VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NOVÉ VERZE PŘEKLADOVÝCH AUTOMATŮ

NEW VERSIONS OF TRANSLATION AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MIROSLAV BALÁŽ
AUTHOR

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

BRNO 2020

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

22346

Student: **Baláž Miroslav**
Program: Informační technologie
Název: **Nové verze překladových automatů**
New Versions of Translation Automata
Kategorie: Překladače
Zadání:

1. Seznamte se podrobně s různými verzemi překladových automatů dle pokynů vedoucího.
2. Dle pokynů vedoucího zaveďte a studujte nové verze překladových automatů. Formalizujte je.
3. Seznamte se s pokročilými metodami lexikální a syntaktické analýzy. Navrhněte nové metody analýzy, které jsou založeny na automatech z bodu 2.
4. Zkoumejte vlastnosti metod z bodu 3. Porovnejte jejich sílu s klasickými metodami analýzy.
5. Dle návrhů a pokynů vedoucího uvažujte řadu komplikovaných lexikálních a syntaktických struktur. Popište jejich analýzu prostřednictvím modifikovaných metod z bodu 3.
6. Implementujte analýzu popsanou v bodě 5 a testujte ji v rámci stávajících nástrojů dle pokynů vedoucího.
7. Zhodnoťte dosažené výsledky. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education, 2006, ISBN 0-321-48681-1
- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Práca skúma nové verzie prekladových automatov. Skúma existujúcu verziu zásobníkového prevodníku a reguláciu gramatiky. Tieto dva existujúce pojmy spája a v práci je zadefinovaný regulovaný zásobníkový prevodník ako nová forma prekladového automatu. Práca skúma vyjadrovaciu silu regulovanej gramatiky, testuje využitie tejto gramatiky v prevodníku. Pomocou implementovaje štruktúry skúma využitie v prekladoch vstupného kódu.

Abstract

This thesis does research on new version of translation automata. It takes closer look at existing pushdown transcuders and regulated grammar. Combining these two concepts together which creates new regulated pushdown transcuder as a new form of translation automata. Examine computational power of regulated grammar and tests its usage combined with transcuder and test usablity of this new defined regulated transcuder in implementation for translating input source code.

Kľúčové slová

Zásobníkový prevodník, prevodník, regulovaná gramatika, gramatika, regulovaný zásobníkový prevodník, zásobníkový automat, formálne jazyky

Keywords

Pushdown transcuder, transcuder, regulated grammar, grammar, regulated pushdown automata, pushdown automata, formal languages

Citácia

BALÁŽ, Miroslav. *Nové verze překladových automatů.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Nové verze překladových automatů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof. RNDr. Alexandra Meduny CSc. . Dalšie informácie mi poskytli... Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

. . . . . .
Miroslav Baláž
17. mája 2020

Poďakovanie

Ďakujem pánovi profesorovi Medunovi za odborné rady, pozitívny prístup a trpezlivosť počas konzultácii tejto práce a za jeho profesionálny prístup k riešeniu problémov. Ďalej by som chcel poďakovať menovite týmto osobám Bc. Boris Kiška, Patrik Baláž, Jaroslav Filo, Peter Backo, Ľubomír Mikula a Jozef Rebro za psychickú a priateľskú podporu počas celého štúdia a hlavne počas záverečného ročníka. A na záver chcem poďakovať celej rodine, ktorá ma podporovala počas štúdia a dodávala mi energiu pokračovať ďalej.

Obsah

1 Úvod **2**

2 Základné pojmy a definície **5**

3 Zásobníkový prevodník a regulovaná gramatika **7**
3.1 Zásobníkový prevodník 7
3.2 Regulovaná gramatika 10

4 Regulovaný zásobníkový prevodník **13**
4.1 Syntaktická analýza výrazov 14
4.2 Analýza s generovaním výstupného kódu 17

5 Implementácia a testovanie **24**
5.1 Implementacia triedy Scanner 25
5.2 Implementácia prevodníku pre spracovanie výrazov 26
5.3 Implementácia prevodníku pre syntaktickú analýzu 28
5.4 Implementácia regulácie pre prevodník 31
5.5 Testovanie implementácie prekladača 32

6 Záver **38**

Literatúra **41**

Kapitola 1

Úvod

Zásobníkový automat tvorí velký základ pre moderné prekladače. Jeho efektívnosť a jednoduchosť dáva možnosť rozšíreniam, ktoré využíva tento jednoduchý koncept a dáva mu veľmi efektívnu silu pri preklade. Práca skúma využitie zásobníkového automatu v prekladových automat, teda, zásobníkový automat, ktorý je modifikovaný pomocou pridania výstupnej pásky do prekladového automatu. Cieľom práce je skúmať nové formy prekladových automat, formálne ich zadefinovať, porovnať ich vyjadrovaciu silu a skúmať pokročilé formy syntaktickej a lexikálnej analýzy pomocou týchto novo formalizovaných a definovaných automatov. Po zadefinovaní a formalizácii novej verzie prekladového automatu a jeho komponentov, ktoré sú potrebné pre fungovanie prekladu, sú všetky komponenty implementované v jazyku Python. Táto implementácia ukazuje využitie teoreticky zadefinovaných častí a testovanie, ktoré ukončuje celú prácu a ukazuje správnosť fungovania celého prekladového automatu.

V práci sa počíta s určitou znalosťou danej problematiky, preto sú v nej používané definície a pojmy, ktoré sú v danom odvetví, tj. teoretickej informatike, bežne zaužívané a v danej sfére dobre poznané. Avšak, pre úplnú kompletnosť práca ponúka na svojom začiatku rýchle zhrnutie pojmov, ktoré sú v nej použité, zadefinované základné štruktúry a pojmy z oblasti prekladačov, konečných automat, lexikálnej a syntaktickej analýzy. V práci sa počíta s tým, že predom definované pojmy sú pochopené a ich dalšie vysvetlenie hlbšie v práci nebude poskytnuté. Pre tento prípad sú v práci použité referencie na definície pojmov, ktoré je potrebné poznať, ak sa ešte v danej kapitole nevyskytli alebo ak je poznatok určitej definície kľúčový k dalšiemu pochopeniu opisovanej časti práce.

Teoretická časť práce pojednáva o definovaní poznatkov, ktoré sú potrebné pre vytvorenie novej formy prekladového automatu. Na začiatku práce je potrebné definovať zásobníkový prevodník. Ako vyzerá štruktúra zásobníkového prevodníku, čo zásobníkový prevodník predstavuje. Aké sú jeho výhody, využitie a akým spôsobom daný prekladový automat funguje. Táto definícia je veľmi dôležitá pre dalšie pokračovanie v práci, keďže zásobníkový prevodník slúži ako základná štruktúra, ktorá je v neskoršej časti práce vylepšená pomocou pokročilej formy syntaktickej analýzy a to pridaním regulovanej gramatiky. Pre lepšie pochopenie tohto typu prekladového automatu su priložené príklady, ktoré ukazujú, ako daný prevodník pracuje. Tieto príklady slúžia pre ilustráciu práce prevodníku ako v implementácii, tak aj ako pomôcka pre pochopenie jeho celkovej definície a definícii jeho časti. Definícia prevodníku nepredstavuje len definíciu jeho časti a definícii jeho štruktúry ako takej, ale je potrebné pre správne pochopenie tohto automatu zadefinovať jeho spôsob práce. Preto štruktúra a postup definície je rozdelený do časti. Prvá časť definície predstavuj oboznámenie s formálnym zápisom prevodníku, čo dané znaky vo formálnom zápise predstavujú a čo

všetko je v nich zahrnuté. Dalšou časťou definície je formálny popis jeho prekladu. To znamená, akým spôsobom s danými komponentami pracuje, ako prechádza z konfigurácie do konfigurácie, ako sa konfigurácia prevodníku líši od konfigurácie od bežného zásobníkového automatu a akým spôsobom prechádza medzi stavmi prekladu.

V tejto časti je formálne zadefinovaná prekladová gramatika, ktorá bude použitá pri implementácii a od ktorej je implementácia odvodená. Spolu s definícou prekladovej gramatiky, je zadefinovaná aj gramatika, ktorá bude použitá pri spracovaní výrazov a spolu s týmito gramatikami je vytvorený výstupný jazyk celého prekladu. Spolu s týmito definíciami sú vytvorené príklady, ktoré ukazujú, akým spôsobom daná prekladová gramatika prijíma vstupný reťazec a akým spôsobom je tento vstupný jazyk preložený do výstupnej podoby. Na príkladoch je ukázaný aj spôsob, akým bude pracovať regulácia v preklade a ukáže, akým spôsobom budú pravidlá v gramatike aplikované do programu.

Druhou veľmi podstatnou častou teoretickej časti je definícia Regulovanej gramatiky. Regulovaná gramatika rozširuje základnú definíciu gramatiky. Pomocou príkladov spolu s definíciou je zadefinovaná gramatika, príklady ukazujú, ako gramatika samotná pracuje a čo vďaka nej je možné dokázať. Ako gramatika generuje Výsledný jazyk, ktorý pomocou pravidiel gramatika generuje. Ukážka práce gramatiky je znázornená pre lepšie pochopenie aj graficky, tj. derivačným stromom. V práci nie sú použité komplikované príklady. Snahou príkladov je skôr ukázať jasne a jednoznačne prácu nad jednoduchou gramatikou tak, aby nebolo zastreté pozadie práce. Príklad 3.4 sa snaží ukázať, že pridanie regulácie do gramatiky môže zvýšiť jej vyjadrovaciu schopnosť. Gramatika pomocou regulácie je schopná zmeniť generovaný jazyk a je možné prijímať reťazce, ktoré nepatria do jej generovaného jazyka bez použitia regulácie.

Ak sa tieto dva princípy spoja, vzniká nová verzia prekladového automatu, regulovaný prevodník. Tento prevodník obsahuje všetky základne komponenty ako už zadefinovaný prevodník,ale jeho prekladovú gramatiku dopĺňa regulácia. Prevodník už nielen že na konci svôjho prekladu vygeneruje výstupný jazyk po prijatí vstupného reťazca, ale je potrebné, aby skontroloval, či použitá postupnosť pravidiel z gramatiky je zhodná so vzorom, ktorý je použitý pri regulácii. Definícia regulovaného prevodníku obsahuje formálny zápis a jeho vysvetlenie. Ďalej popisuje spôsob akým je preklad pomocou regulovaného prevodníku uskutočnený a aké zmeny sú pri tomto preklade oproti bežnému prevodníku v jednotlivých krokoch prekladu. Zásadný rozdiel oproti bežnému prevodníku je schopnosť zapamätania si použitých pravidiel pre preklad, ktoré budú na konci prekladu použité pre kontrolu. Neskôr v impelementácii bude popísaný spôsob, akým je regulácia v prevodníku uložená a ako prebieha kontrola postupnosti použitých pravidiel. A aj to, že na preklad sú použité práve dva regulované prevodníky, pričom jeden slúži na preklad výrazov, ktorý obsahuje samostatnú reguláciu a druhý, ktorý riadi celý preklad a obsahuje taktiež samostatnú reguláciu. Regulácia, ktorá je použitá v prevodníku nad jeho prekladovou gramatikou môže výrazne zmeniť vstupný jazyk, ktorý môže byť prijatý. Regulácia môže obmedzit prijímané vstupné programy bez akejkoľvek potreby zmeny gramatiky alebo zvýšiť jeho vyjadrovaciu silu bez zásahu do prekladovej gramatiky.

Druhá polovica práce pojednáva o implementácii takto formálne zadefinovaného prekladového automatu. Impelementácia využíva jazyk Python vo verzii 3. Tento jazyk je použitý v implementácii, kvôli jeho vlastnostiam ako napríklad jeho veľmi dobrá produktivita, vstavané dátové typy, ktoré veľmi efektívne fungujú pri implementácii práve automatu. Napríklad hlavným pozitívom tohto jazyku v rámci implementácie prevodníku vidím vo využití vstavaných funkcii a typov ako napríklad trieda list, ktorá ponúka vlastnosti zásobníku. Implementácia je rozdelená do niekoľkých častí, ktoré opisujú implementáciu jednotlivých

komponent celého prekladu. Začína sa implementáciou scanneru, pokračuje implementáciou analýzy výrazov, kde využíva už formálnu definíciu regulovaného prevodníku a implementuje tento prekladový automat do prekladu výrazov z infixovej do postfixovej podoby. Následne je popísaná implementácia syntaktickej analýzy, ktorá je taktiež priamo implementáciou formálne definovaného regulovaného prevodníku. Poslednou častou je vysvetlenie implementácie regulácie nad prekladovými gramatikami oboch prevodníkov. Pri tejto implementácii bol využitý vstavaný modul Pythonu a to regex, ktorý je využitý pri kontrole správnej postupnosti použitých pravidiel. Ukazuje, akým spôsobom musel byť formálny tvar zápisu regulácie upravený, aby jeho zápis bol možný použiť pri kontrole regulácii regexom ale aj ukazuje akým spôsobom je tento tvar prevádzaný na regex vzor.

K implementačnej časti patrí aj testovanie, či daný spôsob, akým bol prekladový automat implementovaný, je správny a či splňuje požiadavky, ktoré sú od neho vyžadované. Celý spôsob testovania prebieha v dvoch etapách. Prvou etapou testovanej časti je zisťovanie, či naprogramovaná komponenta zvládne detekovať chybné zápisy vstupného programu, ak nie, daná časť musí byť opravená a v druhej etape je otestované, či implementácia dokáže správne vykonať svoje operácie nad správne zapísaným vstupným programom. Testované sú najskôr všetky komponenty zvlášť a na konci je potrebné tieto komponenty spojiť v jeden celok a otestovať, či ako celok dokážu spolupracovať a bezchybne vykonávať preklad vstupného kódu na výstupný.

Na záver sú v práci zhrnuté získané poznatky. Čím môžu dané poznatky prispieť a zlepšíť aktuálny preklad, ktorý využíva iné formy prekladových automatov. Či vôbec má zmysel takto vytvorený prekladový automat používať a pokračovať v práci na takto definovanom prekladovom automate. Záver taktiež otvorí a načrtne možné pokračovanie tejto práce ako diplomovej práce a akým smerom je možné sa uberať pri tomto pokračovaní.

