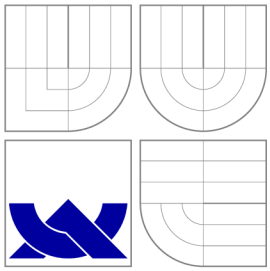
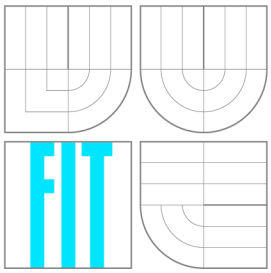


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA MODIFIKOVANÝCH ZÁSOBNÍKOVÝCH AUTOMATECH

PARSING BASED ON MODIFIED PUSHDOWN AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. DAVID PLUHÁČEK
AUTHOR

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

BRNO 2007

Zadání diplomové práce

Řešitel	**Pluháček David, Bc.**
Obor	Informační systémy
Téma	**Syntaktická analýza založená na modifikovaných zásobníkových automatech**
Kategorie	Překladače

Pokyny:

1. Dle pokynů vedoucího se seznamte detailně s modifikovanými zásobníkovými automaty a jejich vlastnostmi
2. Dle pokynů vedoucího nastudujte nové vlastnostnosti těchto automatů
3. Navrhněte jednoduchou metodu obecné syntaktické analýzy, která je založena na modifikovaných zásobníkových automatech.
4. Studujte užití metody syntaktické analýzy navržené v předchozích bodech. Zaměřte se na překladače programovacích jazyků. Navrhněte vhodný programovací jazyk a sestrojte jeho syntaktický analyzátor, který provádí sytaktickou analýzu na základě této metody. Testujte výsledný syntaktický analyzátor.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna, A: Automata and Languages, Springer, London, 2000

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních 3 bodů zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese
`http://www.fit.vutbr.cz/info/szz`

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí	**Meduna Alexander, prof. RNDr., CSc.**, UIFS FIT VUT
Datum zadání	28. února 2007
Datum odevzdání	22. května 2007

Licenční smlouva

Licenční smlouva v kompletním znění je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Výňatek z licenční smlouvy:

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zvěřejněním díla v databázi přístupné v mezinárodní síti:

   ☐ ihned po uzavření této smlouvy

   ■ 1 rok po uzavření této smlouvy

   ☐ 3 roky po uzavření této smlouvy

   ☐ 5 let po uzavření této smlouvy

   ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací).

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Abstrakt

Práce prezentuje nové modely formálních jazyků, m-omezenou stavovou gramatiku a hluboký zásobníkový automat. Uvádí jejich základní definice, vzájemnou ekvivalenci, a charakteristiku jazyků, jež popisují. Následně je představena metoda syntaktické analýzy, založená na těchto nástrojích. Ta vychází z obdobné metody používané u bezkontextových jazyků, tzv. analýzy řízené LL tabulkou. V závěru práce je popsán postup implementace syntaktického analyzátoru, založeného na této metodě.

Klíčová slova

formální jazyk, hluboký zásobníkový automat, stavová gramatika, syntaktická analýza, LL tabulka, LLd tabulka

Abstract

The thesis introduces new models for formal languages, the m-limited state grammar and the deep pushdown automaton. Their basic definitions are presented, so is their mutual equivalence and the characteristics of the language family they describe. Following, a parsing method based on these models is presented. The method is an extension of a similar method used for context-free languages, the table driven parsing. The final part of the thesis describes the implementation of a parser based on the method.

Keywords

formal language, deep pushdown automaton, state grammar, parsing, LL table, LLd table

Citace

David Pluháček: Syntaktická analýza založená na modifikovaných zásobníkových automatech, diplomová práce, Brno, FIT VUT v Brně, 2007

Syntaktická analýza založená na modifikovaných zásobníkových automatech

Prohlášení

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

.....
David Pluháček
22. května 2007

Poděkování

Děkuji prof. RNDr. Alexandru Medunovi, CSc. za odbornou pomoc a ochotu při konzultacích.

