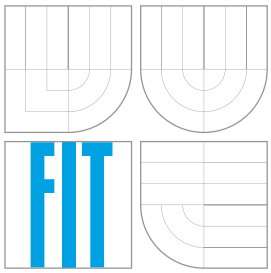
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NOVÉ VERZE AUTOMATOVÝCH A GRAMATICKÝCH SYSTÉMŮ

NEW VERSIONS OF AUTOMATA AND GRAMMAR SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. LUKÁŠ LICHOTA
AUTHOR

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

BRNO 2016

Abstrakt

Táto práca sa zaoberá modifikáciou klasických gramatických systémov na báze bezkontextových gramatík a zavádza nové, modifikované lineárne gramatické systémy na báze gramatík lineárnych. Rovnako sa zaoberá podobnou modifikáciou v automatových systémoch, kde jednotlivé automaty odpovedajú svojou generatívnou silou lineárnym gramatikám, čo spĺňajú jednoobrátkové zásobníkové automaty. Práca ponúka taktiež pohľad na programovú realizáciu teoretických modelov navrhnutých v teoretickej časti a predstavuje dva programy pre syntaktickú analýzu založenú na nich.

Abstract

This work deals with the modification of classical grammar systems with context-free base and defines new, modified linear grammar systems with linear grammar base. Also it deals with modification in automata systems, where each automaton is generative as strong as linear grammar, which is satisfied with the usage of one-turn push-down automata. This thesis also contains view on program realisation of theoretical models, which were described in theoretical part and introduces two programs for syntax analysis built based on it.

Kľúčové slová

gramatické systémy, lineárne gramatické systémy, automatové systémy, SP automatové systémy, syntaktická analýza

Keywords

grammar systems, linear grammar systems, automata systems, SP automata systems, parsing

Citácia

LICHOTA, Lukáš. *Nové verze automatových a gramatických systémů*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Meduna Alexander.

Nové verze automatových a gramatických systémů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána prof. RNDr. Alexandra Medunu, CSc. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Lukáš Lichota
24. mája 2016

Poďakovanie

Týmto by som sa rád poďakoval prof. RNDr. Alexandrovi Medunovi, CSc. za výbornú odbornú pomoc a ochotu pomáhať počas celej tvorby diplomovej práce.

Obsah

Kapitola 1

Úvod

1.1 Cieľ práce

V súčasnej dobe sa v informatike stretávame čím ďalej tým viac s komplexnými problémami a úlohami, na vyriešení ktorých pracuje viacero procesorov, ktoré spolupracujú dohodnutým spôsobom. Objavujú sa tak pojmy ako distribúcia, kooperácia, paralelizmus, komunikácia, či synchronizácia. Nakoľko vo väčšine týchto problémov zohráva úlohu teoretická informatika, či už konkrétnejšie formálne jazyky, vyvstáva výzva pre vytvorenie systému gramatík, resp. automatov, ktoré spolupracujú pre generovanie, resp. rozpoznávanie jazyka [3]. Na scéne teoretickej informatiky sa preto obavujú gramatické a automatové systémy, ktoré túto výzvu premenili na realitu v podobe CD a PC gramatických systémov, ktorých základným prvkom sa stali bezkontextové gramatiky a automatové systémy, ktoré pracujú s rôznymi typmi a počtami automatov.

Táto práca mení klasické poňatie gramatických systémov a transformuje ho na gramatické systémy, ktoré nie sú nutne založené len na báze bezkontextových gramatík, ale definuje nové, lineárne gramatické systémy založené na báze lineárnych gramatík. Pri prvotnej myšlienke by sa mohlo zdať, že generatívna sila takýchto gramatických systémov bude príliš oslabená oproti klasickým gramatickým systémom a vyššia ako pri bežných lineárnych gramatikách, čo však táto práca vyvracia. Získané teoretické poznatky a závery vyvodené z tejto časti, boli použité do praktickej programovej realizácie, ktorá vyústila do vytvorenia programu Linear Grammar System Creator, ktorý ponúka možnosť vytvorenia ako CD, tak aj PC lineárneho gramatického systému. S vytvoreným systémom je potom možné prevádzať syntaktickú analýzu a študovať proces generovania hľadaného reťazca. Vyvodené závery a myšlienkové postupy budú stále demonštrované na príkladoch.

Vzhľadom na sústredenie sa tejto práce na oblasť lineárnych jazykov, aj automaty zvolené pre automatové systémy budú také, aby ich generatívna sila v základnej podobe odpovedala práve lineárnym gramatikám. Tomuto špecifiku odpovedajú jedoobrátkové zásobníkové automaty, pričom budeme skúmať, ako nad nimi vytvoriť systém, ktorý by riadil viacero takýchto automatov s cieľom zvýšenia generatívnej sily. V tejto práci sa zameriame na definíciu a špecifikáciu takéhoto systému, pričom budeme vychádzať z poznatkov nadobudnutých pri skúmaní linárnych gramatických systémov. Výsledkom bude tzv. SP automatový systém, ktorý v sebe skĺbuje kooperatívny a paralelný prístup k riadeniu automatov. Rovnako ako v prípade lineárnych gramatických systémov, aj pri SP automatových systémoch zúžitkujeme teoretické poznatky a premeníme ich na implementáciu do programu SP Automaton System Creator. Podobne, ako pri programe Linear Grammar System Creator, bude možné nadefinovať si SP automatový systém a prevádzať synatakitckú analýzu nad

reťazcami a podrobne skúmať výpočet SP automatového systému.

1.2 Štruktúra dokumentu

Najprv zadefinujeme základné pojmy a definície, budeme postupovať od najzákladnejších a najjednoduchších k zložitejším a prejdeme všetok teoretický základ, potrebný pre zadefinovanie gramatických a automatových systémov, ktoré budú definované v ďalšej kapitole.

Ďalej zadefinujeme modifikované – CD a PC lineárne gramatické systémy a predstavíme ich vlastnosti, hlavne generatívnu silu a porovnáme ju s generatívnou silou klasických gramatických systémov na báze bezkontextových gramatík. Na základe nadobudnutých poznatkov v oblasti gramatických systémov, vytvoríme definíciu SP automatových systémov a zameriame sa na skúmanie rovnakých skutočností, ako v prípade gramatických. Pre lepšie nahliadnutie do problematiky si predstavíme iné modifikácie klasických automatov, pomocou ktorých dochádza k zvýšeniu generatívnej sily, čím sme sa pri tvorbe nášho systému inšpirovali.

Po vyvedení všetkých teoretických poznatkov pristúpime k ich programovej realizácii, kde sa najprv budeme venovať transformácii teórie CD a PC lineárnych gramatických systémov do programovej podoby, vyústením čoho bude spomínaný program Linear Grammar System Creator. Rovnaký postup zvolíme aj v prípade SP automatových systémov a programovým výstupom bude program SP Automaton System Creator. Na záver popíšeme priebežné testovanie týchto programov, ako aj finálne testovanie správnosti ich fungovania.

Kapitola 2

Základné definície a pojmy

V tejto kapitole postupne uvedieme definície jednotlivých pojmov, ktoré budeme následne používať v ďalšom texte. Začneme jednoduchými základnými definíciami, akými sú definícia abecedy, reťazca a unárných a binárnych operácií nad reťazcami, menovite jeho dĺžka, konkatenácia dvoch reťazcov, mocnina reťazca, jeho reverzácia, prefix, sufix a podreťazec. Následne zavedieme definíciu pojmu formálneho jazyka a rozdelíme jazyky na konečné a nekonečné a podobne ako pri reťazci, zadefinujeme aj operácie nad jazykmi, zjednotenie, prienik, rozdiel, konkatenáciu, doplnok, mocninu a iteráciu.

Po uchopení významu základných pojmov pristúpime k definícii gramatiky a jednotlivým variantám gramatík, ktoré sa líšia tvarom pravidiel a svojou generatívnou silou. Pri každej si uvedieme okrem jej definície aj definíciu priamej derivácie, čo následne zovšeobecníme do definície sekvencie derivácií, zadefinujeme pojem vetná forma a popíšeme jazyk generovaný danou gramatikou.

Posledná časť tejto kapitoly bude patriť definíciam automatov, kde popíšeme ich definíciu, konfiguráciu, prechod a následne sekvenciu prechodov a jazyk nimi prijímaný. Taktiež popíšeme ich vzťah vzhľadom ku gramatikám a nakoniec pre zhrnutie a prehľad zavedieme a graficky znázorníme Chomského hierarchiu jazykov.

2.1 Definície základných pojmov

Definícia abecedy

Abeceda je konečná neprázdna množina elementov, ktoré nazývame symboly (viď [6]).

Definícia reťazca

Nech Σ je abeceda. Potom:

- ε je reťazec nad abecedou Σ.
- Pokiaľ x je reťazec nad abecedou Σ a $a \in \Sigma$, potom xa je reťazec nad abecedou Σ.

Poznámka: Symbol ε značí *prázdny reťazec*, teda taký, ktorý neobsahuje žiadny symbol. Symbolom Σ^* budeme značiť množinu všetkých reťazcov nad abecedou Σ.
(viď [6])

Definícia dĺžky reťazca

Nech x je reťazec nad abecedou Σ. Dĺžka reťaza x, $|x|$, je definovaná nasledovne:

- Pokiaľ $x = \varepsilon$, potom $|x| = 0$.
- Pokiaľ $x = a_1 \dots a_n$, potom $|x| = n$, pre $n \geq 1$ a $a_i \in \Sigma$ pre všetky $i = 1, \dots, n$.

(viď [6])

Definícia konkatenácie reťazcov

Nech x a y sú dva reťazce nad abecedou Σ. Konkatenácia reťazcov x a y je reťazec xy.
(viď [6])

Definícia mocniny reťazca

Nech x je reťazec nad abecedou Σ. Pre $i \geq 0$, $i-$tá mocnina reťaca x, x^i, je definovaná nasledovne:

- $x_0 = \varepsilon$
- pre $i \geq 1 : x^i = xx^{i-1}$

(viď [6])

Definícia reverzácie reťazca

Nech x je reťazec nad abecedou Σ. Reverzácia reťaza x, $reversal(x)$, je definovaná nasledovne:

- Pokiaľ $x = \varepsilon$, potom $reversal(x) = \varepsilon$.
- Pokiaĺ $x = a_1, \dots, a_n$ potom $reversal(a_1, \dots, a_n) = a_n, dotsc, a_1$ pre $n \geq 1$ a $a_i \in \Sigma$ pre všetky $i = 1, \dots, n$.

(viď [6])

Definícia prefixu reťazca

Nech x a y sú dva reťazce nad abecedou Σ. Potom x je prefixom y, pokiaľ existuje reťazec z nad abecedou Σ, pričom platí $xz = y$.
(viď [6])

Definícia sufixu reťazca

Nech x a y sú dva reťazce nad abecedou Σ. Potom x je prefixom y, pokiaľ existuje reťazec z nad abecedou Σ, pričom platí $zx = y$.
(viď [6])

Definícia podreťazca

Nech x a y sú dva reťazce nad abecedou Σ. Potom x je podreťazec y, pokiaľ existujú reťazce z, z' nad abecedou Σ, pričom platí $zxz' = y$.
(viď [6])

Definícia formálneho jazyka

Nech je daná abeceda Σ a nech Σ^* značí množinu všetkých reťazcov nad abecedou Σ. Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad abecedou Σ.
(viď [5])

Definícia konečného a nakonečného jazyka

Jazyk L je konečný, pokiaľ L obsahuje konečný počet reťazcov, inak je nekonečný.
(viď [5])

2.2 Operácie nad jazykmi

Definícia zjednotenia dvoch jazykov

Nech L_1 a L_2 sú dva jazyky nad abecedou Σ. Zjednotenie jazykov L_1 a L_2, $L_1 \cup L_2$ je definované nasledovne:

$$L_1 \cup L_2 = \{x : x \in L_1 \vee x \in L_2\}$$

(viď [6])

Definícia prieniku dvoch jazykov

Nech L_1 a L_2 sú dva jazyky nad abecedou Σ. Prienik jazykov L_1 a L_2, $L_1 \cap L_2$ je definovaný nasledovne:

$$L_1 \cap L_2 = \{x : x \in L_1 \wedge x \in L_2\}$$

(viď [6])

Definícia rozdielu dvoch jazykov

Nech L_1 a L_2 sú dva jazyky nad abecedou Σ. Rozdiel jazykov L_1 a L_2, $L_1 - L_2$ je definovaný nasledovne:

$$L_1 - L_2 = \{x : x \in L_1 \wedge x \notin L_2\}$$

(viď [6])

Definícia konkatenácie dvoch jazykov

Nech L_1 a L_2 sú dva jazyky nad abecedou Σ. Konkatenácia jazykov L_1 a L_2, $L_1 L_2$ je definovaná nasledovne:

$$L_1 L_2 = \{xy : x \in L_1 \wedge y \in L_2\}$$

(viď [6])

Definícia doplnku jazyka

Nech L je jazyk nad abecedou Σ. Doplnok jazyka L, $\overline{L}$ je definovaný nasledovne:

$$\overline{L} = \Sigma^* - L$$

(viď [6])

Definícia mocniny jazyka

Nech L je jazyk nad abecedou Σ. Pre $i \geq 0$, i-tá mocnina jazyka L, L^i je definovaná nasledovne:

- $L^0 = \{\varepsilon\}$
- Pre $i \geq 1 : L^i = LL^{i-1}$

(viď [6])

Definícia iterácie jazyka

Nech L je jazyk nad abecedou Σ. Iterácia jazyka L, L^* je definovaná nasledovne:

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

(viď [6])

Definícia pozitívnej iterácie jazyka

Nech L je jazyk nad abecedou Σ. Pozitívna iterácia jazyka L, L^+ je definovaná nasledovne:

$$L^+ = \bigcup_{n=1}^{\infty} L^n$$

(viď [6])

2.3 Základné typy gramatík

Definícia neobmedzenej gramatiky

Neobmedzená gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- P je konečná množina pravidiel tvaru $x \to y$, kde $x \in (N \cap T)^* N (N \cap T)^*$ a $y \in (N \cap T)^*$.
- S je počiatočný neterminálny symbol.

Poznámka: Množinu všetkých jazykov, ktoré sú generované nejakou neobmedzenou gramatikou, nazveme *triedou rekurzívne vyčísliteľných jazykov*, alebo taktiež *triedou jazykov typu 0*.
(viď [5])

Definícia priamej derivácie pri neobmedzenej gramatike

Nech $G = (N, T, P, S)$ je neobmedzená gramatika, nech $u, v \in (N \cap T)^*$ a $p = x \to y \in P$ je pravidlo. Potom hovoríme, že uxv priamo derivuje uyv podľa pravidla p a zapisujeme $uxv \Rightarrow uyv$ $[p]$, alebo skrátene $uxv \Rightarrow uyv$.
(viď [5])

Definícia sekvencie derivácií pri neobmedzenej gramatike

Nech $G = (N, T, P, S)$ je neobmedzená gramatika.

- Nech $u \in (N \cup T)^*$. Potom hovoríme, že u derivuje v 0-krokoch u a zapisujeme $u \Rightarrow^0 u\ [\varepsilon]$, alebo skrátene $u \Rightarrow^0 u$.
- Nech $u_0, u_1, \ldots, u_n \in (N \cup T)^*$, nech pre všetky $i = 1, \ldots, n$ platí $u_{i-1} \Rightarrow u_i\ [p_i]$. Potom hovoríme, že u_0 derivuje v n-krokoch u_n a zapisujeme $u_0 \Rightarrow^n u_n\ [p_1 p_2 \ldots p_n]$, alebo skrátene $u_0 \Rightarrow^n u_n$.
- Nech $u \Rightarrow^n v\ [\pi]$ pre nejaké $n \geq 1$; $u, v \in (N \cup T)^*$. Potom hovoríme, že u netriviálne derivuje v a zapisujeme $u \Rightarrow^+ v\ [\pi]$, alebo skrátene $u \Rightarrow^+ v$.
- Nech $u \Rightarrow^n v\ [\pi]$ pre nejaké $n \geq 0$; $u, v \in (N \cup T)^*$. Potom hovoríme, že u derivuje v a zapisujeme $u \Rightarrow^* v\ [\pi]$, alebo skrátene $u \Rightarrow^* v$.

(viď [5])

Definícia vetnej formy pri neobmedzenej gramatike

Nech $G = (N, T, P, S)$ je neobmedzená gramatika. Hovoríme, že $u \in (N \cap T)^*$ je vetná forma v neobmedzenej gramatike G práve vtedy, keď $S \Rightarrow^* u$.
(viď [5])

Definícia jazyka generovaného neobmedzenou gramatikou

Nech $G = (N, T, P, S)$ je neobmedzená gramatika. Jazyk generovaný neobmedzenou gramatikou G, $L(G)$, je definovaný nasledovne:

$$L(G) = \{w : w \in T^* \wedge S \Rightarrow^* w\}$$

Trieda jazykov generovaných neobmedzenov gramatikov sa nazýva trieda jazykov typu 0.
(viď [5])

Definícia kontextovej gramatiky

Kontextová gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- P je konečná množina pravidiel tvaru $x \to y$, kde $x \in (N \cap T)^* N (N \cap T)^*$ a $y \in (N \cap T)^*$, pričom $|x| \leq |y|$.
- S je počiatočný neterminálny symbol.

Poznámka: Množinu všetkých jazykov, ktoré sú generované nejakou kontextovou gramatikou, nazveme *triedou kontextových jazykov*, alebo taktiež *triedou jazykov typu 1.*
(viď [5])

Definícia priamej derivácie, sekvencie derivácií a jazyka generovaného kontextovou gramatikou

Definície priamej derivácie, sekvencie derivácií a jazyka generovaného kontextovou gramatikou sú totožné s definíciami uvedenými pri neobmedzenej gramatike.
Trieda jazykov generovaných neobmedzenov gramatikov sa nazýva trieda jazykov typu 1. (viď [5])

Definícia bezkontextovej gramatiky

Bezkontextová gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- P je konečná množina pravidiel tvaru $A \to x$, kde $A \in N$ a $x \in (N \cup T)^*$
- S je počiatočný neterminálny symbol.

Poznámka: Množinu všetkých jazykov, ktoré sú generované nejakou bezkontextovou gramatikou, nazveme *triedou bezkontextových jazykov*, alebo taktiež *triedou jazykov typu 2.*
(viď [6])

Definícia priamej derivácie pri bezkontextovej gramatike

Nech $G = (N, T, P, S)$ je bezkontextová gramatika, nech $u, v \in (N \cup T)^*$ a $p = A \to x \in P$ je pravidlo. Potom hovoríme, že uAv priamo derivuje uxv podľa pravidla p a zapisujeme $uAv \Rightarrow uxv$ $[p]$, alebo skrátene $uAv \Rightarrow uxv$.
(viď [6])

Definícia najľavejšej derivácie pri bezkontextovej gramatike

Nech $G = (N, T, P, S)$ je bezkontextová gramatika, $u \in T^*$ a $v \in (N \cup T)^*$ a $p = A \to x \in P$ je pravido. Potom hovoríme, že uAv priamo derivuje v najľavejšej derivácii uxv podľa pravidla p, a zapisujeme $uAv \Rightarrow_{lm} uxv$ $[p]$, alebo skrátene $uAv \Rightarrow_{lm} uxv$.
(viď [6])

Definícia najpravejšej derivácie pri bezkontextovej gramatike

Nech $G = (N, T, P, S)$ je bezkontextová gramatika, $u \in (N \cup T)^*$ a $v \in T^*$ a $p = A \to x \in P$ je pravido. Potom hovoríme, že uAv priamo derivuje v najpravejšej derivácii uxv podľa pravidla p, a zapisujeme $uAv \Rightarrow_{rm} uxv$ $[p]$, alebo skrátene $uAv \Rightarrow_{rm} uxv$.
(viď [6])

Definícia sekvencie derivácií pri bezkontextovej gramatike

Nech $G = (N, T, P, S)$ je bezkontextová gramatika.

- Nech $u \in (N \cup T)^*$. Potom hovoríme, že u derivuje v 0-krokoch u a zapisujeme $u \Rightarrow^0 u$ $[\varepsilon]$, alebo skrátene $u \Rightarrow^0 u$.

- Nech $u_0, u_1, \ldots, u_n \in (N \cup T)^*$, nech pre všetky $i = 1, \ldots, n$ platí $u_{i-1} \Rightarrow u_i$ $[p_i]$. Potom hovoríme, že u_0 derivuje v n-krokoch u_n a zapisujeme $u_0 \Rightarrow^n u_n$ $[p_1 p_2 \ldots p_n]$, alebo skrátene $u_0 \Rightarrow^n u_n$.
- Nech $u \Rightarrow^n v$ $[\pi]$ pre nejaké $n \geq 1$; $u, v \in (N \cup T)^*$. Potom hovoríme, že u netriviálne derivuje v a zapisujeme $u \Rightarrow^+ v$ $[\pi]$, alebo skrátene $u \Rightarrow^+ v$.
- Nech $u \Rightarrow^n v$ $[\pi]$ pre nejaké $n \geq 0$; $u, v \in (N \cup T)^*$. Potom hovoríme, že u derivuje v a zapisujeme $u \Rightarrow^* v$ $[\pi]$, alebo skrátene $u \Rightarrow^* v$.

(viď [6])

Definícia jazyka generovaného bezkontextovou gramatikou

Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Jazyk generovaný bezkontextovou gramatikou G, $L(G)$, je definovaný nasledovne:

$$L(G) = \{w : w \in T^* \wedge S \Rightarrow^* w\}$$

Trieda jazykov generovaných bezkontextovou gramatikou sa nazýva trieda jazykov typu 2. (viď [6])

Ľavá lineárna gramatika

Ľavá lineárna gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- P je konečná množina pravidiel tvaru $A \to Bx$ alebo $A \to y$, kde $A, B \in N$ a $x, y \in T^*$
- S je počiatočný neterminálny symbol.

(viď [6])

Pravá lineárna gramatika

Pravá lineárna gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- P je konečná množina pravidiel tvaru $A \to xB$ alebo $A \to y$, kde $A, B \in N$ a $x, y \in T^*$
- S je počiatočný neterminálny symbol.

Poznámka: Množinu všetkých jazykov, ktoré sú generované nejakou pravou lineárnou gramatikou, nazveme *triedou regulárnych jazykov*, alebo taktiež *triedou jazykov typu 3.* (viď [6])

Lineárna gramatika

Lineárna gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- P je konečná množina pravidiel tvaru $A \to x$, kde $A \in N$ a $x \in T^*(N \cup \{\varepsilon\})T^*$
- S je počiatočný neterminálny symbol.

