VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

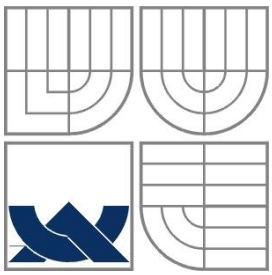
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

Obecný generátor syntaxí řízeného překladu

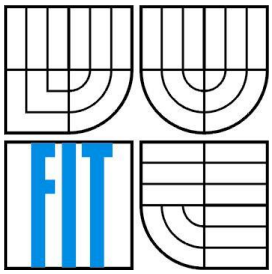
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Juraj Marcin
AUTHOR

BRNO 2016



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OBECNÝ GENERÁTOR SYNTAXÍ ŘÍZENÉHO PŘEKLADU

A GENERAL GENERATOR OF SYNTAX DIRECTED TRANSLATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Juraj Marcin

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Meduna Alexander, RNDr., CSc.

BRNO 2016

Abstrakt

Tato práce se zabývá vytvořením obecného generátoru. Hlavním předmětem tohohle generátoru je vykonávání syntaxi řízeného překladu, který je vykonávaný syntaktickým analyzátorem a to už v době analýzy zdrojového programu. Celý syntaktický analyzátor je založen na syntaktické analýze zdola nahoru. Výsledkem překladu je vyprodukování syntaktického stromu, z kterého se vytvoří instrukce ve tvaru tří-adresního kódu.

Abstract

This thesis deals with the creation of a general generator. The main subject of this one generator is performing syntax directed translation that is performed by the parser. This translation is processing in the analysis of the source program. The whole parser is based on a bottom-up parsing.
The result of the translation is syntax tree form. From this syntax tree are created instructions in the form of three - address code.

Klíčová slova

syntaktická analýza, syntaktický strom, tabulka symbolů, syntaxi řízený překlad, tři adresný kód

Keywords

syntax analysis, syntax tree, symbol table, syntax-direct translation, three-address code

Citace

Marcin Juraj: Obecný generátor syntaxí řízeného překladu, bakalářská práce, Brno, FIT VUT v Brně, 2016

Obecný generátor syntaxí řízeného překladu

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.

……
Juraj Marcin
Datum. 4. 5. 2016

Poděkování

Chcel by som poďakovať prof. RNDr. Alexandru Medunovi, CSc., ktorý túto bakalársku prácu viedol, smeroval a tiež za jeho ochotu a cenné rady, ktoré mi dal.

Obsah

Úvod

Ľudia sa už od pradávna medzi sebou dorozumievajú pomocou rôznych jazykov. Prirodzené jazyky majú veľmi silnú dorozumievaciu schopnosť, avšak pre počítačové spracovanie sú zvyčajne veľmi náročné a komplikované. Medzi hlavné dôvody patrí najmä nejednoznačnosť, keďže jedno slovo môže mať viacero významov. Preto boli pre potreby jednoznačnej komunikácie s počítačmi v informatike zavedené formálne jazyky.

Formálnymi jazykmi označujeme tie jazyky, ktoré sme schopný jednoznačne matematicky popísať a definovať. Tieto jazyky slúžia k zjednodušeniu prirodzených jazykov a sú vhodné na dorozumievanie sa so strojmi. Umožňujú jednoducho určiť presný spôsob a tvar komunikácie. Často sa používajú pre popis programovacích jazykov, dátových formátov alebo prenosových protokolov a spôsobu ich spracovania. Pomocou nich sa definujú aj celé rady dátových štruktúr. Rovnako existuje aj problém ako tieto formálne jazyky popísať. Často ale nie je vhodné, aby sa dal formálny jazyk vyjadriť iba ako konečná množina viet, ktoré do neho patria, ale aby existoval neobmedzený počet viet prislúchajúci danému jazyku. Tento problém odstraňuje jedna z podmnožín formálnych jazykov a tou sú *bez-kontextové gramatiky*.

Bez-kontextové gramatiky umožňujú konečným množstvom pravidiel presne popísať, ako má vyzerať veta, ktorá patrí do tohto jazyka. Negatívom gramatík je to, že sú väčšinou schopné popísať iba tie jazyky, ktorých vyjadrovacia schopnosť je oproti prirodzeným jazykom obmedzená. Ďalšou z podmnožín formálnych jazykov sú *bez-kontextové formálne jazyky*. Oblasť bez-kontextových jazykov bola dôkladne skúmaná z dôvodu využitia v rôznych odvetviach informatiky. Výhodami týchto bez-kontextových formálnych jazykov je ich jednoduchá implementácia. proti prirodzeným jazykom však podobne ako gramatiky neposkytujú vždy dostatočnú vyjadrovaciu silu. Pre získanie plnej vyjadrovacej schopnosti boli v informatike zavedené iné spôsoby spracovania akými sú napríklad gramatické systémy.

Gramatické systémy boli objavené za účelom dosiahnutia lepšieho komplexného výsledku v porovnaní s jednoduchými gramatikami. Ich základná myšlienka je v zložení z jednotlivých gramatík, kde má každá gramatika vlastné pravidlá a vyjadrovaciu silu. Tieto gramatiky medzi sebou zdieľajú základnú množinu terminálov a neterminálov. Aj napriek tomu, že môžu byť jednotlivé gramatiky v gramatickom systéme jednoduché, po ich zlúčení do spoločného celku majú oveľa väčšiu vyjadrovaciu silu, ktorá presahuje vyjadrovaciu schopnosť samotných komponent. Rovnako ako bez-kontextové gramatiky tak aj celé gramatické systémy sa používajú pri procese prekladu, obzvlášť v časti syntaktickej analýzy.

Syntaktická analýza je jednou z niekoľkých častí procesu prekladu vstupného reťazca na výstupný. Jej hlavnou úlohou je zistiť, či zadaný vstupný reťazec zodpovedá zadanému formálnemu jazyku. Táto analýza zvyčajne nasleduje za lexikálnou analýzou, ktorá vytvorí zo vstupného reťazca lexémy a následne tokeny. Tieto tokeny sú postupne predávané syntaktickému analyzátoru podľa jeho potrieb. Syntaktický analyzátor následne z týchto tokenov za pomoci pravidiel vytvorí derivačný strom, ktorý slúži ako vstup pre sémantickú analýzu. Po dokončení sémantickej analýzy vznikne z derivačného stromu štruktúra nazývaná syntaktický strom. Ďalšou fázou prekladu je predanie syntaktického stromu generátoru vnútorného kódu, ktorý z neho vytvorí vnútorný kód. Jedným z najčastejšie používaných formátov vnútorného kódu je trojadresný kód. Tento kód je v prekladačoch výhodný hlavne z dôvodu jeho jednoduchej optimalizácie, ktorá nie je pri preklade nutná, no bez nej je výsledný kód menej efektívny a preto sa často využíva. Poslednou časťou prekladača je generátor cieľového kódu, ktorý má po väčšinou binárnu formu.

Táto bakalárska práca je zameraná na vytvorenie všeobecného generátora syntaxou riadeného prekladu. Predmetom záujmu tejto práce bude syntaxou riadený preklad, ktorý je vykonávaný syntaktickým analyzátorom už počas analýzy zdrojového programu. Tento preklad obvykle produkuje syntaktický strom. Všeobecný generátor spomínaného prekladu označuje to, že práca sa nezameriava na jeden konkrétny spôsob prekladu, ale podľa vloženej tabuľky symbolov a vstupných pravidiel preloží zadaný vstupný reťazec. Po dokončení prekladu vygeneruje program postupnosť trojadresných inštrukcií v tvare trojadresného kódu a vykreslí syntaktický strom, ktorý prislúcha týmto inštrukciám. Celá práca je rozdelená chronologicky do štyroch kapitol.

V kapitole 1 sú uvedené základné pojmy ako abeceda, jazyk a gramatika, na ktoré sa nadväzuje v nasledujúcich kapitolách práce. Tieto pojmy sú vysvetlené formou definícii. Ich cieľom je načrtnutie základov potrebných k danej problematike.

Kapitola **2** popisuje jednotlivé časti prekladača a jeho štruktúru. Podkapitoly postupne pojednávajú o lexikálnej, syntaktickej, sémantickej analýze a nakoniec aj o generovaní výstupného kódu a ich vzájomných prepojeniach.

Ďalšia kapitola **3** je venovaná syntaxou riadenému prekladu, ktorý je tvorený, buď analýzou zdola-nahor alebo zhora-nadol. Okrem týchto analýz je v tejto časti spomenutá aj tabuľka symbolov, ktorá má v tejto práci dôležitú úlohu. Vysvetlené sú tu aj kontroly, ktoré vykonáva sémantická analýza.

Posledná kapitola **4** je venovaná implementácii všeobecného generátora syntaxou riadeného prekladu. V tejto časti je uvedený návrh celej aplikácie, popis vykonávania konkrétnych častí prekladu, príklady spracovania vstupného reťazca a aj vykreslenie syntaktického stromu. Ďalej sa v tejto kapitole nachádza popis implementácie jednotlivých častí programu a aj manuál, v ktorom je uvedené ako sa výsledný program ovláda.

1 Základné pojmy

Keďže je táto práca postavená z veľkej časti na gramatikách, ktorých základom sú práve pojmy ako *abeceda, jazyky, gramatika* a pod., je táto kapitola zameraná na vysvetlenie týchto pojmov, ktoré sú potrebné pre porozumenie a následné prezentovanie ďalšej látky. Ako prvé si zavedieme pojem abeceda, na ktorý potom naviažeme a postupne rozšírime až po gramatiku. Všetky podkapitoly obsahujú definície jednotlivých pojmov a taktiež aj jednotlivé operácie. Definície v tejto kapitole sú prevzaté z [1] a [2].

1.1 Abeceda

Rovnako ako v bežnom živote, tak aj pre prekladače je abeceda základom pre tvorbu textu. Popisuje sadu znakov a symbolov, s ktorými môžeme pracovať. Operácie, ktoré môžeme nad abecedou vykonávať, teda predstavujú počet výskytov jedného konkrétneho písmena samého za sebou alebo taktiež opakovanie znakov. Poskladaním jednotlivých písmen do skupín vznikne reťazec. V reálnom živote sa z takto vzniknutých reťazcov stávajú slová, vety alebo aj celé stránky textu, keďže do reťazca zaraďujeme aj medzery a iné symboly. Nad každým reťazcom môžeme vykonávať rôzne operácie ako napríklad obrátenie, spojenie s iným reťazcom alebo jeho rozdelenie.

Definícia 2.1.1 Abeceda a symboly

Abeceda je ľubovoľná konečná neprázdna množina. Prvky abecedy nazývame symboly.

Definícia 2.1.2 Reťazec

Nech $\sum$ je abeceda.