© David Pluháček, 2007.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **4**

2 Formální jazyky **6**
2.1 Úvod . . . 6
2.2 Základní pojmy . . . 6
2.3 Reprezentace jazyků . . . 7
2.3.1 Popis jazyka gramatikou . . . 7
2.3.2 Klasifikace jazyků . . . 7
2.4 Modely bezkontextových jazyků . . . 8
2.4.1 Bezkontextová gramatika . . . 8
2.4.2 Zásobníkový automat . . . 8

3 Hluboké zásobníkové automaty **11**
3.1 Úvod . . . 11
3.2 M-omezená stavová gramatika . . . 11
3.2.1 Přípravná označení . . . 11
3.2.2 Stavová gramatika . . . 12
3.3 Hluboký zásobníkový automat . . . 13
3.3.1 Formální definice . . . 13
3.3.2 Konfigurace hlubokého automatu . . . 14
3.3.3 Typy přechodů . . . 14
3.3.4 Hloubka automatu . . . 15
3.3.5 Vlastnosti hlubokých automatů . . . 15
3.3.6 Ekvivalence PDA_{deep} a m-omezených gramatik . . . 16
3.3.7 Deterministické hluboké automaty . . . 16
3.3.8 Označení tříd jazyků hlubokých automatů . . . 16
3.3.9 Příklady . . . 17

4 Syntaktická analýza **19**
4.1 Úvod . . . 19
4.2 Nástroje bezkontextové syntaktické analýzy . . . 19
4.2.1 Množina $First$ . . . 20
4.2.2 LL gramatika . . . 20
4.2.3 Převod obecné bezkontextové gramatiky na LL . . . 20

4.2.4 Faktorizace . . . 20
4.2.5 Odstranění levé rekurze . . . 21
4.2.6 Množina Follow . . . 21
4.2.7 Množina Empty . . . 21
4.2.8 Množina Predict . . . 21
4.2.9 LL tabulka . . . 22
4.3 Syntaktická analýza jazyků hlubokých automatů . . . 22
4.3.1 Množina First . . . 23
4.3.2 LL_d gramatika . . . 24
4.3.3 Faktorizace . . . 24
4.3.4 Odstranění levé rekurze . . . 25
4.3.5 Množina Follow . . . 25
4.3.6 Množina Empty . . . 26
4.3.7 Množina *Predict* . . . 26
4.3.8 LL_d tabulka . . . 27
4.3.9 Omezení použití LL_d tabulky . . . 27
4.4 Tvorba LLd tabulky pro jazyk $a^n b^n c^n$ . . . 27
4.4.1 Stavová gramatika jazyka $a^n b^n c^n$ . . . 28
4.4.2 Množiny *First* gramatiky G_1 . . . 28
4.4.3 Množiny *Empty* gramatiky G_1 . . . 28
4.4.4 Množiny *Follow* gramatiky G_1 . . . 28
4.4.5 Množiny Predict gramatiky G_1 . . . 29
4.4.6 LL_d tabulka pro gramatiku G_1 . . . 30
4.4.7 Převod stavové gramatiky na hluboký automat . . . 30

5 Implementace **31**
5.1 Jazykové konstrukce . . . 31
5.1.1 Deklarace a inicializace proměnných . . . 31
5.1.2 Příkaz print_fancy . . . 32
5.2 Důkaz ne-bezkotextovosti představených jazyků . . . 32
5.2.1 Pumping lemma pro bezkontextové jazyky . . . 32
5.2.2 Důkaz nebezkontextovosti jazyka $a^n b^n c^n$ . . . 33
5.2.3 Důkaz nebezkontextovosti jazyka $a^n b^m c^n$ . . . 33
5.3 Postup implementace . . . 34
5.4 Analyzátor jazyka deklarace globálních proměnných . . . 34
5.4.1 Hluboký zásobníkový automat . . . 34
5.4.2 LL_d tabulka . . . 35
5.4.3 Demonstrace analýzy s využitím LL_d tabulky . . . 35
5.5 Požadavky na program . . . 36
5.6 Analýza požadavků . . . 36
5.7 Návrh . . . 37
5.7.1 Lex . . . 37
5.7.2 Automaton . . . 38
5.7.3 Presenter . . . 39

