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 APLIKOVANÉ V SYNTAKTICKÉ ANALÝZE

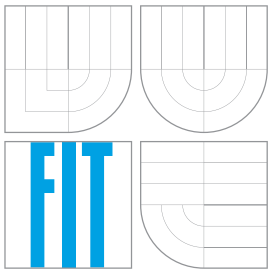
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB MARTIŠKO

BRNO 2015

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 APLIKOVANÉ V SYNTAKTICKÉ ANALÝZE

GRAMMAR SYSTEMS APPLIED TO PARSING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB MARTIŠKO

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2015

Abstrakt

Tato práce se zabývá především různými variantami gramatických systémů. Gramatické systémy kombinují relativní jednoduchost bezkontextových gramatik s generativní silou komplexnějších gramatik. V rámci práce jsou popsány dva základní typy: PC gramatické systémy a CD gramatické systémy. Mimo to, je v rámci této práce zaveden i systém nový, který vychází z CD gramatických systémů. Na základě tohoto nového systému je také zavedena nová metoda syntaktické analýzy. Takto navržený analyzátor pak sestává z více menších syntaktických analyzátorů, které pracují jak metodou zdola nahoru tak i shora dolů.

Abstract

This paper deals with different variants of grammar systems. Grammar systems combine the simplicity of Context Free Grammars with the generative power of more complex gammars. There are two main variants of grammar systems described in this paper: PC grammar systems and CD grammar systems. New type of grammar system, which is a modification of the CD grammar systems, is also described in the paper. New method of parsing, based on this new grammar system is proposed in the paper. This new parser consists of several smaller parsers, which work in both top down and bottom up way.

Klíčová slova

CD gramatické systémy, PC gramatické systémy, Formální jazyky, LL syntaktická analýza, LR syntaktická analýza

Keywords

CD grammar systems, PC grammar systems, Formal languages, LL parsing, LR parsing

Citace

Jakub Martiško: Gramatické systémy aplikované v syntaktické analýze, diplomová práce, Brno, FIT VUT v Brně, 2015

Gramatické systémy aplikované v syntaktické analýze

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandra Meduny, CSc. Uvedl jsem všechny literární prameny, ze kterých jsem čerpal.

.....
Jakub Martiško
26. května 2015

Poděkování

Na tomto místě bych rád poděkoval vedoucímu své práce panu prof. RNDr. Alexandru Medunovi, CSc. za věnovaný čas a pomoc při řešení této práce. Také bych rád poděkoval své rodině za podporu během mého studia.

Obsah

Kapitola 1

Úvod

Formální jazyky jsou využívány v mnoha různých vědeckých disciplínách, jako je bioinformatika, lingvistika a teoretická informatika. Existuje velké množství modelů, které popisují různé varianty formálních jazyků, které se liší mimo jiné tím, jak složité jazyky jsou schopny popsat. Jedním z nejběžněji používaných z těchto modelů jsou v rámci tvorby překladačů bezkontextové gramatiky a jim ekvivalentní zásobníkové automaty. Výhodou bezkontextových gramatik je jejich relativní jednoduchost a snadnost použití. Na druhou stranu, existují poměrně jednoduché jazyky, které nejdou tímto modelem popsat. Příkladem takového jazyku je jazyk, jehož řetězce se skládají pouze ze tří symbolů, a počet jednotlivých symbolů je v rámci řetězce stejný.

Jedním z přístupů, jak zvýšit sílu bezkontextových gramatik bez přílišného zvýšení složitosti modelu, jsou, mimo jiné i gramatické systémy. Gramatické systémy využívají několika dílčích (většinou právě bezkontextových) gramatik, nazývaných komponenty, které určitým předem definovaným způsobem spolupracují na tvorbě výsledného jazyku. Dva základní typy gramatických systému jsou *CD (Cooperating distributed) gramatické systémy* a *PC (parallel communicating) gramatické systémy*.

V této práci je pak zavedena nová varianta CD gramatických systémů. Tyto systémy se skládají z několika dílčích gramatik a mechanismu, který těmto gramatikám jistým způsobem omezuje počet pravidel, která mohou při své činnosti použít. Na základě tohoto modelu je také zavedena nová metoda syntaktické analýzy, která také sestává z více dílčích částí, které kombinují metody LL a LR syntaktické analýzy. Na základě této navržené metody byla implementována demonstrační aplikace, která znázorňuje činnost syntaktického analyzátoru založeného na této metodě.

1.1 Logické členění práce

Tato práce se skládá ze dvou hlavních částí. První část, sestávající z kapitol 2 a 3, se zabývá formálními modely, na kterých jsou založeny metody syntaktické analýzy. Druhá část práce se pak zabývá právě syntaktickou analýzou. Součástí této druhé části jsou pak kapitoly 4, 5 a 6.

V rámci této práce, jsou nejprve zavedeny potřebné formalismy nutné k popisu jednotlivých modelů a vztahů. Tímto úvodem do problematiky se zabývá kapitola 2. V rámci této kapitoly jsou formálně popsány jak PC, tak i CD gramatické systémy. Navíc jsou zde představeny i některé z modifikací těchto systémů.

Kapitola 3 je pak klíčovou částí této práce. V této kapitole je zavedena nová varianta

gramatických systémů. Tento systém vychází z CD gramatických systémů a zavádí určité požadavky na činnost gramatik, z nichž se tento systém skládá. Obdobně jako v případě standardních CD gramatických systémů, jsou i tyto nově zavedené systémy schopny pracovat i s jazyky, které nejsou bezkontextové. V rámci této kapitoly jsou pak také uvedeny některé vlastnosti navrhovaného systému, zejména pak generativní síla tohoto druhu systému vzhledem k jiným používaným modelům.

Kapitola 4 představuje metody, používané při syntaktické analýze. Dále je v této kapitole zavedena metoda nová. Tato nová metoda kombinuje metody běžně užívané a modifikuje je na základě gramatických systémů, zavedených v kapitole 3. Obdobně jako tyto nově zavedené gramatické systémy, je i tato nová metoda syntaktické analýzy schopna pracovat i s jazyky mimo třídu bezkontextových jazyků.

Příklady syntaktické analýzy postavené na těchto nově zavedených metodách jsou popsány v kapitole 5. V této kapitole je demonstrována činnost nově zavedených metod na několika jazycích. Tyto jazyky pak nejsou bezkontextové. Kromě toho jsou také na těchto příkladech demonstrována možná rozšíření a modifikace navržené metody.

Kapitola 6 pak popisuje aplikaci, která byla implementována v rámci této práce. Tato aplikace slouží k demonstraci činnosti zavedených metod. Příklady z kapitoly 5 jsou pak součástí této aplikace.

Kapitola 2

Základní definice a pojmy

V rámci této kapitoly budou definovány základní pojmy z oblasti formálních jazyků. Dále jsou zde také popsány základní varianty gramatických systémů, které jsou pak dále v této práci modifikovány a rozšiřovány.

2.1 Základní definice

Definice 1 (Abeceda)**.** *Abeceda je neprázdná konečná množina prvků, které se nazývají symboly.*

Definice 2 (Řetězec nad abecedou)**.** *Nechť Σ je abeceda. Pak:*

- *ε je řetězcem nad abecedou Σ, ε značí prázdný řetězec, tedy řetězec, který neobsahuje žádné symboly.*
- *Nechť $a \in \Sigma$ a x je řetězcem nad Σ. Pak také řetězec xa je řetězcem nad abecedou Σ.*
- *Množina Σ^* pak značí množinu všech řetězců nad abecedou Σ.*

Řetězec je pak tedy nějaká posloupnost symbolů dané abecedy. Tato posloupnost může být i prázdná, tedy neobsahovat symbol žádný.

Definice 3 (Délka řetězce)**.** *Délka řetězce x (značeno $|x|$) nad abecedou Σ je:*

- *Pokud x=ε, pak $|x| = 0$.*
- *Pokud x=$a_1a_2a_3a_4 \ldots a_n$, kde a_i, pro všechna $i = 1 \ldots n$ jsou symboly abecedy Σ, pak $|x| = n$.*

Délka řetězce tedy odpovídá celkovému počtu symbolů, z kterých se řetězec skládá. Jako $|x|_a$ pak budu značit počet symbolů a v řetězci x.

Definice 4 (Konkatenace řetězce)**.** *Nechť x a y jsou řetězce. Pak řetězec z, vzniklý konkatenací řetězce x s y má tvar $z = xy$.*

Definice 5 (Mocnina řetězce)**.** *Mocnina řetězce x (značeno jako x^i)nad abecedou Σ, je řetězec y nad abecedou Σ takový že:*

- $x^0 = \varepsilon = y$
- $x^i = xx^{i-1} = y$

Definice 6 (Prefix, sufix a podřetězec řetězce)**.** *Nechť v, x, y, z jsou libovolné řetězce (včetně prázdných) nad abecedou Σ. Pak:*

- *Nechť $v = xy$, řetězec x pak nazýváme prefixem řetězce v.*
- *Nechť $v = xy$, řetězec y pak nazýváme sufixem řetězce v.*
- *Nechť $v = xyz$, řetězce x,y a z pak nazýváme podřetězci řetězce v.*

Definice 7 (Reverzace řetězce)**.** *Nechť $x = x_1x_2 \dots x_{n-1}x_n$ je řetězec. Pak řetězec y, vzniklý reverzací řetězce x má tvar $y = rev(x) = x_nx_{n-1} \dots x_2x_1$.*

2.2 Jazyky

Definice 8 ((Formální) Jazyk)**.** *Jazyk nad abecedou Σ je množina L taková, že: $L \subseteq \Sigma^*$.*

Jazyk je tedy množina řetězců, tvořených symboly z nějaké abecedy. Jelikož jsou jazyky množinami, jsou nad nimi standardním způsobem definovány množinové operace průnik ($L_1 \cup L_2$), sjednocení ($L_1 \cap L_2$) a rozdíl ($L_1 - L_2$).

Definice 9 (Doplněk jazyka)**.** *Nechť L je jazyk nad abecedou Σ, doplňkem jazyka L je jazyk $\bar{L} = \Sigma^* - L$.*

Definice 10 (Konkatenace jazyka)**.** *Nechť L_1 a L_2 jsou jazyky. Konkatenací (značeno $L_1.L_2$) těchto jazyků vzniká jazyk $L_3 = \{xy | x \in L_1 \wedge y \in L_2\}$.*

Definice 11 (Mocnina jazyka)**.** *Nechť L_1 je jazyk nad abecedou Σ, jeho i-tou mocninou je jazyk L^i takový, že:*

- $L^0 = \{\varepsilon\}$
- $L^i = L.L^{i-1}$

Definice 12 (Iterace jazyka)**.** *Iterace jazyka L, je jazyk $L^* = \bigcup_{n=0}^{\infty} L^n$.*

Definice 13 (Pozitivní iterace jazyka)**.** *Pozitivní iterace jazyka L, je jazyk $L^+ = \bigcup_{n=1}^{\infty} L^n$.*

Každá iterace jazyka pak obsahuje i prázdný řetězec. Pozitivní iterace jej pak obsahuje pouze tehdy, pokud jej obsahoval i původní jazyk.

2.3 Gramatiky

Definice 14 (Gramatika)**.** *Gramatikou, nazýváme čtveřici $G = (N, T, P, S)$, kde:*

- *N je abeceda symbolů, nazývaných neterminály.*
- *T je abeceda symbolů, nazývaných terminály.*
- *$S \in N$ je počáteční neterminál.*
- *P je konečná množina pravidel tvaru $\alpha \to \beta$, kde α a β jsou řetězce nad abecedou $N \cup T$, kde α obsahuje alespoň jeden neterminál.*

Definice 15 (Gramatika typu 0)**.** *Pro gramatiku typu 0 (nazývaná také* neomezená gramatika*) platí, že pravidla z množiny P jsou tvaru $\alpha \to \beta$, kde $\alpha \in (N \cup T)^* N (N \cup T)^*$ a $\beta \in (N \cup T)^*$*

Definice 16 (Gramatika typu 1)**.** *Pro gramatiku typu 1 (nazývaná také* kontextová gramatika*) platí, že pravidla z množiny P jsou tvaru $\alpha \to \beta$, kde $\alpha \in (N \cup T)^* N (N \cup T)^*$ a $\beta \in (N \cup T)^*$ a $|\alpha| \leq |\beta|$.*

Definice 17 (Gramatika typu 2)**.** *Pro gramatiku typu 2 (nazývaná také* bezkontextová gramatika*) platí, že pravidla z množiny P jsou tvaru $\alpha \to \beta$, kde $\alpha \in N$ a $\beta \in (N \cup T)^*$*

Definice 18 (Gramatika typu 3)**.** *Pro gramatiku typu 3 (nazývaná také* pravá lineární gramatika*) platí, že pravidla z množiny P jsou tvaru $\alpha \to \beta$, kde $\alpha \in N$ a $\beta \in T^* \cup (T^* N)$*

Definice 19 (Větná forma)**.** *Větná forma dané gramatiky je řetězec x nad abecedou $N \cup T$ takový, že existuje posloupnost pravidel této gramatiky taková, že je daná gramatika schopna vygenerovat tento řetězec ze svého počátečního neterminálu.*

Definice 20 (Věta)**.** *Věta je taková větná forma, která obsahuje pouze terminály.*

Gramatiky pak fungují tím způsobem, že na aktuální větnou formu postupně aplikují některé ze svých pravidel, čímž tuto větnou formu postupně přepisují. Toto aplikují tak dlouho, dokud existuje ve větné formě nějaký symbol (či řetězec v případě gramatik typu 0 a 1), který je možno přepsat. Všechny řetězce, které se skládají pouze z terminálů a které je možno tímto způsobem vygenerovat pak odpovídají jazyku generovanému příslušnou gramatikou.

2.4 Automaty

Gramatiky slouží ke generování řetězců daného jazyka na základě množiny pravidel. Automaty pak slouží k ověření, zda daný vstupní řetězec patří do daného jazyka. V této práci budou použity zásobníkové automaty, které jsou schopny popsat stejnou třídu jazyků jako bezkontextové gramatiky.

Definice 21 (Zásobníkový automat)**.** *Zásobníkový automat M je sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde:*

- *Q je konečná množina vnitřních stavů automatu.*
- *Σ je vstupní abeceda automatu.*
- *Γ je zásobníková abeceda automatu.*
- *$s \in Q$ je počáteční stav automatu.*
- *$S \in \Gamma$ je počáteční zásobníkový symbol automatu.*
- *$F \subseteq Q$ je množina koncových stavů automatu.*
- *R je konečná množina pravidel tvaru: $Apa \to wq$, kde $A \in \Gamma$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Gamma^*$.*

Zásobníkový automat se pak tedy skládá ze vstupního řetězce, zásobníku, množiny vnitřních stavů a množiny pravidel. Na základě vstupního řetězce, vnitřního stavu a symbolu na vrcholu zásobníku, mění svůj vnitřní stav dle množiny pravidel a přepisuje symboly na vrcholu zásobníku.

Definice 22 (Konfigurace)**.** *Nechť $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat. Řetězec $\chi \in \Gamma^* Q \Sigma^*$ je konfigurací tohoto automatu.*

Konfigurace pak popisuje obsah zásobníku automatu, jeho vnitřní stav a zbývající obsah vstupního řetězce.

Definice 23 (Přechod)**.** *Nechť $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je zásobníkový automat a řetězce $\chi_1 = xApay$ a $\chi_2 = xwqy$ jeho konfigurace, kde $x, w \in \Gamma^*, A \in \Gamma, p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, y \in |\Sigma^*$. Dále nechť $r := Apa \to wq \in R$ je pravidlo. Pak M provede přechod dle r z χ_1 do χ_2, značeno jako $xApay \vdash xwqy[r]$ případně pouze $xApay \vdash xwqy$.*

Existují tři možnosti, jak může zásobníkový automat přijmout vstupní řetězec:

1. Vyprázdněním zásobníku.
2. Přechodem do koncového stavu.
3. Kombinací předchozích dvou možností.

Pro všechny tyto možnosti pak platí, že automat musí také přečíst *celý* vstupní řetězec.

Modifikací zásobníkového automatu jsou pak *rozšířené* zásobníkové automaty. Ty se liší tím, že povolují číst řetězec z vrcholu zásobníku místo pouhého jediného symbolu. Jejich vyjadřovací síla je ovšem stejná jako u klasických zásobníkových automatů.

2.5 Gramatické systémy

Gramatický systém je soubor několika dílčích gramatik[1] (nazývaných komponenty), které spolu určitým způsobem spolupracují a které dohromady generují jeden společný jazyk. Rozlišujeme dva základní druhy gramatických systémů: *PC* gramatické systémy a *CD* gramatické systémy. Většina definic v této sekci pochází z [6].

2.5.1 PC gramatické systémy

PC (parallel communicating) gramatické systémy jsou tvořeny několika gramatikami, které pracují paralelně, každá nad vlastní větnou formou. Každá komponenta systému je definována pomocí svého počátečního neterminálu a pomocí množiny pravidel. Kromě abeced terminálů a neterminálů obsahují PC systémy ještě abecedu třetí. Tato abeceda, značená jako K, obsahuje tzv. dotazovací symboly (query symbols). Tyto symboly slouží k předávání řetězců mezi jednotlivými komponentami.

Definice 24 (PC gramatický systém)**.** *PC gramatický systém (dále značen jako PCGS) Γ stupně $n, n \geq 1$ je k-tice $\Gamma = (N, K, T, (P_1, S_1), \ldots, (P_n, S_n))$, kde:*

- *N je abeceda neterminálů.*
- *T je abeceda terminálů přičemž $N \cap T = \emptyset$.*

[1] Pokud nebude uvedeno jinak, budu v rámci této práce pod pojmem gramatika předpokládat vždy gramatiku bezkontextovou.