- ε je prázdny reťazec nad abecedou $\sum$
- Pokiaľ x je reťazec nad abecedou $\sum$ a $b \in \sum$, potom xb je reťazec nad abecedou $\sum$

Definícia 2.1.3 Dĺžka reťazca

Nech x je reťazec nad abecedou $\sum$. Dĺžka reťazca x, $|x|$, je definovaná:

- pokiaľ je $x = \varepsilon$, potom $|x| = 0$
- pokiaľ $x = b_1, \ldots, b_n$, potom $|x| = n$ pre $n \geq 1$ a $b_i \in \Sigma$ pre všetky $i \geq 1$

Definícia 2.1.4 Konkatenácia reťazca

Nech x a b sú dva reťazce nad abecedou $\sum$. Konkatenácia x a b je reťazec xb.

Definícia 2.1.5 Prefix reťazca

Nech x a b sú 2 reťazce nad abecedou ∑ . x je prefixom b pokiaľ existuje reťazec c nad abecedou ∑ taký, že platí xc = b.

Definícia 2.1.6 Sufix reťazca

Nech x a b sú 2 reťazce nad abecedou ∑ . x je sufixom b pokiaľ existuje reťazec c nad abecedou ∑ taký, že platí cx = b.

Definícia 2.1.7 Podreťazec

Nech x a b sú 2 reťazce nad abecedou ∑ . x je podreťazcom b pokiaľ existuje reťazec c a c´ nad abecedou ∑ taký, že platí cxc´ = b.

Definícia 2.1.8 Mocnina reťazca

Nech x je reťazec nad abecedou ∑ . Pre i ≥ 0, i-tá mocnina reťazca x, x^i je definovaná ako:

- $x^0 = \varepsilon$
- pre i ≥ 1 : $x^i = xx^{i-1}$

Definícia 2.1.9 Reverzácia reťazca

Nech x je reťazec nad abecedou ∑ . Reverzácia reťazca x, *reversal*(x) je definovaná:

- keď x = Ɛ potom reversal (x) = Ɛ
- keď x = b_1, ... , b_n , potom reversal(x) = b_n, … , b_1 pre n ≥ 1 a $b_i \in \Sigma$ pre všetky i ≥ 1

1.2 Jazyky

Podobne ako abeceda, tak aj jazyky sú do značnej miery inšpirované skutočnými jazykmi. Z toho dôvodu sú im aj v mnohých ohľadoch dosť podobné. Vymenovaním všetkých reťazcov, ktoré patria do daného jazyka, dostaneme samotný jazyk. Ak je jazyk tvorený obmedzeným, teda konečným množstvo reťazcov nazýva sa konečný. Jazyk ktorý nemá zadané limity na počet reťazcov sa nazýva nekonečný. Musí mať limity na počet znakov v reťazcoch, aby reťazec patril do tohto jazyka.

Definícia 2.2.1 Jazyk

Nech ∑ je abeceda a ∑* je množina všetkých reťazcov nad touto abecedou. Potom každá podmnožina L ⊆ ∑* je jazyk nad abecedou ∑.

Z definície vyplýva, že ∅ a {Ɛ} sú jazyky nad každou abecedou, ale ∅ ≠ {Ɛ}, keďže ∅ neobsahuje žiaden element, zatiaľ čo {Ɛ} má práve jeden element Ɛ.

Definícia 2.2.2 Konečný a nekonečný jazyk

Nech L je jazyk. L je konečný, ak obsahuje konečný počet reťazcov, inak je L nekonečný.

Definícia 2.2.3 Konkatenácia jazykov

Nech L_1 a L_2 sú dva jazyky nad abecedou ∑. Konkatenácia jazykov L_1 a L_2, L_1L_2 je definovaná ako

$$L_1L_2 = \{xy : x \in L_1 \textbf{ a } y \in L_2 \}$$

Podľa tejto definície má každý jazyk tieto vlastnosti

1. $L\{Ɛ\} = \{Ɛ\}L = L$
2. $L∅ = ∅L = ∅$

Definícia 2.2.4 Zjednotenie, prienik a rozdiel jazykov

Nech L_1 a L_2 sú dva jazyky nad abecedou ∑.

Zjednotenie jazykov L_1 a L_2, $L_1 \cup L_2$ je definované ako

$$L_1 \cup L_2 = \{x : x \in L_1 \textbf{ alebo } x \in L_2 \}$$

Prienik jazykov L_1 a L_2, $L_1 \cap L_2$ je definovaný ako

$$L_1 \cap L_2 = \{x : x \in L_1 \textbf{ a } x \in L_2\}$$

Rozdiel jazykov L_1 a L_2, $L_1 - L_2$ je definovaný ako

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

Definícia 2.2.5 Doplnok jazyka

Nech L je jazyk nad abecedou ∑. Doplnok jazyka L , $\overline{L}$ je definovaný ako

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

Definícia 2.2.6 Reverzácia jazyka

Nech L je jazyk nad abecedou ∑. Reverzácia jazyka L, reversal(L), je definovaná ako

$$\text{reversal(L)} = \{ \text{reversal(x)} : x \in L \}$$

Definícia 2.2.7 Mocnina jazyka

Nech L je jazyk nad abecedou ∑. Pre i ≥ 0, i-tá mocnina jazyka L, L^i je definovaná ako

1. $L^0 = Ɛ$
2. Pre všetky i ≥ 1, $L^i = LL^{i-1}$

Definícia 2.2.8 Iterácia jazyka

Nech L je jazyk nad abecedou ∑. Iterácia jazyka L, L^* je definovaná ako

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

Pozitívna iterácia jazyka L, L^+ je definovaná ako

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

1.3 Gramatika

K tomu, aby sme definovali nekonečný jazyk existuje niekoľko alternatív. Tá najjednoduchšia, ale neuplatniteľná, je nekonečný zoznam všetkých reťazcov, ktoré patria do daného jazyka. Tým druhým spôsobom je definovanie gramatiky. Gramatika je všeobecne založená na konečnej množine terminálov a neterminálov, počiatočnom neterminále, ako aj konečnej množine gramatických pravidiel. Po uplatnení gramatických pravidiel v gramatickom automate na všetky neterminály a terminály generujeme reťazce jazyka. Vygenerovaný jazyk je potom plne závislý na týchto pravidlách.

Definícia 2.3.1 Bez kontextová gramatika

Bez kontextová gramatika je štvorica G = (N, T, P, S) , kde

- N – je abeceda neterminálov
- T – je abeceda terminálov
- P – je konečná množina pravidiel v tvare $A \rightarrow x$, kde $A \in N$, $x \in (N \cup T)^*$
- S – je počiatočný neterminál, $S \in N$

Pozn.

1. $A \rightarrow x$ znamená, že neterminál A sa po aplikovaní pravidla P prepíše na terminál x
2. $A \rightarrow \varepsilon$ je nazývané ε-pravidlo

1.3.1 Derivačný krok

Používanie jednotlivých pravidiel gramatiky je potrebné riadiť a rovnako tak vedieť popísať. K popisu týchto pravidiel nám slúži derivačný krok. Derivačný krok vyjadruje zjednodušene zamenenie vstupného reťazca za výstupný reťazec s použitím gramatického pravidla. Tento výstupný reťazec sa

v nasledujúcom kroku stane vstupným reťazcom. Niekoľko derivačných krokov za sebou je ponímaných ako sekvencia derivačných krokov, ktorá sa navonok môže javiť ako jeden derivačný krok. Rovnako ako bežný derivačný krok aj sekvencia krokov má svoj vstup a výstup.

Definícia 2.3.2 Derivačný krok v bezkontextovej gramatike

Nech G = (N, T, P, S) je bezkontextová gramatika. Nech $u, v \in (N \cup T)^*$ a $p = A \rightarrow x \in P$. Potom uAv priamo derivuje uxv za použitia p v G, zapísané $uAv => uxv$ $[p]$ alebo zjednodušene $uAv => uxv$.

Definícia 2.3.3 Nula derivačných krokov

Nech G = (N, T, P, S) je bezkontextová gramatika. Nech $u \in (N \cup T)^*$. G vykoná nula derivačných krokov z u do u. Zapisujeme $u =>^0 [\varepsilon]$ alebo zjednodušene $u =>^0 u$.

Definícia 2.3.4 n - derivačných krokov

Nech G = (N, T, P, S) je bezkontextová gramatika. Nech $u_0, \ldots, u_n \in (N \cup T)^*$, $n \geq 1$ a $u_{i-1} => u_i$ $[p_i]$, $p_i \in P$ pre všetky i = 0, ... , n, čo znamená $u_0 => u_1$ $[p_1] => u_2$ $[p_2]$ ... $=> u_n$ $[p_n]$. Potom G vykoná n derivačných krokov $u_0 =>^n u_n$ $[p_1 \ldots p_n]$ alebo zjednodušene $u_0 =>^n$.

1.3.2 Generovaný jazyk

Po vykonaní všetkých možných derivačných krokov alebo sekvencií derivačných krokov dostaneme množinu jazykov, ktorá sa nazýva generovaný jazyk. Tento jazyk je zložený z terminálov, ktoré reprezentujú jednotlivé časti formálneho jazyka.