5.7.4 Parser 39
5.8 Testování 40
5.9 Vývojové prostředí 40

6 Závěr **41**

A Návod k obsluze programu **44**
A.1 Kompilace 44
A.2 Vstupní data 44
A.2.1 Přijaté vstupy 44
A.3 Výstup 44

Kapitola 1

Úvod

Tato práce vychází z konceptu hlubokého zásobníkového automatu, jak byl představen v [2]. Hluboké zásobníkové automaty jsou de facto určitým zobecněním standardního zásobníkového automatu. Ten, jak známo, je schopen přepisovat pouze první neterminální symbol na svém zásobníku. Hluboký zásobníkový automat rozšiřuje jeho schopnosti o možnost přepisu obecně n-tého neterminálu na zásobníku. Pořadí přepisovaného neterminálu je přidáno jako přirozené číslo do pravidel automatu. Vzhledem k tomu, že automaty jsou konečné struktury, a množina přepisovacích pravidel je tedy konečná, existuje vždy maximální takové číslo v rámci automatu. Toto číslo označujeme jako hloubku automatu. Vždy je možné sestavit automat "o jedničku hlubší", a hluboké automaty tak tedy tvoří nekonečnou hierarchii. Tato hierarchie automatů popisuje odpovídající hierarchii jazyků. Jak je dokázáno v [2], tato nekonečná hierarchie jazyků zaujímá prostor mezi jazyky bezkontextovými a kontextovými. Hluboký zásobníkový automat hloubky 1, který je prakticky totožný s klasickým zásobníkovým automatem, popisuje třídu bezkontextových jazyků. Každé další zvýšení hloubky automatu pak rozšiřuje třídu přijímaných jazyků, ale nikdy nedosáhneme přijetí celé třídy kontextových jazyků.

Úvodní kapitoly práce připomínají základní pojmy z teorie formálních jazyků. Na ty pak plynule navazuje představení hlubokých automatů a stavových gramatik. Stavové gramatiky stojí na pomezí gramatik a automatů, protože kromě obvyklých derivačních pravidel zavádí do výpočtu i stavové řízení, tj. přechody mezi stavy. Omezením jejich derivačních pravidel tak, aby každá derivace přepisovala najvýše m-tý neterminál zleva, získává [2] tzv. m-omezenou stavovou gramatiku. Analogicky jako u hlubokých automatů je možné m-omezené gramatiky uspořádat do nekonečné hierarchie. Nejdůležitějším výsledkem [2] pak je důkaz vzájemné ekvivalence těchto nástrojů, tj. že pro každou m-omezenou stavovou gramatiku G existuje zásobníkový automat M hloubky m, takový, že $L(G) = L(M)$.

Následující kapitola se zabývá možnostmi využití hlubokých automatů k syntaktické analýze. Nejprve je popsána metoda syntaktické analýzy bezkontextových jazyků s využitím LL tabulky, tak jak je uvedena v [1]. Na základě této metody je pak sestavena její varianta, která umožňuje syntaktickou analýzu s využitím hlubokých automatů. Metoda zachovává princip řízení výpočtu automatu pomocí tabulky, zde nazvané LL_d. Tabulka určuje, které pravidlo automatu použít, na základě znalosti příštího vstupního symbolu, aktuálního stavu automatu a prvního neterminálu na zásobníku. Výsledkem této kapitoly je algoritmizo-

vatelný postup sestavení LL_d tabulky pro danou stavovou gramatiku.