(viď [6])

Definícia pramej derivácie, sekvencie derivácií a jazyka generovaného pravou, ľavou alebo všeobecnou lineárnou gramatikou

Definície pramej derivácie, sekvencie derivácií a jazyka generovaného pravou, ľavou alebo všeobecnou lineárnou gramatikou sú totožné s definíciami uvedenými pri bezkontextovej gramatike.
(viď [6])

2.4 Základné typy automatov

Konečný automat

Konečný automat M je pätica:

$$M = (Q, \Sigma, R, s, F), \text{ kde}$$

- Q je konečná množina stavov
- Σ je vstupná abeceda
- R je konečná množina pravidiel tvaru $pa \to q$, kde $p, q \in Q, a \in \Sigma \cup \epsilon$
- $s \in Q$ je počiatočný stav
- F $F \subseteq Q$ je množina koncových stavov

Ak $\forall q \in Q$ a $\forall a \in \Sigma$ platí, že počet pravidiel, ktoré majú ľavú stranu pravidla qa je nanajvýš rovný jednej, nazývame takýto konečným automat deterministickým konečným automatom, čo znamená, že sme stále schopní deterministicky rozhodnúť, ktoré pravidlo automat pri spracovávaní vstupného reťazca použije, aký prechod prevedie. Ak by existovalo čo i len jedno pravidlo, kde by bola táto podmienka porušená, a teda rovnaká ľavá strane pravidla by sa vyskytovala aspoň u dvoch pravidiel, konečný automat by bol nedeterministický. Platí, že každý nedeterministický konečný automat je možné previesť na deterministický konečný automat a zavedením nedeterminizmu nezvýšime generatívnu silu konečného automatu. Význam determinizmu má veľký význam najmä v súvislosti so syntaktickou analýzou, kde potrebujeme rozhodnúť, ktoré pravidlo sa má použiť pri spracovávaní reťazca.
(viď [6])

Konfigurácia konečného automatu

Nech $M = (Q, \Sigma, R, s, F)$ je konečný automat. Konfigurácia konečného automatu M je reťazec $\chi \in Q\Sigma^*$. (viď [6])

Prechod konečného automatu

Nech $M = (Q, \Sigma, R, s, F)$ je konečný automat. Nech pax a qx sú dve konfigurácie konečného automatu M, kde $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$ a $x \in \Sigma^*$. Nech $r = pa \to q \in R$ je pravidlo. Potom M môže previesť prechod z pax do qx za použitia pravidla r, zapisujeme $pax \vdash qx[p]$ alebo zjednodušene $pax \vdash qx$. (viď [6])

Sekvencia prechodov konečného automatu

Nech χ je konfigurácia. M prevedie nula prechodov z χ do χ, zapisujeme $\chi \vdash^0 \chi[\epsilon]$ alebo zjednodušene $\chi \vdash^0 \chi$.
Nech $\chi_0, \chi_1, \ldots, \chi_n$ je sekvencia prechodov konfigurácií pre $n \geq 1$ a $\chi_{i-1} \vdash \chi_i[r_i]$, $r_i \in R$ pre všetky $i = 1, \ldots, n$, čo znamená:

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

Potom M prevedie n prechodov z χ_0 do chi_n, zapisujeme $\chi_0 \vdash^n \chi_n[r_1, \ldots, r_n]$ alebo zjednodušene $\chi_0 \vdash^n \chi_n$.
Pokiaľ $\chi_0 \vdash^n \chi_n[\rho]$ pre nejaké $n \geq 1$, potom $\chi_0 \vdash^+ \chi_n[\rho]$.
Pokiaľ $\chi_0 \vdash^n \chi_n[\rho]$ pre nejaké $n \geq 0$, potom $\chi_0 \vdash^* \chi_n[\rho]$. (viď [6])

Jazyk prijímaný konečným automatom

Nech $M = (Q, \Sigma, R, s, F)$ je konečný automat. Jazyk prijímaný konečným automatom $M, L(M)$, je definovaný:

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

(viď [6])

Vzťah medzi triedou jazykov generovaných konečnými automatmi a triedou jazykov typu 3

Vzťah medzi triedou jazykov L_3 a triedou jazykov generovaných konečnými automatmi, označme túto triedu ako L_M je $L_3 = L_M$. Aby sme toto tvrdenie dokázali, prevedieme jeho dôkaz.

Predpokladajme, že L je jazyk typu 3, potom musí existovať konečný automat M taký, že $L = L(M)$, t.j. $L_3 \subseteq L(M)$. Podľa predošlých poznatkov už vieme, že jazyk L_3 je možné generovať gramatikou $G = (N, T, P, S)$, ktorá je typu 3, čo znamená, že jej pravidlá majú tvar $A \to aB$ alebo $A \to \epsilon, A, B \in N, a \in T$. K tejto gramatike zostrojíme konečný (nedeterministický) automat $M = (Q, \Sigma, R, s, F)$, kde:

- $Q = N$
- $\Sigma = T$
- pre všetky pravidlá $A \to aB \in P$ pridaj pravidlo $Aa \to B$ do R

- $s = S$
- $F = \{A | A \to \epsilon \in P\}$

Teraz ukážeme, že $L(G) = L(M)$. Dôkaz prevedieme matematickou indukciou, kde za indukčný predpoklad zvolíme:

$$\text{pre každé } A \in N \text{ platí } A \Rightarrow_G^{i+1} w, w \in \Sigma^* \text{ práve vtedy, keď } Aw \vdash C\epsilon, \text{ pre nejaké } C \in F$$

Najprv dokážeme túto hypotézu pre $i = 0$. Zrejme platí:

$$A \Rightarrow \epsilon \text{ práve vtedy, keď } A\epsilon \vdash^0 A\epsilon, A \in F$$

Ďalej predpokladajme, že dokazovaná hypotéza platí pre $i > 0$ a položíme $w = ax, a \in \Sigma, |x| = i - 1$. Potom platí $A \Rightarrow^{i+1} w$ práve vtedy, keď $A \Rightarrow aB \Rightarrow^i x$. V dôsledku priamej derivácie $A \Rightarrow aB$ bude množina pravidiel R obsahovať pravidlo $Aa \Rightarrow B$. Na základe induktívnej hypotézy platí $B \Rightarrow^i x$, práve vtedy, keď $Bx \vdash^{i-1} C\epsilon, C \in F$. Ak všetko zhrnieme, dostávame:

$$A \Rightarrow aB \Rightarrow^i ax = w, \text{ práve vtedy, keď } Bx \vdash^{i-1} C\epsilon, C \in F$$

$$\text{t.j. } A \Rightarrow^{i+1} w \text{ práve vtedy, keď } Aw \vdash^i C\epsilon, C \in F$$

Tým sme splnili platnosť induktívneho predpokladu a dokázali sme pre všetky $i > 0$. Špeciálne potom platí:

$$S \Rightarrow^* w \text{ práve vtedy, keď } Sw \vdash^8 C\epsilon, C \in F$$

a teda $L(G) = L(M)$.

V dôkaze pokračujeme nasledovným tvrdením: Nech $L = L(M)$ pre nejaký konečný automat M. Potom existuje gramatika G, ktorá je typu 3 taká, že $L = L(G)$ t.j. $L_M \subseteq L_3$.

Dôkaz prevedieme podobne ako pri predošlom tvrdení. Nech $M = (Q, \Sigma, R, s, F)$ je konečný automat. Pokiaľ je tento automat nedeterministický, vieme, že je možné ho previesť na ekvivalentný deterministický konečný automat, preto predpokladajme, že pracujeme s deterministickým konečným automatom. Nech $G = (N, T, P, S)$ je gramatika, ktorá je typu 3 a definujme jej pravidlá nasledovne:

- ak konečný automat obsahuje pravidlo $qa \to r$, potom množina pravidiel P gramatiky G bude obsahovať pravidlo $q \to ar$
- ak $q \in F$, potom množina pravidiel P gramatiky G bude obsahovať pravidlo $q \to \epsilon$

Dôkaz by prebiehal opäť pomocou matematickej indukcie obdobne ako v predošlom príklade.

Z tvrdení, ktoré sa nám podarilo dokázať, bezprostredne plynie fakt, že trieda jazykov prijímaných konečnými automatmi, je totožná s triedou jazykov typu 3.
(viď [11])

Zásobníkový automat

Zásobníkový automat M je sedmica:

$$M = (Q, \Sigma, \Gamma, R, s, S, F), \text{ kde}$$

- Q je konečná množina stavov
- Σ je vstupná abeceda
- Γ je zásobníková abeceda
- R je konečná množina pravidiel tvaru $Apa \to wq$, kde $A \in \Gamma, p, q \in Q, a \in \Sigma \cup \epsilon, w \in \Gamma^*$
- $s \in Q$ je počiatočný stav
- $S \in \Gamma$je počiatočný symbol na zásobníku
- F $F \subseteq Q$ je množina koncových stavov

Podobne ako v prípade konečných automatov, aj pri zásobníkových automatoch rozlišujeme, či ide o automaty nedeterministické, alebo deterministické. Aby bol zásobníkový automat deterministický, musí platiť, že pre každé pravidlo tvaru $Apa \to wq$ z množiny pravidiel, množina $R - \{Apa \to wq\}$ neobsahuje žiadne pravidlo s ľavou stranou Apa alebo Ap. Pri problematike konečných automatov existoval algoritmus prevodu nedeterministického konečného automatu na ekvivalentný deterministický konečný automat, boli sme teda schopní zachovať generatívnu silu a determinizmus. V prípade zásobníkových automatov to však neplatí a deterministické zásobníkové automaty nedosahujú generatívnu silu ich nedeterministikej varianty.
(viď [6])

Konfigurácia zásobníkového automatu

Nech $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat. Konfigurácia zásobníkového automatu M je reťazec $\chi \in \Gamma^* Q \Sigma^*$.
(viď [6])

Prechod zásobníkového automatu

Nech $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat. Nech $xApay$ a $xwqy$ sú dve konfigurácie zásobníkového automatu M, kde $x, w \in \Gamma^*, A \in \Gamma, p, q \in Q, a \in \Sigma \cup \{\epsilon\}$ a $y \in \Sigma^*$. Nech $r = Apa \to wq \in R$ je pravidlo. Potom M môže previesť prechod z $xApay$ do $xwqy$ za použitia pravidla r, zapisujeme $xApay \vdash xwqy[p]$ alebo zjednodušene $xApay \vdash xwqy$.
(viď [6])

Sekvencia prechodov zásobníkového automatu

Definícia sekvencie prechodov zásobníkového automatu je zhodná s odpovedajúcou definícou v prípade konečného automatu.
(viď [6])

Jazyk prijímaný zásobníkovým automatom

Nech $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat. Jazyk prijímaný zásobníkovým automatom $M, L(M)$, je definovaný:

- Jazyk prijímaný zásobníkovým automatom M prechodom do koncového stavu, značíme $L(M)_f$:
$$L(M) = \{w : w \in \Sigma^*, Ssw \vdash^* zf, z \ \in \Gamma^*, f \in F\}$$
- Jazyk prijímaný zásobníkovým automatom M vyprázdnením zásobníku, značíme $L(M)_\epsilon$:
$$L(M) = \{w : w \in \Sigma^*, Ssw \vdash^* zf, z = \epsilon, f \in Q\}$$
- Jazyk prijímaný zásobníkovým automatom M vyprázdnením zásobníku a prechodom do konečného stavu, značíme $L(M)_{f\epsilon}$:
$$L(M) = \{w : w \in \Sigma^*, Ssw \vdash^* zf, z = \epsilon, f \in F\}$$

(viď [6])

Vzťah medzi triedou jazykov generovaných zásobníkovými automatmi a triedou jazykov typu 2

Vzťah medzi triedou jazykov L_2 a triedou jazykov generovaných konečnými automatmi, označme túto triedu ako L_M je $L_2 = L_M$. Aby sme toto tvrdenie dokázali, prevedieme jeho dôkaz.

Najskôr ukážeme, ako je možné zostrojiť nedeterministický syntaktický analyzátor, ktorý pracuje metódou zhora dole. Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Z tejto gramatiky môžeme skonštrulovať nedeterministický zásobníkový automat M tak, že bude platiť $L_\epsilon(M) = L(G)$. Zásobníkový automat M skonštrulujeme tak, aby vytváral ľavú deriváciu vstupného reťazca v gramatike G. Nech $M = (\{q\}, T, N \cup T, R, q, S, \emptyset)$, kde množina pravidiel R obsahuje nasledujúce pravidlá:

- ak je $A \to \alpha \in P$ potom R obsahuje pravidlo $qA \to q\alpha$
- pre všetky $a \in T$ pridaj do R pravidlo $qaa \to q$

Teraz ukážeme, že $A \Rightarrow^m w$ práve vtedy, keď $qw \vdash^n q$, pre nejaké $m, n > 1$. Indukciou pre m predpokladajme, že $A \Rightarrow^m w$. Ak je $m = 1$ a $w = a_1 \dots a_k, k \geq 0$ potom:

$$qa_1 \dots a_k A \vdash a_1 \dots a_k \vdash^* q$$

Teraz predpokladajme, že platí $A \Rightarrow^m w$ pre $m > 1$. Prvý krok tejto derivácie musí mať tvar $A \Rightarrow X_1 X_2 \dots X_k$, pričom $X_i \Rightarrow^{m_i} x_i$ pre $m_i < m, 1 \leq i \ \leq k$ a $x_1 x_2 \dots x_k = w$, teda:

$$qAw \vdash qX_1 \dots X_k w$$

Ak $X_i \in N$, potom podľa induktívneho predpokladu:

$$qX_i x_i \vdash^* q$$

Ak $X_i = x_i, x_i \in T$ potom:

$$qX_i x_i \vdash q$$

Teraz je už možné vidieť, že ľavej derivácii

$$A \Rightarrow X_1 \dots X_k \Rightarrow^{m_1} x_1 X_2 \dots X_k \Rightarrow^{m_2} \dots \Rightarrow^{m_k} x_1 x_2 \dots x_k = w$$

odpovedá táto postupnosť prechodov

$$qAx_1 \dots x_k \vdash qX_1 \dots X_k x_1 \dots x_k \vdash^* qX_2 \dots X_k x_2 \dots x_k \vdash^* q$$

Ak $qAw \vdash^n q$ potom $\Rightarrow^+ w$ dokážeme indukciou pre n. Pre $n = 1, w = \epsilon$ a $A \to \epsilon$ je pravidlo v P. Predpokladajme, že dokazovaná relácia platí pre všetky $n' < n$. Potom prvý prechod automatu musí mať tvar:

$$qAw \vdash qX_1 \dots X_k w$$

a $qx_1X - 1 \vdash q$ pre $1 \leq i \ \leq k$, kde $w = x_1 \dots x_k$. Potom $A \to X_1 \dots X_k$ je pravidlo z P a derivácia $X_i \Rightarrow^+ x_i$ plynie z induktívneho predpokladu. Ak $X_i \in T$ potom $X_i \Rightarrow^0 x_i$. Teda $A \Rightarrow X_1 \dots X_k \Rightarrow^* x_1 X_2 \dots X_k \Rightarrow^* \dots \Rightarrow^* x_1 \dots x_{k-1} X_k \Rightarrow^* x_1 \dots x_k = w$ je ľavá derivácia reťazca w z A. Ak zoberieme $S = A$ ako špeciálny prípad, dostaneme $S \Rightarrow^+ w$, práve vtedy, keď $qsw \vdash^+ q$, teda $L_\epsilon(M) = L(G)$.

Teraz prevedieme druhú časť dôkazu. Nech $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat. Potom existuje bezkontextová gramatika G taká, že platí $L(G) = L(M)$. Dôkaz prevedieme tak, aby ľavá derivácia terminálneho reťazca w priamo korešpondovala s postupnosťou prechodov automatu M. Budeme využívať neterminálne symboly tvaru $[qZr]$, kde $q, r \in Q$ a $Z \in \Gamma$. Definujme gramatiku $G = (N, T, P, S)$ nasledovne:

- $N = \{[qZr] | q, r \in Q, Z \ \in \Gamma\} \cup S$
- $T = \Sigma$
- ak zásobníkový automat obsahuje pravidlo tvaru $qZa \to rX_1 \dots X_k, k \ \geq 1$, potom k množine prepisovacích pravidiel P pridaj všetky pravidlá tvaru:

  $$[qZs_k] \to a[rX_1s_1][s_1X_2s_2] \dots [s - k - 1X_ks_k] \text{ pre každú postupnosť } s_1, \dots, s_k$$

  stavov množiny$Q$
- ak zásobníkový automat M obsahuje pravidlo tvaru $qZa \to r$, potom P bude obsahovať pravidlo $[qZr] \to a$
- pre každý stav $q \in Q$ pridaj do množiny pravidiel P pravidlo $S \to [q_0Z_0q]$

Ako v predošlom prípade, opäť by bolo možné pomocou indukcie dokázať, že pre všetky $q, r \in Q, Z \ \in \Gamma$ platí $[qZr] \Rightarrow^m w$ práve vtedy, keď $qZw \vdash^n r$. Špeciálny prípad tejto ekvivalencie je $S \Rightarrow [q_0Z_0q] \Rightarrow^+ w$ práve vtedy, keď $q_0Z_0w \vdash^+ q, q \in F$. To však znamená, že $L_\epsilon(M) = L(G)$.

Z tvrdení, ktoré sa nám podarilo dokázať, bezprostredne plynie fakt, že trieda jazykov prijímaných zásobníkovými automatmi, je totožná s triedou jazykov typu 2. (viď [11])

Turingov stroj

Turingov stroj sa skladá z konečnej stavovej riadacej jednotky, jednosmerne neohraničenej pásky a čítacej/zapisovacej hlavy. V jednom kroku výpočtu Turingov stroj najprv 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. To všetko podľa konečnej série pravidiel - prechodovej relácie, ktorá je súčasťou riadiacej jendotky. Výpočet potom tvorí nadväzujúca séria výpočetných krokov, vychádzajúca z počiatočnej konfigurácie stroja, t.j. stavu riadiacej jednotky, obsahu pásky a pozície hlavy[11].

Formálne, Turingov stroj je šestica tvaru $M = (Q, \Sigma, \Gamma, , \delta, q_0, q_F)$, kde:

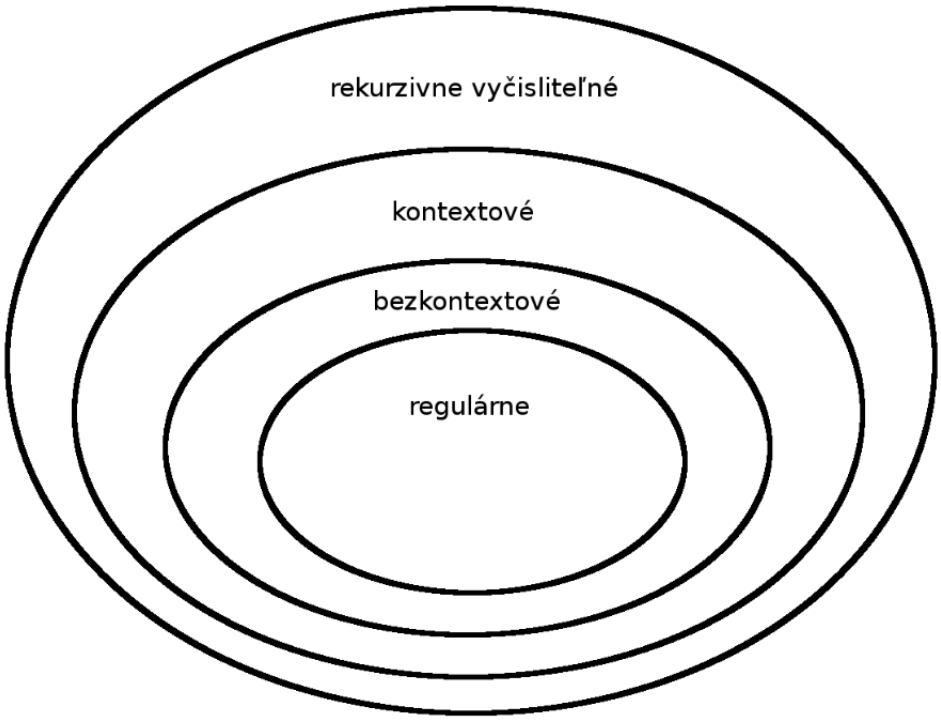
- Q je konečná množina vnútorných riadiach stavov
- Σ je konečná množina symbolov, nazývaná vstupná abeceda, $\delta \notin \Sigma$
- Γ je konečná množina symbolov, $\Sigma \subset \Gamma, \delta \in \Gamma$, nazývaná pásková abeceda
- $\delta : (Q \setminus \{q_F\}) \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$, kde$L, R \notin \Gamma$ je prechodová funkcia
- q_0 je počiatočný stav, $q_0 \in Q$
- q_F je koncový stav, $g_F \in Q$

2.5 Chomského klasifikácia jazykov

Na záver kapitoly 2, kde sme si predstavili základné definície a pojmy a následne všetky typy gramatík a automatov, prevedieme zhrnutie v podobe predstavenia Chomského klasifikácie jazykov (a gramatík), nazývaného tiež Chomského hierarchia jazykov. Chomského hierarchia jazykov vymedzuje 4 základné typy gramatík podľa tvaru prepisovacích pravidiel, ktoré sme si v úvode kapitoly predstavili, neobmedzenú gramatiku, ktorej odpovedajú jazyky typu 0, gramatiku kontextovo obmedzenú, ktorej prislúchajú jazyky typu 1, bezkontextové gramatiky pokrývajúce jazyky typu 2 a nakoniec regulárnu gramatiku (resp. pravú lineárnu gramatiku) pripadajúcu k triede jazykov typu 3. Trieda jazykov, ktoré generujú lineárne gramatiky, nemá v Chomského hierarchii jazykov samostatné postavenie, nachádzajú sa ale medzi jazykmi typu 3 a jazykmi typu 2.

Ekvivalentnými modelmi, pomocou ktorých sme schopní popísať jazyky zo všetkých tried, sú konečné automaty, silovo odpovedajúce regulárnym gramatikám a spolu pokrývajú jazyky typu 3, zásobníkové automaty pripadajú na triedu jazykov typu 2 a sú teda rovnako vyjadrovaco silné ako bezkontextové gramatiky, pre doplnenie modelmi pre popis jazykov typu 1 sú lineárne obmedzené automaty a najsilnejším nástrojom sú Turingové stroje, ktoré odpovedajú neobmedzeným gramatikám [11].