Definícia 2.3.5 Generovaný jazyk

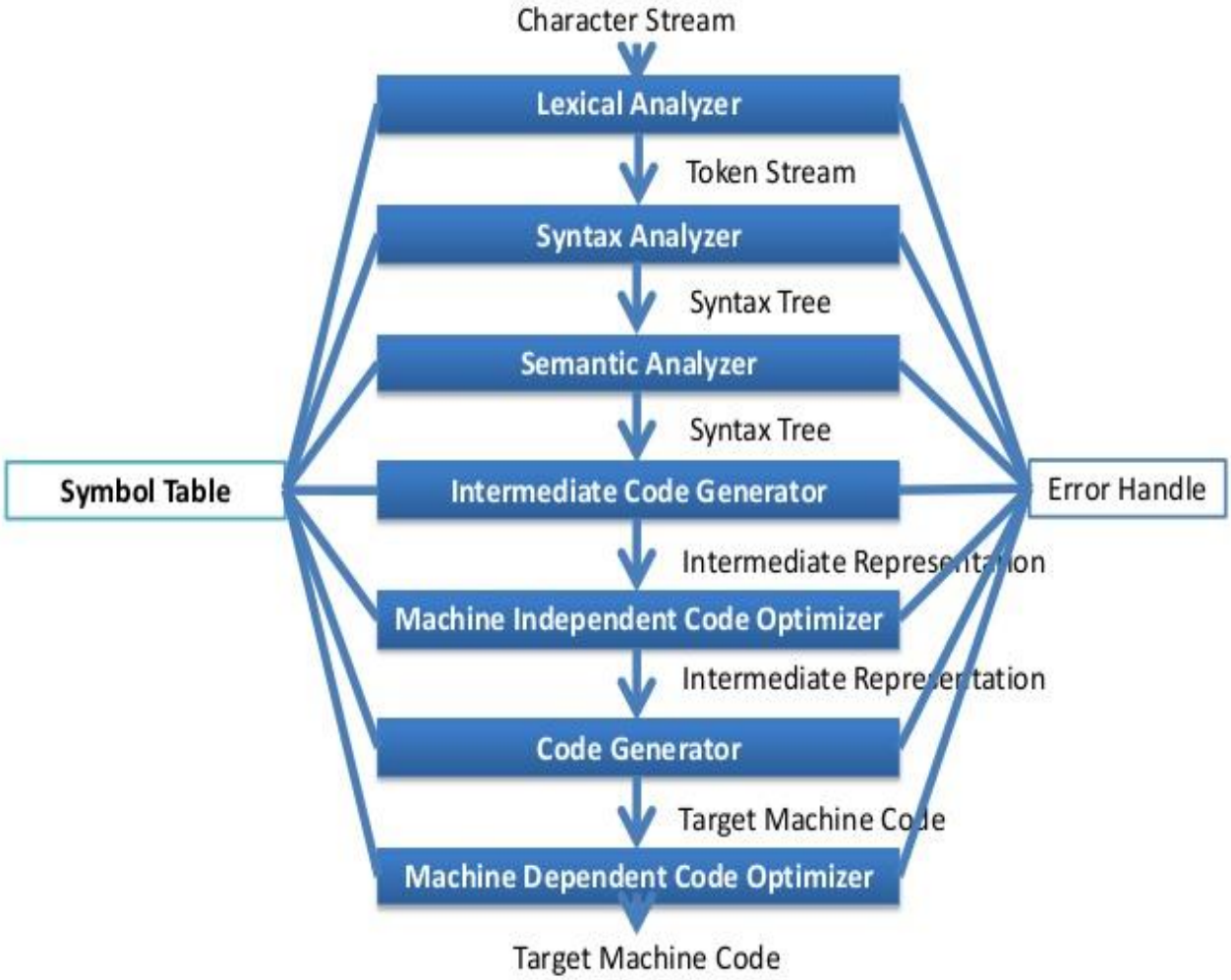
Nech G = (N, T, P, S) je bezkontextová gramatika. Jazyk generovaný bezkontextovou gramatikou G, $L(G) = \{w: w \in T^*, S =>^* w\}$.

Definícia 2.3.5 Bezkontextový jazyk

Nech L je jazyk. L je bezkontextový jazyk, ak existuje bezkontextová gramatika, ktorá generuje tento jazyk.

2 Štruktúra prekladača

Táto kapitola je zameraná na štruktúru prekladača, konkrétne proces prekladu. Vstupom pre prekladač sú obvykle reťazce znakov, ktoré sa počas jednotlivých častí prekladu spracúvajú a analyzujú. V každej fáze sa transformuje jedna reprezentácia zdrojového programu do inej. Typická dekompozícia prekladača je znázornená na obrázku 2.1.



Obrázok 2.1 Dekompozícia prekladača [4]

2.1 Lexikálna analýza

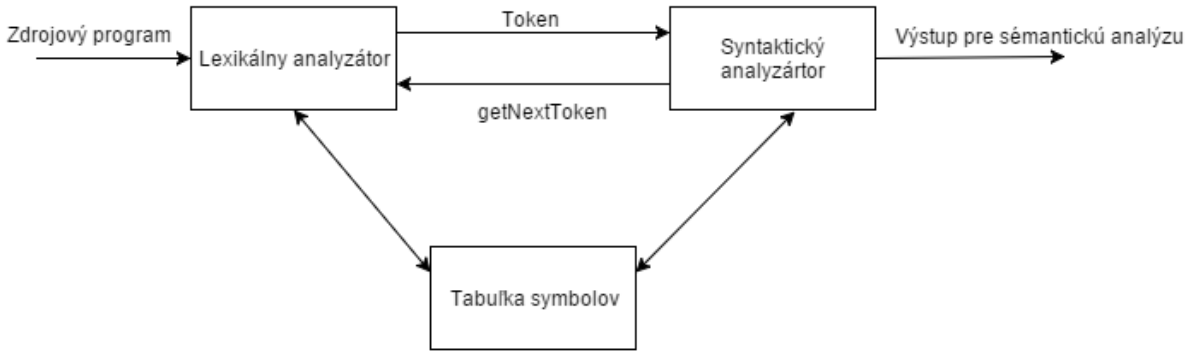
Prvou fázou prekladača je lexikálna analýza, je tiež označovaná ako skener. Vstupom lexikálneho analyzátora je reťazec znakov, z ktorých pozostáva zdrojový program. Následne je tento reťazec rozdelený do skupín znakov podľa významu. Tieto skupiny nazývame lexémy. Pre každú lexému skener vyprodukuje na svoj výstup *token*, ktorý má tvar: < *názov tokenu* , *hodnota jeho atribútu* >. Prvú časť tokenu – označovaná *názov tokenu* je abstraktný symbol, ktorý sa používa počas syntaktickej analýzy. Druhá časť, ktorá je označená ako *hodnota jeho atribútu* je odkaz do tabuľky symbolov, kde je uložená hodnota konkrétneho tokenu. Táto hodnota bude potrebná vo fáze, sémantickej analýzy a generovania kódu. Po vytvorení sú jednotlivé tokeny posielané syntaktickému analyzátoru na spracovanie.

Okrem svojej základnej funkcionality plní lexikálny analyzátor aj niekoľko menších úloh. Jednou z nich je spolupráca s tabuľkou symbolov. Používa sa aj na zjednodušenie zdrojového textu, tým spôsobom, že odstráni biele znaky a komentáre. [3]

2.2 Syntaktická analýza

Druhá fáza prekladu sa nazýva syntaktická analýza alebo parser. Syntax zdrojového programu je definovaná gramatikou, ktorá je založená na konečnom počte pravidiel. Na základe týchto pravidiel dochádza k overovaniu toho, či vstupný reťazec tokenov od skeneru patrí do generovaného jazyka danej gramatiky. Hlavnou úlohou parsera je vygenerovanie syntaktického stromu.

Na obrázku 2.2 je znázornená komunikácia medzi lexikálnym, syntaktickým analyzátorom a tabuľkou symbolov.



Obrázok 2.2

2.3 Sémantická analýza

Tretia fáza prekladu je sémantická analýza. Jej vstupom je syntaktický stom a informácie o tabuľke symbolov. Hlavnou úlohou tejto časti prekladača je overiť, či sa zdrojový program sémanticky zhoduje s definovaným jazykom. Taktiež zhromažďuje informácie o typoch a ukladá ich buď do tabuľky symbolov alebo do syntaktického stromu. Tieto informácie sa následne použijú pri generovaní vnútorného kódu. Dôležitou súčasťou sémantickej analýzy je typová kontrola, kde prekladač kontroluje, či sú správne priradené operandy k jednotlivým operátorom. Mnohé jazyky dovoľujú takzvané *pretypovanie*, ktoré môže byť buď implicitné alebo explicitné. Implicitné pretypovanie robí prekladač na pozadí, bez vedomia programátora. Používa sa to napríklad pri sčítaní dvoch čísel, ktoré nie sú rovnakého typu (celé číslo a desatinné číslo). V tomto prípade si musí prekladač previesť celé číslo na desatinné a uložiť si pretypovaný výsledok do pomocnej premennej. Explicitné pretypovanie zaisťuje programátor.

2.4 Generovanie vnútorného kódu

Kompilátor môže pri procese prekladu zdrojového programu na cieľový program vytvoriť jeden, prípadne aj viac vnútorných reprezentácií prekladaného programu. Tieto reprezentácie nemusia mať rovnakú podobu akú má cieľový program, ale môžu nadobudnúť rozličné formy. Syntaktické stromy sú jednou z foriem vnútornej reprezentácie a preto sú vo všeobecnosti používané pri syntaktickej a sémantickej analýze. Po týchto analýzach väčšina prekladačov pristúpi ku vygenerovaniu nízkoúrovňového vnútorného kódu, ktorý by mal spĺňať dve základné vlastnosti: mal by byť jednoduchý na vytvorenie a mal by byť jednoducho preložiteľný na cieľovom stroji.
Medzi základné vnútorné formy kódu patrí trojadresný kód, post-fixová notácia alebo prefixová notácia.

3 Syntaxou riadený preklad

Ak lexikálna a syntaktická kontrola zdrojového programu neodhalila chybu, preloží kompilátor tento zdrojový program do podoby vnútorného kódu. Tento preklad je obvykle riadený syntaktickým analyzátorom už počas analýzy programu. Preto sa tento preklad označuje ako *syntaxou riadený preklad*. Avšak, syntaxou riadený preklad nepoužíva parsovací strom ako vnútornú reprezentáciu programu, keďže tento strom obsahuje nadbytočné informácie ako napríklad neterminály. Namiesto toho obvykle produkuje *abstraktný syntaktický strom* alebo kratšie *syntaktický strom*. Tento preklad takisto vykonáva aj sémantické kontroly, ako napríklad typová kontrola.
Syntaxou riadený preklad je možné rozdeliť do dvoch skupín podľa spôsobu prekladu zdrojového programu a to na*: zdola-nahor* a *zhora-nadol*. [5]

3.1 Analýza zdola-nahor (Bottom - Up)

V tejto kapitole si vysvetlíme ako funguje preklad pri analýze zdola-nahor. Následne si ukážeme rôzne typy vnútorných reprezentácií kódu, ako napríklad syntaktické stromy, trojadresný kód a post-fixovú notáciu.

Uvažujme o G-založenom zdola-nahor analyzátore $M = ({}_M\sum, {}_MR)$ pre gramatiku $G = ({}_G\sum, {}_GR)$. Syntaxou riadený preklad sprevádzaný analyzátorom M používa *pd* symboly s atribútmi. Na vyjadrenie toho, že je atribút a_i priradený k zásobníkovému symbolu pd_i je použitý zápis $pd_i\{a_i\}$, $1 \leq i \leq |pd|$. Ak M vykoná redukciu pravej strany pravidla ${}_GR$, označovanej ako „handle“, potom syntaxou riadený preklad vykoná akciu, ktorá je pripojená k tomuto pravidlu. Akcia pripojená k pravidlu
r: $X_0 \rightarrow X_1X_2 \ldots X_n \in {}_GR$ je popísaná použitím *formálnych atribútov* \$\$, \$1, \$2, … , \$n pripojených k symbolom $X_0, X_1, X_2, \ldots, X_n$ v r, kde \$\$ je atribút priradený k X_0 a \$j je atribút priradený k j-tému symbolu X_j. Túto akciu označíme ako **ACTION**(\$\$, \$1, \$2, … , \$n) a bude umiestnená za pravidlo r.