Obdobná problematika, ktorá bola študovaná v tejto práci je skúmaná v niektorých literatúrach, z ktorých práca vychádza a to napríklad: Elements of Compiler Design[2], Scattered Context Grammars and their Applications[3], Grammars with Context Conditions and Their Applications[4], Automata and Languages[1] a Handbook of Formal Languages[5]

Kapitola 2

Základné pojmy a definície

Na začiatok práce je dôležité zaviesť definíciu základný pojmov a definíc, s ktorými sa v práci pracuje a sú potrebné pre celkové pochopenie práce. Tieto pojmy a definície budú použité pre zadefinovanie zložitejších štruktúr a definíc.

Definícia 2.1. Abeceda
Abeceda je konečná, neprázdna množina elementov, ktoré sa nazývajú *symboly*. Abeceda je značená $\mathbf{\Sigma}$, potom $\mathbf{\Sigma} = \{\mathbf{f}, \mathbf{q}, \mathbf{0}, \mathbf{12}\}$

Definícia 2.2. Reťazec
Nech Σ je abeceda, potom:

1. ε (prázdny reťazec) je reťazec nad abecedou
2. Nech x je reťazec nad Σ a $c \in \Sigma$, potom xc je reťazcom nad abecedou Σ

Definícia 2.3. Jazyk
Nech Σ^* značí množinu všetkých reťazcov nad Σ. Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad Σ.
Poznámka: $\Sigma^+ = \Sigma^* - \{\varepsilon\}$

Definícia 2.4. Zásobníkový automat
Zásobníkový automat (ZA) je sedmica: $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- Q je konečná množina všetkých stavov
- Σ je vstupná abeceda
- Γ je zásobníková abeceda
- R je konečná množina pravidiel v tvare $Apa \twoheadrightarrow wq$,
  kde $A \in \Gamma, p \vee q \in Q, a \in \ \Sigma \cup \{\varepsilon\}, w \in \Gamma^*$
- $s \in Q$ je počiatočný stav
- $S \in \Gamma$ je počiatočný symbol na zásobníku
- $F \subseteq Q$ je množina koncových stavov

Definícia 2.5. Príjmaný jazyk Jazyk prijímaný zásobníkovým automatom M, L(M) je definovaný ako:

1. Jazyk prijímaný ZA M **prechodom do koncového stavu**, značený ako $L(M)_f$, je definovaný:
   $L(M)_f = \{w : w \in \Sigma^*, Ssw \Vdash^* zf, z \in \Gamma^*, f \in Q\}$

2. Jazyk prijímaný ZA M **vyprázdnením zásobníku**, značený ako $L(M)_\varepsilon$, je definovaný:
   $L(M)_\varepsilon = \{w : w \in \Sigma^*, Ssw \Vdash^* zf, z = \varepsilon, f \in Q\}$

3. Jazyk prijímaný ZA M **prechodom do koncového stavu a vyprázdnením zásobníku**, značený ako $L(M)_{f\varepsilon}$, je definovaný:
   $L(M) = \{w : w \in \Sigma^*, Ssw \Vdash^* zf, z = \varepsilon, f \in F\}$

Definícia 2.6. Derivačný krok
Proces, pri ktorom je vstupný reťazec zmenený použitím práve jedneho pravidla.
Nech $G = (N, T, P, S)$ je gramatika, $u, v \in (N \cup T)^*$, pravidlo $p = A \twoheadrightarrow x \in P$.

Potom uAv priamo derivuje uxv za použitia pravida p, zapísané ako $uAv \Rightarrow uxv[p]$, zjednodušene $uAv \Rightarrow uxv$. Inak povedané, G spraví derivačný krok z uAv do uxv.

Definícia 2.7. Najľavejšia derivácia
Počas najľavejšieho derivačného kroku je prepísaný najľavejší neterminál.
Nech $G = (N, T, P, S)$ je gramatika, $u, v \in (N \cup T)^*$, pravidlo $p_1 = A \twoheadrightarrow x \in P, p_2 = B \twoheadrightarrow w$.

Potom, $uAvB \Rightarrow_{LM} uxvB[p_1]$, teda G spraví derivačný krok z $uAvB$ do $uxvB$ pomocou najľavejšej derivácie použitím pravidla p_1.

Definícia 2.8. Najpravejšia derivácia Počas najpravejšieho derivačného kroku je prepísaný najpravejší neterminál.
Nech $G = (N, T, P, S)$ je gramatika, $u, v \in (N \cup T)^*$, pravidlo $p_1 = A \twoheadrightarrow x \in P, p_2 = B \twoheadrightarrow w$.

Potom, $uAvB \Rightarrow_{RM} uAvw[p_1]$, teda G spraví derivačný krok z $uAvB$ do $uAvw$ pomocou najpravejšej derivácie použitím pravidla p_2.

Definícia 2.9. Derivačný strom
Derivačný strom predstavuje grafickú reprezentáciu použitých pravidiel v derivačných krokoch. Jeho štruktúra predstavuje strom, ktorého koreňom je počiatočný symbol z gramatiky. V uzloch stromu su reprezentované neterminály a terminály podľa použitých pravidiel z daného uzla.

Kapitola 3

Zásobníkový prevodník a regulovaná gramatika

Následujúca kapitola definuje základnú definíciu zásobníkového prevodníku a regulovanej gramatiky.Tieto definície budú slúžiť ako základ pre vytvorenie novej verzie prekladového automatu. Spája tieto dve definície a vytvára novú verziu prekladového automatu. Ďalej skúma jeho vlastnosti a definuje preklad s použitím novo zadefinovanej štruktúry.

3.1 Zásobníkový prevodník

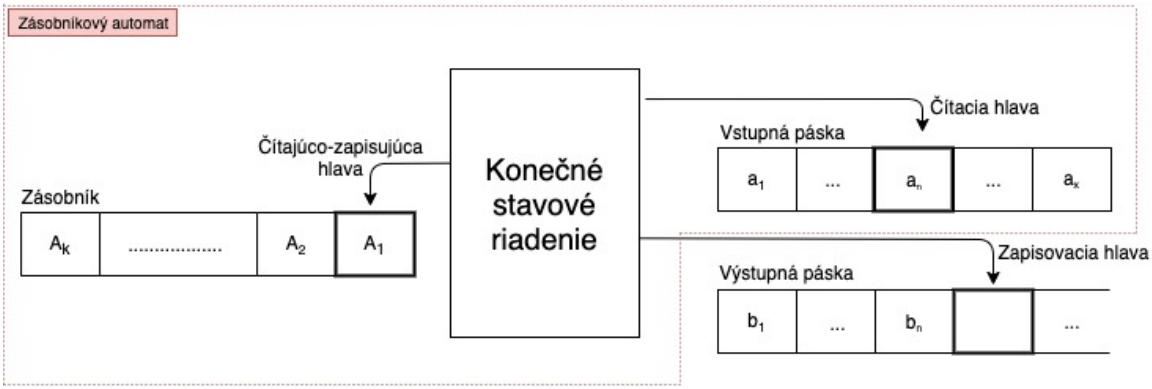
Definícia 3.1. Zásobníkový prevodník je šestica[1]:

$$M = (Q, \Sigma, R, s, S, F)$$

kde

- Q je množina všetkých stavov
- Σ je abeceda symbolov, pre ktorú platí, že $\Sigma = \Sigma_O \cup \Sigma_I \cup \Sigma_P D$, kde Σ_O predstavuje výstupnú abecedu symbolov, Σ_I predstavuje vstupnú abecedu symbolo a $\Sigma_P D$ predstavuje zásobníkovú abecedu obsahujúcu S.
- R je množina všetkých pravidiel.
- s predstavuje štartovací symbol, $s \in Q$
- S značí štartovací symbol na zásobníku
- F je konečná množina koncových stavov, $F \in Q$

Definícia zásobníkového prevodníku vychádza z už zadefinovaného zásobníkového automatu. Pridaním zapisovacej pásky je automat schopný nielen vstupný reťazec prijať, ale môže priamo generovať výstupný kód. Tým dostávame automat, ktorý má jednu zapisovaciu hlavu na výstupnej páske, jednu čítaciu hlavu na vstupnej páske a zapisovaco-čítajúcu hlavu, ktorá číta symbol na vrchole zásobníka a podľa pravidiel symboly na zásobníku prepisuje, maže alebo pridáva.
Prevodník začína s prevodom v stave, kedy jeho zapisovacia hlava je na začiatku výstupnej pásky, čítacia hlava je nastavena na prvý symbol zo vstupnej pásky a zapisovaco-čítajúca hlava ukazuje na štartovací symbol na zásobníku. Následné kroky, ktoré prevodník



Obr. 3.1: Grafické znázornenie zásobníkového prevodníku s vyznačenou častou zásobníkového automatu, z ktorej vychádza

je schopný spraviť určujú pravidlá z množiny R. Množina R obsauje pravidlá, ktoré sú v tvare:

$$Apc \twoheadrightarrow wq$$

kde A a $w \in \Sigma_{PD}$, p a $q \in Q$, $c \in \Sigma_I$
Pravidlá, ktoré je možné použit pre aktuálny stav prevodníku, závisia od jeho aktuálnej konfigurácii.

Definícia 3.2. Konfigurácia zásobníkového prevodníku
Konfiguracia M (zásobníkového prevodníku M) je slovo χ vo forme:

$$\chi = \chi_i / y$$

kde χ_i je konfigurácia zásobníkového prevodníku, pričom "/" $\notin \Sigma$ a $y \in \Sigma_O^*$

Konfigurácia predstavuje aktuálny stav všetkých častí prevodníku. Teda, χ_i predstavuje obsah vrcholu zásobníkovej pásky, symbol na ktorý ukazuje čítacia hlava na vstupnej páske, aktuálny stav prevodníku a obsah na výstupnej páske. Pričom znak "/" predstavuje oddelenie výstupnej pásky od ostatných časti prevodníku.

Definícia 3.3. Prechod medzi konfiguráciami
Prevodník pri použití jedného alebo viacerých pravidiel spraví výpočetný krok, ktorý sa nazýva prechod.

Nech $M = (Q, \Sigma, R, s, S, F)$ je zásobníkový prevodník. Potom:

$$\chi = \chi_i / y \text{ a } \chi' = \chi_i' / yv$$

su dve rôzne konfigurácie M.
Nech $Apa \twoheadrightarrow upv \in R$ je pravidlo z množiny R.

M spraví prechod z $\chi \Vdash \chi'[Apa \twoheadrightarrow upv]$, inak zapisané:

$$\chi \Vdash \chi'[r]$$

M spraví nula prechodov (pohybov) zo χ do χ pomocou ε prechodu, inak zapísane:

$$\chi \Vdash \chi[\varepsilon]$$

Ak v M existuje sekvencia konfigurácií $\chi_0, \chi_1, \chi_2 ... \chi_n$, kde $n \geq 1$, takých, že $\chi_{i-1} \Vdash \chi_i[r_i]$, kde $r_i \in R$, pre $i = 1...n$, tak M spraví n prechodov z χ_0 do χ_n podľa $[p_1...p_n]$, inak zapísane:

$$\chi_0 \Vdash^n \chi_n[p_1...p_n]$$

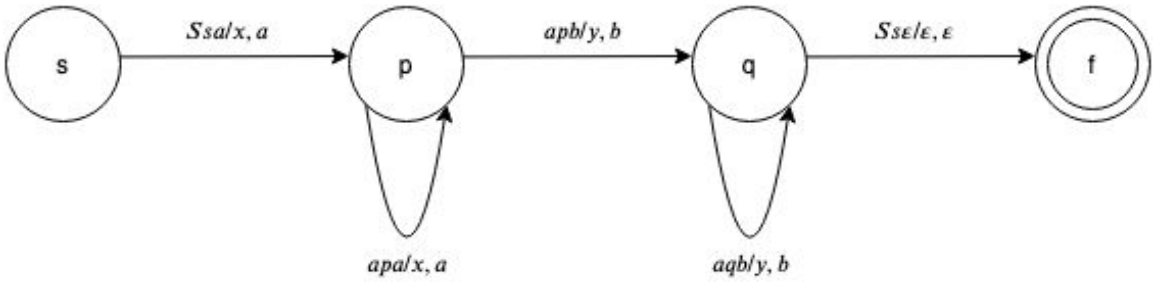
Ak existuje $n \geq 0$, tak $\chi_0 \Vdash^n \chi_n$, tak zápis sa zmení na:

$$\chi_0 \Vdash^* \chi_n[p_1...p_n]$$

Príklad 3.1. Nech $M = (Q, \Sigma, R, s, S, F)$ je zásobníkový prevodník, kde:

$$Q = \{s, q, p, f\}, \Sigma_I = \{a, b\}, \Sigma_O = \{x, y\}, \Sigma_{PD} = \{S, a\}, \mathrm{R} = \left\{ \begin{matrix} 1.Ssa \twoheadrightarrow Sap/x & 2.apa \twoheadrightarrow aap/x \\ 3.apb \twoheadrightarrow q/y & 4.aqb \twoheadrightarrow q/y \\ 5.Sq \twoheadrightarrow f/\varepsilon & \end{matrix} \right\}$$

$F = \{f\}$



Obr. 3.2: Zásobníkový prevodník M

Definovaný prevodník M môže byť znázornený grafom na ktorom su jednotlivé stavy znázornené uzlom,koncové stavy sú zakrúžkované a pravidlá orientovanými hranami v grafe, ktore maju tvar:

[Symbol na zasobníku]<stav>[vstupný symbol]/[zapísaný symbol], [prijatý symbol]

Prevodnik M prijme a preloží vstupný reťazec ab na reťazec xy. Na začiatku prekladu je konfigurácia prevodníku slovo $\chi_0 = Ss/\varepsilon$. Z tejto konfigurácie postupnými prechodmi preloží vstupný reťazec nasledovným spôsobom:

$$\chi_0 = Ss/\varepsilon \Vdash \chi_1 = Sapb/x[1.] \Vdash \chi_2 = Sq\varepsilon/xy[3.] \Vdash \chi_3 = f/xy[5.]$$

Inak zapísané v skrátenom tvare ako prechod:

$$\chi_0 \Vdash^4 \chi_3[1235]$$

Definícia 3.4. Preklad definovaný zásobníkovým prevodníkom
Nech $M = (Q, \Sigma, R, s, S, F)$ je zásobníkový prevodník. Preklad, ktorý je definovaný týmto prevodníkom T(M) vzhľadom na vstupný jazyk $L_I(M)$ je definovaný:

$$T(M) = \{(x, y) : x \in \Sigma_I^*, y \in \Sigma_O^*, M \text{ preloží x na y podľa } r \in R\}$$

Vstupný reťazec, x, vychádza z jazyka, ktorý je definovaný ako vstupný jazyk prekladu $L_I(M)$:

$$L_I(M) = \{x : (x, y) \in T(M) \text{ pre nejake } y \in \Sigma_O^*\}$$

Výstupný reťazec, y, vychádza z jazyka, ktorý je definovaný ako výstupný jazyk prekladu $L_O(M)$:

$$L_O(M) = \{y : (x, y) \in T(M) \text{ pre nejaké } x \in \Sigma_I^*\}$$

3.2 Regulovaná gramatika

Pre definovanie regulovanej gramatiky je potrebné najskôr zadefinovať gramatiku samotnú. Gramatika predstavuje formálny základ jazyka. Jazyk je definovaný pravidlami gramatiky, ktoré predpísaným spôsobom generujú výsledné reťazce.

