedy a inicializácia jadra tohto programu - modulu `pushdown_automata.c`.

Regulovaný zásobníkový automat - `pushdown_automata.c`

Jadrom celej syntaktickej analýzy je tento modul. Činnosť syntaktického analyzátora je implementovaná zásobníkovým automatom regulovaným riadiacim lineárnym jazykom.

Tento zásobníkový automat má nasledujúcu štruktúru:

```
typedef struct {
    char     PDA_state;
    tStack [2] PDA_stack;
    tStack   PDA_control_language;
} tPDA;
```

Na začiatku sa inicializuje zásobníkový automat a jeho zásobníky. Ako stav sa nastaví počiatočný stav s. Na zásobník sa vloží symbol značiaci dno zásobníku \$ a počiatočný symbol S. Na zásobník sa aplikáciou lineárnej gramatiky vložia pravidlá $12X5$. Vstupný reťazec sa vloží na zásobník z dôvodu jednoduchšej manipulácie s reťazcom.

Akceptovanie reťazca prebieha v cykle. V každej iterácii sa aplikuje práve jedno pravidlo. Aplikácia prvého pravidla je implementovaná z dôvodu formálnej definície, ale prakticky nemá žiadnu významnejšiu funkčnosť. Aplikovaním tohto pravidla automat prejde do stavu q, vyjme použité pravidlo a začne so spracovaním vstupného reťazcu.

Zvyšná časť algoritmu je podrobne popísaná v 7.2 pri formálnom zostrojení automatu. Preto je k dispozícii v tejto sekcii stručnejší pseudokód 4.

Pravidlá - `rules.c`

V tomto module sa nachádza päť implementovaných pravidiel, ktoré boli uvedené pri zostrojení automatu 7.2. V pseudokóde 4 môžeme vidieť stručný popis každého pravidla.

[2] Štruktúra `tStack` je implementovaná v module `stack.c`

Tlač výstupu - `print_output.c`

Tento modul je pomocný. Obsahuje funkcie, ktoré sú volané v priebehu činnosti `parser()`-a. Pomocou tohto modulu sa vypisuje prehľadná tabuľka, ktorá je generovaná počas akceptovania reťazca automatom.

Algorithm 4 Zásobníkový automat regulovaný lineárnym jazykom

```
1:  Vlož symbol dno zásobníku na zásobník
2:  Vlož symbol štart symbol na zásobník
3:  stav ← s
4:  Ξ ← 12X5
5:  while not Koniec do
6:      Vyber pravidlo
7:      switch pravidlo do
8:          case 1
9:              stav ← q
10:         case 2
11:             while symbol na vstupnej páske = a do
12:                 Vlož symbol zo vstupnej pásky na zásobník
13:                 Odstráň symbol zo vstupnej pásky
14:                 Ξ ← 3X4
15:             Aplikuj pravidlo 3: X → ε na Ξ
16:         case 3
17:             if symbol na vstupnej páske = b and TOP zásobníku = a then
18:                 Vyjmi symbol a zo zásobníku
19:                 Odstráň symbol zo vstupnej pásky
20:         case 4
21:             if symbol na vstupnej páske = c and TOP zásobníku = štart symbol then
22:                 Odstráň štart symbol zo zásobníku
23:                 Odstráň symbol zo vstupnej pásky
24:         case 5
25:             if Vstupná páska je prázdna then
26:                 stav ← f
27:         default
28:             Koniec ← true
29:     Odstráň pravidlo
30: if dno zásobníka and stav = f and vstupná páska a Ξ sú prázdne then
31:     Syntaktická analýza úspešná
32: else
33:     Syntaktická analýza neúspešná
```

Demonštrácia a manuál tohto programu sa nachádzajú v prílohách B a C.

Kapitola 9

Záver

Cieľom bakalárskej práce bolo ukázať, že zásobníkové automaty, ktoré regulujú použitie pravidiel riadiacim jazykom je možné aplikovať v praxi a sú implementovateľné. Boli skúmané automaty, ktoré túto reguláciu riadia regulárnym jazykom, avšak táto regulácia nemá efekt na silu zásobníkového automatu. Táto práca bola predovšetkým zameraná na automaty regulované lineárnym jazykom. Implementácia a aplikácia bola vykonaná na jednoduchom jazyku, pretože analýza zložitejších jazykov je nad rámec tejto práce.