Na obrázku 2.1 sú prehľadne zobrazené jednotlivé triedy jazykov.



Obr. 2.1: Chomského hierarchia jazykov

Kapitola 3

Gramatické a automatové systémy

Gramatické systémy sú systémy založené na komunikácii všeobecne n-tice gramatík, ktoré spolu komunikujú. V klasickom poňatí gramatických systémov ide o gramatiky bezkontextové. Podľa spôsobu komunikácie medzi gramatikami potom rozdeľujeme gramatické systémy na dva základné typy gramatických systémov: sekvenčne pracujúce CD gramatické systémy a paralelne pracujúce PC gramatické systémy. V tejto kapitole budú uvedené formálne definície týchto gramatických systémov, čo je predpokladom pre ich modifikáciu v kapitole 4.

V druhej časti kapitoly sa zameriame na problematiku automatových systémov. Podobne ako pri gramatikách, aj pri automatoch je možné nad nimi vytvárať systémy so snahou zvýšenia generatívnej sily. Jedným z prístupov je rozdelenie úloh medzi automaty, kde napr. jeden z automatov môže načítavať vstupný reťazec a druhý môže regulovať činnosť prvého. Iným prístupom je rozšírenie automatu o ďalší zásobník, čím dostaneme dvojzásobníkový automat, ktorý si v tejto kapitole bližšie predstavíme. V nasledujúcej kapitole 4 potom zavedieme vlastný automatový systém, tzv. SP automatový systém.

3.1 CD gramatické systémy

CD (cooperating distributed) gramatické systémy pracujú sekvenčne. Skladajú sa z *komponentov* – gramatík. V jednom momente v takomto systéme pracuje len jedna gramatika. V závislosti od *módu* CD gramatického systému sa môže po vykonaní všeobecne k-krokov odovzdať riadenie ďalšej gramatike, ktorá pokračuje v derivovaní reťazca. Riadenie systému spočíva vo výbere komponentu, v závislosti na móde CD gramatického systému. CD gramatický systém buď vygeneruje reťazec zložený z terminálnych symbolov, alebo nastane špeciálna ukončovacia podmienka, t.j. stav, kedy nemôže gramatika, resp. všetky gramatiky (záleží od použitého módu) vykonať ďalší krok a ukončí sa činnosť celého gramatického systému (viac viď [3]).

Definícia CD gramatického systému

CD gramatický systém stupňa n, $n \geq 1$, je definovaný nasledovne:

$$\Gamma = (N, T, S, P_1, \ldots, P_n, \text{kde})$$

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.

- S je počiatočný neterminálny symbol.
- P_i je konečná množina pravidiel tvaru $A \to x$, kde $A \in N$ a $x \in (N \cup T)^*$ pre všetky $i = 1, \ldots, n$.

(viď [8])

Ukončovací derivační mód

Pokiaľ i-tá gramatika obsahuje pravidlo potrebné pre ďalší derivačný krok, prevedie tento krok, inak odovzdá riadenie nasledujúcej gramatike. Ak žiadna gramatika nemá potrebné pravidlo, nastáva špeciálny stav, kedy sa ukončí činnosť celého CD gramatického systému. Definícia:

$$x_i \Rightarrow^t y, \text{ ak}$$

- $x \Rightarrow^* y$ v $G_i = (N, T, P, S)$ a zároveň
- $y \not\Rightarrow z$ pre všetky $z \in (N \cup T)^*$

(viď [8])

k-kroký derivačný mód

Každá gramatika prevedie presne daný počet derivačných krokov k. Následne odovzdá riadenie nasledujúcej gramatike. Ak gramatika nemôže previesť až k krokov, nastáva špeciálny stav, kedy sa ukončí činnosť celého CD gramatického systému. Definícia:

$$x_i \Rightarrow^{=k} y, \text{ ak}$$

- $x \Rightarrow^k y$ v G_i

(viď [8])

Nanajvýš k-kroký derivačný mód

Každá gramatika prevedie n krokov, pričom $n \leq k$ a následne odovzdá riadenie nasledujúcej gramatike. Ak žiadna gramatika nemá potrebné pravidlo, nastáva špeciálny stav, kedy sa ukončí činnosť celého CD gramatického systému. Definícia:

$$x_i \Rightarrow^{\leq k} y, \text{ ak}$$

- $x \Rightarrow^n y$ v G_i pre nejaké $n \leq k$

(viď [8])

Prinajmenšom k-kroký derivačný mód

Každá gramatika prevedie n krokov, pričom $n \geq k$ a následne odovzdá riadenie nasledujúcej gramatike. Ak gramatika nemôže previesť až k krokov, nastáva špeciálny stav, kedy sa ukončí činnosť celého CD gramatického systému. Definícia:

$$x_i \Rightarrow^{\geq k} y, \text{ ak}$$

- $x \Rightarrow^n y$ v G_i pre nejaké $n \geq k$

(viď [8])

Množina derivačných módov

Pre potreby definície o jazyku generovanom CD gramatickým systémom označíme D množinu všetkých derivačných módov nasledovne:

$$D = \{*, t\} \cup \{\leq k, = k, \geq k : k = 1, 2, \dots\}$$

(viď [8])

Jazyk generovaný CD gramatickým systémom

Definícia jazyka generovaného CD gramatickým systémom nie je taká jednoduchá ako pri gramatikách. K definícii potrebujeme okrem množiny derivačných krokov zaviesť aj množinu možných derivácií j-tého komponentu, ktorá nám výslednú definíciu uľahčí. Definícia:

$$F(G_j, u, f) = \{v : u_j \Rightarrow^f v\}, \text{ kde } j \in \{1, \dots, n\}, f \in D, u \in (N \cup T)^*$$

Potom jazyk generovaný CD gramatickým systémom (s ohľadom na určitý mód) má nasledujúcu definíciu:

$$L_f(\Gamma) = \{w \in T^* : \text{ kde sú } v_0, v_1, \dots, v_m \text{ také, že}$$
$$v_i \in F(G_j, u_{i-1}, f),\ i = 1, \dots, m,\ j_i \in \{1, \dots, n\},$$
$$v_o = S,\ v_m = w, \text{ pre nejaké } m \geq 1\}$$

Príklad: Majme nasledujúci CD gramatický systém:

$$\Gamma = (\{S, A, A', B, B'\}, \{a, b, c\}, S, P_1, P_2), \text{ kde:}$$
$$P_1 = \{S \to S, S \to AB, A' \to A, B' \to B\}$$
$$P_2 = \{A \to aA'b, B \to cB', A \to ab, B \to c\}$$

Dostávame tri rôzne jazyky v závislosti na použitom móde.

$$L_f(\Gamma)_{f \in \{=1, \geq 1, *, t\} \cup \{\leq k : k \geq 1\}} = \{a^n b^n c^m : m, n \geq 1\}$$
$$L_{=2}(\Gamma) = L_{\geq 2}(\Gamma) = \{a^n b^n c^n : n \geq 1\}$$
$$L_{=k}(\Gamma)_{k \geq 3} = L_{\geq 3}(\Gamma) = \varnothing$$

Vidíme, že sme dostali aj jazyk $\{a^n b^n c^n : n \geq 1\}$, ktorý nie je bezkontextový. CD gramatické systémy majú teda vyššiu silu ako bezkontextové gramatiky. (viď [8])

Označenie tried jazykov generovaných CD gramatickými systémami

Definujme nasledovné označenie:

$$CD_x^y(f), \text{ kde:}$$

- f je derivačný mód.
- y môže nadobúdať hodnotu ε a potom sú povolené ε-pravidlá alebo ak y zadaný nie je, ε-pravidlá povolené nie sú.
- x môže nadobúdať hodnotu $n,\ n \geq 1$, a teda určitý konečný počet komponentov, alebo hodnotu ∞, kde počet komponentov nie je obmedzený.

(viď [8])

Hybridné CD gramatické systémy

Pri bližšom skúmaní CD gramatických systémov nám napadne, že zadaný mód je rovnaký pre všetky komponenty, čo CD gramatický systém obmedzuje. Hybridný CD gramatický systém je teda CD gramatický systém, pri ktorom má každý komponent vlastný derivačný mód. Definícia:

$$\Gamma = (N, T, S, (P_1, f_1), \ldots, (P_n, f_n)), \text{ kde:}$$

- N, T, S, P_i sú definované rovnako ako v prípade klasického CD gramatického systému.
- f_i je mód i-tého komponentu, $f_i \in D$ pre všetky $i \in \{1, \ldots, n\}$

(viď [8])

Jazyk generovaný hybridným CD gramatickým systémom

Jazyk generovaný CD gramatickým systémom je definovaný nasledovne:

$$L_f(\Gamma) = \{w \in T^* : \text{ kde sú } v_0, v_1, \ldots, v_m \text{ také, že}$$

$$v_i \in F(G_{j_i}, u_{i-1}, f_{j_i}),\ i = 1, \ldots, m,\ j_i \in \{1, \ldots, n\},$$

$$v_o = S,\ v_m = w,\ \text{pre nejaké } m \geq 1\}$$

(viď [8])

Označenie tried jazykov generovaných hybridnými CD gramatickými systémami

Definujme nasledujúce označenie:

$$XCD^y_{x,v}(f) \text{ kde:}$$

- x, y, f sú definované rovnako ako v prípade CD gramatických systémov
- v môže nadobúdať hodnotu $m,\ m \geq 1$, potom každé pravidlo P_i obsahuje nanajvýš m pravidiel, alebo v nepíšeme, resp. dosadíme zaňho hodnotu ∞, čím značíme povolený ľubovoľný počet pravidiel.
- X nadobúda buď hodnotu H, čím značíme, že ide o hybridný CD gramatický systém (potom nepíšeme (f), nakoľko hybridný CD gramatický systém nemá globálny derivačný mód pre všetky komponenty), alebo X nadobúda hodnotu D, čím značíme že ide o deterministický CD gramatický systém (pre všetky $P_i, A \to u, A \to w \in P_i$ platí $u = w$), alebo za X nedosadíme nič, čím dávame najavo, že ide o obyčajný CD gramatický systém.

(viď [8])

Generatívna sila CD gramatických systémov

Ako sme ukázali na príklade v podkapitole 3.1, CD gramatické systémy majú vyššiu generatívnu silu ako bezkontextové gramatiky. V problematike generatívnej sily CD gramatických systémov je ešte veľa otvorených otázok. Zaujímavé teorémy:

- $\mathcal{L}(CF) = CD_1^y \subset CD_2^y \subseteq CD_r^y \subseteq CD_\infty^y \subseteq \mathcal{L}M$, pre všetky $f \in \{= k, \geq k : k \geq 2\}, r \geq 3$
- $CD_r^y(\geq k) \subseteq CD_r^y(\geq k+1)$
- $\mathcal{L}(CF) = CD_1^y(t) = CD_2^y(t) \subset CD_3^y(t) = CD_\infty^y(t) = \mathcal{L}(ET0L)$
- $\mathcal{L}(CF) = HCD_1 \subset HCD_2 \subseteq HCD_3 \subseteq HCD_4 = HCD_\infty = \mathcal{L}(M, ac)$
- $\mathcal{L}(ET0L) \subset HCD_4$
- $CD_\infty(=) \subset HCD_3$

(viac viď [3][8])

3.2 PC gramatické systémy

PC (paralell communicating) gramatické systémy pracujú paralelne. Skladajú sa z *komponentov* – gramatík. V jednom momente v takomto systéme pracujú všetky gramatiky zároveň. Komponent sa teda skladá z pravidiel a štartovacieho symbolu. Okrem klasických derivačných krokov sú v PC gramatických systémoch aj prioritné, *komunikačné kroky*. Komunikačné kroky, ako z názvu vyplýva, slúžia pre komunikáciu medzi komponentami. Ide o poskytnutie vygenerovaného reťazca komponentu, ktorý vygeneruje komunikačný symbol. Pokiaľ sa takýto symbol objaví, vykoná sa komunikačný krok a pokračuje sa v derivovaní. Ak sa niektorý komponent dostane do stavu, kedy nemá potrebné pravidlo pre ďalší derivačný krok, nastane špeciálna podmienka a ukončí sa činnosť celého PC gramatického systému. Ďalšou špeciálnou situáciou je *deadlock*, keď je v komunikačných symboloch vytvorená cyklická závislosť, čo taktiež vedie k ukončeniu činnosti gramatického systému. V praxi sa PC gramatické systémy uplatňujú v niektorých simuláciách z oblasti operačných systémov (viac viď [3]).

Definícia PC gramatického systému

PC gramatický systém stupňa n, $n \geq 1$, je definovaný nasledovne:

$$\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n)), \text{ kde}$$

- N je konečná množina neterminálnych symbolov.
- K je konečná množina komunikačných symbolov, $K = \{Q_1, \ldots, Q_n\}$
- T je konečná množina terminálnych symbolov, pričom N, K, T sú párovo disjunktné.
- S_i je počiatočný neterminálny symbol i-tého komponentu pre všetky $i = 1 \ldots, n$.
- P_i je konečná množina pravidiel tvaru $A \to x$, kde $A \in N$ a $x \in (N \cup T \cup K)^*$ pre všetky $i = 1, \ldots, n$.

(viď [9])

Derivačný krok PC gramatického systému

Derivačný krok u PC gramatického systému je prevádzaný nasledovne:
Ak platí pre všetky $1 \leq i \leq n$

- buď $x_i \Rightarrow y_i$ in $G_i = (N \cup K, T, P_i, S_i)$
- alebo $x_i = y_i \in T^*$

potom:

$$(x_i, \ldots, x_n) \Rightarrow_g (y_1, \ldots, y_n)$$

(viď [9])

Komunikačný krok PC gramatického systému

Komunikačný krok u PC gramatického systému je zložitejší ako derivačný a má pred ním prioritu. Postup prevádzania je nasledovný:

1. nastaviť $z_i = x_i$ pre všetky $i = 1, \ldots, n$.
2. ak pre každé $i = 1, \ldots, n$ platí, že $alph(x_i) \cap K \neq \varnothing$ a zároveň pre každé Q_j v x_i platí, že $alph(x_j) \cap K = \varnothing$, potom pre všetky Q_j v x_i previesť nasledujúci krok.
3. nastaviť $z_j = S_j$, nahradiť Q_j s x_j v x_i a tento novovzniknutý reťazec nastaviť do z_i.
4. preveď komunikačný krok $(x_1, \ldots, x_n) \Rightarrow_c (y_1, \ldots, y_n)$ s $y_i = z_i$ pre všetky $i = 1, \ldots, n$.

(viď [9])

Jazyk generovaný PC gramatickým systémom

Predtým než definujeme jazyk generovaný PC gramatickým systémom, je potrebné zaviesť pojem *priamej derivácie*:
Ak buď

$$(x_i, \ldots, x_n) \Rightarrow_g (y_1, \ldots, y_n)$$

alebo

$$(x_i, \ldots, x_n) \Rightarrow_c (y_1, \ldots, y_n)$$

potom

$$(x_i, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$$

Definícia jazyka generovaného PC gramatickým systémom:

$$L(\Gamma) = \{x \in T^* : (S_1, S_2, \ldots, S_n) \Rightarrow^* (x, \alpha_2, \ldots, \alpha_n)$$

$$\alpha_i \in (N \cup T \cup K)^*, \text{ pre všetky } i = 2, \ldots, n\}$$

Príklad: Majme nasledujúci PC gramatický systém:

$$\Gamma = (\{S_1, S_1', S_2, S_3\}, K, \{a, b, c\}, (S_1, P_1), (S_3, P_3), (S_3, P_3)), \text{ kde:}$$

$$P_1 = \{S_1 \to abc, S_1 \to a^2b^2c^2, S_1 \to aS_1', S_1' \to a^3Q_2, S_2 \to b^2Q_3, S_3 \to c\}$$

$$P_2 = \{S_2 \to bS_2\}$$
$$P_2 = \{S_3 \to cS_3\}$$

$$(S_1, S_2, S_3) \Rightarrow (aS_1', b_S 2, cS_3) \Rightarrow^* (a^n S_1', b^n S_2, c^n S_3) \Rightarrow (a^{n+3} Q_2, b^{n+1} S_2, c^{n+1} S_3)$$
$$\Rightarrow (a^{n+3} b^{n+1} S_2, S_2, c^{n+1} S_3) \Rightarrow (a^{n+3} b^{n+3} Q_3, bS_2, c^{n+2} S_3) \Rightarrow (a^{n+3} b^{n+3} c^{n+2} S_3, bS_2, S_3)$$
$$\Rightarrow (a^{n+3} b^{n+3} c^{n+3}, b^2 S_2, cS_3)$$

Dostávame jazyk $L_r(\Gamma) = L_{nr}(\Gamma) = \{a^n b^n c^n : n \geq 1\}$, ktorý nie je bezkontextový. PC gramatické systémy majú teda vyššiu generatívnu silu ako bezkontextové gramatiky. (viď [9])

Centralizovaný PC gramatický systém

Centralizovaný PC gramatický systém je taký PC gramatický systém, kde len prvý komponent, P_1, môže obsahovať pravidlá, ktoré na svojej pravej strane majú komunikačné symboly. V takomto systéme nemôže dôjsť k deadlocku (viď 3.2). Definícia:

Nech $\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$ je PC gramatický systém. Γ je *centralizovaný* PC gramatický systém, ak $A \to x \in P_i$, pre všetky $i = 2, \ldots, n$ platí $alph(x) \cap K = \varnothing$. (viď [9])

PC gramatické systémy s návratom a bez návratu

PC gramatické systémy s návratom fungujú tak, že ak komponent poskytne svoj reťazec inému komponentu, ktorý ho požadoval vygenerovaním zodpovedajúceho komunikačného symbolu, nepokračuje komponent v generovaní reťazca ďalej, ale vráti sa k štartovaciemu symbolu a pokračuje od začiatku, teda tak, ako je to definované v postupe komunikačného kroku. Jazyk generovaný takýmto systémom označíme $L_r(\Gamma)$ (viď [9]).

PC gramatické systémy s návratom pracujú tak, že komponent, ktorý poskytol svoj reťazec inému komponentu v rámci komunikačného kroku, neprepisuje tento reťazec na štartovací symbol, ale pokračuje ďalej v jeho derivovaní. Jazyk generovaný PC gramatickým systémom bez návratu označíme $L_{nr}(\Gamma)$ (viď [9]).

Označenie tried jazykov generovaných PC gramatickými systémami

Definujme nasledujúce označenie:

$$XPC_xY$$

- X môže nadobúdať hodnotu N a potom ide o gramatický systém bez návratu, hodnotu C, vtedy ide o centralizovaný PC gramatický systém. Ak sa hodnota X neuvedie, ide o klasický PC gramatický systém.
- x označuje počet komponentov, nadobúda rovnaké hodnoty ako pri označení CD gramatických systémov.
- Y špecifikuje typ pravidiel, nadobúda hodnoty REG, ak ide o pravidlá regulárnej gramatiky, LIN, ak ide o pravidlá lineárnej gramatiky alebo CF, ak ide o pravidlá bezkontextovej gramatiky.

(viď [9])

Generatívna sila PC gramatických systémov

Ako sme ukázali na príklade, PC gramatické systémy majú vyššiu generatívnu silu ako bezkontextové gramatiky. Podobne ako u CD gramatických systémov je aj u PC gramatických systémov otázka generatívnej sily široko otvorená, napriek tomu je známych niekoľko zaujímavých teorémov:

- $PC_nREG - \mathcal{L}(LIN) \neq \emptyset$ pre $n \geq 2$
- $PC_nREG - \mathcal{L}(CF) \neq \emptyset$ pre $n \geq 3$
- $PC_nLIN - \mathcal{L}(CF) \neq \emptyset$ pre $n \geq 2$
- $PC_nREG \subset PC_{n+1}REG$ pre $n \geq 1$
- $CPC_nREG \subset CPC_nLIN \subset CPC_nCF$ pre $n \geq 1$
- $NPC_\infty CF \subseteq PC_\infty CF$
- $\mathcal{L}(M) \subset PC_\infty CF$
- $\mathcal{L}(LIN) \subset PC_\infty REG$

(viac viď [3][9])

3.3 Automatové systémy

V tejto časti si predstavíme a zadefinujeme automatový systém, ktorý modifikuje klasický zásobníkový automat a pridáva k nemu ďalší zásobník, čo, ako sa v tejto kapitole dočítame, razantne zvýši jeho generatívnu silu. Najprv si dvojzásobnikový automat formálne zadefinujeme a taktiež predstavíme spôsob jeho práce, to znamená, zadefinujeme jeho konfiguráciu a prechod. V závere uvedieme dôkaz, že jeho generatívna sila sa vyrovná sile Turingových strojov. Tento príklad modifikácie pre nás bude odrazovým mostíkom v ďalšej kapitole 4, kde zavedieme vlastný automatový systém.

