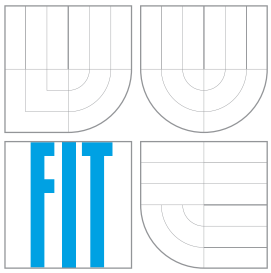
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA MATICOVÝCH GRAMATIKÁCH

PARSING BASED ON MATRIX GRAMMARS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR BEDNÁŘ
AUTHOR

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

BRNO 2014

Abstrakt

Tato práce se zabývá syntaktickou analýzou založenou na maticových gramatik. Zavádí metodu analýzy, která je řízena pomocí automatu. Diskutuje problémy deterministické analýzy maticových gramatik. Práce se zaměřuje na deterministickou analýzu některých jazyků, které nejsou bezkontextové. Studuje sílu této deterministické metody.

Abstract

This thesis is researching parsing based on matrix grammars. Introduces automata controled parsing method. It discusses problems of deterministic parsing of matrix grammars. Thesis is focusing on deterministic parsing of non-deterministic languages. It studies strength of this deterministic method.

Klíčová slova

syntaktická analýza, maticové gramatiky, řízené gramatiky, determinismus, LL gramatiky

Keywords

syntax analysis, matrix grammars, controlled grammars, determinism, LL grammars

Citace

Petr Bednář: Syntaktická analýza založená na maticových gramatikách, bakalářská práce, Brno, FIT VUT v Brně, 2014

Syntaktická analýza založená na maticových gramatikách

Prohlášení

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

.....
Petr Bednář
21. května 2014

Poděkování

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

Obsah

Kapitola 1

Úvod

Jazyky jsou prostředkem komunikace. Přirozené jazyky jsou bohaté, ale podléhají změnám v důsledku změn světa, který mají reprezentovat. Tyto jazyky je tak obtížné analyzovat a popsat neměnnou specifikací. Tuto nevýhodu odstraňují formální jazyky.

Formální jazyky se staly nepostradatelnou součástí teoretické informatiky. Umožňují popsat jak data, tak i použitou výpočetní metodu. Formální jazyky se tak stály základem pro popis programovacích jazyků a protokolů pro přenos a uložení dat.

Oblast bezkontextových jazyků byla dlouhou dobu důkladně zkoumána s cílem použití těchto jazyků v různých odvětvích nejen informatiky. Bezkontextové jazyky a jejich gramatiky jsou jednoduché, avšak často neposkytují dostatečnou vyjadřovací sílu pro reálné využití při specifikaci programovacích jazyků a překladech.

Řízené gramatiky mají za cíl zvětšit vyjadřovací sílu oproti svým neřízeným protějškům, přičemž umožňují zachovat původní formu pravidel takovéto gramatiky. Jednou z možností jsou pak maticové gramatiky, které povolují aplikovat pravidla pouze v přesně předepsaných sekvencích.

Syntaktická analýza je proces analyzující řetězec se záměrem rozhodnout, jestli řetězec náleží příslušnému jazyku. Tento proces je realizován jistým typem automatu. V práci bude uveden nový typ automatu, který umožňuje přijmout řetězec jazyka, definovaného pomocí maticových gramatik a jejich modifikací. Tento automat si klade za cíl zachovat vnitřní strukturu definovanou pomocí maticových gramatik. Syntaktická analýza těchto gramatik by jinak musela být realizována jiným známým modelem, který však těmto gramatikám plně neodpovídá.

Práce je zaměřena na vytvoření takové modifikace maticové gramatiky, která dovolí vytvoření některých kontextových jazyků, bude vstupním řetězcem procházet z levé strany a bude možno pro ni vytvořit deterministický automat, přijímající tento vstupní řetězec. Toto dovolí provést syntaktickou analýzu deterministicky. Tyto modifikace jsou teoreticky popsány a je pro ně vytvořen analyzátor.

Kapitola 2 slouží jako úvod do zkoumané problematiky formálních jazyků, gramatik a jejich hierarchie. Kapitola dává přehled základních známých pojmů.

Kapitola 3 popisuje základy provádění syntaktické analýzy a objasňuje základní omezení kladená na deterministickou syntaktickou analýzu, respektive deterministické gramatiky.

Kapitola 4 zavádí typ automatu, který je v této práci používán k syntaktické analýze maticových gramatik.

Kapitole 5 jsou pak popsány typ řízených gramatik, maticové gramatiky, z kterých tato práce vychází.

Kapitoly 6 a 7 popisují jednotlivé modifikace, vytvořené za účelem umožnění deterministické syntaktické analýzy.

Kapitola 8 popisuje implementaci navržených algoritmů a systémů, umožňujících analýzu nově navržených modifikací, v syntaktickém analyzátoru. Tento program je pak součástí příloh.

Závěr 9 pak poskytuje shrnutí dosažených závěrů a návrhy na možnosti budoucího pokračování vývoje.

Kapitola 2

Základní definice a pojmy

V této kapitole jsou popsány základní pojmy používané v teorii formálních jazyků. Především samotné jazyky a způsob popisu těchto jazyků. Na základě těchto pojmů budeme v této práci budovat další definice.

2.1 Abeceda a jazyk

Formální jazyky se snaží přesně popsat a studovat intuitivní pojem „jazyk“, který je základním kamenem formálních jazyků. Jazyky určují dovolená pořadí jednotlivých symbolů, které je tvoří. Uvedené poznatky jsou čerpány z [8], kde jsou také uvedeny příklady.

Definice 2.1. *Abeceda* je konečná neprázdná množina prvků, které se nazývají symboly.

Definice 2.2. Nechť Σ je abeceda.

- Prázdný řetězec ε je řetězec nad abecedou Σ.
- Pokud x je řetězec nad abecedou Σ a $a \in Sigma$, pak xa je řetězec nad abecedou Σ.

Σ^* značí množinu všech řetězců nad abecedou Σ a $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Definice 2.3. Nechť x je řetězec nad abecedou Σ. *Délka řetězce* x, značená $|x|$, je definována takto:

- pokud $x = \varepsilon$, pak $|x| = 0$ a řetězec x se nazývá prázdným řetězcem,
- pokud $x = a_1 \dots a_n$, pak $|x| = n$, pro $n \in \mathbb{N}^+$ a $a_i \in \Sigma$

Definice 2.4. Nechť x a y jsou řetězce nad abecedou Σ, pak konkatenace, neboli zřetězení, těchto řetězců je nový řetězec xy.

Definice 2.5. Nechť x je řetězec nad abecedou Σ. Pak x^n je *n-tá mocnina řetězce* x pro $n \geq 0$ definována takto:

- $x^0 = \varepsilon$,
- $x^n = xx^n$, pro $n \geq 1$.

Definice 2.6. Nechť x a y jsou dva řetězce nad abecedou Σ. x je *podřetězec* y, pokud existují řetězce z, z' nad abecedou Σ takové, že platí:

$$zxz' = y.$$

Definice 2.7. Nechť Σ^* značí množinu všech řetězců nad abecedou Σ. Každá podmnožina $L \subseteq \Sigma^*$ je *jazyk* nad Σ.

2.2 Gramatiky

Jazyk je definován jako množina řetězců nad konkrétní abecedou. Pokud je tato množina konečná, můžeme tuto vyjádřit výčtem jejích prvků. Toto však neplatí pro nekonečné jazyky. Gramatika je poté systém umožňující popsat nekonečnou množinu řetězců pomocí generativních pravidel.

Definice 2.8. *Neomezená gramatika* [11] je čtveřice

$$G = (N, T, P, S),$$

kde:

- N je abeceda neterminálů,
- T je abeceda terminálů, $N \cap T = \varnothing$,
- $P \subset (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$
- $S \in N$ se nazývá počáteční symbol.

Dvojice $(\alpha, \beta) \in P$ nazýváme přepisující pravidla, která obvykle zapisujeme ve tvaru $r : \alpha \to \beta$. Řetězce α a β označuje levou a pravou stranu pravidla a r je jeho návěstím.

Binární relace dvou řetězců $\gamma\alpha\delta$ a $\gamma\beta\delta$, kde $r : \alpha \to \beta$ se nazývá produkce pomocí pravidla r, kterou značíme jako $\alpha \Rightarrow \beta[r]$.

2.2.1 Chomského hierarchie

Gramatiky můžeme klasifikovat do tříd na základě jejich vlastností. Noam Chomsky vytvořil hierarchii čtyř typů gramatik [2], které se liší svou vyjadřovací silou.

Definice 2.9. Gramatiky **typu 0** jsou všechny neomezené gramatiky.

Jazyky generované pomocí gramatik typu 0 jsou *rekurzivně vyčíslitelné*. Tyto jazyky zkráceně značíme **RE**.

Definice 2.10. Gramatiky **typu 1**, někdy také kontextové gramatiky, jsou takové neomezené gramatiky, které používají pouze pravidla ve tvaru:

$$\alpha \to \beta, |\alpha| \leq |\beta|.$$

Jazyky generované pomocí gramatik typu 1 se nazývají *kontextově senzitivní*. Tyto jazyky zkráceně značíme jako **CS**.

Definice 2.11. Gramatiky **typu 2**, někdy také bezkontextové gramatiky, jsou takové neomezené gramatiky G, které používají pouze pravidla ve tvaru:

$$\alpha \to \beta, \alpha \in N, \beta \in (N \cup T)^*.$$

Jazyky generované pomocí gramatik typu 2 se nazývají *bezkontextové jazyky*. Tyto jazyky zkráceně značíme jako **CF**.

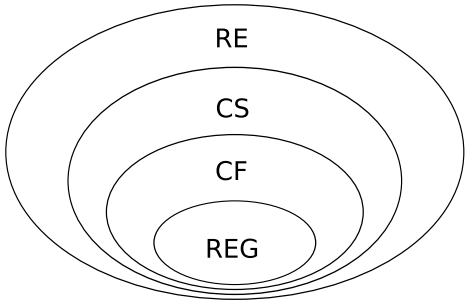
Definice 2.12. Gramatiky **typu 3**, někdy také bezkontextové gramatiky, jsou takové neomezené gramatiky G, které používají pouze pravidla ve tvaru:

$$\alpha \to \beta, \alpha \in N, \beta \in T^* \cup T^*N.$$

Jazyky generované pomocí gramatik typu 3 se nazývají *regulární*. Tyto jazyky zkráceně značíme **REG**.