Mnou definovaný cieľ formálne zostrojiť a implementovať syntaktickú analýzu založenú na tejto metóde bol splnený.

K definovaniu a pochopeniu týchto automatov bolo nevyhnutné čitateľovi predstaviť teóriu formálnych jazykov, preto bol tento text členený hierarchicky. Na jednoduchý úvod teórie formálnych jazykov postupne naväzovali zložitejšie definície. Bola stručne predstavená Chomského klasifikácia gramatík, ktorá nás sprevádzala celou prácou. Nevyhnutnosťou pre predstavenie regulovaných automatov bolo uviesť lineárne a ľavo rozšírené frontové gramatiky.

Po gramatikách, ktoré generujú daný jazyk boli uvedené automaty - zariadenia pre prijatie jazykov. Bol predstavený najjednoduchší konečný automat, automat rozšírený o zásobník a jeho verzie, medzi ktoré patria regulované automaty.

Uvedený bol základný typ týchto automatov a špeciálny automat - jedno-otáčkový atomický zásobníkový automat regulovaný lineárnym jazykom, ktorý počas svojej činnosti zväčšuje dĺžku zásobníka. V momente, keď nastane redukcia dĺžky zásobníka, automat vykoná *otáčku* a začne sa vyprázdňovať. V prípade ak sa už dĺžka nezväčší, automat je *jedno-otáčkový*. Klasické zásobníkové automaty regulované riadiacim jazykom môžu vykonať viac týchto *otáčok*. Sila týchto automatov je rovnaká, ale mechanizmus jedno-otáčkového automatu je jednoduchší.

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

Po teoretickej časti nasledovalo zostrojenie syntaktického analyzátoru založeného na zásobníkových automatoch regulovaným lineárnym jazykom. Bolo ukázané, že tieto automaty sú za silou normálneho zásobníkového automatu a, že prijímaný jazyk nie je bezkontextový ale rekurzívne vyčísliteľný.

Rekurzívne vyčísliteľné jazyky generuje frázová gramatiky, ktorú sme simulovali pomocou dvoch lineárnych gramatík. Pôvodnou ideou bolo implementovať simuláciu frázovej gramatiky a syntaktickú analýzu založenú na zásobníkovom automate, ktorý reguluje použitie lineárnym jazykom ako jeden program. Na začiatok by bol vygenerovaný reťazec pre jazyk

$a^n b^n c^n$ a následne by bol tento reťazec spracovaný syntaktickou analýzou. Pri implementácii ale vznikla myšlienka oddeliť tieto dva celky a implementovať ich ako dva navzájom nezávislé programy. Dôvodom bolo pozorovať, ako sa automat bude správať pri prijatí rôznych dĺžok reťazcov a v prípade prijatia reťazca, ktorý nie je syntakticky správny, teda obsahuje rôzny počet jednotlivých symbolov alebo dokonca symboly, ktoré sa v danom jazyku nenachádzajú.

Syntaktický analyzátor bol implementovaný automatom, ktorý je regulovaný jazykom. Tento jazyk generuje lineárna gramatika. Počas kontroly sekvencie prvého symbolu v reťazci, teda aplikácie pravidla automatu, táto gramatika aplikovala pravidlo na riadiaci jazyk. Po spracovaní prefixu reťazca bola vopred známa celá sekvencia pravidiel, ktoré majú byť použité. Práve vďaka tejto skutočnosti už nemuseli byť na zásobník symboly vkladané a aplikovali sa len pravidlá, ktoré dĺžku zásobníka redukovali. Čo je z implementačného hľadiska veľmi výhodné, kvôli réžii spojenej s pridaním záznamov do pamäte.

