VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

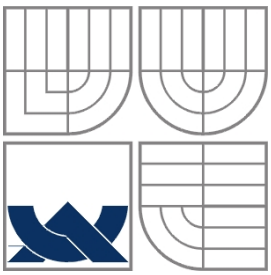
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KOMBINOVANÁ SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA GRAMATICKÝCH SYSTÉMECH

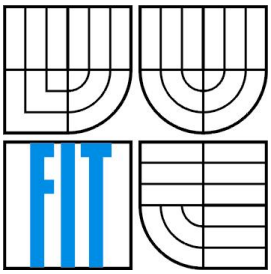
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR CAHA
AUTHOR

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

KOMBINOVANÁ SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA GRAMATICKÝCH SYSTÉMECH

COMBINED PARSING BASED ON GRAMMAR SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR CAHA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2015

Abstrakt

Tato práce se zabývá kombinovanou syntaktickou analýzou založenou na gramatických systémech. Zavádí klasické modifikované metody gramatických systémů. Nejprve budou teoreticky popsané a v další části implementované v syntaktickém analyzátoru. Základem analyzátoru je CD gramatický systém. Implementace využívá rekursivní sestup a precedenční analýzu. Analyzátor je universální, použitelný pro jakékoli gramatické systémy založené na bezkontextových a některých ne bezkontextových.

Abstract

This thesis deals with a combined parsing based on grammar systems. Introduces modified method of classical grammar systems. At first they will be theoretically described and in the next part they will be implemented for parsing. The basis for the parser is a cooperating distributed grammar system. Implementation uses recursive method and case analysis. The parser is universal, applicable to any grammar systems based on context-free and some not context-free.

Klíčová slova

gramatiky, gramatické systémy, CD gramatické systémy, syntaktický analyzátor, syntaktická analýza

Keywords

Grammar, Grammar systems, cooperating distributed grammar system, parser, syntax analysis

Citace

Petr Caha: Kombinovaná syntaktická analýza založená na gramatických systémech, bakalářská práce, Brno, FIT VUT v Brně, 2015

Kombinovaná syntaktická analýza založená na gramatických systémech

Prohlášení

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

……
Petr Caha
20. května 2015

Poděkování

Chtěl bych poděkovat prof. RNDr. Alexandru Medunovi, CSc., který tuto bakalářskou práci vedl a směroval.

Obsah

Obsah....1
1 Úvod....2
2 Základní pojmy....4
2.1 Abeceda....4
2.2 Jazyky....5
2.3 Gramatika....6
2.3.1 Derivační krok a sekvence derivačních kroků....6
2.3.2 Generovaný jazyk....7
2.3.3 Chomského hierarchie....7
3 Gramatické systémy....9
3.1 CD gramatické systémy....9
3.1.1 Derivace....10
3.1.2 Generovaný jazyk....11
3.2 PC gramatické systémy....12
3.2.1 Derivace....12
3.2.2 Generovaný jazyk....13
4 Syntaktická analýza....14
4.1 Analýza shora dolů....14
4.1.1 Syntaktická analýza založená na rekursivním sestupu....17
4.1.2 Prediktivní syntaktická analýza....17
4.2 Analýza zdola nahoru....18
4.2.1 Precedenční syntaktický analyzátor....18
4.2.2 LR syntaktický analyzátor....19
5 Implementace....21
5.1 Úvod....21
5.2 Rozvržení aplikace....21
5.2.1 Třída GetOpt....21
5.2.2 Třída RegEx....22
5.2.3 Třída Xml....22
5.2.4 Třída Lexer....22
5.2.5 Třída Parser....22
5.3 Lexikální analýza....22
5.4 Syntaktická analýza....23
5.5 Formát konfiguračních souborů....23
5.5.1 Lexikální analýza....23
5.5.2 Syntaktická analýza....24
5.6 Zpracování chyb....24
6 Závěr....25
Literatura....27
Seznam příloh....28

1 Úvod

Jazyky jsou prostředkem komunikace už od počátků lidstva. Přirozený jazyk lidí je velmi silný a má velice dobrou vyjadřovací schopnost zdokonalenou dlouholetým vývojem. Strojové zpracování takovýchto jazyků je však pro současné počítače velice komplikované a algoritmy zpracovávající přirozený lidský jazyk jsou velice složité a rozsáhlé. Proto byly v oblasti teoretické informatiky zavedeny formální jazyky.

Formální jazyky jsou zjednodušením jazyků přirozených. Tyto formální jazyky se používají pro popis dat i popis jejich způsobu zpracování. Staly se tak nedílnou součástí teoretické informatiky a jsou používány v nepřeberném množství jak pro definici programovacích jazyků, tak i pro definici dat a celých datových struktur. Formální jazyky se skládají z dalších menších dílčích částí. Tyto části jako jsou bezkontextové, kontextové, regulární, rekursivně spočetné a jiné, jsou vždy podmnožinou formálních jazyků samotných.

Bezkontextové formální jazyky a bezkontextové gramatiky na nich vystavěných jsou podmnožinami formálních jazyků. Tyto bezkontextové formální jazyky mají jednoduché bezkontextové gramatiky a jejich zpracování má jednoduší implementaci než u složitějších jazyků. Avšak pro oblast zpracování programovacích jazyků neposkytují vždy dostatečnou vyjadřovací sílu, která by byla pro daný programovací jazyk vhodná. Pro možnost zpracovávat i složitější formální jazyky, ale zachovat jednoduché gramatiky, byly v teoretické informatice objeveny jiné možnosti zpracování, jako jsou gramatické systémy.

Gramatické systémy byly objeveny za účelem dosáhnout se skupinou jednodušších jazyků lepšího komplexního výsledku než u jednotlivé jednoduché gramatiky. Tyto gramatické systémy jsou složeninou jednotlivých gramatických systémů, kdy každá z jednotlivých gramatik má své vlastní pravidla a svou vlastní vyjadřovací sílu, ale základní množina neterminálů a terminálů všech gramatik v gramatickém systému je sdílená a tedy stejná. I přesto, že jednotlivé gramatiky v jednom gramatickém systému mohou být velice jednoduché, sloučením těchto jednoduchých gramatik do jednoho celku v podobě gramatického systému jim dodává větší vyjadřovací sílu, než jakou by měly samostatné gramatiky. Jak gramatiky samostatné tak i celé gramatické systémy se používají především k syntaktické analýze.

Syntaktická analýza je jednou z několika částí vlastního procesu překladu vstupního řetězce na výstupní. Syntaktická analýza je obvykle prováděna za lexikální analýzou, která ze vstupního řetězce vytvoří lexémy. Lexémy jsou posléze přetvořeny v tokeny. Tokeny jsou nakonec postupně, nebo naráz v orientovaném seznamu, předány syntaktickému analyzátoru. Syntaktický analyzátor převezme tokeny vytvořené lexikálním analyzátorem a za pomocí gramatik a gramatických systému, pomocí jejich gramatických pravidel, přetvoří tento seznam tokenů na derivační strom. Derivační strom je posléze předán sémantickému analyzátoru, který jej přetvoří v abstraktní syntaktický strom. Abstraktní syntaktický strom už je datová struktura připravená pro vlastní generování vnitřního kódu v generátoru vnitřního kódu. Vygenerovaný vnitřní kód je potom předán dál pro optimalizaci a generování výsledného cílového kódu. Tento cílový kód bývá reprezentován povětšinou binární formou.

Vlastní syntaktická analýza je potom proces, který má za úkol zjistit, zda pro vstupní řetězec existuje odpovídající derivační strom za použití gramatických pravidel z gramatik nebo gramatických systémů. Syntaktická analýza tedy určuje zda zadaný vstupní řetězec odpovídá zadanému formálnímu jazyku, zadané gramatice nebo celému gramatickému systému. Vlastní proces syntaktické analýzy je potom prováděn pomocí příslušného gramatického automatu.

Tato bakalářská práce je zaměřena na vytvoření kombinované syntaktické analýzy, která bude založena na gramatických systémech. Syntaktická analýza je zaměřená především na obecné zpracovávání bezkontextových struktur a některých kontextových. Pro potřeby syntaktické analýzy je nutno mít libovolný vhodný vlastní lexikální analyzátor.

Kapitola 2 slouží jako úvod k dané problematice. Jsou zde prezentovány potřebné základní známé pojmy na kterých další části práce následně staví.

Kapitola 3 zavádí obecné gramatické systémy. Tyto gramatické systémy staví na znalostech předešlé kapitoly a dále je rozšiřují. Dále zavádí modifikace obecných gramatických systémů. Z těchto gramatických systémů a jejich modifikací pak tato práce vychází.

Kapitola 4 popisuje obecnou syntaktickou analýzu. Dále se tato kapitola zabývá jednotlivými základními implementačními možnostmi syntaktické analýzy.

Kapitola 5 popisuje vlastní implementaci konečného kombinovaného syntaktického analyzátoru založeného na gramatických systémech.

Závěr 6 pak obsahuje shrnutí dosažených dílčích závěrů a návrhy na možné další pokračování a rozšíření tohoto kombinovaného syntaktického analyzátoru založeného na gramatických systémech.

2 Základní pojmy

V této kapitole jsou popsány základní pojmy potřebné pro prezentaci další látky. Jelikož se jedná o základní pojmy a tato práce staví primárně na systémech, které staví na těchto pojmech, jsou tyto pojmy popsány jen formou definic a krátkého popisu. Nejprve nadefinujeme Abecedu jako takovou, nad kterou postavíme jazyky, které dál rozšíříme do gramatik. Gramatiky jsou už nedílnou součástí této práce. U všech pod-částí jsou také pro úplnost popsány jednotlivé operace. Ať už se jedná o abecedu, jazyky nebo gramatiky.

2.1 Abeceda[1]

Tak jako v každodenním životě, abeceda je základ všeho textu. Popisuje základní sadu znaků s kterými je potom možno dále pracovat. Operace nad abecedou potom představují četnost výskytu jednoho daného písmene sám za sebou, nebo též opakování znaku. Operace s řetězci nad abecedou jsou potom už skládání jednotlivých písmen do skupin, kterým říkáme řetězce. Ve skutečném životě by to potom byly slova, věty nebo celé stránky textu, jelikož do řetězce patří i symboly mezery nebo taky nového řádku. S těmito řetězci je pak dále možné provádět další operace stejně jako se symboly. Spojovat nebo naopak odpojovat od sebe, obracet, opakovat a další.

Definice 2.1. Abeceda je libovolná konečná neprázdná množina. Prvky abecedy nazýváme symboly.

Definice 2.2. Nechť $\sum$ je abeceda.