- *K je abeceda dotazovacích symbolů přičemž $(N \cup T) \cap K = \emptyset$.*
- *$S_i \in N$ je počáteční neterminál i-té komponenty.*
- *$P_1, \ldots, P_n$ jsou konečné množiny přepisovacích pravidel nad $N \cup T \cup K$, nazývané komponenty.*
- *Abeceda $N \cup T \cup K$ pak bude značena jako V.*

Abeceda K pak obsahuje právě n symbolů, kde n značí počet komponent daného systému. Každý symbol této abecedy jednoznačně identifikuje právě jednu komponentu systému. Stejně tak je pak každá komponenta jednoznačně identifikována právě jedním symbolem této abecedy. Symboly této abecedy se nazývají dotazovací symboly a značí se jako $K = \{Q_1, \ldots, Q_n\}$, kde n je počet komponent daného systému a dolní index Q_i pak určuje přiřazenou komponentu.

Definice 25 (Konfigurace)**.** *Nechť Γ je PCGS stupně n. N-tice $(x_1, \ldots, x_n)$, kde $x_i \in V_\Gamma^*$ pro všechna i taková, že $1 \leq i \leq n$ se nazývá konfigurace*[2] *systému Γ.*

Definice 26 (Derivační krok)**.** *Mějme PCGS Γ a dvě konfigurace tohoto systému $(x_1, \ldots, x_n)$ a $(y_1, \ldots, y_n)$, přičemž $x_1 \notin T^*$. Pak zapisujeme $(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n)$ pokud platí jedna z následujících dvou podmínek:*

1. *Pro každé $i, 1 \leq i \leq n$ platí $|x_i|_K = 0$, kde zápis $|x_i|_K$ značí počet výskytů symbolů z abecedy K v řetězci x_i. A pro každé x_i a y_i existuje v komponentě i pravidlo tvaru $x_i \Rightarrow y_i$, nebo $x_i = y_i \in T^*$.*
2. *Existuje $i, 1 \leq i \leq n$ takové, že $|x_i|_K > 0$. Pak nechť pro každé takové $i, x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}, t \geq 1, z_j \in (N \cup T)^*, 1 \leq j \leq t+1$. Pokud $|x_{i_j}|_K = 0$ pro všechna $j, 1 \leq j \leq t$ pak $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ a $y_{i_j} = S_{i_j}, 1 \leq j \leq t$. Pokud pro nějaké $j, 1 \leq j \leq t, |x_{i_j}|_K \neq 0$ pak $y_i = x_i$. Pro všechna $i, 1 \leq i \leq n$, nespecifikovaná dříve v tomto bodě platí, že $y_i = x_i$.*

První podmínka se zabývá případem, kdy se v žádné aktuální větné formě jednotlivých komponent nevyskytuje žádný dotazovací symbol. V takovém případě pracují jednotlivé komponenty obdobně, jako by se jednalo o klasické gramatiky – v každé komponentě, která obsahuje nějaký neterminál je jeden takový neterminál přepsán dle některého z pravidel této komponenty. K tomuto přepsání dochází synchronizovaně ve stejný okamžik ve všech komponentách. Komponenty, jejichž větná forma je zároveň větou, pak samozřejmě žádná přepsání neprovádí.

Druhá podmínka se zabývá případem, kdy se v některé z větných forem vyskytuje alespoň jeden dotazovací symbol. V tom případě provede gramatický systém takzvaný komunikační krok. Při tomto kroku se pozastavuje činnost všech komponent. Komponenta, která obsahuje dotazovací symbol, nahradí tento symbol aktuální větnou formou komponenty, která je k tomuto symbolu přiřazena. Tato komponenta pak v dalším kroku restartuje svůj výpočet opět od počátečního symbolu. K tomuto přepsání dochází pouze v případě, že větná forma, kterou bude příslušný symbol nahrazen, sama neobsahuje žádný dotazovací symbol. Pokud takovýto symbol obsahuje, je nejprve nutné, aby sama provedla jeho nahrazení. V případě, že takovéto nahrazení není možno provést z důvodu křížových závislostí, dochází

[2] Alternativně je také možno se setkat s pojmem multiforma, např v [3].

k zaseknutí celého systému a není vygenerován žádný výstupní řetězec. Tento druh derivačního kroku má vyšší prioritu než standardní derivační kroky popsané první podmínkou. Pokud se tedy v některé větné formě nalézá dotazovací symbol, musí být použit tento typ derivačního kroku.

Výše popsaný PCGS je tzv. synchronizovaný. Všechny komponenty provádějí přepsání jednotlivých neterminálů ve stejném kroku. Existují také nesynchronizované PCGS. Tyto systémy nevyžadují, aby provedla každá komponenta, která obsahuje nějaké neterminály přepsání některého z těchto neterminálů. Toto rozšíření odpovídá přidání pravidel tvaru $A \to A$ pro každý neterminál v každé komponentě.

Definice 27 (Jazyk generovaný PCGS)**.** *Nechť je* Γ *PCGS tak jak byl popsán výše. Jazyk, generovaná tímto systémem je:*$L(\Gamma) = \{x \in T^* | (S_1, S_2, \dots S_n) \Rightarrow (x, \alpha_2, \dots \alpha_n), \alpha_i \in V_\Gamma^*, 2 \leq i \leq n\}$. Jakmile tedy vygeneruje gramatika číslo jedna větu, systém končí svoji činnost nezávisle na větných formách ostatních gramatik a tato věta je pak považována za řetězec vygenerovaný tímto systémem. Tato první gramatika se pak označuje jako *master gramatika.*

Definice 28 (Centralizovaný PCGS)**.** *Pokud platí, že dotazovací symboly může generovat pouze master komponenta, nazývá se takový systém centralizovaný. Mohou-li dotazovací symboly generovat libovolné komponenty, je tento systém decentralizovaný.*

Formálně pak pro centralizovaný systém platí, že $P_i \subseteq (N \cup T)^* \times (N \cup T)^*$ *(pro obecný PCGS bez bližší specifikace třídy použitých gramatik), pro* $2 \leq i \leq n$.

Definice 29 (PCGS s návratem)**.** *PCGS pracuje s návratem, pokud po dosazení větné formy dojde k restartu činnosti nahrazující komponenty od počátečního neterminálu. Systém tedy pracuje tak jak byl popsán výše.*

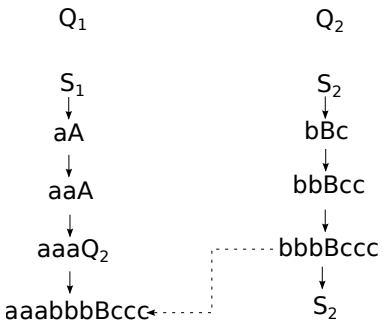
Definice 30 (PCGS bez návratu)**.** *PCGS pracuje bez návratu, pokud po dosazení větné formy pokračuje nahrazující komponenta svůj výpočet právě od této větné formy a nepřepíše ji na svůj počáteční symbol.*

Příklad 1. *Nechť* $\Gamma = (N, T, K, (S_1, P_1), (S_2, P_2))$ *je PCGS s návratem, kde:*

- $N = \{S_1, S_2, A, B\}$
- $T = \{a, b, c\}$
- $K = \{Q_1, Q_2\}$
- $P_1 = \{1 : S_1 \to aA, 2 : A \to aA, 3 : A \to Q_2, 4 : B \to \varepsilon\}$
- $P_2 = \{5 : S_2 \to bBc, 6 : B \to bBc, \}$

Tento systém pracuje tím způsobem, že v každém kroku, přidá první komponenta ke své větné formě jeden symbol a a zároveň druhá komponenta rozšiřuje svou větnou formu o symboly b a c. V daném kroku je celkový počet jednotlivých terminálů stejný. Takto pokračuje rozšiřování jednotlivých řetězců do té doby, dokud první komponenta nepoužije pravidlo 3. *V tuto chvíli obě komponenty zastaví svou činnost. Dotazovací symbol ve větné formě první komponenty je nahrazen větnou formou druhé komponenty, tedy řetězcem* $b^i B c^i$. *Aktuální větná forma druhé komponenty je přepsána na její počáteční symbol. Nyní obsahuje první komponenta jediný neterminál* B, *který v dalším kroku odstraní. Větná forma druhé komponenty je v tomto okamžiku již nadále bezvýznamná. Činnost tohoto systému je znázorněna obrázkem 2.1.*

■



Obrázek 2.1: Příklad 1, činnost PCGS .

2.5.2 Multigenerativní gramatické systémy

Multigenerativní gramatické systémy jsou modifikací PCGS. Těmito gramatickými systémy se zabývá například [3] či [5]. Na rozdíl od klasických PCGS, nepoužívají tyto systémy abecedu dotazovacích symbolů. Jednotlivé komponenty systému pracují opět paralelně, každá nad vlastní větnou formou. Oproti standardním PCGS je zde upraven způsob derivace jednotlivých komponent. Existují dvě základní varianty těchto systémů:

1. n-generativní neterminálově synchronizovaný gramatický systém
2. n-generativní pravidlově synchronizovaný gramatický systém.

Oba tyto systémy obsahují kromě výše popsaných prvků množinu Q. Tato množina obsahuje řetězce, které určují, jakým způsobem budou prováděny jednotlivé derivační kroky v rámci jednotlivých komponent. Zde popsané gramatické systému jsou tzv. kanonické, což znamená, že pracují vždy s nejlevějším neterminálem. Obecné verze jsou popsány v [3].

V případě neterminálově synchronizovaných systémů obsahuje množina Q řetězce nad abecedou neterminálních symbolů. Derivační krok pak probíhá ve dvou krocích. Nejprve je sestaven řetězec z nejlevějších neterminálů jednotlivých komponent. Pokud se tento řetězec nalézá v jazyce Q, jsou na tyto neterminály standardním způsobem aplikována pravidla jednotlivých komponent. V opačném případě dojde k zaseknutí systému.

U pravidlově synchronizovaných systémů jsou prvky množiny Q n-tice o délce rovné počtu komponent systému. Prvky těchto n-tic jsou pak pravidla jednotlivých komponent a to taková, že na pozici i je vždy pravidlo komponenty i. Derivační krok je pak opět proveden ve dvou krocích, nejprve je vybrán jeden prvek z množiny Q. Následně jsou na nejlevější neterminály větných forem jednotlivých komponent aplikována pravidla dle této vybrané n-tice.

Výsledný jazyk je pak opět jazyk generovaný první komponentou. Kromě toho, zavádí [3] i další metody generování jazyků pomocí těchto systémů. Tyto metody využívají věty generované ostatními komponentami systému. Výsledný jazyk je pak vytvořen například konkatenací, případně sjednocením jazyků generovaných jednotlivými komponentami.

2.5.3 CD gramatické systémy

V CD (Cooperating distributed) systémech pracují jednotlivé komponenty sekvenčně, nad společnou větnou formou. V daném momentě je vždy právě jedna gramatika aktivní[3]. To,

[3]Existují modifikace CD systémů, kde toto nemusí být zcela pravdivé, např. CD grammar systems with teams [6].

v jakém pořadí jednotlivé komponenty pracují, určuje kooperační protokol. Kdy má aktivní komponenta ukončit svůj výpočet je pak dáno tzv. módem derivace.

Systém pracuje tím způsobem, že je nejprve vybrána jedna z komponent. V základní verzi CD systémů je tento výběr náhodný. Tato vybraná komponenta provede určitý počet derivačních kroků nad danou větnou formou. Počet derivačních kroků, které komponenta provede, je dán módem derivace. Jakmile je proveden požadovaný počet derivačních kroků, je komponenta deaktivována a stejným způsobem je vybrána komponenta další.

Definice 31 (CD gramatický systém)**.** *CD gramatický systém* Γ *(dále značen jako CDGS) stupně* $n, n \geq 1$ *je k-tice* $\Gamma = (N, T, S, P_1, P_2, \ldots, P_n)$*, kde:*

- N *je abeceda neterminálů*
- T *je abeceda terminálů přičemž* $N \cap T = \emptyset$
- $S \in N$ *je počáteční neterminál*
- $P_1, \ldots, P_n$ *jsou konečné množiny přepisovacích pravidel nad* $N \cup T$*, nazývané komponenty.*

Módy derivace

Dle [6] rozlišujeme následující módy derivace:
Nechť $\Gamma = (N, T, S, P_1, \ldots, P_n)$ je CDGS:

Definice 32 (Normální mód)**.** *Pro každé* $i \in \{1, \ldots, n\}$*, je derivace v normálním módu (značeno* $\Rightarrow^*_{P_i}$*) definována jako:* $x \Rightarrow^*_{P_i} y \Leftrightarrow$ *existuje posloupnost pravidel* $p_1, \ldots, p_n \in P_i$ *taková, že* $x \Rightarrow_{p_1} x_1 \Rightarrow_{p_2} \cdots \Rightarrow_{p_i} y$*, kde* x *a* y *jsou řetězce nad* $N \cup T$*.*

Komponenta pracující v tomto módu není žádným způsobem limitována. Před svou deaktivací může použít libovolný počet pravidel.

Definice 33 (Terminační mód)**.** *Pro každé* $i \in \{1, \ldots, n\}$*, je derivace v terminačním módu (značeno* $\Rightarrow^t_{P_i}$*) definována jako:* $x \Rightarrow^t_{P_i} y \Leftrightarrow x \Rightarrow^*_{P_i} y$ *a zároveň neexistuje žádný řetězec* z *a pravidlo* $p \in P_i$ *takové, že* $y \Rightarrow_p z$*.*

Při tomto módu derivace musí aktivní komponenta přepsat všechny neterminální symboly větné formy, jež se objevují na levé straně některého z pravidel této komponenty. Po ukončení činnosti této komponenty se tedy ve větné formě neobjevuje žádný neterminální symbol, který by se zároveň objevoval i na levé straně pravidla právě deaktivované komponenty. Následně aktivované komponenty samozřejmě mohou takové symboly do větné formy opět zavést.

Definice 34 (Mód k-kroků)**.** *Pro každé* $i \in \{1, \ldots, n\}$*, je derivace v módu* k*-kroků (značeno* $\Rightarrow^{=k}_{P_i}$*) definována jako:* $x \Rightarrow^{=k}_{P_i} y \Leftrightarrow$ *existuje posloupnost* $x_1, \ldots, x_{k+1} \in (N \cup T)^*$ *taková, že* $x = x_1, y = x_{k+1}$ *a pro každé* $1 \leq j \leq k$ *platí* $x_j \Rightarrow_{P_i} x_{j+1}$*.*

Komponenta pracující v tomto módu musí použít právě k pravidel. Tento požadavek se vztahuji i na případ, kdy by aktivní komponenta byla schopna přepsat větnou formu na větu v méně než k krocích.

Definice 35 (Mód nejvíce k-kroků)**.** *Pro každé* $i \in \{1, \ldots, n\}$*, je derivace v módu nejvíce* k*-kroků (značeno* $\Rightarrow^{\leq k}_{P_i}$*) definována jako:* $x \Rightarrow^{\leq k}_{P_i} y \Leftrightarrow x \Rightarrow^{k'}_{P_i} y$ *pro nějaké* $k' \leq k$*.*

Obdobně jako v předchozím případě, komponenta pracující v tomto módu musí vykonat k nebo méně kroků.

Definice 36 (Mód nejméně k-kroků)**.** *Pro každé $i \in \{1, \ldots, n\}$, je derivace v módu nejvíce k-kroků (značeno $\Rightarrow_{P_i}^{\geq k}$) definována jako: $x \Rightarrow_{P_i}^{\geq k} y \Leftrightarrow x \Rightarrow_{P_i}^{k'} y$ pro nějaké $k' \geq k$.*

Tento mód derivaci požaduje, aby aktivní komponenta provedla před svou deaktivací alespoň k kroků. Obdobně jako v módu $= k$, platí tento požadavek i v případě, kdy by byla komponenta schopna vytvořit větu i v méně krocích.

Definice 37 (Generovaný jazyk)**.** *Nechť $D = \{*, t\} \cup \{\leq k, \geq k = k | k \geq 1\}$. Jazyk, generovaný gramatickým systémem Γ, pracujícím v módu derivace $f \in D$ je:*

$$L_f(\Gamma) = \{w \in T^* | S \Rightarrow_{P_{i1}}^{f} w_1 \Rightarrow_{P_{i2}}^{f} \cdots \Rightarrow_{P_{im}}^{f} w_m = w, m \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq m\}$$

Generovaný jazyk se pak liší v závislosti na zvoleném módu derivace.

Příklad 2. *Nechť $\Gamma = \{N, T, S, P_1, P_2\}$ je CD gramatický systém, kde:*

- $N = \{S, A, A', B, B'\}$
- $T = \{a, b, c\}$
- $P_1 = \{1 : S \to S, 2 : S \to AB, 3 : A' \to A, 4 : B' \to B\}$
- $P_2 = \{5 : A \to aA'b, 6 : B \to cB', 7 : A \to ab, 8 : B \to c\}$

Pracuje-li tento systém v $*$-módu, pak jazyk generovaný tímto systémem je $L_1 = \{w | w = a^n b^n c^m,$ pro $n, m \geq 1\}$. Jelikož tento mód umožňuje jednotlivým komponentám pracovat po libovolný počet kroků, nedochází tak mezi komponentami k žádné formě synchronizace. Výsledný jazyk je pak bezkontextový.