Počas formálneho zostrojovania automatu regulovaného riadiacim lineárnym jazykom som zistila, že tento automat by pri tomto jednoduchom jazyku nepotreboval žiadne stavy. Takže by mohol mať o dve pravidlá menej a implementácia by bola tým pádom jednoduchšia. Nebolo by potrebné uchovávať stav, kontrolovať stav a prechádzať z jedného stavu do druhého. Pre reguláciu a akceptovanie tohto jazyka je regulovanie pravidla kontrolným jazykom postačujúce. Keďže riadiaci jazyk je vytvorený v priebehu akceptovania prvého symbolu, automat je regulovaný týmto jazykom a preto nie je možné aby sa automat dostal do iného stavu. Identifikátory pravidiel, ktoré sa nachádzajú v riadiacom jazyku by sme si mohli predstaviť ako stavy. Pri vhodnom zostrojení pravidiel by táto skutočnosť mohla platiť aj pri zložitejších jazykoch.

Zaujímavé by bolo skúmať zotavovanie sa z chýb behom syntaktickej analýzy založenej na tejto metóde. Predstavme si, žeby sa chyba nenachádzala hneď v prvej časti reťazca. Teda v tej časti, ktorá sa vkladá na zásobník a počas čoho sa generuje riadiaci jazyk. Tento jazyk určuje sekvenciu pravidiel, ktoré sa budú aplikovať. Takže po akceptovaní sekvencie prvého symbolu máme vopred známe, ako sa bude automat správať. V prípade, ak by nastala situácia, že niektoré pravidlo nie je aplikovateľné, pretože na vstupe je chyba, mohla by sa táto chyba jednoducho podľa tohto pravidla opraviť.

Simulácia frázovej gramatiky bola implementovaná všeobecne, pre jazyk, ktorý má neobmedzený počet symbolov, pričom každý symbol sa v reťazci nachádza práve n- krát. Ďalším možným pokračovaním tejto bakalárskej práce by bolo upraviť automat tak, aby na základe výstupu tejto metódy frázovej gramatiky - teda pravidiel jej dvoch lineárnych gramatík bol zostrojený automat, ktorý by bol schopný akceptovať všetky reťazce, ktoré táto gramatika generuje.

Takže si myslím, že má zmysel pokračovať v skúmaní tejto problematiky a analyzovaní značne komplexnejších jazykov s využitím zásobníkového automatu, ktorý je regulovaný lineárnym riadiacim jazykom.

Literatúra

[1] Češka, M.: *Teoretická informatika*. učební text 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] 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

[4] 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

[5] Meduna, A.; Masopust, T.: Self-Regulating Finite Automata. *Acta Cybernetica*, ročník 18, č. 1, 2007: s. 135–153, ISSN 0324-721X.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8269

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

Prílohy

Zoznam príloh

Príloha A

Obsah CD

K práci je priložené doprovodné CD s nasledujúcou adresárovou štruktúrou:

- `zdrojovy_kod/` – priečinok obsahuje zdrojové kódy implementácie syntaktického analyzátoru a implementácie frázovéj gramatiky v jazyku C.
- `pisomna_praca/` – zdrojové súbory v LaTeXu potrebné na vygenerovanie písomnej časti bakalárskej práce a taktiež výsledný `PDF` súbor.

Príloha B

Demonštrácia programu

Generovania reťazca programom `re_grammar`

Výstup programu `re_grammar` môžeme vidieť na obrázkoch B.1 a B.2.

V prvej časti sa nachádza zápis jazyka, ktorý bol zadaný argumentami programu. V druhej časti je lineárna gramatika G, ktorá bola vygenerovaná na základe vstupu.

Následne je zobrazený priebeh generovania reťazca derivačným stromom. Pri generovaní prvého symbolu reťazca a posledného je taktiež vygenerovaný riadiaci jazyk (`CONTROL L`).

Ďalej je vygenerovaná gramatika Q a podreťazec, ktorý vznikol aplikáciou pravidiel tejto gramatiky, ktorú reguloval lineárny jazyk (`CONTROL L`).

V prípade, že vygenerovaný reťazec patrí jazyku $L(M)$, ktorý prijíma parser, program vytvorí textový súbor `success.txt` v ktorom sa nachádza tento reťazec.

Na obrázku B.2 je možné vidieť činnosť len jednej gramatiky. Ak je počet terminálov párny reťazec generuje len jedna gramatika.