- ε je řetězec nad abecedou $\sum$
- pokud x je řetězec nad $\sum$ a $a \in \Sigma$, potom xa je řetězec nad abecedou $\sum$

Definice 2.3. Nechť x je řetězec nad abecedou $\sum$. Délka řetězce x, $|x|$, je definována:

- pokud $x = \varepsilon$, pak $|x| = 0$
- pokud $x = a_1, ..., a_n$, pak $|x| = n$ pro $n \geq 1$ a $a_i \in \Sigma$ pro všechny $i = 1, ..., n$

Definice 2.4. Nechť x a y jsou dva řetězce nad abecedou $\sum$. Konkatenace x a y je řetězec xy.

Definice 2.5. Nechť x je řetězec nad abecedou Pro $\sum$. $i \geq 0$, i-tá mocnina řetězce x, x^i je definována:

- $x^0 = \varepsilon$
- pro $i \geq 1$: $x^i = xx^{i-1}$

Definice 2.6. Nechť x je řetězec nad abecedou $\sum$. Reverzace řetězce x, *reversal(x)*, je definována:

- pokud $x = \varepsilon$ pak $reversal(\varepsilon) = \varepsilon$
- pokud $x = a_1, ..., a_n$ pak $reversal(a_1, ..., a_n) = a_n, ..., a_1$ pro $n \geq 1$ a $a_i \in \Sigma$ pro všechna $i = 1, ..., n$

Definice 2.7. Nechť *x* a *y* jsou dva řetězce nad abecedou ∑. *x* je prefixem *y*, pokud existuje řetězec *z* nad abecedou ∑, přičemž platí *xz = y.*

Definice 2.8. Nechť *x* a *y* jsou dva řetězce nad abecedou ∑. *x* je sufixem *y*, pokud existuje řetězec *z* nad abecedou ∑, přičemž platí *zx = y.*

Definice 2.9. Nechť *x* a *y* jsou dva řetězce nad abecedou ∑. *x* je podřetězcem *y*, pokud existují řetězce *z*, *z'* nad abecedou ∑, přičemž platí *zxz' = y.*

2.2 Jazyky[1]

Obdobně jako je tomu u abecedy i jazyky a jejich myšlenka je založená na jazycích reálných a jsou jim tedy v některých ohledech velmi podobné. Jednoduchým výčtem všech možných řetězců, které do daného jazyka patří, dostaneme jazyk samotný. Jazyky mohou být jak konečné tak i nekonečné. Konečný jazyk je takový, které má omezené množství řetězců, které tento jazyk obsahuje. Jazyk nekonečný je potom takový, který má dané pouze limity ve kterých se musí daný řetězec nacházet, aby se jednalo o řetězec tohoto jazyka.

Definice 2.10. Nechť $\sum^*$ značí množinu všech řetězců nad ∑. Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad ∑.

Definice 2.11. Jazyk L je konečný, pokud L obsahuje konečný počet řetězců, jinak je nekonečný.

Definice 2.12. Nechť L_1 a L_2 jsou dva jazyky nad ∑. Sjednocení jazyků L_1 a L_2, $L_1 \cup L_2$, je definováno: $$L_1 \cup L_2 = \{x : x \in L_1 \textbf{ nebo } x \in L_2\}$$

Definice 2.13. Nechť L_1 a L_2 jsou dva jazyky nad ∑. Průnik jazyků L_1 a L_2, $L_1 \cap L_2$, je definován: $$L_1 \cap L_2 = \{x : x \in L_1 \textbf{ a } x \in L_2\}$$

Definice 2.14. Nechť L_1 a L_2 jsou dva jazyky nad ∑. Rozdíl jazyků L_1 a L_2, $L_1 - L_2$, je definován: $$L_1 - L_2 = \{x : x \in L_1 \textbf{ a } x \notin L_2\}$$

Definice 2.15. Nechť *L* je jazyk nad abecedou ∑. Doplněk jazyka L, $\bar{L}$, je definován: $$\bar{L} = \Sigma^* - L$$

Definice 2.16. Nechť L_1 a L_2 jsou dva jazyky nad ∑. Konkatenace jazyků L_1 a L_2, L_1L_2, je definována: $$L_1L_2 = \{xy : x \in L_1 \textbf{ a } y \in L_2\}$$

Definice 2.17. Nechť L je jazyk nad abecedou ∑. Reverzace jazyka *L*, *reverse(L)*, je definována: $$reverse(L) = \{reverse(x) : x \in L\}$$

Definice 2.18. Nechť L je jazyk nad abecedou $\sum$. Pro $i \geq 0$, *i*-tá mocnina jazyka *L*, L^i, je definována:

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

Definice 2.19. Nechť L je jazyk nad abecedou $\sum$. Iterace jazyka *L*, L^*, a pozitivní iterace jazyka *L*, L^+, jsou definovány: $L^* = \bigcup_{i=0}^{\infty} L^i$, $L^+ = \bigcup_{i=1}^{\infty} L^i$

2.3 Gramatika[6]

Lepším a jednotným způsobem jak definovat nekonečný jazyk, než jeho nekonečným výčtem, je gramatika. Gramatika je založena na konečné množině neterminálů a terminálů, konečné množině gramatických pravidel a počátečním neterminálu. Používáním těchto pravidel v gramatickém automatu potom generujeme řetězce daného jazyka. Výsledný generovaný jazyk je potom zcela závislý na zvolených pravidlech, avšak jedná se zpravidla o jazyky nekonečné. Pro jazyky konečné se obvykle použije prostý výčet všech možných řetězců spadajících do daného jazyka.

Definice 2.20. Bezkontextová gramatika je čtveřice G = (N, T, P, S), kde

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

2.3.1 Derivační krok a sekvence derivačních kroků

Detailním rozborem gramatiky zjistíme, že je potřeba používání jednotlivých pravidel řídit a také je především umět popsat. Takovým popisem procesu použití pravidla je derivační krok. Derivačním krokem se rozumí zaměnění vstupního řetězce daného derivačního kroku použitím gramatického pravidla za výstupní řetězec, který bude v dalším derivačním kroku použit opět jako vstupní.

Sekvence derivačních kroků je potom chápána jako několik derivačních kroků po sobě, které se navenek mohou tvářit i jako jeden derivační krok. Mají svůj vstup i výstup, avšak používají více pravidel za dosažením výstupu. Sekvence derivačních kroků využívají derivační kroky ze kterých se skládají a gramatická pravidla, které tyto kroky používají.

Definice 2.21. Nechť *G = (N, T, P, S)* je bezkontextová gramatika. Nechť $u, v \in (N \cup T)^*$ a $p = A \rightarrow x \in P$. Potom *uAv* přímo derivuje *uxv* za použití *p* v *G*, zapsáno $uAv \Rightarrow uxv[p]$ nebo zjednodušeně $uAv \Rightarrow uxv$.

Definice 2.22. Nechť $u \in (N \cup T)^*$. *G* provede nula derivačních kroků z *u* do *u*; zapisujeme: $u \Rightarrow^0 u[\varepsilon]$ nebo zjednodušeně $u \Rightarrow^0 u$

Definice 2.23. Nechť $u_0, \ldots, u_n \in (N \cup T)^*$, $n \geq 1$ *a* $u_{i-1} \Rightarrow u_i[p_i]$, $p_i \in P$ pro všechna *i = 1, ..., n,* což znamená: $u_0 \Rightarrow u_1[p_1] \Rightarrow u_2[p_2] \ldots \Rightarrow u_n[p_n]$. *Pak G* provede *n* derivačních kroků z u_0 do u_n, zapisujeme: $u_0 \Rightarrow^n u_n[p_1, \ldots, p_n]$ nebo zjednodušeně $u_0 \Rightarrow^n u_n$.

2.3.2 Generovaný jazyk

Výsledkem po provedení všech možných derivačních kroků a/nebo sekvencí derivačních kroků je množina jazyků, kterou nazýváme generovaný jazyk. Tento výsledný generovaný jazyk se skládá z terminálů, které reprezentují jednotlivé části formálního jazyka. Gramatika *G* generuje řetězec terminálů *w* pomocí sekvence derivačních kroků z *S* do *w*.

Definice 2.24. Nechť gramatika *G = (N, T, P, S)* je bezkontextová gramatika. Jazyk generovaný bezkontextovou gramatikou *G*, *L(G),* je definován: $L(G)=\{w : w \in T^*, S \Rightarrow^* w\}$

Definice 2.25. Nechť *L* je jazyk. *L* je bezkontextový jazyk, pokud existuje bezkontextová gramatika, která generuje tento jazyk *L*.

2.3.3 Chomského hierarchie

Výsledný jazyk jednotlivých gramatik se může velice lišit. Může být jednoduchý, ale i velice složitý. Proto aby se tento komplexní blok dal dále dělit na menší souvislé celky stanovil Noam Chomsky čtyři typy gramatik, které se liší tvarem pravidel a specializací. [4]

Jednotlivé typy gramatik se od sebe liší nejenom tvarem pravidel, ale v důsledku i způsobem zpracování a možného reálného nasazení pro potřeby formálních jazyků. Základním jednoduchým typem jsou gramatiky generující regulární jazyky, které je možno jednoduše popsat pomocí obyčejného regulárního výrazu. Tyto regulární jazyky jsou omezené především díky absenci rekurze nebo zásobníku a proto s nimi není možné zpracovávat opakující se souvislé bloky řetězců.

Dalším o trochu složitějším typem jsou gramatiky, které generují bezkontextové jazyky, které je možno, popsat pomocí regulárního automatu se zásobníkem. Díky tomuto zásobníku je u tohoto typu již možné zpracovávat i souvislé opakující se celky řetězců.

Gramatiky generující kontextové jazyky už není možné zpracovávat pomocí regulárního automatu se zásobníkem, pro takové jazyky už je potřeba jiných metod, stejně tak pro první a nejobsáhlejší skupinu gramatik generujících rekurzivně spočetné jazyky.