Pro gramatiky v Chomského hierarchii a třídy jazyků, které jsou jimi definovány, poté platí věta 2.1 [3].

Věta 2.1. $REG \subset CF \subset CS \subset RE$



Obrázek 2.1: Chomského hierarchie jazyků

2.2.2 Řízené gramatiky

Řízené gramatiky můžeme rozdělit do několika základních skupin. Mezi tyto skupiny patří gramatiky řízené předepsanými sekvencemi pravidel, kontextovými podmínkami a paralelní gramatiky.

Gramatiky s řízenou derivací [6], patřící do skupiny s předepsanými sekvencemi pravidel dovolují omezit množinu pravidel, která je možno použít pro produkci v následujícím derivačním kroku. Toto omezení je realizováno pomocí dodatečné řídící struktury. Takovéto omezení produkce má za cíl změnit vyjadřovací sílu takovýchto gramatik oproti gramatikám, které řízené nejsou.

Kapitola 3

Syntaktická analýza

Gramatiky představují model pro generování řetězců daného jazyka. Syntaktická analýza [1] je opačný proces, který má za úkol určit, zdali konkrétní řetězec náleží do daného jazyka, či nikoli.

Syntaktickou analýzu provádí syntaktický analyzátor. Výstupem syntaktického analyzátoru je derivační strom pro vstupní řetězec. Vstupní řetězec není řetězcem daného jazyka, pokud není možno nalézt jeho derivační strom.

Metody přístupu k syntaktické analýze můžeme klasifikovat některými jejich základními vlastnosti. Metody se vyznačují směrem, kterým postupují. Pro metody pracující shora dolů platí, že svoji činnost začínají startovacím symbolem gramatiky, který se snaží pomocí produkčních pravidel modifikovat na vstupní řetězec. Metody pracující zdola nahoru pak postupují opačným směrem, kdy se snaží dokázat derivovatelnost vstupního řetězce na startovací symbol.

3.1 Syntaktická analýza hrubou silou

Tato metoda syntaktické analýzy se zaměřuje na hledání derivačního stromu vstupního řetězce pomocí slepého generování podmnožiny řetězců daného jazyka. V případě, že je vygenerován shodný řetězec, je výsledkem metody posloupnost derivací vedoucích k tomuto řetězci. Tento přístup je velmi obecný, jelikož vychází přímo z definice gramatik, které generují řetězce.

Výhodou použití metody hrubé síly k řešení syntaktické analýzy je její jednoduchost. Metoda je postavena na prohledávání stavového prostoru, sestávajícího se ze všech možných derivačních stromů. Při volbě algoritmu, prohledávajícího stavový prostor, však musíme brát na vědomí, že v gramatice se může objevit rekurze. Metoda prohledávání do hloubky [4] by v takovémto případě nemusela být dokončitelná.

Velkou nevýhodou tohoto přístupu je velká paměťová a časová náročnost. Tyto vlastnosti činí tento přístup hůře použitelný v praxi.

3.2 Syntaktická analýza se znalostí

Takováto metoda syntaktická analýza se zaměřuje na problémy analýzy hrubou silou. Hlavní problém použití hrubé síly je počet prohledávaných stavů, které jsou generovány neinformovanou metodou. Uplatnění znalostí omezuje počet následujících prohledávaných stavů

na základě znalosti současného stavu a vstupního řetězce. Metoda prohledávání se tak stává informovanou.

Tato metoda značně vylepšuje časovou náročnost analýzy, jelikož zmenšuje počet generovaných slepých větví a následných návratů. Pokud metoda, na základě znalostí, umožní vždy pokračovat maximálně jednou větví, je takováto metoda deterministická.

V případě použití deterministické syntaktické analýzy není nutné použít žádné navracení zpět. Takováto metoda nemusí uchovávat předchozí stavy, což umožňuje snížit paměťovou a časovou náročnost analýzy.

3.3 Syntaktická analýza řízených gramatik

Řízené gramatiky umožňují manipulovat s množinou pravidel pro další krok. Proto je možno použít i skupiny pravidel, která by v neřízené gramatice způsobila nedeterminističnost, za předpokladu, že deterministočnost v tomto případě zajistí řídící mechanismus. Tento řídící mechanismus pak musí být součástí syntaktické analýzy.

Kapitola 4

Gramatický automat

Syntaktickou analýzu bezkontextových gramatik můžeme provádět pomocí zásobníkového automatu [8], který má s touto gramatikou shodnou vyjadřovací sílu. Řízené gramatiky, založené na bezkontextových gramatikách, však mohou mít vyjadřovací sílu vyšší než samy tyto gramatiky. Zásobníkový automat poté neumožňuje syntaktickou analýzu takovýchto gramatik. Proto musíme doplnit zásobníkový automat o dodatečné řízení, nebo ho nahradit modelem, který má sám vyšší vyjadřovací sílu.

K této kapitole je představen gramatický automat. Gramatický automat je prostředek pro vyjádření syntaktické analýzy gramatik, řízených automatem. Tento automat vyjadřuje řízení posloupnosti derivací a je odvozen z hlubokého zásobníkového automatu [9]. Skládá se ze stavů a výpočetních pravidel. Automat však, narozdíl od svého vzoru, může upravovat do hloubky symboly také na vstupním řetězci a vyžaduje explicitní stanovení funkce, která odstraňuje již přijaté symboly.

4.1 Definice

Definice 4.1. *Gramatický automat* je osmice:

$$GA = (Q, \Sigma, \Gamma, R, s, S, F, d),$$

kde

- $Q = Q_e \cup Q_f$, kde Q_e je konečná množina prázdných stavů, které neuplatňují žádná gramatická pravidla a Q_f je konečná množina neprázdných stavů. Každý neprázdný stav aplikuje právě jedno produkční pravidlo ve tvaru $\Gamma^+ \to \Sigma^*$;
- Σ je vstupní abeceda;
- Γ je abeceda pracovních znaků;
- $R \subseteq (Q \times \mathbb{N} \times f \times Q)$ je konečná množina, kde $f : (\Gamma^* \times \Sigma^*) \to 0, 1$. $(p, v, f, q) \in R$ zapsané jako $p(v, f) \to q$ se nazývá pravidlo, kde f je funkcí proveditelnosti, kdy pravidlo je proveditelné pouze pokud je hodnota funkce 1 a v značí prioritu pravidla. Pokud je funkce f konstantní hodnota 1 a priorita $v = 0$, je takovéto pravidlo nazýváno *spojovací*. Pro $r_1 : p(v_1, f_1) \to q_1$, $r_2 : p(v_2, f_2) \to q_2$, kde $r_1, r_2 \in R$, $v_1 \leq v_2$, $S_1 \in \Gamma^*$ a $S_2 \in \Sigma^*$ platí, že pravidlo r_1 je proveditelné pouze v případě, že $f_1(S_1, S_2) = 1$ a $f_2(S_1, S_2) = 0$;

- $s \in Q$ je počáteční stav;
- $S \in \Gamma^+$ je počáteční pracovní řetězec;
- $F \subseteq Q$ je konečná množina koncových stavů;
- $d : (\Gamma^* \times \Sigma^*) \to (\Gamma^*, \Sigma^*)$ je *minimalizační funkce*, která je aplikována po každé aplikaci produkčního pravidla.

Gramatický automat je úspěšně ukončen v případě, že aktuální stav je koncovým stavem, zásobník je prázdný a vstupní páska taktéž.

Konfigurace GA je trojice (q, P, I), obsahující aktuální stav $q \in Q$, obsah zásobníku $P \in \Gamma^*$ a obsah vstupního řetězce $I \in \Sigma^*$.

Nechť $s_0, s_1, \cdots, s_n$ je *řetězec stavů* gramatického automatu A pro $n \geq 1$. Pak pro $0 \leq m \leq n-1$ existuje alespoň jedno pravidlo gramatického automatu takové, že:

$$s_m(v, f) \to s_{m+1} \in R$$

Nechť $r : p(v, f) \to q$ a $r \in R$ je proveditelným pravidlem GA M. Potom M může provést přechod ze stavu p do stavu q. Pokud je stav q neprázdný, je uplatněno pravidlo, které reprezentuje.

Přechod gramatického automatu ze stavu p do stavu q pomocí pravidla r, je značen jako:

$$p \vdash q[r].$$

Pokud je pravidlo r spojovací, je použito značení:

$$p \vdash_\varepsilon q[r].$$

Krok gramatického automatu $c[r]$ ze stavu p_0 a p_2 pomocí pravidla r je řetězec přechodů gramatického automatu takový, že:

$$c : p_0 \vdash_\varepsilon^* p_1 \vdash p_2[r].$$

Krok gramatického automatu je takový řetězec přechodů, který obsahuje právě jeden nespojovací přechod, který je poslední v řetězci přechodů.

Pro každý stav $p \in Q$ gramatického automatu je definován *spojovací uzávěr* $C(p)$:

$$C(p) = \{q : q \in Q, p \vdash_\varepsilon^* q\}$$

Činnost gramatického automatu začíná v konfiguraci zahrnující počáteční stav, zásobník obsahující počáteční pracovní řetězec a vstup obsahující vstupní řetězec. Každý krok automatu se pak sestává z uplatnění jednoho pravidla reprezentovaného pomocí stavu. Do tohoto stavu je umožněn přesun pouze v případě, že funkce proveditelnosti je vyhodnocena jako 1. Funkce proveditelnosti je vyhodnocována pouze za předpokladu, že neuspěla žádná z funkcí s vyšší prioritou. Po aplikaci pravidla je aplikována minimalizační funkce, která uvede zásobník i vstup do normalizovaného stavu a připraví tak automat pro další krok. V případě, že není možno provést další krok a není možno činnost automatu úspěšně dokončit, je toto vyhodnoceno jako chyba.