Akceptovanie reťazca programom `parser`

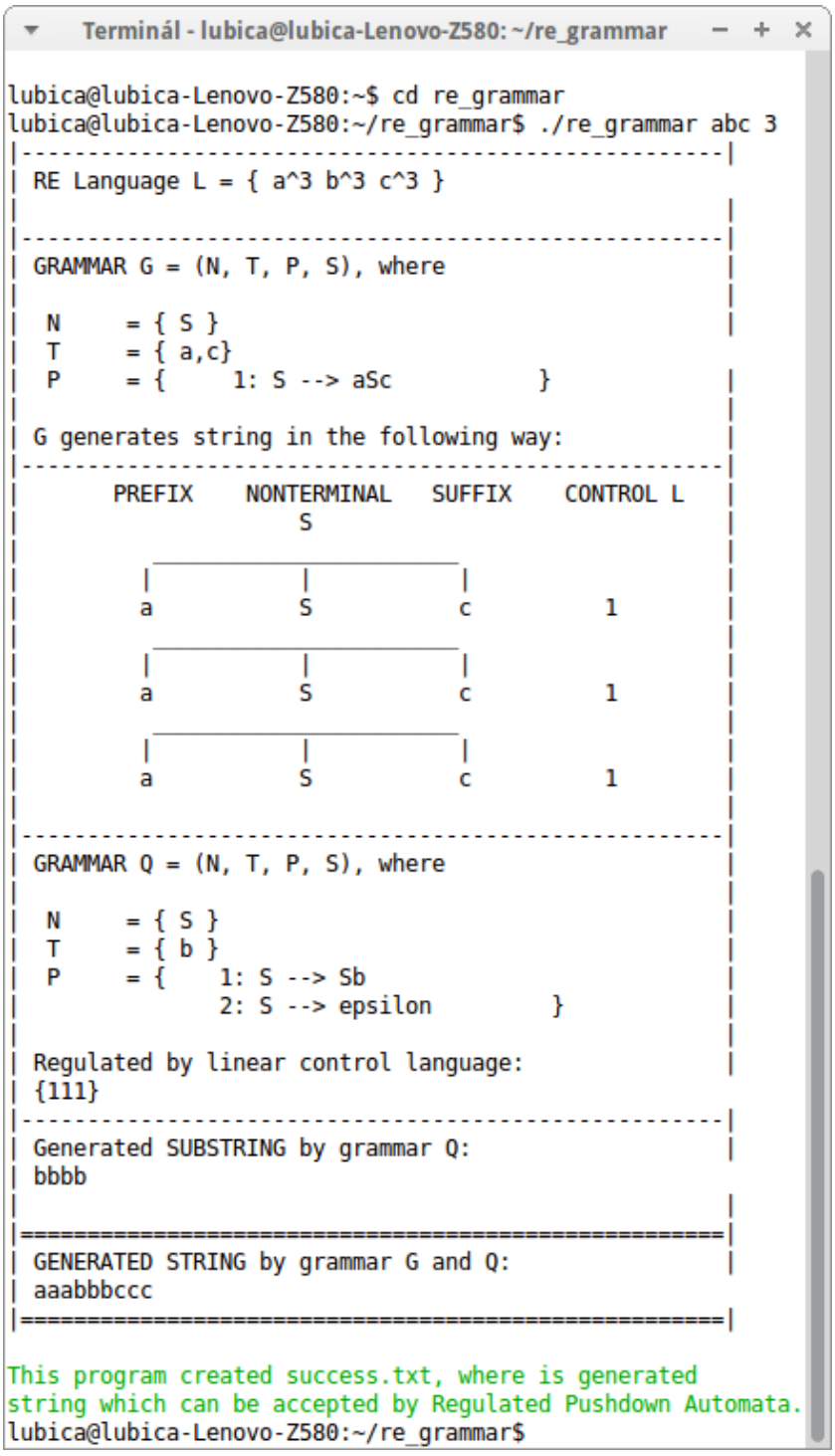
Výstup programu `parser` môžeme vidieť na obrázkoch B.3 a B.4.

Výstup pozostáva z tabuľky, ktorá zobrazuje aktuálny symbol na zásobníku automatu, dĺžku zásobníku, stav automatu, aktuálny symbol, ktorý sa nachádza na vstupnej páske a pravidlo, ktoré bolo v aktuálnom kroku aplikované.