- Gramatiky **Typ-0** , Rekurzivně spočetné jazyky, Neomezené gramatiky, používají pravidla typu:

$$x \rightarrow y, x \in (N \cup T)^* N (N \cup T)^*, y \in (N \cup T)^*$$

- Gramatiky **Typ-1** , Kontextové jazyky, Kontextové gramatiky, používají pravidla typu:

$$x \rightarrow y, x \in (N \cup T)^* N (N \cup T)^*, y \in (N \cup T)^*, |x| \leq |y|$$

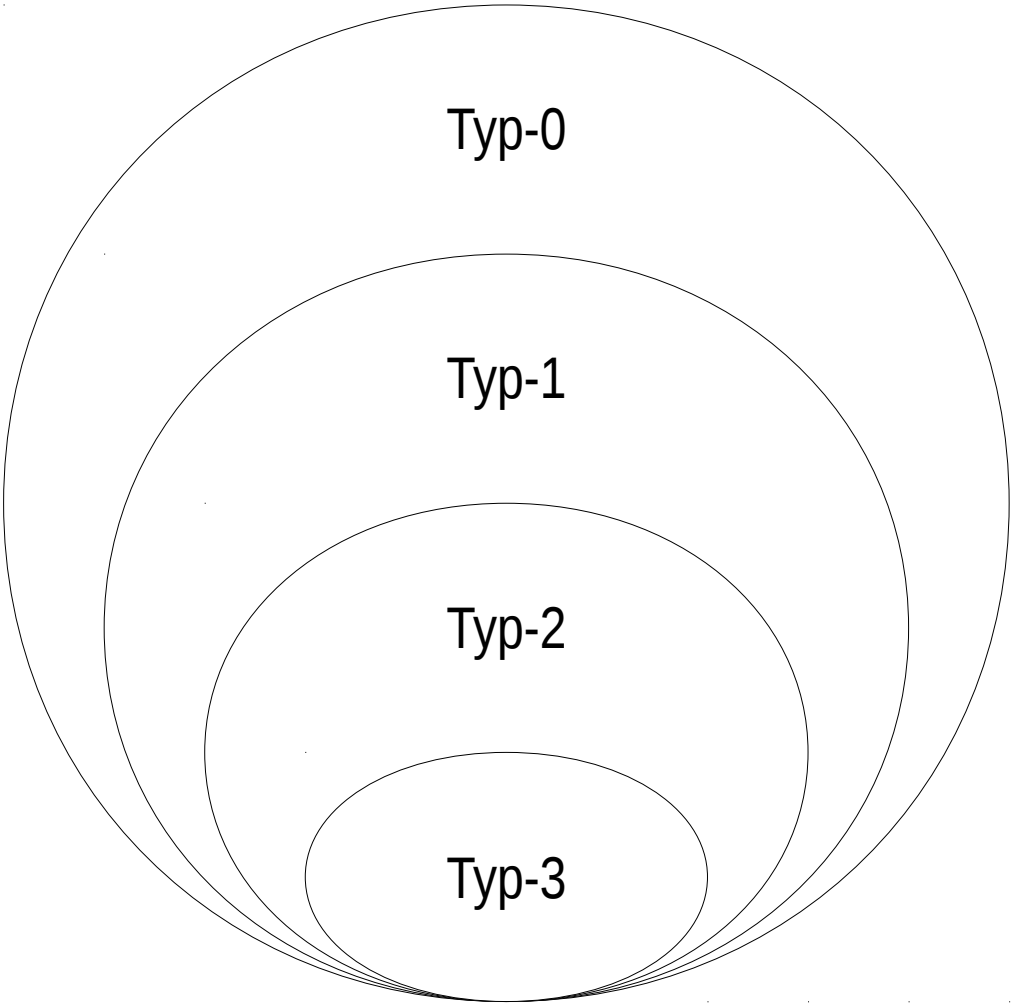
- Gramatiky **Typ-2** , Bezkontextové jazyky, Bezkontextové gramatiky, používají pravidla typu:

$$x \rightarrow y, x \in N, y \in (N \cup T)^*$$

- Gramatiky **Typ-3** , Regulární jazyky, Pravé linearní gramatiky, používají pravidla typu:

$$x \rightarrow y, x \in N, y \in T^* \cup T^* N$$

Obrázek 2.1. Typy formálních jazyku jak je stanovil Noam Chomsky.



$Typ-3 \subset Typ-2 \subset Typ-1 \subset Typ-0$ [5]

3 Gramatické systémy[6]

Dalším pokročilejším způsobem jak zpracovávat bezkontextové jazyky, ale i jazyky ostatních typů, jsou gramatické systémy. Gramatické systémy umožňují pohodlnější zpracování vícero jazyků současně a to včetně možnosti zpracování vícero typů jazyků současně. Gramatický systém je soubor gramatik GS = (G_1, G_2, G_3, ..., G_n). Díky obsahu více gramatik v jednom komplexním celku je možné gramatické systémy zpracovávat taky paralelně, tedy současně více gramatik najednou. více viz. Kapitola 3.2. Gramatické systémy dělíme do dvou skupin: sekvenční a paralelní. Avšak všechny gramatiky obsažené v jednom gramatickém systému obsahují stejné neterminály a terminály včetně startovního neterminálu. Liší se tedy pouze v používaných pravidlech. V případě pokročilejších gramatických systémů, jako jsou například paralelní gramatické systémy, se mohou lišit i v přidaných dalších skupinách symbolů. Tyto další skupiny symbolů mohou být používány například pro vlastní řízení celého gramatického systému.

3.1 CD gramatické systémy

Základním jednoduchým příkladem gramatického systému jsou spolupracující distribuované gramatické systémy, zkráceně CD gramatické systémy. CD gramatické systémy jsou zpracovávány sekvenčně, tedy postupně. Jednotlivá pravidla každé gramatiky v gramatickém systému jsou tedy zpracovávána postupně jedno za druhým. Až se s jednou gramatikou skončí přejde se na další. To znamená, že je vždy aktivní pouze jedna gramatika a jedná se tedy o gramatické systémy sekvenční.

CD gramatický systém stupně *n*, kde $n \geq 1$, je *construct*

$$\Gamma = (N, T, S, P_1, ..., P_n)$$

, kde:

- *N, T, S* je definováno jako obvykle*:*
  - N je abeceda neterminálů,
  - T je abeceda terminálů
  - *S* je počáteční neterminál, $S \in N$
- P_i je konečná množina bezkontextových pravidel pro každé $i \in \{1, ..., n\}$, nazývaná *componenty* Γ

potom *i-tá* gramatika v gramatickém systému je $G_i = (N, T, P_i, S)$

3.1.1 Derivace

Pro každé *i = 1, ..., n,* je konečná derivace podle i-té komponenty

$$x \Rightarrow_i^t y$$

právě tehdy, pokud platí obě následující pravidla:

- existuje n derivací z *x* do *y* $x \Rightarrow^n y$, pro $n \geq 0$, v gramatice G_i = (N, T, P_i, S), nebo zjednodušeně $x \Rightarrow^* y$
- neexistuje žádná derivace z *y* do *z* $y \Rightarrow z$ pro všechna $z \in (N \cup T)^*$

Právě *k*-tá derivace v gramatice $G_i = (N, T, P_i, S)$ je:

$$x \Rightarrow_i^{=k} y$$

právě tehdy, pokud platí následující pravidlo:

- existuje právě *k* derivací z *x* do *y* $x \Rightarrow^k y$ v gramatice $G_i = (N, T, P_i, S)$

Nanejvýš *k*-tá derivace v gramatice $G_i = (N, T, P_i, S)$ je:

$$x \Rightarrow_i^{\leq k} y$$

právě tehdy, pokud platí následující podmínka:

- existuje nanejvýš *k* derivací z *x* do *y* $x \Rightarrow^j y$ v gramatice $G_i = (N, T, P_i, S)$, pro nějaké $j \leq k$

Nejméně *k*-tá derivace v gramatice $G_i = (N, T, P_i, S)$ je:

$$x \Rightarrow_i^{\geq k} y$$

právě tehdy, pokud platí následující podmínka:

- existuje nejméně *k* derivací z *x* do *y* $x \Rightarrow^j y$ v gramatice $G_i = (N, T, P_i, S)$, pro nějaké $j \geq k$

3.1.2 Generovaný jazyk

Množina všech možných derivačních způsobů CD gramatického systému *D*:

$$D = \{*, t\} \cup \{\leq k, = k, \geq k, : k = 1, 2, 3, ...\}$$

Množina možných derivačních způsobů CD gramatického systému je:

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

Výsledný generovaný jazyk CD gramatického systému je potom následující:

$L_f(\Gamma) = \{w \in T^*\}$ tam jsou $v_0, v_1, ..., v_m$ takové, že:

- $v_i \in F(G_{j_i}, v_{i-1}, f)$ pro $i = 1, ..., m$ a $j_i \in \{1, ..., n\}$
- $v_0 = S$, $v_m = w$, pro nějaké $m \geq 1$

Příklad 3.1.

Máme gramatický systém $\Gamma = (\{S, A\}, \{a\}, S, P_1, P_2, P_3)$, *kde:*

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

generovaný jazyk gramatického systému Γ je potom:

$L_t(\Gamma) = \{ a^{2^n} \text{ kde } n \geq 1 \}$

Příklad 3.2.

Máme gramatický systém Γ = ({S, A, A' , B, B'}, {a, b, c}, *S*, P_1 , P_2), *kde:*

$P_1 = \{S \rightarrow S, S \rightarrow AB, A' \rightarrow A, B' \rightarrow B\}$
$P_2 = \{A \rightarrow aA'b, B \rightarrow cB', A \rightarrow ab, B \rightarrow c\}$

generovaný jazyk gramatického systému Γ je potom:

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

3.2 PC gramatické systémy

Další modifikací gramatických systémů jsou paralelně komunikující gramatické systémy, zkráceně PC gramatické systémy. PC gramatické systémy, jak už název napovídá, umožňují paralelní zpracování jednotlivých gramatik. Pro potřeby paralelního zpracování je ovšem potřeba gramatické systémy rozšířit o další aspekty.

Prvním takovým je nová další skupina symbolů, která zajišťuje komunikaci mezi jednotlivými procesy zpracovávajícími jednotlivé gramatiky. Pro potřeby těchto nových symbolů je také potřeba stanovit jejich stálé místo ve zpracování gramatik. Proto byl vlastní proces derivačních kroků rozdělen na dva jednoduší celky.

V PC gramatických systémech tedy máme předně generující derivační kroky, tak jak je známe také z CD gramatických systémů, nebo z obyčejných gramatik. Nazývané též g-krok. Vedle nich máme také nově komunikační derivační kroky, které zajišťují samostatnou komunikaci mezi více procesy. Nazývané také jako c-krok. Generující derivační kroky potom používají, stejně jako obyčejné gramatiky, neterminály a terminály pro nahrazování vstupního řetězce za výstupní. G-krok probíhá vždy v rámci jednoho procesu zpracování. Komunikační derivační kroky využívají nově zavedených komunikačních symbolů, které prování náhradu vstupního řetězce za výstupní. Ovšem komunikační c-krok provádí náhradu napříč procesy zpracovávajícími vstupní řetězce.

PC gramatický systém stupně *n*, kde $n \geq 1$, je *construct*

$$\Gamma = (N, K, T, (S_1, P_1), ..., (S_n, P_n))$$

, kde:

- *N, T* je definováno jako obvykle*:*
  - N je abeceda neterminálů,
  - T je abeceda terminálů
- S_i je počáteční neterminál *i*-té komponenty, $S_i \in N$ pro všechna *i = 1, ..., n*
- P_i je konečná množina bezkontextových pravidel tvaru $A \rightarrow x$, kde $A \in N$ a $x \in (N \cup T \cup K)^*$ pro všechna *i = 1, ..., n*

3.2.1 Derivace

Derivace u PC gramatických systémů je rozdělena na dvě části. V první části, která se nazývá generující, se provádí generování tak jako u klasické gramatiky nebo u CD gramatických systémů. V části druhé, která se nazývá komunikační, se provádí komunikace mezi jednotlivými běžícími gramatikami. Tato komunikace má za účel synchronizaci a předávání výsledku jedné gramatiky gramatice jiné, která tento výsledek původní gramatiky připočte ke svému zpracovávanému řetězci terminálů. PC gramatický systém preferuje vždy komunikační kroky před těmi generujícími.