Dvojzásobníkový automat

Dvojzásobníkový automat M 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 zásobníkova abeceda, $Q \cap (\Sigma \cup \Gamma) = \emptyset$
- R je konečná množina pravidiel tvaru $u_1|u_2qw \to w_1|w_2p$, kde $u_1, u_2 \in \Gamma, p, q \in Q$, $w \in \Sigma^* \cup \epsilon, w_1, w_2 \in \Gamma^*$
- $z \in Q$ je počiatočný stav
- $Z_1 \in \Gamma$ je počiatočný symbol na prvom zásobníku
- $Z_2 \in \Gamma$ je počiatočný symbol na druhom zásobníku
- $S \in \Gamma$je počiatočný symbol na zásobníku
- F $F \subseteq Q$ je množina koncových stavov

(viď [7])

Konfigurácia dvojzásobníkového automatu

Nech M je dvojzásobníkový automat $M = (Q, \Sigma, \Gamma, R, z, Z_1, Z_2, F)$. Konfigurácia dvojzásobníkového automatu M je reťazec $\chi = \$v_1\$v_2qy$, kde $v_1, v_2 \in \Gamma^*, y \in \Sigma^*, q \in Q$ a znak \$ je symbol reprezentujúci spodok ($\$ \notin Q \cup \Sigma \cup \Gamma$).
(viď [7])

Prechod a sekvencia prechodov dvojzásobníkového automatu

Nech M je dvojzásobníkový automat $M = (Q, \Sigma, \Gamma, R, z, Z_1, Z_2, F)$ a nech $y = \$h_1u_1\$h_2u_2qwz$ a $x = \$h_1v_1\$h_2v_2pz$, kde $u_1, u_2 \in \Gamma, h_1, h_2, v_1, v_2 \in \Gamma^*, p, q \in Q, w, z \in \Sigma^*$ potom M prevedie prechod z y do x, zapisujeme $y \vdash x[u_1|u_2qw \rightarrow v_1|v_2p]$ alebo zjednodušene $y \vdash x$. Štandardne, ako pri iných automatoch, môžeme $\vdash$ rozšíriť na $\vdash^n, n \geq 0$ a na $\vdash^n$ definovať $\vdash^*$ a $\vdash^+$
(viď [7])

Jazyk prijímaný dvojzásobníkovým automatom

Nech M je dvojzásobníkový automat $M = (Q, \Sigma, \Gamma, R, z, Z_1, Z_2, F)$. Jazyk prijímaný dvojzásobníkovým automatom $M, L(M)$, je definovaný:

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

(viď [7])

Generatívna sila dvojzásobníkového automatu

Ukážeme, že trieda jazykov prijímaných dvojzásobníkovými automatmi (označíme ju ako L_D) je ekvivalentná s triedou jazykov prijímaných Turingovými strojmi (L_0). Celý dôkaz si rozdelíme na dve časti. V prvej dokážeme, že ku každému Turingovmu stroju existuje ekvivalentný dvojzásobníkový automat (tj. že $L_0 \subseteq L_D$) a v druhej, že pre každý dvojzásobníkový automat existuje ekvivalentný Turingov stroj (tj. že $L_D \subseteq L_0$). Tým bude dokázané, že $L_0 = L_D$.

Idea dôkazu $L_0 \subseteq L_D$ je nasledovná. Ukážeme, že ľubovoľný TS M je možné previesť na ekvivalentný dvojzásobníkový automat D (budeme simulovať činnosť M pomocou D):

- 1. zásobník bude v priebehu výpočtu obsahovať obsah pásky TS naľavo od čítacej hlavy (na vrchole zásobníku bude znak pod čítacou hlavou a na jeho spodku ľavý okraj pásky).
- 2. zásobník bude v priebehu výpočtu obsahovať obsah pásky TS napravo od čítacej hlavy (na vrchole zásobníku bude prvý znak napravo od čítacej hlavy a na jeho spodku posledný znak pred nekonečnou postupnosťou znakov δ).
- Pred začatím samotnej simulácie ešte na dno oboch zásobníkov vložíme špeciálny znak $\$ \notin (\Sigma \cup \Gamma)$.
- Na začiatku simulácie vložíme na druhý zásobník vstupný reťazec w v opačnom poradí (tak, aby sa prvý znak reťazca nachádzal na vrchole zásobníku). Potom vrchol druhého zásobníku presunieme na vrchol prvého (čítacia hlava sa nachádza na pozícii prvého znaku vo vstupnom reťazci). Počiatočná konfigurácia D bude $(q_0, \epsilon, w_0\$, w'\$)$, kde w_0 je prvý znak w a w' je zbytok w bez prvého znaku.

- Jednotlivé operácie TS simulujeme nasledujúcim spôsobom:

  - Posun doprava $(q_1, \gamma, n) \vdash (q_2, \gamma, n+1)$ simulujeme nasledujúcim prechodom medzi konfiguráciami D: $(q_1, \epsilon, \gamma^1\$, a\ \gamma^2\$) \vdash (q_2, \epsilon, a\gamma^1\$, \gamma^2\$)$ kde $\gamma^1 a\ \gamma^2 = \gamma$. V prípade, že $a\gamma^2 = \epsilon$ (t.j. hlava sa nachádza na najpravejšom znaku pásky), druhý zásobník sa nezmení a na vrchol prvého je vložený symbol δ.
  - Zápis znaku na pozíciu pod hlavou simulujeme zmenou vrcholu prvého zásobníku za tento znak (jedná sa o operácie odstránenia vrcholu zásobníku a následného vloženia daného znaku).

Idea dôkazu $L_D \subseteq L_0$. Ukážeme, že ľubovoľný dvojzásobníkový automat D je možné previesť na ekvivalentný TS M (budeme simulovať činnosť D pomocou M).

- Použijeme trojpáskový TS, kde prvá páska bude slúžiť na načítanie vstupu a ďalšie dve pásky budú reprezentovať zásobníky (vrchol zásobníku sa nachádza na danej páske vpravo, čítacia hlava vždy ukazuje na vrchol).
- Vloženie symbolu na zásobník simulujeme posunom na príslušnej páske o jednu pozíciu doprava a zápisom príslušného znaku.
- Odstránenie symbolu z vrcholu zásobníku simulujeme posunom na príslušnej páske o jednu pozíciu doľava.

Vidíme, že spojením oboch podčastí dôkazu, teda že $L_0 \subseteq L_D$ a $L_D \subseteq L_0$, dostávame požadovaný výsledok $L_0 = L_D$. Dvojzásobníkový automat je teda skutočne rovnako silný, ako Turingov stroj, vidíme, že pridaním zásobníka sme dokázali razantne zvýšiť generatívnu silu.

Automatový systém

Aj keď dvojzásobníkový automat môžeme v istom slova zmysle chápať ako automatový systém, určite nás napadne, že automatový systém, podobne ako u gramatických systémov, bude skôr systém, ktorý bude pracovať s viacerými automatmi naraz. Automatový systém, ktorý budeme používať v tejto práci, nebude s ohľadom na výsledky, získané v oblasti lineárnych gramatických systémov, pracovať v čisto sekvenčnom režime, ale bude využívať oba, ako sekvenčný, tak aj paralelný režim, nakoľko od tejto varianty sa očakáva zvýšenie generatívnej sily, oproti prípadu, kedy žiaden systém zavedený nie je, alebo je zavedený sekvenčný systém. Tento systém bude využívať dva komponenty, automaty, medzi ktorými bude zavedený istý spôsob komunikácie. Odpovedajúcimi komponentami budú jednoobrátkové zásobníkové automaty, ktoré svojou generatívnou silou odpovedajú lineárnym gramatikám a predstavíme si ich spolu s celým systémom v nasledujúcej kapitole 4. Reťazec na vstupe bude môcť byť čítaný jedným, alebo taktiež obomi automatmi naraz, v závislosti od režimu, v ktorom bude pravidlo možné použiť. Voľba jednoobrátkového automatu taktiež limituje jednoduchosť dôkazu, kde nebude možné jednoducho popísať simuláciu Turingovho stroja, ako v prípade dvojzásobníkového automatu. Vlastnosť tzv. jednoobrátky je totiž limitujúcim faktorom, ktorá nám prácu so zásobníkom obmedzí a nebude možné ho využívať donekonečna. V momente, keď sa rozhodneme, že chceme redukovať, odstrániť symbol zo zásobníka a skrátiť tak jeho dĺžku, od tohoto momentu už nebudeme môcť ďalej zásobník rozširovať.

Kapitola 4

Modifikácia gramatických a automatových systémov

Klasické gramatické systémy definované v [3] sú založené na báze bezkontextových gramatík. V tejto kapitole bude popísaná modifikácia bázy na lineárne gramatiky. Zameriame sa na definovanie takýchto systémov a na ich generatívnu silu. Budeme skúmať, či dostaneme vyššiu generatívnu silu ako majú lineárne, či bezkontextové gramatiky.

V druhej časti sa budeme venovať automatovým systémom. Zadefinujeme tzv. SP automatový systém a jeho funkcionalitu špecifikujeme s ohľadom na poznatky získané v prvej časti tejto kapitoly. Rovnako sa budeme zaujímať jeho generatívnou silou.

4.1 Modifikácia CD gramatických systémov

Definícia CD lineárneho gramatického systému

CD lineárny gramatický systém stupňa n, $n \geq 1$, je definovaný nasledovne:

$$\Gamma = (N, T, S, P_1, \ldots, P_n, \text{ kde})$$

- N je konečná množina neterminálnych symbolov.
- T je konečná množina terminálnych symbolov, pričom $N \cap T = \varnothing$.
- S je počiatočný neterminálny symbol.
- P_i je konečná množina pravidiel tvaru $A \to x$, kde $A \in N$ a $x \in T^*(N \cup \{\varepsilon\})T^*$ pre všetky $i = 1, \ldots, n$.

Množina derivačných módov a jazyk generovaný CD lineárnym gramatickým systémom

Množina derivačných módov CD lineárneho gramatického systému je definovaná rovnako, ako v prípade klasického CD gramatického systému, taktiež jazyk generovaný CD gramatickým systémom má rovnakú definíciu ako jazyk generovaný klasickým CD gramatickým systémom, viď podkapitola 3.1.

Označenie tried jazykov generovaných CD lineárnymi gramatickými systémami

Definujme nasledujúce označenie:

$$CD_x^y(f)LIN, \text{ kde:}$$

- f je derivačný mód.
- y môže nadobúdať hodnotu ε a potom sú povolené ε-pravidlá, alebo ak y zadaný nie je, ε-pravidlá povolené nie sú.
- x môže nadobúdať hodnotu n, $n \geq 1$, a teda určitý konečný počet komponentov, alebo hodnotu ∞, kde počet komponentov nie je obmedzený.

4.1.1 Generatívna sila CD lineárnych gramatických systémov

V klasických CD gramatických systémoch sme na príklade ukázali, že ich sila bola vyššia ako je sila komponentov, z ktorých sa skladajú – bezkontextových gramatík (príklad v podkapitole 3.1).

Prvá myšlienka je, že rovnakú vlastnosť budú mať aj CD lineárne gramatické systémy, a teda že ich generatívna sila bude silnejšia ako je generatívna sila lineárnych gramatík. Pokúsime sa teda definovať takýto CD lineárny gramatický systém na príklade jazyka, ktorý nie je lineárny, ale bezkontextový.

Príklad: Definujte CD lineárny gramatický systém, ktorý generuje Dyckov jazyk. Dyckov jazyk nad $2n$-prvkovou abecedou je bezkontextový jazyk, ktorý však nie je lineárny, pomenovaný podľa matematika Walthera von Dycka, ktorý je generovaný nasledujúcou gramatikou G.

$$G = (\{S\}, \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}, P, S), \text{ kde:}$$

$$P = \{S \to \varepsilon,\ S \to a_1 S b_1 S,\ S \to a_1 S b_1 S,\ \ldots,\ S \to a_n S b_n S\}$$

V prípade, že $a_1, a_2, \ldots, a_n$ sú nejaké typy zátvoriek, zodpovedá Dyckov jazyk jazyku všetkých dobre uzátvorkovaných nad daným typom zátvoriek [4].

Zjednodušme teda abecedu na jeden typ zátvoriek a dostávame nasledujúcu bezkontextovú gramatiku.

$$G_1 = (\{S\}, \{(,)\}, P, S), \text{ kde:}$$

$$P = \{S \to \varepsilon,\ S \to (S)S\}$$

Reťazec $()(())$, ktorý gramatika G_1 generuje, by sa vygeneroval nasledovne:

$$S \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S)S \Rightarrow ()((S)S)S \Rightarrow ()(()S)S \Rightarrow ()(())S \Rightarrow ()(())$$

Pri pokuse vytvoriť CD lineárny gramatický systém však prídeme na to, že takýto CD lineárny gramatický systém neexistuje. Dôvodom je, že sme obmedzení pravidlami lineárnej gramatiky. Keďže na pravej strane pravidla môžeme mať maximálne jeden neterminál, aj keď budeme tento reťazec postupne predávať z jedného komponentu druhému sekvenčným spôsobom a pri využití derivačných krokov sa nám to nikdy nepodarí. Fakt, že v reťazci sa môže stále vyskytovať nanajvýš jeden neterminál, vedie k záveru, že generatívna sila CD lineárnych gramatických systémov je **rovnaká** ako sila lineárnych gramatík. CD lineárne gramatické systémy nemajú teda vyššiu generatívnu silu ako lineárne gramatiky, čo je

prekvapujúci záver, vzhľadom na klasické CD gramatické systémy na báze bezkontextových gramatík. Z poznatkov z tohoto príkladu môžeme odvodiť nasledujúci teorém o generatívnej sile CD gramatických systémov:

$$\mathcal{L}(LIN) = CD_1^{\varepsilon}LIN = CD_2^{\varepsilon}LIN = \ldots = CD_n^{\varepsilon}LIN$$

$$\mathcal{L}(LIN) = CD_{\infty}^{\varepsilon}LIN$$

4.2 Modifikácia PC gramatických systémov

Definícia PC lineárneho gramatického systému

PC lineárny gramatický systém stupňa n, $n \geq 1$, je definovaný nasledovne:

$$\Gamma = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n)), \text{ kde}$$

- N je konečná množina neterminálnych symbolov.
- K je konečná množina komunikačných symbolov, $K = \{Q_1, \ldots, Q_n\}$
- T je konečná množina terminálnych symbolov, pričom N, K, T sú párovo disjunktné.
- S_i je počiatočný neterminálny symbol i-tého komponentu pre všetky $i = 1 \ldots, n$.
- P_i je konečná množina pravidiel tvaru $A \to x$, kde $A \in N$ a $x \in (T \cup K)^*(N \cup \{\varepsilon\})$ $(T \cup K)^*$ pre všetky $i = 1, \ldots, n$.

Jazyk generovaný PC lineárnym gramatickým systémom

Jazyk generovaný PC gramatickým systémom má rovnakú definíciu ako jazyk generovaný klasickým PC gramatickým systémom.

Označenie tried jazykov generovaných PC lineárnymi gramatickými systémami

Pre definíciu použijeme označenie z podkapitoly 3.2, kde za hodnotu Y dosadíme LIN, čím sa obmedzíme len na používanie lineárnych pravidiel. Výsledné označenie je teda definované nasledovne:

$$XPC_xLIN$$

- X môže nadobúdať hodnotu N a potom ide o gramatický systém bez návratu, hodnotu C, vtedy ide o centralizovaný PC gramatický systém. Ak sa hodnota X neuvedie, ide o klasický PC gramatický systém.
- x môže nadobúdať hodnotu n, $n \geq 1$, a teda určitý konečný počet komponentov, alebo hodnotu ∞, kde počet komponentov nie je obmedzený.

4.2.1 Generatívna sila PC lineárnych gramatických systémov

V podkapitole 3.2 sme uviedli teorém, že klasické PC gramatické systémy na báze bezkontextových gramatík majú dokonca vyššiu generatívnu silu ako maticové gramatiky [10]. Predpoklady na generatívnu silu PC lineárnych gramatických systémov môžu byť preto vysoké, ale taktiež po závere, vyvodenom pri CD lineárnych gramatických systémoch, mierne.

Pomocou príkladu sa pokúsime ukázať silu PC lineárneho gramatického systému.

Príklad 1: Použijeme rovnaký príklad, na ktorom sa nám podarilo demonštrovať silu CD lineárneho gramatického systému, Dyckov jazyk.

Zo skúseností s CD lineárnymi gramatickými systémami vieme, že pri snahe vygenerovať v reťazci viac neterminálnych symbolov za pomoci ľubovoľného počtu komponentov, boli pravidlá lineárnej gramatiky natoľko limitujúce, že sa to nepodarilo. PC lineárne gramatické systémy však majú oproti CD lineárnym gramatickým systémom paralelný chod, kedy pracujú v jednom okamihu všetky komponenty a okrem derivačných krokov existuje aj špeciálny, komunikačný krok, pomocou ktorého sa pokúsime do reťazca dostať viacero neterminálnych symbolov. Budeme sa snažiť vytvoriť reťazec [][[]] (kvôli prehľadnosti pri použití ()(()) reťazca).

Definujme nasledujúci PC lineárny gramatický systém:

$$\Gamma = (\{S\}, K, \{[,]\}, (S, P_1), (S, P_2)), \text{ kde:}$$

$$P_1 = \{S \to \varepsilon,\ S \to [Q_2]Q_2\},\ P_2 = \{S \to S\}$$

Reťazec vygeneruje zadaný PC gramatický systém nasledovne:

$$(S,S) \Rightarrow ([Q_2]Q_2, S) \Rightarrow ([S]S, S) \Rightarrow ([][Q_2]Q_2, S) \Rightarrow ([][S]S, S) \Rightarrow ([][[Q_2]Q_2]S, S)$$

$$\Rightarrow ([][[S]S]S, S) \Rightarrow ([][[]S]S, S) \Rightarrow ([][[]]S, S) \Rightarrow ([][[]], S)$$

PC lineárny gramatický systém generujúci daný bezkontextový jazyk, sa nám podarilo definovať, aby sme obsiahli celý Dyckov jazyk, zmenili by sme definíciu PC lineárneho gramatického systému nasledovne:

$$\Gamma = (\{S\}, K, \{a_1, b_1, a_2, b_2, \ldots, a_n, b_n\}, (S, P_1), (S, P_2)), \text{ kde:}$$

$$P_1 = \{S \to \varepsilon,\ S \to a_1Q_2b_1Q_2,\ S \to a_2Q_2b_2Q_2,\ \ldots,\ S \Rightarrow a_nQ_2b_nQ_2\},\ P_2 = \{S \to S\}$$

Ukázali sme teda, že PC lineárny gramatický systém má **vyššiu** silu ako lineárne gramatiky, čo možno zapísať nasledovným teorémom:

$$\mathcal{L}(LIN) = CD^{\varepsilon}_{\infty}LIN \subset PC_nLIN$$

Príklad 2: V predchádzajúcom príklade sme ukázali, že generatívna sila PC lineárnych gramatických systémov je vyššia ako CD lineárnych gramatických systémov a lineárnych gramatík.

Pokúsime sa ísť ešte ďalej a definovať PC lineárny gramatický systém, ktorý bude generovať jazyk $a^nb^nc^n, n \geq 1$, ktorý nie je bezkontextový. Definujme nasledujúci PC lineárny gramatický systém:

$$\Gamma = (\{S_1, S_1', S_2, S_3\}, K, \{a, b, c\}, (S_1, P_1), (S_3, P_3), (S_3, P_3)), \text{ kde:}$$

$$P_1 = \{S_1 \to abc, S_1 \to a^2b^2c^2, S_1 \to aS_1', S_1' \to a^3Q_2, S_2 \to b^2Q_3, S_3 \to c\}$$

$$P_2 = \{S_2 \to bS_2\}$$
$$P_2 = \{S_3 \to cS_3\}$$

Hľadaný reťazec vygeneruje zadaný PC gramatický systém nasledovne:

$$(S_1, S_2, S_3) \Rightarrow (aS_1', b_S 2, cS_3) \Rightarrow^* (a^n S_1', b^n S_2, c^n S_3) \Rightarrow (a^{n+3} Q_2, b^{n+1} S_2, c^{n+1} S_3)$$
$$\Rightarrow (a^{n+3} b^{n+1} S_2, S_2, c^{n+1} S_3) \Rightarrow (a^{n+3} b^{n+3} Q_3, bS_2, c^{n+2} S_3) \Rightarrow (a^{n+3} b^{n+3} c^{n+2} S_3, bS_2, S_3)$$
$$\Rightarrow (a^{n+3} b^{n+3} c^{n+3}, b^2 S_2, cS_3)$$

Dostávame teda hľadaný jazyk a zapisujeme $L(\Gamma) = \{a^n b^n c^n : n \geq 1\}$. Vidíme, že sila lineárneho gramatického systému je **vyššia** ako sila bezkontextovej gramatiky. Obávané predpoklady, nadobudnuté z výsledku generatívnej sily CD lineárnych gramatických systémov, sa nám podarilo vyvrátiť a ukázať, že aj keď je sila lineárnych pravidiel oproti pravidlám bezkontextovým značne limitujúca, použitím komunikačmých krokov sa dá tento nedostatok obísť a dosiahnuť tak požadovaný výsledok. Po rozbore tohoto príkladu môžeme odvodiť nasledujúci teorém:

$$\mathcal{L}(CF) \subset PC_n LIN$$

Celkovo tak z oboch príkladov môžeme vyvodiť nasledujúci teorém, ktorý hovorí o sile PC a CD lineárnych gramatických systémov:

$$\mathcal{L}(LIN) = CD^{\varepsilon}_{\infty} LIN \subset \mathcal{L}(CF) \subset PC_n LIN$$

4.2.2 Generatívna sila PC lineárnych gramatických systémov v porovnaní s klasickými PC lineárnymi systémami

Predpokladajme pravdivosť nasledujúcich teorémov:

$$PC_\infty LIN \subseteq PC_\infty$$

$$PC_\infty - PC_\infty LIN = \varnothing$$

Ak by boli pravdivé, znamená to, že PC lineárne gramatické systémy sú rovnako silné ako klasické PC gramatické systémy na báze bezkontextových gramatík.

Ak chceme dokázať, resp. vyvrátiť pravdivosť týchto teorémov, musíme sa pozrieť na odlišnosti medzi klasickými a lineárnymi PC gramatickými systémami. Jedinou odlišnosťou sú samozrejme pravidlá. Klasické PC gramatické systémy používajú bezkontextové pravidlá, ktoré sú silnejšie ako lineárne. Ak by sa nám však podarilo ukázať algoritmus, pomocou ktorého dokážeme klasický PC gramatický systém prerobiť na PC lineárny gramatický systém, dokázali by sme pravdivosť horeuvedených teorémov.

Postup pri konverzii klasického PC gramatického systému na PC lineárny gramatický systém je nasledovný:

1. Pre každé pravidlo v každom komponente PC gramatického systému skontrolovať, či je lineárne. Ak nie je, pokračujeme nasledujúcim bodom.

2. Pre každý neterminál A v nelineárnom bezkontextovom pravidle, ktorý sa v ňom vyskytuje viac ako jedenkrát, vytvoríme v PC lineárnom gramatickom systéme nasledujúci komponent: $(A, \{A \to A\})$, ktorý označuje komunikačný symbol Q_x a všetky výskyty neterminálu A v pôvodnom pravidle nahradíme komunikačným symbolom Q_x.