V kapitole o implementaci je nejdříve tento teoretický model uveden do souvislosti s praxí. Jsou představena dvě možná rozšíření programovacích jazyků o konstrukce, jež nejsou popsatelné bezkontextovými prostředky. Pomocí lemmatu o vkládání je dokázána bezkontextovost těchto jazykových konstrukcí. Současně jsou předvedeny omezené stavové gramatiky, resp. hluboké automaty, které je popisují. Následuje popis praktické realizace analyzátoru jednoho z těchto rozšíření, konkrétně jazyka izomorfního s jazykem $L = \{x | x = a^n b^n c^n, n \geq 0\}$. Je uveden seznam požadavků na program, postup analýzy a návrhu a konečně i popis samotné implementace. Zdrojové kódy programu je možné v elektronické podobě nalézt na přiloženém CD.

Závěrečná kapitola shrnuje dosažené výsledky, jak v teoretické tak implementační rovině. Popisuje vlastnosti třídy jazyků, jež je možné analyzovat pomocí hlubokého automatu spojeného s LL_d tabulkou. Uvádí vlastnosti i omezení představené metody, a navrhuje možné směry navazujícího či souvisejícího výzkumu. V samotném závěru je uvedena možná návaznost popsané analytické metody na praktické využití.

Tato práce plynule navazuje na výsledky semestrálního projektu. V jeho rámci byly zpracovány její úvodní kapitoly, tj. úvod do teorie formálních jazyků a představení pojmů hlubokého zásobníkového automatu a omezené stavové gramatiky. Součástí semestrálního projektu byl také návrh jazykových rozšíření programovacích jazyků, která by nenáležela k jazykům bezkontextovým.

Na tyto základy jsem navázal vytvořením metody sytaktické analýzy, popsané výše. Metoda vychází z nástrojů pro syntaktickou analýzu bekontextových jazyků, jež byly převzaty z [1]. Upravený syntaktický analyzátor byl následně implementován; postup implementace je také obsahem jedné z kapitol. Kromě těchto hlavních témat byl semestrální projekt doplněn o další dílčí úpravy, například důkazy bezkontextovosti jazykových rozšíření pomocí lemmatu o vkládání.

Kapitola 2

Formální jazyky

2.1 Úvod

Pro úvodní seznámení s problematikou nyní uvedeme některé základní definice a pojmy z teorie formálních jazyků, které jsou relevantní pro tuto práci. Kromě základů, jako je *abeceda* a *formální jazyk* půjde zejména o popis zásobníkového automatu a souvisejících pojmů.

2.2 Základní pojmy

Abecedou Σ rozumíme množinu symbolů.

Řetězec u nad abecedou Σ je posloupnost symbolů této abecedy. Tedy

$$u = \{a_1 a_2 \dots a_n, \forall i, i \in \{1 \dots n\} : a_i \in \Sigma\}$$

Samozřejmě každý jednotlivý symbol je také řetězcem. Zavádíme speciální **prázdný řetězec**, který není tvořen žádnými symboly. Obvykle jej označujeme ε.

Délku řetězce u označujeme $|u|$; udává počet symbolů tvořících řetězec u.

Binární operaci $\cdot$ nazýváme **zřetězení** neboli **konkatenace**. Pro každé dva řetězce nad abecedou Σ je definována takto: $u = u_1 \dots u_n,\ v = v_1 \dots v_m, \forall i \in \{1 \dots n\} : u_i \in \Sigma\ \forall j \in \{1 \dots m\} : v_j \in \Sigma) : u \cdot v = u_1 \dots u_n v1 \dots vm$. Platí $\varepsilon \cdot u = u = u \cdot \varepsilon$. Pokud nemůže dojít k omylu, obvykle při zápisu symbol pro zřetězení vynecháváme a píšeme přímo uv.

Struktura Σ^* se nazývá **iterací** množiny Σ. Formálně jde o volný monoid s operací konkatenace, jehož (nekonečná) nosná množina obsahuje všechny řetězce vzniklé konkatenací symbolů z abecedy Σ. Neutrálním prvkem je prázdný řetězec ε.

Σ^n je tzv. **n-tá iterace** množiny Σ. Jde o množinu všech řetězců, vzniklých konkatenací symbolů z abecedy Σ, jejichž délka je nejvýše n.