O generující derivační krok se jedná když:

- Buď $x_i \Rightarrow y_i$ v gramatice $G_i = (N \cup K, T, P_i, S_i)$ pro všechna $1 \leq i \leq n$
- nebo $x_i = y_i \in T^*$ pro všechna $1 \leq i \leq n$

Komunikační krok je potom definován takto:

- Množina $z_i = x_i$ pro všechna $i = 1, ..., n$

Pro každé $i = 1, ..., n$, pokud platí $alph(x_i) \cap K \neq \emptyset$ a pro každé Q_j v x_i $alph(x_j) \cap K = \emptyset$, potom pro každé Q_j v x_i :

1. množina $z_j = S_j$,
2. nahradit Q_j s x_j v x_i
3. množina z_j na řetězec vyplývající z (2.)

Příklad 3.3.

Máme gramatický systém Γ = ({S_1 , S'_1 , S_2 , S_3}, K , {a, b, c}, (S_1 , P_1), (S_2 , P_2), (S_3 , P_3)), kde:

$P_1 = \{S_1 \rightarrow abc, S_1 \rightarrow a^2b^2c^2, S_1 \rightarrow aS'_1, S_1 \rightarrow a^3Q_2, S'_1 \rightarrow aS'_1, S'_1 \rightarrow a^3Q_2, S_2 \rightarrow b^2Q_3, S_3 \rightarrow c\}$
$P_2 = \{ S_2 \rightarrow bS_2 \}$
$P_3 = \{ S_3 \rightarrow cS_3\}$

Jednotlivé derivace potom jsou:

$(S_1, S_2, S_3) \Rightarrow (aS'_1, bS_2, cS_3) \Rightarrow^* (a^nS'_1, b^nS_2, c^nS_3) \Rightarrow (a^{n+3}Q_2, b^{n+1}S_2, c^{n+1}S_3)$
$\Rightarrow (a^{n+3}b^{n+1}S_2, S_2, c^{n+1}S_3) \Rightarrow (a^{n+3}b^{n+3}Q_3, bS_2, c^{n+2}S_3)$
$\Rightarrow (a^{n+3}b^{n+3}c^{n+2}S_3, bS_2, S_3) \Rightarrow (a^{n+3}b^{n+3}c^{n+3}S_3, bbS_2, cS_3)$

3.2.2 Generovaný jazyk

Vygenerovaný jazyk PC gramatickými systémy je následující:

$$L_f(\Gamma) = \{ w \in T^* : (S_1, S_2, ..., S_n) \Rightarrow^* (x, \alpha_2, ..., \alpha_n), \alpha_i \in (N \cup T \cup K)^*, \textbf{pro všechna } i = 2, ..., n \}$$

Příklad 3.4.

Máme gramatický systém Γ = ({S_1 , S'_1 , S_2 , S_3}, K , {a, b, c}, (S_1 , P_1), (S_2 , P_2), (S_3 , P_3)), kde:

$P_1 = \{S_1 \rightarrow abc, S_1 \rightarrow a^2b^2c^2, S_1 \rightarrow aS'_1, S_1 \rightarrow a^3Q_2, S'_1 \rightarrow aS'_1, S'_1 \rightarrow a^3Q_2, S_2 \rightarrow b^2Q_3, S_3 \rightarrow c\}$
$P_2 = \{ S_2 \rightarrow bS_2 \}$
$P_3 = \{ S_3 \rightarrow cS_3\}$

generovaný jazyk gramatického systému Γ je potom:

$$L_r(\Gamma) = L_{nr}(\Gamma) = \{a^n b^n c^n\} \quad kde\ n \geq 1$$

4 Syntaktická analýza[7]

Jednou z možností použití gramatik a gramatických systémů, zároveň tou nejčastější, je syntaktická analýza. Syntaktická analýza je proces, při kterém dochází k ověření, že vstupní řetězec patří do generovaného jazyka dané gramatiky nebo jazyka daného gramatického systému.

Před samotnou syntaktickou analýzou zpravidla bývá provedena lexikální analýza, která z jednotlivých znaků, pomocí jednoduchý pravidel, vytvoří ze vstupního řetězce seznam tokenů. Tento seznam tokenů je potom předán syntaktickému analyzátoru jako vstup, který na základě gramatických pravidel převede tento seznam tokenů na derivační strom. Derivační strom může být zároveň i výstupem syntaktického analyzátoru. Derivační strom je potom předán dále sémantickému analyzátoru, který s ním dále pracuje a vytvoří z něj abstraktní syntaktický strom. Abstraktní syntaktický strom je už připraven k vygenerování vnitřního kódu, optimalizaci a generaci výsledného kódu.

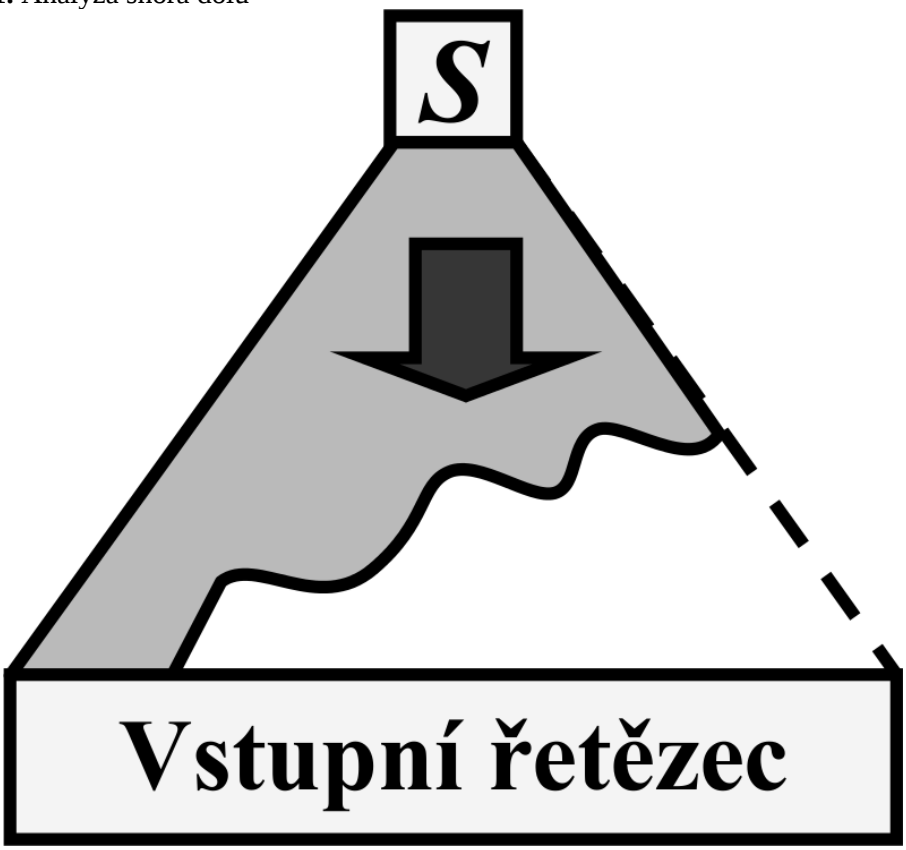
Existuje několik základních způsobů jak na zpracování syntaktické analýzy pohlížet, ale vždy je jádro celého problému syntaktické analýzy stejné. Vždy se syntaktický analyzátor snaží za pomocí gramatických pravidel nalézt vhodný derivační strom ze vstupního seznamu tokenů tak, aby po skončení procesu generace nezbyl na vstupním seznamu tokenů žádný token. Zároveň všechny z uzlů derivačního stromu musí splňovat předložené gramatické pravidla. Existují dva základní způsoby hledání vhodného derivačního stromu ze seznamu tokenů. Tyto dva způsoby jsou analýza shora dolů a analýza zdola nahoru.

4.1 Analýza shora dolů

Analýza shora dolů, jak již napovídá název samotný, je situovaná tak, že se začíná shora a postupuje se dolů. Na začátku je počáteční symbol, neterminál. Tento neterminál reprezentuje vrchol budoucího derivačního stromu. Model této syntaktické analýzy potom obsahuje expanzivní pravidla, která z počátečního symbolu a následně z dalších neterminálů expandují do nového derivačního stromu tak, aby konce jednotlivých větví budoucího derivačního stromu končily terminály. Levý rozbor těchto terminálů pak musí souhlasit se vstupním řetězcem a celý strom tedy potom znázorňuje, který jednotlivý terminál zapadá do jaké části výsledného derivačního stromu, přesně podle obrázku 4.1.

Pro vlastní analýzu shora dolů existují opět dva základní přístupy. Avšak pro oba dva přístupy je nutno nejprve sestavit *LL-Tabulku. P*ro sestavení *LL-Tabulky* je nutné nejprve stanovit několik základních množin. Tyto základní množiny jsou celkem čtyři a nazývají se *Empty*, *First*, *Follow a Predict.*

Obrázek 4.1. Analýza shora dolů



Empty(x) je množina znázorňující zda prvek x derivuje ε či nikoli. Tato množina nabývá pouze hodnot {ε} a Ø. Množina *Empty* nabývá hodnot Ø pro všechny terminály. Pro neterminál *A* nabývá množina *Empty* hodnoty {ε} v případě, že existuje pro daný neterminál *A* pravidlo tvaru $A \rightarrow \varepsilon$ nebo v případě, že existuje pro daný terminál *A* libovolné pravidlo tvaru $A \rightarrow X_1X_2...X_n$, kde pro všechna $i = 1, ..., n$ je $Empty(X_i) = \{\varepsilon\}$.

Algoritmus 4.1. *Empty(x)*

Vstup: gramatika $G = (N, T, P, S)$

Výstup: *Empty(X)* pro každé $X \in N \cup T$

(1) ***for-each*** *terminál* ***in*** T ***:***

(2) $Empty(terminál) = \emptyset$ ***;***

(3) ***for-each*** *neterminál* ***in*** N ***:***

(4) ***if*** *pravidlo* $A \rightarrow \varepsilon$ ***in*** P ***:***

(5) $Empty(A) = \{\varepsilon\}$ ***;***

(6) ***else :***

(7) $Empty(A) = \emptyset$ ***;***

(8) ***while*** *dokud bude možné měnit nějakou množinu Empty(x)* ***:***

(9) ***if*** *pravidlo* $A \rightarrow X_1X_2 ... X_n$ ***in*** P ***and*** $Empty(X_i) = \{\varepsilon\}$ *pro všechna* $i = 1, ..., n$ ***:***

(10) $Empty(A) = \{\varepsilon\}$ ***;***

First(x) je množina, která znázorňuje, jaký prvek se smí vyskytovat jako nejlevější v dané větné formě. Množina *First* nabývá hodnoty *{a}* pro každý libovolný terminál *a*. Pro neterminál *A* při libovolném pravidle $A \rightarrow X_1X_2...X_{k-1}X_k...X_n$ nabývá stejné hodnoty jako $First(X_k)$, když $Empty(X_i)$ pro všechna i $=1, ..., k-1$ je $\{\varepsilon\}$, kde $k \leq n$. Jinak pro pro daný neterminál *A* je hodnota stejná jako $First(X_i)$.

