VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

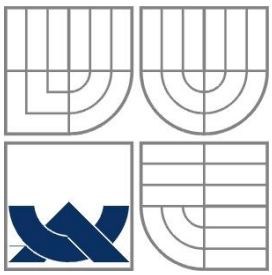
GRAMATICKÉ SYSTÉMY A SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA NICH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

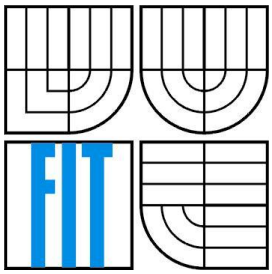
AUTOR PRÁCE
AUTHOR

JAROSLAV HANDLÍŘ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GRAMATICKÉ SYSTÉMY A SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA NICH

GRAMMAR SYSTEMS AND PARSING BASED ON THEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAROSLAV HANDLÍŘ
AUTHOR

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

BRNO 2015

Abstrakt

Tato práce se věnuje problematice gramatických systémů. Definuje jak kooperující distribuované CD gramatické systémy, tak i paralelně komunikující PC gramatické systémy ve spojení s bezkontextovými gramatikami. Dále zkoumá jejich principy a generativní síly v různých režimech a vůči různým jazykům. Studuje využití gramatických systémů při syntaktické analýze. Zaměřuje se především na spojení syntaktické analýzy shora dolů s precedenční analýzou zdola nahoru ve spojitosti s PC gramatickými systémy.

Abstract

This work is devoted to grammar systems. Defines cooperating distributed grammar systems CD and parallel communicating grammar systems PC in conjunction with context-free grammars. It also examines their principles and generative forces in different regime and to different languages. Studying the use of grammatical parsing systems. It focuses primarily on top-down parsing and precedential bottom-up analysis in conjunction with the PC grammar systems.

Klíčová slova

gramatické systémy, CD, PC, bezkontextové gramatiky, syntaktická analýza, precedenční analýza

Keywords

grammar systems, CD, PC, context-free grammars, syntax analysis, precedence analysis

Citace

Jaroslav Handlíř: Gramatické systémy a syntaktická analýza založená na nich, bakalářská práce, Brno, FIT VUT v Brně, 2015

Gramatické systémy a syntaktická analýza založená na nich

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením profesora Alexandra Meduny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Handlíř
20. května 2015

Poděkování

Zde bych rád poděkoval svému vedoucímu bakalářské práce profesoru Alexandru Medunovi za jeho čas, odborný dohled, konzultace a vedení.

Obsah

1 Úvod

Jádro této práce je věnováno problematice gramatických systémů, které vznikly jako odezva na stále více se rozšiřující trend v oblasti distribuovaného zpracování a paralelismu. Postupem let přešly technologie od jednojádrových procesorů a čipů, které nesporně položily základ současného informačního věku, k vícejádrovým procesorům a dnes bychom těžko hledali moderní zařízení, ať už stolní počítač, notebook, tablet či chytrý telefon, který by nebyl multijádrový, nemluvě o serverech nebo výzkumných stanicích. Avšak už v dřívějších dobách se rozvinula stále využívaná technologie kvaziparalelismu, který simuluje paralelní zpracování na jednojádrových systémech, a byl tak jakýmsi předchůdcem současných trendů. Tento velký „informační boom“ postihl i ostatní oblasti výpočetních technologií, z nichž za zmínku stojí rozvoj počítačových sítí jak v souvislosti s dnes již všudypřítomnou globální sítí, internetem, tak i vzájemná komunikace pracovních stanic, přerozdělování a distribuce výpočetní zátěže mezi více zařízení, která mohou být rozmístěna třeba po celém světě. To vše má velký dopad i na teorii programovacích jazyků, protože svět informačních technologií tvoří nepřeberné množství vzájemně spolupracujícího hardwaru, nad nímž pracuje software v celé škále programovacích jazyků a jejich derivací, proto je existence formálních aparátů nepostradatelná a umožňuje nám tak konečnou definici formálních jazyků. Specifikace konečných jazyků lze provést výčtem jejich slov, avšak to nelze u nekonečných jazyků, které tvoří téměř veškeré programovací jazyky. Zde nastupují již zmíněné formální aparáty (gramatiky, automaty a další), které dokáží popsat konečné i nekonečné jazyky.

Již v roce 1956 zavedl americký filozof a lingvista Noam Chomsky klasifikaci čtyř základních typů gramatik (typ 0 až typ 3), přičemž se tato práce věnuje typu 2 a okrajově v praktické části typu 3, které se jinak nazývají bezkontextová gramatika resp. regulární gramatika a jsou nejvýznamnějšími právě při definici programovacích jazyků. Tyto dva typy lze dále definovat pomocí automatů, zásobníkovým resp. konečným automatem, kterými lze realizovat první dvě části kompilátorů (lexikální a syntaktická analýza) a tvoří jádro praktické části této práce [3].

Jak již bylo uvedeno, bezkontextové gramatiky mají velký význam nejen při samotné definici programovacích jazyků, ale i následné kontrole a překladu zdrojových kódů daných jazyků. Zde v praxi narážíme na situaci, kdy by bylo v mnoha ohledech výhodnější mít pro daný programovací jazyk více než jednu gramatiku, která pro danou část kódu pracuje rychleji a efektivněji, než kdyby celý jazyk pokrývala sice jen jedna gramatika, ale za cenu časové a výpočetně náročné nekonzistence při zpracování. Ruku v ruce se stále větším využitím paralelismu a distribuce v informačních technologiích vznikly gramatické systémy, které nejenže umožnují koordinaci více gramatik, ale do zpracování zapojují i paralelismus.

Ačkoli se gramatické systémy zdají být novinkou v teorii formálních jazyků, jejich vývoj započal již ve druhé polovině dvacátého století a postupně se tyto techniky zpracování vyvíjely i v dalších odvětvích. Můžeme tak znát různé již zaběhnuté modely pro distribuované či paralelní zpracování/řízení činnosti z praktického světa. Mezi nejznámější patří nepochybně model nástěnky nebo tabule. O některých modelech bude v této práci ještě zmínka, protože na dobře známých nebo snadno představitelných reálných situacích lze pak snadno vysvětlit principy jednotlivých systémů.

Gramatické systémy se dělí na dva základní typy. Distribuované CD gramatické systémy pracují sekvenčně. Všechny komponenty sdílí jednu větnou formu, na které pracuje vždy jen jedna komponenta, a střídají se dle konkrétního módu systému. Naopak PC gramatické systémy pracují paralelně. Nesdílejí větnou formu, ale každá gramatika pracuje na své vlastní, což umožňuje

požadované souběžné zpracování. Vzájemně spolu komunikují pomocí tzv. komunikačních symbolů (query symbols) Q.

V následujících kapitolách se postupně dočteme o jednotlivých oblastech gramatických systémů. Nejdříve si však v kapitole 2 nastíníme teorii bezkontextových gramatik spolu s jejich obecnou definicí. Zajímavé jsou v této kapitole příklady, které by měly čtenáři názorně přiblížit celou problematiku, protože alespoň základní znalost bezkontextových gramatik je nutná pro pochopení dalších kapitol. Dále se již můžeme plně věnovat samotným gramatickým systémům. Jednotlivé podkapitoly vysvětlí důvody jejich vzniku, teoretické pozadí a popíšeme si i samotné definice. V podkapitole 3.1 se dočteme o kooperujících distribuovaných gramatických systémech, neboli zkráceně CD gramatických systémech. Podkapitola 3.2 je pak zaměřena na paralelně komunikující gramatické systémy, zkráceně PC gramatické systémy. Obě podkapitoly jsou podobně rozděleny do jednotlivých sekcí, které nejdříve popíší obecné definice, poté vysvětlí principy, jak oba systémy pracují a jaké jazyky generují. Závěr podkapitol uvádí třídy generovaných jazyků a porovnává jejich generativní síly. Obě podkapitoly, stejně jako kapitola o bezkontextových gramatikách, obsahují názorné příklady s podrobným rozborem pro co nejlepší vysvětlení a pochopení principů. Poslední teoretická část, popsána ve 4. kapitole, je zaměřena na syntaktickou analýzu, která je velmi významná pro konstrukci kompilátorů. Uvedeme si tedy různé přístupy, které se běžně v praxi využívají. Z pochopitelných důvodů je větší pozornost věnována metodám použitým v následné praktické části, o které se dočteme v kapitole 5. Poslední kapitola, kapitola 6, shrne závěrem celou problematiku a navrhne možné směry dalšího vývoje. Po ní následuje seznam použité literatury a přílohy.

2 Bezkontextové gramatiky

Přestože tato práce předpokládá základní přehled v oblasti formálních jazyků, uvedeme si na úvod alespoň definice aparátů, které budeme potřebovat. $Abeceda$ je konečná, neprázdná množina elementů, které nazýváme $symboly$. Zpravidla ji značíme Σ, potom abecedu obsahující např. symboly a, b, c lze zapsat $\Sigma = \{a, b, c\}$. Jakmile jsme definovali abecedu Σ, můžeme nad ní vytvořit $řetězec$. Platí, že ε je řetězec nad abecedou Σ. Symbol ε reprezentuje prázdný řetězec, to znamená, že neosahuje žádný symbol. Pokud je x řetězec nad Σ a $a \in \Sigma$, potom i xa je řetězec nad abecedou Σ. Nyní máme vše potřebné, abychom mohli definovat $jazyk$, o kterém budeme v této práci mluvit velmi často. Nechť Σ^* značí množinu všech řetězců nad Σ, potom každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad abecedou Σ. Dalším velmi důležitým pojmem je $terminál$ resp. $neterminál$. Za terminály považujeme znaky abecedy Σ a zpravidla je značíme malými písmeny, zatímco neterminály označují ostatní symboly a zapisujeme je velkými písmeny. Podrobnější výklad spolu s dalšími definicemi lze nalézt v publikaci [2]. Nyní můžeme přejít k bezkontextovým gramatikám.

Bezkontextovou gramatiku G lze definovat jako čtveřici $G = (N, T, P, S)$, kde:

- N je abeceda neterminálů
- T je abeceda terminálů, přičemž $N \cap T = \emptyset$
- P je konečná množina pravidel tvaru $A \to x$, kde $A \in N, x \in (N \cup T)^*$
- $S \in N$ je počáteční neterminál

Bezkontextová gramatika pracuje s konečnou množinou pravidel P, pomocí kterých se generují řetězce daného bezkontextového jazyka $L(G)$ nebo lze již existující řetězce měnit. Pak mluvíme o tzv. derivačním kroku, kterým nahrazujeme jednotlivé symboly na základě pravidel P. Této schopnosti, měnit již existující řetězec, se využívá i při analýze.

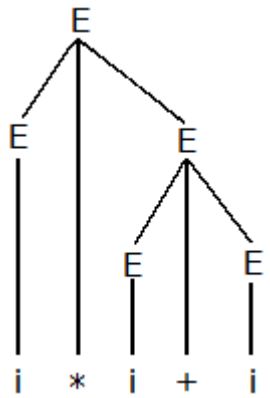
Příklad 1.: Máme-li řetězec uEv, přičemž $u, v \in (N \cup T)^*$ a $p = E \to i \in P$, pak lze matematicky zapsat $uEv \Rightarrow uiv[p]$, neboli slovně uEv přímo derivuje uiv za použití p v G – tzv. derivační krok.