Každou množinu $L \subseteq \Sigma^*$ nazveme **formální jazyk** nad abecedou Σ.

2.3 Reprezentace jazyků

Jak je vidět z předchozího, formální jazyky jsou de facto množiny řetězců. Tyto množiny je samozřejmě možné popsat čistě matematickým způsobem. Například zápis $L = \{u | \, u \in \{a, bb\}\}$ definuje jazyk nad abecedou $\Sigma = \{a, b\}$, který obsahuje řetězce a a bb. Pro reálnou práci s jazyky ale tento suchý popis není dostačující. Potřebujeme způsob, jakým budeme jazyk reprezentovat - *modelovat*. Proto byly zavedeny nástroje, které takové modelování umožňují.

2.3.1 Popis jazyka gramatikou

Gramatika je nejobecnějším modelem jazyků. Dále uváděné nástroje (zejména automaty) jsou omezeny pouze na popis některé třídy jazyků.

Gramatika je čtveřice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina tzv. *neterminálních* symbolů
- Σ je konečná množina tzv. *terminálních* symbolů
- P je konečná podmnožina kartézského součinu $(N \cup \Sigma)^* \, N \, (N \cup \Sigma)^* \times (N \cup \Sigma)^*$
- $S \in N$ je tzv. počáteční neterminální symbol.

Neterminální symboly obvykle označujeme velkými písmeny latinské abecedy, terminální pak písmeny malými.
Množinu P nazýváme **množinou (sadou) přepisovacích pravidel**. Její prvky obvykle namísto tvaru $(\alpha, \beta) \in P$ zapisujeme $\alpha \to \beta$. Řetězec α nazveme **levou stranou** přepisovacího pravidla, řetězec β pak jeho **pravou stranou**.
Základní princip gramatiky spočívá v tom, že se z počátečního neterminálu S snažíme postupnou aplikací přepisovacích pravidel vytvořit (*vyderivovat*) řetězec terminálních symbolů. Všechny takto vytvořitelné řetězce pak tvoří jazyk gramatiky G.

2.3.2 Klasifikace jazyků

Jedním ze základních pilířů teorie formálních jazyků je takzvaná Chomského hierarchie jazyků. Je to zavedený systém, který rozděluje jazyky do několika tříd. Kritériem tohoto rozdělení je tvar přepisovacích pravidel gramatiky, která daný jazyk popisuje. Chomského hierarchii znázorňuje tabulka 2.1.

Jak je vidět v tabulce 2.1, *vyšší* třída jazyka znamená přísnější omezení gramatiky. Obecně platí [1]:

$$L_3 \subset L_2 \subset L_1 \subset L_0$$

Existují také jazyky, které nejsou ani typu 0. Jedná se například o jazyky rekurzivně nevyčíslitelné.

Typ jazyka	Název gramatiky	Tvar pravidel
Typ 0	Obecná gramatika	$\alpha \to \beta$
Typ 1	Kontextová gramatika	$\alpha A \beta \to \alpha \gamma \beta, S \to \varepsilon$
Typ 2	Bezkontextová gramatika	$A \to \gamma$
Typ 3	Pravá lineární gramatika	$A \to xB, A \to x$
Typ 3	Levá lineární gramatika	$A \to Bx, A \to x$
Typ 3	Pravá regulární gramatika	$A \to aB, A \to x, S \to \varepsilon$
Typ 3	Levá regulární gramatika	$A \to Ba, A \to x, S \to \varepsilon$

Tabulka 2.1: Chomského hierarchie jazyků

2.4 Modely bezkontextových jazyků

Jak bylo naznačeno v úvodu, tato práce se zabývá analýzou jazyků ležících mezi třídami bezkontextových a kontextových jazyků. Představíme zde tedy základní modely pro bezkontextové jazyky, tak jak jsou uvedeny v [1]. Z nich poté budeme vycházet.