Algoritmus 4.2. First*(x)*

Vstup: gramatika *G = (N, T, P, S)*, množina *Empty(x)*

Výstup: *First(X)* pro každé $X \in N \cup T$

(1) ***for-each*** *terminál* ***in*** *T* ***:***

(2) *First****(****terminál****)*** *= {a}* ***;***

(3) ***for-each*** *neterminál* ***in*** *N* ***:***

(4) *First****(****neterminál****)*** *= Ø* ***;***

(5) ***while*** *dokud bude možné měnit nějakou množinu First(x)* ***:***

(6) ***if*** *pravidlo* $A \rightarrow X_1X_2 ... X_{k-1}X_k ... X_n$ ***in*** *P* ***:***

(7) *Přidej všechny symboly z* $First(X_1)$ *do First(A)* ***;***

(8) ***if*** $Empty(X_i) = \{\varepsilon\}$ *pro všechna* $i = 1, ..., k-1$, *kde* $k \leq n$ ***:***

(9) *Přidej všechny symboly z* $First(X_k)$ *do First(A)* ***;***

Follow(x) je množina všech neterminálů, která znázorňuje, jaký z nich se smí vyskytovat vpravo vedle něj ve větné formě. Pro startovní neterminál *S* nabývá množina *Follow* hodnoty koncového znaku *$*. Pro jiné neterminály nabývá množina *Follow(B)* při pravidle $A \rightarrow xBy$ symbolů z množiny *First(y)* tehdy, když $y \neq \varepsilon$. Když $Empty(y) = \{\varepsilon\}$ nabývá hodnoty symbolů z množiny *Follow(A)*.

Algoritmus 4.3. Follow*(A)*

Vstup: gramatika *G = (N, T, P, S)*, množiny *First(x)* a *Empty(x)*

Výstup: *Follow(A)* pro každé $A \in N$

(1) *Follow(S) = {$}* ***;***

(2) ***while*** *dokud bude možné měnit nějakou množinu Follow(A)* ***:***

(3) ***if*** *pravidlo* $A \rightarrow xBy$ ***in*** *P* ***:***

(4) ***if*** $y \neq \varepsilon$ ***:***

(5) *Přidej všechny symboly z First(x) do Follow(B)* ***;***

(6) ***if*** *Empty****(****y****)*** $= \{\varepsilon\}$ ***:***

(7) *Přidej všechny symboly z Follow(A) do Follow(B)* ***;***

Poslední zmíněná množina *Predict(A → x)* je množina všech terminálů, které mohou být aktuálně nejlevěji vygenerovány, pokud pro libovolnou větnou formu použijeme pravidlo $A \rightarrow x$. Množina *Predict(A → x)* je rovna *First(x)* ∪ *Follow(A)*, když množina *Empty(x) = {ε}*. V opačném případě, když množina *Empty(x) = Ø*, tak množina *Predict(A → x)* je rovna *First(x)*.

Algoritmus 4.4. *Predict(A → x)*

Vstup: gramatika *G = (N, T, P, S)*, množiny *First(x)*, *Follow(x)* a *Empty(x)*

Výstup: *Predict(A → x)* pro každé $A \rightarrow x \in P$

(1) ***for-each*** *pravidlo* $A \rightarrow x$ ***in*** *P* ***:***

(3) ***if*** *Empty**(x)** = {ε}* ***:***

(4) *Přidej všechny symboly z First(x) a Follow(A) do Predict(A → x)* ***;***

(5) ***else :***

(6) *Přidej všechny symboly z First(x) do Predict(A → x)* ***;***

Po sestavení jednotlivých množin můžeme za pomocí poslední zmíněné množiny *Predict* sestavit *LL-Tabulku*, u níž jednotlivé sloupce představují aktuální terminál ve vstupním řetězci. Řádky představují konkrétní poslední neterminál na zásobníku a jednotlivé buňky tabulky pak představují dané pravidlo, které je potřeba pro tuto konkrétní situaci terminálu ve vstupním řetězci a neterminálu na zásobníku použít.

Algoritmus 4.5. *Sestavení LL-Tabulky*

Vstup: gramatika *G = (N, T, P, S)*, *množina Predict(x)*

Výstup: *LL-Tabulka* pro všechna *N a T*

(1) ***for-each*** $A \rightarrow x$ ***in*** *P* ***:***

(3) ***if*** *A = řádek* ***and*** *sloupec* ***in*** *Predict(A)* ***:***

(4) *LL-Tabulka* ***[****řádek****]*** ***[****sloupec****]*** = $A \rightarrow x$

4.1.1 Syntaktická analýza založená na rekursivním sestupu

Prvním popsaným přístupem pro analýzu shora dolů vstupního řetězce ke konstrukci derivačního stromu je syntaktická analýza založená na rekursivním sestupu. Syntaktická analýza založená na rekursivním sestupu je jednoduchý princip založený na funkcionálním programování, kdy pro každý neterminál existuje právě jedna funkce, jejíž obsah reflektuje dané pravidla. Obsahem funkce je tedy vyhodnocení správnosti části vstupního řetězce a zavolání dalších podobných funkcí dle pravidel tohoto neterminálu. Tato další funkce vyhodnotí další části vstupního řetězce a výsledek vrátí do funkce ze které byly volány. Prvotní funkce startovního neterminálu pak vrátí výsledek korektnosti celého vstupního řetězce. *LL-Tabulka* je v tomto přístupu použita tedy pouze ke konstrukci daných funkcí.

Algoritmus 4.6. *Rekursivní sestup*

Vstup: gramatika $G = (N, T, P, S)$, *LL-Tabulka*

Výstup: ***true*** nebo ***false***

Pro neterminál E $\in N$ *a přidruženými pravidly* $E \rightarrow +TE$ *a* $E \rightarrow \varepsilon$ ***:***

```
(1)  function E :
(2)      E = false ;
(3)      if token = '+' :                    // simulace pravidla E → +ET
(4)          GetNextToken() ;
(5)          E = T() and E() ;
(6)      else-if token in [')', '$'] :       // simulace pravidla E → ε
(7)          E = true ;
```

4.1.2 Prediktivní syntaktická analýza

Dalším popsaným přístupem pro analýzu shora dolů vstupního řetězce ke konstrukci derivačního stromu je prediktivní syntaktická analýza. Prediktivní syntaktická analýza stejně jako syntaktická analýza založená na rekursivním sestupu využívá *LL-Tabulku*, avšak na rozdíl od předchozího přístupu je zde *LL-Tabulka* aktivně využívána po celou dobu analýzy. Kromě *LL-Tabulky* prediktivní syntaktická analýza vyžaduje také zásobník, který je při inicializaci naplněn koncovým a startovním symbolem. Pracuje se vždy s nejvrchnějším symbolem na zásobníku, který je po inicializaci startovní symbol. Tento startovní symbol je pomocí expanzních pravidel rozšířen na jiné další symboly, které jsou uloženy v opačném pořadí než v expanzním pravidle na zásobník místo vyjmutého startovního symbolu. V případě, že nejsvrchnějším symbolem na zásobníku je neterminál, tak se použije pravidlo dle *LL-Tabulky.* V případě, že se jedná o terminál a tento terminál se shoduje s aktuálním terminálem na vstupu, tak jsou oba terminály ze zásobníku i ze vstupního řetězce symbolů odstraněny a postupuje se dál do té doby, než na zásobníku a zároveň na vstupním řetězci nezbude žádný symbol ke zpracování. Jestliže zásobník skončí prázdný a vstupní řetězec ne nebo naopak vstupní řetězec už je prázdný a zásobník ještě ne, není vstupní řetězec zpracovatelný danou gramatikou.