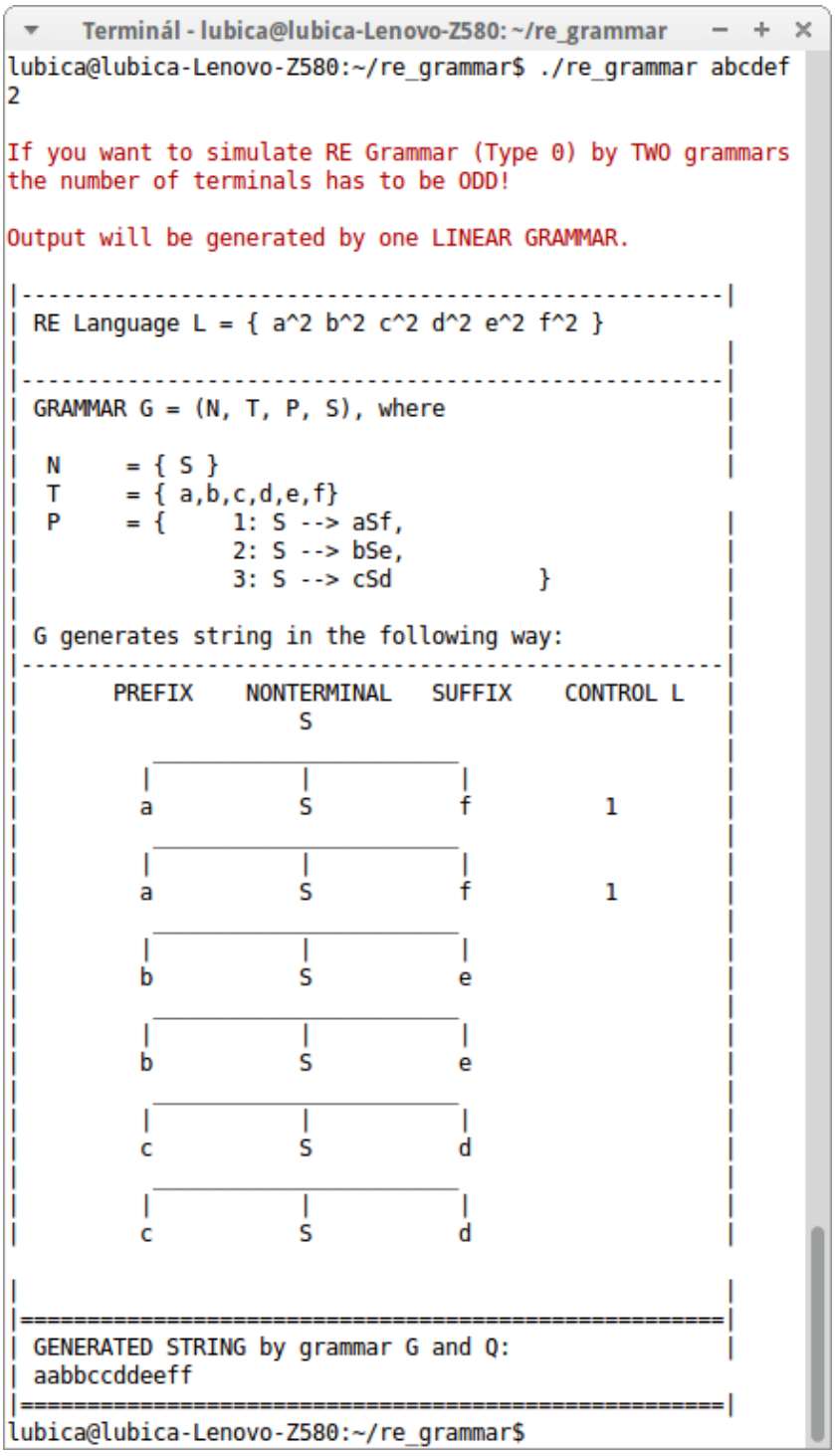
Na konci výstupu je vypísaný stav automatu a riadiaceho jazyka. Na obrázku B.4 môžeme vidieť, že nám zostáva aplikovať posledné pravidlo riadiaceho jazyka (`CONTROL LANGUAGE`), ale toto pravidlo nie je možné aplikovať. Toto pravidlo slúži pre odstránenie zásobníkového symbolu S zo zásobníku a prevedenia automatu do finálneho stavu f. To však ale nie je možné aplikovať. Vstupná páska nie je prázdna. Syntaktická analýza je neúspešná, pretože tento automat prijíma jazyk finálnym stavom a prázdnym zásobníkom.