2.4.1 Bezkontextová gramatika

Bezkontextová gramatika je čtveřice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina *neterminálních* symbolů
- Σ je konečná množina *terminálních* symbolů
- P je konečná podmnožina kartézského součinu $N \times \{N \cup \Sigma\}^*$
- $S \in N$ je tzv. počáteční neterminální symbol.

Vidíme, že oproti základní definici gramatiky je tvar pravidel striktněji omezen - na levé straně může stát právě jediný neterminál.

2.4.2 Zásobníkový automat

Zásobníkový automat je druhým základním modelem bezkontextových jazyků. Jedná se o nástroj, vycházející z konečného automatu. Oproti němu je doplněn o zásobník, na který může zapisovat symboly, číst symbol na vrcholu zásobníku a také tento symbol mazat.

Zásobníkový automat

Zásobníkový automat je sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- Q je konečná množina stavů
- Σ je konečná množina symbolů - vstupní abeceda
- Γ je konečná množina symbolů - zásobníková abeceda
- $R \subseteq (\Gamma\ Q\ (\Sigma \cup \{\varepsilon\})) \times (\Gamma^*\ Q)$ je přechodová funkce

- $s \in Q$ je počáteční stav
- $S \in \Gamma$ je počáteční symbol na zásobníku
- $F \subseteq Q$ je množina koncových stavů

Obvykle bývá zásobníkový automat ještě doplněn o speciální symbol #, označující dno zásobníku.

Prvky relace R nazýváme pravidly a obvykle je zapisujeme ve tvaru $Apa \rightarrow wq$, kde $A \in \Gamma, p, q \in Q, a \in (\Sigma \cup \{\varepsilon\}), w \in \Gamma^*$. Význam pravidla je následující: automat provede přechod ze stavu p do stavu q, symbol A na vrcholu zásobníku nahradí řetězcem w a ze vstupu přečte symbol a.

Konfigurace zásobníkového automatu

Uvažujme zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$. Konfigurace automatu M je řetězec

$$\chi = \Gamma^* \, Q \, \Sigma^*$$

.

Konfigurace automatu obsahuje všechny informace nutné k popisu aktuální situace v automatu: obsah zásobníku, aktuální stav a nepřečtený obsah vstupní pásky.

Přechod v zásobníkovém automatu

Uvažujme zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$. Mějme dvě konfigurace tohoto automatu, $xApay$ a $xwqy$, kde $x, w \in \Gamma^*, A \in \Gamma, p, q \in Q, a \in (\Sigma \cup \{\varepsilon\}, y \in \Sigma^*$. Dále nechť existuje pravidlo $r = Apa \rightarrow wq, r \in R$. Pak říkáme, že automat M provede **přechod** z konfigurace $xApay$ do konfigurace $xwqy$ podle pravidla r. Zapisujeme symbolicky

$$xApay \vdash xwqy \; [r]$$

popř.

$$xApay \vdash xwqy$$

Sekvence přechodů

Mějme konfiguraci χ automatu M. Definujeme, že M provede z χ do χ přechod v nula krocích. Symbolicky zapisujeme $\chi \vdash^0 \chi$.

Nechť $\chi_0, \chi_1, \ldots, \chi n$ je posloupnost konfigurací, $n \in N, n \geq 1$. Dále nechť existují pravidla $r_i \in R$ taková, že $\chi_{i-1} \vdash \chi_i[r_i] \; \forall i \in \{1, \ldots, n\}$. Jinak řečeno, konfigurace jsou "spojeny" pravidly do sekvence: $\chi_0 \vdash \chi_1[r_1] \vdash \chi_2[r_2] \cdots \vdash \chi_n[r_n]$. Potom řekneme, že automat M provede **sekvenci n přechodů** z χ_0 do χ_n. Sekvenci přechodů zapisujeme symbolicky

$$\chi_0 \vdash^n \chi_n[r_1 \ldots r_n]$$

případně

$$\chi_0 \vdash^n \chi_n$$

.