Algoritmus 4.7. *Prediktivní syntaktická analýza*

Vstup: *LL-Tabulka* pro gramatiku $G = (N, T, P, S)$, $x \in T^*$

Výstup: *Levý rozbor pro x*, pokud $x \in L(G)$ jinak chyba

Nechť *X* je vrchol zásobníku a *a* aktuální token

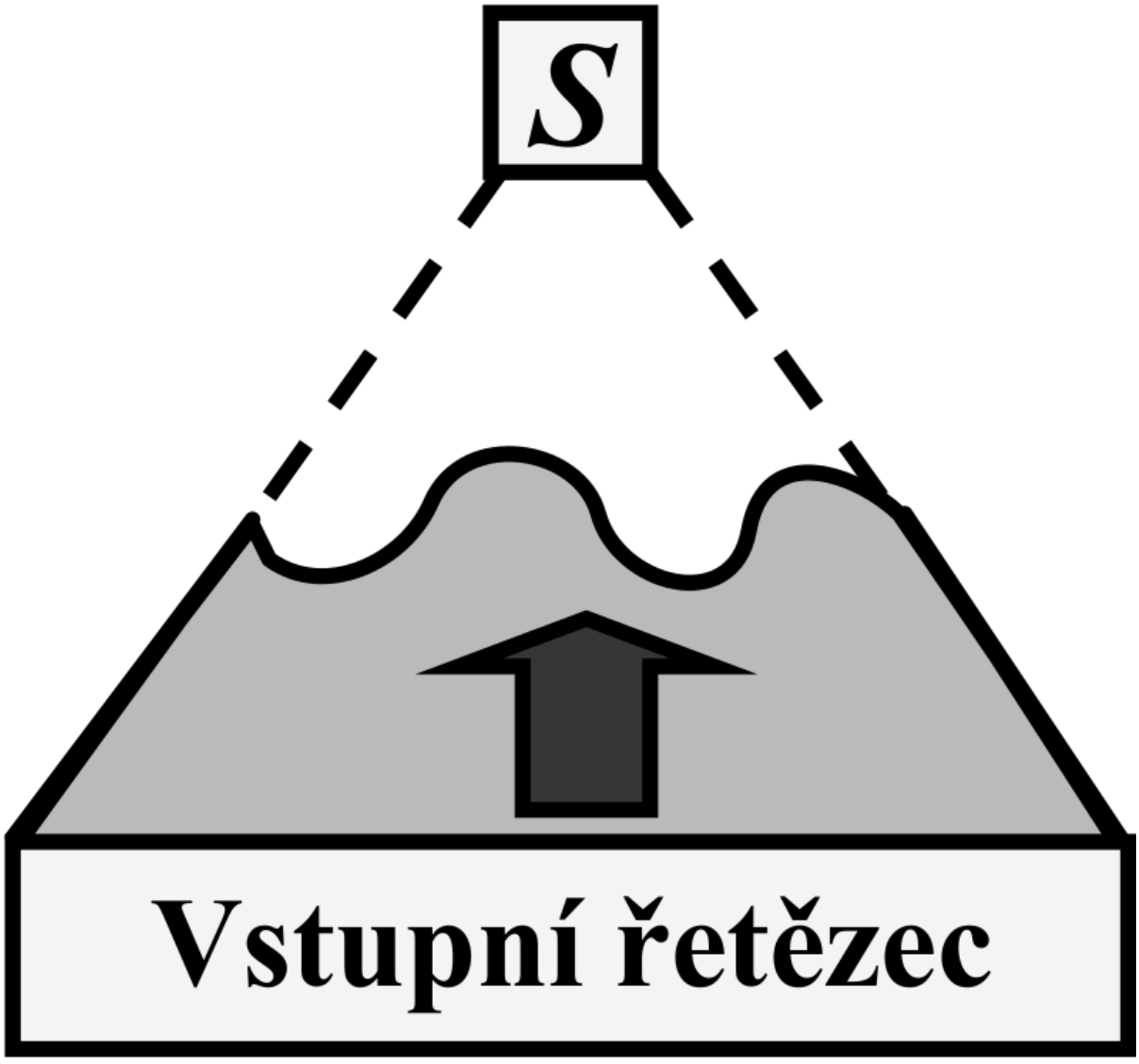
```
(1)  push($) a push(S) na zásobník ;
(2)  while dokud úspěch nebo chyba :
(3)      switch X :
(4)          case X = $ :
(5)              if a = $ : úspěch ;
(6)              else : chyba ;
(7)          case X ∈ T :
(8)              if X = a : pop(X) a přečti další token a ze vstupního řetězce ;
(9)              else : chyba ;
(10)         case X ∈ N :
(11)             if r: X → x in LL-Tabulka[X][a] :
(12)                 zaměň na vrcholu zásobníku X za reversal(x) a zapiš r na výstup
(13)             else : chyba ;
```

4.2 Analýza zdola nahoru

Analýza zdola nahoru postupuje přesně opačným směrem. Na počátku je symbol reprezentující konec vstupního seznamu tokenů. K němu se potom přidávají nové a nové symboly. Tyto nové symboly reprezentují jednotlivé tokeny ze vstupního řetězce. Model této syntaktické analýzy pak obsahuje redukční pravidla. Redukční pravidla redukují symboly jako jsou terminály, později i neterminály na abstraktnější celky. Nakonec posledním zbylým symbolem je neterminál reprezentující samotný vrchol budoucího nového derivačního stromu, startovní neterminál. Výsledný derivační strom, pokud nejsou gramatická pravidla ne-jednoznačná, je zpravidla stejný, jako ten vygenerovaný předešlým způsobem.

Pro vlastní analýzu zdola nahoru existují stejně jako u předešlého způsobu analýzy dva základní způsoby algoritmizace implementace. Tím lépe implementovatelným avšak nejméně silným způsobem je precedenční syntaktický analyzátor. Druhým způsobem pro syntaktickou analýzu zdola nahoru je LR syntaktický analyzátor.

Obrázek 4.2. Analýza zdola nahoru



4.2.1 Precedenční syntaktický analyzátor

Precedenční syntaktický analyzátor je jednoduchý syntaktický analyzátor pro analýzu zdola nahoru. Pracuje na základě precedenční tabulky. Pro tvorbu této precedenční tabulky je nutné, aby žádné ze zadaných gramatický pravidel neobsahovalo *ε-pravidlo* a aby žádná dvě pravidla v jedné gramatice neměla stejnou pravou stranu pravidla. Precedenční tabulka je tabulka určující prioritu jednoho terminálu před druhým nebo jejich shodnou prioritu. Základem pro tvorbu precedenční tabulky je priorita a asociativita. Vlastní analýza vstupního řetězce pak probíhá pomocí precedenční tabulky, která sama určuje v jakém případě se má použít redukční pravidlo a v jakém případě se má pouze načítat další znak vstupního řetězce.

Algoritmus 4.8. *Precedenční syntaktická analýza*

Vstup: Precedenční tabulka pro gramatiku $G = (N, T, P, S)$, $x \in T^*$

Výst*up:* *Pravý rozbor pro x*, pokud $x \in L(G)$ jinak chyba

Nechť funkce *top* vrací terminál na zásobníku nejblíže vrcholu

```
(1)  push($) na zásobník ;
(2)  while dokud a = $ and top = $ :
(3)      a = top ;
(4)      b = aktuální znak na vstupu ;
(5)      switch Tabulka [a] [b] :
(6)          case '=' : push(b) and přečti další symbol b ze vstupu ;
(7)          case '<' : zaměn a za a< na zásobníku and
(8)                     push(b) and přečti další symbol b ze vstupu ;
(9)          case '>' : if <y je na vrcholu zásobníku and r: A → y ∈ P :
(10)                        zaměň <y za A and vypiš r na výstup ;
(11)                    else : chyba ;
(12)         case ' ' : chyba ;
```

4.2.2 LR syntaktický analyzátor

LR syntaktický analyzátor je druhou a tou těžší možností pro analýzu zdola nahoru. LR syntaktický analyzátor využívá *LR-Tabulky*, která se skládá ze dvou částí. První částí *LR-Tabulky* je část akční, která je určena pouze pro tokeny. Akční část *LR-Tabulky* udává která akce se má uskutečnit po načtení zadaného symbolu ze vstupního řetězce a při aktuálním stavu. Možnými uskutečnitelnými akcemi jsou užití redukčního pravidla, nebo pouhé posunutí na další znak vstupního řetězce a změna aktuálního stavu. Druhou částí *LR-Tabulky* je část přechodová, která je určena pouze pro neterminály. Tato část tabulky obsahuje pouze akci změna aktuálního stavu a posun na další symbol vstupního řetězce. Přechodová část *LR-Tabulky* je využívána vždy po uplatnění pravidla z akční části tabulky.

Algoritmus 4.9. *LR syntaktický analyzátor*

Vstup: LR-Tabulka pro gramatiku $G = (N, T, P, S)$, $x \in T^*$

Výstup: *Pravý rozbor pro x,* pokud $x \in L(G)$ jinak chyba

(1) ***Push(***<$, q_0>***)*** *na zásobník* ***;*** *stav =* q_0***;***

(2) ***while*** *dokud úspěch nebo chyba* ***:***

(3) *a = aktuální znak na vstupu* ***;***

(4) ***switch*** *α* ***[****stav****] [****a****] :***

(5) ***case*** *sq* ***: push(****a, q****) and*** *přečti další symbol a ze vstupu* ***and*** *stav = q****;***

(6) ***case*** *rp* ***: if*** *p:* $A \rightarrow X_1X_2 \ldots X_n \in P$ ***and***

(7) <?, q><X_1, ?><X_2, ?>...<X_n, ?> *je na vrcholu zásobníku* ***:***

(8) *stav =* β ***[****q****] [****A****] and***

(9) *zaměň* <X_1, ?><X_2, ?>...<X_n, ?> za <A, stav> **and**

(10) ***z****apiš p na výstup* ***;***

(11) ***else :*** *chyba* ***;***

(12) ***case*** *' ' :* *chyba* ***;***

Pro konstrukci *LR-Tabulky* existuje několik algoritmů, které se liší jednoduchostí implementace i vyjadřovací sílou. Základními třemi algoritmy pro konstrukci *LR-Tabulky* jsou:

- Simple LR (SLR): nejslabší, ale jednoduchý a vytvoří málo stavů
- Canonical LR: více silný, ale vytvoří poměrně hodně stavů
- Lookahead LR (LALR): nejlepší, protože je nejsilnější a vytvoří stejný počet stavů jako SLR

5 Implementace

V této kapitole se věnuji návrhu a výsledné implementaci konečné aplikace. Dle zadání bylo potřeba, aby konečná aplikace byla schopná, prostřednictvím zadané gramatiky, vstupní soubor se vstupním řetězcem rozdělit pomocí lexikální analýzy na jednotlivé tokeny. Dále pak především prostřednictvím syntaktické analýzy najít pro tyto tokeny vhodný derivační strom a ověřit zda zadaný vstupní řetězec je zpracovatelný zadaným gramatickým systémem.

5.1 Úvod

Za účelem vstupu pro syntaktickou analýzu byla sestrojena univerzální lexikální analýza založená na regulárních výrazech zpracovávaných regulárními automaty obsaženými přímo v jazyku C samotném. Protože bakalářská práce byla psaná v jazyce C++, ale regulární automaty v jazyce C++ obsaženém nemají možnost zpracovávat pokročilé *ECMAScript syntax* byla pro jednoduší práci s regulárními výrazy vytvořena třída zastřešující C knihovnu pro použití v jazyku C++. Nastavení lexikální analýzy se prování prostřednictvím konfiguračního souboru ve formátu XML, jehož název se specifikuje povinným parametrem *--lexer-config [file]* nebo popřípadě jeho zkrácenou verzí *-l [file]*. Přípustný formát konfiguračního XML souboru je popsán v kapitole 5.2.1.

Jádrem byla samotná syntaktická analýza, u které byl ze zadání vyplývající požadavek, aby bylo možné touto syntaktickou analýzou zpracovávat bezkontextové jazyky a některé další ne bezkontextové jazyky. Syntaktická analýza dále byla rozdělena do různých možností zpracování, přičemž jednotlivá nastavení jsou dostupná pod přepínači aplikace *--up-down, --down-up,* popřípadě jejich zkrácenými verzemi *-u* a *-d.* Samostatná definice gramatik je potom obsažena v konfiguračním XML souboru, který je zadán jako parametr aplikace *--parser-config [file]* nebo pod jeho zkrácenou verzí *-p [file]*.