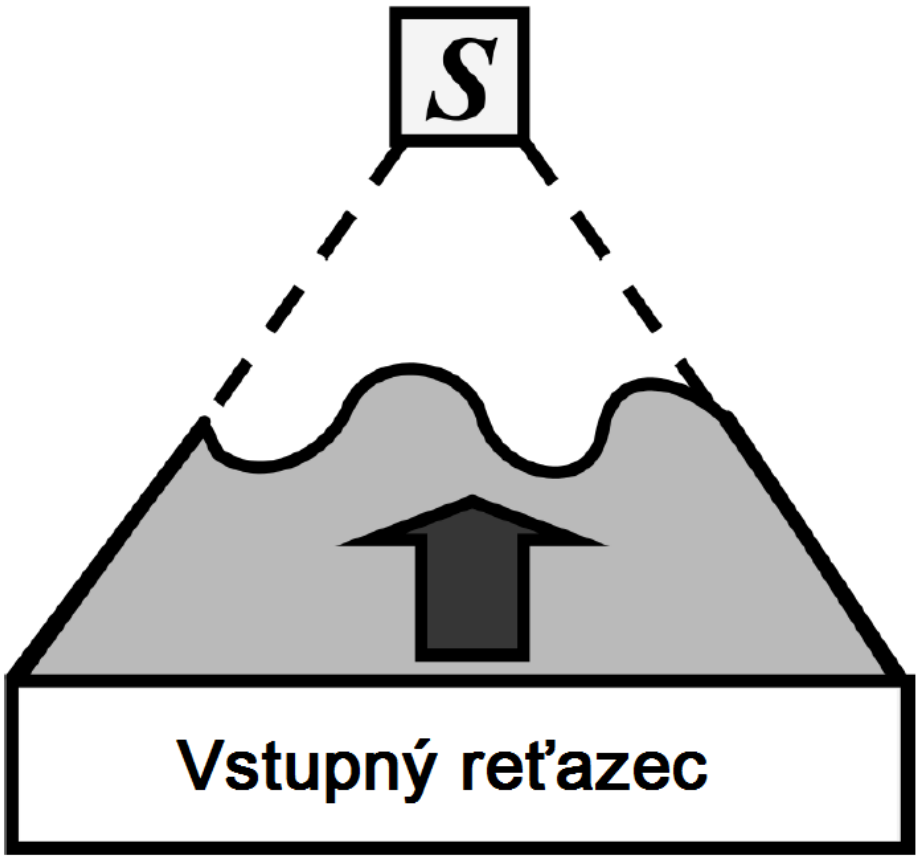
$$X_0 \rightarrow X_1X_2 \ldots X_n \quad \{\ \textbf{ACTION}(\$\$, \$1, \$2, \ldots, \$n)\ \}$$

Pre explicitné označenie spojenia medzi X_j a \$j, sa niekedy pridáva {\$j} za symbol X_j. [5]

$$X_0\{\$\$\} \rightarrow X_1\{\$1\}\ X_2\{\$2\} \ldots\ X_n\{\$n\} \quad \{\ \textbf{ACTION}(\$\$, \$1, \$2, \ldots, \$n)\ \}$$

Na druhej strane je v praxi nie je potrebné pridávať atribút hneď za symbol v pravidle. Preto sa v pravidle atribút vynecháva. Takto upravené pravidlá označujeme ako atribútové pravidlá. Zmenou

gramatických pravidiel (G) na atribútové pravidlá dostaneme *atribútovú G založenú gramatiku*. Na obrázku 3.1 je znázornená analýza zdola-nahor.

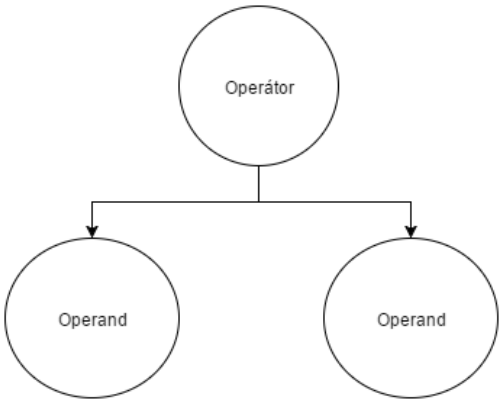


Obrázok 3.1 Analýza zdola-nahor

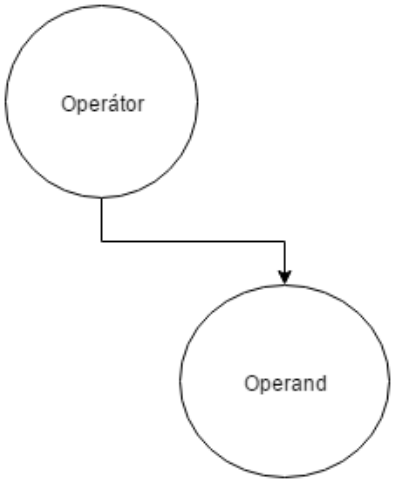
3.1.1 Syntaktické stromy

Syntaktické stromy rovnako ako parsovancie stromy graficky zobrazujú syntaktickú štruktúru zdrojového programu. Avšak na rozdiel od parsovacích stromov sú zbavené nadbytočných informácií, ako sú napríklad neterminály.

Syntaktický strom je binárny strom, ktorého listy sú tvorené uzlami, ktoré obsahujú operátory alebo operandy. Ak má uzol dvoch potomkov je teda binárnym operátorom (obrázok 3.2) a ak má iba jedného potomka je unárnym operátorom (obrázok 3.3).



Obrázok 3.2 – Syntaktický strom s doma potomkami.



Obrázok 3.3 – Syntaktický strom s jedným potomkom.

Syntaxou riadený preklad, ktorý generuje syntaktický strom využíva tri akcie pripojené ku gramatickým pravidlám.

1. $$:= UZOL($l) alokuje nový uzol označený $l a nastaví $$, tak aby ukazovala na tento novovytvorený uzol.

2. $$:= STROM(o, $j) vytvorí syntaktický strom o <strom($j)> , kde *o* je unárny operátor a nastaví $$ na ukazovateľ novovytvoreného stromu.

3. $$:= STROM(o, $j, $k) vytvorí syntaktický strom o <strom($j), strom($k)>, kde *o* je binárny operátor. Potom táto akcia nastaví $$ na ukazovateľ koreňa tohto stromu.

3.1.2 Trojadresný kód

Inštrukcie trojadresného kódu reprezentujú jednoduché príkazy, ktoré sú ľahko konvertovateľné do jazyka symbolických inštrukcií (asembleru). Vo všeobecnosti sa *trojadresný kód* zapisuje ako štvorica,

[O, X, Y, Z]

ktorej komponenty O, X, Y a Z označujú operátor, prvý operand, druhý operand a výsledok. X, Y a Z sú obvykle reprezentované ako adresy jednotlivých premenných v tabuľke inštrukcií. Z tohto dôvodu sa označuje táto vnútorná reprezentácia ako *trojadresný kód.*

Inštrukcia trojadresného kódu, ktorá reprezentuje unárny operand má túto podobu:

[O, X, ,Z]

V tomto zápise O reprezentuje unárny operátor, X je jediný operand tejto inštrukcie, tretí komponent je prázdny a Z je výsledok. Táto inštrukcia nastaví Z ako výsledok operácie O aplikovanej na X. Mnohokrát častejšie sa však používajú trojadresné inštrukcie s dvoma operandmi, ktoré majú túto podobu:

[O, X, Y, Z]

kde O je binárny operátor, X a Y sú operandy a Z je výsledok. V tejto reprezentácii nie je žiadna položka prázdna. Tento inštrukčný set vloží do Z výsledok operácie O aplikovanej na X a Y.

Trojadresný kód reprezentuje jeden výraz vo vyššom programovacom jazyku a je obvykle tvorený dlhou postupnosťou trojadresných inštrukcií. V tomto kóde sa nachádza veľa dočasných premenných, ktoré kompilátor generuje, aby si v nich udržiaval jednotlivé medzivýsledky. Tieto medzivýsledky sú potrebné pre vykonanie nasledujúcich inštrukcií.

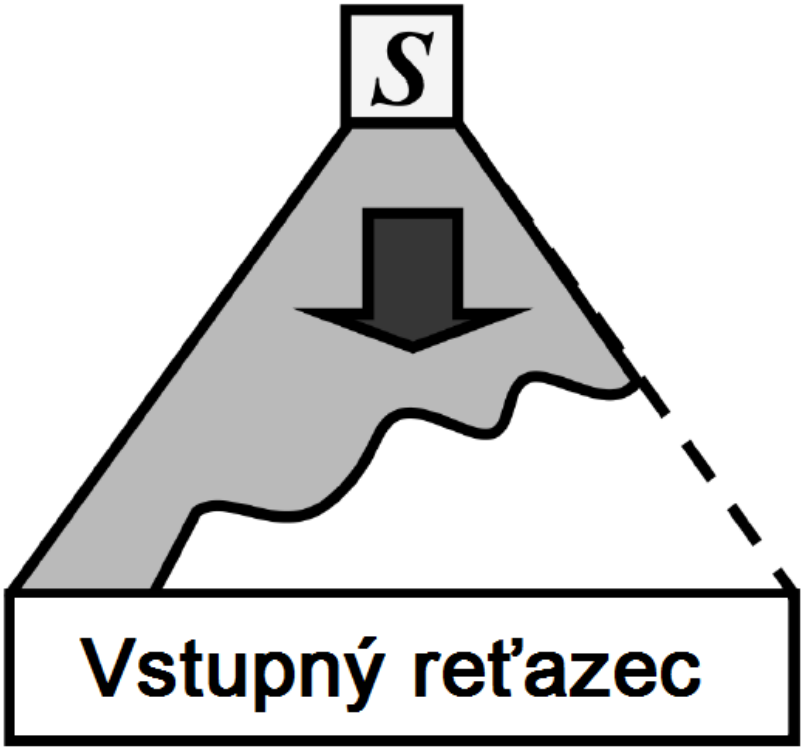
3.1.3 Poľská notácia

V prevažnej väčšine prípadov sa v praxi pri symboloch v gramatických pravidlách nevyskytujú atribúty. V skutočnosti je pri generovaní vnútorného kódu, ako je post-fixová Poľská notácia potrebné priradiť atribúty k jednotlivým terminálom [6].
V post-fixovej notácií uvádzame operátor za jeho operandy. Pre výraz E môžeme Poľskú notáciu definovať nasledovne:

1. Ak je E premenná alebo konštanta, potom post-fixová notácia pre E je E samotné.
2. Ak je E výraz, ktorý má formu E_1 **op** E_2, kde **op** binárny operátor, tak post-fixová notácia pre E má tvar $E_1´$ $E_2´$ **op,** kde $E_1´$ a $E_2´$ sú tvoria Poľskú notáciu pre E_1 a E_2 .
3. Ak je E v zátvorkách a má podobu (E_1), potom post-fixová notácia pre E je rovnaká ako pre E_1.