Príklad: Transformujte zadaný PC gramatický systém Γ na PC lineárny gramatický systém Γ_{LIN}.

$$\Gamma = (\{S, A\}, K, \{a, b\}, (S, P_1)), \text{ kde:}$$

$$P_1 = \{S \to AA,\ A \to aAb,\ A \to ab\}$$

Jazyk generovaný týmto systémom je $L(\Gamma) = \{a^n b^n a^m b^m : m, n \geq 1\}$. Napríklad reťazec a^2b^2ab dostaneme nasledovne:

$$S \Rightarrow AA \Rightarrow aAbA \Rightarrow aabbA \Rightarrow aabbab = a^2b^2ab$$

Aplikujme algoritmus popísaný vyššie a dostaneme nasledujúci PC lineárny gramatický systém:

$$\Gamma_{LIN} = (\{S, A\}, K, \{a, b\}, (S, P_1), (A, P_2)), \text{ kde:}$$

$$P_1 = \{S \to Q_2Q_2,\ A \to aAb,\ A \to ab\},\ P_2 = \{A \to A\}$$

K rovnakému reťazcu dospejeme nasledujúco:

$$(S, A) \Rightarrow (Q_2Q_2, A) \Rightarrow (AA, A) \Rightarrow (aAbA, A) \Rightarrow (aabbA, A) \Rightarrow (aabbab, A)$$

Dostávame reťazec a^2b^2ab, teda rovnaký, ako pri použití klasického PC gramatického systému s bezkontextovými pravidlami. Na jednoduchom prípade sme teda ukázali, že je možné transformovať každý PC gramatický systém na PC lineáarny gramatický systém, ktorý generuje rovnaký jazyk.

Pre názornosť ukážka transformácie zložitejšieho PC gramatického systému na PC lineárny gramatický systém (pozn.: v uvedenom PC gramatickom systéme ide o tvar pravidiel kvôli ilustrácii algoritmu, nie o generovaný jazyk).

Príklad: Transformujte zadaný PC gramatický systém Γ na PC lineárny gramatický systém Γ_{LIN}.

$$\Gamma = (\{A, B, C\}, K, \{a, b, c\}, (A, P_1), (B, P_2)), \text{ kde:}$$

$$P_1 = \{A \to abBCbBc,\ B \to CabcQ_2C,\ B \to \varepsilon,\ C \to \varepsilon\},\ P_2 = \{B \to bB\}$$

Použitím algoritmu pre transformáciu PC gramatického systému na PC lineárny gramatický systém dostávame:

$$\Gamma = (\{A, B, C\}, K, \{a, b, c\}, (A, P_1), (B, P_2), (B, P_3), (C, P_4)), \text{ kde:}$$

$$P_1 = \{A \to abQ_3CbQ_3c,\ B \to Q_4abcQ_2Q_4,\ B \to \varepsilon,\ C \to \varepsilon\},\ P_2 = \{B \to bB\},$$

$$P_3 = \{B \to B\},\ P_4 = \{C \to C\}$$

Na príkladoch sme dokázali pravdivosť vyššie uvedených teorémov, a teda generatívna sila PC gramatických systémov a PC lineárnych gramatických systémov s neobmedzeným počtom komponentov je rovnaká, rozdiel jazykov, ktoré generuje klasický PC gramatický systém a PC lineárny gramatický systém, je prázdna množina.

4.3 Modifikácia automatových systémov

V tejto podkapitole zavedieme automatový systém, ktorý nebude založený na dvojzásobníkovom automate, ale budeme v ňom využívať tzv. jednoobrátkové jednozásobníkové automaty. Voľba jednoobrátkového zásobníkového automatu nie je náhodná, ale pramení z toho, že jeho generatívna sila odpovedá lineárnym gramatikám. V tomto kontexte nás teda zaujímajú odpovede na otázky, či bude možné dosiahnuť zavedením systému rovnaké výsledky, ako v prípade PC lineárnych gramatických systémov, prípadne, či sa nám jeho generatívnu silu nepodarí navýšiť a dospejeme tak k záverom podobným, ako v prípade CD lineárnych gramatických systémov. Takto zavedený automatový systém nazveme PS (parallel-sequence) jednoobrátkový automatový systém.

Neformálne, jednoobrátkový zásobníkový automat pracuje ako klasický, no svoj zásobník môže zväčšovať len do momentu, pokiaľ sa ho nerozhodne skrátiť. Vtedy vykonáva tzv. obrátku a nemôže už ďalej svoj zásobník znova zväčšovať. Toto pravidlo je veľmi limitujúce a ako sme už načrtli v predchádzajúcej podkapitole 3.3, nebude možné vytvoriť jednoduchý dôkaz, kde by sme ukázali, že dokážeme odsimulovať prácu Turingovho stroja.

4.4 Jednoobrátkový automat

Jednoobrátkový automat M je sedmica:

$$M = (Q, \Sigma, \Gamma, R, s, S, F), \text{ kde}$$

- Q je konečná množina stavov
- Σ je vstupná abeceda
- Γ je zásobníkova abeceda
- R je konečná množina pravidiel tvaru $Apa \to wq$, kde $A \in \Gamma, p, q \in Q, a \in \Sigma \cup \{\epsilon\}$, $w \in \Gamma^*$
- $s \in Q$ je počiatočný stav
- $S \in \Gamma$je počiatočný symbol na zásobníku
- $F \subseteq Q$ je množina koncových stavov

V jednoobrátkovom zásobníkovom automate však musí platiť jedno obmedzujúce pravidlo, inak by sa nijako nelíšil od zásobníkového automatu. Zásobník môžeme zväčšovať, predĺžovať jeho dĺžku, až do momentu prvej redukcie, teda prvého odstránenia symbolu z vrcholu zásobníka, čo vedie k jeho skráteniu. Po tomto úkone už nie je možné ďalej zásobník rozširovať, vykonali sme tzv. obrátku.

4.5 Definícia PS automatového systému

PS jednoobrátkový automatový systém je definovaný nasledovne:

$$\Gamma = (\Sigma, \Gamma, (Q_1, R_1, s_1, S_1, F_1), (Q_2, R_2, s_2, S_2, F_2)), \text{ kde}$$

- Σ je vstupná abeceda, ktorá je spoločná pre oba automaty

- Γ je zásobníkova abeceda, ktorá je spoločná pre oba automaty
- $(Q_1, R_1, s_1, S_1, F_1)$ je prvý komponent PS jednoobrátkového automatového systému, ktorého jednotlivé časti definujú prvý z dvojice jednoobrátkových automatov $M_1 = (Q_1, \Sigma, \Gamma, R_1, s_1, S_1, F_1)$, kde:
  - Q_1 je konečná množina stavov
  - R_1 je konečná množina pravidiel tvaru $X : Apa \rightarrow wq$, kde $A \in \Gamma, p, q \in Q_1, a \in \Sigma \cup \{\epsilon\}, w \in \Gamma^*, X \in \{P, S\}$, kde P označuje pravidlo v paralelnom režime a S označuje pravidlo v sekvenčnom režime
- $(Q_2, R_2, s_2, S_2, F_2)$ je druhý komponent PS jednoobrátkového automatového systému, ktorého jednotlivé časti definujú druhý z dvojice jednoobrátkových automatov $M_2 = (Q_2, \Sigma, \Gamma, R_2, s_2, S_2, F_2)$, kde jednotlivé množiny sú definované analogicky k prvému komponentu.

PS jednoobrátkový automatový systém pracuje tak, že sa pozrie na symbol, ktorý sa nachádza na jeho vstupe. Následne sa skontroluje, či pre daný vstupný symbol a súčasný vnútorný stav jedného z automatov a stav ich zásobníkov existuje pravidlo, ktoré by bolo možné využiť. Pre využitie sekvenčného pravidla stačí, aby sa táto kombinácia vyskytovala na jeho ľavej strane, v prípade pravidla paralelného, musí byť schopný previesť výpočetný krok ako prvý, tak aj druhý automat. V prípade, že je možné previesť aj pravidlo sekvenčné a zároveň aj dvojicu pravidiel paralelných, prioritu má sekvenčné pravidlo. Ak by nastal konflikt medzi sekvenčnými pravidlami naprieč automatmi, prioritu má prvý automat. Automaty môžu byť, samozrejme, ako nedeterministické, tak aj deterministické, taktiež je možné využiť ich rozšírený variant. Automat môže vstupný reťazec prijať prechodom do koncového stavu, vyprázdnením zásobníka alebo kombináciou oboch prístupov.

Automatový systém by nemusel využívať nutne len dva komponenty - dva jednoobrátkové automaty. Pre úplnosť môžeme zaviesť všeobecný PS jednoobrátkový automatový systém, ktorý môže mať ľubovoľný počet komponentov nasledovne:

$$\Gamma = (\Sigma, \Gamma, (Q_1, R_1, s_1, S_1, F_1), (Q_2, R_2, s_2, S_2, F_2), \ldots, (Q_n, R_n, s_n, S_n, F_n))$$

Číslo n by odpovedalo počtu komponentov. V takomto variante by bolo možné taktiež pozmeniť pravidlo, kedy je možné previesť paralelné pravidlá, v závislosti na zvolenom móde by nemuseli previesť paralelné pravidlá v jednom kroku všetky komponenty, ale bolo by možné upraviť tvar pravidiel nasledovne:

$$X\{x_1, x_2, \ldots, x_m\} : Apa \rightarrow wq, \text{ kde}$$

$A \in \Gamma, p, q \in Q_1, a \in \Sigma \cup \{\epsilon\}, w \in \Gamma^*, X \in \{P, S\}$, kde P označuje pravidlo v paralelnom režime a S označuje pravidlo v sekvenčnom režime, $\{x_1, x_2, \ldots, x_m\}, m < n$, označuje komponenty, ktoré musia dané paralelné pravidlo vykonať spoločne. V prípade sekvenčných pravidiel sa táto množina neuvádza.

V tomto texte budeme však naďalej pracovať s PS jednoobrátkovým automatovým systémom, ktorý bude využívať len dva komponenty, v zmysle pôvodnej základnej definície, a preto, ak sa v ďalšej časti textu budeme odkazovať na PS jednoobrátkový automatový systém, budeme myslieť práve takýto systém s dvomi komponentami, iné prípady uvedieme explicitne.

4.6 Generatívna sila PS automatového systému

V tejto podkapitole sa pokúsime demonštrovať silu PS jednoobrátkového automatového systému a pokúsime sa dokázať, že zavedením systému sa dostaneme nad generatívnu silu jednoobrátkového automatu. Poučení z prípadu CD lineárnych gramatických systémov, sme zostavili PS jednoobrátkový automatový systém tak, aby dokázal pracovať paralelne, paralelizmus, totiž v prípade PC lineárnych gramatických systémov dokázal pozitívne ovplyvniť generatívnu silu.

Využijeme rovnaký príklad ako v prípade lineárnych gramatických systémov, pokúsime sa teda zadefinovať PS jednoobrátkový automatový systém, ktorý bude prijímať jazyk $\{a^n b^n c^n, n \geq 1\}$, ktorý nie je bezkontextový. V prípade, že takýto systém nájdeme a zadefinujeme, dokážeme, že sa nedostávame len nad generatívnu silu jednoobrátkových automatov, ktoré sú ekvivalentné lineárnym gramatikám, ale dokonca nad silu bezkontextových automatov, resp. zásobníkových automatov, ktoré sú im ekvivalentné.

Prvým kľúčovým prvkom je uvdeomiť si, že musíme postupovať z opačného konca, ako pri lineárnych gramatických systémoch, kde sme jazyk pomocou jednotlivých gramatík generovali. V prípade automatov všeobecne jazyk negenerujeme, ale prijímame, čo vedie k inému postupu. Pri PC lineárnom gramatickom systéme sme si v jednotlivých komponentoch generovali v každom kroku jednotlivé symboly, pričom prvý komponent riadil celý proces a po požadovanom počte vygenerovaných symboloch a pomocou komunikačného symbolu požiadal druhý, resp. tretí komponent o zaslanie odpovedajúcich symbolov b a c, z ktorých následne vytvoril požadovaný reťazec.

V prípade PS jednoobrátkového automatového systému musíme zvoliť iný myšlienkový postup a pokúsiť sa rozdeliť jednotlivé časti vstupného reťazca do správy jednému, či druhému jednoobrátkovému automatu a využiť paralelizmus. To docielime tak, že prvý komponent bude načítavať symboly a zo vstupu a ukladať si ich na zásobník, až kým nenarazí na prvý symbol b. V tom momente prevezme kontrolu pomocou svojich sekvenčných pravidiel druhý komponent, ktorý aplikuje rovnaký postup na symboly b. Aby sme dodržali rovnaký počet všetkých symbolov, symboly c budú komponenty spracovávať paralelne, pričom po každom načítaní symbolu c, si jeden i druhý komponent vymaže zo svojho zásobníka prislúchajúci symbol a resp. b. V prípade, že bol skutočne počet symbolov a, b, c rovnaký, zostali na oboch zásobníkoch len ich počiatočné symboly, ktoré zmažeme a jazyk prijmeme na základe vyprázdnenia oboch zásobníkov. V opačnom prípade, by neboli vyprázdnené oba zásobníky, alebo by sa na vstupe nachádzali ďalšie symboly c, pričom aspoň v jednom zásobníku by sa nachádzal už len počiatočný symbol zásobníka, nebolo by tak možné previesť paralelné pravidlo a PS jednoobrátkový automatový systém by vstupný reťazec odmietol.

Definujme nasledujúci PS jednoobrátkový automatový systém, ktorý bude prijímať reťazec, pokiaľ po jeho načítaní zostanú oba zásobníky prázdne:

$$\Gamma = (\{a, b, c\}, \{a, b, c\}, (\{q_1\}, R_1, q_1, S_1, \emptyset), (\{q_2\}, R_2, q_2, S_2, \emptyset)), \text{ kde}$$

$$R1 = \{S : S_1 q_1 a \to S_1 a q_1, S : a q_1 a \to a a q_1, P : a q_1 c \to q_1, P : S_1 q_1 \to q_1\}$$

$$R2 = \{S : S_2 q_2 b \to S_2 b q_2, S : b q_2 b \to b b q_2, P : b q_2 c \to q_2, P : S_2 q_2 \to q_2\}$$

Prechody definovaného PS jednoobrátkového automatového systému budú nasledovné:

$$(S_1 q_1, S_2 q_2, a^n b^n c^n) \vdash (S_1 a q_1, S_2 q_2, a^{n-1} b^n c^n) \vdash \ldots \vdash (S_1 a^n q_1, S_2 q_2, b^n c^n) \vdash$$

$$\vdash (S_1 a^n q_1, S_2 b q_2, b^{n-1} c^n) \vdash \ldots \vdash (S_1 a^n q_1, S_2 b^n q_2, c^n) \vdash (S_1 a^{n-1} q_1, S_2 b^{n-1} q_2, c^{n-1}) \vdash \ldots \vdash$$

$$\vdash (S_1 q_1, S_2 q_2, \epsilon) \vdash (q_1, q_2, \epsilon)$$

Vidíme, že definovaný automat skutočne generuje jazyk $a^n b^n c^n$, nedeterminizmus môžeme odstrániť jemnou úpravou a prevodom na rozšírený zásobníkový automat nasledovnou úpravou množín R_1 a R_2:

$$R1 = \{S : S_1 q_1 a \to S_1 S_1 a q_1, S : a q_1 a \to a a q_1, P : a q_1 c \to q_1, P : S_1 S_1 q_1 \to q_1\}$$

$$R2 = \{S : S_2 q_2 b \to S_2 S_2 b q_2, S : b q_2 b \to b b q_2, P : b q_2 c \to q_2, P : S_2 S_2 q_2 \to q_2\}$$

Vidíme, že sila PS jednoobrátkového automatového systému je **vyššia** ako sila jednoobrátkového automatu, resp. lineárnych gramatík. Zavedením paralelizmu a ďalšieho komponentu sme dokázali zvýšiť generatívnu silu. Tento poznatok má veľký význam, nakoľko dokážeme syntakticky zložité štruktúry, ktoré nie sú bezkontextové, popísať za pomoci automatového systému, ktorý, ako sme ukázali, má dokonca deterministický variant, čo je v problematike prekladačov žiadané.

Kapitola 5

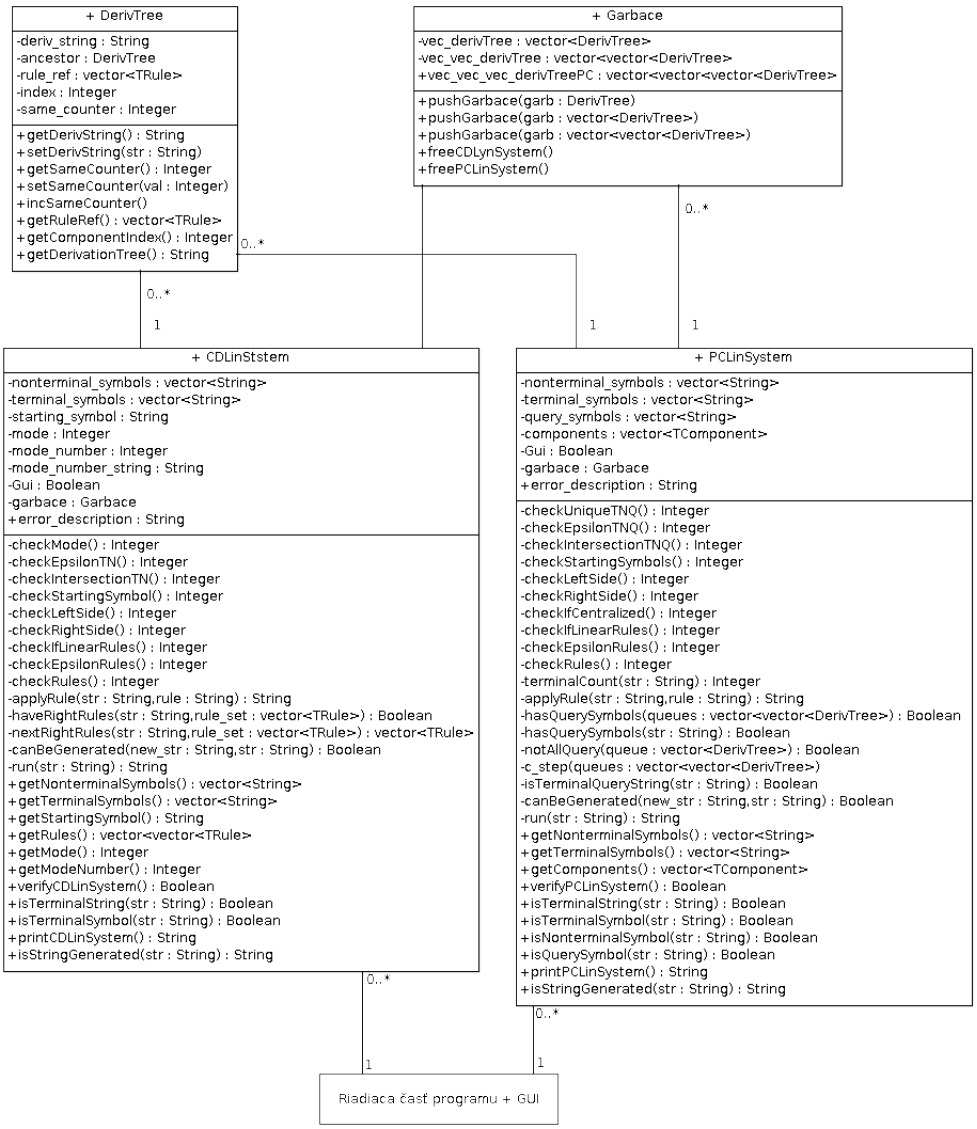
Implementácia lineárnych gramatických systémov

V tejto kapitole transformujeme teoreticky navrhnutý koncept CD a PC lineárnych gramatických systémov na reálnu programovú implemetáciu. Popíšeme algoritmy realizujúce určité činnosti CD lineárnych gramatických systémov a PC lineárnych gramatických systémov a bližšie sa pozrieme na niektoré zložitejšie problémy, ktoré sa pri implementácii objavili. Na záver budú niektoré príklady z teoretickej časti (podkapitola 4.1.1 a 4.2.1) demonštrované v programe Linear Grammar System Creator, ktorý je výsledkom implementačnej časti.

V rámci implementačnej časti boli vytvorené triedy v jazyku C++, ktoré reprezentujú CD a PC lineárny gramatický systém. Tieto triedy je možné použiť pri výstavbe syntaktického analyzátora, pomocou objektov reprezentujúcich CD a PC gramatické systémy. Je možné si nadefinovať vlastné CD alebo PC lineárne gramatické systémy, ktoré následne dokážu vyhodnotiť, či zadaný reťazec nimi je, alebo nie je generovaný. Nad týmito triedami bolo vytvorené užívateľské rozhranie Linear Grammar System Creator pre lepšiu ilustráciu práce implementovaných CD a PC lineárnych gramatických systémov, kde je možné prehliadnuť si, akými derivačnými krokmi daný reťazec vznikal.

5.1 Štruktúra aplikácie

Základné jadro programu ukazuje obrázok 5.1. Tento diagram tried neobsahuje väzbu na grafické užívateľské rozhranie, zobrazuje len objektový návrh jadra programu. Významné funkcie jednotlivých tried budú ďalej popísané v podkapitolách 5.3 a 5.4. Triedy `cdLinSystem` a `pcLinsystem` zodpovedajú implementácii CD a PC lineárneho gramatického systému, trieda `derivTree` reprezentuje reťazce tvorené pri generovaní hľadaného reťazca lineárnym gramatickým systémom a trieda `garbace` je implementácia jednoduchého kvázi garbace collectoru, kde sú ukladané referencie na naalokovanú pamäť, ktorá je podľa potreby uvoľnená.



Obr. 5.1: Diagram tried aplikácie Linear Grammar System Creator

5.2 Reprezentácia dát

5.2.1 Reprezentácia CD lineárneho gramatického systému

Z definície v podkapitole 3.1 vidíme, že CD lineárny gramatický systém sa skladá z neterminálov, terminálov, štartovacieho symbolu, módu a komponentov – množiny množín pravidiel. Neterminály a terminály sú reprezentované pomocou vektoru reťazcov, štartovací symbol pomocou reťazca. V programovej implementácii je možný výber z dvoch módov, k-krokého a t-módu (ukončovacieho módu), ktoré sú reprezentované zodpovedajúcim celým číslom (makrami) a pri k-krokom móde je potrebné ďalšie celé číslo reprezentujúce k. Komponenty obsahujúce pravidlá, sú reprezentované vektorom vektorov reťazcov, kde tvar pravidla $A \rightarrow x$ odpovedá reťazcu Ä x". V konečnej implementácii ide o vektor vektorov štruktúry `TRule`, ktorá okrem reťazca reprezentujúceho celé pravidlo, obsahuje aj zvlášť jeho pravú a ľavú stranu, kvôli lepšej a prehľadnejšej práci s pravidlami.