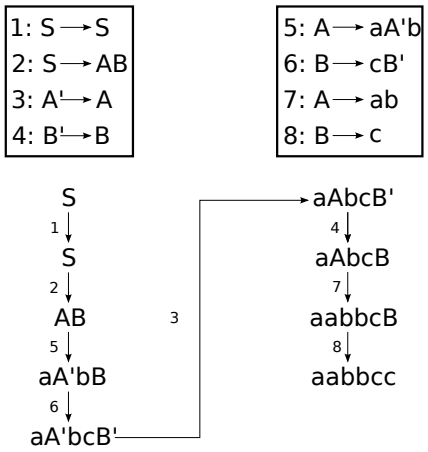
Při použití t-módu začíná výpočet první komponenta. Tato komponenta může nejprve přepisovat neterminál S opět na S pomocí pravidla 1. Jelikož ovšem musí časem ukončit svou činnost, v určité bodě použije pravidlo číslo 2. Nyní se již ve větné formě nenachází žádný neterminál přepsatelný první komponentou a ta se tedy deaktivuje. Druhá komponenta nyní může přepsat symbol A pomocí pravidla 5, případně pomocí pravidla 7. Při použití pravidla 7 dojde k zastavení generování symbolů a a b. Naopak při použití pravidla 5 bude generování těchto symbolů pokračovat alespoň po dobu příští aktivace této komponenty. Obdobný vztah platí i pro symbol B a pravidla 6 a 8. Jelikož nemá žádná z komponent, nějakým způsobem určeno kolik přesně derivačních kroků musí provést, může druhá komponenta zvolit možnost pokračování generování pro jeden ze symbolů a ukončit generování pro symbol druhý. Výsledný jazyk je pak opět $L_2 = \{w | w = a^n b^n c^m,$ pro $n, m \geq 1\}$.

V libovolné větné formě jazyka generovaného tímto systémem (bez ohledu na použitý mód derivace) se mohou vyskytovat 0,1 nebo 2 neterminály. Při použití jednotlivých verzí k-módů se tedy zaměříme na tyto hodnoty.

Je zřejmé, že varianta $\geq k$, pro $k = 1$ odpovídá činnosti $*$-módu. Obdobně pak $\leq k$ pro $k \geq 2$. Při použití $= k$ pro $k = 1$ se jednotlivé komponenty deaktivují po jednom provedeném derivačním kroku. Toto opět nevede k žádné formě synchronizace, jelikož druhá komponenta si opět může vybrat, který z neterminálů přepíše a použije-li pravidlo, které ukončí generování či pravidlo, které umožní generovat i při další aktivaci této komponenty.

Při použití $k = 2$ a módu $= k$ ovšem nastává změna. První komponenta tak nejprve vygeneruje řetězec AB, pravidlo 1 pak slouží k dodržení podmínky dvou derivačních kroků právě pro tuto první aktivaci první komponenty (to, že tato komponenta může opět několik prvních aktivací používat stále jen první pravidlo, neuvažujme). Následně je aktivována

komponenta druhá. Ta nyní musí použít kombinaci pravidel 5 *a* 6*, nebo kombinaci* 7, 8*. Pokud by totiž použila například kombinaci* 5 *a* 7*, nacházel by se těsně po její deaktivaci ve větné formě jediný neterminál. První komponenta ovšem musí provést dva derivační kroky, což při existenci jediného neterminálu není možné. Jazyk, generovaný při použití tohoto derivačního modu je tedy* $L_3 = \{w | w = a^n b^n c^n,\ pro\ n \geq 1\}$*, což již není jazyk bezkontextový. Činnost tohoto systému v módu* $= 2$ *je pak znázorněna na obrázku 2.2.* ■



Obrázek 2.2: Příklad 2, činnost CDGS v módu $= 2$.

2.5.4 Hybridní CD gramatické systémy

Hybridní CD gramatické systémy jsou modifikací gramatických systémů popsaných výše. Na rozdíl od standardních CD systémů, kde je mód derivace svázán s celým systémem, umožňují hybridní systémy přiřadit mód derivace k jednotlivým komponentám.

Variantou hybridních systémů jsou pak tzv. vnitřně hybridní gramatické systémy (internally hybrid). Těmito systémy se zabývá například [1]. Tyto systémy opět přiřazují módy derivace k jednotlivým komponentám. Zavádějí ovšem další nové módy derivace, které vznikly kombinací módů již existujících. Konkrétně pak kombinací terminačního módu s některou z variant k-módů, případě kombinací více k-módů. Tyto nové módy se pak zapisují například jako $(t\wedge = k)$. Komponenta pracující v tomto typu derivačního módu, pak musí provést přepsání všech neterminálů, které je schopna přepsat a zároveň tak musí učinit právě v k krocích.

Kapitola 3

Navržený systém

Tato kapitola popisuje navržený gramatický systém[1]. Tento systém vychází z CD gramatických systémů. Jednotlivé komponenty tedy pracují sekvenčně nad společnou větnou formou. Oproti standardním CD gramatickým systémům došlo ke dvěma modifikacím. První modifikace upravuje způsob výběru jednotlivých komponent. Jednotlivé komponenty jsou zde vybírány pomocí speciální řídicí komponenty. Druhá modifikace pak upravuje módy derivace. Podobně jako v hybridních systémech, mód derivace nebude dále určen pro systém jako takový, místo toho bude přesunut na úroveň jednotlivých komponent. Mimoto budou zavedeny i některé nové módy derivace.

3.1 Aktivace komponent

Navržený systém se skládá z jedné řídicí komponenty (bude dále značena indexem 0) a n komponent řízených. Řídicí komponenta se nepodílí přímo na generování výsledného řetězce. Tato komponenta pouze určuje, v jakém pořadí, přesněji, pro kterou část generovaného řetězce, budou jednotlivé řízené komponenty aktivovány. Toho je docíleno tím způsobem, že tato komponenta má vlastní abecedy neterminálů a terminálů. Abeceda terminálů řídicí komponenty pak odpovídá množině, která vznikne sjednocením jednoprvkových množin, které obsahují počáteční symboly jednotlivých řízených komponent. Počátečním symbolem celého systému je pak počáteční symbol řídicí komponenty. Jazyk, který generuje řídicí komponenta jako taková, pak tedy určuje pořadí aktivace jednotlivých řízených komponent.

Formálně je pak CDGS Γ s touto formou řízení definován jako

$$\Gamma = (N, T, S, (N_0, T_0, S, P_0), (P_1, S_1), (P_2, S_2), \ldots, (P_n, S_n)),$$

kde:

- N_0 je abeceda neterminálů řídicí gramatiky.
- T_0 je abeceda terminálů řídicí gramatiky. $T_0 = \bigcup_{i=1}^{n} \{S_i\}$, $N_0 \cap T_0 = \emptyset$.
- $S \in N_0$ je počáteční neterminál celého systému a zároveň i řídicí komponenty.
- P_0 je množina přepisovacích pravidel typu $P_0 \subseteq N_0 \times (N_0 \cup T_0)^*$.

[1] Nebude-li uvedeno jinak, budu nadále předpokládat pod pojmem *gramatický systém* právě tuto navrhovanou variantu.

- $N \cap T = \emptyset$.
- P_i pro všechna $i, 1 \leq i \leq n$ je množina přepisovacích pravidel tvaru $P_i \subseteq N \times (N \cup T)^*$. Jedná se o pravidla jednotlivých řízených komponent.
- $S_i \in N$ pro všechna $i, 1 \leq i \leq n$ jsou počáteční symboly jednotlivých řízených komponent.
- Pro jednotlivé abecedy pak platí že $(N \cup T) \cap N_0 = \emptyset$.

Příklad 3. *Předpokládejme CDGS* Γ *pracující výše popsaným způsobem, který má dvě řízené komponenty, přičemž platí:*

- $T = \{a, b, c\}$
- $T_0 = \{A, B\}$
- $S_1 = A, S_2 = B$
- $P_0 = \{1 : S \to ABS, 2 : S \to AB\}$
- $P_1 = \{3 : A \to aAb, 4 : A \to ab\}$
- $P_2 = \{5 : B \to cB, 6 : B \to c\}$

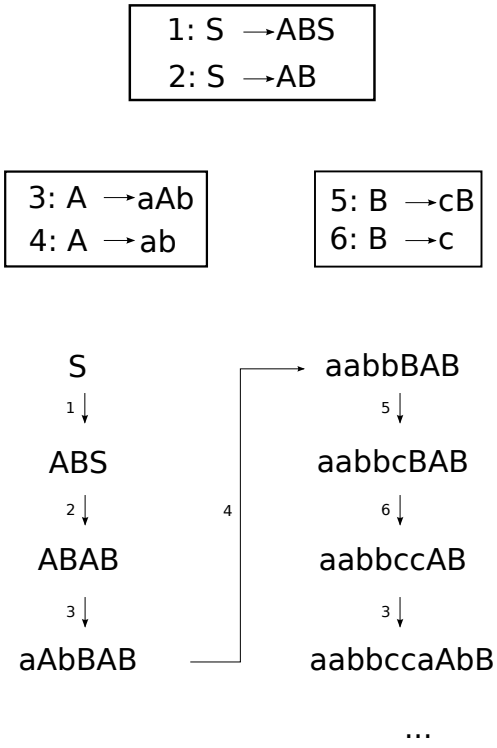
Tento systém pak bude pracovat tím způsobem, že řídicí komponenta nejprve vygeneruje řetězec počátečních symbolů pro jednotlivé komponenty. Tento řetězec je generován pomocí pravidel 1 *a* 2*. Na základě takto vygenerovaného řetězce, jsou pak postupně aktivovány jednotlivé komponenty. Pro jednoduchost předpokládejme, že je vždy vybrán nejlevější počáteční symbol, v tomto případě ovšem na pořadí aktivace jednotlivých komponent nezáleží. Tyto komponenty pak přepíší řetězec počátečních symbolů na výslednou větu celého systému. V tomto případě pak generuje systém jazyk* $L = \{w | w = a^{n_1} b^{n_1} c^{m_1} \ldots a^{n_i} b^{n_i} c^{m_i}; i, n, m \geq 1\}$*. Činnost tohoto systému je pak znázorněna na obrázku 3.1.*

■

3.2 Módy derivace

Obdobně jako v případě hybridních CDGS, přiřazuje navržený gramatický systém módy derivace jednotlivým komponentám. Na rozdíl od hybridních systémů, v tomto případě je ovšem mód svázán nejen s komponentou jako takovou, ale s konkrétní aktivací této komponenty. Toto vychází ze způsobu, jakým jsou jednotlivé komponenty aktivovány. Mód derivace je pak předepsán v rámci počátečních neterminálů, na pravých stranách pravidel řídicí komponenty. Předpokládejme, že D je množina možných módů derivace. Zavedeme pak zobrazení d, které přiřazuje každému pravidlu z P_0 vektor, jehož délka je rovna počtu počátečních symbolů na pravé straně příslušného pravidla a jehož prvky jsou z množiny D. Tento vektor pak určuje mód derivace komponenty, která svůj výpočet začne z odpovídajícího počátečního symbolu.

V případě, že je daná komponenta aktivována jinak, než pomocí komponenty řídicí (tedy některou z řízených komponent), bude mód derivace šířen dále, směrem z počátečního symbolu, který k této aktivaci vedl.



Obrázek 3.1: Aktivace jednotlivých komponent

Definice 38 (Zobrazení d)**.** *Mějme gramatický systém* Γ *popsaný výše. Zobrazení (přesněji funkce)* d *je pak dáno jako* $d \subseteq P_0 \times D^*$*, pro konkrétní pravidlo* p_j *pak platí, že* $d(p_j) \in D^n$ $n = |p_j|_T$*, kde* $|p_j|_T$ *značí počet terminálů řídicí komponenty na pravé straně pravidla* $p_j \in P_i$*.*

Pro jednoduchost budu značit mód derivace příslušející danému počátečnímu symbolu pomocí pravého horního indexu u tohoto neterminálu a nebudu tak přímo uvádět zobrazení d v rámci definice gramatických systémů.

Příklad 4. *Předpokládejme CDGS* Γ *pracující výše popsaným způsobem, který má dvě řízené komponenty, přičemž platí:*

- $T = \{a, b, c\}$
- $T_0 = \{A, B\}$
- $S_1 = A, S_2 = B$
- $P_0 = \{1 : S \to AB\}$
- $P_1 = \{2 : A \to aAb, 3 : A \to ab\}$
- $P_2 = \{4 : B \to cB, 5 : B \to c\}$
- *Přičemž pro módy derivace platí:* $\{1 \to (t, t\wedge = 2)\}$

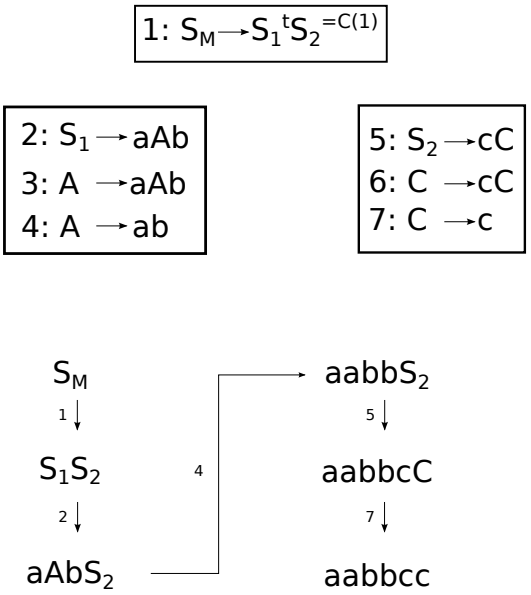
Řídicí komponenta nejprve vygeneruje řetězec $A^tB^{t\wedge=2}$. První komponenta pak bude pracovat v terminačním módu, druhá ve vnitřně hybridním módu $t\wedge = 2$. První komponenta nejprve vytvoří řetězec symbolů a následovaných symboly b, přičemž počet těchto symbolů je shodný. Druhá komponenta pak provede právě dva kroky, ve kterých provede vygenerování řetězce cc. Jazyk generovaný tímto systémem pak je $L = \{w|w = a^nb^ncc, n \geq 1\}$. ■

Kromě dříve uvedených módů derivace zavádím i módy nové. Tyto módy využívají přesunutí platnosti módu derivace na úroveň jednotlivých počátečních symbolů jednotlivých komponent. Tyto nové módy, značené jako $= C(i), \leq C(i), \geq C(i)$, neudávají konkrétní délku výpočtu dané komponenty. Místo toho se odkazují na komponentu jinou, jejíž počáteční symbol se nalézá ve stejném pravidle. Komponenta, pracující v některém z těchto C-módů, pak pracuje vlastně v odpovídajícím $t \wedge k$-módu, kde toto k, je dáno počtem derivačních kroků, které provedla komponenta začínající svůj výpočet z i-tého počátečního symbolu (počítáno v rámci pravé strany pravidla). Tato odkazovaná komponenta pak musí pracovat v některé z variant terminačního módu. V rámci navrženého systému pak připouštím pouze vnitřně hybridní varianty k-módů, tedy $t\wedge = k$, $t\wedge \leq k$ a $t\wedge \geq k$, módy t a $*$ a nově zavedené $C(i)$ módy.

Obdobně jako v případě PCGS, může i zde dojít k zaseknutí celého systému, a to v případě křížových odkazů, kdy množina komponent používá některý z C-módů, a jako odkazovanou komponentu využívá opět některou z komponent v této množině.

Jelikož se komponenty v těchto módech odkazují na komponenty jiné, je třeba zajistit správné provázání při přepsání části větné formy. Pro danou větnou formu je sestavena j-tice δ, jejíž délka je rovna počtu neterminálů (neterminálů řízených komponent, přesněji) v dané větné formě. Jednotlivé prvky této j-tice pak určují derivační mód odpovídající komponenty. V rámci derivačního kroku je určen jeden neterminál, který bude přepsán. Všechny položky této j-tice, které se nalézají před tímto bodem, zůstanou ponechány. Po vykonání derivačního kroku, je určen počet počátečních neterminálů, které se nachází v podřetězci, který byl v rámci tohoto derivačního kroku vygenerován. O tuto hodnotu jsou pak aktualizovány všechny reference na komponenty, které se nalézají za tímto zvoleným neterminálem. Všechny neterminály, které nově vzniknou přepsáním zvoleného neterminálu, pak budou mít přiřazeny stejný mód derivace jako původní přepsaný neterminál.