Definícia 3.5. Gramatika
Gramatika G je štvorica $G = (N, T, P, S)$, kde:

- N je abeceda neterminálov
- T je abeceda terminálov, pričom $N \cap T = \varnothing$
- P je konečná množina pravidiel v tvare $A \twoheadrightarrow x$, kde $A \in N, x \in (N \cup T)^*$
  - Pravidla v tvare $A \twoheadrightarrow x$, značí, že znak A je prepísaný na znak x.
  - Pravidlo v tvare $A \twoheadrightarrow \varepsilon$ je nazývané ε-pravidlo
- S je počiatočný neterminál, pričom $S \in N$

Príklad 3.2. Uvažujme jednoduchú gramatiku $G = (N, T, P, S)$, kde:
$N = \{A, C, S\}$, $T = \{a, c, b\}$ P $= \begin{Bmatrix} 1 : S \twoheadrightarrow AC & 2 : A \twoheadrightarrow aAb & 3 : A \twoheadrightarrow ab \\ 4 : C \twoheadrightarrow Cc & 5 : C \twoheadrightarrow c & \end{Bmatrix}$

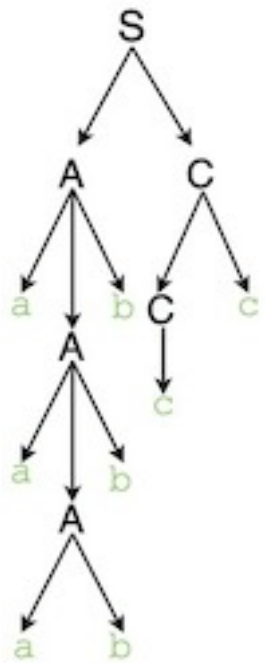
Gramatika obsahuje tri neterminály, konkrétne A, C a S, pričom neterminál S je zároveň štartovacím symbolom. Ďalej tri terminály a, c a b a sadu pravidiel v množine P. Takto vytvorená gramatika generuje reťazec $aaabbbcc$ týmto spôsobom:

$$\begin{aligned} S &\Rightarrow_{LM} AC && [1.] \\ &\Rightarrow_{LM} aAbC && [2.] \\ &\Rightarrow_{LM} aaAbbC && [2.] \\ &\Rightarrow_{LM} aaabbbC && [3.] \\ &\Rightarrow_{LM} aaabbbCc && [4.] \\ &\Rightarrow_{LM} aaabbbcc && [5.] \end{aligned}$$

Na základe stanovených pravidiel gramatiky G, je možné pomocou regulárneho výrazu definovať jazyk, z ktorého sú výsledné generované reťazce danej gramatiky G. Generovaný jazyk gramatikou G je možné formálne zapísať: $\mathbf{L(G) = \{a^n b^n c^m; n, m \geq 1\}}$.

Ako je ukázané v príklade 3.2 generovaný jazky je definovaný pravidlami gramatiky. To znamená, že daný jazyk, ktoré generuje nie je možné modifikovať žiadnym iným spôsobom, len zmenou gramatický pravidiel.

Pridaním regulácie $\mathbf{\Xi}$, tj. kontrolným reťazcom použitých gramatických pravidiel, je možné výsledný jazyk regulovať, pričom nie je potrebný zásah do gramatických pravidiel. Kontrolný reťazec (regulácia) upresňuje poradie použitých pravidiel v jednotlivých krokoch derivácie. Spôsob regulácie je ukázaný v nasledovnom príklade 3.3, ktorý používa gramatiku definovanú v príklade 3.2.



Obr. 3.3: Derivačný strom generujúci reťazec aaabbbcc pomocou najľavejšej derivácie

Príklad 3.3. Nech gramatika G je rovnaká ako v príklade 3.2
Jazyk generovaný gramatikou G bez regulácie má tvar $L(G) = \{a^n b^n c^m; m, n \geq 1\}$ a generuje reťazec *aaabbbcc*.
Ak je ku gramatike pridaný kontrolný reťazec $\Xi = \{1\}\{2\}^*\{35\}$, tak gramatika G negeneruje reťazec *aaabbbcc* a jej generovaný jazyk sa zmení podľa danej regulácie. Gramatika G s reguláciou Ξ generuje teraz reťazec *aabbc*:

$$
\begin{aligned}
S &\Rightarrow AC && [1.]\\
&\Rightarrow aAbC && [2.]\\
&\Rightarrow aaAbbC && [2.]\\
&\Rightarrow aaabbbC && [3.]\\
&\Rightarrow aaabbbc && [5.]
\end{aligned}
$$

Generovaný jazyk gramatikou G s použitím regulácie Ξ je: $L(G, \Xi) = \{a^n b^n c; n \geq 1\}$.

Ako je vidieť podĺa príkladu 3.3 pomocou regulácie je možné generovaný jazyk gramatikou upraviť. Buď je možné gramatiku obmedziť podľa potreby tak, aby generovali nami požadované reťazce alebo je možné pomocou vhodnej regulácie zvýšiť vyjadrovaciu silu danej gramatiky.

Príklad 3.3 ukazuje ako je možné pomocou regulácie obmedzit gramatiku na príjimanie reťazcov len podľa potreby aktuálnej situácie. Další príklad chce ukázať, že vďaka regulácii je možné rozšíriť vyjadrovaciu schopnosť definovanej gramatiky, kedy pred reguláciou gramatika G generuje bezkontextový jazyk, $L = \{a^n b^n c^m; m, n \geq 1\}$. Vhodnou reguláciou

je možné tento generovaný jazyk upraviť tak, aby generoval určitú kontextovú gramatiku. Tento jav ukazuje další príklad.

Príklad 3.4. Nech gramatika G je rovnaká ako v príklade 3.2
Jazyk generovaný gramatikou G bez regulácie má tvar $L(G) = \{a^n b^n c^m; m, n \geq 1\}$ a generuje reťazec *aaabbbcc*.
Ak je ku gramatike pridaný kontrolný reťazec $\Xi = \{1\}\{24\}^*\{35\}$, tak gramatika G negeneruje reťazec *aaabbbcc* a jej generovaný jazyk sa zmení podľa danej regulácie. Gramatika G s reguláciou Ξ generuje teraz reťazec *aabbcc*:

$$
\begin{aligned}
S &\Rightarrow AC && [1.]\\
&\Rightarrow aAbC && [2.]\\
&\Rightarrow aAbCc && [4.]\\
&\Rightarrow aabbC && [3.]\\
&\Rightarrow aabbcc && [5.]
\end{aligned}
$$

Generovaný jazyk gramatikou G s použitím regulácie Ξ je: $L(G, \Xi) = \{a^n b^n c^n; n \geq 1\}$. Tento generovany jazyk patri k do skupiny kontextových jazykov. Týmto príkladom sa podarilo ukázať, že vhodnou reguláciou sme schopný rozšíriť vyjadrovaciu silu gramatiku a docieliť prijatie reťazca, ktorý patrí do inej jazykovej skupiny ako je gramatika sama o sebe schopná prijať.

Kapitola 4

Regulovaný zásobníkový prevodník

Kapitola vytvára kompletnú definícu regulovaného zásobníkového prevodníku. Využíva definície z kapitoly 3 ktoré sú základom pre vytvorenie novej štruktúry. Ďalej pojednáva o syntaktickej analýze, ktorá využíva novo definovaný prevodník a ukážku a spôsob implementácie jeho prekladu na príkladoch.

Definícia 4.1. Regulovaný zásobníkový prevodník $M = (Q, \Sigma, R, s, F, \Xi)$, kde:

- Q je množina všetkých stavov
- Σ je abeceda symbolov, pre ktorú platí $\Sigma = \Sigma_O \cup \Sigma_I \cup \Sigma_P D$, kde
  - Σ_O predstavuje výstupnú abecedu symbolov
  - Σ_I predstavuje vstupnú abecedu symbolov
  - $\Sigma_P D$ predstavuje zásobníkovú abecedu obsahujúcu S
- R je množina všetkých pravidiel.
- s predstavuje štartovací symbol, $s \in Q$
- S značí štartovací symbol na zásobníku
- F je konečná množina koncových stavov, $F \in Q$
- Ξ značí kontrolný reťazeč pre použité pravidlá

Nech Ψ predstavuje abecedu označení(z anglického 'label') pravidiel. Potom, každé pravidlo v tvare $Apa \twoheadrightarrow wq$ ma svoje unikátne značenie p, $p \in \Psi$. Výsledný tvar označení pravidiel bude vo forme:

$$p.Apa \twoheadrightarrow wq$$

Definícia 4.2. Konfigurácia regulovaného zásobníkového prevodníku

Konfigurácia sa nemení vzhľadom na regulovaný zásobník, ale pre úplnosť je potrebné ju zadefinovať.
Konfigurácia M je slovo Ξ vo forme:

$$\chi = \chi_i / y$$

kde χ_i je konfigurácia prevodníku, pričom "/" $\notin \Sigma$ a $y \in \Sigma_O^*$

Definícia 4.3. Prechod medzi konfiguráciami

Nech $M = (Q, \Sigma, R, s, F)$ je prevodník, potom:

$$\chi = \chi_I/y \text{ a } \chi' = \chi'_I/yv$$

su dve rôzne konfigurácie M. Nech $p.Apa \twoheadrightarrow upv \in R$ je pravidlo z R
M spraví prechod $\chi \vdash \chi_I[p.Aqa \twoheadrightarrow wpv]$, inak zapisané:

$$\chi \Vdash \chi'[p]$$

Pričom $[p]$ v tomto prípade nepredstavuje pravidlo z R, ale predstavuje označenie $p \in \Psi$.

Nech χ je konfiguráciou M, potom M spraví 0 prechodov z χ do χ pomocou ε, teda:

$$\chi \Vdash \chi[\varepsilon]$$

Ak v M existuje sekvencia konfigurácii $\chi_0, \chi_1, \chi_2 ... \chi_n, n \geq 1$, takých, že $\chi_{i-1} \Vdash \chi_i[p_i]$, kde $p_i \in \Psi$, pre $i = 1...n$, tak M spraví n prechodov z χ_0 *do* χ_n podľa $[p_1...p_n]$, zapísané:

$$\chi_0 \Vdash^n \chi_n[p_1..p_n]$$

Ak existuje $n \geq 0$, $\chi_0 \vdash^n \chi_n\ [p_1...p_n]$, tak zapisujeme:

$$\chi_0 \Vdash^* \chi_n\ [p_1...p_n]$$

Definícia 4.4. Preklad definovaný regulovaným zásobníkovým prevodníkom

Nech Ξ je kontrolným jazykom nad Ψ, $\Xi \subseteq \Psi^*$.
Vsupný jazyk, $L_I(M)$ vzhľadom na $T(M)$, preklad definovaný M, je:

$$T(M) = \{(x, y) : x \in \Sigma_i^*, y \in \Sigma_o^*, \text{ M preloží x na y podľa } p \in \Xi\}$$
$$L_I(M) = \{x : (x, y) \in T(M) \text{ pre nejaké } y \in \Sigma_o^*\}$$
$$L_o(M) = \{y : (x, y) \in T(M) \text{ pre nejaké } x \in \Sigma_I^*\}$$

4.1 Syntaktická analýza výrazov

Využitím zásobníkového prevodníku pri syntaktickej analýze vzniká možnosť nielen vstupný reťazec príjimať, ako tomu je pri využití zásobníkových automatov, ale je možné priamo generovať výstupný reťazec, vďaka pridaniu zapisovacej pásky.
Pridaním kontrolného jazyku Ξ nad abecedou Ψ je možné rozšíriť vyjadrovaciu schopnosť prevodníku, kedy pomocou regulácie je rozšírená množina príjmaných reťazcov alebo je možné pomocou regulácie obmedziť príjmaný reťazeč podľa potreby.