Reprezentácia PC lineárneho gramatického systému

Podľa definície v podkapitole 3.2 je zrejmé, že pri PC lineárnom gramatickom systému budeme reprezentovať neterminály a terminály rovnako. Reprezentácia komponentov bude iná. Pravidlá budú reprezentované rovnako ako v CD lineárnom gramatickom systéme, taktiež aj jednotlivé štartovacie symboly. Každý komponent má navyše unikátny komunikačný symbol, ktorý ho jednoznačne identifikuje (nie nutne $Q_1, \ldots, Q_n$). Ak spojíme všetky tieto zložky, ktoré tvoria komponent, dostaneme výslednú štruktúru `TComponent` zloženú zo štartovacieho symbolu, komunikačného symbolu a vektoru vektorov pravidiel (štruktúra `TRule`).

5.3 Implementácia metód CD lineárneho gramatického systému

V tejto podkapitole budú predstavené významné a implementačne zaujímavé metódy, ktoré reprezentujú činnosť CD lineárneho gramatického systému.

5.3.1 Kontrola zadania validného CD lineárneho gramatického systému

Predtým, než sa systém začne používať k zisťovaniu, či zadaný reťazec je alebo nie je systémom generovaný, je potrebné systém skontrolovať. K tomuto slúži metóda `verifyCDLinSystem`, ktorá postupne volá privátne metódy, ktoré riešia podproblémy validácie systému. V prvom rade ide o kontrolu validne zadaného módu a kontrolu, či prienik terminálnych a neterminálnych symbolov nie je náhodou neprázdna množina, taktiež, či sa nejedná o multimnožiny. Taktiež sa kontrolujú terminály a neterminály na prítomnosť symbolu 'e', ktorý reprezentuje symbol ε. Kontrolou musí prejsť aj štartovací symbol, a to konkrétne kontrolami dvomi, či sa nachádza v množine neterminálnych symbolov, a či existuje v prvom komponente pravidlo s ľavou stranou rovnajúcou sa štartovaciemu symbolu, čo nás privádza k ďalšej konrole, kontrole pravidiel.

Najprv sa skontroluje ľavá strana pravidiel, teda či ide skutočne o neterminálne symboly, následne pravá strana, kde sa kontroluje, či sa tu vyskytujú skutočne len neterminálne a terminálne symboly, alebo v špeciálnom prípade len symbol 'e' reprezentujúci ε. Poslednou kontrolou je skontrolovať, či sa v každom pravidle vyskytuje najviac jeden neterminálny symbol, podmienka lineárnosti pravidiel.

5.3.2 Kontrola aplikácie pravidiel na reťazec

Pri generovaní hľadaného symbolu začíname štartovacím symbolom, na ktorý aplikujeme pravidlá. Vzniknú nám tak reťazce, ktoré, ak sú zložené len z terminálnych symbolov, porovnáme s hľadaným na zhodu a reťazce obsahujúce neterminálny symbol, na ktoré opäť aplikujeme pravidlá. Ak by sme na daný reťazec aplikovali všetky pravidlá, ktoré sú teoreticky aplikovateľné, počet reťazcov by stúpal exponenciálne a pri hľadaní dlhších reťazcov by nám pravdepodobne došla pamäť.

Aby sme naivne neaplikovali všetky pravidlá, existuje metóda `canBeGenerated`, ktorá zistí, či z daného reťazca je možné získať hľadaný reťazec. Algoritmus je nasledovný:

1. ak je reťazec zložený len z terminálnych symbolov, porovnáme ho s hľadaným, ak sú rovnaké, vrátime **true**, inak **false**.

2. ak počet neterminálnych symbolov v reťazci je väčší ako dĺžka hľadaného reťazca, vrátime **false**.

3. ak sa reťazec skladá len z jedného neterminálu, vrátime **true**.

4. ak sa neterminál vyskytuje v reťazci na prvom mieste, porovnáme vyznačené podreťazce (ilustrujeme príklad, kde hľadaným reťazcom je *baabbab* a náš aktuálny reťazec je *Aaabb*): *A***aabb** *b***aabb***ab*. Pri označení hľadaného reťazca ako `str` a aktuálneho ako `newStr`, dostávame nasledujúci výraz:

   ```
   newStr.substr(1,newStr.size()-1).compare(str.substr(str.size()
               -(newStr.size()-1), newStr.size()-1))
   ```

   Ak sú vyznačené časti rovnaké, vraciame **true**, inak vraciame **false**.

5. ak sa neterminál vyskytuje v reťazci na poslednom mieste, porovnáme vyznačené podreťazce (ilustrujeme príklad, kde hľadaným reťazcom je *aabbab* a náš aktuálny reťazec je *aabbA*): **aabb***A* **aabb***ab*. Pri označení hľadaného reťazca ako `str` a aktuálneho ako `newStr`, dostávame výraz:

   ```
   newStr.substr(0,newStr.size()-1).compare(str.substr(0,newStr.size()-1))
   ```

   Ak sú vyznačené časti rovnaké, vraciame **true**, inak vraciame **false**.

6. ak sa neterminál nevyskytuje v reťazci na prvom ani poslednom mieste, porovnáme vyznačené podreťazce (ilustrujeme príklad, kde hľadaným reťazcom je *aaabbb* a náš aktuálny reťazec je *aaAbb*): **aa***Abb* *b***aa***abbbab* a *aaA***bb** *aaab***bb**. Ak opäť označíme hľadaný reťazec ako `str` a aktuálny `newStr`, dostávame výraz:

   ```
          newStr.substr(0,index).compare(str.substr(0,index)) &&
        newStr.substr(index+1,newStr.size()-index-1).compare(str
   .substr(str.size()-(newStr.size()-index-1),newStr.size()-index-1))
   ```

   Ak sú vyznačené časti v oboch dvojiciach rovnaké, vraciame **true**, inak vraciame **false**.

Prípadom, ktoré sme ošetrili, sa vymyká ešte jeden, a to, ak by pravidlá, ktoré majú na pravej strane len jeden neterminál a žiadne terminály, mali cyklické závislosti, napr. množina pravidiel $P = \{A \to B,\ B \to C,\ C \to A\}$. Tieto situácie rieši počítadlo, ktoré sa zvýši vtedy, keď reťazec má po prevedení derivačného kroku rovnakú veľkosť ako pred ním. Po prekročení stanovenej hranice je takýto reťazec vyradený z derivačného procesu.

5.3.3 Implementácia hlavného cyklu CD lineárneho gramatického systému

Beh CD lineárneho gramatického systému zabezpečuje metóda `isStringGenerated(string str)`, kde parameter `str` je hľadaný reťazec. Metóda sa stará o správu pamäte a volá privátnu metódu `run(string str)`, v ktorej je implementovaný samotný hlavný cyklus CD lineárneho gramatického systému.

Pre ukladanie generovaných reťazcov sú vytvorené dve fronty, jedna obsahujúca reťazce pred aplikovaním pravidiel a druhá, do ktorej sa uložia reťazce po aplikovaní pravidiel. Popis algoritmu realizujúceho samotný beh je nasledovný:

1. skontroluj, či je hľadaný reťazec zložený len z terminálov, ak nie, vráť prázdny reťazec.
2. vytvor prvý reťazec zodpovedajúci štartovaciemu symbolu a vlož ho do prvej fronty, ak je mód CD lineárneho gramatického systému k-kroký, vynuluj počítadlo.
3. ak je prvá fronta prázdna, vráť prázdny reťazec.
4. pre každý prvok z prvej fronty rozhodni, či je zložený len z terminálnych symbolov, ak áno, porovnaj ho s hľadaným reťazcom a v prípade zhody vráť reťazec popisujúci celý derivačný proces, inak pokračuj ďalším bodom
5. ak je mód CD lineárneho gramatického systému ukončovací, aplikuj na každý prvok z prvej fronty aktuálnu sadu pravidiel, pokiaľ je to možné, inak nájdi nasledujúcu možnú sadu a aplikuj tú. Nové reťazce po aplikácii pravidiel ulož do druhej fronty.
6. ak je mód CD lineárneho gramatického systému k-kroký, zvýš hodnotu počítadla a porovnaj ju s hodnotou k. Ak sú si rovné, počítadlo nastav na hodnotu 1 a aplikuj na každý prvok z prvej fronty nasledujúcu sadu pravidiel, inak aplikuj aktuálnu sadu. Nové reťazce po aplikácii pravidiel ulož do druhej fronty.
7. Vymaž obsah prvej fronty a prekopíruj do nej obsah druhej fronty, následne obsah druhej fronty vymaž. Pokračuj bodom 3.

Po ukončení behu privátnej metódy `run(string str)` je uvoľnená dynamicky naalokovaná pamäť a vrátený výsledok vyprodukovaný touto metódou.

5.4 Implementácia metód PC lineárneho gramatického systému

V tejto podkapitole ukážeme metódy, ktoré reprezentujú činnosť PC lineárneho gramatického systému. Pôjde hlavne o algoritmus generovania hľadaného reťazca a heuristiky urýchľujúce generovanie takéhoto reťazca.

5.4.1 Kontrola zadania validného PC lineárneho gramatického systému

Kontrola validity PC lineárneho gramatického systému je v mnohých veciach podobná až rovnaká ako pri kontrole CD lineárneho gramatického systému. Odlišnosť pri kontrole neterminálnych a terminálnych symbolov je tá, že je potrebné vziať do úvahy aj komunikačné symboly a skontrolovať, či množiny neterminálnych, terminálnych a komunikačných symbolov sú párovo disjunktné, taktiež či nejde o multimnožiny. Keďže program podporuje len centralizované PC lineárne gramatické systémy, je potrebná aj kontrola na výskyt komunikačných symbolov v komponentoch. Existencia pravidla pre štartovací symbol sa testuje pri každom komponente na rozdiel od CD lineárnych gramatických systémov, kde kontrola prebieha len v prvom komponente. Všetky tieto kontroly realizuje metóda `verifyPCLinSystem`, ktorá volá privátne metódy implementujúce riešenia jednotlivých problémov.

5.4.2 Kontrola aplikácie pravidiel na reťazec

Oproti kontrole v CD lineárnych gramatických systémoch je pri PC lineárnych gramatických systémoch situácia o niečo zložitejšia. V prvom rade je potrebné vziať do úvahy komunikačné symboly. Ak sa v reťazci vyskytujú, nebudeme takýto reťazec kontrolovať hneď, ale počkáme, kým sa za komunikačný symbol nahradí zodpovedajúci reťazec.

Existencia komunikačných symbolov prináša situácie, kedy je možné dostať v reťazci viacero neterminálnych symbolov, musíme teda zvoliť iný prístup, ako keď sme očakávali maximálne jeden. Reťazec si rozdelíme na časti, tak, že prvú časť vezmeme od začiatku reťazca po prvý neterminálny symbol, ďalšie časti od jedného neterminálneho symbolu po ďalší neterminálny symbol a nakoniec od posledného neterminánehо symbolu po koniec reťazca (neterminálne symboly do jednotlivých častí nezaratávame). V hľadanom reťazci postupne vymazávame jednotlivé časti, ktoré sme získali, prvú a poslednú je možné skontrolovať a vymazať pochopiteľne len na začiatku a konci hľadaného reťazca. Ak sa tam nejaká časť nenachádza, znamená to, že z daného reťazca nie je možné získať hľadaný reťazec, a teda ho z procesu aplikácie pravidiel vylučujeme. Napr. ak by hľadaným reťazcom bol reťazec *aabbababaaabaa* a my sa rozhodujeme, či ho môžeme získať z reťazca *aaAababBaabAa*, kde množina terminálnych symbolov je $\{a, b\}$ a množina neterminálnych symbolov je $\{A, B\}$, rozdelíme najprv reťazec na časti *aa*, *abab*, *aab*, *a*. Pokúsime sa porovnať začiatok a koniec hľadaného reťazca s prvou a poslednou časťou, **aa***bbababaaaba***a** a po vymazaní nám ostáva *bbababaaaba*. Teraz hľadáme jednotlivé časti, najprv teda nájdeme a vymažeme *abab*, ostane nám *bbaaaba*, vezmeme nasledujúcu a v tomto príklade poslednú časť *aab*, ktorú opäť nájdeme a po vymazaní nám ostáva *bbaa*. V danom príklade sa nám podarilo úspešne nájsť všetky časti. V prípade, že by sa to nepodarilo, nie je z daného reťazca možné vygenerovať hľadaný reťazec.

Pokiaľ máme reťazec zložený len z jedného neterminálneho symbolu, postupujeme rovnako ako v prípade CD lineárnych gramatických systémov, aby sme ošetrili možné cyklické závislosti.

5.4.3 Implementácia hlavného cyklu PC lineárneho gramatického systému

Beh PC lineárneho gramatického systému zabezpečuje rovnomenná metóda ako v prípade CD gramatického systému, `isStringGenerated(string str)`, ktorá sa taktiež stará

o správu pamäte a volá privátnu funkciu `run(string str)`, ktorá implementuje hlavný cyklus PC gramatického systému.

Pre ukladanie generovaných reťazcov nestačí vytvoriť dve fronty, ako tomu bolo v prípade CD lineárnych gramatických systémov, ale vytvoriť dve fronty pre každú komponentu PC lineárneho gramatického systému. V prvej budú opäť ukladané reťazce pred aplikovaním pravidiel a v druhej reťazce po aplikovaní pravidiel. Výsledný algoritmus potom vyzerá takto:

1. skontroluj, či je hľadaný reťazec zložený len z terminálov, ak nie, vráť prázdny reťazec.
2. vytvor prvé reťazce zodpovedajúce štartovacím symbolom jednotlivých komponentov a vlož ich do prvých front zodpovedajúcich komponentov.
3. ak je niektorá prvá fronta prázdna, vráť prázdny reťazec.
4. preveď komunikačný krok – nahraď komunikačné symboly za zodpovedajúce reťazce
5. pre každý prvok z prvej fronty z prvého komponentu rozhodni, či je zložený len z terminálnych symbolov, ak áno, porovnaj ho s hľadaným reťazcom a v prípade zhody vráť reťazec popisujúci celý derivačný proces, inak pokračuj ďalším bodom
6. na každý prvok z prvých front komponentov aplikuj pravidlá. Nové reťazce po aplikácii pravidiel ulož do druhých front zodpovedajúcich komponentov.
7. vymaž obsahy prvých front a prekopíruj do nich obsahy druhých front, následne obsahy druhých front vymaž. Pokračuj bodom 3.

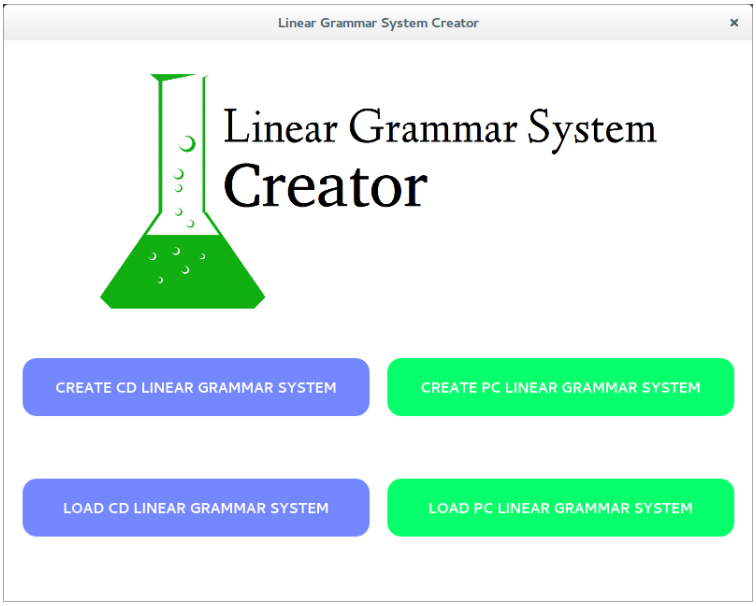
5.5 Linear Grammar System Creator

Linear Grammar System Creator je program využívajúci triedy vytvorené pre implementáciu CD a PC lineárnych gramatických systémov. Ide o nadstavbu s grafickým užívateľským rozhraním. Grafické užívateľské rozhranie bolo vytvorené pomocou frameworku Qt (viac viď [1]). Linear Grammar System Creator slúži pre demonštráciu teoretických poznatkov v praxi.

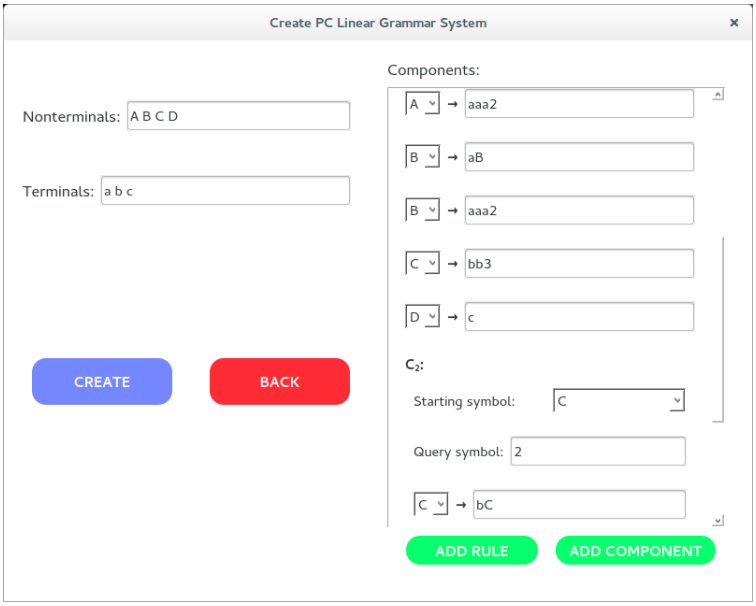
V programe je možné vytvoriť si vlastný CD, či PC lineárny gramatický systém, následne s ním pracovať, prípdne si ho uložiť v podobe XML súboru a v budúcnosti načítať. Po zadaní lineárneho gramatického systému program skontroluje sémantickú validitu zadaných údajov, syntax kontroluje už pri zadávaní. Po úspešnom vytvorení lineárneho gramatického systému je možné testovať, či zadaný reťazec je daným systémom generovaný alebo nie. V prípade, že je, program vypíše celý derivačný proces, teda kroky, ktoré viedli k vygenerovaniu výsledného reťazca lineárnym gramatickým systémom. Lineárne gramatické systémy je možné kedykoľvek upravovať.

Na obrázku 5.5 je vidieť, ako by sme nadefinovali a pracovali s PC lineárnym gramatickým systémom z druhého príkladu v teoretickej časti v podkapitole 4.2.1, ktorý generuje jazyk $\{a^n b^n c^n : n \geq 1\}$ a vyskúšali ním vygenerovať reťazec *aaabbbccc*, ktorý týmto systémom generovaný je a reťazec *aaabbbcccc*, ktorý, naopak, PC lineárny gramatický systém negeneruje.

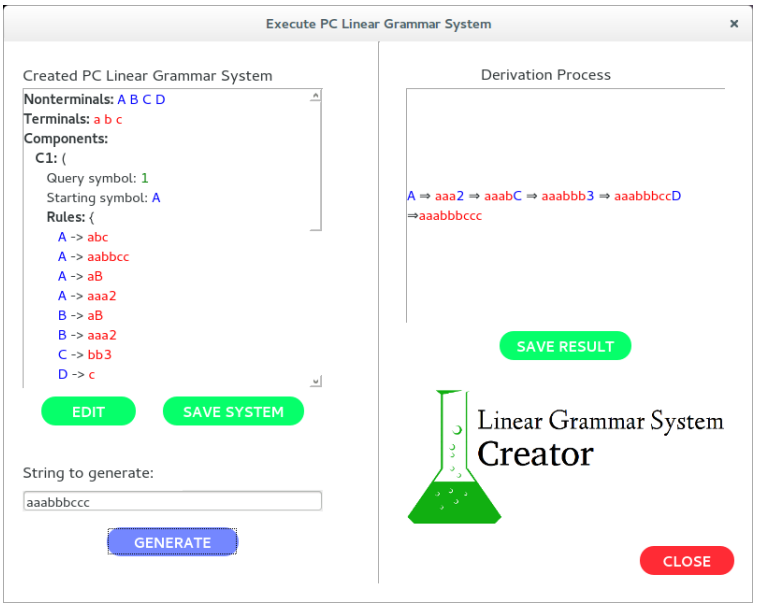
Z hlavného menu je možné otvoriť si viacero okien a vytvoriť a pracovať súčasne s viacerými lineárnymi gramatickými systémami. Výpočet, resp. beh lineárneho gramatického systému, je stále spúšťaný na novom vlákne.



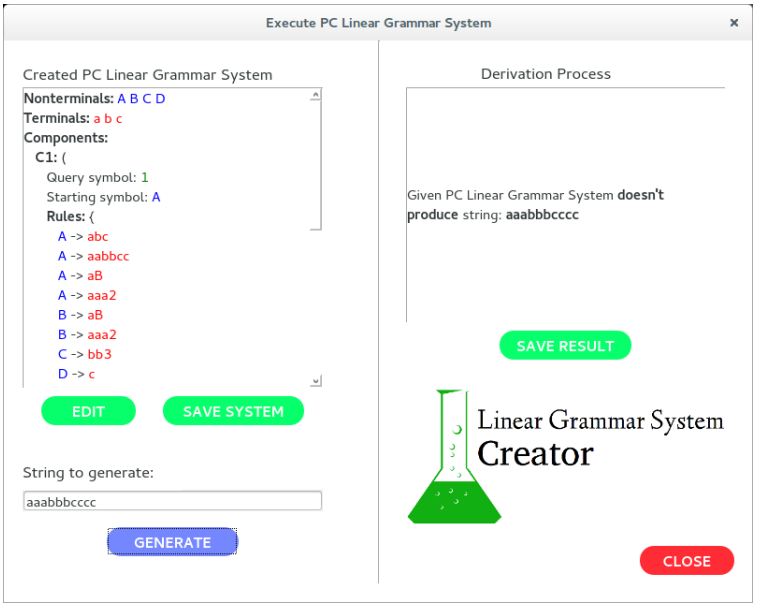
Obr. 5.2: Úvodné okno



Obr. 5.3: Vytvorenie PC lineárneho gramatického systému



Obr. 5.4: Ukážka reťazca, ktorý PC lineárny gramatický systém generuje



Obr. 5.5: Ukážka reťazca, ktorý PC lineárny gramatický systém negeneruje

Kapitola 6

Implementácia automatových systémov

V tejto kapitole sa budeme podrobne venovať programovej realizácii SP automatových systémov. Ako sme sa dozvedeli v teoretickej časti, generatívna sila SP automatových systémov ich predurčuje pre silný nástroj v oblasti syntaktickej analýzy. Oproti lineárnym gramatickým systémom majú výhodu v tom, že sa jedná o systém pracujúci nad automatmi, pričom v prípade použitia deterministických automatov, nie je potrebné generovať veľký priestor možných riešení a následne tento priestor prehľadávať za pomoci prehľadávacej metódy (napr. BFS), aj keď sa tento priestor môže použitím vhodných heuristík výrazne zmenšiť. Pri použití deterministických automatov v každom kroku výpočtu dokážeme presne určiť nasledujúci výpočetný krok, navyše, s použitím jednoobrátkových automatov sme schopní čiastočne limitovať nárast zásobníka, nakoľko po jeho prvom zmenšení už nie je možné jeho ďalšie rozširovanie.