Generovania príkladov `generate_examples.sh`

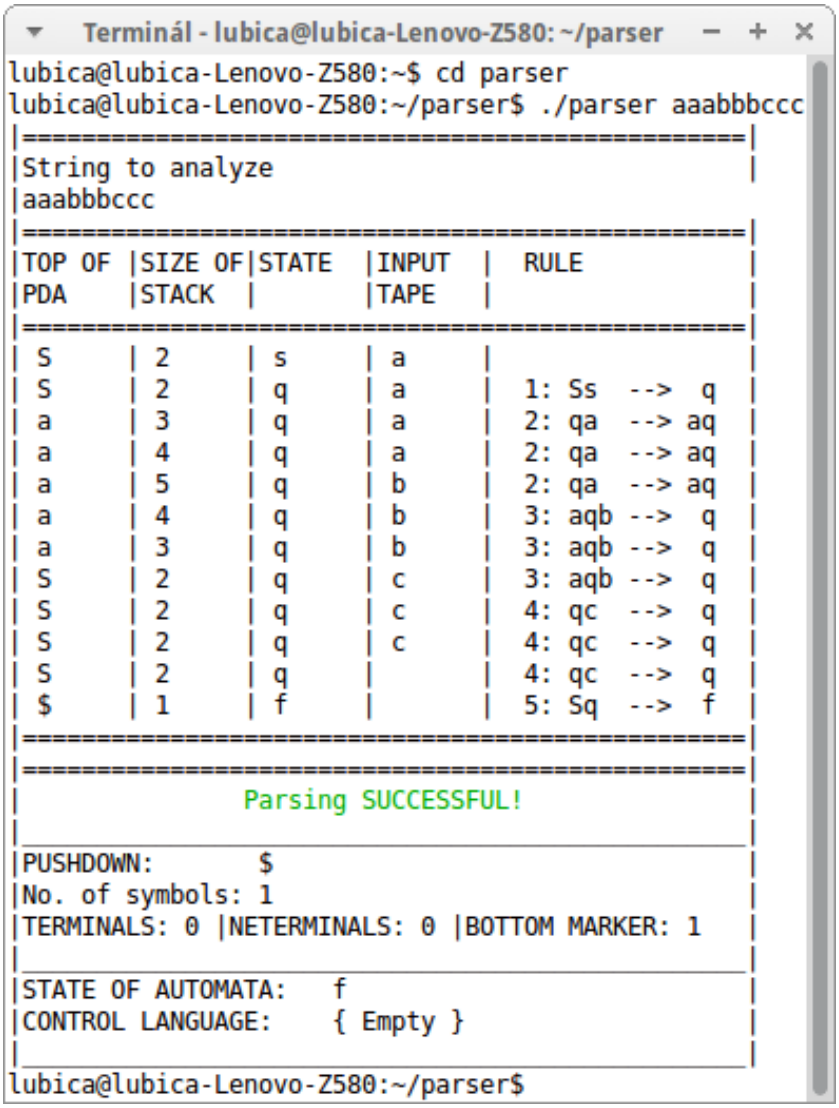
Pre vygenerovanie príkladov použitia týchto programov `re_grammar` a `parser` je pripravený skript `generate_examples.sh`. Tento skript generuje príklady do adresára `examples`. Volá programy `re_grammar` a `parser` s rôznymi argumentami. Výstup `STDOUT` presmerováva do textových súborov. V prípade ak nebol niektorý program skompilovaný, program skompiluje.