3.2 Analýza zhora-nadol (Top - Down)

Analýza zhora-nadol, je situovaná tak, že sa začína zhora a postupne sa postupuje smerom dole. Na začiatku je neterminál počiatočný symbol. Tento neterminál predstavuje vrchol budúceho derivačného stromu. Modelom tejto analýzy sú expanzívne pravidlá, ktoré z počiatočného symbolu a neskôr z ďalších neterminálov expandujú a vytvárajú tak nový derivačný strom, tak aby boli konce jednotlivých vetiev tohto stromu zakončené terminálmi. Celý strom teda znázorňuje, ktorý jednotlivý terminál zapadá do konkrétnej časti výsledného derivačného stromu, podľa obrázku 3.4. [7].



Obrázok 3.4 Analýza zhora-nadol

3.3 Sémantická analýza

Popri generovaní vnútorného kódu, vykonáva syntaktický analyzátor prekladača aj iné akcie. Jednou z týchto činností je aj overovanie rôznych sémantických aspektov zdrojového programu. Všetky tieto akcie sú označované ako *sémantická analýza*. Táto analýza obvykle vykonáva štyri kontroly.

Prvou z týchto kontrol je *kontrola toku*. Ak prekladač nájde príkaz skoku, dôjde pomocou tejto kontroly k overeniu, či miesto kam sa má pomocou tohto výrazu skočiť existuje. V prípade, že miesto kam sa má skočiť neexistuje dôjde v prekladači k sémantickej chybe.

Kontrola jedinečnosti je druhou zo sémantických kontrol. Niektoré objekty, ako napríklad identifikátori a pod. musia byť definované v zdrojovom programe iba raz. Táto sémantická kontrola overuje unikátnosť pre tieto definície.

Tretia kontrola je *kontrola mennej príslušnosti*. Niektoré názvy sa v zdrojovom programe objavujú aj viac ako dva-krát. Sémantická analýza musí vedieť určiť príslušnosť každého z týchto mien. To znamená určiť, ktorého bloku v zdrojovom texte sa tento názov týka.

Poslednou a pravdepodobne najdôležitejšou z týchto štyroch kontrol je *kontrola typu operandu*. Sémantický analyzátor vykonáva typovú kontrolu tak, že zistí, či sú operandy kompatibilné s danou operáciou, ktorá sa má nad nimi vykonať. V prípade nekompatibility sa pokúsi tento problém vyriešiť. Najčastejšími riešeniami sú typové konverzie alebo ohlásenie sémantickej chyby. Počas typovej kontroly dochádza k zmene typu jedného z operandov na kompatibilný typ pre danú operáciu. Napríklad pri sčítaní celého a desatinného čísla dochádza ku konverzii celočíselného formátu čísla na desatinný formát.

3.4 Tabuľka symbolov

Použitie tabuľky symbolov je takmer v každej fáze prekladu. Najvýraznejší využitie je pravdepodobne pri analýze zdrojového programu, kedy poskytuje sémantickému analyzátoru informácie potrebné ku sémantickej kontrole zdrojového programu. Následne pri generovaní kódu pomáha generátoru k tomu, aby bol výsledný cieľový program správny a efektívny. Z týchto dôvodov je potrebné, aby mechanizmus práce s tabuľkou symbolov umožňoval vkladať nové položky do tabuľky a vyhľadávať už existujúce položky v čo najkratšom čase a čo najefektívnejšie.

Tabuľka symbolov je obvykle reprezentovaná ako jedna z týchto štruktúr:

- *Pole*
- *Zoznam*
- *Binárny vyhľadávací strom*
- *Hašovacia tabuľka*

Pole je jednoduchý a ekonomický ukladací priestor. Tieto vlastnosti zároveň reprezentujú jeho výrazné výhody. Neobsahuje žiadne ukazovatele, ktoré zaberajú pamäť, ale samé neobsahujú žiadne dáta. Základnou nevýhodou poľa je jeho stabilná veľkosť, ktorú je potrebné poznať v predstihu. Tento nedostatok limituje počet identifikátorov. Ďalšia nevýhoda sa prejavuje pri vyhľadávaní, pretože pole je potrebné prejsť sekvenčne a takýto prechod je v praxi príliš pomalý.

Druhou štruktúrou je *zoznam*. Hlavná výhoda zoznamu oproti poľu je v rozšíriteľnosti. Nie je teda potrebné vedieť vopred koľko položiek sa v tabuľke bude nachádzať. Vyhľadávanie v zozname je rovnako ako v poli sekvenčné avšak pri správnom zoradení sa dá rýchlosť vyhľadávania výrazne zvýšiť.

Binárny vyhľadávací strom ponúka oproti predchádzajúcim variantom rýchlejší prístup k dátam. Táto dátová štruktúra je flexibilná z pohľadu priestoru v pamäti a zároveň rýchla z pohľadu prístupu k dátam. Jednou z nevýhod binárneho vyhľadávacieho stromu je fakt, že pre každý uzol sú potrebné dva ukazovatele k adresovaniu ďalších uzlov.

Poslednou štruktúrou je *hašovacia tabuľka,* ktorá používa pole ukazovateľov, ktoré adresujú lineárne zoznamy nazývané reťaze. Samotné dáta sú uložené v týchto reťaziach, ktorých dĺžka je neobmedzená. Vyhľadávanie prebieha v konštantnom čase nezávisle na počte prvkov v tabuľke. Z tohto dôvodu sú hašovacie tabuľky symbolov v praxi veľmi obľúbené.

4 Implementácia

Táto kapitola sa venuje návrhu a implementácií konečnej aplikácie. Podľa zadania bolo potrebné, aby výsledná aplikácia bola schopná, prostredníctvom zadanej tabuľky symbolov a syntaktických pravidiel, spracovať vstupný reťazec a rozhodnúť, či je reťazec správny. Ak je reťazec správny aplikácia vykreslí syntaktický strom a vypíše inštrukcie v podobe trojadresného kódu.
Celá aplikácia je napísaná v jazyku Java - verzia 8. Na tvorbu grafického užívateľského rozhrania som použil toolkit Swing.

Jadrom programu je samotná syntaktická analýza. Zo zadania vyplýva potreba vykonávania syntaxou riadený preklad podľa požiadaviek používateľa. Základom programu sú dva textové súbory, ktoré zadáva užívateľ. Prvý obsahuje tabuľku symbolov a druhý obsahuje redukčné pravidlá. Na záver zadá používateľ reťazec, ktorý chce spracovať. Následne prebieha samotná syntaktická analýza, ktorá sa riadi jednotlivými pravidlami. V tejto fáze sa postupne volá lexikálna analýza pre postupné spracovanie vstupného reťazca. Ak sa v reťazci nevyskytuje žiadna chyba a je teda lexikálne, syntakticky aj sémantický správny dôjde k vygenerovaniu trojadresného kódu a vykresleniu syntaktického stromu.
Program pozostáva z viacerých tried, ktoré sú popísané v nasledujúcej podkapitole.

4.1 Rozvrhnutie aplikácie

Ako už bolo spomenuté, aplikácia je napísaná v Jave a preto bolo možné rozčleniť aplikáciu na jednotlivé logické celky a celú aplikáciu písať objektovo. Tieto objekty vznikli z tried, ktoré sú uvedené v tejto podkapitole. Základnými objektami v aplikácií sú

- *scanner* - venuje sa lexikálnej analýze,
- *token* - predstavuje výsledok lexikálnej analýzy,
- *parser* – spracúva vstupný reťazec pomocou syntaktickej analýzy,
- *rules* - obsahuje jednotlivé redukčné pravidlá,
- *table* – obsahuje tabuľka symbolov,
- *instructions* – predstavuje inštrukcie v podobe trojadresného kódu.

Trieda Token

Objekty vytvorené touto triedou tvoria základ v celej aplikácií. Trieda token je tvorená týmito základnými atribútmi, ktoré sú nevyhnutné pre prácu s tokenom:

- *type* – predstavuje typ tokenu, ktorý je reprezentovaný celým číslom a nadobúda hodnotu na základe toho, či tento token reprezentuje operátor, premennú alebo iný typ symbolu.
- *atribut* – reprezentuje znak, ktorý je v tomto tokene uložený.
- *pos_X* a *pos_Y* – tieto atribúty sú potrebné pri vykresľovaní syntaktického stromu na plátno, na ktorom určujú pozíciu daného tokenu.

V tejto triede sa nachádzajú aj metódy pre získanie a úpravu hodnôt jednotlivých atribútov. Metódy na získanie hodnôt atribútov tokenu sú: *get_Type(), get_Atribut(), get_Pos_X(), get_Pos_Y()*. Na zmenu jednotlivých atribútov slúžia metódy *set_TypeAtribut(Type, Atribut), set_Position(x, y)*.

Trieda Scanner

Táto trieda slúži na uloženie a spracovanie vstupného reťazca. Obsahuje dve základné metódy pre prácu so vstupným reťazcom. Prvou metóda je *get_next_token(Table)*. Táto metóda má ako svoj parameter tabuľku symbolov. Na základe tejto tabuľky a vstupného reťazca táto metóda vráti token, ktorý odpovedá symbolu vo vstupnom reťazci. Tento token predstavuje, buď operátor alebo premennú. Druhou metódou je *return_token(Token)*, ktorá slúži k vráteniu načítaného tokenu.

Trieda Table

Trieda Table slúži k uloženiu tabuľky symbolov. Obsahom tejto triedy je aj zoznam symbolov. Symbol predstavuje nejaký operátor napríklad: +, -, *, &, |, (,), …

Táto trieda obsahuje konštruktor, ktorý vytvorí objekt Table. Tento konštruktor má ako parameter názov súboru, v ktorom je uložená tabuľka symbolov. Pri vytváraní objektu Table sa postupne napĺňa aj zoznam symbolov. Samotná tabuľka je v tomto objekte uložená ako dvojrozmerné pole Tokenov. Pre prístup jednotlivým elementom tabuľky je k dispozícií metóda *get_table_element(i, j)*. Táto metóda má dva parametre *i* a *j*, ktoré slúžia ako indexy do tabuľky.

Trieda Rules

Trieda Rules obsahuje zoznam terminálov a zoznam pravidiel. Každé pravidlo pozostáva z ľavej a pravej strany. Ľavú stranu tvorí terminál a na pravej strane sa môžu nachádzať symboly aj terminály. jednotlivé pravidlá. Pre vytvorenie objektu tejto triedy sa požíva konštruktor, ktorý má dva parametre.

Prvým je cesta k súboru, kde sú jednotlivé pravidlá uložené v textovej podobe a druhý je tabuľka symbolov.