Ukážeme a popíšeme implementačné problémy spojené s SP automatovými systémami a ukážeme najdôležitejšie časti implementácie. Jednotlivé príklady, vytvorené v teoretickej časti, budeme prezentovať na finálnom programe.

Pre implementáciu bol zvolený jazyk Java, najmä pre jeho prenositeľnosť a fakt, že ide o jazyk objektový, taktiež zaváži la jednoduchosť jeho použitia. Pre SP automatový systém a jeho podčasti, z ktorých sa skladá, boli priradené zodpovedajúce objekty, ktoré implementujú funkcionalitu jednotlivých podčastí. Okrem modelu SP automatového systému bolo vytvorené taktiež grafické užívateľské rozhranie s využitím JavaFX, ktoré nepotrebuje pridanie externých knižníc a celá aplikácia je teda jednoducho prenositeľná.

6.1 Štruktúra aplikácie

Dátový model systému bol rozdelený do niekoľkých základných tried. Trieda `Rule` reprezentuje pravidlo v jednoobrátkovom zásobníkovom automate, pričom samotný objekt je zložený z dvoch podobjektov, `RuleLeftPart` a `RuleRightPart`, teda objektov reprezentujúcich ľavú a pravú časť pravidla, taktiež je potrebné uchovávať informáciu, či sa jedná o pravidlo sekvenčné, alebo pravidlo paralelné. Z množiny pravidiel, štartovacieho symbolu na zásobníku a počiatočného stavu je možné zložiť objekt samotného jednoobrátkového zásobníkového automatu, ktorý reprezentuje trieda `OneTurnAutomaton`. Posledným objektom, a zároveň finálnym, je trieda reprezentujúca samotný SP automatový systém, ktorý je zložený z dvoch jednoobrátkových zásobníkových automatov, samotná trieda má názov

`SPAutomatonSystem`.

Na nasledujúcom obrázku 6.1 je možné vidieť diagram tried, ktorý popisuje dátovú štruktúru programu. Pre jednoduchosť a prehľadnosť neobsahuje triedy tvoriace grafické užívateľské rozhranie.

6.2 Reprezentácia dát

6.2.1 Reprezentácia jednoobrátkového zásobníkového automatu

Na základe definície sa jednoobrátkový zásobníkový automat skladá z počiatočného symbolu, ktorý by sme mohli jednoducho reprezentovať za pomoci znaku, zvolíme však dátový typ `String`, nakoľko ho budeme využívať pri reťazcoch, ktoré sú zložené z viacerých znakov a vyhneme sa tak zbytočným konverziám. Rovnako teda reprezentujeme aj počiatočný stav automatu. Okrem počiatočného stavu si ale potrebujeme pamätať aj jeho aktuálny stav, ktorý sa v priebehu výpočtu pravidelne mení. Pre uchovanie stavu výpočtu si okrem aktuálneho stavu potrebujeme zapamätať aj stav zásobníku, ktorý reprezentujeme taktiež za pomoci reťazca znakov. To si môžeme dovoliť, nakoľko symbol zásobníkovej abecedy môže byť reprezentovaný len jedným jediným znakom. Aby mohol automat vykonávať nejakú činnosť, potrebuje ešte pravidlá, na základe ktorých má výpočet prevádzať. To reprezentujeme za pomoci zoznamu jednotlivých pravidiel. Takto definovaný jednoobrátkový zásobníkový automat je pripravený na prijatie vstupného reťazca nad ktorým vykoná výpočet a rozhodne, či daný reťazec prijíma alebo nie.

6.2.2 Reprezentácia SP automatového systému

Automatový systém je zložený z dvoch jednoobrátkových zásobníkových automatov. Aby sme uchovali jeho súčasný stav, nepotrebujeme okrem spomínaných dvoch jednoobrátkových automatov žiadne ďalšie dáta, či dátové štruktúry, stav výpočtu si totižto pamätajú oba jednoobrátkové zásobníkové automaty. Je potrebné zvoliť len správny spôsob ich riadenia.

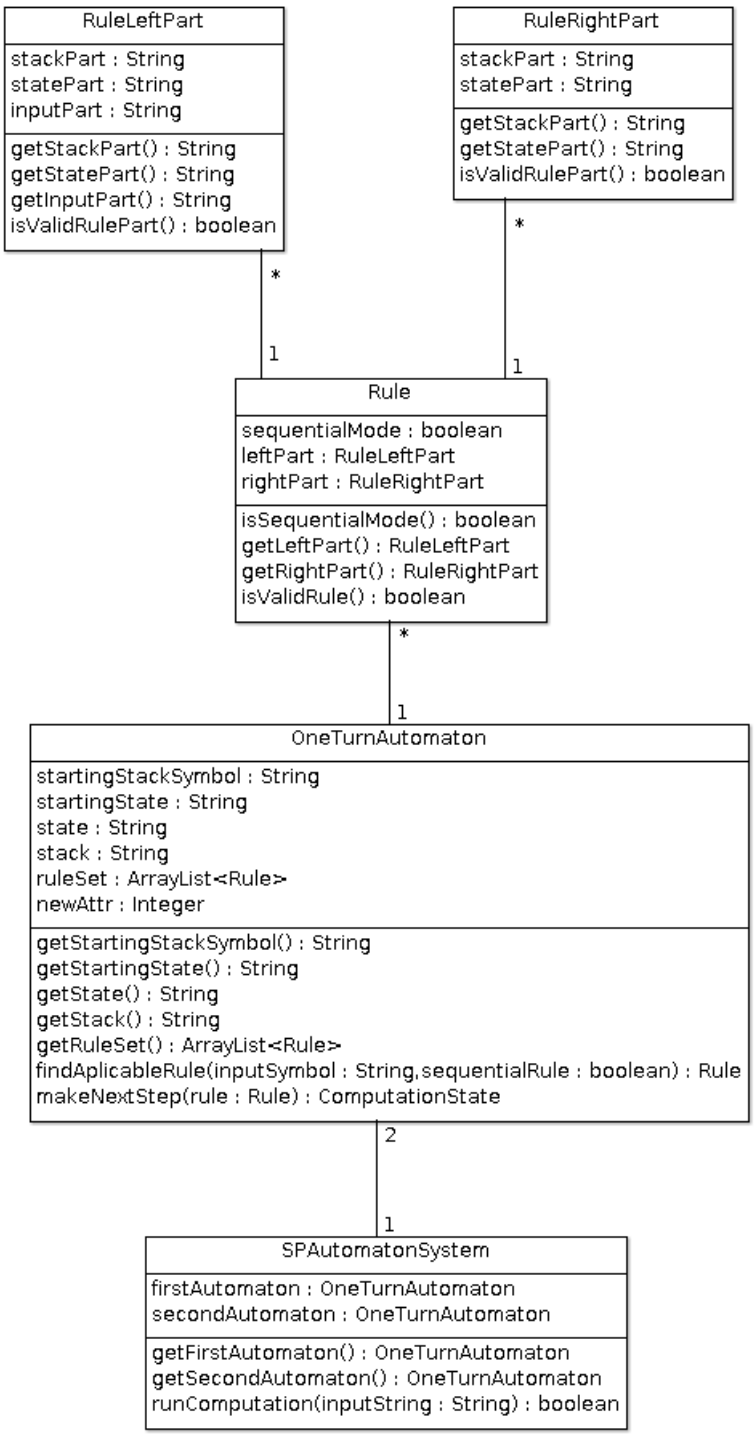
6.3 Implementácia metód automatového systému

V tejto podkapitole predstavíme najdôležitejšie metódy, ktoré implementujú jednotlivé programové riešenia spojené s automatovými systémami. Zameriame sa na chod jednoobrátkového zásobníkového automatu a taktiež na riadenie dvoch jednoobrátkových zásobníkových automatov, ktoré prevádza SP automatový systém.

6.3.1 Vyhľadanie správneho pravidla v jednoobrátkovom zásobníkovom automate

Prvým problémom, ktorý je potrebné riešiť, je výber správneho pravidla z množiny pravidiel, ktoré jednoobrátkový zásobníkový automat obsahuje. Aby bol výber pravidla možný, potrebujeme okrem stavu automatu, ktorý popisuje jeho súčasný stav a taktiež aktuálny obsah jeho zásobníka, vedieť aj symbol vstupného reťazca, ktorý sa aktuálne načítava a taktiež, s ohľadom na automatový systém, či hľadáme pravidlo sekvenčné, alebo paralelné.

Prvým krokom teda bude obmedziť sa len na pravidlá odpovedajúceho typu, čím vyfiltrujeme len pravidlá sekvenčné, resp. pravidlá paralelné a ďalej pokračujeme už len s mno-



Obr. 6.1: Diagram tried aplikácie Linear Grammar System Creator

žinou vyfiltrovaných pravidiel. Na základe súčasného stavu automatu a jeho aktuálneho stavu na zásobníku (nepozeráme sa len na vrchol zasobníka, keďže implementujeme rozšírený jednoobrátkový zásobníkový automat, ktorý nám dokáže mnoho problémov popísať deterministicky, pričom pri použití jeho nerozšírenej verzie by sme sa nedeterminizmu nevyhli) môžeme množinu pravidiel ďalej skresať. Takto zmenšenú množinu pravidiel jednoobrátkového zásobníkového automatu už len porovnáme s predaným symbolom vstupného čítaného reťazca a pokiaľ sme našli zhodu, nájdené pravidlo vrátime.

6.3.2 Prevedenie výpočetného kroku v jednoobrátkovom zásobníkovom automate

Pokiaľ sme boli schopní nájsť pravidlo, ktoré je možné aplikovať, aplikujeme ho a dostaneme sa tým do ďalšieho stavu výpočtu jednoobrátkového zásobníkového automatu. Môže nastať situácia, že sme žiadne pravidlo nenašli, čo však ešte neznamená, že reťazec môžeme hneď odmietnuť a prehlásiť, že automat ho neprijíma. Práve naopak, pokiaľ je zásobník prázdny a prečítali sme celý vstupný reťazec, prehlásime, že daný reťazec je prijatý.

Po aplikácii pravidla dostávame nový stav jednoobrátkového zásobníkového automatu, ktorý získame jednoducho z pravej strany pravidla a jeho časti, ktorá špecifikuje následný stav. Obdobne získame taktiež aj nový vrchol zásobníka, musíme však dávať pozor, či na zásobník niečo pridávame, alebo z neho mažeme. Pokiaľ sa teda na pravej strane pravidla reprezentujúceho nasledujúci nový vrchol zásobníka nachádza ϵ, novým vrcholom bude prázdny reťazec, inak reťazec špecifikovaný pravidlom. Nakoľko pracujeme s rozšírenými jednoobrátkovými zásobníkovými automatmi, nestačí zdmazať jeden znak z vrcholu zásobníka, ale zmazujeme toľko znakov, koľko je špecifikovaných v ľavej časti pravidla, ktoré aplikujeme.

6.3.3 Riadenie výpočtu v SP automatovom systéme

SP automatový systém sa stará o kontrolu výpočtu a riadi oba jednoobrátkové zásobníkové automaty. Nasledujúci algoritmus tvorí jeho riadiacu časť:

1. Vyber nasledujúci symbol vstupného reťazca a nastav ho ako aktuálne spracovávaný symbol vstupného reťazca. V prípade, že bol načítaný celý vstupný reťazec, ale SP automatový systém ešte stále neprijal ani neodmietol vstupný reťazec, nastav ako aktuálne spracovávaný symbol vstupného reťazca prázdny reťazec.

2. Nájdi sekvenčné pravidlo v prvom jednoobrátkovom zásobníkovom automate, ktoré je pre aktuálny symbol vstupného reťazca aplikovateľné. V prípade, že sa takéto pravidlo našlo, pokračuj bodom 5, inak pokračuj nasledujúcim bodom.

3. Nájdi sekvenčné pravidlo v druhom jednoobrátkovom zásobníkovom automate, ktoré je pre aktuálny symbol vstupného reťazca aplikovateľné. V prípade, že sa takéto pravidlo našlo, pokračuj bodom 6, inak pokračuj nasledujúcim bodom.

4. Ak sa nepodarilo nájsť sekvenčné pravidlo ani v jednom jednoobrátkovom zásobníkovom automate, pokús sa nájsť paralelné pravidlá v oboch automatoch. Pokiaľ sa takéto pravidlá našli, pokračuj bodom 7, v prípade, že sa pravidlá, ktoré je možné aplikovať nenašli a zároveň platí, že aktuálny symbol vstupného reťazca je prázdny reťazec, to znamená, celý vstupný reťazec bol načítaný a zároveň oba jednoobrátkové

zásobníkové automaty majú prázdne zásobníky, ukonči výpočet a prijmi vstupný reťazec, inak ho odmietni.

5. Preveď výpočetný krok v prvom automate a skontroluj jeho stav. Pokiaľ sa nachádza v stave výpočtu, pokračuj bodom 1. Ak automat prijal vstup a zároveň bol načítaný celý vstupný reťazec, rovnako aj druhý jednoobrátkový zásobníkový automat má prázdny zásobník, prijmi vstupný reťazec. Ak automat odmietol, ukonči výpočet a odmietni vstupný reťazec.

6. Preveď výpočetný krok v druhom automate a skontroluj jeho stav. Pokiaľ sa nachádza v stave výpočtu, pokračuj bodom 1. Ak automat prijal vstup a zároveň bol načítaný celý vstupný reťazec, rovnako aj prvý jednoobrátkový zásobníkový automat má prázdny zásobník, prijmi vstupný reťazec. Ak automat odmietol, ukonči výpočet a odmietni vstupný reťazec.

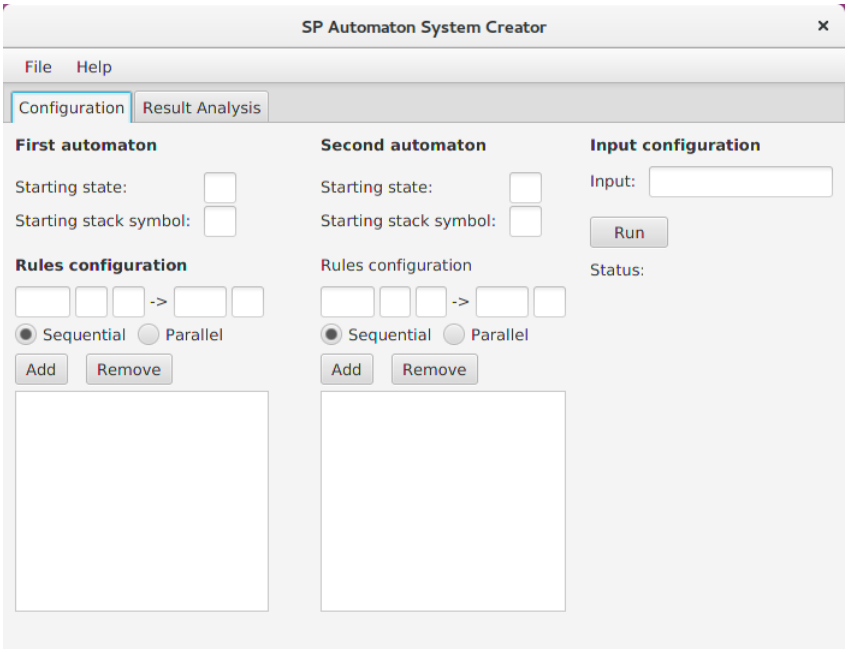
7. Preveď výpočetný krok v prvom aj druhom jednoobrátkovom zásobníkovom automate a skontroluj ich stav. Pokiaľ je stav oboch pokračovanie vo výpočte, choď na bod 1, ak jeden z nich odmietol, odmietni celý celý vstupný reťazec. Pokiaľ oba jednoobrátkové zásobníkové automaty prijali, skontroluj, či bol načítaný celý vstupný reťazec a pokiaľ áno, prijmi ho.

6.4 SP Automaton System Creator

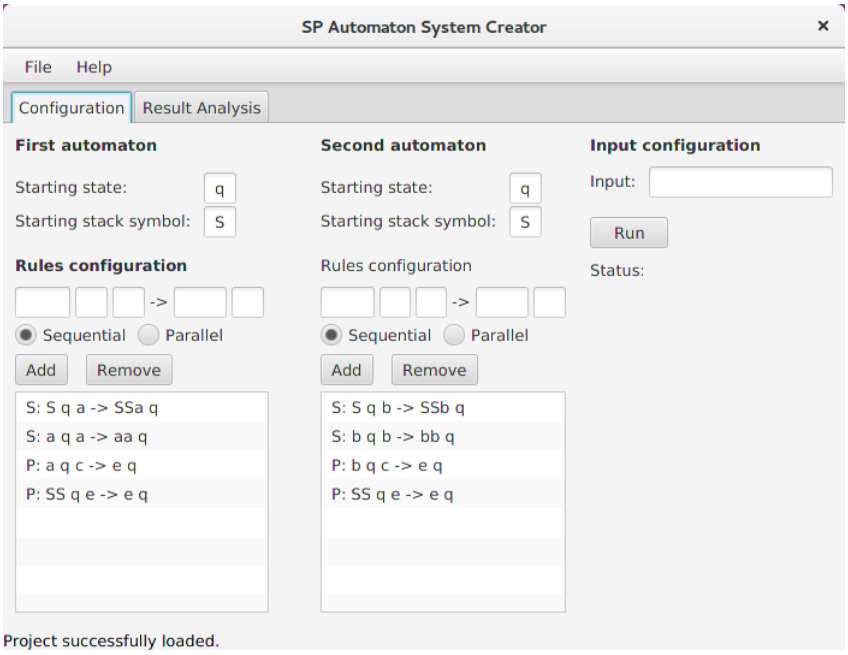
SP Automaton System Creator je program, ktorý obsahuje grafické užívateľské rozhranie a využíva model SP automatového systému. Umožňuje jednoduchú prácu za pomoci grafického užívateľského rozhrania a abstrahuje tak od použitia príkazového riadku, kde je zložité a nepraktické definovanie celého SP automatového systému. Grafická nadstavba bola vytvorená za pomoci JavaFX (viac viď. [2]).

Program umožňuje definovať si jednotlivé jednoobrátkové zásobníkové automaty. Užívateľ si môže zvoliť ich počiatočné stavy a počiatočné symboly na zásobníku a taktiež nadefinovať jednotlivé pravidlá pre oba jednoobrátkové zásobníkové automaty. Aby užívateľ nemusel po každom novom spustení programu definovať celý SP automatový systém odznova, je možné uložiť si finálny alebo rozpracovaný projekt v podobe XML súboru (schopnejší užívateľ môže editovať priamo tento súbor, ktorý má jednoduchú štruktúru). Po úspešnom nadefinovaní celého SP automatového systému užívateľ zadá zvolený vstupný reťazec a spustí výpočet, ktorý beží na samostatnom vlákne. Odpoveďou je mu buď, že SP automatový systém reťazec prijíma alebo naopak, neprijíma. V druhej záložke je možné si prehliadnuť podrobnú analýzu jednotlivých krokov výpočtu, kde je vždy uvedený stav jednoobrátkových zásobníkových automatov a ich zásobníkov v priebehu výpočtu, použité pravidlá a zostávajúca časť vstupného reťazca.

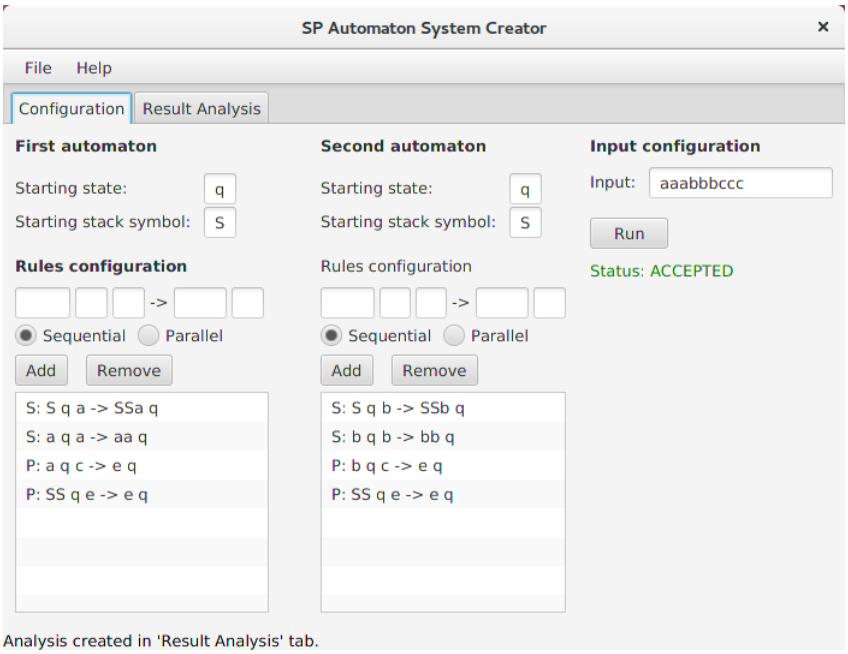
Na obrázku 6.3 je možné pozorovať, ako by sme nadefinovali jazyk $\{a^n b^n c^n, n \geq 1\}$, ktorý bol podrobne rozoberaný v teoretickej časti. Obrázok 6.4 ilustruje prijatie validného reťazca *aaabbbccc* a jeho podrobnú analýzu na obrázku 6.5.