Vhodným nástrojem pro názorné zobrazení dané derivace je tzv. derivační strom. Jedná se o orientovaný graf. Je však nutné brát v potaz nejednoznačnost bezkontextové gramatiky, která nastává v okamžiku, kdy pro řetězec z daného jazyka $L(G)$ existuje více jak jeden derivační strom. Nyní si uvedeme jednoduchý příklad pro ukázku derivačního stromu.

Příklad 2.: Mějme $G = (N, T, P, \mathrm{E})$, kde

$$N = \{\mathrm{E}\},\ \mathrm{T} = \{\mathrm{i}, +, *\},\ \mathrm{P} = \{1: E \to E * E, 2: E \to E + E, 3: E \to i\}$$

a) Derivační strom:

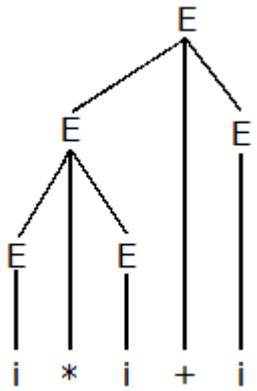


Derivace:

$$\underline{E} \Rightarrow E * \underline{E}$$
$$\Rightarrow E * E + \underline{E}$$
$$\Rightarrow E * \underline{E} * i$$
$$\Rightarrow \underline{E} * i + i$$
$$\Rightarrow i * i + i$$

Problém nastává v okamžiku, kdy bychom zaměnili pořadí pravidel, např.:

b) Derivační strom:



Derivace:

$$\underline{E} \Rightarrow \underline{E} + E$$
$$\Rightarrow \underline{E} + E * E$$
$$\Rightarrow i + \underline{E} * E$$
$$\Rightarrow i + i * \underline{E}$$
$$\Rightarrow i + i * i$$

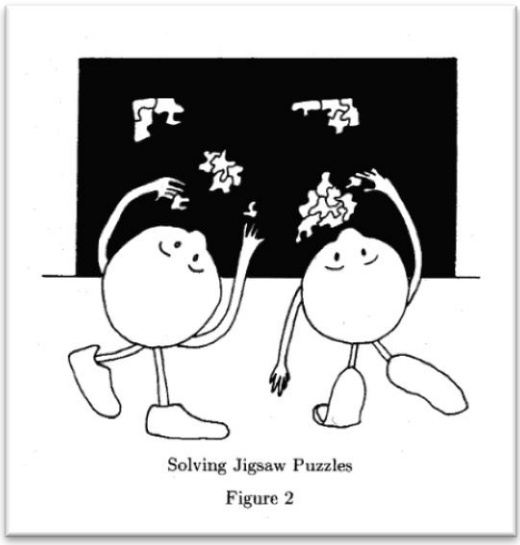
Dostaneme dva různé derivační stromy pro stejnou gramatiku, takže vzniká problém nejednoznačnosti, nehledě na matematické pozadí týkající se např. pořadí vyhodnocení operátorů, jak lze vidět na příkladu 2b. První strom se vygeneroval matematicky správně, druhý však nikoli. Generování vždy jednoho derivačního stromu pro stejnou posloupnost pravidel lze řešit tzv. kanonickými derivacemi, tj. zavedením vhodných omezení jednotlivých derivačních kroků. Existují dva typy kanonických derivací, pravá a levá neboli také nejpravější a nejlevější derivace. Vznikly ustanovením jednoznačného pravidla, kdy při pravé derivaci nahrazujeme v každém derivačním kroku vždy nejpravější neterminál. U levé derivace je tomu analogicky opačně, nahrazujeme vždy nejlevější neterminál v každém kroku derivace. Když se vrátíme zpět k příkladu 2 s derivačním stromem, můžeme si všimnout, že u prvního byla použita pravá derivace a u druhého levá, takže při opakování celé sekvence derivací vznikne vždy stejný strom. Toto však neřeší celý problém nejednoznačnosti gramatiky, ten je třeba řešit např. použitím vhodného deterministického zásobníkového automatu. Teorii automatů a jejich souvislosti s gramatikami v této práci hlouběji rozebírat nebudu. Více se o nich ještě zmíním v souvislosti s postupy syntaktické analýzy v kapitole 4. Pro podrobnější seznámení s touto problematikou doporučuji publikaci Regulated Grammars and Automata [1].

3 Gramatické systémy

Ačkoli se gramatické systémy jeví jako nová záležitost, rozvoj tohoto mechanismu probíhá již delší dobu a jedny z prvních myšlenek byly publikovány již v roce 1978 [6] v souvislosti se spolupracujícími gramatikami. Byly definovány první kooperující gramatické systémy. Autoři řešili problematiku dvouúrovňových gramatik ve spojitosti se syntaktickou analýzou a položili základ systémů vzájemně spolupracujících gramatik. V následujících dvou podkapitolách nastíním základy obou typů gramatických systémů jak kooperujících distribuovaných, tak paralelních.

3.1 CD gramatické systémy

Kooperující distribuované gramatické systémy pracují sekvenčně na jedné sdílené větné formě. Představit si tento model můžeme jako skupinu hráčů sedících u stolní hry. Každý hráč reprezentuje jednu gramatiku resp. konečnou množinu bezkontextových pravidel (tzv. komponentu systému). Jednotliví hráči se střídají a dle pravidel hry může každý z nich provést určité množství tahů, změn či jiných zásahů do hry ať už omezených např. počtem nebo dokud může zásahy provádět. Poté předá aktivitu dalšímu na řadě. V publikacích lze nalézt mnoho obecných příkladů. Jedním z nich je týmová práce agentů na společném problému, z nichž každý řeší jeden subproblém [7, s. 156]. Dále je velmi často uváděna analogie problému tabule (anglicky The Blackboard Model nebo The Blackboard System), kdy opět více lidí řeší společný problém. V tomto případě se střídají u tabule. Podrobnější informace lze nalézt v článku [8], odkud jsem si před definicemi a příklady dovolil převzít velmi výstižný obrázek.



Obrázek 1: Skupinové skládání puzzle - analogie s CD gramatickými systémy [7, s. 40]

3.1.1 Definice CD gramatického systému

Bezkontextové gramatiky jsou definovány v předchozí kapitole, takže budeme vycházet z těchto znalostí. CD gramatický systém stupně n, $n \geq 1$ je n-tice

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

- N – abeceda neterminálů
- T – abeceda terminálů, platí $N \cap T = \emptyset$
- S – startující symbol, $S \in N$
- $P_1 \dots P_n$ – konečná množina bezkontextových pravidel tvaru $a \to x,\ a \in N,\ x \in (N \cup T)^*$ (tzv. komponenty systému)

Jiná forma zápisu pomocí jednotlivých gramatik:

$$\Gamma = G_1, \dots, G_n$$
$$G_i = (N, T, P_i, S) \text{ pro } 1 \leq i \leq n$$

U tohoto typu zápisu většinou požadujeme, aby abecedy *N*, *T* a startující symboly *S* byly u všech gramatik stejné a jednotlivé gramatiky se lišily pouze množinami pravidel *P*.

3.1.2 Definice derivačních módů

CD gramatické systémy rozlišují dva základní derivační módy, které určují kdy, resp. po kolika derivačních krocích předá aktivní komponenta řízení další komponentě. Prvním módem je tzv. ukončovací mód (v literatuře Terminating Mode), ve kterém každá komponenta provádí přímé derivační kroky tak dlouho, dokud může uplatnit některé z pravidel a přepisovat větnou formu. V okamžiku, kdy už dále pracovat nemůže, přepne na další komponentu. Zde se dostáváme k analogiím z úvodu této kapitoly, kdy každá z komponent přispívá k řešení daného problému, dokud může. Druhý mód lze označit za skupinu pravidel, která definují, kolik derivačních kroků *k* provede každá komponenta, než přepne na další. V literatuře se setkáváme s výstižným názvem K-Step Mode neboli k-krokový mód. Jednotlivá omezení pro *k* jsou právě *k* kroků $(= k)$, maximálně *k* kroků $(\leq k)$ a alespoň *k* kroků $(\geq k)$. Dále se můžeme setkat ještě se třetím módem, značí se $*$, a každá komponenta v tomto módu pracuje, dokud potřebuje. V následujících odstavcích nalezneme podrobnější definice jednotlivých módů.

Terminating Mode (ukončující mód)

$x_i \Longrightarrow^t y$, *x* derivuje *y* v i-té komponentě v t-módu, právě když:

1. *x* derivuje *y* v G_i ($x \Longrightarrow^* y$ v $G_i = (N, T, P_i, S)$) a
2. *y* nederivuje/neexistuje přímá derivace *y* pro jakékoliv *z* nad abecedou $N \cup T$
   $y \nRightarrow z\ \forall\ z \in (N \cup T)^*$

Provádíme přímé derivační kroky, dokud se nedostaneme k řetězci terminálů a nemůžeme pokračovat dále, v tomto okamžiku skončíme.

K-Step Mode

1. k-step derivace $x_i \Longrightarrow^{=k} y$ pokud $x \Longrightarrow^{k} y$ v G_i
2. nejvýše k-step derivace $x_i \Longrightarrow^{\leq k} y$ pokud $x \Longrightarrow^{j} y$ v G_i pro $j \leq k$
3. alespoň k-step derivace $x_i \Longrightarrow^{\geq k} y$ pokud $x \Longrightarrow^{j} y$ v G_i pro $j \geq k$

3.1.3 Definice generovaného jazyka

Derivační módy společně tvoří množinu derivačních módů

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

a spolu s gramatikou množinu možných derivací

$$F(G_j, u, f) = \{v : u_j \Longrightarrow^{f} v\}, \text{ kde } j \in \{1, \dots, n\}, f \in D, u \in V^*.$$

V je množina všech řetězců takových, že z řetězce *u* f-kovým módem v j-té komponentě provedeme derivaci z *u* do *v*.

Nyní spojíme jednotlivé definice dohromady a pro CD gramatický systém $\Gamma = (N, T, S, P_1, \dots, P_n)$ v daném derivačním módu $f \in D$ je definice generovaného jazyka následující:

$$L_f(\Gamma) = \{w \in T^* : \text{kde jsou } v_0, v_1, \dots, v_n \text{ tak, že}$$
$$v_i \in F(G_{j_i}, v_{-1}, f),\ i = 1, .., m,\ j_i \in \{1, \dots, n\},\ v_0 = S,\ v_m = w \text{ pro } m \geq 1\}.$$

3.1.4 Příklady

Příklad 1.: $L_t(\Gamma) =$? pro CD gramatický systém $\Gamma = (\{S, A\}, \{a\}, S, P_1, P_2, P_3)$ s komponentami:

$$P_1 = \{S \to AA\}$$
$$P_2 = \{A \to S\}$$
$$P_3 = \{A \to a\}$$

První a jediná komponenta, která může zahájit činnost, je P_1. Provede derivační krok z S do AA a musí předat řízení další komponentě, protože již nemá pravidlo, které by mohla aplikovat.

$$S \Rightarrow_{P_1} AA$$

Pokud by řízení získala komponenta P_3, získali bychom rovnou řetězec terminálů a činnost systému by skončila. Necháme tedy pro názornější ukázku pracovat komponentu P_2, která přepíše jedno A na S, a protože může přepsat i druhé A a jsme v ukončovacím módu, tak musí. Poté může pracovat komponenta P_1.

$$AA \Rightarrow_{P_2} SA \Rightarrow_{P_2} SS \Rightarrow_{P_1} AAAA$$

Ta následně může předat řízení komponentám P_2 i P_3, jejíž činnost si pro ukázku ukážeme.