Príklad pravidiel: **E → E + E**

E → i

Trieda Parser

Táto trieda je z pohľadu funkcionality najdôležitejšia. Základnou metódou v tejto triede je metóda *parse (Table, input_string, rules)*. Táto metóda má tri parametre. Prvý z nich je tabuľka symbolov, druhým je vstupný reťazec a posledným sú pravidlá. Táto metóda postupne spracúva vstupný reťazec pomocou lexikálnej analýzy. K redukcii sa používa metóda *redukuj (stack, index_rule, rules, Table),* kde parameter *stack* je zásobník ktorý používa syntaktický analyzátor k ukladaniu tokenov (neterminálov) a ich následnú redukciu na terminály. Parameter *index_rule* určuje konkrétne pravidlo, podľa ktorého budú symboly zo zásobníka zredukované. Posledné dva parametre sú *rules,* reprezentuje objekt so všetkými pravidlami a *Table* je tabuľka symbolov. Výsledkom celého redukčného procesu je uložený zoznam inštrukcií, ktoré majú tvar trojadresného kódu.

4.2 Lexikálna analýza

Lexikálna analýza je v tejto aplikácií navrhnutá tak, aby bola čo najjednoduchšie implementovateľná a súčasne bola čo najúčinnejšia. Hlavnou funkciou lexikálnej analýzy je spracovať vstupný reťazec znakov a rozdeliť ho na jednotlivé tokeny. Delenie prebieha za pomoci informácií uložených v tabuľke symbolov. V tejto tabuľke sa nachádza zoznam všetkých symbolov, ktoré reprezentujú operátory. Na základe týchto znalostí vie lexikálny analyzátor rozhodnúť, aký typ tokenu má vrátiť syntaktickému analyzátoru. Symboly zo vstupného reťazca sa spracúvajú postupne jeden po druhom, pričom sa ignorujú medzery medzi jednotlivými znakmi. Ak sa načítaný znak zhoduje so symbolom uloženým v zozname symbolov potom lexikálny analyzátor vytvorí token, ktorého atribútom je práve tento operátor. V opačnom prípade bude mať novovytvorený token označenie premennej. Ak lexikálny analyzátor dôjde na koniec vstupného reťazca a nemá už žiaden znak, ktorý by spracoval, vráti hodnotu null, ktorá reprezentuje koniec.

Takto vytvorené tokeny sú postupne predávané syntaktickému analyzátoru, ktorý z nich zostavuje derivačný strom.

4.3 Syntaktická analýza a generovanie kódu

Táto aplikácia používa syntaktickú analýzu zdola nahor. Na začiatku je vytvorený prázdny zásobník. Najprv dôjde k identifikácií jednotlivých symbolov zo vstupného reťazca. K tomuto účelu sa využíva už spomínaná lexikálna analýza, ktorej výsledkom je vždy token. Syntaktický analyzátor (parser) následne porovná tento token z vrcholom zásobníka. Po tomto porovnávaní určí podľa tabuľky symbolov akou činnosťou má pokračovať. V tabuľke sa vyskytujú tri druhy operácií, ktoré môžu nastať.

Prvou operáciou je posuv (shift). Táto operácia je jednoduchá, keďže spočíva iba v presunutí vstupného tokenu na zásobník. Po tejto operácií sa presunutý token ocitne na vrchole zásobníka a veľkosť zásobníka sa zväčší.

Druhou operáciou je redukcia. Redukcia je na rozdiel od posuvu zložitá operácia. Hlavnou funkciou redukcie je nahradenie jedného alebo viacerých symbolov na zásobníku neterminálom. Prvou úlohou syntaktického analyzátora pri vykonávaní redukcie je určenie pravidla, podľa ktorého bude redukcia prebiehať. Je potrebné overiť každé pravidlo a nájsť to, ktoré presne vystihuje obsah zásobníka. Ak sa nenájde žiadne pravidlo, podľa ktorého by sa dal obsah zásobníka zredukovať dôjde k redukčnej chybe. V prípade, že bolo nájdené príslušné pravidlo pristúpi syntaktický analyzátor k samotnej redukcii.

Podľa veľkosti pravej strany pravidla postupne odoberá prvky na zásobníku. Ak sa v pravidle vyskytuje, operátor je potrebné aj vytvorenie príslušnej inštrukcie. Toto vytváranie sa deje počas odoberania elementov zo zásobníka. Inštrukcia má podobu trojadresného kódu. Potom na zásobník vloží token z ľavej strany príslušného pravidla. Nakoniec je táto inštrukcia uložená do zoznamu inštrukcií.

Poslednou operáciou, ktorá môže nastať je chyba. Táto chyba je označovaná ako chyba detegovaná v tabuľke symbolov. Táto chyba zvyčajne označuje chýbajúci operátor medzi dvomi premennými alebo naopak dva operátory za sebou alebo chýbajúcu zátvorku a podobne.

Syntaktická analýza je úspešná v prípade ak nenastala žiadna zo spomínaných chýb. V takom prípade sa na vrchole zásobníka vyskytuje iba jeden neterminál, ktorý je výsledkom úspešných redukcií vstupného reťazca. Z toho vyplýva, že musel byť spracovaný celý vstupný reťazec.

4.3.1 Príklad práce syntaktického analyzátora

V tejto časti bude predvedená práca syntaktického analyzátora. Ako demonštračné príklady som si zvolil dva výrazy. Prvý z nich je správny a neobsahuje žiadnu chybu. Druhý výraz je chybný a obsahuje syntaktickú chybu, pretože v ňom chýba operátor medzi **)** a znakom **c.**

1. Príklad: a * (b + c)
2. Príklad: (a + b) c

Na to, aby mohol syntaktický analyzátor v aplikácii rozhodnúť o tom, či je tento výraz správny alebo nesprávny, potrebuje tabuľku symbolov a redukčné pravidlá. Na ich základe je postavená celá precedenčná analýza, ktorá spracúva tento výraz.

Tabuľka symbolov, ktorú bude syntaktický analyzátor používať je znázornená na obrázku 4.1.

		Vstupný token					
		+	*****	**(**	**)**	**i**	**$**
Obsah zásobníka	**+**	>	<	<	>	<	>
	*****	>	>	<	>	<	>
	(	<	<	<	<	<	>
	)	>	>	!	>	!	>
	i	>	>	!	>	!	>
	$	<	<	<	!	<	!

Obrázok 4.1 Tabuľka symbolov

Symboly **+**, *****, **(**, **)**, reprezentujú operátory, **i** reprezentuje premennú a **$** reprezentuje koniec vstupného reťazca a zásobníka.

Jednotlivé položky v tabuľke označujú akciu, ktorá sa vykoná v každom kroku analýzy. Symbol < označuje posuv, > označuje redukciu a ! je označenie pre chybu.

Pravidlá používané pri redukcii:

1. E → E + E
2. E → E * E
3. E → (E)
4. E → i

V týchto pravidlách E vyjadruje neterminál a je symbol z tabuľky symbolov.

Po zadefinovaní tabuľky symbolov a redukčných pravidiel sa môže syntaktická analýza začať.

Spracovanie prvého príkladu a * (b + c) syntaktickým analyzátorom.

Obsah zásobníka	Vstupný reťazec	Akcia	Redukčné pravidlá
$	a * (b + c)	<	
$a	* (b +c)	>	4. E → i
$E	* (b +c)	<	
$E*	(b +c)	<	
$E*(	b + c)	<	
$E*(b	+ c)	>	4. E → i
$E*(E	+ c)	<	
$E*(E+	c)	<	

$E*(E+ c	)	>	4. E → i
$E*(E+E	)	>	1. E → E + E
$E*(E	)	<	
$E*(E)		>	3. E → (E)
$E*E		>	2. E → E * E
$E			

Po spracovaní celého reťazca ostal na zásobníku iba jeden neterminál E. Počas spracovania nedošlo k žiadnej chybe a teda zadaný reťazec: a * (b + c) je syntakticky správny. Po úspešnej syntaktickej analýze je vytvorený aj zoznam inštrukcii v trojadresnom kóde.

[+,b,c,$0]

[*,a,$0,$1]

Na prvom mieste v každej inštrukcii sa nachádza operátor. Na druhom a treťom mieste sú operandy a na poslednom mieste sa nachádza pomocná premenná, do ktorej je vložený výsledok spracovania operácie danej operátorom. Konečný výsledok spracovávaného výrazu je uložený v premennej označenej $1.

Spracovanie druhého príkladu (a + b) c syntaktickým analyzátorom. Pri spracovaní tohto výrazu bude využitá rovnaká tabuľka symbolov aj redukčné pravidlá ako v prvom príklade.

Obsah zásobníka	Vstupný reťazec	Akcia	Redukčné pravidlá
$	(a + b) c	<	
$(	a + b) c	<	
$(a	+ b) c	>	4. E → i
$(E	+ b) c	<	
$(E+	b) c	<	
$(E+b	) c	>	4. E → i
$(E+E	) c	>	1. E → E + E
$(E	) c	<	
$(E)	c	!	

Ako bolo spomenuté v tomto výraze sa vyskytuje syntaktická chyba. Preto nedošlo k úplnému spracovaniu vstupného reťazca. Trojadresné inštrukcie sa v programe vytvárajú priebežne, preto aj napriek tomu, že sa celý vstupný reťazec nepodarilo analyzovať vznikla po redukcii a + b inštrukcia

[+,a,b,$0].

4.4 Vykresľovanie syntaktického stromu

Významnou funkciou mojej aplikácie je vykreslenie syntaktického stromu. Vykreslenie prebieha po dokončení syntaktickej analýzy. Základ pre vykreslenie tvorí zoznam inštrukcií v tvare trojadresného kódu.