4.2 Determinismus

Gramatický automat umožňuje syntaktickou analýzu. Pokud má být řetězec touto syntaktickou analýzou přijat deterministicky, musí být přijímající automat také deterministický.

Pravidla gramatiky jsou reprezentována pomocí neprázdných stavů gramatického automatu, a proto se mezi těmito stavy musí automat pohybovat deterministicky. Musí proto splnit dvě podmínky. Nesmí být možno udělat krok z jednoho neprázdného stavu do druhého neprázdného stavu využitím pouze spojovacích přechodů. Druhá podmínka je pak, taková, že pro libovolnou konfiguraci gramatického automatu musí existovat maximálně jedna cesta, tvořená pomocí stavů automatu, vedoucí mimo spojovací uzávěr stavu vybrané konfigurace.

Definice 4.2. Gramatický automat je deterministický, pokud neexistuje žádný spojovací uzávěr obsahující dva nebo více neprázdných stavů, a pro každý spojovací uzávěr U, stavy $p_1, p_2 \in U$, $S_1 \in \Gamma^*$ a $S_2 \in \Sigma^*$ a všechna pravidla $r_1 : p_1(v, f_1) \to q_1, r_2 : p_2(v, f_2) \to q_2$, která nejsou spojovacími pravidly, platí, že maximálně jedna z hodnot $f_1(S_1, S_2)$ a $f_2(S_1, S_2)$ může být zároveň rovna 1. Zároveň neexistují žádné tři různé stavy $p_1, p_2, p_3 \in U$ pro které existují spojovací přechody $p_1 \vdash_\varepsilon p_3$ a $p_2 \vdash_\varepsilon p_3$.

V takovémto gramatickém automatu se můžeme pohybovat deterministicky. Zachováváme však původní strukturu gramatického automatu bez přidávání nadbytečných hran, které by vznikly odstraněním všech spojovacích pravidel.

Existující spojovací pravidla jsou tak pro činnost automatu transparentní. Jejich přítomnost nijak neovlivňuje množinu uskutečnitelných pravidel pro konkrétní stav, neboť byl ustanoven determinismus nejen pro každý přechod gramatického automatu, ale také pro každý krok gramatického automatu, který obsahuje libovolné množství spojovacích přechodů a končí právě jedním nespojovacím přechodem. Při výběru mezi více spojovacími přechody je vybrán ten, jenž vede k jediné možné přechodové funkci.

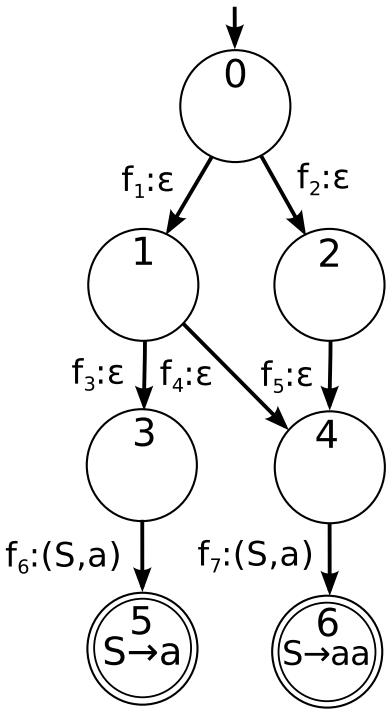
4.2.1 Příklad

Uvedené myšlenky o deterministočnosti gramatického automatu si představíme na příkladu.

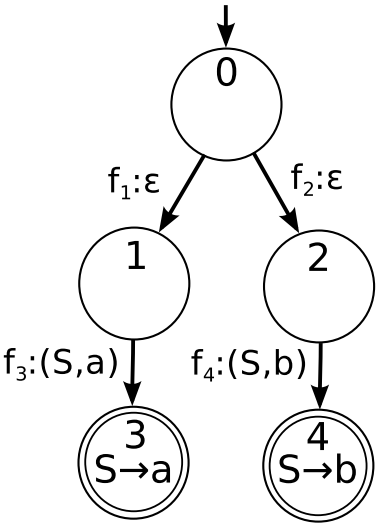
Mějme gramatický automat, jehož jádro tvořené stavy a přechody mezi nimi, je graficky reprezentováno na obrázku 4.1. Tento gramatický automat obsahuje 7 stavů, z nichž dva jsou stavy koncové. Všechny stavy jsou v grafické reprezentaci označeny číslem. Pokud se jedná o neprázdný stav, je v jeho popisku uvedeno pravidlo, jež zastupuje. Každá hrana má pak v popisku svoji funkci proveditelnosti. Znak ε značí spojovací přechod. Dvojice pak reprezentuje přechodovou funkci s prioritou 1, která je pravdivá právě v případě, že zásobník obsahuje jako podřetězec první prvek dvojice a vstupní páska obsahuje druhý prvek dvojice jako svůj podřetězec.

Tento gramatický automat není deterministický. Porušuje dvě podmínky deterministočnosti. Pokud bude automat v konfiguraci, obsahující stav 0, bude možno vyhodnotit obě funkce f_6 a f_7 jako pravdivé za shodných podmínek. Toto má za následek dosažitelnost obou stavů 5 a 6. Druhým porušením podmínek je pak existence dvou různých cest ze stavu 0 do stavu 6. Tyto cesty jsou tvořeny pořadím navštívených stavů $(0, 1, 4, 6)$ a $(0, 2, 4, 6)$.

Gramatický automat, reprezentovaný na obrázku 4.2 obdobným způsobem jako předchozí automat, je deterministický. Neporušuje žádnou z podmínek deterministočnosti.



Obrázek 4.1: Nedeterministický automat



Obrázek 4.2: Deterministický automat

Kapitola 5

Maticové gramatiky

Maticové gramatiky [10] patří mezi gramatiky s řízenou derivací. Jádrová gramatika je v tomto případě obohacena o konečnou množinu sekvencí jejích pravidel. Tyto sekvence se nazývají matice. Maticová gramatika poté provádí derivace pomocí matic. Při derivaci pomocí konkrétní matice jsou všechna pravidla v ní obsažená použita v uvedeném pořadí a poté je derivace ukončena, nebo je zvolena další matice pro další krok derivace.

Maticové gramatiky mohou používat libovolné omezení pravidel jádrové gramatiky. Pokud však není výchozí typ gramatiky stanoven jinak, je jím bezkontextová gramatika.

Řízené gramatiky obvykle definujeme tak, že definice jádrové gramatiky je doplněna o definici řídícího mechanismu. V tomto případě o matice.

Definice 5.1. *Maticová gramatika* [10], zkráceně MAT, je dvojice

$$H = (G, M),$$

kde

- $G = (N, T, P, S)$ je bezkontextová gramatika, nazývaná jádrová gramatika;
- $M \subseteq P^+$ je konečný jazyk, jehož řetězce se nazývají matice.

Bezkontextové gramatiky považují za derivační krok použití jednoho pravidla. Pro maticové gramatiky je však derivačním krokem použití celé matice, stávající se z použití jednotlivých pravidel. Jazyk je poté vyjádřen pomocí použití takovýchto složených derivačních kroků.

Definice 5.2. Pro dva řetězce $x, y \in (N \cup T)^*$, matici $r_1 r_2 \cdots r_n \in M$ a $n \geq 1$ je derivační krok maticové gramatiky H, zapsán jako $\Rightarrow_H$, definován tak, že:

$$x \Rightarrow_H y$$

právě když

$$x = x_0 \Rightarrow x_1[r_1] \Rightarrow x_2[r_2] \Rightarrow \cdots \Rightarrow x_n[r_n] = y,$$

kde každé $x_i \in (N \cup T)^*$ pro všechna $i, 0 \leq i \leq n$.

Jazyk $L(H)$ definovaný pomocí maticové gramatiky H je definován jako:

$$L(H) = \{w | w \in T^*, S \Rightarrow_H^* w\}$$

5.1 Příklad

Funkci maticových gramatik si ukážeme na následujícím příkladu.

Příklad 5.1. Nechť $H = (G, M)$ je maticová gramatika, kde $G = (N, T, P, S)$ je bezkontextová gramatika mající $N = \{S, A, B\}$ a $T = \{a, b\}$. Množina pravidel P poté obsahuje tato pravidla:

$$\begin{aligned} r_1 &: S \to AB, & r_4 &: A \to yA, & r_7 &: B \to x, \\ r_2 &: A \to xA, & r_5 &: B \to yB, & r_8 &: A \to y, \\ r_3 &: B \to xB, & r_6 &: A \to x, & r_9 &: B \to y. \end{aligned}$$

Matice $M = \{r_1, r_2r_3, r_4r_5, r_6r_7, r_8r_9\}$.

Derivace pomocí této gramatiky vždy začíná použitím pravidla r_1, které jako jediné umožňuje přepsat neterminál S. Každá matice poté generuje dvojici shodných terminálních symbolů na dvou různých místech současného řetězce. Neexistuje však žádná matice, která dovoluje vygenerovat dva rozdílné symboly. Takováto gramatika poté představuje jazyk

$$L = \{ww | w \in \{x, y\}^+\}.$$

Průběh generování řetězce může vypadat například takto:

$$\begin{aligned} S &\Rightarrow_H AB && [m_1] \\ &\Rightarrow_H xAxB && [m_2] \\ &\Rightarrow_H xyAxyB && [m_3] \\ &\Rightarrow_H xyxxyx && [m_4] \end{aligned}$$

5.2 Generativní síla

Pro každou bezkontextovou gramatiku můžeme sestavit její maticovou verzi, kdy pro každé jedno pravidlo je vytvořena matice, obsahující právě jedno pravidlo. Můžeme také sestavit takovou gramatiku, která nedovoluje všechny kombinace po sobě uplatněných pravidel.