$$AAAA \Rightarrow_{P_3} aAAA \Rightarrow_{P_3} aaAA \Rightarrow_{P_3} aaaA \Rightarrow_{P_3} aaaa$$

Ukončením činnosti komponenty P_3 končí i práce celého systému – získali jsme výsledný řetězec terminálů. Obecně tedy můžeme říct, že podle střídání komponent P_1 a P_2 bude jazyk generovaný tímto systémem:

$$\underline{L_t(\Gamma) = \{a^{2^n} : n \geq 1\}}$$

Příklad 2.: $L_{=2}(\Gamma) =?$ CD gramatický systém $\Gamma = (\{S, A, A', B, B'\}, \{a, b, c\}, S, P_1, P_2)$
s komponentami: $P_1 = \{S \to S,\ S \to AB,\ A' \to A,\ B' \to B\}$
$P_2 = \{A \to aA'b,\ B \to cB',\ A \to ab,\ B \to c\}$

Nyní se nacházíme v k-step módu. Jednotlivé derivace, až na podmínku počtu kroků, jsou analogické s příkladem 1, proto postup stručně nastíním hned na začátku. Začít může opět pouze P_1, u které se první pravidlo z S do S může zdát zbytečné, ale umožňuje nám splnění omezení $k = 2$. Poté předá řízení komponentě P_2, kde máme na výběr z více pravidel, ale aby existovalo řešení pro tento systém, musíme použít vždy dvojici terminálních nebo neterminálních pravidel. Pokud bychom tak neudělali, neexistovalo by následně řešení pro komponentu P_1. Jako výsledek dostaneme opět řetězec terminálů.

$$S \Rightarrow_{P_1} S \Rightarrow_{P_1} AB \Rightarrow_{P_2} aA'bB \Rightarrow_{P_2} aA'bcB' \Rightarrow_{P_1} aAbcB' \Rightarrow_{P_1} aAbcB \Rightarrow_{P_2}$$
$$\Rightarrow_{P_2} aaA'bbcB \Rightarrow_{P_2} aaA'bbccB' \Rightarrow_{P_1} aaAbbccB' \Rightarrow_{P_1} aaAbbccB \Rightarrow_{P_2} \ldots$$
$$\Rightarrow_{P_1} a^n Ab^n c^n B \Rightarrow_{P_2} a^{n+1} b^{n+1} c^n B \Rightarrow_{P_2} a^{n+1} b^{n+1} c^{n+1}$$

$$\underline{L_{=2}(\Gamma) = \{a^n b^n c^n : n \geq 1\}}$$

Příklad 3.: $L_{=3}(\Gamma) =?$ pro CD gramatický systém z příkladu 2.

Tento příklad jsem zvolil úmyslně pro ukázku, kdy v k-step módu, v tomto případě $k = 3$, ale platilo by totéž i pro $k \geq 3$, neexistuje kvůli podmínce řešení. Opět by sekvence derivací započala komponentou P_1, která jako jediná má pravidlo pro přepsání starujícího symbolu S, avšak po vyčerpání prvních dvou pravidel již nemá další pravidlo, které by mohla uplatnit a k-podmínku nelze splnit.

$$S \Rightarrow_{P_1} S \Rightarrow_{P_1} AB \Rightarrow_{P_1} neexistuje$$

$$\underline{L_{=3}(\Gamma) = L_{\geq 3}(\Gamma) = \emptyset}$$

3.1.5 Přehled tříd jazyků a generativní síla

Pro popis jednotlivých tříd jazyků generovaných CD gramatickými systémy používáme následující značení:

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

- f – derivační mód, $f \in D$
- y – chybí, pak nepovolujeme ε-pravidla
  – ε, pak jsou ε-pravidla povolena
- x – n, nejvyšší stupeň CD gramatického systému, $n \geq 1$
  – ∞, počet komponent systému není omezen

$CD_\infty(=)$ sjednocení všech tříd jazyků $CD_\infty(= k)$ pro $k = 1,2, \ldots$
$CD_\infty(\geq)$ sjednocení všech tříd jazyků $CD_\infty(\geq k)$ pro $k = 1,2, \ldots$

Na základě takto zavedeného popisu jednotlivých tříd můžeme určit generativní sílu CD gramatických systémů:

1. $CD_\infty^y(f) = \mathscr{L}(CF)$, pro všechny $f \in \{= 1, \geq 1, *\} \cup \{\leq k : k \geq 1\}$
2. $\mathscr{L}(CF) = CD_1^y(f) \subset CD_2^y(f) \subseteq CD_r^y(f) \subseteq CD_\infty^y(f) \subseteq \mathscr{L}(M)$,
   pro všechny $f \in \{= k, \geq k : k \geq 2\},\ r \geq 3$
3. $CD_r^y(\geq k) \subseteq CD_r^y(\geq k+1)$
4. $CD_\infty^y(\geq) \subseteq CD_\infty^y(=)$
5. $\mathscr{L}(CF) = CD_1^y(t) = CD_2^y(t) \subset CD_3^y(t) = CD_\infty^y(t) = \mathscr{L}(ET0L)$

V prvním případě se neomezuje počet komponent, ε-pravidla jsou/nejsou připouštěna a jednotlivé komponenty mohou přepnout již po jednom derivačním kroku – získáváme generativní sílu bezkontextové gramatiky. Ve druhém případě při použití dvou a více krokových módů roste generativní síla se vzrůstajícím počtem komponent. Třetí resp. čtvrtá věta nám říká, že při stejném počtu komponent resp. nekonečném počtu je systém s alespoň k-krokovým módem podmnožinou systému s k+1-krokovým módem resp. k-krokovým módem. Poslední pátý případ popisuje ukončující mód, kdy systém s jednou nebo dvěma komponentami má generativní sílu bezkontextové gramatiky a od alespoň tří komponent je nadmnožinou rovnou ET0L jazykům. Další definice a teorémy lze nalézt např. v publikaci [7].

3.1.6 Hybridní CD gramatické systémy

Na závěr této kapitoly se zmíním pro úplnost o tzv. hybridních CD gramatických systémech, které jsou zajímavým rozšířením klasických CD gramatických systémů. Umožňují totiž definovat každé komponentě její vlastní derivační mód, takže můžeme např. explicitně nastavit, že některé komponenty budou pracovat právě k kroků, naopak jiné budou fungovat v ukončujícím módu a provádět derivační kroky tak dlouho, dokud to bude možné atd. Definice těchto systémů je následující:

$$\Gamma = (N, T, S, (P_1, f_1), \dots, (P_{n,}, f_n)), \text{ kde:}$$

- N – abeceda neterminálů
- T – abeceda terminálů, platí $N \cap T = \emptyset$
- S – startující symbol, $S \in N$
- P_i – komponenty systému, totožné s normálním CD gramatickým systémem
- f_i – mód i-té komponenty, $f_i \in D$ pro všechna $i \in \{1, \dots, n\}$

Jazyk generovaný tímto rozšířeným CD gramatickým systémem lze definovat:

$L_f(\Gamma) = \{w \in T^*$: kde jsou $v_0, v_1, \dots, v_n$ tak, že
$v_i \in F(G_{j_i}, v_{-1}, f_j),\ i = 1, .., m,\ j_i \in \{1, \dots, n\},\ v_0 = S,\ v_m = w$ pro nějaké $m \geq 1\}$.

3.1.7 Přehled tříd jazyků a generativní síla hybridních CD gramatických systémů

Stejně jak u normálních, tak i u hybridních CD gramatických systémů lze zavést definici tříd generovaných jazyků:

$$XCD^{y}_{x,v}(f), \text{ kde}$$

- f – derivační mód, $f \in D$
- y – chybí, pak nepovolujeme ε-pravidla
  – ε, pak jsou ε-pravidla povolena
- x – n, nejvyšší stupeň CD gramatického systému, $n \geq 1$
  – ∞, počet komponent systému není omezen
- v – m, každá komponenta P_i obsahuje maximálně m pravidel, $m > 1$
  – ∞ nebo nic, počet pravidel není omezen
- X – nic, nedeterministický systém
  – D, deterministický (pro každé pravidlo z P_i platí, že pro každý neterminál A existuje nejvýše jedna pravá strana; $A \to u, A \to w \in P_i$, pak $u = w$)
  – H, hybridní systém (nepíšeme (f), protože neexistuje globální mód)

Následující teorémy ukazují generativní sílu hybridních CD gramatických systémů:

1. $CD_{\infty,\infty}(f) = CD_{\infty,1}(f) = \mathscr{L}(CF)$, pro všechny $f \in \{=1, \geq 1, *\} \cup \{\leq k : k \geq 1\}$
2. $\mathscr{L}(CF) \subset CD^{\varepsilon}_{\infty,1}(t) \subset CD^{\varepsilon}_{\infty,2}(t) \subseteq CD^{\varepsilon}_{\infty,3}(t) \subseteq CD^{\varepsilon}_{\infty,4}(t) \subseteq CD^{\varepsilon}_{\infty,5}(t) = CD^{\varepsilon}_{\infty,\infty}(t) = = \mathscr{L}(ETOL)$
3. $CD_{n,m}(f) \subset CD_{n+1,m}(f),\ f \in \{*, t\}$
4. $CD_{n,m}(f) \subset CD_{n,m+1}(f),\ f \in \{*, t\}$
5. $\mathscr{L}(CF) = HCD_1 \subset HCD_2 \subseteq HCD_3 \subseteq HCD_4 = HCD_\infty = \mathscr{L}(M)$
6. $\mathscr{L}(ETOL) \subset HCD_4$
7. $CD_\infty(=) \subset HCD_3$

Obdobně jako tomu bylo u CD gramatických systémů, i v tomto případě dostáváme hierarchie generativních sil pro jednotlivé třídy jazyků, kde opět generativní síla roste s počtem komponent, popř. se zvyšujícím se počtem dovolených pravidel pro každou z komponent. První čtyři teorémy vycházejí ze stejných předpokladů jako u normálních CD gramatických systémů, pouze zavádějí omezení na počet pravidel v každé komponentě. Za zmínku stojí až teorémy pro samotné hybridní systémy, kde např. v pátém teorému začínáme od síly bezkontextových jazyků a se zvyšujícím se počtem komponent dostaneme od čtvrté komponenty sílu maticových jazyků. V šestém teorému pak zjišťujeme, že třída ET0L jazyků je vlastní podmnožinou hybridních CD gramatických systémů se čtyřmi komponentami. Veškeré podrobnější informace lze opět nalézt v publikaci [7].

3.2 PC gramatické systémy

Velkou výhodou PC gramatického systému oproti předchozí CD variantě je, že všechny komponenty systému mohou pracovat souběžně a přepisovat tak své řetězce. Opět tedy můžeme zmínit analogii blackboard nebo např. práci agentů na společném problému. V tomto případě však může každý zúčastněný řešit svůj vlastní subproblém souběžně s ostatními, mezivýsledky si předávat mezi sebou a výsledné řešení vznikne složením dílčích částí. Pro lepší představu lze uvést např. vývoj softwaru, který je složen z jednotlivých modulů/pluginů, které jsou na sobě do jisté míry nezávislé a každý vývojářský tým implementuje jeden modul. Poznatky a informace si týmy mezi sebou předávají a výsledná aplikace je pak tvořena spojením všech částí. Stejně tak komponenty PC gramatického systému mezi sebou navzájem komunikují pomocí tzv. komunikačních symbolů.