Příklad 5. *Předpokládejme gramatický systém z příkladu 4 s módy derivace danými jako $\{1 \rightarrow (t, = C(1))\}$. Řídicí komponenta nejprve opět vygeneruje řetězec počátečních symbolů. Následně je spuštěna první komponenta. Tato komponenta pracuje v terminačním módu, provede tedy libovolný počet kroků, ovšem ve větné formě nezanechá žádný neterminál, který by byla schopna dále přepsat. Následně je spuštěna komponenta druhá. Tato komponenta pak musí provést stejný počet kroků jako komponenta první. Jazyk generovaný tímto systém tedy je $L = \{w|w = a^nb^nc^n; 1 \leq n\}$. Vygenerování řetězce aabbcc tímto systémem je znázorněno na obrázku 3.2.*



Obrázek 3.2:

■

Definice 39 (Konfigurace)**.** *Předpokládejme, že D je množina možných módů derivace. Konfigurace daného systému $\Gamma = (N, T, S, (N_0, T_0, S, P_0), (P_1, S_1), (P_2, S_2) \ldots, (P_n, S_n))$ je dvojice (χ, δ), kde:*

- $\chi \in (N \cup T \cup N_0 \cup T_0)^*$
- $\delta = (f_1, \ldots, f_k)$*, kde* $f_j \in D, 1 \leq j \leq k$ *a* $k = |\chi|_N$

Definice 40 (Derivační krok)**.** *Mějme gramatický systém*

$$\Gamma = (N, T, S, (N_0, T_0, S, P_0), (P_1, S_1), (P_2, S_2), \ldots, (P_n, S_n))$$

a dvě jeho konfigurace $\kappa_1 = (\chi_1, \delta_1)$ a $\kappa_2 = (\chi_2, \delta_2)$. Systém Γ pak provede derivační krok z κ_1 do κ_2, značeno $\kappa_1 \Rightarrow \kappa_2$ pokud platí, že $\chi_1 = \alpha A \gamma$, $\chi_2 = \alpha\beta\gamma$, $\delta_1 = (f_1, \ldots, f_{|\chi_1|_N}), A \in N$, $\delta_2 = (f'_1, \ldots, f'_{|\alpha|_N}, f'_{|\alpha|_N+1}, \ldots, f'_{|\alpha|_N+|\beta|_N}, f'_{|\alpha|_N+|\beta|_N+1}, \ldots, f'_{|\chi_2|_N})$ kde[2]:

[2]Módem C(l) resp. K zde označuji libovolnou variantu $(\leq, \geq, =)$, přičemž se tato varianta samozřejmě nesmí v rámci f a f' změnit.

$$f_i' = \begin{cases} f_{|\alpha|_N+1}, & \text{pro } |\alpha|_N+1 \leq i \leq |\alpha|_N+|\beta|_N \text{ pokud } A \notin T_0 \\ \text{dle zobrazení } d, & \text{pro } |\alpha|_N+1 \leq i \leq |\alpha|_N+|\beta|_N \text{ pokud } A \in T_0 \\ f_{i-|\beta|_N}, & \text{pro } |\alpha|_N+|\beta|_N+1 \leq i \leq |\chi_2|_N \text{ pokud } f_{i-|\beta|_N} \\ & \text{není žádnou variantou } C \text{ módu} \\ C(l+|\beta|_N), & \text{pro } i \leq |\alpha|_N \text{ pokud } f_i \\ & \text{odpovídá módu } C(l) \wedge l > |\alpha|_N+1 \\ f_i, & \text{pro } 1 \leq i \leq |\alpha| \text{ pokud } f_i \\ & \text{není žádnou variantou } C \text{ módu,} \\ & \text{nebo pokud je variantou } C(l) \text{ módu a } l \leq |\alpha|_N \\ t \wedge K & \text{pro } |\alpha|_N+|\beta|_N+1 \leq i \leq |\chi_2|_N \text{ pokud } f_{i-|\beta|_N} \\ & \text{odpovídá módu } C(l) \wedge l = |\alpha|_N+1 \\ t \wedge K & \text{pro } 1 \leq i \leq |\alpha|_N \text{ pokud } f_i \\ & \text{odpovídá módu } C(l) \wedge l = |\alpha|_N+1 \end{cases}$$

a zároveň existuje v rámci komponenty m posloupnost pravidel taková, že $A \Rightarrow_{P_m}^{f_{|\alpha|_N+1}} \beta$, délka této posloupnosti pak určuje hodnotu K v rámci posledních dvou bodů výše popsaného výpočtu f_i'.

3.3 Vlastnosti navrženého systému

Věta 41. *Nechť $\mathcal{L}_*$ je třída jazyků generovaných navrženým systémem, ve kterém pracují všechny komponenty vždy v $*$-módu a $\mathcal{L}_{CF}$ třída bezkontextových jazyků. Pak platí $\mathcal{L}_{CF} \subseteq \mathcal{L}_*$.*

Důkaz. Důkaz bude proveden pomocí algoritmu, který převede libovolnou bezkontextovou gramatiku na ekvivalentní gramatický systém.

Nechť $G = (N, T, S, P)$ je bezkontextová gramatika. Ekvivalentní gramatický systém pak bude mít tvar $\Gamma = (N, T, S', (\{S'\}, \{S\}, S', P_0), (P, S))$, kde $P_0 = \{S' \to S^{(*)}\}$. Tento systém obsahuje pouze jedinou řízenou komponentu, během výpočtu bude tedy vždy po deaktivaci aktivována opět ta stejná komponenta. Jelikož je použit $*$-mód, není kladen žádný požadavek na počet aktivací a deaktivací této komponenty. V určitém čase pak komponenta vygeneruje řetězec terminálů, přičemž počet derivačních kroků, které provedla, je irelevantní. □

Věta 42. *Nechť $\mathcal{L}_*$ je třída jazyků generovaných navrženým systémem, ve kterém pracují všechny komponenty vždy v $*$-módu a $\mathcal{L}_{CF}$ třída bezkontextových jazyků. Pak platí $\mathcal{L}_* \subseteq \mathcal{L}_{CF}$.*

Důkaz. Nechť $\Gamma = (N, T, S, (N_0, T_0, S, P_0), (P_1, S_1), \ldots (P_n, S_n))$ je gramatický systém, kde všechny komponenty pracují vždy v $*$-módu. Ekvivalentní bezkontextová gramatika G pak bude mít tvar $G = (N_G, T_G, S_G, P_G)$, kde:

- $N_G = N \cup N_0 \cup T_0$
- $T_G = T$
- $S_G = S$

- $P_G = \bigcup_{i=0}^{n} P_i$

Jelikož není kladen díky použitému módu derivace žádný požadavek na pořadí aktivace a deaktivace jednotlivých komponent, není třeba ani žádným způsobem upravovat jednotlivé složky jednotlivých komponent. □

Věta 43. *Nechť $\mathcal{L}_t$ je třída jazyků generovaných navrženým systémem, ve kterém pracují všechny komponenty vždy v t-módu a $\mathcal{L}_{CF}$ třída bezkontextových jazyků. Pak platí $\mathcal{L}_{CF} \subseteq \mathcal{L}_t$.*

Důkaz. Důkaz bude opět proveden pomocí algoritmu, který převede libovolnou bezkontextovou gramatiku na ekvivalentní gramatický systém.

Nechť $G = (N, T, S, P)$ je bezkontextová gramatika. Ekvivalentní gramatický systém pak bude mít tvar $\Gamma = (N, T, S', (\{S'\}, \{S\}, S', P_0), (P, S))$, kde $P_0 = \{S' \rightarrow S^{(t)}\}$. Tento systém obsahuje pouze jedinou řízenou komponentu, která odpovídá původní bezkontextové gramatice. Tato komponenta pak pracuje v terminačním módu. To znamená, že pokud obsahovala původní gramatika posloupnost pravidel, která vedla k vygenerování věty, pak i tato komponenta tuto posloupnost obsahuje a je schopna ji provést během jedné své aktivace. Pro libovolný řetězec generovaného jazyka pak platí, že pokud byla původní gramatika schopna tento řetězec vytvořit během i kroků, pak ekvivalentní GS je toho schopen v $i + 1$ krocích. □

Věta 44. *Nechť $\mathcal{L}_t$ je třída jazyků generovaných navrženým systémem, ve kterém pracují všechny komponenty vždy v t-módu a $\mathcal{L}_{CF}$ třída bezkontextových jazyků. Pak platí $\mathcal{L}_t \subseteq \mathcal{L}_{CF}$.*

Důkaz. V rámci důkazu bude opět předveden možný algoritmus pro sestavení ekvivalentní bezkontextové gramatiky $G = (N_G, T_G, S_G, P_G)$ pro daný gramatický systém $\Gamma = (N, T, S, (N_0, T_0, S, P_0), (P_1, S_1), \ldots (P_n, S_n))$, kde pracují jednotlivé komponenty vždy v t-módu. Kvůli tomuto módu derivace je nutné zajistit, že pravidla, která simulují právě aktivní komponentu, nejsou „přerušena“ pravidly, která simulují komponentu jinou.

Pro každou z komponent i zavedeme dvě pomocné množiny $N_i^{'}$ a $P_i^{'}$. Nechť $p_j \in P_i$ pro $\forall p_j \in P_i$ a nechť p_j je tvaru $x_j \rightarrow y_j$. Pak $N_i^{'} = N_i^{'} \cup \{n\}$ tehdy a jen tehdy pokud pro $n \in N$ platí, že n je podřetězcem x_j. Určíme tedy, které neterminály se vyskytují na levé straně některého z pravidel komponenty.

Množinu $P_i^{'}$ pak sestavíme následovně. Pokud pravidlo $p_j \in P_i$ obsahuje některý symbol z $N_i^{'}$, pak toto pravidlo odebereme z P_i, nahradíme všechny výskyty toho symbolu v pravidle neterminálem $n_i^{'}$ (kde i identifikuje danou komponentu) a přidáme takto vzniklé pravidlo do množiny $P_i^{'}$. Tímto bude zajištěno, že pokud byla aktivní komponenta schopna přepsat daný neterminál, bude tento neterminál přepsán pravidlem právě této komponenty a žádné jiné. Pro každý neterminál $n_i^{'} \in N_i^{'}$ pak přidáme do $P_i^{'}$ také pravidlo tvaru $n \rightarrow n_i^{'}$. Toto pravidlo ošetří případy, kdy aktivní komponenta vygeneruje neterminál, který není sama schopna přepsat, ale existují jiné komponenty, které jej jsou schopny přepsat (což platí pro řídicí komponentu a počáteční symboly jednotlivých řízených komponent).

Pro výslednou gramatiku $G = (N_G, T_G, S_G, P_G)$ pak platí:

- $N_G = N \cup N_0 \cup T_0 \cup \bigcup_{i=0}^{n} N_i^{'}$
- $T_G = T$

- $S_G = S$
- $P_G = \bigcup_{i=0}^{n} P_i'$

□

Věta 45. *Nechť $\mathcal{L}_{=C(j)}$ je třída jazyků generovaných navrženým systémem, ve kterém pracuje alespoň jedna komponenta v módu $= C(j)$ a $\mathcal{L}_{CF}$ třída bezkontextových jazyků. Pak platí $\mathcal{L}_{CF} \subseteq \mathcal{L}_{=C(j)}$.*

Důkaz. Nechť $G = (N_G, T_G, S_G, P_G)$ je bezkontextová gramatika, pro ekvivalentní GS $\Gamma = (N, T, S, (N_0, T_0, S', P_0), (P_1, S_1), (P_2, S_2), (P_3, S_3))$, pak platí:

- $S_1 = S_G, P_1 = P_G$
- $P_2 = \{S_2 \to \varepsilon\}$
- $P_3 = \{S_3 \to \varepsilon\}$
- $P_0 = \{S \to S_1^{(t)} S_2^{(t)} S_3^{(=C(2))}\}$
- $N = N_G \cup \{S_1, S_2, S_3\}$
- $N_0 = \{S_0\}$
- $T = T_G$
- $T_0 = \{S_1, S_2, S_3\}$

Komponenta 1 pak provádí samotný výpočet, komponenty 2 a 3 pak nemají pro generování jazyka žádný význam. Podobný důkaz by bylo možno sestavit pro libovolné množství komponent pracujících v $= C(j)$ módu. □

Věta 46. *Nechť $\mathcal{L}_{=C(j)}$ je třída jazyků generovaných navrženým systémem, ve kterém pracuje alespoň jedna komponenta v módu $= C(j)$. Pak existuje jazyk $L \in \mathcal{L}_{=C(j)}$, který není bezkontextový.*

Důkaz. Ukázka takového jazyku je popsána v příkladu 5. Obdobně jako v případě předešlého důkazu, by bylo možno rozšířit tento důkaz pro libovolné větší množství komponent pracujících v módu $= C(j)$. □

Důsledek 47 (Vět 41 a 42)**.** $\mathcal{L}_{CF} = \mathcal{L}_*$

Důsledek 48 (Vět 43 a 44)**.** $\mathcal{L}_{CF} = \mathcal{L}_t$

Důsledek 49 (Vět 45 a 46)**.** $\mathcal{L}_{CF} \subset \mathcal{L}_{=C(j)}$

Důsledek 50 (Vět 45 a 46)**.** $\mathcal{L}_{CF} \subset \mathcal{L}_{\leq C(j)}$

Důsledek 51 (Vět 45 a 46)**.** $\mathcal{L}_{CF} \subset \mathcal{L}_{\geq C(j)}$

Důkaz. Mód $= C(j)$ je zvláštním případem módů $\leq C(j)$ resp. $\geq C(j)$ □

Otevřený problém 1. $\mathcal{L}_{\leq C(j)} \stackrel{?}{=} \mathcal{L}_{=C(k)}$

Otevřený problém 2. $\mathcal{L}_{\geq C(j)} \stackrel{?}{=} \mathcal{L}_{=C(k)}$, *v tomto případě bude pravděpodobně platit rovnost, mód* $\geq C(j)$ *by mohlo jít simulovat pomocí kombinace dvou komponent pracujících v módech* $= C(j)$ $a \geq k$.

Otevřený problém 3. $\mathcal{L}_{\leq C(j)} \stackrel{?}{=} \mathcal{L}_{\geq C(k)}$

Otevřený problém 4. *Platí věta 45 i v případě GS bez možnosti použití* ε*-pravidel?*

Otevřený problém 5. *Vzniká přidáním dalších komponent pracujících v některém z* C*-módů nekonečná hierarchie z pohledu výpočetní síly, nebo se výpočetní síla od určitého počtu takovýchto komponent pro danou variantu nemění?*

Dalšími otevřenými problémy pak jsou například uzávěrové vlastnosti tohoto druhu systémů. Systémy budou pravděpodobně uzavřeny vůči konkatenace a sjednocení (důkaz by vedl na přidání nového počátečního symbolu řídicí gramatiky a pravidel, která by tento symbol přepsala na počáteční symboly původních systémů v požadovaném tvaru). Dále není v této práci zaveden algoritmus, který by prováděl převod jiného druhu modelů na zde představené systémy.

Kapitola 4

Syntaktická analýza

V rámci této kapitoly, budou popsány základní principy existujících metod syntaktické analýzy a tyto metody budou modifikovány na základě gramatických systému představených v kapitole 3. Sekce 4.1 a 4.2 vychází z [2] a [4], kde je možno nalézt jednotlivé, zde popsané algoritmy a metody.

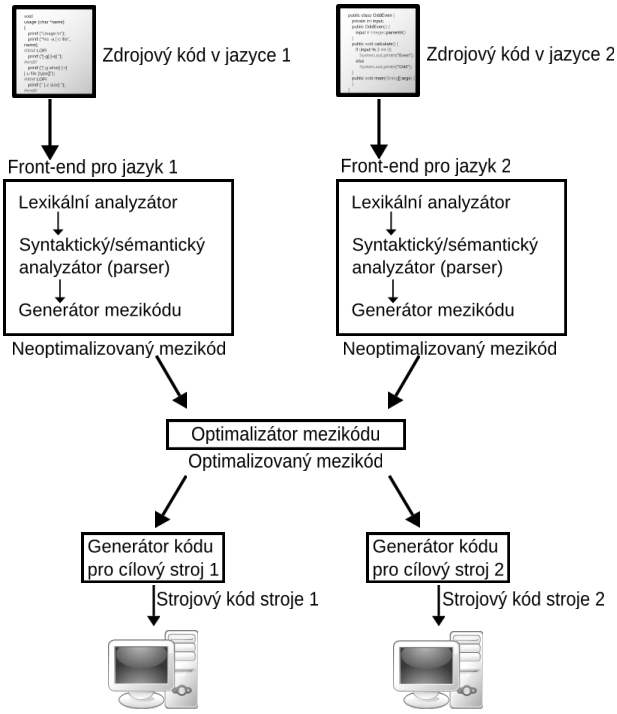
4.1 Obecný úvod do syntaktické analýzy

Syntaktická analýza je jednou z částí překladu řetězce vstupního jazyka, na řetězec jazyka výstupního. Schéma, znázorňující práci typického překladače je na obrázku 4.1. Vstupní řetězec je nejprve zpracován lexikálním analyzátorem. Ten rozdělí vstup na sekvenci symbolů nazývaných tokeny. Tato sekvence je předána syntaktickému analyzátoru nazývanému též parser. Většina syntaktických analyzátorů je postavena na modelu bezkontextových gramatik a jim odpovídajícím zásobníkových automatech. Jednotlivé tokeny, pak v tomto modelu, odpovídají terminálním symbolům příslušné gramatiky[1]. Syntaktická analýza má pak dva hlavní cíle. Prvním je určit, patří-li tento vstupní řetězec do jazyka, popsaného gramatikou, dle které je parser vytvořen, to znamená, že se analyzátor snaží určit, jsou-li jednotlivé tokeny ve vstupním řetězci použity ve správném pořadí a jsou-li správně vzájemně kombinovány. Druhým cílem je určit posloupnost přepisovacích pravidel dané gramatiky, která z počátečního neterminálu vytvoří vstupní řetězec. Tento druhý cíl má pak samozřejmě smysl pouze v tom případě, kdy je splněn i cíl první. Na základě této posloupnosti pravidel, je pak vytvořena jistá vnitřní reprezentace vstupního řetězce, která je pak dále zpracována.

Existují dva základní přístupy k syntaktické analýze. První z nich vychází z počátečního symbolu a aplikuje na něj nějaké přepisovací pravidlo dané gramatiky, čímž vznikne větná forma. Na tuto větnou formu pak dále aplikuje přepisovací pravidla a postupně ji porovnává se vstupním řetězcem. Tento přístup se nazývá analýza shora dolů. Výstupem je pak sekvence aplikovaných přepisovacích pravidel.