Nech G_{expr} je jednoduchá prekladová gramatika s desiatimi pravidlami:

1. $< E > \twoheadrightarrow < T >< E' >\ |\ < T >< E' >$

2. $< E' > \twoheadrightarrow + < T >< E' >\ |\ < T >< E' > +$

3. $<E'> \twoheadrightarrow \varepsilon \mid \varepsilon$

4. $<T> \twoheadrightarrow <F><T'> \mid <F><T'>$

5. $<T'> \twoheadrightarrow - <F><T'> \mid <F><T'> -$

6. $<T'> \twoheadrightarrow * <F><T'> \mid <F><T'> *$

7. $<T'> \twoheadrightarrow / <F><T'> \mid <F><T'> /$

8. $<T'> \twoheadrightarrow \varepsilon \mid \varepsilon$

9. $<F> \twoheadrightarrow (<E>) \mid <E>$

10. $<F> \twoheadrightarrow i \mid i$

Prekladová gramatika G_{expr} obsahuje päť neterminály. A to konkrétne $<E>, <E'>, <T>$, $<T'>$ a $<F>$. Neterminál $<expression>$ je štartovací symbol pre danú prekladovú gramatiku. Vstupná abeceda Σ_I pozostáva z piatich symbolov - $i, +, -, /, *, ($ a $)$. Výstupná abeceda Σ_O pozostáva zo symbolov $i, +, -, *$ a $/$.
Prekladová gramatika predstavuje prevod z infixovej formy zápisu do postfixovej formy.

Príklad 4.1. Preklad pomocou gramatiky G_{expr}

Ako jednoduchý príklad prekladu za pomoci už definovanej gramatiky G je možné ako vstupný reťazec zvoliť: $(i + i) * i$.
Potom rozklad, ktorý je získaný za pomoci najpravejšej derivácie je nasledovný:

$$
\begin{aligned}
<expression> &\Rightarrow <T><E'> \mid <T><E'> && [1.]\\
&\Rightarrow <F><T'><E'> \mid <F><T'><E'> && [4.]\\
&\Rightarrow (<E>)<T'><E'> \mid <E><T><E'> && [9.]\\
&\Rightarrow (<T><E>)<T'><E'> \mid <T><E><><T><E'> && [1.]\\
&\Rightarrow (<F><T'><E>)<T'><E'> \mid <F><T'><E><T'><E'> && [4.]\\
&\Rightarrow (i<T'><E>)<T'><E'> \mid i<T'><E><T'><E'> && [10.]\\
&\Rightarrow (i<E>)<T'><E'> \mid i<E><T'><E'> && [8.]\\
&\Rightarrow (i+<T><E'>)<T'><E'> \mid i<T><E'>+<T'><E'> && [2.]\\
&\Rightarrow (i+<F><T'><E'>)<T'><E'> \mid i<F><T'>+<E'><T'><E'> && [4.]\\
&\Rightarrow (i+i<T'><E'>)<T'><E'> \mid ii<T'>+<E'><T'><E'> && [10.]\\
&\Rightarrow (i+i<E'>)<T'><E'> \mid ii+<E'><T'><E'> && [8.]\\
&\Rightarrow (i+i)<T'><E'> \mid ii+<T'><E'> && [3.]\\
&\Rightarrow (i+i)*<F><T'><E'> \mid ii+<F><T'>*<E'> && [6.]\\
&\Rightarrow (i+i)*i<T'><E'> \mid ii+i<T'>*<E'> && [10.]\\
&\Rightarrow (i+i)*i<E'> \mid ii+i*<E'> && [8.]\\
&\Rightarrow (i+i)*i \mid ii+i* && [3.]
\end{aligned}
$$

*Poznámka: V rozklade znak $\Rightarrow$ predstavuje $\Rightarrow_{RM}$. Z dôvodu prehladnosti bol znak najpravejšej derivácie vypustený.

Syntaktická analýza príjma vstupný reťazec, $(i+i)*i$, a jej výsledkom, je infixový tvar vstupného reťazca v tvare: $ii+i*$.

Gramatika G_{expr} je zatiaľ bez regulácie. Preklad bez regulácie príjme akýkoľvek tvar postfixovej notácie. Teda, nezáleží na použitých operandoch a poradí operandov.

Príklad 4.2. Nech G_1 je podmnožinou prekladovej gramatiky G_{expr}.
G_1 ma nasledovný tvar:

1. $<E> \twoheadrightarrow <T><E'> \mid <T><E'>$
2. $<E'> \twoheadrightarrow +<T><E'> \mid <T><E'>+$
3. $<E'> \twoheadrightarrow \varepsilon \mid \varepsilon$
4. $<T> \twoheadrightarrow <F><T'> \mid <F><T'> \mid i$
5. $<T'> \twoheadrightarrow \varepsilon \mid \varepsilon$
6. $<F> \twoheadrightarrow i \mid i$

Z takto definovanej gramatiky je jasné, že syntaktická analýza prijme len reťazce, ktoré obsahujú sčítanie.
Pridaním kontrolného reťazca $\Xi = \{1465(2465)^m 3; m \geq 2\}$ už preklad neprijíma akékoľvek reťazce v postfixovom tvare a na počte operandov záleží.
Nech vstupný reťazec je $i+i+i+i$. Potom preklad daného reťazca je nasledovný:

$$
\begin{aligned}
<E> &\Rightarrow <T><E'> \mid <T><E'> && [1.]\\
&\Rightarrow <F><T'><E'> \mid <F><T'><E'> && [4.]\\
&\Rightarrow i<T'><E'> \mid i<T'><E'> && [6.]\\
&\Rightarrow i<E'> \mid i<E'> && [5.]\\
&\Rightarrow i+<T><E'> \mid i<T><E'>+ && [2.]\\
&\Rightarrow i+<F><T'><E'> \mid i<F><T'><E'>+ && [4.]\\
&\Rightarrow i+i<T'><E'> \mid ii<T'><E'>+ && [6.]\\
&\Rightarrow i+i+<E'> \mid ii<E'>+ && [5.]\\
&\Rightarrow i+i \mid ii+ && [3.]
\end{aligned}
$$

Množina pravidiel, ktorá bola použitá, je **146524653**. Ak by nebol nepoužitý kontrolný reťazec Ξ, vstupný reťazec by bol prijatý a prevedený na infixovú notáciu. Avšak, špecifikovaný kontrolný reťazec nedovoluje prijať reťazec $i+i+$, pretože $146524653 \notin \Xi$, preto je preklad ukončený, vstupný reťazec je odmietnutý a žiadny prevod nie je vykonaný. Tj. na výstupnú pásku nie su zapísané žiadne symboly.

Definovana gramatiku G_{expr} bude považovaná za gramatiku, ktorá bude v celej syntaktickej analýze použitá pre vyhodnocovanie výrazov. Takto definovaný preklad bude implementovaný v 4 nasledujúcich výpočtových krokoch:

1. Prečíta vstupný výraz x, ktorý sa nachádza medzi aktuálnym vstupným symbolom a prvým výskytom symbolov: ; , >, <, =
2. Preloží získaný vstupný výraz x do ekvivalentnej infixovej y podoby aplikovaním prekladovej gramatiky G_{expr}
3. Vloží y ako poslednú položku do tabulky TE, kde TE značí tabuľku výrazov spravovanú prekladačom
4. Vráti integer i, ktorý špecifikuje pozíciu v tabulke TE[i].

4.2 Analýza s generovaním výstupného kódu

S definovanou prekladovou gramatikou G_{expr}, ktorá slúži pre spracovanie výrazov je možné vytvoriť prekladovú gramatiku G, ktorá bude slúžiť pre preklad kódu v jednoduchom jazyku. Nech tento jazyk má prekladovú gramatiku definovanú takto:

1. $<program> \twoheadrightarrow Begin <statement - list>$
2. $<statement - list> \twoheadrightarrow <statement>; EOL <statement - list>$
3. $<statement - list> \twoheadrightarrow End$
4. $<statement> \twoheadrightarrow var\ ID = <expression>$
5. $<statement> \twoheadrightarrow label\ ID$
6. $<statement> \twoheadrightarrow if(<expression>$relačný operátor$<expression>)\ goto\ label$
7. $<statement> \twoheadrightarrow write(<item>)$
8. $<statement> \twoheadrightarrow read(ID)$
9. $<statement \twoheadrightarrow ID = <expression>$
10. $<item> \twoheadrightarrow ID$
11. $<item> \twoheadrightarrow Integer$
12. $<item> \twoheadrightarrow Double$
13. $<statement> \twoheadrightarrow goto\ labelName$

Gramatika G pozostáva zo štyroch neterminálov: $<program>$, $<statement-list>$, $<statement>$ a $<item>$. Ďalej zo štrnástich terminálov: Begin, ;, EOL, var, ID, label, if, goto, relačný operator, write, read, Integer, Double a labelName. $<program>$ je štartovací neterminál.

Nad danou gramatikou bude pred prekladom zavedený kontrolný reťazec Ξ, podľa ktorého sa po skončení prekladu rozhodne, či daný vstupný program bude prijatý a preložený, alebo vstupný program bude odmietnutý.

Takto definovaná gramatika spolu s prevodníkom umožní vstupný program prekladať priamo do výstupného jazyku. Výstupný jazyk bude definovaný takto:

Vstup

I T1[j]

Kde $j \geq 1$. Táto inštrukcia, vloží vstup zo štandardného vstupu do tabuľky identifikátorov T1 na pozíciu j.

Vyhodnotenie

E T1[j], TE[i]

Kde $i, j \geq 1$. Táto inštrukcia vyhodnotí postfixový tvar výrazu na i-tej pozícii v tabuľke výrazov TE a vloží výsledok výrazu do tabuľky identifikátorov, T1, na j-tú pozíciu.

Označenie(z anglického 'label')

L T3[i]

Kde $i \geq 1$. Táto inštrukcia uloží návěští do tabulky náveští T3, ktorá je použitá pri vyhodnocovaní podmienok a skokov do rôznych častí programu.

Vetvenie

B T3[i]

Kde $i \geq 1$. Inštrukcia spôsobí zmenu výpočtu na inštrukcia T3[i].

Podmienka

C TE[i], r,TE[j], T3[k]

Kde $r \in \{=, <, >\}$. A $i, j, k \geq 1$. Inštrukcia najskôr určí hodnoty z postfixových výrazov v tabulke TE na pozíciach TE[i] a TE[j]. Ak hodnoty spolu s operátorom dávajú hodnotu - true, tak výpočet pokračuje od inštrukcie na pozícii T3[k], inak pokračuje dalsou inštrukciou.

Výpis

W TE[i]

Kde $i \geq 1$. Inštrukcia vypíše na štandardný výstup hodnotu z postfixovej notácie na pozícii TE[i].

Definovaním výstupného jazyka, je možné pôvodnú prekladovú gramatiku G modifikovať pridaním prevodov v jednodtlivých pravidlách:

1. $<program> \twoheadrightarrow Begin <statement - list> \mid \varepsilon$
2. $<statement - list> \twoheadrightarrow <statement>; EOL <statement - list> \mid \varepsilon$

3. $<statement-list> \twoheadrightarrow End \mid \varepsilon$
4. $<statement> \twoheadrightarrow var\ ID = <expression> \mid E\ T1[j]\ TE[i]$
5. $<statement> \twoheadrightarrow label\ ID \mid L\ T3[i]$
6. $<statement> \twoheadrightarrow if(<expression>$relačný operátor$<expression>)\ goto\ label \mid C\ TE[i]$, r, $TE[j]$, $T3[k]$
7. $<statement> \twoheadrightarrow write(<item>) \mid W\ TE[i]$
8. $<statement> \twoheadrightarrow read(ID) \mid I\ T1[j]$
9. $<statement \twoheadrightarrow ID = <expression> \mid E\ T1[j]\ TE[i]$
10. $<item> \twoheadrightarrow ID \mid T1[j]$
11. $<item> \twoheadrightarrow Integer \mid Integer$
12. $<item> \twoheadrightarrow Double \mid Double$
13. $<statement> \twoheadrightarrow goto\ labelName \mid B\ T3[i]$

Gramatika obsahuje rovnake neterminály a terminály ako predchádzajúca gramatika G. Jediná zmena v gramatike je určenie generovaných reťazcov na základe pravidiel. Pravidlá, ktoré ako výstup obsahujú ε, negenerujú žiadny výstup.

So zadefinovanou prekladovou gramatikou je možné vytvoriť funkcie, ktoré zodpovedajú každému neterminálu z prekladovej gramatiky. To znamená, že pre vykonanie syntaktickej analýzy z vytvorenej prekladovej gramatiky je potrebné vytvoriť tri funkcie. Vo funkciach je potrebné pre konečné rozhodnutie či daný reťazec bude prijatý, aby boli všetky označenia (z anglického label) použitých pravidiel uložené. Vďaka uloženiu všetkých použitých pravidiel je prevodnik schopný na konci analýzy rozhodnúť, či daná postupnosť pravidiel vyhovuje nami zvolenej regulácii, tj. reťazec pravidiel $\in \Xi$. Pre tento účel bude v nasledujúcom kóde vystupovať premenná **rules**, do ktorej budú vkladané všetky označenia použitých pravidiel.