Další velké uplatnění dovoluje schopnost, kdy si jedna komponenta může vyžádat práci jiné, čehož se využívá např. u překladu jazyků, kdy je zdrojový kód tvořen více programovacími jazyky nebo je pro jednotlivé části kódu vhodnější využít jinou gramatiku. Tento mechanismus však v některých případech může s sebou přinášet riziko uváznutí (deadlock), kterému je potřeba předcházet.

3.2.1 Definice PC gramatického systému

Podobně jako u CD gramatického systému lze PC gramatický systém stupně n, $n \geq 1$ definovat jako n-tici:

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

- N – abeceda neterminálů
- K – konečná množina komunikačních symbolů, $K \in \{Q_1, \dots, Q_n\}$
- T – abeceda terminálů
- S_i – startující symbol i-té komponenty, $S_i \in N$ pro $i = 1, .., n$
- P_i – konečná množina pravidel tvaru $A \to x$, kde $A \in N$ a $x \in (N \cup T \cup K)^*$ pro $i = 1, \dots, n$
- Dále platí, že $K \notin N, T$

3.2.2 Definice derivačních kroků

U PC gramatického systému opět rozlišujeme dva druhy derivačních kroků, ale zatímco u CD gramatického systému se módy lišily pouze podmínkou na počet kroků, zde má každý krok svůj vlastní účel. Generující krok (g-Step) slouží k provádění přímých generativních kroků z $(x_1, \dots, x_n)$ do $(y_1, \dots, y_n)$, neboli x přímo derivuje y, což je totožné s derivačními kroky CD gramatických systémů. N-tice $(x_1, \dots, x_n)$ a $(y_1, \dots, y_n)$, kde $x_i, y_i \in V_\Gamma^+$ a pro všechna $i: 1 \leq i \leq n$, se nazývají konfigurace. Naopak komunikační krok (c-Step) se provede v okamžiku, kdy se někde v konfiguraci komponenty (zpracovávaném řetězci) vygeneruje komunikační symbol. Dojde k přepsání tohoto symbolu odpovídajícím řetezcem z komponenty, na kterou symbol odkazuje. Ta však nesmí obsahovat žádné komunikační symboly. Komunikační krok má vždy přednost před generativním krokem.

g-Step

Pokud $x_i \Rightarrow y_i$ v $G_i = (N \cup K, T, P_i, S_i)$ nebo
$x_i = y_i \in T^*$ pro všechny $1 \leq i \leq n$, pak

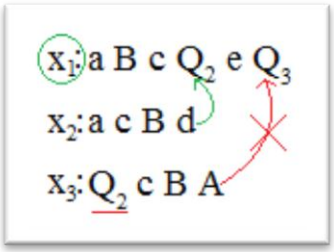
$$(x_1, \ldots, x_n)_g \Rightarrow (y_1, \ldots, y_n)$$

c-Step

Nastav $z_i = x_i$ pro všechny $i = 1, \ldots, n$
Pro každé $i = 1, \ldots, n$ pokud
abeceda$(x_i) \cap K \neq \emptyset$
a pro každé Q_j v x_i
abeceda$(x_j) \cap K = \emptyset$
potom pro každé Q_j v x_i

1. Nastav $z_j = S_j$
2. Nahraď Q_j za x_j v x_i
3. Nastav do z_i výsledek z kroku 2

Proveď
$(x_1, \ldots, x_n)_c \Rightarrow (y_1, \ldots, y_n)$
se $y_i = z_i$ pro všechny $i = 1, \ldots, n$



Obrázek 2: Příklad porušení podmínky výskytu K symbolů. Požadavek Q_3 nebude v tomto kroku uspokojen.

Matematický zápis generativního kroku je prostý a popisuje běžné derivační kroky z x_i do y_i pro každou z komponent. Může dojít ke dvěma situacím. Buďto se provede klasická derivace nebo jsou x_i a y_i stejné (tvoří je řetězec terminálů) a nedojde v tomto kroku k žádné změně.

Komunikační krok je výrazně složitější. Nejdříve se do pomocných proměnných z_i uloží všechna x_i, která představují větné formy (rozgenerovávané řetězce) jednotlivých komponent systému. Následně se ověřuje pro každou komponentu x_i, jestli neobsahuje komunikační symboly. Pokud obsahuje komunikační symbol Q_j (může jich být více), musí platit druhá podmínka, že komponenta x_j žádný komunikační symbol neobsahuje. Komponenta x_j je komponenta, na kterou odkazuje komunikační symbol Q_j, a protože komunikační symbol generuje požadavek na její vkopírování, nesmí sama obsahovat žádné komunikační symboly. Jestliže platí obě podmínky, x_i obsahuje alespoň jeden komunikační symbol a komponenty, na které odkazuje, žádný komunikační symbol neobsahují, pak se pro každý komunikační symbol v x_i provedou následující kroky. Nejdříve se nastaví pomocná proměnná z_j na startující symbol S_j. Toto platí pro PC gramatický systém s návraty, protože tento krok zajistí reset komponenty, kterou kopírujeme. Dále nahradíme komunikační symbol Q_j v komponentě x_i komponentou x_j, tedy komponentou, na kterou symbol odkazoval. Poté do pomocné proměnné z_i nastavíme výsledek z předchozího kroku, tedy aktuální x_i. Na závěr přepíšeme všechny y_i na z_i, čímž provedeme derivační kroky pro všechny komponenty. Tento postup nám zajistí, že pokud x_i neobsahovalo žádné komunikační symboly a zároveň na něj neodkazoval žádný komunikační symbol, pak zůstalo z_i od začátku algoritmu stejné a nedojde ke změně, tedy $x_i = y_i$. Naopak pokud x_i obsahovalo komunikační symboly, potom došlo k jejich nahrazení, pakliže to bylo v aktuálním kroku možné. Pokud by totiž odkazovaná komponenta obsahovala také komunikační symboly, tak by se v tomto kroku kopírování neprovádělo a k pokusu o zpracování by došlo v další iteraci. Dále pak z_i zajišťuje reset komponenty. Tento krok algoritmu je však nutné odstranit u PC gramatického systému bez návratů.

3.2.3 Definice generovaného jazyka

Pokud se provádí buďto jeden generativní krok z konfigurace $(x_1, \dots, x_n)$ do konfigurace $(y_1, \dots, y_n)$, zapsáno $(x_1, \dots, x_n)_g \Rightarrow (y_1, \dots, y_n)$, nebo $(x_1, \dots, x_n)$ provádí přímý komunikační krok do $(y_1, \dots, y_n)$, zapsáno $(x_1, \dots, x_n)_c \Rightarrow (y_1, \dots, y_n)$, pak píšeme, že konfigurace $(x_1, \dots, x_n)$ provádí přímý derivační krok do konfigurace $(y_1, \dots, y_n)$, $(x_1, \dots, x_n) \Rightarrow (y_1, \dots, y_n)$. Potom jazyk generovaný PC gramatickým systémem je množina všech řetězců terminálů takových, že vycházíme z počáteční konfigurace $(S_1, \dots, S_n)$, která derivuje konfiguraci $(x, \alpha_1, \dots, \alpha_n)$, přičemž α je libovolný řetězec. Z toho plyne, že výsledek činnosti PC gramatického systému generuje první komponenta a jakmile v ní vznikne řetězec terminálů, je členem jazyka tohoto systému.

$$L(\Gamma) = \{x \in T^*: (S_1, \dots, S_n) \Rightarrow^* (x, \alpha_1, \dots, \alpha_n),\ \alpha_i \in (N \cup T \cup K)^*, \text{ pro všechny } i = 2, \dots, n\}$$

3.2.4 Centralizovaný PC gramatický systém

Centralizovaný PC gramatický systém řeší problém uváznutí, protože pouze první komponenta P_1 může generovat komunikační symboly, tedy si vyžádat vkopírování jiné komponenty.

Nechť $\Gamma = (N, K, T, (S_1, P_1), \dots, (S_n, P_n))$ je PC gramatický systém.
Γ je centalizovaný, pokud pro všechny $A \rightarrow x \in P_i$, kde $i = 2, \dots, n$
abeceda$(x_j) \cap K = \emptyset$

3.2.5 PC gramatický systém s návraty a bez návratů

PC gramatický systém s návraty po vkopírování vyžádané komponenty nastaví její obsah zpět na startující symbol S_i. Jinak řečeno neprovádí kopírování komponenty, ale její vyjmutí a nahrazení startujícím symbolem. V tomto případě může nastat např. situace, kdy v první komponentě x je již řetězec terminálů, ale generování nekončí, nejedná se o centralizovaný systém a jiná komponenta si vyžádá vkopírování této první komponenty. Ta je poté nahrazena startujícím symbolem a její generování začíná od začátku. Jazyk generovaný tímto systémem se značí $L_r(\Gamma)$.

Naopak PC gramatický systém bez návratů nepřepisuje vkopírovanou komponentu startujícím symbolem, ale pokračuje rozgenerovávání stávajícího řetězce. Je nutné však v definici komunikačního kroku zrušit bod „1. Nastav $z_j = S_j$“, který zajišťuje návraty. Jazyk generovaný tímto systémem se značí $L_{nr}(\Gamma)$.

3.2.6 Příklad

Příklad 1.: $L_r(\Gamma) =$? pro PC gramatický systém

$\Gamma = (\{S_1, S_1', S_2, S_3\},\ K,\ \{a, b, c\},\ (S_1, P_1),\ (S_2, P_2),\ (S_3, P_3)\}$ s komponentami:

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

$P_2 = \{\ S_2 \to b\ S_2\}$

$P_3 = \{\ S_3 \to c\ S_3\}$

Zadání definuje, že se jedná o PC gramatický systém s návraty, a protože komunikační symboly generuje pouze komponenta P_1, je tento systém i centralizovaný. S rozgenerováváním začneme obdobně jako u CD gramatických systémů, pouze s tím rozdílem, že každá komponenta přepisuje pouze svůj řetězec. U komponent P_2 a P_3 nemáme na výběr, musíme použít v každém kroku pouze to jedno pravidlo, které obsahují. V komponentě P_1 máme při startu na výběr hned ze tří pravidel, ale pokud bychom použili první nebo druhé, generování by skončilo hned v prvním kroku, zvolíme tedy třetí pravidlo.

$$(S_1, S_2, S_3) \Rightarrow (aS_1', b\ S_2, c\ S_3)$$

Stále setrváváme v generujícím kroku, protože se nevygeneroval žádný komunikační symbol. Nyní můžeme opakovat n-krát pravidlo pět z první komponenty, u ostatních nemáme na výběr, a dostaneme tak n-té mocniny pro všechny terminály.

$$(aS_1', b\ S_2, c\ S_3) \Rightarrow (a^nS_1', b^n\ S_2, c^n\ S_3)$$

Nyní již nemá smysl pokračovat v pravidle pět a musíme vybrat pravidlo šest. Tím vygenerujeme v první komponentě komunikační symbol a musíme provést komunikační krok, ve kterém se přepíše symbol Q_2 komponentou P_2, a protože jsme v režimu s návraty, P_2 se resetuje.

$$(a^nS_1', b^n\ S_2, c^n\ S_3) \Rightarrow (a^{n+3}Q_2, b^{n+1}\ S_2, c^{n+1}\ S_3) \Rightarrow (a^{n+3}b^{n+1}\ S_2,\ S_2, c^{n+1}\ S_3)$$

Další komunikační symbol v konfiguraci již není, takže pokračujeme generujícím krokem. V první komponentě můžeme použít pouze pravidlo sedm, které nám vygeneruje komunikační symbol Q_3 a vše se provede analogicky s předchozím krokem.