Nakonec je potřeba specifikovat vstupní soubor s daty, který bude procházet přes lexikální a posléze i syntaktickou analýzu. Tento soubor obsahuje vstupní řetězec pro zpracování syntaktickou analýzou. Jeho umístění lze zadat prostřednictvím parametru aplikace pod názvem *--file [file]* nebo jeho zkrácenou variantou *-f [file].*

5.2 Rozvržení aplikace

Jak již bylo zmíněno dříve, aplikace je psána v C++ a proto bylo možné jednotlivé logické celky oddělit od sebe a celou aplikaci tak psát jako objektovou. Jednotlivé objekty výsledné aplikace bych rozdělil ještě do dvou větších skupin. V první skupině se jedná o objekty třídy převážně pro usnadnění práce s programováním aplikace a pro podpůrnou činnost. To jsou především načítání konfiguračních XML souborů, zpracovávání vstupních parametrů nebo zpracovávání pokročilých regulárních *ECMAScript syntax* výrazů. V druhé skupině se jedná o třídy samotného syntaktického a lexikálního analyzátoru. Tyto třídy především zpracovávají data, dle zvolených parametrů výsledné aplikace a konfigurace jednotlivých modulů v konfiguračních souborech. Modul lexikální analýzy zpracovává vstupní řetězec, zadaný též v souboru, na seznam tokenů, který je předán syntaktickému analyzátoru. Tento syntaktický analyzátor posléze vytvoří ze seznamu tokenů konečný derivační strom a ověří zda je vstupní řetězec možno zpracovat daným gramatickým systémem.

5.2.1 Třída GetOpt

Tato třída slouží především jako abstraktní C++ vrstva pro knihovnu jazyka C *getopt.h* pro snadné používání v jazyku C++. Dále byla základní C knihovna rozšířena o podporu některých kombinací povinných a nepovinných parametrů aplikace, jejíž podpora v základní C knihovně chybí a její používání proto není ve výsledné aplikaci dostatečně přátelské pro použití uživateli. Výsledkem této třídy je seznam zadaných vstupních parametrů indexovaný pomocí jejich jednoznakové varianty.

5.2.2 Třída RegEx

Tato třída byla navržena především kvůli nekompatibilitě staré C knihovny *regex.h* a nové C++ knihovny <*regex*>. Nová C++ knihovna <*regex*>*bohužel* nepodporuje zpracovávání pokročilých regulérních výrazů, především *ECMAScript syntax*. Proto byla navržena abstraktní vrstva zastřešující C knihovnu *regex.h*, která už implementaci *ECMAScript syntax* podporuje. Pro co nejlepší dopřednou kompatibilitu byla výsledná třída RegEx navržena tak, aby její metody byly co nejpodobnější nové C++ knihovně <*regex*>. Díky této třídě je pak možné zpracovávat pokročilé *ECMAScript syntax* regulární výrazy a usnadní se tím práce jak při načítání XML souborů, tak i při samotné syntaktické analýze.

5.2.3 Třída Xml

Třída Xml slouží především pro validaci správné syntaxe Xml souboru a jeho následného zpracování do objektové podoby ve vnitřní paměti počítače. Tato třída je založená na rekursi sama sebe, kdy třída sama je element jazyka Xml, přičemž obsahuje jeho název a případné odkazy na pod-elementy nebo obsah elementu jako text. Tato třída využívá především regulárních výrazů zpracovávaných pomocí předešlé třídy RegEx. Výsledný objektový strom je pak předán dalším třídám programu pro další zpracovávání.

5.2.4 Třída Lexer

Třída Lexer slouží jako univerzální lexikální analyzátor. Je konfigurovatelný pomocí konfiguračního souboru ve formátu XML. Jeho běh závisí především na definici regulérních výrazů, pomocí kterých tento konfigurovatelný lexikální analyzátor rozdělí vstupní řetězec na jednotlivé tokeny. Tyto tokeny jsou potom v podobě seznamu tokenů uloženy v paměti a předávány postupně na vyžádání syntaktickému analyzátoru.

5.2.5 Třída Parser

Tato třída je syntaktický analyzátor samotný. Je konfigurovatelný prostřednictvím konfiguračního souboru ve formátu XML zadaného jako parametr aplikace. Tento syntaktický analyzátor podporuje několik režimů mezi kterými se dá přepínat pomocí přepínačů zmíněnými již v kapitole 5.1. Syntaktický analyzátor samotný pracuje s celým gramatickým systémem. Vstup pro tento syntaktický analyzátor mu dodává lexikální analyzátor uvedený v předchozí podkapitole 5.2.4. Výstupem tohoto syntaktického analyzátoru je potom vypsán na standardní výstup.

5.3 Lexikální analýza

Lexikální analýza je v této aplikaci navržena tak, aby byla co nejsnáze implementovatelná a zároveň co nejúčinnější. A aby měl seznam tokenů předávaný syntaktickému analyzátoru možnost velkého počtu různých tokenů. V konstruktoru třídy je načtena konfigurace a v užitné podobě uložena jako argument třídy. Jednotlivé tokeny jsou definovány pomocí regulárního výrazu. Samotnou lexikální analýzu potom provádí jediná metoda, která v cyklu prochází regulární výrazy a zkouší, jestli některému z nich neodpovídá vstupní řetězec tak, aby po odstranění části vstupního řetězce, které odpovídá regulární výraz, nezbyl před touto odstraněnou částí vstupního řetězce žádný symbol. Po odstranění části vstupního řetězce, která odpovídá regulárnímu výrazu, je tento odstraněný řetězec, spolu s pojmenováním regulárního výrazu, prohlášen jako lexém a po určení jeho typu, na základě názvu regulárního výrazu, uložen na konec seznamu tokenů načtených ze vstupního řetězce jako token. Samotný syntaktický analyzátor pak použitím vestavěné metody načítá jeden lexém po druhém a sestavuje z nich derivační strom.

5.4 Syntaktická analýza

Syntaktická analýza je v této aplikaci založena na gramatických systémech. Gramatický systém je vícero gramatik na jednom místě. Tento syntaktický analyzátor zpracovává vícero různých gramatik najednou, přičemž mezi nimi přepíná. Pro tuto bakalářkou práci jsem zvolil pro aplikaci implementaci algoritmů shora dolů i zdola nahoru.

V případě algoritmu shora dolů se jedná o rekursivní sestup, kde místo velkého množství statických funkcí se jedná o jednu univerzální funkci, která volá vždy rekursivně sama sebe a podle předaných parametrů se přizpůsobuje roli dané teoretické statické funkci. Pro správný běh této metody je nutné v konfiguračním souboru stanovit pouze sadu gramatických pravidel. Syntaktický analyzátor sám si z nich vytvoří *LL-Tabulku*, kterou použije pro syntaktickou analýzu samotnou.

Pro druhý algoritmus, který je orientován zdola nahoru byla zvolena implantace algoritmu precedenční analýzy. Precedenční analýza je založená na precedenční tabulce, kterou využívá po celou dobu běhu analýzy. Z tohoto důvodu je potřebné stanovit v XML konfiguračním souboru pro danou gramatiku precedenční tabulku pro precedenční analýzu. Bez precedenční tabulky by nešlo uskutečnit vlastní syntaktickou analýzu zdola nahoru založenou na precedenční analýze.

5.5 Formát konfiguračních souborů

Jako jednotný jazyk všech konfiguračních souborů pro tuto aplikaci byl zvolen jazyk XML pro jeho velikou flexibilitu a potenciální snadnou a velikou rozšířitelnost. Vnitřní struktura konfiguračních XML souborů byla zvolena tak, aby bylo možné veškerou konfiguraci zadat do jednoho jediného souboru, nebo aby bylo možné konfiguraci rozdělit podle jednotlivých modelů do více souborů a konfigurovat tedy potom každý modul zvláštním souborem. Pro konfigurační soubory se používá podmnožina jazyka XML. Nadefinujme si tedy jeho základní pojmy.

Element jazyka XML je jedno slovo ohraničené z obou stran šipkami < a > nazývané název elementu, přičemž existuje několik druhů elementů. Počáteční element je takový, který má pouze otevírací značku a po skončení jeho těla samostatného musí následovat značka konečná. Konečný element je pravý opak elementu počátečního, má pouze konečnou značku a před jeho tělem musí následovat značka počáteční. Konečná značka se vyznačuje tím, že její název začíná znakem lomítka a že neobsahuje žádné argumenty. Mezi značkou počáteční a značkou konečnou se nachází vlastní tělo elementu. Tělo elementu se může skládat z dalších elementů nebo z čistého textu, kombinaci obou možností není v tomto případě možná. V případě, že tělo elementu chybí úplně je možnost značku počáteční a konečnou zapsat jako jednu značku s charakteristickým lomítkem na konci. Jedná se potom o prázdný element. V případě počátečního a tedy i prázdného elementu je dále možnost doplnit tyto značky o atributy elementu. Atributy elementu jsou napsány za jeho názvem a jsou odděleny mezerou. Jednotlivé atributy se pak zapisují jako název atributu, rovná se, a jeho obsah uzavřený do uvozovek nebo apostrofů.

V rámci jednoho elementu není možné tyto atributy stejného jména opakovat. Dále není možné, pro jednoznačnost identifikace, aby název atributu elementu obsahoval mezeru. Název elementu nebo text elementu nesmí obsahovat šipky ohraničující element a obsah atributu elementu nesmí obsahovat uvozovky nebo apostrofy, podle toho kterými byl obsah ohraničen.

5.5.1 Lexikální analýza

Konfigurace lexikálního analyzátoru je zapsána do formátu XML v souboru. Tento soubor musí obsahovat element <*lexer*>. V elementu <*lexer*> pak jsou jednotlivé definice lexémů uzavřeny do elementu <*lexem*>. Definice lexému, element <*lexem*> obsahuje parametry. Tyto parametry jsou elementy <*name*> pro pojmenování daného lexému a používání v konfiguraci pro syntaktický analyzátor. Dále potom element <*regexp*> pro definování regulárního výrazu, kterému musí daný lexém odpovídat. V současné době jsou zkoušeny regulární výrazy lexému postupně od prvního tak, jak jsou zapsány v XML konfiguračním souboru, ale pro vylepšení aplikace by bylo vhodné umožnit u každého lexému zapsat prioritu s jakou má být tento lexém zpracováván. Jména lexémů jsou pak dále používány pro zápis konfigurace pro vlastní syntaktický analyzátor, kde reprezentují tokeny načtené od lexikálního analyzátoru.

5.5.2 Syntaktická analýza

Konfigurace syntaktického analyzátoru stejně jako u toho lexikálního je zapsána ve formátu XML v souboru. Podobně jako lexikální analyzátor i konfigurace celého syntaktického analyzátoru musí být zabalena do jednoho jediného elementu, tímto elementem je <*parser*>. V tomto elementu jsou pak jednotlivé gramatiky <*gramar*> ve kterých jsou obsaženy jednotlivé gramatické pravidla a informace pro konstrukci precedenční tabulky. Jednotlivá pravidla jsou uzavřeny v elementu <*rule*>, který obsahuje elementy <*in*> a <*out*> jako pravou a levou stranu pravidla. Konfigurace precedenční tabulky je uzavřena v elementu <*prec*>.

5.6 Zpracování chyb

K chybám může docházet ve všech částech programu. Vlivem chybně napsaných konfiguračních xml souborů pro lexikální nebo syntaktickou analýzu. Lexikálním nebo syntaktickou chybou obsaženou přímo ve zpracovávaném vstupním souboru. V samotné implementaci algoritmů konečné aplikace.