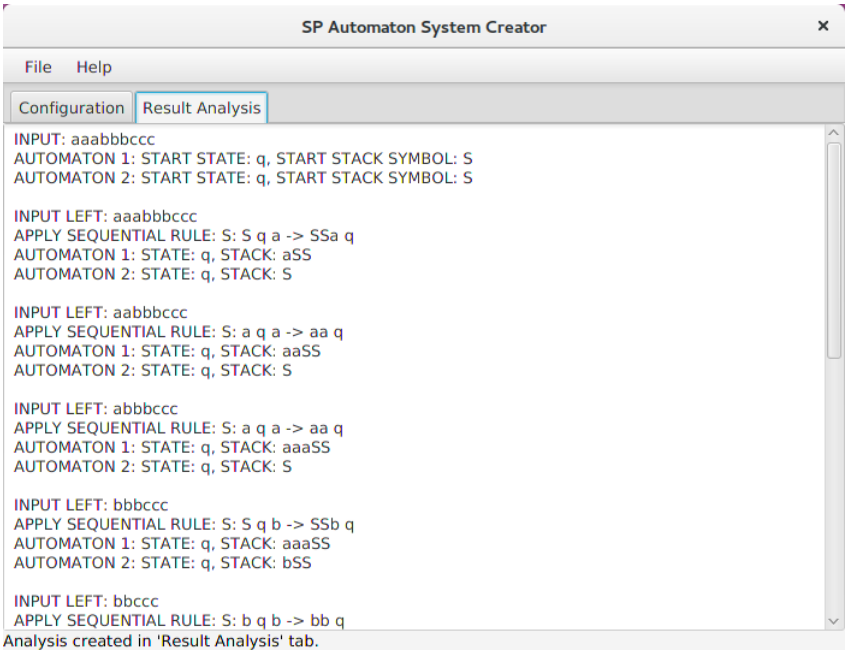
Obr. 6.2: Úvodné okno



Obr. 6.3: Vytvorenie SP automatového systému



Obr. 6.4: Ukážka reťazca, ktorý SP automatový systém prijima



Obr. 6.5: Ukážka podrobnej analýzy

Kapitola 7

Testovanie implementovaných programov

V tejto časti sa zameriame na testovanie jednotlivých programov. Rovnaké testovacie metódy boli využité ako v prípade programu Linear Grammar System Creator, tak aj pri programe SP Automaton System Creator. Počas písania zdrojových kódov pre oba programy boli priebežne vytvárané testovacie súbory, ktoré stále otestovali novopridanú funkcionalitu. V záverečnej fáze testovania boli programy otestované ako celok.

7.1 Testovanie jednotlivých častí programov

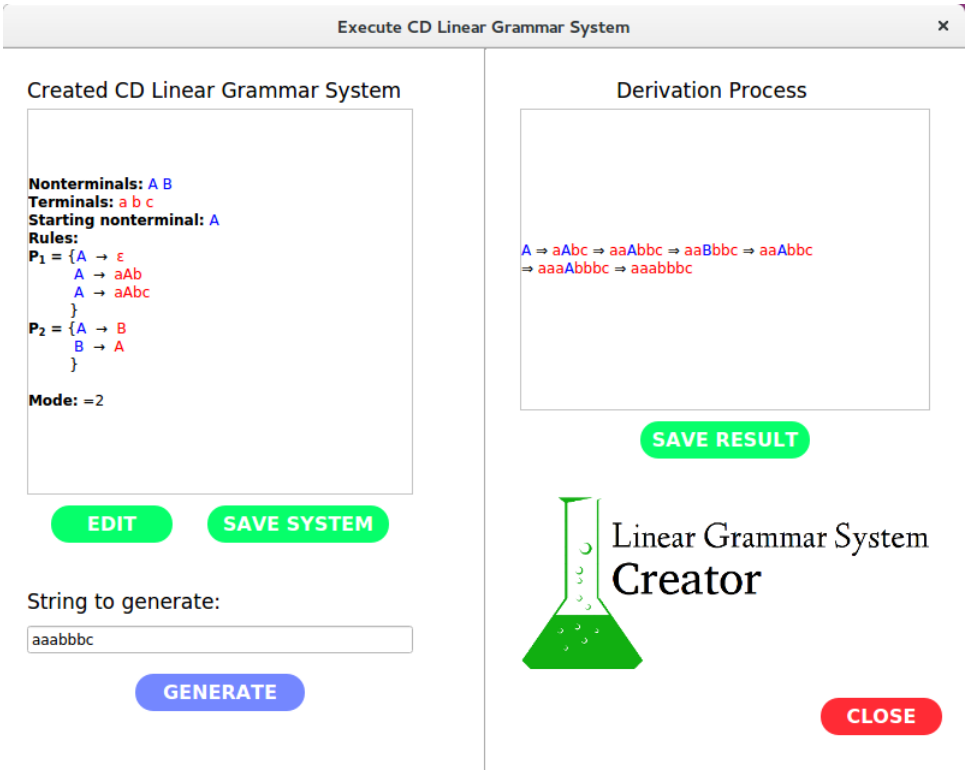
Ako sme už spomenuli v úvode, programy boli priebežne testované počas svojej tvorby. Spočiatku to zahŕňalo najmä testovanie základnej funkcionality, akou je ošetrenie validne zadaných pravidiel, či už v prípade lineárnych gramatických systémov, ale aj v prípade SP automatového systému.

Validita pravidiel bola predpokladom pre tvorbu výpočetného algoritmu operujúceho nad vstupným reťazcom pri zadaných pravidlách. Ďalším krokom bolo otestovať práve tento riadiaci algoritmus. V prípade lineárnych gramatických systémov išlo o správne generovanie stavového priestoru riešení a elimináciu zbytočného generovania potencionálnych riešení, ktoré však nemohli k reálnemu riešeniu nikdy dospieť. To znamená, že bolo potrebné otestovať všetky heuristiky algoritmu a nakoniec algoritmus samotný. Nakoľko SP automatový systém používa iný výpočetný model, ktorý nemusí generovať priestor riešení, ale je v každom kroku výpočtu schopný určiť nasledujúci krok, hlavné zameranie testovania sledovalo práve riadiaci algoritmus a správny výber pravidiel s ohľadom na prioritu sekvenčných pravidiel a prvého definovaného jednoobrátkového zásobníkového automatu.

Po pridaní grafického užívateľského rozhrania bolo potrebné otestovať metódy implementujúce validáciu polí, do ktorých užívateľ zadáva definície systémov a správne vytvorenie modelu. Taktiež sa muselo vytvoriť viacero vlákien pre výpočet, aby grafické užívateľské rozhranie nezamrzlo.

7.2 Testovanie programov ako celku

Po aplikácii testov jednotlivých dôležitých častí programu a významných metód sme pristúpili k spojeniu jednotlivých modulov do výsledných programov. V oboch programoch

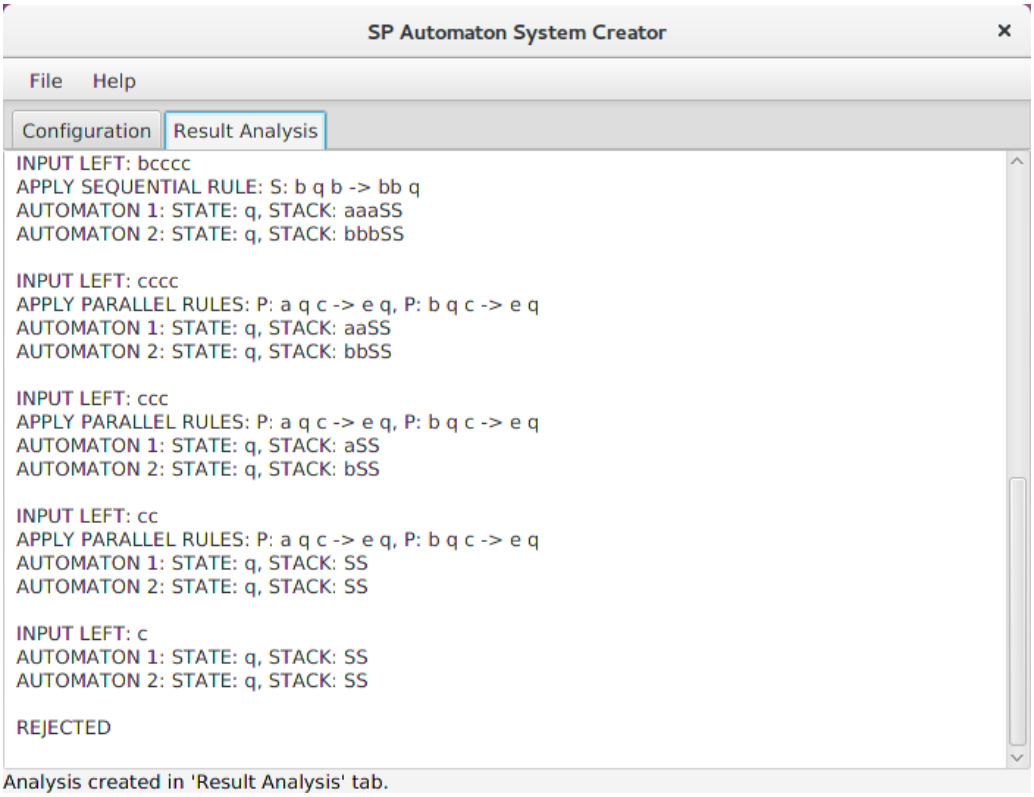


Obr. 7.1: Výsledok pri správnom fungovaní heuristiky

bolo potrebné otestovať, či nevznikli nové chyby, ktoré sa v jednotlivých častiach neprejavili. Jednalo sa najmä o komunikáciu medzi modelom a rozhraním, upozorňovanie užívateľa na chybný syntax alebo sémantiku jeho definície a pod.

Na obrázku 7.1 môžeme vidieť test na fungovanie celého programu v prípade, kedy gramatika obsahuje cyklické pravidlá. Vidíme, že program celkovo funguje správne a nedošlo k zamrznutiu aplikácie a bola využitá heuristika práve pre tento prípad.

V prípade programu Automaton System Creator môžeme na obrázku 7.2 vidieť správne odmietnutie reťazca, ktorý nie je generovaný zadaným SP automatovým systémom a vytvorenú analýzu spolu s informáciou, po ktorý znak reťazca prebiehal výpočet a kedy sa rozhodlo o zamietnutí reťazca.



Obr. 7.2: Výsledok analýzy zamietnutého reťazca

Kapitola 8

Záver

Prvým cieľom tejto práce bolo naštudovať si problematiku gramatických systémov a následne ich modifikovať na systémy, ktoré nebudú založené na klasickej báze bezkontextových gramatík, ale obmedziť silu pravidiel na lineárne gramatiky. Študovali sme, či prevedená modifikácia nebude limitujúca natoľko, aby zabránila zvýšeniu ich generatívnej sily. Podobný postup sme zvolili aj pri práci s automatovými systémami. Za základnú jednotku automatového systému sme si vybrali jednoobrátkový zásobníkový automat, ktorý je svojou vyjadrovacou silou ekvivalentný lineárnym gramatikám, čím je jasne vidieť, že tento zámer bol prevedený s ohľadom na zvolenú bázu v prípade gramatických systémov.

Po naštudovaní teoretického základu klasických gramatických systémov, ktoré sú založené na báze bezkontextových gramatík a vyšetrení ich generatívnej sily, sme pristúpili k samotnej modifikácii. Skúmali sme modifikované CD a PC lineárne gramatické systémy a dospeli sme k zaujímavým poznatkom. Očakávali sme, že ich generatívna sila sa určite zvýši, oproti použitiu samostatných lineárnych gramatík, taktiež nás zaujímalo, či sa ich sila bude líšiť od klasických gramatických systémov založených na báze bezkontextových gramatík, kde sme očakávali, že ich generatívna sila bude pravdepodobne menšia. Prekvapujúco, CD lineárne gramatické systémy nie sú schopné za pomoci sekvenčného riadenia a distribúcie úloh medzi jednotlivé lineárne gramatiky zvýšiť generatívnu silu oproti lineárnym gramatikám, čo sa vzhľadom na klasické gramatické systémy neočakávalo. Naopak, PC lineárny gramatický systém dokázal zavedením paralelizmu a konštrukciou jednotlivých častí reťazca v odpovedajúcich komponentoch paralelne zvýšiť generatívnu silu oproti CD lineárnym gramatickým systémom. V tomto prípade bolo ďalším prekvapujúcim faktom to, že sme ukázali, že každý klasický PC gramatický systém založený na báze lineárnych gramatík, sme schopní transformovať na PC lineárny gramatický systém, čím sme potvrdili, že PC lineárne gramatické systémy majú rovnakú generatívnu silu, ako ich bezkontextové varianty.

Poznatky, nadobudnuté pri skúmaní CD a PC gramatických systémov, sme pretavili pri vytváraní automatového systému. Vychádzali sme z prekvapivých poznatkov a rozhodli sme sa, že pokiaľ chceme generatívnu silu zvýšiť oproti jednoobrátkovému zásobníkovému automatu, ktorý pracuje sám, musíme do systému zaviesť paralelizmus. Zvolili sme teda zaujímavý prístup k riešeniu problému, kde sme nezanevreli ani na sekvenčný kooperatívny prístup a vytvorili sme systém pravidiel automatu tak, aby bolo možné definovať, ktoré pravidlá môžu byť vykonávané sekvenčne, a ktoré naopak, musia byť vykonávané paralelne v oboch jenoobrátkových zásobníkových automatoch SP automatového systému. Úspešne sa nám podarilo ukázať, že generatívnu silu sa nám podarilo zvýšiť a že zvolený myšlienkový postup a inšpirácia v oblasti lineárnych CD a PC gramatických systémov bola správna.

V závere práce sme nezostali len pri téorii, ale získané teoretické poznatky sme využili pri tvorbe a implementácii programov, v ktorých je možné si vytvoriť ako CD a PC lineárne automatové systémy, tak aj SP automatové systémy. Linear Grammar System Creator je, ako už jeho názov napovedá, program pre syntaktickú analýzu reťazcov, ktorý umožnuje prevádzať syntaktickú analýzu pomocou CD alebo PC lineárnych gramatických systémov. Pre prácu s SP automatovými systémami bol vytvorený obdobný program s názvom SP Automaton System Creator. Oba programy okrem odpovede na otázku, či je daný reťazec generovaný gramatickým, resp. prijímaný automatovým systémom ponúkajú aj náhľad na podrobnú analýzu ako výpočet prebiehal.

Do budúcnosti by práca mohla byť odrazovým mostíkom pre návrh komplexnej syntaktickej analýzy, ktorá by si vyžadovala prostriedky schopné analyzovať syntaktické štruktúry, ktoré nie sú bezkontextové. Najmä použitie SP automatového systému je veľmi sľubné, nakoľko automaty sú neoddeliteľnou súčasťou syntaktickej analýzy a fakt, že sa nám podarilo ukázať, že sme zvýšili silu, aj keď SP automatový systém využíval deterministické jednoobrátkové zásobníkové automaty, je pre praktické využitie veľmi zaujímavy a perspektívny.

Literatúra

[1] Qt Project – Documentation [online]. Dostupné na: http://www.qt-project.org/doc/, Posledná zmena 2014-02-14 [cit. 2016-04-17].

[2] JavaFX – Documentation [online]. Dostupné na: http://www.oracle.com/technetwork/java/javafx/documentation/index.html, Posledná zmena 2016-04-20 [cit. 2016-05-01].

[3] Csuhaj-Varjú E., e. a.: *Grammar systems: A Grammatical Approach to Distribution and Cooperation*. Langhorne, Pennsylvania: Gordon and Breach, 1994, ISBN 28-812-4957-4.

[4] Daintith J., E., Wright: *A dictionary of computing*. New York: Oxford University Press, 6 vydání, 2008, ISBN 01-992-3401-9.

[5] Lukáš, R.: *Multigenerativní gramatické systémy*. Brno, FIT VUT v Brně: disertační práce, 2006.

[6] Meduna, A.: *Automata and Languages: theory and applications*. London: Springer, 2000, ISBN 18-523-3074-0.

[7] Meduna, A.: Simultaneously one-turn two-pushdown automata [online]. Dostupné na: http://www.fit.vutbr.cz/~meduna/zaznam/am_articlespdf.pdf, Posledná zmena 2003-06-01 [cit. 2015-12-22].

[8] Meduna, A.: CD Grammar Systems [online]. Dostupné na: www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:09-cdgspres.pdf, Posledná zmena 2009-10-21 [cit. 2016-01-12].

[9] Meduna, A.: PC Grammar Systems [online]. Dostupné na: http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:10-pcgspres.pdf, Posledná zmena 2009-10-21 [cit. 2016-01-16].

[10] Meduna, A.: Matrix Grammars [online]. Dostupné na: http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:04-matgrampres.pdf, Posledná zmena 2009-10-21 [cit. 2016-01-18].

[11] Češka, M.: Teoretická informatika [online]. Dostupné na: https://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf, Posledná zmena 2014-09-20 [cit. 2016-02-17].

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje text diplomovej práce v súbore `diplomova_praca.pdf` a dva adresáre, `Linear Grammar System Creator` a `SP Automaton System Creator`, ktoré obsahujú zdrojové a spustiteľné súbory odpovedajúcich programov a majú nasledujúcu adresárovú štruktúru:

`bin`	adresár obsahujúci binárny spustiteľný súbor po preklade
`jar`	adresár obsahujúci java archív s programom
`examples`	adresár obsahujúci príklady PC a CD lineárnych gramatických resp. automatových systémov systémov definovaných v XML súboroch
`obj`	adresár obsahujúci objektové súbory po preklade
`src`	adresár obsahujúci zdrojové súbory
`Makefile`	súbor pre preklad a spustenie programu

Príloha B

Inštalácia

B.1 Inštalácia programu Linear Grammar System Creator

Program využíva Qt framework pre grafické užívateľské rozhranie, konkrétne verziu 5.2.0, ktorá je dostupná aj na školskom serveri Merlin. Pre inštaláciu stačí skopírovať adresárovú štruktúru z CD do požadovaného adresára a preklad previesť pomocou príkazu `make`, ktorý prevedie preklad programu za pomoci programu `qmake`. Pokiaľ sa preklad uskutočňuje na školskom serveri Merlin alebo na počítači s viacerými verziami Qt frameworku, pričom prioritne je volaný program `qmake` staršej verzie ako je verzia 5.2.0, je potrebné zavolať program `qmake` explicitne pomocou absolútnej cesty (v prípade prekladu na školskom serveri Merlin stačí zakomentovať v Makefile riadok s príkazom `qmake` a odkomentovať riadok s explicitne zadanou absolútnou cestou `/usr/local/share/Qt-5.2.1/5.2.1/gcc_64/bin/qmake`).

B.2 Inštalácia programu SP Automaton System Creator

Keďže program je vytvorený v jazyku Java a taktiež aj grafické rozhranie je vytvorené za pomoci JavaFX, preklad ani spustenie programu nie je vôbec komplikovaný. V programe boli využívané lambda konštrukcie, kvôli čomu je potrebná minimálne verzia Java 1.8. Pre spustenie stačí použiť príkaz `make`, ktorý spustí program z JAR archívu.

Príloha C

Manuál

C.1 Manuál k programu Linear Grammar System Creator

Po spustení programu Linear Grammar System Creator sa zobrazí úvodná obrazovka s hlavným menu, kde je možné vybrať si jednu zo štyroch hlavných možností: vytvorenie nového CD alebo PC lineárneho gramatického systému, alebo načítanie už vytvorených. Hlavné menu je stále prístupné a je možné otvoriť z neho viacero okien a pracovať tak súčasne s viacerými lineárnymi gramatickými systémami.

Pri voľbe vytvorenia nového CD či PC lienárneho gramatického systému je potrebné dodržiavať nasledujúce pravidlá: neterminály a terminály sa môžu skladať len z jedného znaku anglickej abecedy, pričom sa rozlišujú veľké a malé písmená, jednotlivé znaky sa oddeľujú medzerou. Predvolený komunikačný symbol je možné zmeniť na vlastný, pričom platia rovnaké pravidlá ako pri neterminálach a terminálach, naviac je možné okrem písmen anglickej abecedy použiť pre označenie aj čísla od 0 do 9. Pravá strana pravidiel môže, pochopiteľne, pozostávať len z neterminálov, terminálov a v prípade PC lineárneho gramatického systému aj z komunikačných symbolov, medzi znakmi sa nezadáva žiadna medzera. Pri vytváraní PC gramatického systému je potrebné rešpektovať, že program podporuje len centralizované PC lineárne gramatické systémy bez návratu. Program informuje užívateľa o syntaktických chybách už pri vytváraní systému v stavovom riadku v spodnej časti okna. Sémantická stránka sa skontroluje po požiadavke pre vytvorenie systému, pričom je užívateľ na prípadné chyby upozornený rovnako, ako v prípade syntaktických.

Po úspešnom vytvorení lineárneho gramatického systému sa zobrazí okno aplikácie, kde je možné zistiť, či vytvorený lineárny gramatický systém generuje alebo negeneruje zadaný reťazec. Výsledok je možné si uložiť, rovnako ako aj vytvorený lineárny gramatický systém. Na výber je taktiež možnosť editácie systému, teda návrat do predchádzajúceho okna.

C.2 Manuál k programu SP Automaton System Creator

Po spustení programu SP Automaton System Creator sa zobrazí hlavné okno aplikácie. Toto okno je rozdelené na tri základné časti: definícia prvého jednoobrátkového zásobníkového automatu, definícia druhého jednoobrátkového zásobníkového automatu a časť pre zadanie vstupného reťazca a zobrazenie výsledku.

Pri definícii jednotlivých automatových systémov je potrebné dodržiavať dĺžku jedného znaku pre zadanie štartovacieho stavu a počiatočného symbolu na zásobníku, čo je aj programovo ošetrené a nie je možné zadať viacero znakov. Na chybne zadané pravidlá je užívateľ

upozornený pomocou stavového riadku v spodnej časti okna. Zadané pravidlo je potrebné pridať, už vytvorené pravidlá je taktiež možné vymazávať.

Po zadaní všetkých požadovaných informácií je možné spustiť analýzu, ktorej výsledky je možné nájsť na druhej záložke hlavného okna. Výslednú prácu je možné si uložiť v menu File, pre nápovedu je možné zobraziť pomoc v menu Help.