$$(a^{n+3}b^{n+1}\ S_2,\ S_2, c^{n+1}\ S_3) \Rightarrow (a^{n+3}b^{n+3}Q_3,\ bS_2, c^{n+2}\ S_3) \Rightarrow (a^{n+3}b^{n+3}c^{n+2}\ S_3,\ bS_2,\ S_3)$$

Na závěr můžeme provést v P_1 pouze poslední pravidlo a generování tím končí.

$$(a^{n+3}b^{n+3}c^{n+2}\ S_3,\ bS_2,\ S_3) \Rightarrow (a^{n+3}b^{n+3}c^{n+3},\ b^2S_2, c\ S_3)$$

Získali jsme v první komponentě řetězec terminálů, který musíme ještě doplnit o výsledky prvního a druhého pravidla téže komponenty, protože těmi jsme mohli celé generování již na začátku ukončit. Výsledný jazyk generovaný tímto systémem je:

$$\underline{L_r(\Gamma) = \{a^nb^nc^n : n \geq 1\}}$$

3.2.7 Přehled tříd jazyků a generativní síla

Stejně jako u CD gramatických systémů máme i pro PC gramatické systémy popis jednotlivých tříd jazyků:

$$XPC_nY, \text{ kde}$$

- $X - N$, systém bez návratů (non-returning mode)
  $- C$, centralizovaný PC gramatický systém
- n – počet komponent systému, $n \geq 1$
- Y – typ pravidel, které smíme použít (REG, LIN, CF)

Následně můžeme určit generativní sílu jednotlivých PC gramatických systémů:

1. $PC_nREG \subset PC_{n+1}REG$ pro $n > 1$
2. $CPC_nREG \subset CPC_nLIN \subset CPC_nCF$ pro $n > 1$
3. $NPC_\infty CF \subset PC_\infty CF$
4. $\mathscr{L}(\mathrm{M}) \subset PC_\infty CF$
5. $\mathscr{L}(\mathrm{LIN}) \subset PC_\infty REG$

Ve stručnosti si jednotlivé hiearchie vysvětlíme. Teorém jedna nám říká, že pokud použijeme PC gramatický systém pouze s regulárními pravidly, pak čím více komponent máme, tím větší třídu jazyků dostaneme. Druhý teorém popisuje vztah mezi použitím různě silných typů pravidel u centralizovaného PC gramatického systému s n komponentami, kdy dle předpokladu nejmenší generativní sílu mají regulární pravidla, silnější jsou lineární a nejsilnější bezkontextová. Zbylé tři případy popisují vztah při neomezeném počtu komponent. PC gramatický systém bez návratů s bezkontextovými pravidly je vlastní podmnožinou stejného systému, ale s návraty. Stejně tak třída maticových jazyků je vlastní podmnožinou PC gramatického systému s návraty s bezkontextovými pravidly a na závěr třída lineárních jazyků je vlastní podmnožinou PC gramatického systému s návraty a regulárními pravidly. Podrobnější informace s dalšími teorémy a důkazy lze nalézt opět v publikaci [7].

4 Syntaktická analýza

Syntaktická analýza by se dala nazvat jádrem kompilátorů a tvoří zpravidla druhou fázi překladu. Jejím vstupem je proud tokenů, který je výstupem předcházející lexikální analýzy. V této kapitole budeme předpokládat alespoň základní znalosti či povědomí o činnosti lexikálního analyzátoru. Pouze pro ujasnění uvedu, že lexikální analýza je první fází překladu. Zpracovává zdrojový řetězec znaků (program/kód) a převádí jej na řetězec lexikálních symbolů neboli tokenů, se kterými již pracuje syntaktická analýza. Jsou dva přístupy, buď lexikální analyzátor zpracuje celý vstupní text a předá syntaktické analýze kompletní řetězec tokenů, nebo pracuje jako podprogram a na vyžádání předá syntaktickému analyzátoru vždy další token. Dále může ze vstupního textu odstraňovat např. komentáře a provádí kontrolu nejzákladnějších chyb, jako může být např. chybný zápis identifikátorů dle konvencí daného jazyka, chybně zapsané číslo atd.

Syntaktická analýza má tedy za úkol zkontrolovat syntaktickou strukturu vstupního programu. Jinými slovy syntaktický analyzátor sestrojí z tokenů na vstupu derivační strom. Z pochopitelných důvodů při konstrukci stromů požadujeme jednoznačnost, což nás vrací k úvodu o bezkontextových gramatikách, kde jsme si uvedli, že se využívají dva principy, a to pravá a levá derivace, ze kterých vznikne vždy stejný derivační strom. Pak tedy v praxi rozlišujeme dva základní přístupy, syntaktickou analýzu shora dolů a zdola nahoru.

4.1 Shora dolů

Předpokládejme, že máme klasickou bezkontextovou gramatiku, kterou jsme si definovali v kapitole 2. Jednotlivá pravidla z množiny P jsou číslována od čísla jedna. Pak při syntaktické analýze shora dolů získáme posloupnost čísel jednotlivých pravidel odpovídajících levé derivaci vstupního řetězce, v tomto případě řetězce tokenů. Provádíme tedy konstrukci derivačního stromu od kořene k listům. Tento proces lze nazývat levý rozbor a nejčastějšími metodami jsou syntaktická analýza pomocí LL tabulky nebo rekurzivní sestup. V obou případech se využívá tzv. LL gramatika, které bude věnována následující kapitola. Samozřejmě existují i další systémy a postupy, o kterých se lze dočíst např. v publikaci [3].

4.1.1 LL gramatiky

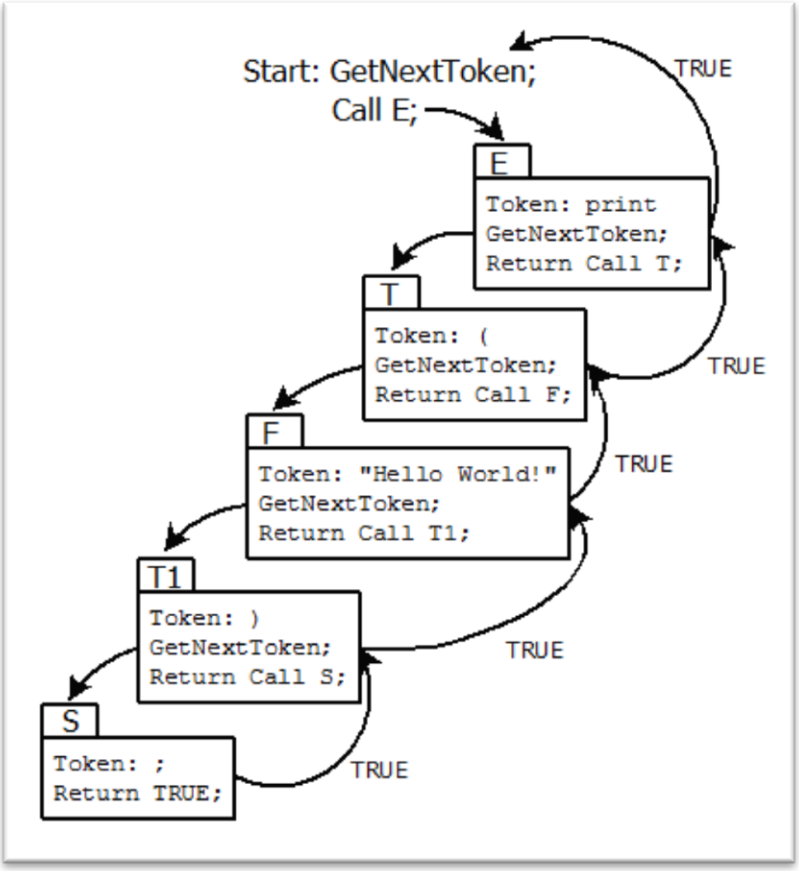
Jeden z důvodů zkoumání a vzniku LL gramatik byla potřeba aparátu, který by umožňoval vytvořit pro bezkontextové gramatiky deterministický syntaktický analyzátor, což byl problém při tvorbě deterministických zásobníkových automatů u některých bezkontextových jazyků. Vznikly tak LL gramatiky jako speciální typ bezkontextových gramatik, pro které lze vždy vytvořit deterministický syntaktický analyzátor shora dolů. Z následující definice LL gramatiky bez ε-pravidel bude vše pochopitelnější:

> Nechť $G = (N, T, P, S)$ je bezkontextová gramatika bez ε-pravidel. G je LL gramatika, pokud pro každé $a \in T$ a $A \in N$ existuje maximálně jedno pravidlo $A \to X_1 X_2 \dots X_n \in P$ takové, že $a \in First(X_1 X_2 \dots X_n)$. Pokud bychom zavedli ε-pravidla, pak by platilo, že $a \in Predict(A \to X_1 X_2 \dots X_n)$.

Pro pochopení je nutné obeznámit se s množinami $First$ a $Predict$. $First(x)$ definuje množinu všech terminálů, kterými může začínat řetězec derivovatelný z x, zatímco $Predict(A \rightarrow X)$ je o něco složitější. Jedná se o množinu všech terminálů, které mohou být aktuálně nejlevěji vygenerovány, pokud pro libovolnou větnou formu použijeme pravidlo $A \rightarrow X$. Pro její získání však potřebujeme další dvě množiny, a to $Empty$ a $Follow$. Množina $Empty$ souvisí úzce s ε-pravidly. $Empty(x)$ obsahuje pouze prvek ε, pokud x derivuje ε, nebo je prázdná. $Follow(A)$ je pak množina všech terminálů, které se mohou vyskytovat vpravo od A ve větné formě. Podrobné definice a teoretické pozadí lze nalézt v publikaci [3], stejně jako princip konstrukce LL tabulky.

4.1.2 Implementace

Při implementaci LL syntaktického analyzátoru shora dolů lze využít dvou přístupů. Prediktivní syntaktickou analýzu, která využívá analyzátor se zásobníkem řízeným LL tabulkou, a tzv. rekurzivní sestup. Tuto metodu jsem využil i já v praktické části této práce a spočívá v přiřazení funkcí jednotlivým neterminálům. První neterminál, zpravidla začátek programu, po startu analýzy spustí sérii funkcí, které se podle jednotlivých pravidel vzájemně volají a při vyhodnocení vstupního tokenu vrátí logickou hodnotu $true$ nebo $false$. Při úspěchu se pokračuje v dalším provolávání funkcí a kontrole, zatímco při syntaktické chybě se $false$ vrátí přes všechny úrovně zanoření a syntaktická analýza skončí s chybou.



Obrázek 3: Ukázka rekurzivního sestupu v praxi pro vstupní řetězec: `print("Hello World!");`

4.2 Zdola nahoru

Syntaktická analýza zdola nahoru je analogicky opačná k analýze shora dolů. Opět chceme získat posloupnost aplikovaných pravidel, v tomto případě však provádíme redukci od listů ke kořeni a výsledná posloupnost pravidel je obrácená vůči metodě shora dolů. Jedná se tak o sérii pravých derivací, kterou nazýváme pravý rozbor.

Hlavními problémy, které musíme řešit, je jednak situace, kdy dvě nebo více pravidel mají totožnou pravou stranu. Pak nevíme, které z pravidel použít. Dalším problémem jsou nejednoznačné gramatiky, které by nám za normálních okolností umožňovaly sestrojit v některých případech pro stejné vstupní řetězce více různých derivačních stromů, což je samozřejmě při syntaktické analýze nepřípustné. Tento problém jsme si již názorně ukázali v kapitole 2.1 při konstrukci derivačních stromů u bezkontextových gramatik.