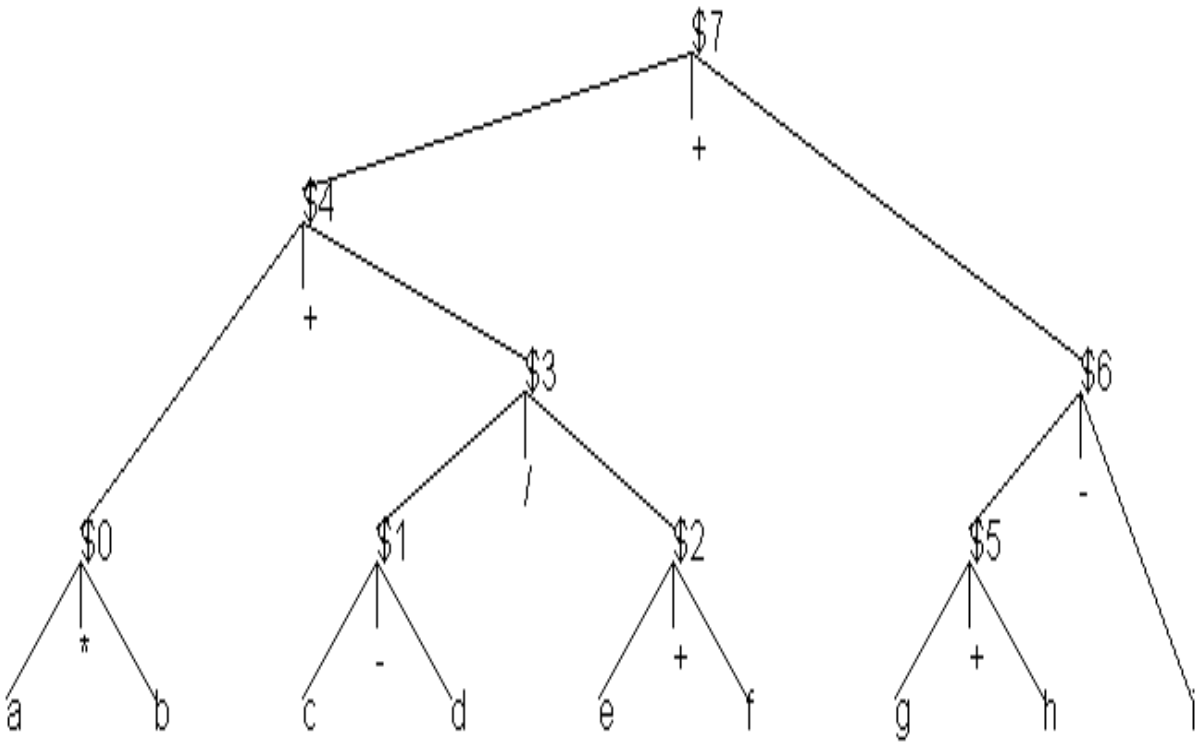
Algoritmus vykresľovania nie je veľmi zložitý. Program postupne prechádza celý zoznam inštrukcií. Potom spracúva každý symbol inštrukcie nasledovne:

- Ak aktuálny prvok nie je operátor a súčasne nebol ešte vykreslený tak sa vykreslí. Po vykreslení sa prvok uloží do zoznamu vykreslených prvkov spolu so súradnicami, na ktorých sa vykreslil.
- Po vykreslení dôjde k úprave vykresľovanej pozície, buď v smere osy x alebo osy y podľa toho aký prvok bol vykreslený.
- Operátor sa vykreslí medzi dvojicu operandov.
- Po vykreslení všetkých elementov inštrukcie sa vykreslia čiary medzi týmito prvkami.
- V prípade, že prvok už bol vykreslený, znovu sa vykresľovať nebude, pretože sa použije už ten vykreslený.

Majme výraz: *a * b + (c - d) / (e + f) + (g + h - i)*. Inštrukcie v tvare trojadresného kódu po vyhodnotení syntaktickým analyzátorom sú:

[*,a,b,$0],
[-,c,d,$1],
[+,e,f,$2],
[/,$1,$2,$3],
[+,$0,$3,$4],
[+,g,h,$5],
[-,$5,i,$6],
[+,$4,$6,$7].

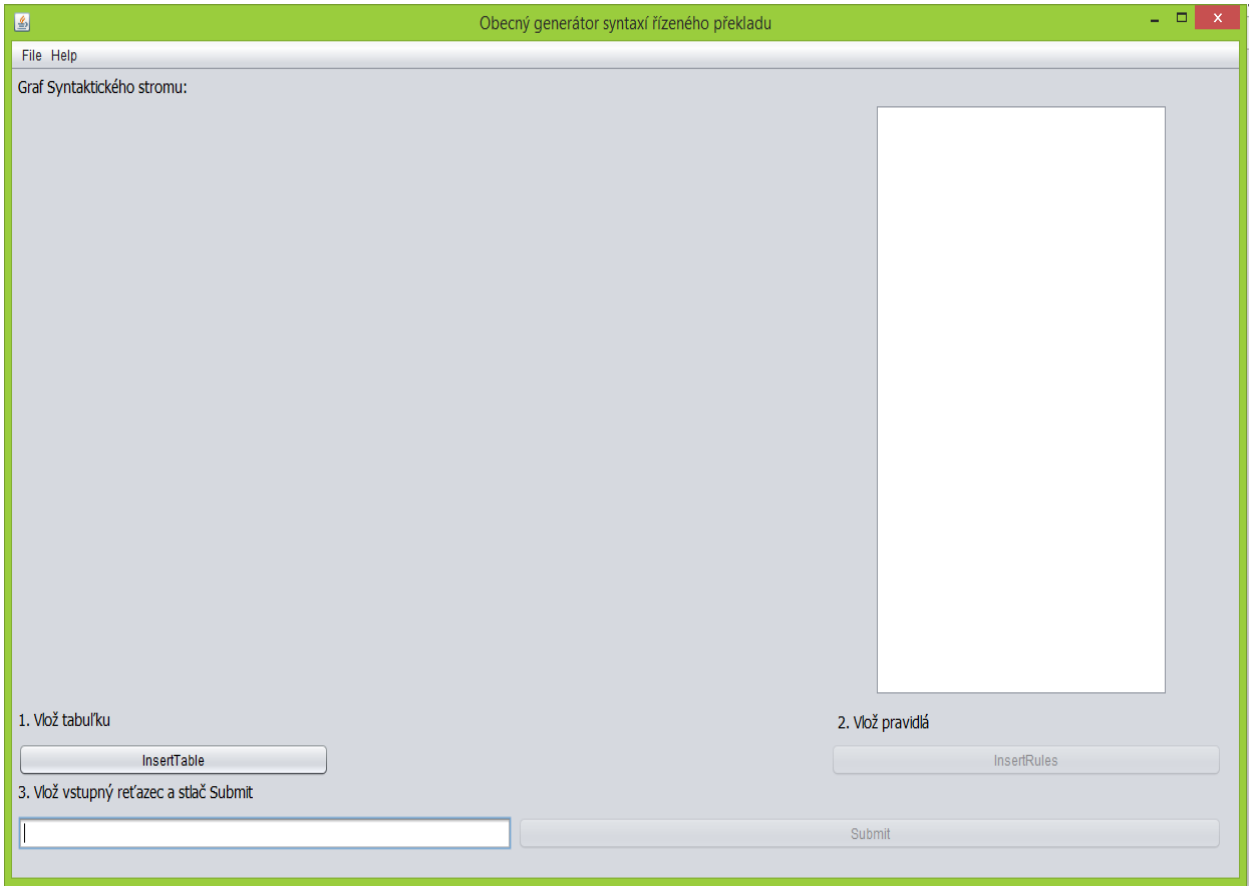
Vykreslený graf syntaktického stromu pre tento výraz je na obrázku 4.2.



Obrázok 4.2 Syntaktický strom

4.5 Manuál

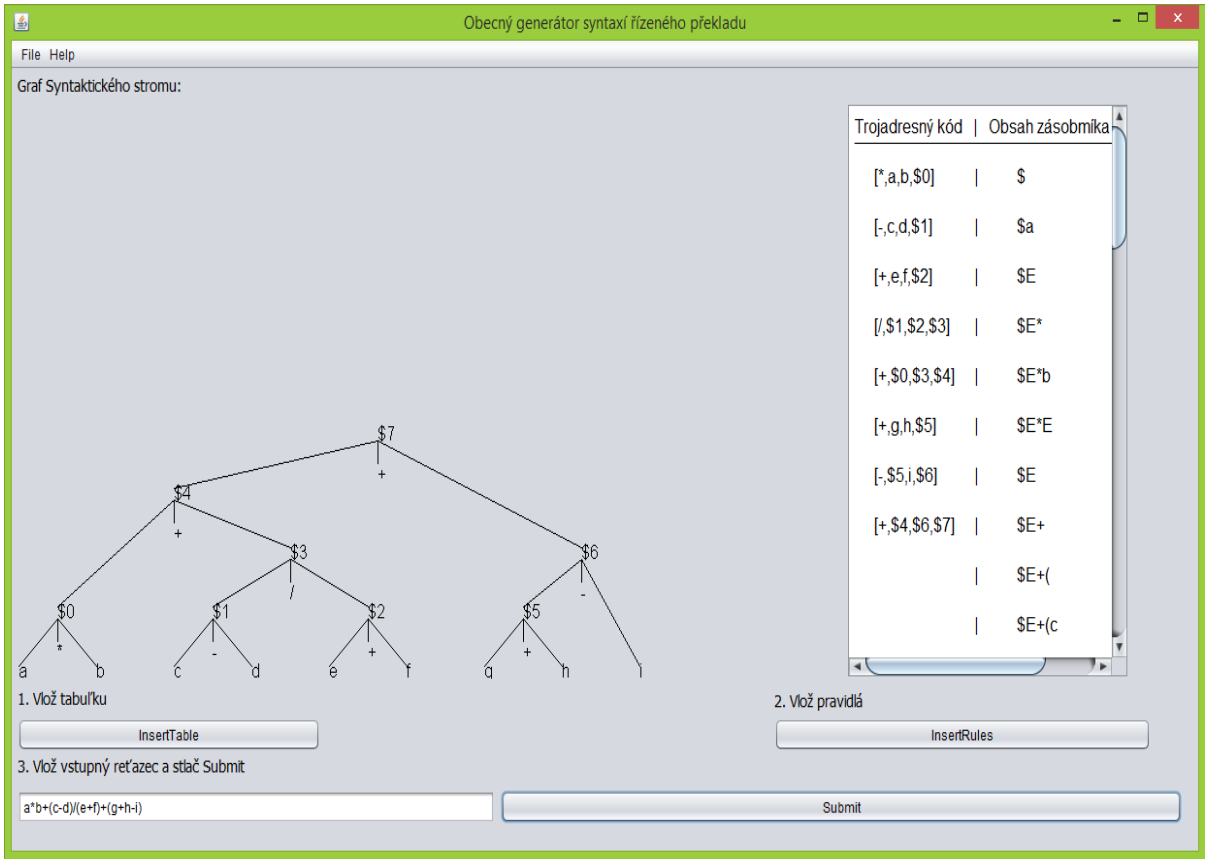
Táto podkapitola je zameraná na používanie celej aplikácie. Celý program je navrhnutý tak, aby bolo jeho používanie a práca s ním jednoduchá a intuitívna. K tomuto účelu sú prispôsobené aj tlačidlá, ktoré sú iba tri. Po spustení aplikácie je aktívne iba jedno *InsertTable*. Toto tlačidlo slúži na vloženie tabuľky symbolov. Po úspešnom vložení tabuľky symbolov sa aktivuje druhé tlačidlo *InsertRules*, ktoré zabezpečuje vloženie redukčných pravidiel pre program. Posledné je tlačidlo *Submit*, ktoré vezme výraz zo vstupného riadku a začne celý proces prekladu. Po tomto procese sa v aplikácii ako výstup zobrazia trojadresné inštrukcie, obsah zásobníka a graf syntaktického stromu.
Na obrázku 4.3 je znázornená aplikácia hneď po spustení. V tejto fáze je aktívne iba prvé tlačidlo.