Při nalezení chyby v chybně napsaném konfiguračním xml souboru je tato chyba ihned po identifikování nahlášena uživateli prostřednictvím oznámení vypsaného na standardní chybový výstup. Hledání chyb v chybně napsaném xml konfiguračním souboru je zaměřeno pouze na chyby týkající se samotného standartu konfiguračního XML. Oznámení samotné pak je zaměřeno na konkrétní chybný element a je uživateli prostřednictvím tohoto oznámení naznačená správná forma zápisu dané části konfiguračního xml souboru. V případě, že v některém ze zadaných konfiguračních xml souborů byla nalezena chyba nedovolující samotné spuštění lexikální analýzy a posléze i syntaktické, je aplikace po oznámení chyb uživateli ukončena s chybovým kódem.

Chyby obsažené přímo v zadaném vstupním souboru, ať už při lexikální nebo syntaktické analýze, jsou hlášeny také co nejdříve prostřednictvím oznámení daného modulu výsledné aplikace. Tyto chyby jsou považovány za kritické a proto po oznámení těchto chyb uživateli je celý modul včetně aplikace samotné ukončen a je navrácen návratový chybový kód.

6 Závěr

Bezkontextové jazyky a gramatiky mají nezastupitelné místo v teoretické informatice. Tato bakalářská práce se zabývala gramatickými systémy, které staví na základech gramatik samotných. V této souvislosti byly představeny dva základní přístupy pro gramatické systémy. Sekvenční a paralelní.

První z těchto přístupů zastupují kooperativní distribuované gramatické systémy, zkráceně CD gramatické systémy. Tato modifikace gramatických systémů je založena na sekvenčním zpracovávání vstupního řetězce. Jednotlivé gramatiky jsou tedy zpracovávány postupně vždy ta, která je pro danou část vstupního řetězce určena. V jednu chvíli je tedy možné zpracovávat pouze jednu gramatiku.

Dalším představeným přístupem jsou potom paralelní kooperativní gramatické systémy, zkráceně PC gramatické systémy. Tento další přístup využívá oddělenosti jednotlivých gramatik od sebe a zavádí možnost zpracovávání vícero gramatik současně. Pro účely souběžného zpracovávání více gramatik jsou také nově zavedeny další symboly, které zajišťují komunikaci mezi jednotlivými vlákny zpracovávajícími jednotlivé gramatiky.

Jádrem samotné práce pak bylo vytvoření syntaktického analyzátoru, který tyto gramatické systémy zpracovává. Syntaktická analýza je proces ověření validnosti vstupního řetězce vzhledem k gramatikám a jejím zadaným pravidlům. Syntaktický analyzátor je potom nástroj, který tuto syntaktickou analýzu provádí. Tento syntaktický analyzátor bývá často součástí překladače jednoho jazyka do jiného. Před vlastním syntaktickým analyzátorem se, zejména v překladačích, vyskytuje lexikální analyzátor, který vstupní řetězec zpracuje na seznam tokenů a sjednotí tím jednotlivé lexikální prvky na menší počet. Ty se nazývají tokeny a jsou děleny především podle typu.

Vlastní syntaktická analýza se pak dále dělí na analýzu shora dolů a analýzu zdola nahoru. Analýza shora dolů začíná se startovním symbolem, neterminálem. Startovní symbol je posléze za pomocí expanzivních gramatických pravidel nahrazen za pravou stranu daného expanzního gramatického pravidla, které obsahuje další neterminály a terminály. Tyto další neterminály jsou nahrazovány vhodnými expanzními pravidly stejně jako tomu bylo u startovního neterminálu. Při postupném nahrazování všech neterminálů pravou stranou expanzního gramatického pravidla vzniká derivační strom. Jednotlivé uzly derivačního stromu potom znázorňují použití vhodného expanzního pravidla na neterminál a jeho nahrazení pravou stranou pravidla. Všechny neterminály jsou nahrazovány expanzními pravidly do té doby, než na konci jednotlivých větví derivačního stromu zbudou pouze terminály. Pro tuto metodu existují dvě základní implementace. Metoda rekurzivního sestupu, kdy je pro každý neterminál vytvořena funkce. Definice této funkce potom obsahuje simulaci daného gramatického pravidla. Nevýhodou tohoto způsobu je především nutnost vytvoření funkce pro každý neterminál. Další nevýhodou je možné přetečení zásobníku při zanořování do rekurse u velice velikých a složitých gramatik. Výhodou je potom velice lehký princip implementace. Druhou metodou je prediktivní syntaktická analýza. Tato metoda je zcela řízena LL-Tabulkou a obsahuje zásobník.

Analýza zdola nahoru postupuje opačným směrem. Začíná od jednotlivých terminálů, které pomocí vhodných redukčních pravidel sloučí na jeden neterminál. Neterminály jsou pak stejným postupem slučovány pomocí vhodných gramatických redukčních pravidel do dalších neterminálů. Při tomto slučování jednotlivých terminálů a neterminálů na neterminál dochází též ke vzniku derivačního stromu, kde jednotlivé uzly stromu představují sloučení terminálů a neterminálů na neterminál. Toto slučování postupuje do té doby, než jsou sloučeny všechny terminály ze vstupního řetězce a neterminály, které vznikly při slučování, na jeden jediný neterminál. Tento poslední zbylý neterminál je zároveň startovním neterminálem.

Pro implementaci v této bakalářské práci jsem se rozhodl implementovat CD gramatické systémy. Každý gramatický systém se musí také skládat z gramatických systémů. Tyto jednotlivé gramatiky obsažené v gramatickém systému je možno zpracovávat různými způsoby. Pro implementaci jsem se rozhodl zvolit jak syntaktickou analýzu shora dolů, tak i syntaktickou analýzu zdola nahoru.

Pro syntaktickou analýzu shora dolů jsem implementoval rekursivní sestup. Tato implementace byla upravena, aby nebylo potřeba psát pro každý neterminál zvláštní funkci. Tato implementace je tedy universální pro jakýkoli neterminál a jakékoli pravidlo, přičemž jednotlivé odlišující znaky jednotlivých funkcí z rekursivní analýzy jsou dodány dynamicky na základě toho, pro který neterminál se tato universální funkce volá. Pro implementaci rekursivního sestupu je potřeba sestavit LL-Tabulku. LL-Tabulka je složena automaticky na základě zadaných pravidel.

Pro syntaktickou analýzu zdola nahoru jsem zvolil implementaci precedenční analýzy. Precedenční analýza potřebuje pro svoji činnost precedenční tabulku. Precedenční tabulka není sestavována automaticky a je ji nutné specifikovat v konfiguračním souboru pro syntaktický analyzátor. Specifikace precedenční tabulky se uvádí pro konkrétní gramatiku, pro kterou má být možnost použití precedenční analýzy.

Jelikož v gramatických systémech je použito více gramatik, které mohou být na sobě navzájem závislé, je nutné závislosti mezi těmito gramatikami specifikovat. Tato závislost se uvádí jako speciální symbol, který určuje kdy se má přestat provádět analýza a předat řízení jinému analyzátoru. Tento druhý analyzátor až práci dokončí tak poskytne výsledek prvnímu analyzátoru a ten pokračuje v práci tam kde skončil.

Výsledkem celé syntaktické analýzy je potom hodnota, která říká zda vstupní řetězec je zpracovatelný tímto gramatickým systémem nebo ne. Tuto bakalářkou práci je dále možnost rozšiřovat. Tato rozšíření spočívají v několika možnostech. Jednou z možností je přidání dalších možných způsobů syntaktické analýzy jednotlivých gramatik. Přidání ať už analýzy shora dolů nebo analýzy zdola nahoru spočívá v napsání nové třídy, která bude implementovat tento nový způsob analýzy. Tato nová třída se dále začlení do samotného syntaktického analyzátoru, aby ji bylo možné používat. Další z možností je rozšířit možnosti stávající analýzy. Pro účel možného rozšiřování stávající syntaktické analýzy byl zvolen formát konfiguračního souboru XML. Tento formát dovoluje téměř neomezené možnosti rozšiřování konfiguračních možností.

Budoucím možným rozšířením je také implementace PC gramatických systémů. PC gramatické systémy se vyznačují především paralelním zpracováním jednotlivých gramatik. To znamená, že jednotlivé gramatiky jsou zpracovávány souběžně a mezi jednotlivými procesy syntaktické analýzy probíhá komunikace, která má za účel synchronizaci celkové paralelní syntaktické analýzy.

V oblasti teoretické informatiky pak další možný vývoj představuje především upřesnění síly navrhovaného řešení. Upřesnění síly navrhovaného řešení je možné zpracovat díky porovnání s jinými známými systémy nebo algoritmy.

Literatura

[1] Meduna, A.: *Automata and Languages*: Theory and Applications. Springer, 2005, ISBN 81-8128-333-3.

[2] Meduna, A.: Deep Pushdown Automata. *Acta Informatica*, ročník 2006, č. 98, 2006: s. 114–124, ISSN 0001-5903.

[3] Meduna, A.; Zemek, P.: *Regulated Grammars and Their Transformations*. Brno University of Technology, 2010, ISBN 978-80-214-4203-0, 239 s.

[4] Chomsky, N.: Three models for the description of language. *Information Theory, IRE Transactions on*, ročník 2, č. 3, September 1956: s. 113–124, ISSN 0096-1000, doi:10.1109/TIT.1956.1056813.

[5] Chomsky, N.: On Certain Formal Properties of Grammars. *Information and Control*, ročník 2, č. 2, 1959: s. 137–167.

[6] Rozenberg, G.; Salomaa, A. (editoři): Handbook of Formal Languages, Vol. 1-3. Springer, 1997, ISBN 3-540-60649-1.

[7] Aho, A. V.; aj.: Compilers: Principles, Techniques, and Tools (2nd Edition). Pearson Education, druhé vydání, 2006, ISBN 0-321-48681-1.

[8] Csuhaj-Varju, E. a. k.: Grammar systems: a grammatical approach to distribution and cooperation. OPA (Amsterdam) B.V., 1994.

[9] Csuhaj-Varju, E. a. k.: Grammar Systems. A gramatical approach to distribution and cooperation. Gordon and Breach, London, 2004.

[10] Linz, P.: An Introduction to Formal Language and Automata (3rd Edition). USA: Jones and Bartlett Publishers, Inc., třetí vydání, 2000, ISBN 0-7637-1422-4.

Seznam příloh

Příloha A: Obsah CD

Přiložené CD obsahuje:

- text písemné zprávy ve formátu PDF
- text písemné zprávy ve zdrojovém formátu ODT
- zdrojové soubory syntaktického analyzátoru
- spustitelný soubor syntaktického analyzátoru
- vzorové konfigurační nastavení
- automatickou dokumentaci syntaktického analyzátoru ve formátu HTML