Druhý přístup pracuje opačným postupem. Postupně čte vstupní řetězec a snaží se najít pravidla, jejichž pravá strana odpovídá přečtenému podřetězci. Jakmile takové pravidlo nalezne, nahradí tento podřetězec jeho levou stranou. Tímto způsobem postupně redukuje analyzátor vstupní řetězec a snaží se dosáhnout stavu, kdy je celý vstup zredukován na

[1] Důsledně vzato, terminály a tokeny nejsou zcela totožné. Token předaný z lexikálního analyzátoru často obsahuje kromě terminálu samotného nějakou další hodnotu nazývanou *atribut*. Terminálem pak může být například symbol reprezentující číslo a příslušný token pak kromě tohoto symbolu také obsahuje hodnotu tohoto čísla jako svůj atribut. Tento rozdíl je však pro další výklad nepodstatný a nebude proto brán v úvahu.



Obrázek 4.1: Schéma činnosti překladače. Obrázek byl převzat z [7].

počáteční neterminál. Tento přístup se nazývá analýza zdola nahoru. Výstupem je pak opět sekvence pravidel, ovšem tentokrát je tato sekvence v opačném pořadí (poslední pravidlo této sekvence přepisuje počáteční symbol).

4.2 Používané metody

V této části budou popsány některé základní metody syntaktické analýzy. Budou zde popsáni jak zástupci metod pracující shora dolů, tak i zdola nahoru.

4.2.1 Obecný algoritmus pro analýzu shora dolů

Tato metoda vychází z obecného popisu metod pracujících shora dolů, popsaných výše. Syntaktický analyzátor, který pracuje tímto způsobem, je možno sestavit pro libovolnou bezkontextovou gramatiku. Zásobníkový automat, který provádí syntaktickou analýzu shora dolů jazyka $L(G)$, kde $G = (N, T, S, P)$ je bezkontextová gramatika lze sestavit pomocí algoritmu 1.

Automat nejprve umístí na zásobník počáteční symbol gramatiky. Následně pak v každém kroku vybere jedno pravidlo, které použije. Výběr pravidla, je dán symbolem na vrcholu zásobníku. Nalézá-li se zde neterminál (z pohledu vstupní gramatiky), je použito jedno z expanzních pravidel (expansion rule), které provede nahrazení tohoto neterminálu pravou stranou tohoto pravidla, přičemž jednotlivé symboly jsou vkládány na zásobník v opačném pořadí. Tato pravidla jsou tvořena na základě pravidel vstupní gramatiky. Nalézá-li se na vrcholu terminál, je přečten jeden symbol ze vstupní pásky. Shoduje-li se tento přečtený symbol se symbolem na vrcholu zásobníku, je tento symbol ze zásobníku odstraněn pomocí vymazávacího pravidla (poping rule). Pokud se symboly neshodují, končí výpočet

Algoritmus 1: Algoritmus pro tvorbu zásobníkového automatu provádějícího syntaktickou analýzu shora dolů.

Vstup: Bezkontextová gramatika $G = (N, T, S, P)$.
Výstup: Zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$, který provádí syntaktickou analýzu shora dolů.
$Q = \{s\}$
$\Sigma = T$
$\Gamma = T \cup N$
$F = \emptyset$
foreach $a \in \Sigma$ **do**
    Přidej pravidlo tvaru $asa \to s$ do R.
foreach $A \to \alpha \in P$ **do**
    Přidej pravidlo tvaru $As \to rev(\alpha)$ do R.

chybou. Automat se pak snaží dočíst vstupní řetězec a při tom vyprázdnit svůj zásobník. Na výstupu pak produkuje sekvenci použitých expanzních pravidel. Tato činnost je formálně popsána pomocí algoritmu 2. Tento algoritmus popisuje činnost zásobníkového automatu M_{TD}, který provádí syntaktickou analýzu shora dolů jazyku L_{TD} nad abecedou Σ_{TD}, generovaného gramatikou $G_{TD} = \{N_{TD}, \Sigma_{TD}, P_{TD}, S_{TD}\}$.

Výhodou tohoto druhu syntaktické analýzy je, jak už bylo řečeno, že je možno ji použít pro libovolnou bezkontextovou gramatiku. Nevýhodou pak je, že tato metoda nemá zajištěn deterministický průběh. Pro daný neterminál může existovat více pravidel, dle kterých je možno jej rozgenerovat, což je nutno řešit například pomocí backtrackingu. V praxi se proto tato metoda příliš nepoužívá.

Algoritmus 2: Algoritmus popisující činnost automatu provádějícího syntaktickou analýzu shora dolů.

Vstup: $w = x_1 x_2 \dots x_n$, $x_i \in \Sigma_{TD}$, $1 \leq i \leq n$.
Výstup: Úspěch/Neúspěch, sekvence použitých pravidel.
Vlož \$ a S_{TD} na vrchol zásobníku.
while *Zásobník obsahuje nějaký symbol* **do**
    Nechť X značí symbol na vrcholu zásobníku a a aktuální vstupní token.
    switch X *je:* **do**
        case $X = \$$
            if $a = \$$ **then**
                Ukonči algoritmus, a vrať úspěch.
            else
                Ukonči algoritmus, a vrať neúspěch.
        case $X \in T_{TD}$
            if $a = X$ **then**
                Odstraň X z vrcholu zásobníku, a načti další vstupní symbol a.
            else
                Ukonči algoritmus, a vrať neúspěch.
        case $X \in N_{TD}$
            if $r : X \to \alpha \in P_{TD}$ **then**
                Existuje-li více možných pravidel r, vyber jedno nedeterministicky. Nahraď symbol X na vrcholu zásobníku řetězcem $\text{rev}(\alpha)$. Vypiš r na výstup.
            else
                Ukonči algoritmus, a vrať neúspěch.

4.2.2 LL-analýza

Variantou výše popsané metody je LL analýza (první L zde značí čtení vstupu zleva doprava, druhé pak simulaci nejlevější derivace). Jedná se tedy opět o metodu pracující shora dolů. Základní činnost této metody je podobná jako v předešlém případě, liší se však metodou výběru pravidla, dle kterého dojde k rozgenerování neterminálu na vrcholu zásobníku. Tento výběr je realizován pomocí takzvané LL tabulky. Tato tabulka je sestavena pomocí množin (formálně přesněji se jedná o funkce, které generují množiny) *predict*, k jejichž sestavení je nejprve potřeba definovat několik pomocných množin (opět jde o funkce) a algoritmů. Tyto Množiny jsou sestavovány vzhledem ke zpracovávané gramatice $G = (N, T, S, P)$.

První z těchto množin, je množina $first(\alpha)$, kde α je libovolný řetězec nad abecedou všech symbolů gramatiky G. Tato množina určuje terminály, kterými může začínat řetězec, který vznikne derivací řetězce α. Algoritmus, který vytvoří množinu $first(X)$, kde X, je

nějaký symbol gramatiky G, odpovídá algoritmu 3.

Algoritmus 3: Algoritmus pro výpočet množiny $first(X)$.

Vstup: Bezkontextová gramatika $G = (N, T, S, P)$.
Výstup: Množina $first$, pro každý symbol $X \in T \cup N$.
foreach $X \in N \cup T \cup \{\varepsilon\}$ **do**
  if $X \in T \cup \{\varepsilon\}$ **then**
    $first(X) = \{X\}$
  else
    $first(X) = \emptyset$
repeat
  foreach $X \to \alpha \in P$ **do**
    switch α*:* **do**
      case $alpha = \varepsilon$
        $first(X) = first(X) \cup \{\varepsilon\}$
      case $alpha = x_1 x_2 \ldots x_n \varepsilon$
        $first(X) = first(X) \cup (first(x_1) - \{\varepsilon\})$. Pokud $\varepsilon \in first(x_1)$, pak také $first(X) = first(X) \cup (first(x_2) - \{\varepsilon\})$. Pokud $\varepsilon \in (first(x_1) \cap first(x_2))$, pak také $first(X) = first(X) \cup (first(x_3) - \{\varepsilon\})$ atd. (v případě, že se tato sekvence dostane až k poslednímu symbolu ε, pak také $first(X) = first(X) \cup \{\varepsilon\}$).
until *Žádná množina nebyla již dále změněna.*

Na základě algoritmu 3 pak pracuje algoritmus 4, který vypočte množinu $first(\alpha)$ pro libovolný řetězec.

Algoritmus 4: Algoritmus pro výpočet množiny $first(\alpha)$.

Vstup: Bezkontextová gramatika $G = (N, T, S, P)$.
Výstup: Množina $first$, pro řetězec $\alpha \in (T \cup N)^*$.
$first(\alpha) = \emptyset$
switch α *je :* **do**
  case $\alpha = \varepsilon$
    $first(\alpha) = first(\alpha) \cup \{\varepsilon\}$
  case $\alpha = x_1 x_2 \ldots x_n \varepsilon$
    $first(\alpha) = first(\alpha) \cup (first(x_1) - \{\varepsilon\})$. Pokud $\varepsilon \in first(x_1)$, pak také $first(\alpha) = first(\alpha) \cup (first(x_2) - \{\varepsilon\})$. Pokud $\varepsilon \in (first(x_1) \cap first(x_2))$, pak také $first(\alpha) = first(\alpha) \cup (first(x_3) - \{\varepsilon\})$ atd. (v případě, že se tato sekvence dostane až k poslednímu symbolu ε, pak také $first(\alpha) = first(\alpha) \cup \{\varepsilon\}$).

Druhou množinou je pak množina $follow(A)$, kde $A \in N$. Tato množina obsahuje terminály, které se mohou kdykoliv během výpočtu objevit v rámci větné formy bezprostředně za symbolem A. Jedná se tedy o všechny terminály a takové, že existuje posloupnost derivací taková, že $S \to^* \alpha A a \beta$, kde α, β jsou nějaké řetězce nad abecedou symbolů gramatiky G. Do této množiny může patřit také speciální symbol \$, který značí konec vstupu. Postup,

který pro danou gramatiku sestaví množiny $follow(A)$ je popsán algoritmem 5.

Algoritmus 5: Algoritmus pro výpočet množiny $follow(A)$.

Vstup: Bezkontextová gramatika $G = (N, T, S, P)$, množiny $first$ pro její symboly
Výstup: Množina $follow$, pro každý symbol $X \in N$.
$follow(X) = \emptyset$
$follow(S) = \{\$\}$
Aplikuj následující pravidla, dokud je možno změnit nějakou množinu follow:
if $r : X \to xBy \in P$ **then**
    if $y \neq \varepsilon$ **then**
        $follow(B) = follow(B) \cup (first(y) - \{\varepsilon\})$
    if $\varepsilon \in first(y)$ **then**
        $follow(B) = follow(B) \cup follow(X)$

Na základě těchto množin je pak možno sestavit množiny $predict(p)$, pro všechna $p \in P$ tvaru $A \to \alpha$. Množiny $predict(p)$ obsahují terminály, které mohou být aktuálně nejlevěji vygenerovány při použití pravidla p. Tyto množiny pak lze sestavit pomocí algoritmu 6.

Algoritmus 6: Algoritmus pro výpočet množiny $predict(r)$.

Vstup: Bezkontextová gramatika $G = (N, T, S, P)$, množiny $first$ a $follow$ pro její symboly
Výstup: Množina $predict$, pro každé pravidlo $r \in P$.
foreach $r : A \to \alpha \in P$ **do**
    if $\varepsilon \in first(\alpha)$ **then**
        $predict(r) = first(\alpha) \cup follow(A)$
    else
        $predict(r) = first(\alpha)$

Na základě množin predict je pak možno vytvořit LL tabulku. LL tabulka je matice M, kde sloupce jsou indexovány pomocí množiny T a symbolu \$ a řádky pak pomocí symbolů z množiny N. Pro jednotlivé buňky pak platí, že $M(A, a) = p$, kde $p = A \to \alpha$ tehdy a jen tehdy, pokud $a \in predict(p)$ Všechny buňky, pro které není tato podmínka splněna, zůstávají prázdné. Tyto prázdné buňky pak reprezentují syntaktickou chybu ve vstupním řetězci.

Existují dvě základní metody syntaktické analýzy založené na LL tabulce – metoda rekurzivního sestupu a prediktivní syntaktická analýza. V této práci bude popsána jen

druhá z těchto metod. Tuto metodu popisuje algoritmus 7.

Algoritmus 7: Metoda prediktivní syntaktické analýzy.

Vstup: $w = x_1 x_2 \dots x_n$, $x_i \in \Sigma_{LL}$, $1 \leq i \leq n$, LL-tabulka pro popisovanou gramatiku
Výstup: Úspěch/Neúspěch, sekvence použitých pravidel.
Vlož \$ a S_{LL} na vrchol zásobníku.
while *Zásobník obsahuje nějaký symbol* **do**
    Nechť X značí symbol na vrcholu zásobníku a a aktuální vstupní token.
    switch X *je:* **do**
        case $X = \$$
            if $a = \$$ **then**
                Ukonči algoritmus, a vrať úspěch.
            else
                Ukonči algoritmus, a vrať neúspěch.
        case $X \in T_{LL}$
            if $a = X$ **then**
                Odstraň X z vrcholu zásobníku, a načti další vstupní symbol a.
            else
                Ukonči algoritmus, a vrať neúspěch.
        case $X \in N_{LL}$
            if $r : X \to \alpha \in LL_{LL}[X, a]$ **then**
                Nahraď symbol X na vrcholu zásobníku řetězcem $\text{rev}(\alpha)$. Vypiš r na výstup.
            else
                Ukonči algoritmus, a vrať neúspěch.

4.2.3 Obecný algoritmus pro analýzu zdola nahoru

Oproti předchozím metodám, pracuje tato opačným způsobem. Algoritmus popisující stavbu zásobníkového automatu provádějícího syntaktickou analýzu zdola nahoru na základě bezkontextové gramatiky $G = (N, T, S, P)$ odpovídá algoritmu 8.

Algoritmus 8: Algoritmus pro tvorbu zásobníkového automatu provádějícího syntaktickou analýzu zdola nahoru.

Vstup: Bezkontextová gramatika $G = (N, T, S, P)$
Výstup: Rozšířený zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$, který provádí syntaktickou analýzu zdola nahoru.
$Q = \{s, f\}$
$\Sigma = T$
$\Gamma = T \cup N \cup \{\$\}$
$F = \{f\}$
foreach $a \in \Sigma$ **do**
    Přidej pravidlo tvaru $sa \to as$ do R.
foreach $A \to \alpha \in P$ **do**
    Přidej pravidlo tvaru $\alpha s \to As$ do R.
Přidej pravidlo tvaru $\$Ss \to f$ do R.

Role jednotlivých symbolů je v tomto případě jiná, než u analýzy shora dolů. Pro každý

terminál popisované gramatiky je vytvořeno vkládací pravidlo (shift rule), které přečte symbol ze vstupní pásky a vloží jej na vrchol zásobníku. Druhým typem jsou redukční pravidla (reduction rule). Ta jsou, obdobně jako expanzní pravidla, tvořena na základě pravidel vstupní gramatiky. Tato pravidla provádí nahrazení řetězce symbolů, který se nachází na vrcholu zásobníku (což odpovídá pravé straně příslušného pravidla gramatiky), za symbol jediný (levá strana pravidla). Překladač pak pracuje tím způsobem, že pomocí vkládacích pravidel postupně ukládá na zásobník jednotlivé symboly. Jakmile se na vrcholu zásobníku nachází řetězec, který by bylo možno pomocí některého redukčního pravidla přepsat, je aplikována příslušná redukce a řetězec je nahrazen jedním neterminálem (z pohledu gramatiky). Analýza je úspěšná, pokud je přečten celý vstupní řetězec a zásobník obsahuje pouze počáteční neterminál. Na výstupu je pak produkována sekvence redukčních pravidel, která odpovídá obrácenému pravému rozboru.

V rámci tohoto typu syntaktické analýzy mohou nastat dva typy konfliktů, nazývaných *Reduce-Reduce* (R-R) a *Shift-Reduce* (S-R) konflikt. Reduce-Reduce konflikt nastane v případě, kdy gramatika obsahuje více pravidel se stejnou pravou stranou. Analyzátor pak musí zvolit jedno z těchto pravidel náhodně. Shift-Reduce pak nastává v případě, kdy pravá strana jednoho pravidla, je prefixem pravé strany pravidla jiného. Analyzátor pak neví, má-li provést redukci pomocí prvního z těchto pravidel, nebo pokračovat ve vkládání vstupních symbolů na zásobník a následně provést redukci pomocí pravidla delšího.

Syntaktickou analýzu, která pracuje tímto způsobem, pak popisuje algoritmus 9. Tento algoritmus uvažuje pouze variantu bez *Shift-Reduce* konfliktů.

Algoritmus 9: Algoritmus popisující činnost automatu provádějícího syntaktickou analýzu zdola nahoru.

Vstup: $w = x_1 x_2 \dots x_n$, $x_i \in \Sigma_{BU}$, $1 \leq i \leq n$
Výstup: Úspěch/Neúspěch, sekvence použitých pravidel.
Vlož \$ na vrchol zásobníku.
while *Zásobník obsahuje nějaký symbol* **do**
  Nechť α značí řetězec na vrcholu zásobníku (poslední symbol řetězce je na vrcholu zásobníku, první je pak směrem k počátku zásobníku) a a aktuální vstupní token.
  switch α *je:* **do**
    case $\alpha = \$S_{BU}$
      if $a = \$$ **then**
        Ukonči algoritmus, a vrať úspěch.
      else
        Ukonči algoritmus, a vrať neúspěch.
    case $\exists r : X \to \alpha \in P_{BU}$
      Nahraď α na vrcholu zásobníku za X. Pokud existuje více pravidel (R-R konflikt), vyber jedno z těchto pravidel nedeterministicky. Zapiš r na výstup.
    case $\nexists r : X \to \alpha \in P_{BU}$
      if $a = \$$ **then**
        Ukonči algoritmus, a vrať neúspěch.
      else
        Vlož a na vrchol zásobníku a načti nový vstupní symbol a.