Pokud $n \geq 1$, označujeme sekvenci nenulové délky $\chi_0 \vdash^+ \chi_n$.
Pokud $n \geq 0$, označujeme sekvenci obecné délky $\chi_0 \vdash^* \chi_n$.

Jazyk přijímaný zásobníkovým automatem

Víme, že klasický konečný automat přijímá jazyk tvořený těmi řetězci, po jejichž zpracování skončí automat v koncovém stavu. Doplnění konečného automatu o záobník přineslo další možnost, jak definovat přijímaný jazyk. Jde o takzvané *přijímání prázdným zásobníkem.* V tomto případě tvoří přijímaný jazyk ty řetězce, jejichž zpracování způsobí vyprázdnění zásobníku automatu.
Je samozřejmě možné oba tyto způsoby spojit. Automat pak přijímá ty řetězce, se kterými přejde do koncového stavu a zároveň vyprázdní zásobník.

Je dokázáno[1], že všechny tři způsoby jsou ekvivalentní. Tedy, že libovolný z těchto tří typů zásobníkového automatu lze algoritmicky převést na zbývající dva tak, že výsledné automaty budou přijímat stejný jazyk.

Definujme teď všechny tři typy formálně. Vždy předpokládáme automat
$M = (Q, \Sigma, \Gamma, R, s, S, F)$.

- Jazyk, přijímaný koncovým stavem automatu M:
  $L(M)_f = \{w |\ w \in \Sigma^*, Ssw \vdash^* zf, z \in \Gamma^*, f \in F\}$.
- Jazyk, přijímaný vyprázdněním zásobníku automatu M:
  $L(M)_\varepsilon = \{w | w \in \Sigma^*, Ssw \vdash^* zf, z = \varepsilon, f \in Q\ \}$
- Jazyk, přijímaný vyprázdněním zásobníku a koncovým stavem automatu M:
  $L(M)_{f\varepsilon} = \{w | w \in \Sigma^*, Ssw \vdash^* zf, z = \varepsilon, f \in F\}$

Kapitola 3

Hluboké zásobníkové automaty

3.1 Úvod

Hluboký zásobníkový automat (*deep pushdown automaton, DPDA*) je rozšířením klasického zásobníkového automatu. Základní princip rozšíření spočívá v tom, že DPDA je schopen přepisovat nejen symbol na vrcholu zásobníku, ale také libovolný neterminál hlouběji v zásobníku.

3.2 M-omezená stavová gramatika

Než se dostaneme k samotnému pojmu hlubokého zásobníkového automatu, uvedeme nejdříve třídu gramatik, ze které hluboké automaty vychází. Jde o tzv. m-omezené stavové gramatiky.

Stavové gramatiky stojí na pomezí klasických gramatik,jak je známe z Chomského hierarchie, a automatů. Kromě obvyklých přepisovacích pravidel je totiž jejich součástí i konečná množina stavů, mezi kterými gramatika během derivace přechází.

Variantou základního modelu stavové gramatiky je m-omezená gramatika. Taková gramatika je schopná přepisovat nejvýše m-tý nejlevější neterminál v řetězci. Jak je uvedeno v ([2]), nekonečná hierarchie těchto gramatik (tedy 1-omezená, 2-omezená,...) definuje nekonečnou hierarchii jazyků. Ta zaujímá místo mezi jazyky kontextovými a bezkontextovými.

3.2.1 Přípravná označení

Před vlastní definicí zaveďme následující označení:

- $card(Q)$ je kardinalita množiny Q
- I je množina všech přirozených čísel.
- V^* je volný monoid nad nosnou množinou V s operací konkatenace (jinými slovy, jde o monoid, jehož prvky jsou všechny konečné řetězce vytvořené z prvků množiny V užitím operace konkatenace)