Následné funkcie sú napísané v pseudokóde. Ich účelom je ukážka, akým spôsobom budú v naslednej implementácii spracované pravidlá v definovanej podobe.
Funkcia *program* zodpovedá neterminálu $<program>$, ktorý je štartovacím neterminálom prekladu.

```
function program: boolean {
    rules.add(1);
    status = getToken();

    if ( !status ) then return status;

    status = false;
    if ( token.value == Begin ) then
        rules.add(1);
        status = statement-list();

    return status;
}
```

Funkcia $statement - list$ zodpovedá neterminálu $< statement - list >$. Tento neterminál stojí na ľavej strane dvoch pravidiel, ktoré negenerujú žiadne výstupné reťazce:

2. $< statement - list > \twoheadrightarrow < statement >; EOL < statement - list > \mid \varepsilon$

3. $< statement - list > \twoheadrightarrow End \mid \varepsilon$

```
function statement-list: boolean {
    status = getToken();
    if ( !status ) then return status;

    status = false;

    if ( token.value == End ) then
        rules.add(3);
        status = true;
        return status;

    else
        status = statement();
        if ( !status ) then return status;

        getToken();
        if ( token.value == ';' ) then
            getToken();
            if ( token.value == EOL ) then
                rules.add(2);
                status = statement-list();
            else
                return false;
        else
            return false;

  return status;
}
```

Funkcia *statement* zodpovedá neterminálu $< statement >$. Tento neterminál stojí na ľavej strane siedmich pravidiel, z ktorých všetky generujú výstupné reťazce zodpovedajúce výstupnému programu.

```
// Pre väčšiu prehladnosť, funkcia read je použitá pre získavanie
// nových tokenov. V prípade lexikalnej chyby, funckia ukonci preklad
// pomocou funkcie exit()
function read: void {
    status = getToken;
    if ( !status ) exit(status);
}
```

```
function statement: boolean {
    status = false;

    read()

    if ( token == { type tokenu var } ) then
        // simulácia produkcie pravidla č. 4
        read();

        if ( token == ID ) then
            newVariable = saveVariable(token);

        read();

        if ( token == { type tokenu = } ) then
            read();
            if ( token.type == { typ tokenu výraz } then
                index = processExpression();

                print("E T1[" + newVariable.index + "] TE[" + index + "]");
                rules.add(4);

    else if ( token.type == { typ tokenu label } then
        // simulácia produkcie pravidla č.5
        read();

        if ( token.type == { typ tokenu ID } then
            label = saveLabel(token);

            print("L T3[" + label.index + "]");


    else if ( token.type == { typ tokenu if } then
        // simulácia produkcie pravidla č.6
        read();

        if ( token.type == { typ tokenu ( } then
            index1 = processExpression();

            read();

            operand = token;

            read();
            index2 = processExpression();

            read();
            if ( token.type == { typ tokenu goto } ) then
```

```
            read();
            if ( token.type == { typ tokenu labelName } ) then
                labelIndex = getIndex(labelName);
                print("C TE[" + index1 + "]," + operand +
                      ", TE[" + index2 + "], T3[" + labelIndex + "]");
                rules.add(6);

else if ( token.type == { typ tokenu write } then
    // simulácia produkcie pravidla č.7
    read();

    if ( token.type == { typ tokenu ( } ) then
        read();
        index = processExpression();

        read();
        if ( token.type == { typ tokenu ) } ) then
            print("W TE[" + index + "]");
            rules.add(7);

 else if ( token.type == { typ tokenu read } then
    // simulácia produkcie pravidla č.8
    read();
    variable = getVariable(token);

    print("I T1[" + variable.index + "]");
    rules.add(8);

else if ( token.type == { typ tokenu ID } ) then
    // simulácia produkcie pravidla č.9
    variable = getVariable(token);

    read();
    if ( token.type == { typ tokenu = } ) then
        read();
        index = processExpression();

        print("E T1[" + variable.index + "] TE[" + index + "]");
        rules.add(9);

else if ( token.type == { typ tokenu goto } ) then
    read();

    labelIndex = getIndex(token);

    print("B T3[" + labelIndex + "]");
    rules.add(13);
 return status;
```

```
}
```

Ku kompletnej syntaktickej analýze je potrebné okrem týchto zadefinovaných funkcií zadefinovať riadiacu funkciu prekladu, ktorá využije štartovaciu funkciu, ktorá na konci prekladu vráti boolean hodnotu, ktorá odpovedá úspechu prekladu.

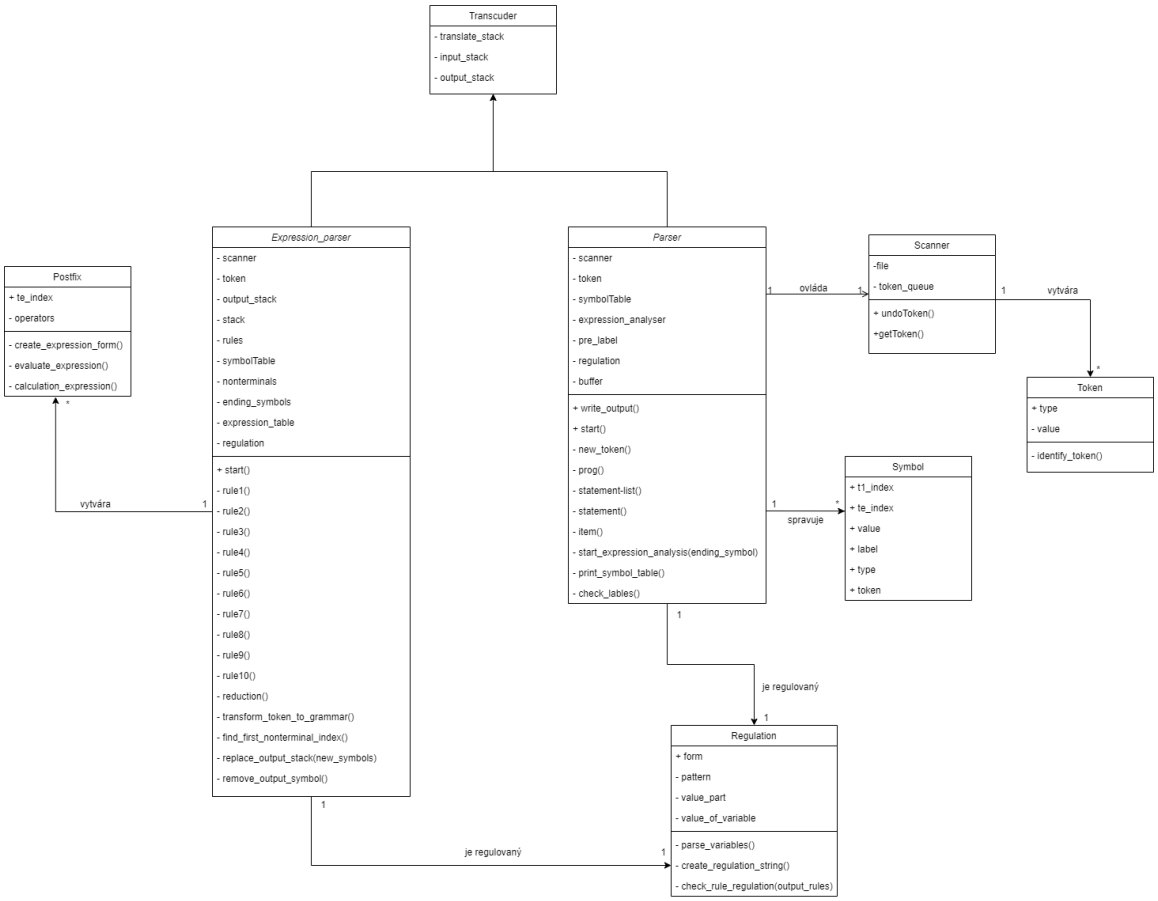
```
function main: void {
    regulation = { lubovolná forma regulácie };

    if ( program() ) then
        if ( eof ) then // end-of-file symbol
            if ( rules.belongs(regulation) ) then
                print("Úspešný preklad a generovanie kódu");
}
```

S takto zadefinovanymi boolean funkciami a hlavnou funkciou, ktorá spúšta celý preklad je mozné vstupný retazeč syntakticky analyzovať a priamo generovať výstupný kód. Takto definovaný syntaktická analýza stručne načrtáva logiku a nie je priamou impelemntáciou. Kód je zjednodušený a zapísaný v pseudokóde pre lepšiu prehladnosť a pochopenie celkovej logiky akou bude syntaktická analýza spracovaná v implementácii.

Kapitola 5

Implementácia a testovanie



Obr. 5.1: Diagram tried realizujúcich preklad

Kapitola popisuje spôsob implementácie scanneru pre vytváranie tokenov zo vstupného kódu a zásobníkového prevodníku s reguláciou ako parseru jazyku, ktorý bol definovaný v kapitole 4.2. Program je napisaný jazyku python, vo verzii 3. Implementácia využíva objektové vlastnosti jazyku. Návrh a štruktúra programu je znázornená v diagrame tried 5.1 ktorý popisuje vzťahy medzi objektami, metódy tried a atribúty. Pomocou vytvoreného programu sú demonštrované príklady, ktorá ukazujú použitie regulácie v prevodníkoch, tak-

tiež prácu prevodníku, kedy nie je potrebná ďalšia komponenta pre generovanie výstupného kódu.

V druhej časti kapitoly je popísaný spôsob testovania a konkrétne testovacie prípady, ktoré sú súčasťou programu. K programu je vytvorený Makefile, ktorý poskytuje príkazy k spusteniu jednotlivých testov.

5.1 Implementacia triedy Scanner

Scanner implementuje prevod symbolov zo vstupného jazyku na instanciu triedy Token. Token predstavuje zakladnú štruktúru, s ktorou pracuje následný celý preklad. Verejná metóda trieda Scanner getToken() je použitá pre vytvorenie nového tokenu zo vstupu.

Metóda GetToken() spracováva vstupný kód po riadkoch. Každý riadok je zakončený dvojbodkou. Syntaktická analýza prebieha na základe postupného toku tokenov, Scanner ukladá vytvorené tokeny z celého riadku do štruktúry deque[1], z ktorej sú následne tokeny poskytované pri zavolaní getToken() metódy.

Štruktúra Token poskytuje potrebné informácie pre syntaktickú analýzu získanú zo vstupných symbolov. Pri vytváraní tejto štruktúry je potrebná hodnota vstupného reťazca, tj. String podoba slova, ktoré bolo prečítané zo vstupu. Na základe vstupného reťazca, Token v privátnej metóde _identify_token() vyhodnotí svôj typ. Pre zachovanie prehladnosti fungovania prekladu, je typ tokenu obmedzený len na päť typov:

1. TOKEN_KEYWORD: Predstavuje typ tokenu, ktorý reprezentuje kľúčové slovo vstupného jazyku. Dané kľúčové slová nesmú byť použité v názve identifikátorov.

2. TOKEN_OPERATORS: Predsatavuje typ tokenu, ktorý reprezentuje operátory zo vstupného programu. Ich hlavné využitie je v spracovaní výrazov.

3. TOKEN_IDENTIFICATOR: Predstavuje typ tokenu, ktor=y reprezentuje identifikátory pre označenia (z anglického 'label') a názvy premenných. Na ich identifikáciu je použitých regex v tvare: ^[a-z_]+[a-z0-9A-Z]*$

4. TOKEN_INTEGER: Predstavuje typ tokenu, ktorý reprezentuje celé čísla. Pre ich identifikáciu je použitý regex v tvare: ^-{0,1}[1-9][0-9]*$|^0$

5. TOKEN_DOUBLE: Predstavuje typ tokenu, ktorý reprezentuje desatinné čísla. Pre ich identifikáciu je použitý regex v tvare: ^-{0,1}[0-9]+.[0-9]*$

[1] Štruktúra poskytujúca vlastnosti zásobníku a fronty zároveň

5.2 Implementácia prevodníku pre spracovanie výrazov

Spracovanie výrazov prebieha na základe LL tabuľky, 5.1, ktorá ukazuje použitie pravidiel vzhľadom na aktuálny stav vrcholov zásobníkov. Prvý stĺpec tabuľky označuje symboly na vrcholu prekladového zásobníku, prvý riadok predstavuje symboly na vrcholu vstupnej pásky.

	i	**+**	**-**	**/**	*****	**(**	**)**	**$**
E	1.					1.		
E'		2.					3.	3.
T	4.					4.		
T'		8.	5.	7.	6.		8.	8.
F	10.					9.		

Tabuľka 5.1: LL-tabuľka použitá pre spracovanie výrazov

Trieda Expression_parser implementuje pravidlá z tabulky 5.1. Tabuľka je v triede reprezentovaná štruktúrou slovník. Slovnik obsahuje kľúče, ktoré v tomto prípade sú neterminály prekladovej gramatiky a hodnotu pre daný kľúč. Hodnota je reprezentovaná druhým slovníkom, ktoréh kľúče sú terminály z abecedy a ich hodnoty predstavujú referencie na pravidlá z gramatiky G_{expr}. Výsledný slovník ma nasledovnú štruktúru:

```
Expression_parser._expression_table = {
    "E"  : {"i": self._rule1, "(": self._rule1},
    "E'" : {"+": self._rule2, ")": self._rule3, "$": self._rule3},
    "T"  : {"i": self._rule4, "(": self._rule4},
    "T'" : {"+": self._rule8, "-": self._rule5,
            "/": self._rule7, "*": self._rule6,
            ")": self._rule8, "$": self._rule8},
    "F"  : {"i": self._rule10, "(": self._rule9}
}
```

Ak vstupný neterminál sa nenechádza v kľúčoch príslušného slovníka, tak preklad je zastavený a pre danú kombináciu neterminálu na zásobníku vstupného symbolu neexistuje pravidlo, ktorým je možné pokračovať v preklade.

Pre každé pravidlo, trieda Expression_parser implementuje zvlášť funkciu. Preklad definovaný v pravidle je implementovaním pridávaním symbolov na prekladový zásobník a odoberaním expandovaného neterminálu z prekladového zásobniku. Ak dane pravidlo je ε-pravidlo, tj. deriváciou dostávame len ε, v tom prípade je zo zásobniku odstránený znak na ľavej strane pravidla a preklad pokračuje. Pre tieto operácie nad zásobníkom su implementované v triede nasledovnými funkciami:

- _find_first_nonterminal_index() - Na výstupnom zásobníku nájde prvý neterminál, ktorý bude nahradený podľa konkrétneho pravidla alebo vymazaný v prípade ε-pravidla nasledovnými funkciami.
- _replace_output_symbol - Z výstupného zásobníku zamení prvý neterminál za symboly podľa aplikovaného pravidla.
- _remove_output_symbol - Z výstupného zásobníku vymaže prvý nájdený neterminál, na ktorý bolo aplikované pravidlo, ktoré má na svojej pravej strane len ε

V pripade zhody terminálov na vrchole prekladového zásobníku a vrchole vstupného zásobníku je potrebné aby z prekladového zásobníku bol odstránený daný terminál a čítacia hlava vstupnej pásky bola posunutá. To znamená, z prekladového zásobníku odstranit terminál, pomocou funkciu _reduction() a vyžiadať od scanneru nový token, čím sa posune čítacia hlava na vstupnej páske.