U syntaktické analýzy zdola nahoru rozlišujeme dva nejběžnější přístupy, a to precedenční syntaktický analyzátor a LR syntaktický analyzátor. Precedenční analyzátor je sice nejslabší, ale to nesporně vyvažuje jednoduchost jeho implementace, zatímco LR analyzátor je nejsilnější, ale implementace je složitá a vyžaduje konstrukci LR tabulky namísto jednoduché precedenční tabulky v předchozí variantě.

4.2.1 Precedenční syntaktická analýza

Při použití precedenční syntaktické analýzy musíme zajistit, aby v gramatice, předpokládejme opět bezkontextovou gramatiku, neexistovalo více pravidel se stejnou pravou stranou. Dále pak gramatika nesmí obsahovat ε-pravidla. Celá precedenční analýza je řízena jednoduchým algoritmem, který na vstupu přijímá precedenční tabulku pro danou bezkontextovou gramatiku G a řetězec terminálů x. Výstupem je potom pravý rozbor x, pokud $x \in L(G)$, jinak chyba. Precedenční tabulka nám určuje prioritu jednotlivých terminálů. Záhlaví tabulky obsahuje jednotlivé terminály na vstupu $a_j, \dots, a_n$ pro j= $\{1, \dots, n\}$, zatímco řádky odpovídají terminálům na vrcholu zásobníku $a_i, \dots, a_n$ pro i= $\{1, \dots, n\}$. Jednotlivá pole potom definují vztah mezi těmito dvěma terminály $[a_i, a_j] = \{<, =, > , nic\}$, které určují především asociativitu a precedenci operátorů. Algoritmus pracuje se zásobníkem a rozhoduje o dalším kroku na základě průniku terminálů, terminálem na vrcholu zásobníku s terminálem na vstupu, v precedenční tabulce. Princip celého algoritmu si více přiblížíme na praktickém příkladu v kapitole 5.5. Další podrobnější informace o fungování precedenční analýzy lze naleznout opět v publikaci [3].

4.2.2 LR syntaktická analýza

LR syntaktický analyzátor pracuje na podobném principu jako precedenční analyzátor. Opět funguje na základě jasně daného algoritmu a ke své činnosti využívá tabulku, v tomto případě však speciální LR tabulku, kterou tvoří dvě části, akční část neboli tabulka akcí a přechodová část neboli tabulka přechodů. LR analyzátor je založen na rozšířeném zásobníkovém automatu se stavy $Q = \{q_0, q_1, \dots, q_k)$, kde q_0 je počáteční stav. Vstupem je LR tabulka pro danou bezkontextovou gramatiku G a řetězec terminálů x. Výstupem je potom pravý rozbor x, pokud $x \in L(G)$, jinak chyba.

5 Aplikace

Praktická část této práce, tj. webová aplikace, má za úkol co nejnázorněji ukázat spolupráci více gramatik a předávání řízení mezi nimi. Zaměřil jsem se na princip PC gramatických systémů v centralizovaném režimu pro dvě spolupracující gramatiky, který uplatňuji v syntaktické analýze. Centrální gramatikou je tedy LL gramatika. Jejími pravidly se řídí celá syntaktická analýza a v případě potřeby si vyžádá práci druhé komponenty pro zpracování výrazů, pro jejichž analýzu je vhodnější využít precedenční analyzátor, než vše řešit např. jednou LL gramatikou. Jazyk přijímaný tímto systémem je podmnožinou programovacího jazyka C. V následujících podkapitolách postupně vysvětlím funkcionalitu celé aplikace od uživatelského rozhraní přes jednotlivé principy, až k samotnému způsobu implementace.

5.1 Použité nástroje

Aplikace spojuje více nástrojů dohromady. Celá struktura je zabalená do obálky php pro verzi 5.3.29, která je aktuálně nainstalovaná na školním serveru Eva, pod kterým proběhl celý vývoj. Serverová část aplikace implementovaná v php provádí vzdáleně veškeré výpočty spojené se syntaktickou analýzou vstupního textu, zatímco klientská část je implementovaná v html s použitím JavaScriptu a CSS3 pro dynamické zobrazení zpracovaných dat od php serveru. Dokončení vývoje a testování uživatelského rozhraní proběhlo na internetových prohlížečích Mozzila Firefox verze 37.0.1 a Google Chrome verze 42.0.2311.90m. Starší verze těchto prohlížečů zobrazovaly také vše v pořádku, takže kompatibilitu a správné zobrazení lze předpokládat i v dalších novějších verzích, které vývojáři uvolní, ale není již plně zaručena. Naopak nedoporučuji použití Internet Exploreru, obzvláště starých verzí.

5.2 Uživatelské rozhraní

Grafické uživatelské rozhraní této aplikace je zaměřeno jak na co největší jednoduchost a názornost, tak i na univerzálnost, a proto je v angličtině. Jedná se pouze o jednu stránku přístupnou zadáním URL cesty adresáře se zdrojovými soubory v internetovém prohlížeči, která má za úkol zobrazit uživateli veškeré potřebné informace v co nejjednodušší podobě. Po načtení stránky se uživateli zobrazí vstupní pole `textarea`, do kterého se automaticky nahraje příklad ze souboru `example.c`, který obsahuje ukázku základních konstrukcí, které dokáže aplikace analyzovat. Slouží tak uživateli jako názorná pomůcka. Dále se uživateli vypíše kompletní LL gramatika s očíslovanými pravidly, které jsou při analýze uplatňovány (soubor `LL-grammarC.txt`). Lze tak snadno pochopit, jaké konstrukce systém přijímá. Hned vedle je zobrazena i kompletní tabulka symbolů precedenční analýzy vč. jejích pravidel pro redukci (soubor `SymbolsTable.txt`). Tato textová pole není možné upravovat, ale lze text zkopírovat, což umožňuje uživateli snadné provedení vlastního návrhu za použití tohoto již stávajícího.

Vstupní pole s ukázkovým příkladem jde libovolně upravovat, takže lze spustit analýzu již nachystaného kódu nebo jej upravit či smazat a vložit vlastní program. Poté stačí stisknout tlačítko

Analyze. Vstupní text se předá php serveru, který provede analýzu, a výsledek vrátí zpět klientské části, která jej zobrazí uživateli v několika úrovních.

Pod vstupním polem se zobrazí nové okno, které obsahuje přepis analyzovaného kódu. Zobrazí jej s očíslovanými řádky, aby bylo možné přehledně odkazovat výstup analýzy k jednotlivým řádkům. Pro text je použito písmo typu monotype, konkrétně Courier New, který definuje všechny znaky vč. mezer stejně široké, což zajišťuje větší přehlednost kódu. Toto okno nelze upravovat, protože je svázáno s výstupem analýzy, ale umožňuje opět kopírování textu a jsou k němu přidružena dvě tlačítka. První tlačítko s popisem Show S/P/K zobrazí, v případě úspěšného provedení syntaktické analýzy, které části kódu byly zpracovány kterou gramatikou. Komentáře a direktivy zůstanou neobarvené, analýza je automaticky přeskakuje. Část kódu zpracovaná syntaktickou analýzou shora dolů, které přísluší LL gramatika, se obarví zeleně. Výrazy zpracované precedenční analýzou zdola nahoru se obarví červeně a komunikační symboly, kterými se předává výsledek precedenční analýzy zpět syntaktické analýze, neboli ukončovač precedenční analýzy, se obarví modře. Druhé tlačítko Cancel coloring zruší obarvení jednotlivých tokenů. Tento výstup analýzy slouží pro větší přehlednost a názornost, ale to nejdůležitější se zobrazí v dalším okně – detailní výpis celé analýzy.

První řádek okna Analysis output informuje o úspěchu nebo neúspěchu analýzy. Pokud analýza proběhla v pořádku, vypíše „Syntax: ok“, v opačném případě informuje uživatele o chybě. Může nastat „Lexical error“, pokud se na vstupu objeví neznámý znak nebo řetězec, popř. pokud zadáme číslo ve špatném formátu. Dále může nastat „Syntax error“ při chybě syntaktické analýzy shoda dolů. V tomto případě výpis informuje, u kterého tokenu a na kterém řádku chyba vznikla, popř. jaký token byl očekáván. Nebo vznikne „Precedence error“ při analýze výrazu zdola nahoru. Opět výstup informuje, na kterém řádku chyba vznikla, a pokud je toho program schopen, konkrétně definuje danou chybu, např. chybějící operand. Pokud nedojde k chybě lexikální analýzy, v tom případě se syntaktická analýza vůbec neprovádí, následuje informační sdělení velká tabulka obsahující veškerá pravidla, která byla při analýze aplikována v pořadí jejich vyhodnocení. První sloupec tabulky informuje o tom, které pravidlo bylo použito. Ve druhém sloupci je uveden startující token daného pravidla a třetí sloupec obsahuje řádek, na kterém byl startující token identifikován. Toto vše platí pro syntaktickou analýzu shora dolů, tedy za pomoci LL pravidel. Pokud analyzátor přepnul na precedenční analýzu zdola nahoru, daný řádek tabulky bude obsahovat přes první dva sloupce oranžový text „Precedence analysis“ a ve třetím sloupci opět číslo řádku, na kterém se analýza prováděla.

Poslední věcí, která je uživateli v aplikaci k dispozici, jsou dvě zaškrtávací políčka u tabulky pravidel. První checkbox nazvaný „Detailed rules“ po zaškrtnutí přidá do tabulky pravidel čtvrtý řádek s celým pravidlem, které bylo u daného tokenu vyhodnoceno. Není pak nutné při čtení analýzy posouvat stránku neustále nahoru na vstupní zdrojový kód a dolů na tabulku pravidel LL gramatiky. Vše je přehledné a tabulku lze číst sekvenčně shora dolů. Druhý checkbox označený „precedence“ po zaškrtnutí přidá pod každý výskyt precedenční analýzy v tabulce pravidel další podtabulku s kompletním vyhodnocením daného výrazu. Podtabulka obsahuje čtyři sloupce, terminály/neterminály na zásobníku, operátor, vstup a pravidlo. Díky těmto všem informacím poskytuje výstup analýzy kompletní rozbor zdrojového kódu včetně všech uplatněných pravidel a provedených redukcí vč. mezikroků, a to dokonce i v situaci, kdy došlo k syntaktické nebo precedenční chybě. Analýza se sice ukončí při první nalezené chybě, ale dosavadní postup zůstává uložený a aplikace jej v pořádku zobrazí až do místa chyby.

Celé uživatelské rozhraní je dynamické a dobře zobrazitelné jak na velkých monitorech, tak i na menších, protože při zúžení okna pod určitou mez se některé tabulky zarovnají pod sebe, zatímco na velkém monitoru jsou pro lepší přehlednost vedle sebe. Stejně tak obnovování stránky probíhá pouze při provedení analýzy, kdy je potřeba předat php serveru vstupní kód a získat zpět výstup analýzy. Ostatní manipulace se stránkou probíhá dynamicky přes JavaScript bez obnovování na straně klienta. Implementaci rozhraní vč. spuštění analýzy zavoláním příslušných funkcí obsahuje soubor `index.php`, který zároveň zajišťuje automatické načtení celé aplikace.

5.3 Lexikální analýza

Nyní přejděme od uživatelského rozhraní k jádru aplikace a použitým postupům. Po spuštění analýzy se metodou `POST` předá php serveru vstupní zdrojový kód, který se má zpracovat. Nejdříve se nad ním provede lexikální analýza, protože potřebujeme převést vstupní textový řetězec na řetězec tokenů, se kterými již ostatní analýzy dokáží pracovat. Zároveň se tím vstupní text zkontroluje na případné lexikální chyby nebo neznámé znaky.