Věta 5.1. $CF \subseteq MAT \subseteq MAT^\lambda$ *[6]*

5.3 Syntaktická analýza pomocí gramatického automatu

Syntaktická analýza je prováděna jistým automatem. Analýzu obecných maticových gramatik nemůžeme vyjádřit pomocí zásobníkového automatu. Neposkytuje dostatečnou vyjadřovací sílu. Můžeme však vytvořit analýzu založenou na gramatickém automatu. Reprezentace maticových gramatik pomocí maticového automatu nám dovolí zachovat sémantiku maticových gramatik, plynoucí z existence jednotlivých dovolených řetězců pravidel.

Tento popis je společný všem modifikacím maticových gramatik. Takovýto gramatický automat má jeden počáteční stav, který je prázdný. Každou matici pak vyjádříme jako řetězec stavů gramatického automatu. Pro $n \geq 1$, $1 \leq m \leq n$, každou matici $r = (r_1r_2 \cdots r_n) \in M$ maticové gramatiky H, počáteční stav $p_0 \in R$ a další stavy $p_m \in R$, která jsou navzájem různé, platí, že:

$$p_0(v_1, f_1) \to p_1(v_2, f_2) \to \cdots \to p_n(v_{n+1}, f_{n+1}) \to p_0, v_m \neq 0, v_{n+1} = 0, f_{n+1} = konst.1.$$

Stav $p_n \in F$ je koncovým stavem, kde využití spojovacího přechodu, vedoucího z tohoto stavu, značí úspěšnou aplikaci celé matice r, pro kterou je tento stav koncový.

Derivační krok maticových gramatik je definován tak, že po výběru matice je provedena derivace celou maticí. Syntaktická analýza pomocí tohoto přístupu by vyžadovala definovat první funkci proveditelnost matice tak, aby vyhodnotila následnou možnost aplikace všech pravidel matice a ostatní funkce proveditelnosti ve zbytku matice by mohly být vždy pravdivé. Funkce proveditelnosti zahrnující uplatnění všech pravidel matice by byla značně složitá.

Tuto jednu složitou funkci můžeme rozdělit do jednotlivých funkcí proveditelnosti vyhodnocovaných před přechodem do každého následujícího stavu matice. Může existovat více matic, které využívají stejnou posloupnost pravidel na svém počátku, a takovéto rozdělení funkce by vyústilo v existenci více stejných funkcí pro umožnění uplatnění stejných pravidel v rozdílných maticích. Proto pokud dvě nebo více matic mají prvních k pravidel shodných, poté mají prvních k neprázdných stavů shodných. Tímto dojde k odstranění duplicitních funkcí proveditelnosti. Díky spojení těchto stavů však nemůžeme přesně identifikovat matici, pomocí které provádíme derivaci, do doby než použijeme poslední její pravidlo.

Po každém uplatnění produkčního pravidla gramatickým automatem po vstupu do stavu, jsou vstupní páska a zásobník formátovány pomocí minimalizační funkce d.

Pro všechny neprázdné stavy takto vzniklého gramatického automatu platí, že mají právě jeden vstupní přechod a jeden nebo více výstupních přechodů. Stavy propojené nespojovacími pravidly tvoří strom. Ze všech listových a některých jiných stavů, která patří mezi koncové, tohoto stromu existují spojovací přechody do počátečního stavu. Případ, kdy spojovací přechod existuje pro stav, který není listem stromu, nastane, když existují dvě matice, kde řetězec pravidel první matice je předponou řetězce pravidel matice druhé.

Funkce proveditelnosti jsou odvozeny od vlastností cílových stavů. Pokud je cílový stav prázdný, jedná se vždy o spojovací pravidlo. Konkrétní transformace cílového stavu na funkci proveditelnosti a minimalizační funkce, jsou závislé na konkrétní modifikaci maticové gramatiky.

Pro nemodifikované gramatiky je pak funkce proveditelnosti definována tak, že je povolen přechod v případě, že řetězec na zásobníku obsahuje přepisovaný symbol. Pokud takovýto symbol neexistuje, pak není možno takovéto pravidlo uplatnit. Není však zaručeno, že takto sestavený automat bude deterministický. Přechodové funkce nepracují se znalostí vstupu, ale spoléhají pouze na zásobník, reprezentující současnou větnou formu.

Takovýto gramatický automat přijímá jazyk $L(H)$, který je definován pomocí maticové gramatiky H.

Kapitola 6

Modifikace LL1

První modifikací maticových gramatik je modifikace typu $LL1$. Maticové gramatiky, používající modifikaci typu $LL1$ se vyznačují tím, že v každém kroku je právě nejlevější neterminál, který existuje v současné větné formě, přepsán některým pravidlem. Pokud nejlevější neterminál není možné přepsat pomocí některého dostupného pravidla, dojde k chybě. Tato modifikace je obdobou modifikace $LL(1)$ [1] známé z neřízených bezkontextových gramatik.

Definice 6.1. Maticová gramatika $H = (G, M)$, která vznikla rozšířením bezkontextové gramatiky $G = (N, T, P, S)$ o matici pravidel M, je LL maticovou gramatikou prvního typu, dále jen MAT^{λ}_{LL1}, pokud dokážeme převést tuto gramatiku na deterministický gramatický automat, kde všechny nespojovací přechody mají stejnou nenulovou prioritu.

Abeceda vstupní pásky gramatického automatu $GA(H)$, je rovna $N \cup \{\$\}$ a abeceda zásobníku, je rovna $N \cup T \cup \{\$\}$. Počáteční řetězec zásobníku, je roven řetězci $S\$$.

Minimalizační funkce d je definována jako:

$$d(s_p s, s_p i) = (s, i).$$

Funkce proveditelnosti f, která je součástí pravidla r, vedoucího do neprázdného stavu, jehož součástí je produkční pravidlo tvaru $n \to p$, je odvozena následujícím způsobem:

$$f(s_0 s, i_0 i) = \begin{cases} s_0 = n \wedge i_0 \in \{p_0\} & \text{pokud řetězec } p = p_0 p_r \text{ a } p_0 \in N \\ s_0 = n & \text{jinak.} \end{cases}$$

Funkci proveditelnosti f je možno jednoznačně zapsat pomocí dvojice $(n, t)_f$, kde $n \in N$ reprezentuje nahrazovaný neterminál a $t \subseteq T$ reprezentuje predikční množinu pravidla. Pokud funkce nedefinuje závislost na vstupním symbolu p_0, pak $t = \varnothing$. Pro takové pravidlo neexistuje žádná predikce.

Překrytí dvou nespojovacích funkcí proveditelnosti $f_1 = (n_1, t_1)_{f1}$ a $f_2 = (n_2, t_2)_{f2}$, které značí nedeterminismus, je možné z tohoto zápisu odhalit tak, že existuje alespoň jedno t_x takové, že:

$$t_x \in t_1 \wedge t_x \in t_2.$$

Pokud vytvořený gramatický automat obsahuje alespoň jednu funkci proveditelnosti, jejíž hodnota nezávisí na vstupním symbolu i_0 a je součástí nespojovacího pravidla r, je potřeba přistoupit k další úpravě gramatického automatu, která má za cíl takovéto funkce s prázdnou predikční množinou nahradit jinými, které budou přímo záviset na nejlevějším zbývajícím symbolu vstupní pásky. Pravidlo r se nazývá ε-pravidlo. Přechod pomocí tohoto ε-pravidla se nazývá ε-přechod.

6.1 Algoritmus odstranění ε-přechodů

ε-přechody je možné eliminovat rekurzivním prohledáváním gramatického automatu a následnou úpravou funkcí proveditelnosti. Úprava těchto funkcí je provedena pomocí spojení původní funkce reprezentované pomocí $(n_1, t_1)_{f1}$ a nové funkce reprezentované pomocí $(n_2, t_2)_{f2}$ do funkce f_j tak, že:

$$f_j = (n_1, t_1 \cup t_2)_{fj}.$$

Algoritmus 6.1 Algoritmus odstranění ε-přechodů

Vstup: Stav s neznámou prediktivní množinou
Výstup: Nalezená prediktivní množina

```
 1: nalezeno ← Chodec((stav, stav.řetězec, ∅), (stav, (stav, 0)))
 2: function CHODEC((cStav, řetězec, cNavštívené), (gStav, gNavštívené))
 3:     možné ← ∅
 4:     if první znak řetězce je terminál then
 5:         možné ← {první znak}
 6:     else if řetězec ≠ ε then
 7:         for all p ∈ výchozíStav.další where p.přepisuje = první znak do
 8:             if cStav ∈ cNavštívené then
 9:                 Chyba(levá rekurze)
10:             end if
11:             N ← if stav je produkční then cNavštívené ∪ cStav else ∅
12:             možné ← možné ∪ Chodec((p, derivuj(řetězec, p), N), (gStav, gNavštívené))
13:         end for
14:     else
15:         N ← gStav.přepisuje
16:         if gStav.předchozí je prázdnýStav then
17:             P ← všechny prázdnýStav.předchozí s výskytem N × všechny pozice N
18:             if gStav.přepisuje = S then
19:                 G ← G ∪ (virtuální stav pro pravidlo S → S$, 1)
20:             end if
21:         else
22:             P ← gStav.předchozí × první pozice N
23:         end if
24:         for all (stav, n) ∈ P do
25:             if (stav, n) ∈ gNavštívené then
26:                 Chyba(levá rekurze)
27:             end if
28:             G ← gNavštívené ∪ (stav, n+1)
29:             řetězec ← Stav.řetězec[n..]
30:             for all p ∈ cStav.další where p.přepisuje = první znak do
31:                 možné ← možné ∪ Chodec((p, derivuj(řetězec, p), ∅), (stav, G))
32:             end for
33:         end for
34:         return možné
35:     end if
36: end function
```

Algoritmus 6.1 má za cíl nalézt množinu terminálů, které se mohou vyskytovat jako první symbol produkovaného řetězce, tedy predikční množinu. Na základě této množiny je pak možno definovat přechodovou funkci pro hranu vedoucí do stavu, na který je tento algoritmus aplikován.