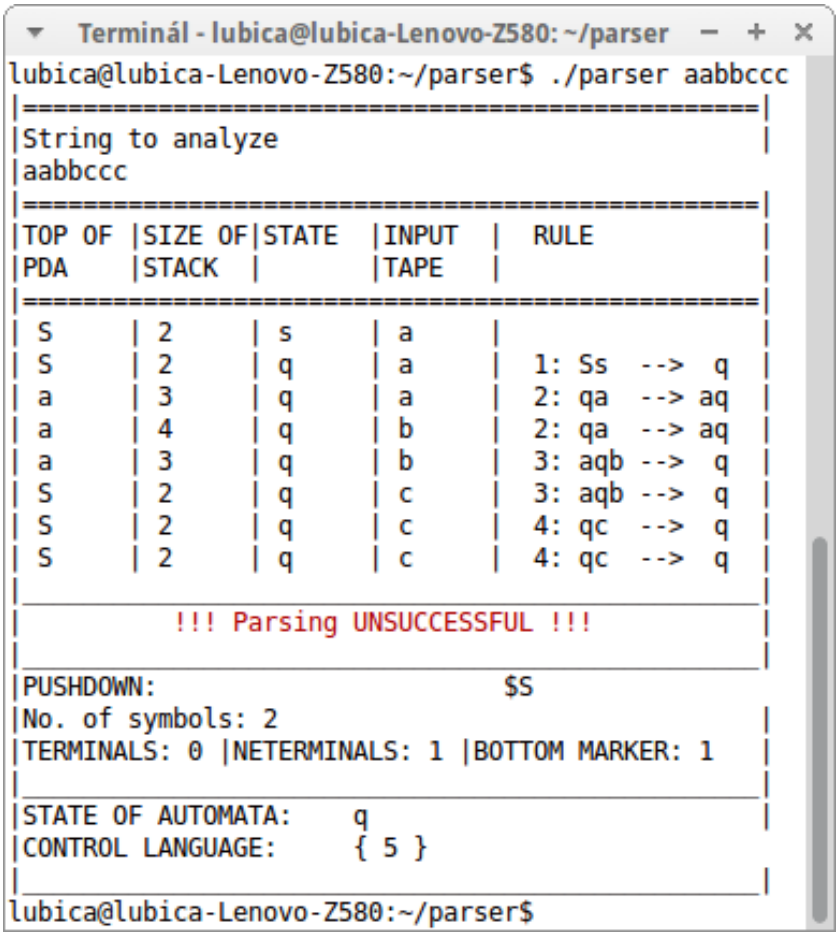
Obr. B.1: Generovanie reťazca, ktorý je vhodný ako vstup pre parser



Obr. B.2: Reťazec generovaný jednou lineárnou gramatikou



Obr. B.3: Akceptovanie reťazca, ktorý je syntakticky správny



Obr. B.4: Akceptovanie reťazca, ktorý nie je syntakticky správny

Príloha C

Manuál

Tento projekt sa skladá z dvoch častí.

1. `re_grammar` Táto časť generuje reťazce rekurzívne vyčísliteľných jazykov tvaru: $(symbol_1)^n(symbol_2)^n....(symbol_y)^n$, kde $n \geq 1$ a $y \geq 2$.

2. `parser` Program akceptuje reťazce rekurzívne vyčísliteľného jazyka: $L(M) = \{a^n b^n c^n\}$, kde $n \geq 1$

Programy boli vyvíjané pod operačným systémom Linux, distribúcia Ubuntu 14.04.3 LTS. Boli skompilované, spustené a testované pod operačnými systémami Windows a Linux.

Kompilácia

Programy sú dodané vo forme zdrojových kódov jazyka `C`, spolu so súborom `Makefile` (pre každý program vlastný). Na skompilovanie programov je potrebný program `make` a kompilátor `gcc`.

Použitie programu `re_grammar`

Vstupné dáta sú zadané pomocou argumentov programu. Vstupom je jazyk, pre ktorý sa bude generovať reťazec. Ako `argument 1` sa zadá reťazec symbolov jazyka a ako `argument 2` sa zadá n - koľko krát sa má každý symbol v reťazci nachádzať.
Príklad vstupu:

```
$ ./re_grammar abc 3
```

Použitie programu `parser`

Vstupné dáta sú zadané pomocou argumentov programu. Vstupom je reťazec, alebo súbor, ktorý obsahuje reťazec, ktorý bude analyzovaný.
Príklad vstupu:

```
$ ./parser aaabbbccc
```

alebo

```
$ ./parser -f string.txt
```