Vo výrazoch su okrem konštant použité aj premenné. Tieto premenné musia byť pred použitím definované a zapísané v tabuľke symbolov v prekladači. Tento zápis premenných zabezpečuje syntaktický analyzátor (parser). Pri vytváraní inštancie parser-u výrazov mu hlavný parser predá referenciu na túto tabuľku výrazov, ktorú spravuje. Pri použití pravidla _rule10 v prípade výskytu premennej použitá táto referencia na kontrolu, či daná premenná bola definovaná a bola jej priradená hodnota pred použitím vo výraze. Ak sa daná hodnota nenachádza v tabuľke symbolov, znamená to chybu UNDEFINED_VARIABLE_ERROR.

Pri použití pravidla _rule10 sa na konci nahrádzajú symboly na výstupnom zásobníku. V prípade, že sa ma nahradit konštanta, dané číslo je jednoducho zapísané na výstup. V prípade existujúcej premennej je premenná zapísaná vo výstupnom formáte, tj. pozíciou v tabuľke premenných T1.

Výstup prevodníku pre výrazu je format výrazu v postfixovej forme. Táto postfixová forma je uložená v triede Postfix. Trieda Postfix pomocou triednej premenne te_index indexuje vytvorené postfix reťazce do tabuľky TE. Zo vstupného poľa znakov spraví reťazec, v ktorom su jednotlivé hodnoty výrazu oddelené medzerou pre lepšiu prehľadnosť. Popri Postfixovej forme je výstupom stavová premenná prekladu a reťazec, ktorý po preklade výrazu je pridaný k výstupnému bufferu parseru.

Kedže prevodník má schopnosť zápisu, je možné aby rovno vytvoril výstupný kus kódu, ktorý zodpovedá vytvoreniu výrazu v cieľovom jazyku. Prevodník na výstupnú pásku v prípade správneho zápisu výrazu, zapíše uloženie práve kontrolovaného výrazu do tabuľky TE s indexom novovytvoreného postfix výrazu v tvare:

```
buffer = "S TE[postfix_index] postfix_value"
```

Celý preklad vyŕazu sa odohráva vo funkcii start(ending_symbol). Táto funkcia je zavolaná z hlavného parseru, ktorý pri preklade narazí na očakávaný výskyt výrazu. Po zavolaní tejto funkcie prevodník nastaví základné hodnoty pre svoje interné premenné, vymaže zásobníky, vloží na vrchol prekladového zásobníku počiatočný symbol E gramatiky G_{expr} a symbol $ ako znak konca zásobníku. Preklad je riadený podľa prekladovej LL-tabuľky 5.1. Koniec prekladu nastáva v momente, kedy na vrcholy prekladoveho zásobníku je symbol $ a zo vstupného reťazca je pomocou funkcie _transform_token_to_grammar() je vygenerovaný taktiež symbol $. Symbol $ zo vstupného reťazca sa generuje na základe parametru ending_symbol, ktorý určuje ukončujúci symbol výrazu vo vstupnom programe. Tieto symboly môžu byť:

- ';': Dvojbodkou sa končia výrazy, ktoré sa vyskytujú v prípadoch priradenia k premennej.
- '>/</==/>=/<=': Týmito znakmi sa môžu končiť výrazy, ktoré sú použité pri rozhodovacej podmienke na ľavej strane.
- ')': Pravou zátvorkou sa končia výrazy na pravej strane v podmienke

V prípade, kedy sa na vrchole prekladového zásobníku nachádza ukončujúci symbol $ a vstupný zásobník nie je prázdny nastáva chyba a preklad je ukončený jedným z možných chybových kódov, ktoré su následne predané hlavnému parseru a ten preklad ukončí z dôvodu výskytu chyby.

Ak preklad prebebol bez výskytu chyby, je potrebná posledná kontrola v rámci prekladu. Je potrebné skontrolovať, či výsledná postupnosť použitých pravidiel zodpovedá zadanej regulácii. Pre tieto účely si trieda Expression_parser uchováva všetky postupne použité pravidlá. Na ukladanie je použitá zásobníková štruktúra, premenná _rules, do ktorej sú postupne pridávané čísla pravidla pri prvom vstúpení do funkcie zodpovedajúcej danému pravidlu. Na konci prekladu, je daná premnná použitá pre porovnanie so vstupnou reguláciou. V prípade, že použité pravidla nezodpovedajú príslušnej regulácii, výsledný stav prekladu je označený za neúspešný, prevodník vracia stavový kód REGULATION_ERROR a celý preklad je ukončený hlavným parserom po zistení chyby vo výraze.

5.3 Implementácia prevodníku pre syntaktickú analýzu

Syntaktická analýza prebieha v súlade s vytvorenou prekľadovou tabuľkou 5.2 pomocou pravidiel First zo stanovených pravidiel 4.2. Tabuľka určuje, ktoré pravidlo má byť použité pri konkrétnych hodnotách na prekladovom a vstupnom zásobníku. Prvý riadok zobrazuje vstupné symboly a prvý stĺpec určuje aktuálny neterminál na prekladovom zásobníku. Syntaktická analýza implementuje pre každý neterminál jednu samostatnú funkciu. Derivácia daného pravidla je simulovaná rekurzívnym volaním práve príslušných funkcii pre novo derivované neterminály.

	Begin	**End**	**var**	**label**	**if**	**write**	**read**	**id**	**int**	**double**
<prog>	1.									
<statement-list>		3.	2.	2.	2.	2.	2.	2.		
<statement>			4.	5.	6.	7.	8.	9.		
<item>								10.	11.	12.

Tabuľka 5.2: Tabuľka pravidiel pre syntaktickú analýzu

Začiatok celého prekladu začína vo funkcii `start()`. Súbor so vstupným zdrojovým kódom je prevodníku predaný parametrom pri vytváraní inštancie triedy reprezentujúcej syntaktickú analýzu. Po zavolaní funkcie `start()`, ako pri zavolaní všetkých nasledujúcich, sa nastaví stavová premenná prekladu `status` na hodnotu SYNTAX_ERROR a následne sa jej priradí hodnota, ktorá bude získaná z funkcie `prog()`. Táto premenná je ako lokálna premenná vytvorená v každej metóde, ktorá zodpovedá príslušnému pravidlu a je na začiatku každej funkcie priradená práve chybovej hodnote. Ak preklad prebehne správne, z rekurzívneho volania funkcii, sa na konci prekladu dostane hodnota SYNTAX_OK do prvotnej stavovej premennej, čo značí, že preklad prebehol bez chyby. Ak je v priebehu prekladu zavolaná funkcia, ktorá vracia stavovu hodnotu a jej stav je chybný, preklad sa zastaví, začne sa návrat zo zanorení pomocou rekurzie a chybová hodnota premennej `status` z miesta chyby prekladu je propagovaná na začiatok do funkcie `prog()`, kde sa preklad ukončí s chybovou hláškou.

Celý preklad je syntaxou riadený, tj. na základe syntaktických pravidiel sú všetky komponenty prekladu volané práve parserom. Na jeho požiadavky scanner spracováva vstupné reťazce na tokeny a predáva ich parseru pomocou metódy `_new_token()`. Táto metóda

využíva verejnú metódu triedy Scanner `getToken()`. Ak metóda vráti token, ktorý reprezentuje chybu, tak scanner narazil na neznámy reťazec, ktorý znamená lexikálnu chybu. Metóda ignoruje prázdne riadky, ktoré su reprezentované scannerom ako hodnota `None` namiesto tokenu.

Pri výskyte pravidla, v ktorom sa nachádza výraz, parser zavolá komponentu zodpovednú za preklad výrazov do postfixovej podoby pomocou metódy `start(ending_symbol)` v triede `Expression_parser`, ktorá v prípade úspechu vráti inštanciu triedy Postfix, v ktorej je uložený preložený výraz a reťazec, ktorý je potrebné konkatenovať s bufferom, do ktorého sú ukladané výstupné reťazce prekladu. Kedy je potrebné zavolať pomocný parser pre výrazy je v pravidlách definované neterminálom <expression>. Tento neterminál má len informatívny charakter. Nie je v gramatike definovaný ako neterminál a to z dôvodu, že gramatika s daným neterminálom nemá ako pracovať a je použitý len ako zarážka pre správne zavolanie parseru pre výrazy.

Ako je v úvode syntaktickej analýzy spomenuté, prevodník implementuje pre každý neterminál z 4.2 jednu samostatnú funkciu. Celý preklad teda pozostáva z piatich prekladových funkcií, ktoré zabezpečujú preklad podľa pravidiel a tabuľky použitia pravidiel 5.2. Všetky tieto funkcie sú vytvorené genericky s rovnakou postupnosťou a logikou funovania. Pre každý neterminál je vytvorená funkcia, ktorá pomocou vetvenia `if` používa konkrétne pravidlo pre terminál na vrchole vstupnej pásky. Ako príklad spôsobu vetvenia a použitia pravidiel pre neterminály na základe symbolu na vrchole vstupnej pásky nech stačí táto funkcia, ktorá je zobratá z implementácie a je zjednodušená len na znázornenie potrebného vetvenia:

```
def statement(self):
    status = SYNTAX_ERROR

    if ( token.type == TOKEN_KEYWORD ):
        elif ( token.value == "label" ):
            // Pravidlo 4. <statement> -> var ID = <expression>
            ...
            ...
        elif ( token.value == "var"):
            // Pravidlo 5. <statement> -> label ID
            ...
            ...
    elif ( token.type == TOKEN_IDENTIFICATOR ):
            // Pravidlo 9. ID = <expression>
            ...
            if ( token.type == TOKEN_OPERATOR and token.value == "=" ):
                status, expression = start_expression_analysis([";"])

                if ( status != SYNTAX_OK ): return status
                ...
                ...

    return status
```

Pri poslednom vetvení a použití pravidla 9. je ukázané spustenie analýzy pre výraz pomocou privátnej funkcie `start_expression_analysis`. V tejto funkcii je spracovaný vrátený

reťazec od triedy `Expression_parser`, ktorý je konkatenovaný do bufferu, v ktorom sa ukladá preložený program. Zavolaná funkcia scanneru `getToken()` pre získanie ukončujúceho znaku. Keďže analýza z fronty scanneru vyberie aj token, ktorý reprezentuje ukončujúci znak, je potrebné aby ho pomocou `undoGetToken(value)` vrátila naspäť na začiatok fronty, aby parser mohol daný token opäť analyzovať a zaradiť do svojej syntaktickej kontroly.

Pri použítí pravidla 9. je potrebné skontrolovať existenciu identifikátoru v tabuľke symbolov. Do tabuľky symbolov su identifikátory, ako premenných tak aj označení (z anglického 'label'), vložené pri ich prvej definícii. Pre premennú to znamená pravidlo č.4. Premenná je do tabuľky symbolov uložená ako inštancia triedy Symbol. V prípade premennej je príznak `label` nastavený na `false` a v prípade označenia je príznak `true`. Trieda predovšetkym slúži pre indexovanie premenných a výrazov pre výsledné prekladové tabuľky, T1 je premenné a T3 pre označenia. Toto indexovanie je zabezpečené triednymi premennými `t1_index` a `t3_index`, ktoré su zdieľané naprieč všetkými inštanciami, tz. ak je vytvorená nová inštancia Symbol premennej, ikrementácia `t1_index` zaručí dalšej inštancii nový index.

Pri označeniach môže dôjsť k situácii, kedy je požadovaný skok na označenie, ktoré ešte nebolo definované.Tento prípad môže napríklad nastať pri podmienke `if`, napriklad:

```
/*** Ukážka skoku na označenie, ktoré nie je definované pred použitím ***/
    var a = 1;
    label start;
    if ( a < 100 ) goto end;
    a = a * 5;
    goto start;

    label end;
```

V tomto prípade je skok na označenie `end` zaznamenaný, označenie je uložené do jednoduchého poľa a nenastane chyba v preklade. Na konci prekladu, v ktorom nenastala žiadna chyba, je toto poľe prejdené a v symbolickej tabuľke sa hladajú hodnoty, ktoré zodpovedajú hladanému označeniu. V prípade, keď sa označenie nájde, je vymazané z poľa a hľadá sa dalšie označenie. Ak pole ostane prázdne, preklad je vyhodnotený ako úspešný a pokračuje sa ďalej. V prípade, kedy sa niektoré z uložených označení nenachádza v tabuľke symbolov ani na konci celého prekladu, znamená to, že označenie nebolo definované a preklad je ukončený chybou NOT_DEFINED_LABEL.

Ako posledná kontrola úspešnosti prekladu je kontrola postupnosti pravidiel, ktoré boli aplikované na vstupný kód. Táto postupnosť musí splňovať reťazec, ktorý zodpovedá regulácii prekladovej gramatiky. Použité pravidlá sú ukladané do premennej `used_rules`. V prípade nezhody regulačného reťazca a použitých pravidiel preklad končí chybou REGULATION_ERROR. V prípade zhody s reťazcom je posledná kontrola úspešná prekladač na `stdout` vypíše vstupný kód vo výstupnom jazyku a ukončí preklad s hodnotou SYNTAX_OK

5.4 Implementácia regulácie pre prevodník

Trieda Regulation reprezentuje reguláciu prekladovej gramatiky prevodníku. Táto trieda ukladá vstupnú reguláciu, spracováva jej prevod na formu, v ktorej je možné porovnanie s použitými pravidlami a vyhodnotením správneho zápisu regulácie. Formálny zápis regulácie je definovaný v 3.2 v tvare: $\Xi = \{12^n(34)^m : n >= 1, m == 2\}$. Tento formát je v implementácii zachovaný, avšak mocnina pre jednoduchšií prevod na implementačnú formu je vyžadovaná aby bola zapísaná v tomto tvare: $\Xi = \{12n(34)m : n >= 1, m == 2\}$.