Tyto terminály jsou nalezeny tak, že jsou hrubou silou vyprodukovány všechny řetězce, které je možno vyprodukovat při pokračování produkce řetězce ze stavu, který je cílem ε-přechodu. Řetězce jsou upravovány produkčními pravidly do doby, než je prvním symbolem řetězce terminální symbol. V takovémto případě je rekurze ukončena, protože predikční množina je tvořena pouze prvními symboly produkovaných řetězců a další pokračování by nepřineslo žádnou novou informaci. Každý první terminál vyprodukovaného řetězce je přidán do nalezené množiny terminálů.

Algoritmus využívá rekurze pomocí dvou kontextů. První kontext je používán pro nalezení výsledných terminálů pomocí uplatňování jednotlivých pravidel na první neterminál v prohledávaném řetězci. Druhý kontext pak prodlužuje prohledávaný řetězec v případě, že je kompletně smazán pomocí ε-pravidel. Prochází gramatický automat v opačném směru oproti prvnímu kontextu a generuje všechny možnosti pokračování věty za neterminálem, který je nahrazován původním pravidlem.

Algoritmus 6.1 je nutno aplikovat na všechny neprázdné stavy, pro které platí, že reprezentace funkce proveditelnosti jejich vstupní hrany je tvaru $(n, \varnothing)_f$.

Pokud algoritmus nalezne pouze prázdnou množinu, pak je gramatika chybně sestavena a produkce řetězce, zahrnující zkoumané pravidlo obsažené ve zkoumané matici nebude možno dokončit. V průběhu derivace řetězce se nutně objeví neterminál, který nebude možno derivovat pomocí žádného dostupného pravidla.

Pokud některý běh algoritmu 6.1 skončí chybou, pak gramatika obsahuje levou rekurzi a pokus zpracovat takovouto gramatiku pomocí algoritmu 6.2 by mohl potencionálně skončit zablokováním.

Věta 6.1. *Algoritmus 6.1 je dokončitelný v konečném čase.*

Důkaz. Gramatický automat pro maticovou gramatiku je sestaven z konečného množství neprázdných stavů, které tento algoritmus navštěvuje. Každý neprázdný stav je pak sestaven z pravidla konečné délky. Algoritmus navštěvuje jednotlivé stavy a podmnožiny řetězců, které tyto stavy generují. Existuje tedy konečné množství různých možností rekurzivního zanoření algoritmu. Návštěva jednoho produkčního stavu více než jednou v jednom rekurzivním řetězci plynoucímu z jednoho symbolu je explicitně zakázána, neboť reprezentuje levou rekurzi. ■

6.2 Algoritmus zpracování věty

Algoritmus 6.2 umožňuje zpracování věty jazyka, definovaného pomocí MAT^{λ}_{LL1} gramatiky.

Algoritmus nahrazuje vždy nejlevější neterminál zásobníku pomocí dostupného pravidla. Pokud je prvním symbolem zásobníku terminál, pak je tento odstraněn. Neterminál je tak při produkci vždy prvním symbolem zásobníku. Tento princip je shodný s principem příjmu bezkontextové gramatiky pomocí zásobníkového automatu. Simultánně při uplatnění pravidla je proveden přechod do nového stavu gramatického automatu. Tímto je omezena množina dále použitelných pravidel pouze na pravidla, která následují v matici.

Algoritmus 6.2 Syntaktická analýza MAT^{λ}_{LL1} gramatikou

Vstup: MAT^{λ}_{LL1} gramatický automat nastartovaný vstupním řetězcem
Výstup: Pořadí pravidel uplatněných na vstup nebo chyba

```
1: stav ← prázdnýStav
2: push('S$')
3: while zásobník a vstup není prázdný do
4:     token ← prvníTokenVstupu()
5:     if jeToken(top()) then
6:         if top() = token then
7:             pop()
8:             popVstup()
9:             continue
10:        else
11:            Chyba()
12:        end if
13:    end if
14:    D ← stav.další
15:    if stav je koncový then
16:        D ← D ∪ prázdnýStav.další
17:    end if
18:    nový ← D where přepisuje = top() ∧ token ∈ predikce
19:    if neexistuje nový then
20:        Chyba()
21:    end if
22:    pop(nový.přepisuje)
23:    push(nový.řetězec)
24:    stav ← nový
25:    if stav je prázdnýStav then
26:        Matice byla dokončena
27:    end if
28: end while
```

6.3 Generativní síla

Takto definovaná gramatika uplatňuje derivace striktně nejlevějším způsobem pomocí všech pravidel v matici. Definice MAT^{λ}_{LL1} gramatiky je proto ekvivalentní k definici MAT^{λ}_{SS} dle [5], kde je také uvedena věta 6.2.

Věta 6.2. $MAT^{\lambda}_{SS} = MAT_{SS} = CF$

Z tohoto tvrzení vyplývá, že jazyky, definované pomocí MAT^{λ}_{LL1} gramatiky, je možno přijmout pomocí zásobníkového automatu. Obdobně jako $LL(1)$ bezkontextové gramatiky.

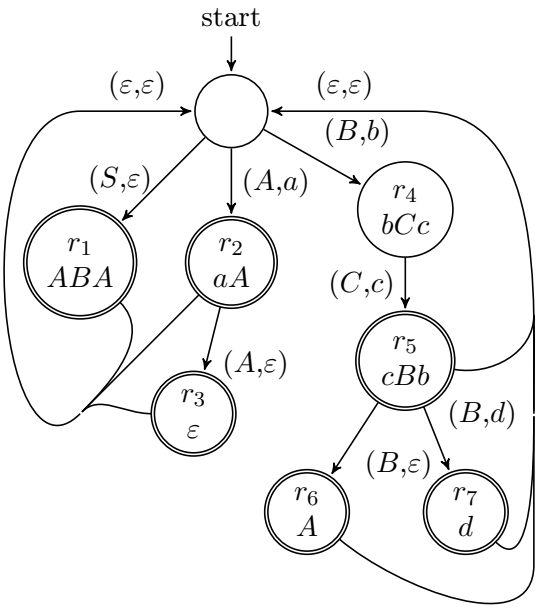
6.4 Příklad

Příklad 6.1. Nechť H je MAT^{λ}_{LL1} gramatika, kde gramatika G a matice M jsou definovány takto:

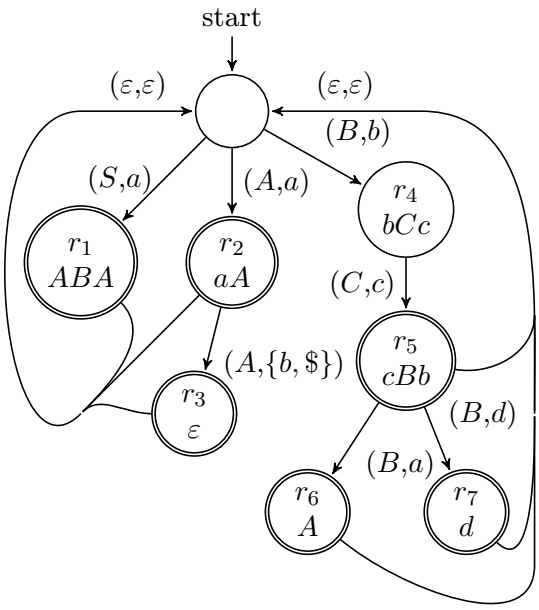
$$G = (\{S, A, B, C\}, \{a, b, c, d\}, R, S),$$
$$R = \{S \to ABA, A \to aA, A \to \varepsilon, B \to bCc, C \to cBb, B \to A, B \to d\},$$
$$M = \{r_1, r_2, r_2r_3, r_4r_5, r_4r_5r_6, r_4r_5r_7\}.$$

Tato gramatika definuje jazyk $\{a^+(bc)^n(a^+|d)(bc)^n a^+ | n \geq 1\}$

Pro tuto gramatiku můžeme vytvořit gramatický automat obsahující ε-přechody. Tyto přechody pak můžeme z tohoto automatu odstranit pomocí aplikace algoritmu 6.1.



Obrázek 6.1:
Gramatický automat s ε-přechody



Obrázek 6.2:
Upravený gramatický automat

Derivace pro řetězec *abcabca* poté bude probíhat pomocí pravidel takto:

$$\begin{aligned}
S &\Rightarrow ABA && [r_1]\\
&\Rightarrow aABA && [r_2]\\
&\Rightarrow aBA && [r_3]\\
&\Rightarrow abCcA && [r_4]\\
&\Rightarrow abcBbcA && [r_5]\\
&\Rightarrow abcAbcA && [r_6]\\
&\Rightarrow abcaAbcA && [r_2]\\
&\Rightarrow abcabcA && [r_3]\\
&\Rightarrow abcabcbA && [r_2]\\
&\Rightarrow abcabca && [r_3]
\end{aligned}$$

Kapitola 7

Modifikace LL2

Modifikace MAT^{λ}_{LL1} maticových gramatik měla velice přísné požadavky na pozici právě derivovaného neterminálu a neumožnila nám definovat jazyk, který není bezkontextový.

Maticová bezkontextová gramatika, využívající nejlevější derivace typu 2 [6], dále jen MAT^{λ}_{LL2}, je taková maticová gramatika, kde je v každém kroku přepsán nejlevější neterminál, který je možno přepsat na základě právě uplatnitelných pravidel. Nejlevější přepsatelný neterminál nemusí být absolutně nejlevějším neterminálem. Toto nastane v případě, že pro neterminály umístěné více vlevo v současné větné formě neexistuje žádné pravidlo dostupné ze současné pozice uvnitř matice.

Tato úprava značně upravuje možnosti gramatiky, neboť dovoluje větší volnost výběru pozice přepisovaného neterminálu při konstrukci gramatiky. Dovoluje nám ve značně omezené míře takzvané skákání, kdy produkční pravidla jsou aplikována v současné větné formě, z hlediska místa produkce, více nelineárně. Následující pravidlo je možno uplatnit na neterminál, který je hlouběji uvnitř řetězce oproti jednostrannému přístupu nejlevějších a nejpravějších gramatik.