Lexikální analýza je implementovaná v modulu `lexa.php` jako jedna funkce, která požaduje za parametr vstupní text a vrací `true` resp. `false` při úspěchu resp. neúspěchu analýzy. Algoritmus je konstruován jako klasický konečný automat. Při zavolání funkce `lexAnalyza($vstup)` se nejdříve inicializují potřebné proměnné, stav automatu se nastaví na `START` a program vstoupí do nekonečné smyčky. Postupně čte vstupní text po jednotlivých znacích a na základě jejich hodnoty přepíná mezi stavy a vyhodnocuje jednotlivé tokeny. Protože je celá aplikace ve výsledku zaměřená více na informace o analýze a jejím postupu, než na analýze samotné, je tomu celý algoritmus přizpůsoben. V praxi se uplatňuje většinou přístup, kdy analýzu řídí syntaktický analyzátor a žádá si jednotlivé tokeny od lexikální analýzy. V tomto případě však z důvodu potřeby kompletních informací o analýze a především přepisu vstupního textu zpět na výstup, ukládá lexikální analýza veškeré zpracované tokeny do globálního pole `$lexPole` a zpracuje tak celý vstupní text najednou. Do pole ukládá samotný řetězec, typ tokenu, číslo řádku, na kterém byl analyzován, a do posledního sloupce barvu tokenu pro pozdější rozlišení, která analýza zpracovala který token. Další zvláštností je, že se správně rozpoznávají komentáře, bílé znaky a direktivy, ale nezahazují se. Mají svůj vlastní typ a ukládají se spolu s ostatními tokeny, aby bylo možné vstupní text správně interpretovat na výstupu. První nultý řádek pole uchovává informaci, jestli analýza proběhla v pořádku, popř. u kterého tokenu a na kterém řádku chyba vznikla.

5.4 Syntaktická analýza

Po úspěchu lexikální analýzy se zavolá řídící syntaktická analýza, která je implementovaná pomocí metody rekurzivního sestupu. To znamená, že pravidlům gramatiky, v tomto případě LL gramatiky, jsou přiřazeny jednotlivé funkce, jinak řečeno každé pravidlo má svoji funkci, které se mezi sebou vzájemně volají, dokud nezanalyzují celý řetězec tokenů nebo nedojde k chybě.

Z praktických důvodů je gramatika značně omezená. Jedná se o podmnožinu jazyka C, která se zaměřuje na základní struktury a konstrukce jako je deklarace, přiřazení výrazu do proměnné, jednoduché volání funkcí, cyklus `while` a podmínka `if` vč. větve `else`. Syntaktický analyzátor je

implementován takovým způsobem, že není problém kdykoli provést rozšíření gramatiky, ale pro přehlednost aplikace je přidávání dalších přijímaných konstrukcí nepraktické, protože rapidně zvyšuje počet pravidel. Celý systém se tak stává nepřehledný a ztrácí svoji názornost. Kompletní gramatiku lze nalézt na konci dokumentu v příloze B.

Syntaktický analyzátor je implementován ve svém vlastním modulu `syna.php`, který obsahuje veškeré funkce a konstrukce. Nejdříve je zavolána funkce `synAnalyza()`, ta provede počáteční inicializaci proměnným a druhého globálního pole `$rulPole`, které obsahuje výstup pravidel ve formátu: pravidlo v prvním sloupci, počáteční token ve druhém a informaci, jestli se jednalo o syntaktickou nebo precedenční analýzu ve třetím sloupci. Hlavička, tedy první nultý řádek, nese informaci, jestli analýza proběhla v pořádku, popř. chybovou zprávu v prvním sloupci, dále na kterém tokenu (druhý sloupec) ve kterém pravidle (třetí sloupec) nastala chyba. Poté se již zavoláním funkce `prog()`, která bude očekávat první tokeny `int main` a spustí sekvenci analýzy. V případě, že narazí na výraz, vyžádá si práci precedenční analýzy zavoláním funkce `precAnalyza()`.

5.5 Precedenční analýza

Jak již bylo několikrát zmíněno, precedenční analyzátor provádí analýzu zdola nahoru. Jedná se tedy redukci od listů ke kořeni v derivačním stromu a využívá k tomu jednoduchý algoritmus, zásobník a tabulku symbolů. Používá se pro zpracování výrazů, kde záleží na pořadí vyhodnocení operátorů, závorek atd. Implementován je v modulu `preca.php`, kde je jak hlavní funkce `precedencni()`, která se volá ze syntaktické analýzy, tak i nadefinovaná tabulka symbolů a další pomocné funkce. Algoritmus má přístup k oběma globálním polím, takže pokračuje ve zpracovávání tokenů z `$lexPole` od místa, kde si vyžádala syntaktická analýza práci precedenční analýzy a vyhodnocené kroky ukládá do `$rulPole`, stejně jak to prováděla syntaktická analýza, ve formátu: obsah zásobníku do prvního sloupce, aktuální operátor z tabulky do druhého sloupce, token na vstupu do třetího a pokud v daném kroku dojde k redukci, uloží se do čtvrtého sloupce. Pokud při analýze dojde k chybě, nastaví se opět do nultého řádku tabulky pravidel.

Precedenční analyzátor na začátku činnosti inicializuje zásobník a nastaví do něho ukončující symbol `$`. Poté čte jednotlivé tokeny, porovnává je s terminálem na vrcholu zásobníku a vyhodnocuje na základě algoritmu:

```
repeat
  a = terminál na vrcholu zásobníku
  b = token na vstupu
  switch TabulkaSymbolu[a,b]:
    case = : push(b) a načti další token do b
    case < : přidej < za a (tedy na zásobníku místo a bude a<),
             push(b) a načti další token do b
    case > : pokud <b je na vrcholu zásobníku a r: E -> b ∈ P,
             pak zaměň <b za E (provede redukci) a zapamatuj si
             použití pravidla r, jinak chyba
    default: chyba (prázdné políčko)
until a = $ a b = $
```

Ukončující symbol na vrcholu zásobníku je vždy jasně dán a nastaven na počátku, zatímco u ukončujícího symbolu na vstupu se musí rozlišovat, jestli se nejedná o středník na místě ukončení příkazu, např. přiřazení, nebo v případě podmínky `if` či `while` hledáme odpovídající párovou závorku. Oba tyto symboly slouží jako komunikační symbol, který ukončí činnost precedenční analýzy a předá její výsledek syntaktické analýze, která pokračuje v kontrole syntaxe shora dolů.

Precedenční analyzátor přijímá všechny základní matematické operace $(+, -, *, /)$ a porovnávací operátory $(<, >, <=, >=, ==, !=)$ pro operandy typu `integer` nebo identifikátory za použití libovolného počtu kulatých závorek. Kompletní precedenční tabulku lze nalézt v příloze C. Řádky označují terminály na zásobníku, zatímco sloupce udávají terminál na vstupu. Béžově označená pole vyjadřují stejnou prioritu operátorů a **>** říká, že se jedná o levě asociativní operátory, opačná šipka **<** by značila pravě asociativní operátory. Ostatní šipky vyjadřují prioritu operátorů a ostatních symbolů. Párové závorky jsou definovány pomocí **=** a červeně vyplněná prázdná pole udávají, že tyto operátory nebo symboly se nesmí vyskytnout vedle sebe, např. dva identifikátory vedle sebe nebo třeba závorky **)(**.

6 Závěr

Zkoumání modelů pro paralelní či distribuované zpracování probíhá již několik desítek let v mnoha odvětvích nejen v oblasti formálních jazyků. Jedním z velkých předpokladů tohoto směru vývoje je snaha o maximalizaci využití více zdrojů současně při řešení daného problému, což nás přivádí k paralelismu. Další směřování vede na situaci, kdy máme opět více zdrojů, ale každý z nich nese jiné informace nebo je schopen přispět jinou částí řešení. Potřebujeme pak vhodné mechanismy, které by nám umožnili rozdělit práci mezi tyto zdroje a správným způsobem je řídit. Můžeme se v obecné šíři dočíst například o tzv. nástěnkových modelech nebo modelech tabule, které popisují situace z reálného světa týkající se spolupráce lidí na řešení problému. Jako příklad obecného modelu pro distribuované zpracování lze uvést v literatuře často zmiňovaný The BlackBoard Model [8], kdy se při řešení společného problému střídají jednotliví aktéři u tabule a přispívají postupně svými nápady. Využívají tak znalostí všech, přičemž každý provede pouze dílčí část řešení. Naopak dobře pochopitelná analogie s paralelními systémy je např. vývoj softwaru, kdy každá divize firmy vyvíjí samostatný modul současně s ostatními. Výsledná aplikace je následně tvořena spojením jednotlivých částí. Obecné modely přispěly velkou mírou k rozvoji těchto systému, ale nebyly jediné. Technický pokrok v informačních technologiích a rozvoji počítačových sítí v podstatě vyžaduje existenci aparátů jak pro rozložení zátěže či výkonu, tak pro sdílení informací a využití vzdálených zdrojů.

První myšlenky gramatických systémů, které by využívaly těchto modelů a znalostí, se objevily již v sedmdesátých letech minulého století [6] a souvisely s návrhy systémů pro syntaktickou analýzu s využitím kooperujících gramatik. Od té doby vývoj gramatických systémů pokračoval a dnes rozeznáváme dva základní modely. Kooperující distribuované gramatické systémy, zkráceně CD gramatické systémy, a paralelně komunikující gramatické systémy neboli PC gramatické systémy. V této práci jsem se věnoval gramatickým systémům ve spojitosti s bezkontextovými gramatikami, o kterých pojednává kapitola 2, které jsou nejběžnější právě při definici programovacích jazyků, jimiž jsem se zabýval v druhé části práce ve spojení se syntaktickou analýzou.

CD gramatické systémy, jak již z názvů vyplývá, pracují na principu spolupráce více gramatik na řešení daného problému. V případě gramatických systému sdílejí jednu větnou formu a střídají se v jejím zpracování. Máme pak gramatický systém s n množinami bezkontextových pravidel, které nazýváme komponenty, a podle módu systému každá z komponent provede daný počet derivačních kroků, než předá řízení další komponentě. Derivační módy jsou dva, ukončující mód (Terminating Mode), v něm komponenta pracuje tak dlouho, dokud může, a k-krokový mód (K-Step Mode), ve kterém každá komponenta provede k kroků, nejvýše k kroků nebo alespoň k kroků. Generativní síla těchto systémů se odvíjí od generativní síly bezkontextových gramatik a podle počtu použitých komponent a módu může nabýt až síly maticových jazyků nebo ET0L jazyků.

Naopak PC gramatické systémy pracují paralelně a každá komponenta pracuje na své vlastní větné formě, přičemž mezi sebou mohou komunikovat pomocí tzv. komunikačních symbolů. U těchto systémů rozeznáváme dva typy derivačních kroků. Generativní derivační krok, v tom se komponenty nacházejí za normálních okolností a provádějí klasické derivační kroky. Pokud však některá z komponent vygeneruje komunikační symbol, systém se přepne do komunikačního derivačního kroku, ve kterém má za úkol nahradit komunikační symbol vygenerovaný jednou komponentou za větnou formu komponenty, na kterou komunikační symbol odkazoval. Generativní síla PC gramatických systémů závisí opět na počtu komponent systému a typu pravidel, které smíme použít.