Obrázok 4.3

Po vložení tabuľky symbolov, pravidiel a vstupného reťazca, program začne tento reťazec vyhodnocovať. Ak je reťazec správny program zareaguje informačným oknom so správou: „Zadaný výraz je správny.“ Následne nakreslí graf syntaktického stromu a vypíše obsah zásobníka a trojadresné inštrukcie. Správny výstup aplikácie je demonštrovaný na obrázku 4.4.

V prípade, že sa v programe vyskytla chyba, program reaguje vypísaním chybovej hlášky, ktorá bližšie špecifikuje vzniknutú chybu. Napríklad v prípade zlého formátu tabuľky symbolov, vie program konkrétne určiť, na ktorom riadku nastala chyba. Ak sa chyba vyskytuje vo vstupnom reťazci a je detegovaná počas prekladu program určí, medzi ktorými symbolmi vstupného reťazca sa chyba vyskytuje.



Obrázok 4.4

Aplikácia obsahuje aj dva menu panely *File* a *Help*. Hlavná úloha panelu *File* je v ukladaní výstupov. Obsahuje tri položky *Save stack content, Save Instructions* a *Save syntax tree to PNG.* Prvá možnosť slúži k uloženiu obsahu zásobníka do vybraného súboru. Po zvolení druhej položky sa uložia inštrukcie v tvare trojadresného kódu do zvoleného súboru. V oboch prípadoch sa uloží textový výstup, ktorý je zobrazený v aplikácií. Tretia a zároveň posledná možnosť je uloženie syntaktického stromu ako obrázok vo formáte PNG. Táto možnosť uloženia odstraňuje problém veľkosti plátna. Druhý panel *Help* obsahuje dve položky. Prvá je rovnako označená ako samotný panel a má funkciu krátkej nápovedy k obsluhe programu. Druhá položka je *About* je stručný popis programu spolu s kontaktom na autora.

Záver

Gramatiky patria neodmysliteľne do teórie formálnych jazykov, pretože nám umožňujú popísať svet okolo nás takým spôsobom, aby mu mohli porozumieť aj počítače. Tie sú našou súčasťou a sme na nich zo dňa na deň viac naviazaný. Preto je pochopiteľné, že by sme chceli komunikovať s počítačmi rovnako ako komunikujeme medzi sebou navzájom. O tento cieľ sa usiluje teória formálnych jazykov, ktorá každým dňom popisuje a analyzuje čím ďalej tým zložitejšie systémy. Týmto sa automaticky rozširujú možnosti a výpočtové schopnosti strojov.

Samotné gramatické systémy, ktoré boli predstavené na začiatku práce sú inšpirované ľudským svetom. Ak si človek nevie s nejakým problémom rady, je pravdepodobné, že tento problém môže byť jednoduchšie vyriešený skupinou ľudí, ktorí navzájom spolupracujú. Pretože, každý človek má svoje vlastné prednosti, znalosti a skúsenosti a aj náhľad na daný problém. Rovnaká myšlienka sa uplatňuje aj v gramatických systémoch, kde je spojených viacero jednoduchých gramatík, ktoré spolu vytvárajú jeden silný gramatický systém.

Cieľom tejto práce bolo navrhnutie a vytvorenie univerzálneho softwarového prostriedku, ktorý by vykonával syntaxou riadený preklad podľa požiadaviek používateľa. Takto vytvorený program by dokázal na základe vloženej tabuľky symbolov a redukčných pravidiel rozhodnúť o tom, či vstupný reťazec je možné na základe týchto pravidiel zredukovať. Pretože človek nie je neomylný môžu sa v zadanom vstupnom reťazci vyskytnúť chyby. Tieto chyby sa môžu vyskytovať v niekoľkých úrovniach. Ak sa jedná o zle napísané slová, sú to lexikálne chyby, s ktorými si poradí lexikálny analyzátor. Ak je v programe použitá nedefinovaná premenná alebo sa jedná o porovnávanie hodnôt rôznych dátových typov dôjde k sémantickej chybe. V tejto práci sa zameriame skôr na chyby, ktoré nastávajú počas syntaktického prekladu a teda syntaktické chyby. Medzi tieto chyby patrí napríklad chýbajúci operátor alebo operand. V prípade správnosti vstupného výrazu sa vytvorí ako výstup postupnosť inštrukcií v podobe post-fixovej notácie alebo trojadresného kódu. Následne na základe inštrukcií tento generátor vykreslí syntaktický strom.

Pretože sa nejedná o bežný prekladač s pevne danými redukčnými pravidlami a tabuľkou symbolov, bolo potrebné vyriešiť najprv vkladanie týchto dvoch elementov do programu. Až následne je možné pristúpiť k samotnému prekladu. Pri návrhu programu som sa riadil štruktúrou prekladača, ktorá pozostáva z lexikálnej, syntaktickej, sémantickej analýzy a generovania kódu. Na záver po dokončení prekladu program oznámi jeho výsledok a na základe trojadresných inštrukcií vykreslí syntaktický strom. Ako prvé je potrebné spracovať tabuľku symbolov od používateľa, keďže tá obsahuje symboly. Jednotlivé symboly sú potrebné vo fáze syntaktickej a sémantickej analýzy. Po spracovaní tejto tabuľky je potrebné ešte pred samotným prekladom spracovať redukčné pravidlá. Tieto pravidlá obsahujúce symboly a neterminály sú nevyhnutné pre syntaktickú analýzu a to zvlášť pri redukcii.

Jadrom samotnej práce bolo vytvorenie syntaktického analyzátora, ktorý spracúva vstupný reťazec znakov. Syntaktická analýza predstavuje proces validácie tohto reťazca vzhľadom k redukčným pravidlám. Syntaktický analyzátor je nástroj, ktorý vykonáva túto syntaktickú analýzu. Pred syntaktickým analyzátorom vystupuje v prekladači lexikálny analyzátor. Ten rozdelí a spracuje vstupný reťazec a vytvorí z neho zoznam tokenov. Samotná práca je zameraná na vygenerovanie syntaxou riadeného prekladu pre zadané výrazy. Preto je tento generátor založený na syntaktickej analýze zdola-nahor. Pri tejto analýze postupuje analyzátor od jednotlivých terminálov, ktoré pomocou vhodných redukčných pravidiel zlúči do jedného neterminálu. Neterminály sú potom rovnakým spôsobom zlučované do ďalších neterminálov. Pri tomto spájaní dochádza ku vzniku derivačného stromu. Toto zlučovanie prebieha dovtedy pokým nie sú zlúčené všetky terminály zo vstupného reťazca a vzniknuté neterminály v jeden koncový neterminál.

V dnešnej dobe existuje mnoho generátorov, ktoré sú schopné vygenerovať na základe gramatiky a pravidiel preklad zdrojového textu. Medzi najpoužívanejšie a najznámejšie generátory patrí YACC (Yet Another Compiler Compiler). Tento generátor je používaný pre svoju kompatibilitu v mnohých Unixových systémoch. Využíva sa k vyprodukovaniu kompletnej syntaktickej analýzy. Pri porovnaní s ním vyzerá môj všeobecný generátor ako menej významný. Avšak môj generátor má jednu významnú funkciu navyše oproti silnejšiemu generátoru YACC. Touto funkciou je vykreslenie syntaktického stromu na základe trojadresných inštrukcií. Vďaka tomuto vykresleniu používateľ jasne vidí ako jednotlivé redukcie prebiehali. Program zobrazí aj celú históriu používania zásobníka a tiež inštrukcie v podobe trojadresného kódu. Používateľ môže jasne vidieť ako tento generátor pri procese prekladu postupoval.

Testovanie celého programu prebiehalo celkovo na dvoch tabuľkách symbolov spolu s dvomi sadami pravidiel. Prvá tabuľka obsahovala aritmetické operátory +, -, *, /, ^, (a). K týmto operátorom sa viazali aj príslušné redukčné pravidlá ako E → E + E, E → E – E, E → E * E, E → (E) a tak ďalej. Druhá tabuľka obsahovala logické operátory -, &, |, (a), kde - reprezentuje negáciu, & reprezentuje logický súčin AND, | vyjadruje logický súčet OR. Pravidlá pre túto druhú tabuľku sú podobné ako v prvom prípade E → E & E, E → E | E. Jediný rozdiel je pri negácií, pretože tá reprezentuje unárny operátor a teda pravidlo má tvar E → - E. Testovaním bolo zistené, že navrhnutý a implementovaný syntaktický analyzátor dokáže spracovať a vyhodnotiť výrazy na základe rôznych tabuliek symbolov a redukčných pravidiel.

Ďalší vývoj tejto aplikácie vidím v rozšírení syntaktického analyzátora o podporu analýzy zhora-nadol. Toto rozšírenie by umožňovalo všeobecnejšie spracovanie zdrojového textu. Pridaním gramatík do syntaktickej analýzy by sa výrazne zvýšila sila tohto generátora. To by malo za následok vytvorenie plnohodnotného všeobecného generátora prekladu.

Literatúra

[1] MEDUNA, A. Automata and languages: theory and applications. London: Springer, 2000. ISBN 1-85233-074-0.

[2] ROZENBERG, Grzegorz and Arto SALOMAA (eds.). Handbook of formal languages. Berlin: Springer-Verlag, c1997. ISBN 3-540-61486-9.

[3] AHO, A. V. Compilers: principles, techniques, & tools. 2nd ed. Boston: Pearson/Addison Wesley, c2007. ISBN 0-321-48681-1.

[4] WWW stránky: SlideShare - LinkedIn Corporation [online] [cit. 2015-11-06]. Dostupné z: http://image.slidesharecdn.com/phasescompiler-141221115333-conversion-gate01/95/phases-of-a-compiler-3-638.jpg?cb=1419162903

[5] MEDUNA Alexander. Elements of Compiler Design. New York: Taylor & Francis Informa plc, 2008. ISBN 978-1-4200-6323-3.

[6] Aho, A. V.: Principles of Compiler Design, Addison-Wesley, Reading, Massachusetts, 1977.

[7] MEDUNA, A. Formal languages and computation: models and their applications. New York: CRC Press, c2014. ISBN 978-1-4665-1345-7.

Zoznam príloh

Príloha A: Obsah CD

- Text bakalárskej práce vo formáte PDF.
- Zdrojový text práce vo formáte ODT (docx).
- Zdrojové súbory aplikácie spolu z manuálom a už spustiteľnú verziu aplikácie v archíve JAR
- Vzorové súbory obsahujúce tabuľky symbolov a redukčné pravidlá