7.1 Determinismus

Definice pro MAT^{λ}_{LL2} je postačující pro případy, kdy je gramatika používána pro generování řetězců jazyka, který definuje. Tato definice je však nedostačující pro vytvoření deterministického automatu, který má za cíl deterministicky rozhodnout, jestli daný řetězec náleží jazyku definovanému pomocí dané gramatiky. Definice MAT^{λ}_{LL2} sama přímo pokrývá podmínku možnosti uplatnění pravidla na jediný možný neterminál. Omezení možnosti výběru pravidla na maximálně jedno však definováno není. Tento problém jsme se rozhodli řešit dalším upřesněním definice, kdy jsou na gramatiku a jí generovaný jazyk uvaleny dodatečná omezení. Tato dodatečná omezení jsou vytvořena pro potřeby vytvoření deterministického automatu, který provádí syntaktickou analýzu shora dolů a zpracovává gramatiky postupem zleva doprava.

7.1.1 Lokalita

Derivační strom je reprezentací řetězce jazyka. V deterministické gramatice musí existovat právě jeden derivační strom pro právě jeden řetězec definovaného jazyka. Pro derivační strom a řetězec, který reprezentuje, musí platit podmínka lokálnosti. Symboly patřící každému jednomu podstromu musí být v řetězci umístěny těsně vedle sebe a podřetězce, generované z neprotínajících se podstromů, musí oproti sobě zachovat stejné pořadí jako

kořenové uzly těchto podstromů. Pokud by tato podmínka neexistovala, bylo by možno pro jeden derivační strom nalézt více řetězců, které ho specifikují. Tyto řetězce by se lišily tak, že by obsahovali stejné symboly pouze v rozdílném pořadí.

Pokud daná gramatika skáče uvnitř větné formy, přičemž asociuje generované terminály s terminály na vstupu, je nutné tuto podmínku zajistit dodatečným mechanismem. Toho jsem docílil pomocí zavedení pojmu *sekce*. Větná forma je rozdělena na jednotlivé sekce. Každá sekce vyjadřuje skupinu podstromů, které jsou v řetězci reprezentovány těsně za sebou. Řetězec obsahující pouze počáteční neterminál, je celý tvořen jednou sekcí.

7.1.2 Výběr terminálu

Pokud přepisujeme neterminál, který není absolutně nejlevější nebo absolutně nejpravější, musíme mít možnost produkovaný řetězec asociovat také s terminálem jiným než absolutně nejlevějším a nejpravějším. Zamezení možnosti takovéto produkce by automaticky implikovalo, že předcházející netermiály budou vymazány, nebo dojde k chybě. Terminálů, které by mohly být při produkci asociovány s produkovaným řetězcem, může být ve zpracovávaném řetězci více.

Mnou modifikovaná metoda postupuje řetězcem zleva. Proto konkrétní terminál, který bude asociován s produkovaným řetězcem, je vybrán tak, že se jedná o nejlevější možný.

7.1.3 Výběr pravidla

Bezkontextová gramatika postupující zleva vybírá produkční pravidlo, které bude použito v následujícím kroku, na základě množiny terminálů, které je možno očekávat jako první symbol v řetězcích, produkovaných z přepisovaného neterminálu. Pokud pravidlo přímo produkuje terminál jako první znak, tak tato množina obsahuje právě tento terminál. Pokud však obsahuje neterminál, musí být tato množina sestavena jinak. Tento postup však předpokládá úplnou znalost všech povolených kombinací následujících pravidel a pořadí jejich použití, dokud není prvním znakem produkovaného řetězce terminál, nebo není celá věta smazána a terminál je hledán ve zbytku věty. Pokud gramatika nedovoluje levou rekurzi, je tato množina odvoditelná přímo z pravidel gramatiky. Levě rekurzivní gramatika je při postupu zleva nedeterministická.

Pro gramatiky, které dovolují skákat uvnitř větné formy, a jejichž pravidla neprodukují vždy terminál jako nejlevější symbol, je však situace složitější. Umožnění nelineárního průchodu větou značně navyšuje počet možných pořadí uplatnění pravidel, použitých pro vytvoření konkrétního podstromu. Tento problém nastává, pokud v průběhu kompletní derivace konkrétního podstromu aplikuje jedno nebo více pravidel mimo sledovaný podstrom derivačního stromu.

Pro nalezení prediktivní množiny terminálů, které by indikovaly možnost použití pravidla, je tak potřeba prohledání jak pravidel gramatiky a její řídící struktury, tak vstupního řetězce. Pokud by měla být tato prediktivní množina vytvořena staticky, mohla by být hodně rozsáhlá. Tyto rozsáhlé množiny by snadno kolidovali a tím znemožnily možnost deterministického rozhodnutí.

Proto jsem omezil produkční pravidla takovým způsobem, aby nebylo nutné prediktivní terminály hledat, ale aby byl vždy určitelný pouze na základě samotného pravidla. To tak musí samo produkovat terminál. Tato úprava vylučuje použití pravidel, která neprodukují žádný terminál. Proto jsem přistoupil na řešení, kdy pravidla mají různé priority. Pravidla produkující terminál mají vyšší prioritu, než pravidla, která terminál neprodukují. Rozho-

dování mezi pravidly v nižší prioritě se pak neřídí přítomností konkrétních terminálů uvnitř vstupního řetězce.

7.2 Modifikace

Na základě omezení popsaných v sekci 7.1 jsem definoval dvě deterministické modifikace MAT^{λ}_{LL2} gramatik.

7.2.1 Striktně nejlevější

Tato modifikace určuje možnost použití pravidla na základě nejlevějšího terminálu, který bude součástí řetězce vygenerovaného z přepisovaného neterminálu.

Definice 7.1. MAT^{λ}_{LL2S} gramatika $H = (G, M)$ je taková MAT^{λ}_{LL2} gramatika, která vznikla rozšířením bezkontextové gramatiky $G = (N, T, P, S)$ o matici pravidel M. Množina pravidel P obsahuje pouze pravidla tvaru tn, kde $t \in T$ a $n \in N^*$, nebo ε-pravidla. Gramatiku H je možno převést na deterministický gramatický automat, kde:

- Pravidla vzniklá z produkčních ε-pravidel mají prioritu 1, ostatní nespojovací pravidla pak prioritu 2.
- Abeceda vstupní pásky gramatického automatu je rovna $N \cup \{|\}$ a prvním a posledním znakem na vstupní pásce je znak $|$ značící předěl skupin.
- Abeceda zásobníku je rovna $N \cup T \cup \{|\}$ a výchozím stavem zásobníku je řetězec $|S|$.
- Minimalizační funkce d je definována tak, že odstraní všechny korespondující terminály, na počátku každé sekce ze zásobníku i ze vstupní pásky. Pokud jsou obě korespondující sekce prázdné, jsou odstraněny. Pokud vstupní páska nebo zásobník obsahuje jedinou prázdnou sekci, tuto sekci odstraní.
- Nechť existuje nespojovací pravidlo $r : p(v, f) \to q$, produkční pravidlo $n_p \to t_p n$, které je aplikováno stavem q, množina R_p všech pravidel, umožňujících provést krok ze stavu p, N_p jako množina všech levých stran produkčních pravidel, ze kterých je vytvořena množina pravidel R_p, T_p jako množina všech terminálů obsažených v pravých stranách pravidel R_p, jejichž levá strana je n_p. Přepisovaným symbolem je nejlevější symbol zásobníku, patřící do množiny N_p. Přepisovanou sekcí je sekce, v níž přepisovaný neterminál leží. Pak funkce proveditelnosti f je definována takto:
  - Pokud n_p není přepisovaný symbol, pak není možno tímto pravidlem přepsat.
  - Pokud je přepisovaný symbol nejlevějším symbolem skupiny a $t_p \in T$, pak je výsledkem rovnost mezi t_p a prvním symbolem skupiny korespondující k přepisované.
  - Pokud produkční pravidlo není ε-pravidlem, pak je výsledkem možnost rozdělit skupinu korespondující k přepisované na tři části $s_0 s_1 s_3$ tak, že s_0 neobsahuje žádný symbol T_p a $s_2 = t_p$.
  - Pokud je produkční pravidlo ε-pravidlem, je možno provést přepsání přepisovaného symbolu vždy.

7.2.2 Současný nejlevější

Tato modifikace vybírá pravidlo na základě nejlevějšího terminálu, který může být kdekoli uvnitř řetězce generovaného z přepisovaného neterminálu. Tento terminál je však ve vstupním řetězci nejlevější, který je možno získat za pomoci právě dostupných pravidel.

Definice 7.2. MAT^{λ}_{LL2R} gramatika $H = (G, M)$ je taková MAT^{λ}_{LL2} gramatika, která vznikla rozšířením bezkontextové gramatiky $G = (N, T, P, S)$ o matici pravidel M. Množina pravidel P obsahuje pouze pravidla tvaru mtn, kde $t \in T$ a $m, n \in N^*$, nebo ε-pravidla. Gramatiku H je možno převést na deterministický gramatický automat, kde:

- Pravidla vzniklá z produkčních ε-pravidel mají prioritu 1, ostatní nespojovací pravidla pak prioritu 2.
- Abeceda vstupní pásky gramatického automatu je rovna $N \cup \{|\}$ a prvním a posledním znakem na vstupní pásce je znak | značící předěl skupin.
- Abeceda zásobníku je rovna $N \cup T \cup \{|\}$ a výchozím stavem zásobníku je řetězec $|S|$.
- Minimalizační funkce d je definována tak, že odstraní všechny korespondující terminály, na počátku každé sekce ze zásobníku i ze vstupní pásky. Pokud jsou obě korespondující sekce prázdné, jsou odstraněny. Pokud vstupní páska nebo zásobník obsahuje jedinou prázdnou sekci, tuto sekci odstraní.
- Nechť existuje nespojovací pravidlo $r : p(v, f) \to q$, produkční pravidlo $n_p \to mt_pn$, které je aplikováno stavem q, množina R_p všech pravidel, umožňujících provést krok ze stavu p, N_p jako množina všech levých stran produkčních pravidel, ze kterých je vytvořena množina pravidel R_p, T_p jako množina všech terminálů obsažených v pravých stranách pravidel R_p, jejichž levá strana je n_p. Přepisovaným symbolem je nejlevější symbol zásobníku, patřící do množiny N_p. Přepisovanou sekcí je sekce, v níž přepisovaný neterminál leží. Pak funkce proveditelnosti f je definována takto:
  - Pokud n_p není přepisovaný symbol, pak není možno tímto pravidlem přepsat.
  - Pokud je přepisovaný symbol nejlevějším symbolem skupiny, $m = \varepsilon$ a $t_p \in T$, pak je výsledkem rovnost mezi t_p a prvním symbolem skupiny korespondující k přepisované.
  - Pokud je přepisovaný symbol nejpravějším symbolem skupiny, $n = \varepsilon$ a $t_p \in T$, pak je výsledkem rovnost mezi t_p a posledním symbolem skupiny korespondující k přepisované.
  - Pokud produkční pravidlo není ε-pravidlem, pak je výsledkem možnost rozdělit skupinu korespondující k přepisované na tři části $s_0s_1s_3$ tak, že s_0 neobsahuje žádný symbol T_p a $s_2 = t_p$.
  - Pokud je produkční pravidlo ε-pravidlem, je možno provést přepsání přepisovaného symbolu vždy.