V takto špecifikovanom formáte je možné reguláciu zapísať do inštancie triedy Regulation. V prípade, kedy do inštancie nie je vložený žiadny regulačný reťazec, prevodník po ukončení svôjho prekladu požiada reguláciu o kontrolu a pri tejto kontrole bude vždy vrátená hodnota SYNTAX_OK. Ak regulačný reťazec je vložený, regulácie ho prijme a následne ho spracuje do regexového tvaru regulácie. Pre tento prevod trieda Regulation po vytvoreni svojej inštancie vytvorí zo vstupného reťazca 2 reťazce, ktoré sú oddelené na základe výskytu symbolu ':'. Ľavá strana po rozdelení tvorí samostatný reťazec a pravá strana predstavuje stranu, ktorá uchováva hodnoty premenných použitých v reťazci. Pravá strana je využitá v privátnej metóde `parse_variables()`, v ktorej sú premenné zapísané v pravej strane regulácie uložené do slovnika `value_of_variable`. Ako kľúč tohto slovníka slúži názov premennej a hodnot predstavuje regex výraz pre požadovanú hodnotu danej premennej. Relačné operátory su prevedené na tvar, ktorý je vhodný pre regex výraz a to:

1. 'n > x': Symbol je prevedený na formát zápisu {(x+1),}
2. 'n >= x': Symbol je prevedený na formát zápisu {x,}
3. 'n == x': Symbol je prevedený na formát zápisu {x}

Prvé číslo v tomto formáte určuje minimálnu hodnotu výskytu reťazca x, a čiarka za daným číslo predstavuje neobmedzenú hranicu maximálneho výskytu reťazca. V 1. prípade je k číslu x pričítaná jednotka, pretože výskyt reťazca sa nemôže rovnať číslu x. V 2. prípade je počet výskytov reťazca rovný aj číslu x. V 3. prípade už chýba čiarka, ktora neobmedzuje maximálny výskyt reťazca.

Po takto prevedených premenných z definície regulácie je možné zavolať funkciu `create_regulation_string`, ktorá používa funkciu `replace(old, new)`. Pomocou tejto funkcie sú vo vstupnom reťazci nahradené všetky výskyty premenných. Tymto sa vytvorí vzor pre regex porovnanie, ktorým je neskôr pri kontrole postupnosti použitých pravidiel použitý.

Trieda poskytuje verejnú metódu `check_rule_regulation(output_rules)` pre porovnanie použitých pravidiel so zadanou reguláciou. V tejto metóde je pole `output_rules`, ktoré tvoria hodnoty typu int, prevedené na pole, v ktorom sú hodnoty typu string. Toto pole je následne spojené do jedného reťazca, ktorý reprezentuje reťazec použitých pravidiel. Pomocou funkcie `re.match(pattern, string)` je sprostredkované porovnanie na základne vzoru, ktorý bol vytvorený vo funkcii `create_regulation_string` a uložený do premennej `pattern`. Ak reťazec zodpovedá použitému regulačnému reťazcu, je vrátená hodnota SYNTAX_OK syntaktickej analýze. V prípade nezhody reťazca s regulačným reťazcom je vrátená hodnota REGULATION_ERROR, ktorá aj v prípade bezchybného prekladu ukončí celý preklad s chybou upozorňujúcu na nedodržanie požadovanej formy zápisu.

5.5 Testovanie implementácie prekladača

Testovanie prekladu je rozdelené na časti, ktoré zodpovedajú komponentom, ktoré vytvárajú celý prekladač. Keďže prekladač v implementácii tvoria v podstate 3 samostatné časti, je potrebné ich odtestovať najskôr samostatne a až potom ako celok. Samostatne bude odtestovaný scanner, ktorý vytvára tokeny, či dané tokeny sú tvorené správne, či lexikálna analýza vie detekovať chyby a správne ich ukončiť. Následne je odtestovaný parser spracujúci výrazy, ktorý využíva scanner pre získavanie tokenov. Preto je dôležité, aby scanner fungoval bezproblémovo, inak by nebola možná syntaktická analýza. Po úspešnom testovaní spracovania výrazov je odtestovaný syntaktický analyzátor (ďalej len parser). Tento parser potrebuje funkčný analyzátor výrazov pre výskyt výrazov vo vstupnom kóde. Ako posledné je testovaná implementácia regulácie. Toto testovanie bude prebiehať ako na úrovni spracovania výrazov tak na úrovni syntaktickej analýzy vstupného kódu, keďže pre obe analýzy sú použité dve rôzne prevodníky.

Testovanie scanneru

Priebeh testovania scanneru je znázornený v tabuľke 5.3. Tabuľka obsahuje typické chyby, ktoré je potrebné detekovať pri vytváraní tokenov, tj. pri lexikálnej analýze. Tabuľka obsahuje tri typické lexikálne chyby, ktoré je nutné, aby scanner detekoval a ohlásil ich výskyt. Prvý prípad chyby, ktorý ukazuje chyby v kľúčovom slove Begin. Druhý prípad ukazuje použitie zlej zátvorky, treti prípad ukazuje použitie premennej end, namiesto kľúčového slova End na konci programu a v poslednom testovacom príklade je vytvorená premenná, ktorá začína nepovoleným znakom, tj. číslicou.

Vstupný program	**Výstupný kód**	**Očakávaný výstup kód**	**Stav**
Bgin var a = 0; End	LEXICAL_ERROR	LEXICAL_ERROR	OK

Begin
var a = 0;
if (a >0 } goto end;
label end;
End
	LEXICAL_ERROR	LEXICAL_ERROR	OK

Begin
var end = 0;
end
	LEXICAL_ERROR	LEXICAL_ERROR	OK

Begin
var 1a = 0;
End
	LEXICAL_ERROR	LEXICAL_ERROR	OK

Tabuľka 5.3: Tabuľka testovaní s chybným zápisom spôsobujúcim lexikálnu chybu

Scanner všetky očakávané chyby zaznamenal a vrátil chybový kód 401, čo predstavuje LEXICAL_ERROR konštantu. Je možné skonštatovať, že scanner pracuje správne a vie detekovať lexikalne chyby, ktoré sa vyskytnú vo vstupnom programe. Dalšie testy pre kontrolu zápisu kľúćových slov nie sú potrebné, kedže správnosť zápisu je kontrolovaná pomocou ich definícii v poli. Ak sa dané slovo nachádza v poli, patrí ku kľúčovým slovám. Ak sa dané slovo nenachádza v danom poli, jedná sa o chybu a nepatrí do kľúčových slov. Podľa regex

vzorov sa skontroluje iná zhoda s ďalšími možnostami. Preto kontrolovať ďalšie možnosti je nepotrebné.

Pri ďalšom teste je použitý správne zapísaný vstupný program. Od scanneru je požadované, aby správne zapísaný program rozložil na jednotlivé tokeny v správnom poradí a so správnym priradeným typom daného tokenu, ktorý je potrebný pre syntaktickú analýzu.

```
/** Vstupný program **/
Begin

var a1 = 0;
if ( a1 == 0 ) goto end;

label end;
End
```

```
/** Vytvorené tokeny *//
Begin [ TOKEN_KEYWORD ]
var [ TOKEN_KEYWORD ]
a1 [ TOKEN_IDENTIFICATOR ]
= [ TOKEN_OPERATOR ]
0 [ TOKEN_INTEGER ]
; [ TOKEN_KEYWORD ]
; [ TOKEN_KEYWORD ]
EOL [ TOKEN_KEYWORD ]
if [ TOKEN_KEYWORD ]
( [ TOKEN_OPERATOR ]
a1 [ TOKEN_IDENTIFICATOR ]
== [ None ]
== [ None ]
0 [ TOKEN_INTEGER ]
) [ TOKEN_OPERATOR ]
) [ TOKEN_OPERATOR ]
goto [ TOKEN_KEYWORD ]
end [ TOKEN_IDENTIFICATOR ]
; [ TOKEN_KEYWORD ]
EOL [ TOKEN_KEYWORD ]
label [ TOKEN_KEYWORD ]
end [ TOKEN_IDENTIFICATOR ]
; [ TOKEN_KEYWORD ]
EOL [ TOKEN_KEYWORD ]
End [ TOKEN_KEYWORD ]
```

Scanner správne rozdelil vstupný program do tokenov. Taktiež rozdelenie tokenov do skupín na základe typu hodnoty tokenu je správne. Všetky kľúčové slová vo vstupnom programe - Begin, var, if, goto, ;, label a End boli správne rozoznané a identifikované ako kľúčové slovo. Identifikátory vo vstupnom programe - a1 a end boli správne rozoznané ako identifikátory a všetko zapísane operátory - =, ==, (a) boli správne zaradené do skupiny operátorov. Vo vstupnom programe boli taktiež správne vygenerované tokeny, ktoré označujú koniec riadku - EOL (End Of Line znak). Scanner v prípade výskytu výrazov musí vrátiť dvakrát rovnaký ukončujúci znak. A to pre prípad, kedy syntaktická analýza predá riadenie prekladu analýze pre výrazy, ktorá od scanneru vyžiada ukončujúci znak. V tomto prípade, na konci analýzy výrazu je ukončujúci znak vloženi naspäť do fronty v scanneri. Preto je požadované aby scanner vrátil 2x posledné znaky v prípade spracovania výrazov.

Testy ukazujú, že scanner funguje požadovaným spôsobom. V prípade výskytu lexikálnej chyby vo vstupnom programe, scanner chybu zaznamená a predá chybový kód parseru, ktorý ukončí preklad s daným chybovým kódom. V prípade bezchybného vstupného programu, scanner v správnom poradí a so správnym typom predá parseru vytvorené tokeny s hodnotou a typom.

Testovanie spracovania výrazov

Testovanie analýzy výrazov prebieha rovnakým spôsobom ako predchadzajúci test scanneru. Prvý test vyžaduje detekciu chýb, ktorá sa vyskytuje v zlej forme zápisu výrazu. Odskúšané sú rôzne varianty zápisov výrazu. Prvý test ukazuje zápis dvoch po sebe idúcich operátorov. Druhý a tretí test ukazuje klasický matematický zápis výrazu, kedy vynechanie operátoru * znamená násobenie, avšak G_{expr} takýto zápis nedovoľuje. Štvrtý test používa nedefinovanú premennú vo výraze a posledný test odmieta postfixový zápis výrazu.

Vstupný reťazec	**Očakávaný výstup**	**Stav**
1-+8/12	WRONG_USED_EXPRESSION	OK
1+(5*1)8	WRONG_USED_EXPRESSION	OK
(5*8)(4*1)	WRONG_USED_EXPRESSION	OK
(a + (4 + 1))	UNDEFINED_VARIABLE_ERROR	OK
58-	WRONG_USED_EXPRESSION	OK

Tabuľka 5.4: Tabuľka testovaných výrazov, u ktorých sa očakáva výskyt chyby

Analýza všetky vstupné výrazy, ktoré obsahovali chybu, odmietla prijať. Parseru, ktorý je zodpovedný za predanie kontroly nad prekladom, bol vrátený chybový kód 450, ktorý predstavuje konštantu WRONG_USED_EXPRESSION. Táto chyba je detekovaná v prípade, kedy v slovníku analýzy výrazov sa nevyskytuje kombinácia neterminálu/terminálu na prekladovom zásobníku a vstupnom zásobníku. V prípade nedefinovanej premennej pred použitím vo výraze, analýza správne odmieta prijat daný výraz z dôvodu nedefinovanej premennej. V tomto prípade vráti parseru chybový kód 402, ktorý zodpovedá konštante UNDEFINED_VARIABLE_ERROR. Táto chyba je detekovaná v prípade, kedy použitá premenná sa nenachádza v tabuľke symbolov, ktorú spravuje hlavný parser prekladu.

Další test overí správnosť prevodu infixovej podoby výrazu do požadovaného postfixového tvaru. Ako príklady testu sú použité klasické príklady výrazu. Najskôr bez použitia zátvoriek, následne s použitím zátvoriek. Tieto výrazy sú následne porovnané so správny postfixovým tvarom, ktorému zodpovedajú.

Vstupný reťazec	**Očakávaný výstup**	**Stav**
5 + 19 - 2	5 19 2 + -	OK
5 * 8 + (14 * 2 + 4)	5 8 * 14 2 * 4 + +	OK
112 / (3 * 9)	112 3 9 * /	OK

Tabuľka 5.5: Tabuľka testovaných výrazov s úspešným prekladom na postfix formu

Analýza výrazov prijíma správne zapísane infixové výrazy a následne ich v správnom postfixovom formáte predáva ďalej. Chyby, ktoré sa môžu vo výrazoch vyskytovať taktiež správne detekuje a chybu správne predáva ďalej.

Testovanie syntaktickej analýzy

Testovanie dalšej komponenty prebieha rovnakým spôsobom. Najskôr sú otestované chybné varianty vstupného programu. Od parseru sa očakáva správna detekcia chýb, zastavenie prekladu a ukončenie s chybovým kódom. Testovacie príklady sú typické chyby, ktoré sa vyskytnú pri kontrole syntaxe vstupného programu. Jedná sa o chýbajúcu bodkočiarku na konci riadku, chýbajúce kľúčové slovo Begin pri začiatku programu, nedefinované označenie programu, na ktoré je uskutočnený skok, redefinícia premennej, ktorá už bola vytvorená a skok na premennú namiesto označenia. Všetky tieto príklady, očakávaný výstup prekladaču a aj stav, ktorý označuje úspešnosť testu, sú uvedené v tabuľke 5.6.

Poradie	**Vstupný program**	**Očakávaný výstup**	**Stav**
1.	var end = 0; End	SYNTAX_ERROR	OK
2.	Begin var end = 0 End	SEMICOLON_MISSING	OK

3.	Begin
	var a = 0;
	if (a >5) goto new_label;
	End
		UNDEFINED_LABEL_USED	OK

4.	Begin
	var a = 0;
	if (a >5) goto a;
	End
		NOT_LABEL	OK

5.	Begin
	var a = 0;
	var a = 2;
	
	End
		VARIABLE_REDEFINITION	OK

Tabuľka 5.6: Testovacia tabuľka s chybnými vstupnými programami

Syntaktická analýza zaznamenala v každom v prípade chybný zápis vstupného programu. V prípade testu č. 2, je chyba zaznamenaná už pri scanneri. Avšak, test na chýbajúcu bodkočiarku bol požadovaný od parseru. To, či je chyba zaznamenaná už pri lexikálnej analýze, nehrá žiadnu roľu. Parser dostal oznámenie o chybe, ukončil preklad so správnym chybovým kódom. Parser je schopný rozoznať chybné zápisy vstupného kódu, vyhodnotiť ich a ukončiť svôj preklad chybovým kódom. Dokáže detekovať chybové stavy, ktoré prichádzajú z jeho dalších komponent ako scanner a parser spracovávajúci výrazy.