Obdobně jako v případě obecné analýzy shora dolů, platí i zde, že je možné sestavit příslušný analyzátor pro libovolnou bezkontextovou gramatiku. Stejně tak jsou stejné i

nevýhody této metody, tedy to že pracuje nedeterministickým způsobem. To je zapříčiněno R-R a S-R konflikty popsanými výše. Pro praktické využití se proto příliš nehodí.

4.2.4 LR-analýza

Obdobně jako v případě analýzy shora dolů a LL analýzy, LR (L zde opět značí čtení vstupu zleva doprava, R pak značí tvorbu pravého rozboru) analýza je variantou analýzy zdola nahoru, která zavádí systém, který zajišťuje deterministický výběr jednotlivých pravidel použitých při syntaktické analýze. Pro výběr použitého pravidla slouží LR tabulka. Tato tabulka sestává ze dvou částí – akční (action) části a GOTO části. Základní variantou LR syntaktického analyzátoru je SLR (Simple LR) analyzátor. Tato varianta je tzv. LR(0) variantou, kde 0 zde značí počet dalších symbolů vstupního řetězce nutných k výběru pravidla. Dalšími variantami jsou pak LALR, či kanonické LR. Tyto varianty pracují jako LR(1) analyzátory a jsou tedy schopny popsat složitější jazyky. Na druhou stranu, jsou také složitější a obsahují větší množství pomocných stavů. V rámci této práce budu používat pouze SLR variantu a jako LR pak tedy budu chápat právě SLR.

Na rozdíl od výše popsaných metod analýzy, LR analýza využívá kromě zásobníku i vnitřní stavy automatu. Tyto stavy slouží analyzátoru k zapamatování si, kde v rámci pravé strany možných redukčních pravidel se nachází. Algoritmus, který slouží k sestavení LR tabulky je možno nalézt v [2]. Jak již bylo řečeno, LR tabulka sestává ze dvou částí, akční části (značeno jako α), kde jednotlivé sloupce odpovídají terminálům gramatiky a GOTO části (značeno jako β), kde jsou sloupce určeny pomocí neterminálů. Řádky obou těchto částí jsou pak určeny pomocí stavů příslušného zásobníkového automatu. Buňky akční části pak obsahují čtyři možné druhy záznamů:

1. Záznam tvaru sq, který říká, že má automat vložit aktuální token na zásobník a přejít do stavu q.
2. Záznam tvaru rp, který říká, že má automat provést redukci pomocí pravidla p.
3. Prázdnou buňku, která značí syntaktickou chybu.
4. Buňku, značící úspěch (značeno jako :-)).

Činnost LR analyzátoru pro gramatiku $G_{LR} = (N_{LR}, T_{LR}, S_{LR}, P_{LR})$ popisuje algoritmus 10. Symboly, vkládané na zásobník, jsou v tomto druhu analýzy dvojice, tvořené symbolem samotným a stavem automatu.

Algoritmus 10: Algoritmus popisující činnost automatu provádějícího LR syntaktickou analýzu zdola nahoru.

Vstup: $w = x_1 x_2 \dots x_n$, $x_i \in T_{LR}$, $1 \leq i \leq n$, LR tabulka pro gramatiku G_{LR}
Výstup: Úspěch/Neúspěch, sekvence použitých pravidel.
Vlož na vrchol zásobníku dvojici $(\$, q_0)$, kde \$ značí dno zásobníku a q_0 je počáteční symbol automatu.
$stav = q_0$
repeat
  Nechť a je aktuální token.
  switch $\alpha[stav, a]$ *je:* **do**
    case *sq*
      Vlož (a, q) na vrchol zásobníku.
      $stav = q$
      Přečti další token a.
    case *rp*
      Kde $p : A \to x_1 x_2 \dots x_n$.
      if *Vrchol zásobníku je tvaru* $(?, q)(x_1, ?)(x_2, ?) \dots (x_n, ?)$ **then**
        $stav = \beta[q, A]$
        Nahraď symboly $(x_1, ?)(x_2, ?) \dots (x_n, ?)$ na vrcholu zásobníku za symbol $(A, stav)$ a vypiš p na výstup.
      else
        Ukonči analýzu jako neúspěšnou.
    case *:-)*
      Ukonči analýzu jako úspěšnou.
    case *Prázdná buňka*
      Ukonči analýzu jako neúspěšnou.
until *Úspěch nebo neúspěch.*

4.3 Syntaktická analýza založená na navržených gramatických systémech

V této části bude představena metoda syntaktické analýzy, založená na gramatických systémech zavedených v kapitole 3. Navržená metoda kombinuje principy LL a LR syntaktické analýzy. V rámci této metody se budou uvažovat pouze gramatické systémy, ve kterých jsou jednotlivé řízené komponenty aktivovány pouze řídicí komponentou a řízené komponenty tedy neobsahují pravidla, která by aktivovala jinou řízenou komponentu.

Navržený syntaktický analyzátor se skládá z několika dílčích analyzátorů, které využívají mírně modifikované existující metody syntaktické analýzy. Ke každé komponentě systému je vytvořen vlastní syntaktický analyzátor a tyto analyzátory pak spolupracují podobným způsobem jako navržený gramatický systém. Standardní metody analýzy je třeba rozšířit o mechanismus, který se postará o šíření informace o počtu provedených kroků jednotlivých komponent. V rámci modelu zásobníkových automatů, odpovídá počtu přepisovacích pravidel počet expanzních (při analýze shora dolů) nebo redukčních (analýza zdola nahoru) pravidel.

Celý syntaktický analyzátor je sestaven pomocí dvou níže popsaných metod, kdy první z nich je použita v rámci řídicí komponenty a druhá pak v rámci všech řízených komponent. Každá komponenta pak pracuje s vlastním zásobníkovým automatem. Jednotlivé zásobníky

ani stavová řízení tak nejsou žádným způsobem sdíleny.

4.3.1 Řídicí komponenta

Analyzátor, který reprezentuje řídicí komponentu, vychází z LL-analýzy. Oproti standardní LL analýze, jsou v tomto případě rozšířeny jednotlivé terminály (tedy počáteční neterminály řízených komponent) na dvojice. Prvním prvkem této dvojice je pak symbol samotný. Druhým je pak derivační mód dané komponenty. Jednotlivé terminály tedy mají tvar $X = (x, d)$, kde x je původní symbol a d je jemu přiřazený derivační mód. Toto rozšíření se pak využívá při ověřování počtu použitých pravidel v rámci ostatních komponent. Pro práci s LL tabulkou apod. pak slouží pouze původní symbol.

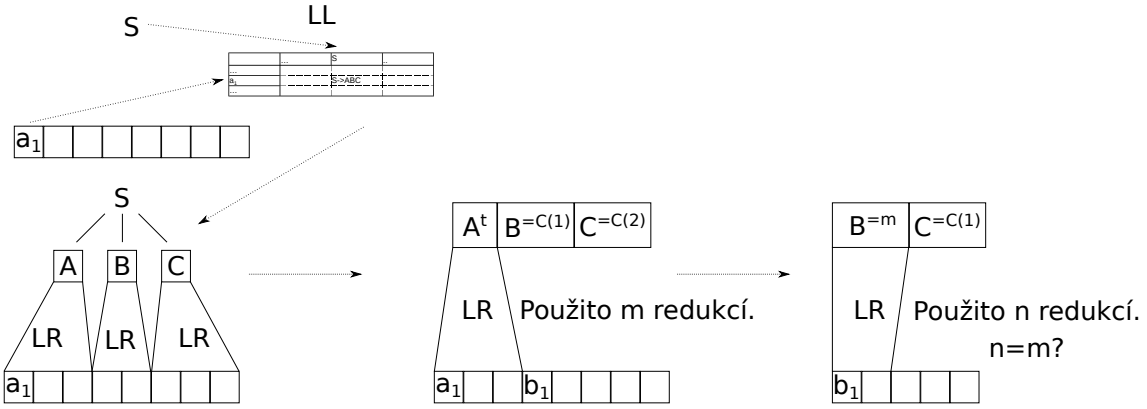
Stejně jako v případě standardní LL analýzy, jsou i zde dva druhy pravidel – první rozgenerují neterminál na vrcholu zásobníku na pravou stranu pravidla, ve kterém se nalézá tento neterminál na jeho levé straně. Druhý typ pravidel porovnává terminál na vrcholu zásobníku se symbolem na vstupní pásce a shodují-li se, odstraní tento symbol ze zásobníku. Tento druhý typ pravidel je v navrženém systému modifikován. Jakmile by měl být použit tento typ pravidla, je aktivována příslušná řízená komponenta. Jakmile tato komponenta dokončí svůj výpočet, vrátí řídicí komponentě počet pravidel, která použila. Řídicí komponenta porovná tento počet s požadovaným módem derivace a případně rozšíří tuto informaci ke všem počátečním neterminálům, které se na tuto komponentu odkazují. Pokud komponenta splnila požadavky derivačního módu, je příslušný symbol odstraněn ze zásobníku. V opačném případě je výpočet ukončen jako neúspěšný. Při tvorbě množin $first$, $follow$ a $predict$, pro jednotlivé počáteční neterminály řízených komponent, se uvažují i pravidla obsažená v těchto řízených komponentách.

Udržování odkazů na jednotlivé komponenty je zajištěno pomocí dvou mechanismů. První se stará o případy, kdy se ve větné formě nalézají počáteční symboly, které se na sebe nějakým způsobem odkazují a těmto symbolům předchází nějaký neterminál řídící gramatiky. Jakmile je tento neterminál rozgenerován, je k odkazujícím indexům všech počátečních neterminálů, které se nalézají *za* tímto místem rozgenerování, přičten počet počátečních neterminálů, které byly tímto rozgenerováním přidány do větné formy. Všechny nově přidané počáteční neterminály pak jsou inicializovány pomocí hodnot daných použitým pravidlem a zobrazením d (definice 38). Druhý mechanismus se pak stará o aktualizaci indexů při odstranění symbolu ze zásobníku. Při každém odstranění počátečního neterminálu jsou sníženy hodnoty všech odkazujících se symbolů o 1. Jakmile hodnota některého indexu dosáhne nulové hodnoty, znamená to, že se tato komponenta odkazuje na právě ukončenou komponentu a proto je jí předána informace o počtu požadovaných kroků. Algoritmus 11 pak popisuje činnost analyzátoru přiřazenému řídicí komponentě.

Obrázek 4.2 demonstruje činnost řídicí komponenty pracující modifikovanou LL metodou.

Algoritmus 11: Algoritmus popisující činnost řídicí komponenty.

Vstup: $w = x_1 x_2 \dots x_n$, $x_i \in \Sigma_{LL}$, $1 \leq i \leq n$, LL-tabulka pro řídicí gramatiku
Výstup: Úspěch/Neúspěch, sekvence použitých pravidel.
Vlož \$ a S_{LL} na vrchol zásobníku
while *Zásobník obsahuje nějaký symbol* **do**
  Nechť X značí symbol na vrcholu zásobníku a a aktuální vstupní token.
  switch X *je:* **do**
    case $X = \$$
      if $a = \$$ **then**
        Ukonči algoritmus, a vrať úspěch.
      else
        Ukonči algoritmus, a vrať neúspěch.
    case $X \in T_{LL}$
      Spusť řízenou komponentu odpovídající symbolu X a urči symboly, které pro ni budou značit konec vstupu (podsekce 4.3.2).
      if *Řízená komponenta dokončila analýzu úspěšně.* **then**
        Porovnej požadovaný počet kroků se skutečným počtem kroků.
        if *Komponenta provedla správný počet kroků.* **then**
          Patřičně aktualizuje počáteční symboly na zásobníku. Odstraň X ze zásobníku.
        else
          Ukonči algoritmus, a vrať neúspěch.
      else
        Ukonči algoritmus, a vrať neúspěch.
    case $X \in N_{LL}$
      if $r : X \to \alpha \in LL_{LL}[X, a]$ **then**
        Nahraď symbol X na vrcholu zásobníku řetězcem rev(α). Ke všem odkazovacím indexům na zásobníku, které se nachází za řetězcem α přičti počet počátečních symbolů v řetězci (α). Vypiš r na výstup.
      else
        Ukonči algoritmus, a vrať neúspěch.

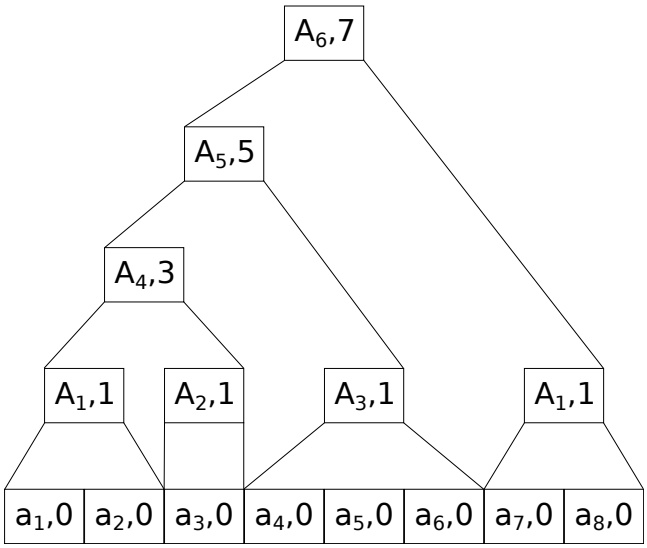


Obrázek 4.2: Ukázka modifikované LL metody

4.3.2 Řízené komponenty

Jednotlivé řízené komponenty pracují metodou zdola nahoru, na základě LR syntaktického analyzátoru. Obdobně jako v případě řídicí komponenty, jsou i zde jednotlivé symboly rozšířeny na dvojice. V tomto případě jsou ovšem rozšířeny všechny symboly. Na rozdíl od LL analyzátoru řídicí komponenty, neobsahuje druhá část tohoto rozšíření informaci o módu derivace. Místo toho, je zde ukládán počet kroků, které daná komponenta uskutečnila. Jednotlivé symboly pak mají tedy tvar $X = (x, r)$, kde x je původní symbol a r je počet redukcí, které k jeho vytvoření vedly. Obdobně jako v případě standardní LR analýzy jsou na zásobník postupně ukládány symboly ze vstupní pásky. K těmto vstupním terminálům je při ukládání připojena i informace o počtu použitých redukcí, tedy hodnota 0. Jakmile dojde k redukci části zásobníku, je k neterminálu, na který je tento řetězec redukován, přidána suma všech hodnot r symbolů, které byly redukovány a dále je k této sumě přičtena hodnota 1. Tímto způsobem je postupně šířena informace o počtu použitých pravidel směrem k počátečnímu neterminálu aktivní komponenty. Toto šíření je znázorněno na obrázku 4.3. Algoritmus 12 pak popisuje činnost analyzátoru přiřazenému řízené komponentě.

Jelikož je při tomto druhu analýzy potřeba znát konec vstupního řetězce, je nutné pro každou řízenou komponentu určit, které symboly gramatického systému budou považovány jako konec vstupního podřetězce přiřazeného dané komponentě. Jako tyto symboly je možno použít symboly z množiny $follow(S_i)$, kde S_i je počáteční neterminál této komponenty. Tyto symboly pak jsou ponechány ve vstupním řetězci, pro zpracování další komponentou. Alternativně je také možno počítat tyto symboly dynamicky, jako $first(\alpha)$, kde α je řetězec na vrcholu LL zásobníku, který se nalézá těsně za počátečním neterminálem této řízené komponenty.



Obrázek 4.3: Ukázka šíření počtu redukcí v řízené komponentě.

Algoritmus 12: Algoritmus popisující činnost řízené komponenty.

Vstup: $w = x_1 x_2 \dots x_n$, $x_i \in T_{LR}$, $1 \leq i \leq n$, LR tabulka pro gramatiku G_{LR}
Výstup: Úspěch/Neúspěch, sekvence použitých pravidel.
Vlož na vrchol zásobníku dvojici $((0, \$), q_0)$, kde \$ značí dno zásobníku a q_0 je počáteční symbol automatu a 0 je počet redukcí vedoucích k tomuto symbolu.
$stav = q_0$
repeat
  Nechť a je aktuální token.
  switch $\alpha[stav, a]$ *je:* **do**
    case sq
      Vlož $((0, a), q)$ na vrchol zásobníku.
      $stav = q$
      Přečti další token a.
    case rp
      Kde $p : A \to x_1 x_2 \dots x_n$.
      if *Vrchol zásobníku je tvaru* $(?, q)((c_1, x_1), ?)((c_2, x_2), ?) \dots ((c_n, x_n), ?)$
      then
        $stav = \beta[q, A]$
        Nahraď symboly $((c_1, x_1), ?)((c_2, x_2), ?) \dots ((c_n, x_n), ?)$ na vrcholu zásobníku za symbol $((1 + \sum_{i=1}^{n} c_i, A), stav)$ a vypiš p na výstup.
      else
        Ukonči analýzu jako neúspěšnou.
    case *:-)*
      Ukonči analýzu jako úspěšnou a předej řídicí komponentě počet redukcí vedoucích na počáteční symbol.
    case *Prázdná buňka*
      Ukonči analýzu jako neúspěšnou.
until *Úspěch nebo neúspěch.*

Kapitola 5

Ukázky a modifikace syntaktické analýzy pomocí navržené metody

V rámci této kapitoly bude demonstrována činnost navržené metody syntaktické analýzy na několika jazycích, které nejsou bezkontextové. Budou zde také zavedeny určité modifikace navržené metody.