Zbylou část práce jsem věnoval syntaktické analýze s možností využití gramatických systémů, která je obecně velmi aktuální a využití paralelního zpracování nebo distribuce při syntaktické analýze může přinést úsporu času i výkonu. V praktické části jsem se zaměřil na PC

gramatické systémy v jednom z možných režimů, a to když máme jednu řídící gramatiku, pouze ta může generovat komunikační symbol. Daný gramatický systém se pak nazývá centralizovaný a řídící gramatika si v případě potřeby může vyžádat práci jiné gramatiky. Jedná se o velmi užitečný režim v rámci syntaktické analýzy, kdy je analyzovaný kód složen z více různých jazyků nebo je vhodnější pro různé části kódu uplatnit jiné přístupy. V mém případě jsem tak propojil syntaktickou analýzu shora dolů využívající LL gramatiku pro kontrolu syntaktických konstrukcí s precedenční analýzou, která pracuje zdola nahoru, využívá zásobník a tabulku symbolů, pro analýzu výrazů.

Mým původním záměrem bylo vytvořit konzolový program, který by v rámci možností terminálu ukázal spolupráci gramatik při syntaktické analýze. Po následných konzultacích bakalářské práce vznikla myšlenka názornější aplikace, která by nejen byla součástí této práce, ale mohla by posloužit i jako studijní pomůcka dalším ročníkům v předmětu Formální jazyky a překladače (IFJ). Rozhodl jsem se tedy vytvořit webovou aplikaci, která provede syntaktickou analýzu vstupního kódu v podmnožině jazyka C a přehledným způsobem zobrazí detailní výstup analýzy. Podmnožinu programovacího jazyk C jsem zvolil úmyslně, aby aplikace byla pro studenty co nejužitečnější, protože se s jazykem C seznámí již v prvním ročníku a v rámci IFJ v něm programují projekt.

Výsledná webová aplikace je dostupná i přes internet. Umožňuje uživateli zadat libovolný vstupní kód, nad kterým provede nejdříve lexikání analýzu, po které následuje syntaktická analýza, která si v případě potřeby vyžádá práci precedenčního analyzátoru a na základě ukončujících symbolů (středník, párová závorka), které reprezentují komunikační symboly, předá řízení zpět syntaktickému analyzátoru. Následný výstup analýzy zobrazí nejen sekvenci aplikovaných pravidel, ale i kompletní posloupnost redukcí provedenou při precedenční analýze. Formát výstupu zároveň odpovídá konvencím ze studijních materiálů předmětu IFJ. Zavedl jsem pouze omezenou sadu pravidel pro nejzákladnější konstrukce jazyka C, protože mým cílem nebylo vytvořit další kompilátor, ale poskytnout co nejpřehlednější výstup analýzy pro lepší pochopení jejích principů.

Obecně lze říci, že jsem touto prací vytvořil odrazový můstek pro podrobnější zkoumání gramatických systémů. Zatímco začátek práce je věnován bezkontextovým gramatikám, se kterými jsem následně pracoval, další kapitoly řeší problematiku gramatických systémů od jejich teoretických základů. Práce tedy popisuje na teoretické rovině, jaké pohnutky a motivace vedly k samotnému zkoumání paralelních a distribuovaných systémů a jaké obecné modely vznikly v průběhu let v různých odvětvích. To vše vedlo k postupnému vzniku gramatických systémů do podoby, v jaké jsem je následně definoval, popsal jejich obecné principy a uvedl doplňující příklady. Na tomto základu lze pokračovat mnoha směry. Jednou z možností by bylo ponechání PC a CD systémů tak, jak jsem je definoval a zkoumat je podrobněji s jinými gramatikami, než jenom bezkontextovými. Lze zavést modely pro lineární jazyky, samozřejmě regulární jazyky, které využívám v praktické části pro lexikální analýzu, dále pak maticové jazyky nebo např. ET0L jazyky. Kromě změny gramatiky sytému by mohlo přinést další poznatky experimentování se samotnými systémy. Zkoumat podrobněji jejich chování v různých módech, které jsem popsal. Na jejich základě se pokusit definovat nové módy či derivační kroky, případně zkoumat možnost většího propojení jednotlivých režimů, než jak to umožňují např. hybridní CD gramatické systémy. Tyto a jistě i mnoho dalších vlastností lze zkoumat u samotných gramatických systémů. Druhá polovina této práce řeší oblast syntaktické analýzy, kterou jsem postihl spíše základním přehledem. Zde se nabízí nepřeberné množství jednotlivých oblastí výzkumu. Podrobněji jsem zkoumal syntaktickou analýzu na základě LL gramatiky, v praxi pomocí rekurzivního sestupu, a precedenční analýzu. Takže další samostatnou práci by mohl pokrýt výzkum např. prediktivního LL syntaktického analyzátoru shora dolů spolu s LR syntaktickou analýzou pro zpracování výrazů, která pracuje zdola nahoru. Pochopitelně velmi přínosné by mohlo být hlubší propojení gramatických systémů se syntaktickou analýzou. Zkoumat chování syntaktické analýzy za použití CD gramatických systémů, jejichž komponenty by např. běžely na více serverech. Podobně u PC gramatických systémů, které by mohly fungovat více

vláknově a provádět paralelní zpracování analyzovaného kódu. Musely by se pak nejspíše zavést mechanismy, které by umožnily vyhledávání ve zdrojovém kódu, aby si každá komponenta předem našla tu část kódu, pro kterou je určená a mohla souběžně pracovat s ostatními, namísto čekání na vyžádání od řídící komponenty, jako jsem to zkoumal v této práci já. V neposlední řadě je zde mnoho možností budoucího vývoje mnou vytvořené aplikace z praktické části. Aktuálně provádí syntaktickou analýzu a na výrazy si žádá práci precedenčního analyzátoru. Aplikace je implementována takovým způsobem, že umožňuje bez větších komplikací provést jak rozšíření přijímaného jazyka, tak lze rozšířit celou práci např. o sémantickou analýzu nebo vytvořit tímto způsobem celý interpret. Interpret programovacího jazyka, který by fungoval v prohlížeči a navíc by uživateli poskytnul podrobný výpis analýzy, by mohl být opravdu přínosem nejen pro studenty, ale všechny, kdo se tímto chtějí zabývat a snadno celé problematice porozumět. S ohledem na interaktivitu aplikace by šlo navázat na tuto práci rozvojem dalších dynamických možností, které by poskytovala. Velmi zajímavé by mohlo být např. upravení stávajícího programu takovým způsobem, že by šla jednotlivá pravidla LL gramatiky za běhu aplikace upravovat. Implementace takovýchto možností by byla jistě složitější, ale ne nemožná. Při využití např. rekurzivního sestupu mají jednotlivá pravidla vlastní funkce, takže by mělo být možné je upravit i přímo v rozhraní aplikace. Analyzátor by je tedy neměl deterministicky implementované, ale převzal by si jejich podobu z rozhraní a uplatnil při analýze v takovém pořadí, jak byly uživatelem definovány. Na tuto práci lze tedy navázat nejen po stránce teoretické zkoumáním gramatik a jazyků, ale rozvíjet možnosti syntaktické analýzy, kompilátorů a uživatelských rozhraní.

Literatura

[1] MEDUNA, Alexander a Petr ZEMEK. *Regulated grammars and automata*. xx, 694 pages. ISBN 1493903683.

[2] MEDUNA, Alexander. *Formal languages and computation: models and their applications*. pages cm. ISBN 9781466513457.

[3] MEDUNA, Alexander. *Elements of compiler design*. Boca Raton, FL: Auerbach Publications, c2008, xiii, 286 p. ISBN 1420063235.

[4] TECHET, Jiří, Tomáš MASOPUST a Alexander MEDUNA. Modern Formal Language Theory: Cooperating Distributed Grammar Systems [online]. Brno, 2007 [cit. 2015-04-06]. Dostupné z: http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:09-cdgspres.pdf

[5] TECHET, Jiří, Tomáš MASOPUST a Alexander MEDUNA. Modern Formal Language Theory: Parallel Communicating Grammar Systems [online]. Brno, 2007 [cit. 2015-04-06]. Dostupné z: http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:10-pcgspres.pdf

[6] MEERSMAN, R. a G. ROZENBERG Cooperating grammar systems. *Mathematical Foundations of Computer Science 1978: Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1978, s. 364-373. ISBN 9783540089216.

[7] ROZENBERG, G. a Arto SALOMAA, eds. *Handbook of formal languages: background and application*. Berlin: Springer-Verlag, 1997, xxii, 528 s. ISBN 3540614869.

[8] NII, H. Penny *The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures*. AI Magazine Volume 7 Number 2, 1986, pp. 38–53

Příloha A

Obsah CD

1. Text bakalářské práce ve formátu PDF.
2. Zdrojový text práce ve formátu ODT (docx).
3. Zdrojové soubory aplikace a manuál.

Příloha B

LL gramatika

```
1.  <prog>     ->  int main ( <m-list> ) { <st-list>

2.  <m-list>   ->  void
3.  <m-list>   ->  eps

4.  <st-list>  ->  <state> <st-list>
5.  <st-list>  ->  }

6.  <state>    ->  int id ;
7.  <state>    ->  id <id>
8.  <state>    ->  if ( <EXPR> ) <if-body>
9.  <state>    ->  while ( <EXPR> ) <body>
10. <state>    ->  return <EXPR> ;

11. <id>       ->  = <EXPR> ;
12. <id>       ->  ( <call-fce>

13. <if-body>  ->  <state> <else>
14. <if-body>  ->  { <st-list> <else>

15. <else>     ->  else <body>
16. <else>     ->  eps

17. <body>     ->  <state>
18. <body>     ->  { <st-list>

19. <call-fce> ->  ) ;
20. <call-fce> ->  <item> <it-list>

21. <it-list>  ->  , <item> <it-list>
22. <it-list>  ->  ) ;

23. <item>     ->  string
24. <item>     ->  <EXPR>

<EXPR>  ->  precedence analysis
eps     ->  epsilon rule
```

Příloha C

Tabulka symbolů

Symboly na zásobníku \ Symboly na vstupu	+	-	*	/	<	>	<=	=>	==	!=	(	)	i	$
+	>	>	<	<	>	>	>	>	>	>	<	>	<	>
-	>	>	<	<	>	>	>	>	>	>	<	>	<	>
*	>	>	>	>	>	>	>	>	>	>	<	>	<	>
/	>	>	>	>	>	>	>	>	.>	>	<	>	<	>
<	<	<	<	<	>	>	>	>	>	>	<	>	<	>
>	<	<	<	<	>	>	>	>	>	>	<	>	<	>
<=	<	<	<	<	>	>	>	>	>	>	<	>	<	>
=>	<	<	<	<	>	>	>	>	>	>	<	>	<	>
==	<	<	<	<	<	<	<	<	>	>	<	>	<	>
!=	<	<	<	<	<	<	<	<	>	>	<	>	<	>
(	<	<	<	<	<	<	<	<	<	<	<	=	<	
)	>	>	>	>	>	>	>	>	>	>		>		>
i	>	>	>	>	>	>	>	>	>	>		>		>
$	<	<	<	<	<	<	<	<	<	<	<		<	

Tabulka 1: Tabulka symbolů precedenční analýzy

```
1. E -> E + E    6.  E -> E > E     11. E -> (E)
2. E -> E - E    7.  E -> E <= E    12. E -> i
3. E -> E * E    8.  E -> E >= E        i -> id
4. E -> E / E    9.  E -> E == E        i -> int
5. E -> E < E    10. E -> E != E
```