7.2.3 Reprezentace funkcí proveditelnosti

Funkce proveditelnosti obou modifikací, vycházejících z MAT^{λ}_{LL2}, je možno jako v případě MAT^{λ}_{LL1} reprezentovat jako dvojici $f_1 = (n_1, t_1)_{f1}$, kde n_1 je nahrazovaný neterminál a t_1 je množina terminálů. Tato množin však bude mýt maximální velikost 1. Takováto

reprezentace pomocí dvojice je však jednoznačná pouze v kontextu ostatních funkcí výchozího stavu. Funkce, je závislá na definici ostatních funkcích vycházejících ze stejného stavu a zpracovávajících stejný neterminál.

7.3 Algoritmus

Každé uplatnění přepisovacího pravidla rozdělí sekci, ve které je nahrazovaný neterminál na tři podsekce. Část před terminálem pravidla, terminál a část za terminálem pravidla. Korespondující sekce na vstupní pásce je rozdělena obdobně okolo terminálu, který indikoval přepsání neterminálu.

Pro vybraný neterminál je v příslušné skupině nalezen nejlevější terminál vstupního řetězce, který je součástí některého z pravidel přepisujících vybraný neterminál. Pokud přepisujeme první neterminál skupiny a pravá strana pravidla má jako první symbol terminál je prohledávání omezeno na první vstupní terminál skupiny. Naopak, pokud přepisujeme poslední neterminál skupiny a pravá strana pravidla má terminál jako poslední symbol je prohledávání omezeno na poslední vstupní terminál skupiny. Pokud takovýto terminál neexistuje a existuje ε-pravidlo pro vybraný neterminál, je toto pravidlo uplatněno.

Algoritmus 7.1 provede v cyklu uplatnění jednoho pravidla pomocí obou uvedených nejlevějších modifikací MAT^{λ}_{LL2}. Krok se provádí do té doby, než nastane chyba nebo je současná větná forma prázdná.

Algoritmus 7.1 Syntaktická analýza LL2 gramatikou

Vstup: MAT^{λ}_{LL2} gramatický automat nastartovaný vstupním řetězcem
Výstup: Derivační strom vstupu nebo chyba

```
1:  while zásobník není prázdný do
2:      Vyber aktuální množinu uplatnitelných pravidel z automatu
3:      Najdi nejlevější neterminál v zásobníku, který je možno přepsat vybranými pravidly
4:      if Takový neterminál neexistuje then
5:          Chyba
6:      else
7:          Vyber pouze pravidla přepisující nalezený neterminál
8:          Hledej první terminál umožňující přechod automatu vybranými pravidly
9:          if Terminál nenalezen then
10:             if Možno smazat neterminál then
11:                 Smaž neterminál
12:             else
13:                 Chyba
14:             end if
15:         else
16:             Přepiš neterminál a rozděl sekci
17:         end if
18:         Proveď krok gramatického automatu
19:         Eliminuj terminály v současné větné formě porovnáním se vstupem
20:     end if
21: end while
```

Tento algoritmus je shodný pro obě modifikace MAT^{λ}_{LL2S} a MAT^{λ}_{LL2R}. Liší se však použitým způsobem vyhledávání nejlevějšího neterminálu, který pro MAT^{λ}_{LL2R} gramatiky

implementuje dodatečné pravidlo týkající se nejpravějšího neterminálu v kombinaci s nejpravějším terminálem a bere v potaz možnost použít pravidla, která neobsahují terminál pouze jako první symbol.

Princip činnosti algoritmu vychází přímo z definice zpracovávaných gramatik, kdy je nalezen nejlevější možný neterminál a nejlevější možný terminál. Tento terminál je vyhledáván pouze v korespondující skupině, aby nebyla porušena lokálnost derivačního stromu. Prohledávání je prováděno lineárním průchodem z levé strany současně pro všechna pravidla přepisující daný neterminál. Takto je dosaženo nalezení nejlevějšího terminálu.

Při uplatnění nalezeného produkčního pravidla, je nalezený terminál zkonzumován pomocí vyčlenění do samostatné sekce a následným odstraněním pomocí minimalizační funkce gramatiky. Pokud vhodný terminál není nalezen, ale existuje vhodné ε-pravidlo, je nalezený neterminál vymazán.

Vyhledávání pro MAT^{λ}_{LL2S} gramatiky má jedinou výjimku když hledáme terminál pro první neterminál skupiny, je vyhledávání omezeno pouze na první terminál skupiny.

Vyhledávání pro MAT^{λ}_{LL2R} gramatiky pak rozdělí pravidla a jejich predikční množiny do třech skupin, podle pozice terminálu uvnitř řetězce pravé strany výchozího pravidla. V případě extrémního umístění neterminálu používá extrémně orientovaná pravidla na vyhledávání pouze na prvním respektive posledním symbolu vstupního řetězce. V ostatních případech použije tato extrémní pravidla na vyhledávání v celém řetězci podobně jako ostatní pravidla.

7.4 Generativní síla

Modifikace maticových gramatik, kdy není přepisován vždy nejlevější neterminál, jsou vytvořeny za účelem zvýšení vyjadřovací síly takovýchto modelů, oproti modelům přepisujícím vždy právě nejlevější neterminál.

Věta 7.1. $CF_{NDET} \cap MAT^{\lambda}_{LL2S} \neq \varnothing$

Důkaz. Jazyk $L_1 = \{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$ je bezkontextový a nedeterministický [7]. Pro tento jazyk existuje MAT^{λ}_{LL2S} gramatika $H_1 = (G, M)$, kde:

$$
\begin{aligned}
G &= (\{S, A, B, P, T, X\}, \{a, b\}, R, S), \\
R &= \{S \to \varepsilon, S \to aABT, \\
&\quad A \to aA, A \to \varepsilon, \\
&\quad B \to bBPX, B \to \varepsilon, \\
&\quad P \to b, P \to \varepsilon, X \to \varepsilon, T \to \varepsilon\}, \\
M &= \{r_1, r_2 r_5, r_3 r_5, r_4 r_6 r_7 r_9, r_7 r_9, r_4 r_6 r_8, r_9 r_8, r_9 r_{10}, r_{10}\}.
\end{aligned}
$$

Tato gramatika postupuje tak, že nejprve generuje jazyk $\{a^n b^n | n \geq 0\}$, přičemž generuje n skupin tvaru PX. Gramatika následně může provést derivaci prvního PX na terminál b, a dále pokračovat pouze maticí provádějícími derivace neterminálů v pořadí PX, kdy každá matice vyprodukuje jeden terminál b. Druhou možností je první neterminál P vymazat, kdy je dále gramatika nucena používat matici vymazávající neterminály v pořadí XP. Takto je držena informace o nutnosti vyprodukovat n symbolů b pomocí pořadí n skupin dvou neterminálních symbolů.

Deterministicky pracující modifikace MAT^{λ}_{LL2S} tedy umožnuje generovat i některé jazyky, které nejsou deterministicky zpracovatelné pomocí zásobníkového automatu. ■

Věta 7.2. $MAT^{\lambda}_{LL2S} \setminus CF \neq \varnothing$

Důkaz. Do třídy deterministických MAT^{λ}_{LL2S} gramatik patří i gramatika $H_2 = (G, M)$, kde:

$$\begin{aligned} G &= (\{S, A, B, C\}, \{a, b\}, R, S), \\ R &= \{S \to aABC, A \to aA, A \to \varepsilon, B \to bB, B \to \varepsilon, C \to cC, C \to \varepsilon\}, \\ M &= \{r_1 r_4 r_6, r_2 r_4 r_6, r_3 r_5 r_7\}. \end{aligned}$$

Tato gramatika generuje řetězce jazyka $L_2 = \{a^n b^n c^n | n \geq 1\}$ takovým způsobem, kdy každá matice změní řetězec obsahující tři skupiny m stejných po sobě jdoucích terminálů na tři skupiny o délce $m+1$, nebo ukončí derivaci. Tento jazyk nenáleží do třídy bezkontextových jazyků, a proto není možné tento jazyk generovat pomocí maticových gramatik, které aplikují pravidlo vždy na nejlevější neterminál. Modifikace MAT^{λ}_{LL2S} tedy umožnuje deterministicky generovat některé jazyky, které nelze pracovat pomocí žádného zásobníkového automatu.

Jazyk L_2 nenáleží do třídy bezkontextových jazyků. ■

Mějme jazyk $L_3 = \{ww^r | w \in \{a, b\}^+\}$, který je nedeterministickým bezkontextovým jazykem.

Věta 7.3. $CF \setminus MAT^{\lambda}_{LL2S} \neq \varnothing$

Důkaz. MAT^{λ}_{LL2S} gramatika postupuje zleva doprava a produkční pravidlo generuje právě jeden terminál řetězce na levé straně produkovaného řetězce.