5.1 Příklad 1

Předpokládejme jazyk $L_1 = \{a^i b^j c^i d^j | i, j \geq 1 \wedge i \neq j\}$. Každý řetězec tohoto jazyka je možno rozdělit na čtyři podřetězce, které se vždy skládají z jednoho druhů terminálů. Ke každému tomuto podřetězci pak bude přiřazena jedna řízená komponenta, která bude daný podřetězec analyzovat. Řídicí komponenta bude obsahovat jediné pravidlo tvaru $P_M = \{1 : S \rightarrow A^t B^{\neq C(1)} C^{=C(1)} D^{=C(2)}\}$, kde derivační mód $\neq C(1)$ značí variantu C-módu, kde odkazující a odkazovaná komponenta *nesmí* použít stejný počet pravidel. Pro řízené komponenty pak platí, že $P_A = \{1 : A \rightarrow Aa, 2 : A \rightarrow a\}$, analogicky pak pro komponenty B, C a D. Tabulka 5.1 popisuje LL tabulku pro řídicí komponentu. Tabulka 5.2 pak LR tabulku pro komponentu A[1]. Tabulky pro B, C a D jsou pak obdobné.

Tabulka 5.1: LL tabulka řídicí komponenty pro jazyk L_1

	a	b	c	d	$
S	1				

Činnost syntaktického analyzátoru pro první dvě komponenty a pro řetězec *aabbbccddd* je pak znázorněna v tabulce 5.3. Jednotlivé zásobníkové symboly LR analýzy pak jsou tvaru $((redukce, symbol), stav)$.

[1] Symbol S_A slouží jako pomocný počáteční symbol rozšířené gramatiky A (viz konstrukce LR tabulky v [2])

Tabulka 5.2: LR tabulka pro řízenou komponentu A pro jazyk L_1.

	Action		GOTO	
	a	$ (b)	S_A	A
0	s2			1
1	s3	:-)		
2	r2	r2		
3	r1	r1		

Tabulka 5.3: Ukázka činnosti analyzátoru pro jazyk L_1. Zobrazena je pouze analýza první půlky vstupního řetězce.

LL				
Vstup	Zásobník	Derivační módy	Akce	Pravidlo
aabbbccddd	S		expanduj	1
aabbbccddd	DCBA	[=C(2)],[=C(1)],[≠C(1)],[t]	Spusť LR pro A, $=b	
LR				
Vstup	Zásobník	Stav	Akce	Pravidlo
aa$	((0,$),0)	0	s2	
a$	((0,$),0),((0,a),2)	2	r2	2
a$	((0,$),0),((1,A),1)	1	s3	
$	((0,$),0),((1,A),1),((0,a),3)	3	r1	1
$	((0,$),0),((2,A),1)	1	:-)	
LL				
Vstup	Zásobník	Derivační módy	Akce	Pravidlo
bbbccddd	DCBA	[=C(2)],[=C(1)],[≠C(1)],[t]	Konec LR pro A, 2 redukce	
bbbccddd	DCBA	[=C(2)],[=C(1)],[≠C(1)],[t]	Kontrola der. módu	
bbbccddd	DCB	[=C(1)],[=2],[≠2]	Aktualizace der. Módů, odstranění ze zásobníku	
bbbccddd	DCB	[=C(1)],[=2],[≠2]	Spusť LR pro B, $=c	
LR				
Vstup	Zásobník	Stav	Akce	Pravidlo
bbb$	((0,$),0)	0	s2	
bb$	((0,$),0),((0,b),2)	2	r2	2
bb$	((0,$),0),((1,B),1)	1	s3	
b$	((0,$),0),((1,B),1),((0,b),3)	3	r1	1
b$	((0,$),0),((2,B),1)	1	s3	
$	((0,$),0),((2,B),1),((0,b),3)	3	r1	1
$	((0,$),0),((3,B),1)	1	:-)	
LL				
Vstup	Zásobník	Derivační módy	Akce	Pravidlo
ccddd	DCB	[=C(2)],[=C(1)],[≠2]	Konec LR pro B, 3 redukce	
ccddd	DCB	[=C(2)],[=C(1)],[≠2]	Kontrola der. Módu 2≠3	
ccddd	DC	[=3],[=2]	Aktualizace der. Módů, odstranění ze zásobníku	
ccddd	DC	[=3],[=2]	Spusť LR pro C, $=d	

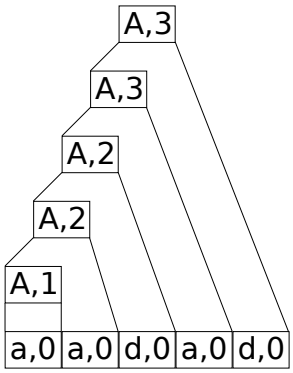
5.2 Příklad 2

V tomto příkladu bude modifikována činnost řízených komponent. Jednotlivá pravidla budou rozdělena do dvou množin. Toto rozdělení určuje, jakým způsobem bude počítán počet užitých přepisovacích pravidel. Pravidla v první skupině budou započítávána standardním způsobem, tak jak bylo popsáno dříve. Pravidla z druhé množiny pak nebudou započítávána. Tento princip bude demonstrován na jazyku $L_2 = \{uvw|u \in \{a,d\}^*, v \in \{b,e\}^*, w \in \{c,f\}^*, |u|_a = |v|_b = |w|_c\}$. Pravidla komponenty S_A pak budou tvaru $\{1 : A \to Aa, 2 : A \to Ad, 3 : A \to a, 4 : A \to d\}$. Započítávána pak budou pouze pravidla 1 a 3. Ostatní komponenty budou pak obdobné.

Tabulka 5.4: LR tabulka pro řízenou komponentu A pro jazyk L_2

	Action			GOTO	
	a	d	$	S_A	A
0	s2	s3			1
1	s4	s5	:-)		
2	r3	r3	r3		
3	r4	r4	r4		
4	r1	r1	r1		
5	r2	r2	r2		

Obrázek 5.1 pak znázorňuje činnost první řízené komponenty. Tato komponenta pak počítá pouze redukce se symbolem a. Redukce se symbolem d jsou pak ignorovány. Ostatní řízené komponenty pak pracují obdobným způsobem. Řídicí komponenta pak funguje obdobně jako v předešlém příkladu.



Obrázek 5.1: Ukázka činnosti řízené komponenty s různě ohodnocenými pravidly.

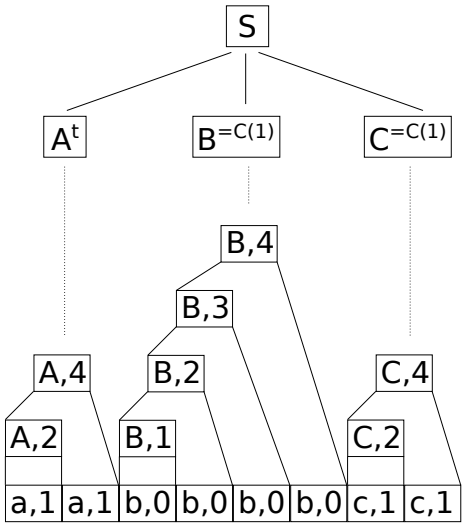
5.3 Příklad 3

Další možnou modifikací je rozšířit C módy o možnost jednoduchých aritmetických výrazů. Příkladem jazyka, který využívá této modifikace je jazyk $L_3 = \{a^i b^{2i} c^i | i \geq 1\}$. Pro tuto modifikaci je možno využít dva přístupy. První z nich spočívá v zavedení pomocných pravidel v řízených komponentách. Tato pravidla je možno použít pro přičtení konstanty, případně

vynásobení požadovaného resp. uskutečněného počtu kroků konstantou. V případě přičtení stačí zavést pravidla, která sestávají z jednoto neterminálu na levé straně a jednoho neterminálu na straně pravé. Dále se zavede nový počáteční neterminál pro danou komponentu, který se pomocí sekvence těchto pravidel *napojí* na počáteční neterminál původní. Takto pracující komponenta pak bude muset provést i tuto sekvenci, čímž dojde k použití několika redukcí navíc, oproti komponentě, která tuto sekvenci pravidel nemá zavedenu. V případě násobení, lze zavést podobné sekvence k neterminálům na všech levých stranách pravidel dané komponenty. Alternativně lze násobení také simulovat pomocí podobného způsobu jako v předešlém příkladu. Jednotlivá pravidla však nebudou rozdělena do dvou množin, místo toho budou *všechna* pravidla dané gramatiky mít váhu rovnu nějaké konstantě. Při šíření počtu redukcí (obrázek 4.3), pak nebude při každé redukci přičítána hodnota 1, ale právě tato konstanta.[2] Podobným způsobem je možno dále upravovat systém počítání redukcí. Například, pokud by se hodnota nepřičítala, ale hodnotou by se násobilo (čítače u terminálů by pak byly inicializovány na hodnotu 1), bylo by možno simulovat požadavky typu $L_3^{'} = \{a^i b^{2^i} c^i | i \geq 1\}$.

Alternativně je také možno implementovat samostatný interpret aritmetických výrazů jako součást překladače. Tento interpret by pak mohl být schopen zpracovávat i složitější výrazy.

Obrázek 5.2 pak znázorňuje činnost systému pro syntaktickou analýzu jazyku $L_3^{'} = \{a^i b^{2^i} c^i | i \geq 1\}$. V tomto systému komponenty A a C počítají počet redukcí pomocí násobení. Komponenta B pak počítá standardním způsobem pomocí sčítání.



Obrázek 5.2: Systém s kombinací „sčítajících“ a „násobících“ komponent.

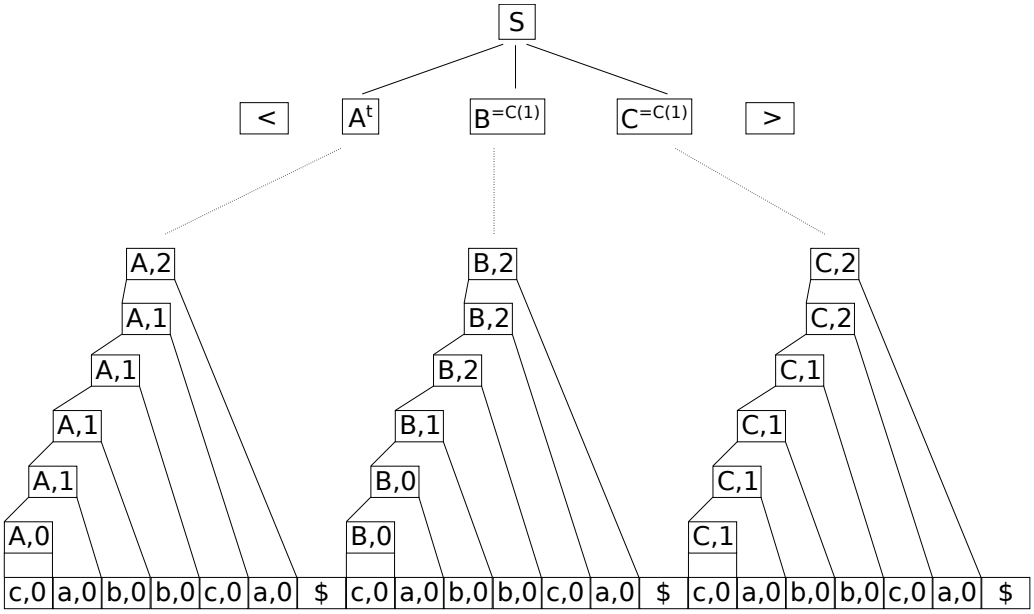
5.4 Příklad 4

Výše uvedené metody mají tu nevýhodu, že předpokládají, že jednotlivé symboly jsou v rámci řetězce odděleny a je tak možno sestavit ke každému takovému podřetězci příslušnou gramatiku, které jsou pak konkatenovány. Systém je pak schopen snadno analyzovat

[2] Samozřejmě by šlo také použít pravidla, která přepisují více terminálů na vrcholu zásobníku (např. $A \rightarrow aa$ místo pouhého $A \rightarrow a$).

jazyky tvaru $L = \{a^i b^i c^i | i \geq 1\}$, ale s podobným jazykem $L' = \{w | w \in \{a,b,c\}^*, |w|_a = |w|_b = |w|_c\}$ by si již neporadil. Toto by mohlo jít vyřešit úpravou aktivace komponent. Předpokládejme tři řízené komponenty S_a, S_b, S_c a jednu komponentu řídicí S_m. Každá z řízených komponent by měla pravidla rozdělena do dvou množin, tak jako v příkladu 5.2, přičemž každá komponenta by započítávala redukce vedoucí na jeden ze tři terminálů a redukce ostatní by nezapočítávala. V ničem jiném by se tyto komponenty od sebe nelišily (jiné pojmenování počátečního symbolu zanedbejme). Řízená komponenta by pak obsahovala jediné pravidlo, které by vygenerovalo řetězec $S_a S_b S_c$, přičemž jednotlivé počáteční symboly by byly propojeny C módy. Oproti výše navrženému gramatickému systému, by ovšem tyto komponenty nepracovaly nad společnou větnou formou. Místo toho, by příslušné syntaktické analyzátory obdržely kopii vstupního řetězce, kterou by analyzovaly. Jakmile by takto každá komponenta provedla svou analýzu (jednotlivé analyzátory by byly na sobě nezávislé a mohli by tak běžet paralelně), byly by porovnány počty redukcí jednotlivých komponent a vstup by byl přijat případně zamítnut, sekvence použitých pravidel by pak byla stejná ve všech komponentách. Počáteční symboly takto paralelně aktivovaných komponent se pak musí, v rámci pravidla řídicí komponenty, nacházet vedle sebe a nesmí tak být rozděleny nějakými dalšími symboly. V případě kombinace se standardní sekvenční analýzou popsanou dříve, bude vhodné zavést speciální symboly pro řídicí komponentu, které oddělí jednotlivé paralelně pracující počáteční symboly. Příkladem takového pravidla by pak mohlo být například $S \rightarrow S_1 < S_2 S_3 > S_4$. Nejprve by pak byla spuštěna komponenta 1. Následně by byly paralelně spuštěny komponenty 2 a 3. Po jejich ukončení by pak byla spuštěna komponenta 4. Takto pracující komponenty pak připomínaji PCGS případně multigenerativní GS.

Níže popsaný příklad popisuje analýzu jazyku $L' = \{w | w \in \{a,b,c\}^*, |w|_a = |w|_b = |w|_c\}$. Řídicí komponenta pak obsahuje pravidlo $S \rightarrow < AB^{=C(1)}C^{=C(1)} >$. Řízené gramatiky pak sestávají z pravidel $\{1 : A \rightarrow Aa, 2 : A \rightarrow Ab, 3 : A \rightarrow Ac, 4 : A \rightarrow a, 5 : A \rightarrow b, 6 : A \rightarrow c\}$ (obdobně B a C). Jednotlivé Komponenty pak počítají pouze dvojice pravidel $\{1,4\}$, $\{2,5\}$ nebo $\{3,6\}$, každá komponenta jednu z těchto dvojic.



Obrázek 5.3: Systém s paralelně pracujícími komponentami.

5.5 Další možná rozšíření

Dalším možným rozšířením by bylo, povolit jistou formu nedeterminismu v rámci derivačních módů při syntaktické analýze. Konkrétně pak povolit více pravidel, která se liší pouze použitými derivačními módy. Řídicí komponenta by tak měla ke každé levé straně množinu řetězců, které přiřazují derivační módy stejné posloupnosti počátečních symbolů. Na základě postupného odstraňování symbolů z vrcholu zásobníku by pak na základě prefixů těchto řetězců jednotlivé řetězce z této množiny odstraňovala.

Dále by bylo možno povolit možnost odkazování se na více komponent v rámci jedné aktivace. Takto pracující komponenta by pak musela splnit všechny takto vzniklé požadavky na počet kroků. Tyto požadavky by si pak samozřejmě nesměly protiřečit (například odkazovat na dvě komponenty pracující v $= k$ módu, kde k není shodné pro obě z těchto komponent.

Další možná rozšíření by bylo možno založit na modifikacích popsaných v příkladu 5.3. Bylo by možno zkoumat, jaké jazyky by byly akceptovány při použití jiných operací pro počítání počtu redukcí než sčítání, či násobení, například logických operací.

Kapitola 6

Demonstrační aplikace

Jako součást této práce byla implementována demonstrační aplikace v jazyce Java. Tato aplikace slouží k ukázce činnosti navržené metody syntaktické analýzy. Aplikace podporuje některá z výše popsaných rozšíření. Jednotlivé gramatické systémy jsou popsány pomocí vstupního souboru ve formátu xml. Tento soubor obsahuje informace o přepisovacích pravidlech jednotlivých gramatik. Dále obsahuje LL resp. LR tabulky pro tyto komponenty. Na základě tohoto souboru a vstupního řetězce pak probíhá vizualizace syntaktické analýzy. Hlavní okno aplikace sestává ze dvou částí. První část zobrazuje činnost aktuální komponenty v podobě LR případně LL analyzátoru. Pro komponentu je pak znázorněn zásobník příslušného zásobníkového automatu a sekvence použitých pravidel. Druhá část hlavního okna pak zobrazuje derivační strom pro současnou větnou formu. Jelikož probíhá výpočet kombinací metod shora dolů a zdola nahoru, jsou jednotlivé podstromy reprezentující řízené komponenty přidávány jakmile tato komponenta dokončí svůj výpočet. Větve pro komponentu řídicí jsou pak přidávány průběžně. Aplikace pak podporuje pouze sekvenční variantu navrhované metody, paralelní varianta popsaná v rámci příkladu 5.4 podporována není.