Dalšie testy dostanú na vstup správne zapísaný vstupný program. Test bude zameraný na správny preklad vstupného programu do výstupného programu. Pre informáciu budú zapísané aj použité pravidlá, ktoré parser použil, aby sa dostal do ukončujúcého stavu.

Poradie	Vstupný program	Očakávaný výstup	Stav

1.	Begin
	var a = 0;
	
	End
		S TE[0], 0
		E T1[0], TE[0]
			OK

2.	Begin label new_label; End	L T3[0], new_label	OK

3.	Begin
	var a = 4;
	
	if (a >5) goto end;
	
	label end;
	End
		S TE[0], 4
		E T1[0], TE[0]
		S TE[1], T1[0]
		S TE[2], 5
		C TE[1] >TE[2] T3[0]
		L T3[0], end
			OK

4.	Begin
	var a = 4;
	var b = a * 2 - 1;
	End
		S TE[0], 4
		E T1[0], TE[0]
		S TE[1], T1[0] 2 1 - *
		E T1[1], TE[1]
			OK

5.	Begin
	var a = 4;
	var b = 0;
	read(b);
	
	label start;
	if (b == 0) goto end;
	a = a * b;
	goto start;
	
	label end;
	End
		S TE[0], 4
		E T1[0], TE[0]
		S TE[1], 0
		E T1[1], TE[1]
		I T1[1]
		L T3[0], start
		S TE[2], T1[1]
		S TE[3], 0
		C TE[2] == TE[3] T3[1]
		S TE[4], T1[0] T1[1] *
		E T1[0], TE[4]
		B T3[0]
		L T3[1], end
			OK

Tabuľka 5.7: Testovacia tabuľka so správne zapísanými vstupnými programami

Syntaktická analýza je schopná správne zapísané programy prijať a následne preložiť do výstupného jazyka. Taktiež vie správne indexovať označenia, výrazy a premenné v tabuľkách T3, TE a T1. Preklad je správne vyhodnotený podľa definovanej gramatiky.

Testovanie regulácie

Testovanie zoberie hodnoty z tabuľky 5.5 a aplikujú sa na dané výrazy regulačné výrazy. Pri testovaní bude použití jeden regulačný reťazec. $\Xi = \{14(106)^n 914(107)^m 10(83)^n : n > 1, m == 2\}$ je regulácia, ktorá je použitá pri testovaní.

Poradie	Vstupný program	Očakávaný výstup	Regulácia	STAV
1.	2 * 5 * (5 / 1 / 1)	2 5 * 5 1 / 1 / *	ANO	OK
2.	2 * 5 * 5	REGULATION_ERROR	ANO	OK
3.	2 * 5 * 5	25*5*	NIE	OK

Tabuľka 5.8: Testovacia tabuľka testovaní regulácie

Testovanie ukazuje odmietnutie vstupneho reťazca, ktorý nezodpovedá tvaru, ktorý reťazec má máť pri pridanej regulácii. Analýza túto chybu rozpozná na základe postupnosti použitých pravidiel a ukončí preklad s chybou REGULATION_ERROR. V prípade správneho tvaru reťazca, je prijatý a preložený na postfixový tvar. V prípade, kedy regulácie nie je pridaná, je reťazec prijatý a preložený.

Kapitola 6

Záver

Cieľ práce, získanie vedomostí o nových formách prekladových automatoch, ich formalizácia a následná implementácia sú v práci obsiahnuté a cieľe práce sú splnené. V práci sa podarilo formálne zadefinovať novú verziu prekladového automatu, porovnať jej schopnosti s prekladovým automatom bez pridaných vylepšení a následne implementovať.

Skúmanie prekladového automatu, ktorý je rozšírenou verziou zásobníkového automatu, pridaním výstupnej pásky dovolilo v práci zadefinovať zásobníkový prevodník. Tento prevodník v práci poslúžil ako základná štruktúra pri vytváraní novej verzie prekladového automatu. Danou definíciou sa v práci podarilo získať vedomosti o fungovaní zásobníkového prevodníku a jeho rozdieloch od klasického zásobníkového automatu. Spôsob, akým sú zápísané prekladové pravidlá v prevodníku pri pridaní dalšej pásky je potrebné pravidlá upraviť do formy, kedy dané pravidla je nutné obohatiť o výstupný formát daného pravidla. Spôsob prekladu bol v práci zo získaných vedomostí definovaný a formálne zapísany. Z tohto spôsobo fungovania ďalej vychádza celá práca, preto bolo potrebné aby tento spôsob bol správne pochopený a formálne definovaný. Oproti klasickému zásobníkovému automatu je spôsob prekladu pridaním výstupnej pásky z môjho pohľadu intuitívnejší a zároven nevyžaduje pridanie dalšieho komponentu do celého prekladu a to z dôvodu, že prevodník má vlastnosť zápisu pomocou pridanej zápisovej pásky a zápisovej hlavy, ktorá je teoreticky z pravej strany neohraničená, čo poskytuje neobmedzený zapisovací priestor. Zapisovacia hlava má možnosť pohybu v oboch smeroch, takže je možné prepisovať už zapísané znaky. Týmto spôsobom je prevodník schopný v preklade zapisovať na výstup aj neterminály, ktoré sa neskôr rozvinú na terminály podľa pravidiel. Pridaním výstupnej pásky sa z prekladu odstraňuje samostatná komponenta, ktorá slúži na generovanie výstupného kódu. Následkom toho syntaktická analýza preberá úlohu generovania kódu a nie je potrebná dalšia komponenta prekladu.

Skúmanie prekladových automat nemôže fungovať bez zavedení a definície gramatiky, ktorou sa následne prekladový automat riadi a prekladá vstupné reťazce. Gramatika je pojem, ktorý v informatike označuje štruktúru, ktorá definuje formálny jazyk. Pre preklad je neoddeliteľnou súčasťou a bez nej nie je preklad schopný ani len začať. Gramatika udáva každý krok celého prekladu, určuje, či daný vstupný reťazec môže byť prijatý a aké následovné pravidlo danej gramatiky má byt použité. Na jej základe sa celý preklad rozhoduje o ukončení alebo pokračovaní. Inak povedané, gramatika je najdôležitejšia súčasť celého prekladu. V práci je dôkladne rozobratá gramatika z formálneho hladiska, je ukázaná jej práca a spôsob, akým prijíma vstupné reťazce. Keďže gramatické pravidlá sa striktne určujú a následne sú implementované spôsobom, ktorý nedáva veľa voľnosti v ich úprave po implementácii, preto nie je spôsob ako zvýšiť vyjadrovaciu silu definovanej gramatiky bez

zásahu do jej pravidiel. Tento problém alebo skôr obmedzenie je možné vyriešiť pridaním regulácie. Regulácia v podstate predstavuje postupnosť použitých pravidiel. Vďaka tejto postupnosti sa v práci ukázalo, že je možné ako gramatiku obmedzit, aby prijímala len nami požadované reťazce. Ako napríklad v prípade použitia regulácie nad gramatikou príjmajúcu výrazy. Nad touto gramatikou bola použitá regulácia, ktorá zakazuje špecitický typ výrazu a preto každý zápis takéhoto výrazu bol odmietnutý. Toto by klasickým spôsobom bolo možné docieliť len kompletnou zmenou gramatiky, čo by sa prejavilo na implementačnej úrovní kompletnou zmenou kódu v spracovaní výrazu. Avšak, pridaním regulácie, ktorá na konci celého prekladu rozhodne o správnosti vstupného jazyka je možné bez akejkoľvek implementačnej zmeny v už fungujúcom prekladači ovplyvniť koncový výstup.

V práci sa podarilo ukázať, že gramatika, ktorá je regulovaná určitým spôsobom, je schopná obmedzit (regulovať) prijímané vstupné reťazce. Táto vlastnosť je veľmi zaujímavá a dá sa s ňou pracovať pri zmene požiadavok na prekladový automat s takouto gramatikou a podobne. Avšak, čo je zaujímavejší poznatok, že pomocou vhodnej regulácie je možné rozšíriť vyjadrovaciu schopnosť danej gramatiky. Tým je možné rozšíriť gramatiku, ktorá generuje bezkontextový jazyk na gramatiku, ktorá je schopná prijať niektoré reťazce z kontextového jazyka. Takto je možné posilniť vyjadrovaciu silu danej gramatiky bez akýchkoľvek vnútorných zmien gramatiky. Týmto spôsobom môže byť akákoľvek existujúca gramatika bez jej vnútorných zmien zmenená tak, aby prijímala a generovala jazyk, ktorý je v daný moment od nej požadovaný. Práca ukazuje príklad, v ktorom je možné vidieť schopnosť rozšíriť existujúcu gramatiku, ktorá generuje bezkontextový jazyk. Zavedním správnej regulácie nad danou gramatikou je možné prijať niektoré reťazce, ktoré patria do kontextovej gramatiky a tým rozšíriť jej vyjadrovacie schopnosti.

V implementačnej časti práci bolo potrebné overiť, či takto zadefinovaný prekladový automat, ktorý využíva reguláciu nad svojou prekladovou gramatikou, je možné využiť pri preklade. V práci je po formálnom zadefinovaní štruktúr opísaný spôsob, akým je možné implementovať tento nový typ automatu. Pre implementáciu, ukážku a overenie práce novo zadefinovaného automatu je použitý jazyk Python 3. Tento jazyk hodnotím ako veľmi vhodný jazyk pre implementáciu podla popisu v práci. Implementácia následuje v práci definované funkcie a spôsob prekladu. Neskôr bolo možné overiť funkčnosť a schopnosť regulovaného zásobníkového prevodníku v reálnych skúškach a konkrétnych prípadoch. Gramatiky, ktoré boli definované v teoretickej časti, boli použité v implementácii a boli testované na konkrétnych prípadoch, ktoré ukázali, či daná implementácia funguje a či je možné takto definovaný prekladový automat využiť. Prekladač, ktorý využíva novošpecifikovaný prekladový automat, bolo v rámci práce možné naprogramovať a otestovať. Samostatný program funguje a je schopný prekladu špecifikovaného jazyku. Preto môžem povedať, že definícia celého prekladu je správna a novo zavedený prekladový automat je schopný prekladu bez akýchkoľvek problémov, čím práca spĺňa ďalší z cieľov. Implementáciou bolo zistené, že aplikácia regulácie prináša jednoduchý kontrolný nástroj nad definovanou gramatikou a v prípade požadovanej zmeny je možné danú gramatiku upraviť pomocou regulácie do formátu, ktorý je vyždadovaný. Samozrejme je možné túto gramatiku upraviť len do formy, ktorá samotná gramatika dovoľuje.

Testovanie prináša overenie, že implementácia, definícia a zavedenie novej formy, je správne. Testované boli rôzne formy vstupných reťazcov s použitím alebo bez použitia regulácie. Na základe testov bolo v práci preukázané, že je možné využiť reguláciu pre obmedzenie vstupných reťazcov, prevodník je schopný prekladu aj bez zavedenia dalšej komponenty, ktorá sa samostatne stará o generovanie výstupného jazyka a že výstupná páska s obojsmernou hlavou má svoje výhody pri implementácii prekladača.

Práca poskytuje dobrý základ, ktorý je možné využiť pri pokračovaní v diplomovej práci. V diplomovej práci by som chcel nadviazať na aktuálnu demonštráciu využitia regulácie v zásobníkovom prevodníku. Keďže práca prináša demonštračný príklad, ktorý ukazuje, že vhodnou reguláciou je možné prijať niektorý nebezkontextový jazyk, tak pokračovanie v tejto práci by obnášalo formálne dokázať, že vyjadrovacia schopnosť zásobníkového prevodník s reguláciou sa zvýši, a to tak, že je schopná prijať kontextový jazyk. Zaujímavé by bolo skúmať, či takýto prekladový automat je možné porovnať s lineárne ohraničeným automatom, ktorý príjma kontextové jazyky. Aké rozdiely sú medzi týmito automatmi a či by bolo efektívne využiť regulovaný zásobníkový automat ako alternatívu pri preklade kontextových jazykov. V rámci implementácie a aplikovaní prevodníku s reguláciou by mohlo byť zaujímavé sa pozrieť na možnosť, kedy by sa preklad riadil reguláciou. Tzn. poradie použitých pravidiel počas prekladu by bol riadený práve reguláciou. V tomto prípade by avšak bolo potrebné, aby automat mal schopnosť obojsmerného pohybu na vstupnej pásky. Preklad by sa neriadil najľavejšou alebo najpravejšou deriváciou vstupu, ale riadil by sa reguláciou, ktorá by určovala, na ktorý terminál má byť použité pravidlo a v akom poradí. Na rozdiel od príkladu v tejto práci, kedy je regulácia použitá v demoštrácii, v ktorej je reguláciou kontrolným spôsobom správneho zápisu vstupného programu . by mohlo riadenia prekladu reguláciou priniesť zaujímavé výsledky. Myslím, že má zmysel sa zaoberať aj naďalej zásobníkovým prevodníkom s reguláciou a jeho využitím v komplexnejších jazykoch.

Literatúra

[1] Meduna, A. *Automata and Languages.* 1. vyd. Springer-Verlag London, 2000. ISBN 978-1-85233-074-3.

[2] Meduna, A. *Elements of Compiler Design.* Taylor & Francis, New York, 2008. ISBN 9781420063233.

[3] Meduna, A. a Techet, J. *Scattered Context Grammars and their Applications.* WIT Press, Ashurst Lodge, Southampton, SO40 7AA, UK, 2010. ISBN 978-1-84564-426-0.

[4] Meduna, A. a Švec, M. *Grammars with Context Conditions and Their Applications.* Wiley, Hoboken, New Jersey, 2005. ISBN 0-471-71831-9.

[5] Rozenberg, G. a Salomaa, A. *Handbook of Formal Languages.* Springer-Verlag Berlin Heidelberg, 1997. ISBN 978-3-540-60648-2.

-