L_3 je bezkontextový jazyk pro který neexistuje žádná deterministická MAT^{λ}_{LL2S}. Tato gramatika nemá dostatečné prostředky pro nalezení středu řetězce. ■

Věta 7.4. $MAT^{\lambda}_{LL2S} \subset MAT^{\lambda}_{LL2R}$.

Důkaz. Důkaz tohoto tvrzení můžeme rozdělit na dvě části. První část je důkaz tvrzení $MAT^{\lambda}_{LL2S} \subseteq MAT^{\lambda}_{LL2R}$. Každá gramatika MAT^{λ}_{LL2S} je zároveň MAT^{λ}_{LL2R} gramatikou.

Druhou částí je pak důkaz tvrzení $MAT^{\lambda}_{LL2R} \setminus MAT^{\lambda}_{LL2S} \neq \varnothing$. Pro jazyk L_3 existuje MAT^{λ}_{LL2R} gramatika $H_3 = (G, M)$, kde:

$$\begin{aligned} G &= (\{S, A, B\}, \{a, b\}, R, S) \\ R &= \{S \to aAB, S \to bAB, A \to aA, A \to bA, A \to \varepsilon, B \to Ba, B \to Bb, B \to \varepsilon\} \\ M &= \{r_1 r_6, r_2 r_7, r_3 r_6, r_4 r_7, r_5 r_8\} \end{aligned}$$

Tato gramatika postupuje tak, že střídavě uplatňuje dvojice derivací pomocí pravidel používajících pouze nejlevější a pouze nejpravější terminál. Poté ukončí derivaci pomocí ε-pravidel.

Existuje tedy jazyk, který není generovatelný pomocí MAT^{λ}_{LL2S}, ale je generovatelný pomocí MAT^{λ}_{LL2R} gramatiky. ■

Věta 7.5. $CS \not\subseteq MAT^{\lambda}_{LL2R}$.

Důkaz. Jazyk L_4 patří do třídy kontextových jazyků. Neexistuje žádná MAT^{λ}_{LL2R} gramatika G taková, aby přijímala jazyk $L_4 = \{ww | w \in \{a, b\}^*\}$. MAT^{λ}_{LL2R} gramatika nemůže v takovémto jazyku asociovat pravidlo s terminálem, které není prvním nebo posledním uvnitř řetězce. ■

Kapitola 8

Implementace

V této kapitole se věnuji popisu implementace navrženého syntaktického analyzátoru pro deterministické gramatiky typu MAT^{λ}_{LL2S} a MAT^{λ}_{LL2R}. Pro implementaci byl použit funkcionální programovací jazyk Haskell 2010. Tento jazyk byl zvolen pro možnost konzistentní reprezentace specifikované syntaktické analýzy, díky principu líného vyhodnocování.

Výsledkem implementace je konzolová aplikace, provádějící syntaktickou analýzu vstupního řetězce pomocí specifikované gramatiky. Pokud vstupní řetězec náleží specifikovanému jazyku, pak je výstupem textová reprezentace derivačního stromu vstupního řetězce. V opačném případě je pak výstupem bližší specifikace chyby, ke které došlo během běhu aplikace.

8.1 Specifikace gramatiky

Gramatiku je možno definovat pomocí jednoduchého konfiguračního souboru, který je rozdělen na jednotlivé sekce. Sekce `OVERALL` uvádí typ specifikované gramatiky, použité neterminály a startovací symbol. Za terminál je vždy implicitně považován každý jeden znak vstupního řetězce. Sekce `RULES` specifikuje jednotlivá pravidla gramatiky a sekce `MATRICES` pak seskupuje pravidla do jednotlivých matic.

8.2 Rozvržení aplikace

Aplikace je rozdělena do několika modulů, které reprezentují jednotlivé logické celky aplikace.

Modul `Lexer` provádí převod vstupního řetězce na jednotlivé lexémy.

Modul `Grammar` je zodpovědný za čtení definice gramatiky, podle které bude dále probíhat syntaktická analýza. Tento modul zajišťuje pouze omezenou kontrolu vstupní gramatiky.

Modul `Automaton` zajišťuje vytvoření řídící strukturu gramatického automatu na základě specifikované gramatiky. Tuto strukturu je možné pro demonstrační účely převést do formátů `svg` a `dot` nebo vykreslit přímo na obrazovku Xserveru. V takovém případě musí být Xserver již spuštěn a musí být dostupný. V tomto modulu je na základě vytvořeného gramatického automatu ověřena determinističnost specifikované gramatiky. Tato kontrola je provedena na základě splnění podmínek determinističnosti pro každý uzel automatu.

Proces syntaktické analýzy zajišťuje modul `Parser`, který analyzuje vstupní řetězec pomocí gramatického automatu s cílem vytvořit derivační strom, který tomuto řetězci odpovídá.

8.3 Zpracování chyb

K chybám může docházet ve všech částech zpracování vstupu aplikace. K těmto chybám může docházet z důvodů chybného zápisu gramatiky do konfiguračního souboru, specifikování nedeterministické gramatiky nebo vstupu řetězce, který nenáleží do specifikovaného jazyka.

Převod gramatiky na gramatický automat ani následná syntaktická analýza není tolerantní vůči nastalým chybám, které jsou tak považovány za kritické. Vzniklá chyba způsobí ukončení aplikace. Chybové hlášení se pak snaží uživateli přiblížit důvod nastalé chyby.

8.4 Testování

Testování se primárně zaměřuje na funkci syntaktického analyzátoru. Pro účely testování bylo vytvořeno několik gramatik obou implementovaných modifikací. Tyto gramatiky ověřují základní vlastnosti syntaktického analyzátoru. Pro každou gramatiku pak byla vytvořena sada testovacích řetězců s cílem ověřit správnost rozhodnutí, jestli daný řetězec patří do specifikovaného jazyka či nikoliv.

Kapitola 9

Závěr

Bezkontextové gramatiky mají své nezastupitelné místo ve specifikaci počítačových systémů. Tato práce se zabývala maticovými gramatikami založenými právě na bezkontextových gramatikách a dále na maticových gramatikách založenou syntaktickou analýzou.

V této souvislosti byl představen gramatický automat. Tento umožňuje maticové gramatiky reprezentovat bez ztráty struktury dané gramatiky a dále umožňuje zkoumat vlastnosti této gramatiky pomocí grafových algoritmů.

Pomocí gramatického automatu pak byla popsána syntaktická analýza jedné známé modifikace maticových gramatik využívající nejlevějších derivací. Tato modifikace využívá predikce produkovaných neterminálů pro umožnění deterministické syntaktická analýza.

V této práci byly prezentovány dvě nové modifikace maticových gramatik využívajících bezkontextová pravidla. Obě tyto modifikace se zaměřují na možnost deterministického přijímání nejen bezkontextových jazyků pomocí nejlevějších derivací.

Nevýhodou obou těchto modifikací je nutnost vyhledávat symboly v řetězcích. Toto má za následek zvýšení časové složitosti syntaktické analýzy. Pro zavedené modifikace maticových gramatik byl vytvořen algoritmus, umožňující provádět syntaktickou analýzu, pracující shora dolů.

Bylo dokázáno, že tyto nově uvedené modifikace umožňují, i při podmínce deterministického běhu syntaktické analýzy, definovat některé nedeterministické bezkontextové jazyky a některé jazyky, které nejsou bezkontextové.

Pro nové modifikace pak byl v poslední fázi vytvořen demonstrační syntaktický analyzátor, který umožňuje definovat a použít vlastní maticovou gramatiku.

Další vývoj práce by mohl zahrnovat další upřesnění síly navržených modifikací, na základě porovnání s jinými, lépe známými, systémy, či s obdobnými modifikacemi provedenými nad jinou formou řízených gramatik. Mohly by být vytvořeny další modifikace, které zavádějí jinou formu restrikce použitých pravidel, za účelem dosažení možnosti deterministického zpracování pro větší třídu jazyků.

Zajímavou problematiku poskytuje také uvedený gramatický automat. Bylo by možno definovat jeho vyjadřovací sílu s ohledem na různá omezení jeho jednotlivých součástí. Dále by bylo možno ukázat syntaktickou analýzu jiných typů řízených gramatik pomocí tohoto systému.

Literatura

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

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

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

[4] Cormen, T. H.; aj.: *Introduction to Algorithms, Third Edition*. The MIT Press, třetí vydání, 2009, ISBN 978-0-262-03384-8.

[5] Dassow, J.; Fernau, H.; Păun, G.: On The Leftmost Derivation In Matrix Gammars. *International Journal of Foundations of Computer Science*, ročník 10, č. 01, 1999: s. 61–79, doi:10.1142/S012905419900006X.

[6] Dassow, J.; Păun, G.: *Regulated rewriting in formal language theory*. EATCS monographs on theoretical computer science, Springer-Verlag, 1989, ISBN 9783540514145.

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

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

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

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

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

Příloha A

Přílohy

A.1 Obsah CD

Přiložené CD obsahuje:

- text písemné zprávy ve formátu PDF,
- zdrojový text písemné zprávy ve formátu LaTeX,
- zdrojové kódy navrženého syntaktického analyzátoru,
- přeložený syntaktický analyzátor ve formátu PE32 spustitelného souboru,
- vzorové příklady testovacích vstupů syntaktického analyzátoru,
- dokumentaci syntaktického analyzátoru ve formátu HTML.

A.2 Příklad definice gramatiky

Příklad definice maticové gramatiky pro jazyk $\{a^n b^n c^n | n \geq 1\}$ z důkazu věty 7.2.

```
[OVERALL]
type: MATllType2S
nonterminals: S,A,B,C
start: S
[RULES]
r1: S -> "a"ABC
r2: A -> "a"A
r3: A -> eps
r4: B -> "b"B
r5: B -> eps
r6: C -> "c"C
r7: C -> eps
[MATRICES]
m1: [r1,r4,r6]
m2: [r2,r4,r6]
m3: [r3,r5,r7]
```