6.1 Popis vstupního souboru

Soubor s popisem vstupního gramatického systému je ve formátu xml. Všechny uvedené prvky pak jsou *case sensitive*. Celý systém je obsažen v kořenovém elementu `<system>`. V rámci tohoto elementu se pak nalézají jednotlivé komponenty. Každá z těchto komponent je popsána samostatným elementem `<grammar>`. Tento element má jeden povinný atribut `id="X"`, kde `X` je celé číslo. Atribut `id` slouží k vnitřní identifikaci jednotlivých komponent. Komponenta s `id="0"` je pak brána jako komponenta řídicí. Dále může tento element obsahovat volitelný atribut `multiplication="true"`, který zajistí, že daná komponenta bude počítat počet použitých pravidel pomocí násobení (příklad 5.3), pak bude pravděpodobně potřeba upravit váhy jednotlivých pravidel (popis níže).

Každá komponenta pak sestává ze tří částí:

- Definice symbolů.
- Definice pravidel.
- Definice tabulek.

Definice symbolů sestává z posloupnosti elementů tvaru `<symbol starting=["true"|"false"] nonterminal=["true"|"false"] >S</symbol>`. Atribut `starting` určuje, jedná-li se o počáteční symbol. Atribut `nonterminal` pak určuje, jedná-li se o neterminál. Pro

řídicí komponentu pak platí, že jednotlivé počáteční symboly řízených komponent jsou považovány za *počáteční terminály*. Záznam tohoto typu pak musí existovat pro každý symbol gramatiky včetně případného symbolu prázdného, který je značen jako `<symbol starting ="false" nonterminal="false" ></symbol>`. Pro terminály pak platí, že vstupní řetězec je načítán po jednotlivých znacích, terminály řízených komponent tedy musí být jednoznakové. Symbol `$` je pak rezervován pro označení konce vstupu a nesmí tak být součástí abecedy. Neterminály pak mohou být i víceznakové.

Definice pravidel pak sestává z posloupnosti elementů `<rule id="X" value="Y"></ rule>`, kde `X,Y` jsou celá čísla. Atribut `id` slouží k identifikaci pravidla a při odkazování na pravidla v rámci LL/LR tabulek a je povinný. Atribut value pak určuje váhu daného pravidla při výpočtu počtu použitých pravidel. Tento atribut je volitelný, a pokud není přítomen, používá se váha 1. Každé pravidlo je pak definováno jedním tímto elementem. V rámci tohoto elementu se pak nachází element `<lhs >S</lhs>`, který určuje symbol na levé straně pravidla (v tomto případě S) a posloupnosti elementů `<rhs mode = ["T"|" C"|"not!C"|"maxC"|"minC"|"K"|"notK"|"maxK"|"minK"] modeval="X">A</rhs>` , kde atribut `mode` určuje použitý derivační mód pro daný počáteční symbol a `modeval` pak parametr toho módu (konstanta pro `K` a index pro `C`). Tyto atributy jsou volitelné, a pokud nejsou použity, bere se hodnota `"T"` resp. `1`.

Poslední částí každé komponenty je pak definice LL/LR tabulek. Tyto tabulky jsou uzavřeny v elementu `<tables>`. LL tabulka je pak obsažena v rámci elementu `<LL>`. Tento element pak obsahuje sekvenci elementů `<nonterm sym="S">`, kde atribut `sym` značí jednotlivé neterminály (řádky) tabulky. Tyto elementy pak obsahují elementy `<term sym="a ">id</term>`, kde atribut `sym` značí jednotlivé terminály (sloupce) tabulky. Hodnota `id` je pak identifikátor pravidla v dané buňce. Prázdné buňky není třeba vyplňovat.

LR tabulka sestává ze dvou částí, akční a GOTO části. Akční část je ve vstupním souboru označena elementem `<action>` a obsahuje sekvenci záznamů tvaru `<state st="X ">`, kde atribut `st` určuje stav příslušného automatu (řádek tabulky). Stav `st="0"` je pak brán jako počáteční. Jednotlivé elementy se stavy pak obsahuje posloupnost elementů tvaru `<symbol sym="a" val="X">[shift|reduction|success]</symbol>`, přičemž atribut `sym` určuje terminál (sloupec tabulky) a atribut `val` pak parametr prováděné akce, tedy pravidlo pro redukci a nový stav pro vkládání. GOTO část tabulky pak má obdobnou strukturu. Tato část je obsažena v elementu `goto`. Ten obsahuje opět elementy `<state st="X">`, tyto elementy pak obsahují opět záznamy o jednotlivých symbolech, tentokrát ovšem tvaru `< symbol sym="A">X</symbol>`, kde `X` odpovídá stavu automatu.

Ukázku vstupního souboru je pak možno nalézt v rámci příloh k této práci, kde se také nalézá návod na překlad a spuštění této aplikace.

Kapitola 7

Závěr a další směřování práce

V rámci této práce byla zavedena nová varianta gramatických systémů. Tato varianta, vychází z CD gramatických systémů. Navrhovaný systém se skládá z několika dílčích gramatik nazývaných komponenty. Jedna z těchto komponent je určena jako řídicí komponenta, která se stará o aktivaci komponent ostatních, nazývaných komponenty řízené. Je zaveden také mechanismus, který umožňuje zavést určité požadavky na počet pravidel použitých jednotlivými řízenými komponentami během výpočtu. Jednotlivé řízené komponenty se pak mohou na sebe odkazovat a je pak možno zavést určitá omezení počtu použitých pravidel právě na základě činnosti těchto odkazovaných komponent. Tento systém je pak možno použít i pro generování jazyků, které nejsou bezkontextové.

Na základě tohoto navrženého systému byla navržena i metoda syntaktické analýzy. Tato navrhovaná metoda opět pracuje na základě kombinace více gramatik. Ke každé gramatice je pak vytvořen vlastní syntaktický analyzátor. Řídicí komponenta pak pracuje metodou shora dolů pomocí LL syntaktické analýzy. Komponenty řízené pak metodou zdola nahoru pomocí LR syntaktické analýzy. Podobně jako v případě navrhovaného gramatického systému je možno i v tomto syntaktickém analyzátoru zavést jisté provázání počtu použitých pravidel jednotlivými komponentami. Systém pak může zpracovávat i jazyky, které nejsou bezkontextové, speciálně pak jazyky, které kladou určité požadavky na počty jednotlivých symbolů vzhledem k jiným symbolům v daném řetězci.

7.1 Možná pokračování této práce

V této práci jsou popsány některé základní vlastnosti navrženého systému. Především je popsána vyjadřovací síla těchto systémů vzhledem k bezkontextovým jazykům. Chybí ovšem důkladnější porovnání s jinými modely, které jsou také schopny popsat nadmnožinu bezkontextových jazyků. Jelikož jsou jednotlivé komponenty aktivovány komponentou řídicí, pracují, oproti standardním CD gramatickým systémům, jednotlivé komponenty navrhovaného systému poměrně „odděleně“ v rámci větné formy. Dalším možným pokračováním této práce, by mohlo tedy být studium toho, jak tento způsob aktivace ovlivňuje celkovou výpočetní sílu vzhledem ke standardním CD gramatickým systémům. Výzkum by se pak také mohl zabývat vztahem mezi počtem řízených komponent a vyjadřovací silou, tedy tím, roste-li vyjadřovací síla systému s počtem komponent, nebo jestli se od určitého počtu komponent síla systému nemění. V této práci pak není prezentován žádný algoritmus, který by dokázal převádět mezi navrženým systémem a nějakým jiným modelem, všechny zde popsané systémy jsou pak tvořeny intuitivně, pro aplikaci navržených metod by pak takový

algoritmus byl poměrně užitečný. Dalším otevřeným problémem jsou pak uzávěrové vlastnosti navrženého systému. Systém je s největší pravděpodobností uzavřen vůči sjednocení a konkatenaci, uzavřenost vůči dalším operacím, jako je průnik či doplněk však není známa.

Co se navržené metody syntaktické analýzy týče, je největším otevřeným problémem automatizovaná tvorba příslušného modelu na základě vstupních požadavků. Tento problém pak především vychází z neexistence algoritmu pro převod mezi navrženými systémy a jinými modely. Sestavit syntaktické analyzátory pro jednotlivé řízené komponenty je pak poměrně jednoduché, jelikož tyto komponenty používají jen lehce modifikovanou LR syntaktickou analýzu, stejně tak komponenta řídicí používá LL analýzu, kde tvorba příslušného analyzátoru je také automatizovatelná. Problémem je pak rozdělení celého zpracovávaného jazyka na tyto jednotlivé komponenty a následné přiřazení korektních derivačních módů. Zajímavou oblastí dalšího zkoumání by pak mohlo být porovnání různých metod počítání počtu použitých pravidel, jak bylo naznačeno v rámci příkladu v sekci 5.3.

Literatura

[1] Fernau H, Holzer M, Freund R.: Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization. *Theoretical Computer Science*, ročník 259, 2001.

[2] Lam, M.; Sethi, R.; Ullman, J.; aj.: *Compilers: Principles, techniques, and tools.* Addison-Wesley, 2006.

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

[4] Meduna, A.: *Formal Languages and Computation: Models and Their Applications.* Taylor & Francis, 2014, ISBN 978-1-4665-1345-7.

[5] Meduna A, Zemek P.: *Regulated grammars and automata.* Springer Science & Business Media, 2014, iSBN 978-1-4939-0369-6.

[6] Rozenberg G, Salomaa A.: *Hadbook of Formal Languages, vol.2.* Springer Science & Business Media, 1997, iSBN 3-540-60648-3.

[7] Wikipedia: Překladač — Wikipedia, The Free Encyclopedia. 2008, [Online; navštíveno 26. května 2015].
URL http://commons.wikimedia.org/wiki/File:Compiler-cs.svg

Příloha A

Obsah CD

Přiložný disk obsahuje:

- Tuto práci ve formátu PDF.
- Zdrojové soubory a obrázky nutné k přeložení této práce pomocí systému LaTeX.
- Demonstrační aplikaci.
- Soubory nutné k přeložení této aplikace.
- Sadu konfiguračích souborů s příklady pro demonstrační aplikaci.
- Článek, vycházejicí z této práce, který byl prezentován na konferenci Excel@FIT 2015.

Příloha B

Manuál

Součástí přiloženého disku je složka `app`. V této složce se nalézají soubory související s implementovanou aplikací. V této složce se nalézá spustitelný soubor `xmarti52_dip.jar`. Dále se zde nalézají konfigurační `.xml` soubory s jednotlivými příklady. V podsložce `app/src` se pak nalézají zdrojové soubory implementované aplikace. Ve složce `app/src` se nálézají složky `libs` a `src`, která obsahuje podlsožky `abego` a `xmarti52_dip`. Složky `libs` a `abego` obsahují zdrojové soubory a knihovy zodpovědné za vykreslování derivačních stromů. Tyto soubory jsou převzaty z https://code.google.com/p/treelayout/. Složka `xmarti52_dip` pak obsahuje vlastní zdrojové soubory implementované aplikace. Ve složce `app/src` se pak nalézá také konfigurační soubor pro systém `ant`. Překlad zdrojových souborů lze provést pomocí příkazu `ant`, použitého ve složce `app/src`. Překlad pak vytvoří spustitelný soubor `xmarti52_dip.jar`, který je možno spustit příkazem `ant run`, případně přímo ze složky `app/src/jar`.

Po spuštění aplikace je nejprve nutno nastavit vstupní konfigurační soubor pro gramatický systém. Dále je třeba nastavit vstupní řetězec, který bude analyzován. Tato nastavení je možno provést pomocí tlačítek v horní části obrazovky. Po nastavení těchto vstupních parametrů je možno systém inicializovat. Po inicializaci je možno krokovat analýzu pomocí tlačítka `Make Step`. Při každé změně vstupního souboru nebo řetězce je třeba systém znovu inicializovat. Pro restart výpočtu pak slouží také tlačítko `Initialize`, přičemž není nutné znova nastavovat vstupní hodnoty. Případné syntaktické chyby se zobrazují v horní části hlavního okna. Vstupní řetězec je pak zadávan jako sekvence tokenů. Za token je pak považován každý znak. Víceznakové tokeny nejsou podporovány. Symbol `$` je rezervován pro reprezentaci konce vstupu. Lze jej také použít k ručnímu rozdělení vstupního řetězce mezi jednotlivé komponenty.

Aplikace slouží jen pro demonstrační účely a není žádným způsobem optimalizovaná pro složité gramatické systémy a dlouhé vstupní řetězce a není pro taková vstupní data zamýšlena.

Příloha C

Ukázka konfiguračního souboru implementované aplikace

Níže uvedený konfigurační soubor odpovídá jazyku $L_3^{'} = \{a^i b^{2^i} c^i | i \geq 1\}$ ze sekce 5.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<system>
    <grammar id="0">
        <symbol starting="true" nonterminal="true" >S</symbol>
        <symbol starting="true" nonterminal="false" >A</symbol>
        <symbol starting="true" nonterminal="false" >B</symbol>
        <symbol starting="true" nonterminal="false" >C</symbol>
        <rule id="0">
            <lhs >S</lhs>
            <rhs mode="T" modeval="3">A</rhs>
            <rhs mode="C" modeval="1">B</rhs>
            <rhs mode="C" modeval="1">C</rhs>
        </rule>
        <tables>
            <LL>
                <nonterm sym="S">
                    <term sym="a">0</term>
                </nonterm>
            </LL>
        </tables>
    </grammar>
    <grammar id="1" multiplication="true">
        <symbol starting="false" nonterminal="true" >SA</symbol>
        <symbol starting="true" nonterminal="true" >A</symbol>
        <symbol starting="false" nonterminal="false" >a</symbol>
        <symbol starting="false" nonterminal="false" ></symbol>
        <rule id="0" >
            <lhs >A</lhs>
            <rhs >A</rhs>
        </rule>
        <rule id="1" value="2">
```

```
        <lhs>A</lhs>
        <rhs>a</rhs>
        <rhs>A</rhs>
    </rule>
    <rule id="2" value="1">
        <lhs>A</lhs>
        <rhs></rhs>
    </rule>
    <tables>
        <action>
            <state st="0">
                <symbol sym="a" val="2">shift</symbol>
                <symbol sym="$" val="2">reduction</symbol>
            </state>
            <state st="1">
                <symbol sym="$" val="1">success</symbol>
            </state>
            <state st="2">
                <symbol sym="a" val="2">shift</symbol>
                <symbol sym="$" val="2">reduction</symbol>
            </state>
            <state st="3">
                <symbol sym="$" val="1">reduction</symbol>
            </state>
        </action>
        <goto>
            <state st="0">
                <symbol sym="A">1</symbol>
            </state>
            <state st="2">
                <symbol sym="A">3</symbol>
            </state>
        </goto>
    </tables>
</grammar>
<grammar id="2" >
    <symbol starting="false" nonterminal="true" >SB</symbol>
    <symbol starting="true" nonterminal="true" >B</symbol>
    <symbol starting="false" nonterminal="false" >b</symbol>
    <symbol starting="false" nonterminal="false" ></symbol>
    <rule id="0">
        <lhs >B</lhs>
        <rhs >B</rhs>
    </rule>
    <rule id="1">
        <lhs>B</lhs>
        <rhs>b</rhs>
        <rhs>B</rhs>
```

```
    </rule>
    <rule id="2" value="0">
        <lhs>B</lhs>
        <rhs></rhs>
    </rule>
    <tables>
        <action>
            <state st="0">
                <symbol sym="b" val="2">shift</symbol>
                <symbol sym="$" val="2">reduction</symbol>
            </state>
            <state st="1">
                <symbol sym="$" val="1">success</symbol>
            </state>
            <state st="2">
                <symbol sym="b" val="2">shift</symbol>
                <symbol sym="$" val="2">reduction</symbol>
            </state>
            <state st="3">
                <symbol sym="$" val="1">reduction</symbol>
            </state>
        </action>
        <goto>
            <state st="0">
                <symbol sym="B">1</symbol>
            </state>
            <state st="2">
                <symbol sym="B">3</symbol>
            </state>
        </goto>
        <startState>0</startState>
    </tables>
</grammar>
<grammar id="3" multiplication="true">
    <symbol starting="false" nonterminal="true" >SC</symbol>
    <symbol starting="true" nonterminal="true" >C</symbol>
    <symbol starting="false" nonterminal="false" >c</symbol>
    <symbol starting="false" nonterminal="false" ></symbol>
    <rule id="0">
        <lhs >C</lhs>
        <rhs >C</rhs>
    </rule>
    <rule id="1" value="2">
        <lhs>C</lhs>
        <rhs>c</rhs>
        <rhs>C</rhs>
    </rule>
    <rule id="2" value="1">
```

```
            <lhs>C</lhs>
            <rhs></rhs>
        </rule>
        <tables>
            <action>
                <state st="0">
                    <symbol sym="c" val="2">shift</symbol>
                    <symbol sym="$" val="2">reduction</symbol>
                </state>
                <state st="1">
                    <symbol sym="$" val="1">success</symbol>
                </state>
                <state st="2">
                    <symbol sym="c" val="2">shift</symbol>
                    <symbol sym="$" val="2">reduction</symbol>
                </state>
                <state st="3">
                    <symbol sym="$" val="1">reduction</symbol>
                </state>
            </action>
            <goto>
                <state st="0">
                    <symbol sym="C">1</symbol>
                </state>
                <state st="2">
                    <symbol sym="C">3</symbol>
                </state>
            </goto>
            <startState>0</startState>
        </tables>
    </grammar>
</system>
```