- ε je jednotkový prvek monoidu V^*
- homomorfismus $f \subseteq V^* \times V^*$ nazveme **kódováním**, pokud pro každé $A \in V$ platí $f(A) \in \{A, \varepsilon\}$
- $V^+ = V^* \setminus \{\varepsilon\}$ je volná pologrupa nad V s operací konkatenace
- $|w|$ označuje délku řetězce $w \in V^*$
- $alph(w)$ označuje množinu symbolů, jež se vyskytují v řetězci w
- $occur(w, W)$ pro $W \subseteq V, w \in W^*$ udává počet výskytů symbolů z W v řetězci w
- $\lfloor w, i, W \rfloor$ pro $W \subseteq V, w \in W^*, i = 1, \ldots, |w|$ označuje i-tý výskyt neterminálu z W v řetězci w. Pokud takový neterminál neexistuje, $\lfloor w, i, W \rfloor = 0$

3.2.2 Stavová gramatika

Stavová gramatika je šestice $G = (V, W, T, P, S)$.

- V je konečná abeceda
- W je konečná množina stavů
- $T \subseteq V$ je množina terminálů
- $S \in (V \setminus T)$ je startovací symbol
- $P \subseteq (W \times (V \setminus T)) \times (W \times V^+)$ je konečná relace, obvykle nazývaná **množina pravidel**.

Jednotlivé prvky relace P, takzvaná **pravidla**, obvykle namísto relačního tvaru $(q, A, p, v) \in P$ zapisujeme jako $(q, A) \to (p, v)$.

Množina ${}_G states$

Pro každé $z \in V^*$ definujeme množinu
${}_G states(z) = \{q | (q, B) \to (p, v) \in P \text{ kde } B \in (V \setminus T) \cap alph(z), v \in V^+, q, p \in W\}$.

Derivační krok a derivace

Pokud $(q, A) \to (p, v) \in P, x, y \in V^*, {}_G states(x) = \emptyset$, říkáme, že G provede **derivační krok** z (q, xAy) do (p, xvy) podle pravidla $(q, A) \to (p, v)$.
Symbolicky zapisujeme

$$(q, xAy) \Rightarrow (p, xvy) \; [(q, A) \to (p, v)]$$

.

Pokud existuje přirozené číslo n, pro které platí $occur(xA, V \setminus T) \leq n$, říkáme, že derivační krok $(q, xAy) \Rightarrow (p, xvy) \; [(q, A) \to (p, v)]$ je **n-omezený**.

Symbolicky zapisujeme $(q, xAy) \ {}_n \Rightarrow \ (p, xvy)$ $[(q, A) \rightarrow \ (p, v)]$. Pokud nemůže dojít k omylu, obvykle se zápis zjednodušuje na $(q, xAy) \ \Rightarrow \ (p, xvy)$, resp. $(q, xAy) \ {}_n \Rightarrow \ (p, xvy)$.

Obdobným způsobem jako u základního zásobníkového automatu je definice derivačního kroku $\Rightarrow$ rozšířena na posloupnost derivací délky $m \Rightarrow^m$, $m \ \geq \ 0$. Z ní je pak odvozena derivace obecné délky $\Rightarrow^*$ a derivace nenulové délky $\Rightarrow^+$.

Mějme číslo $n \ \in \ I$ a dvě dvojice $\upsilon, \ \varpi \ \in \ (W \times V^+)$. Zápisy $\upsilon \ {}_n \Rightarrow^m \ \varpi$, $\upsilon \ {}_n \Rightarrow^* \ \varpi$ a $\upsilon_n \Rightarrow^+ \ \varpi$ vyjadřují, že derivace $\upsilon \ \Rightarrow^m \ \varpi$, $\upsilon \ \Rightarrow^* \ \varpi$ a $\upsilon \ \Rightarrow^+ \ \varpi$ jsou n-**omezené**.

Množina *strings*

Zápis $strings(\upsilon \ {}_n \Rightarrow^* \ \varpi)$ označuje množinu všech řetězců, které se vyskytují v derivaci $\upsilon \ {}_n \Rightarrow^* \ \varpi$.

Jazyk n-omezené gramatiky

Jazyk gramatiky G, $L(G)$, je definován takto:

$$L(G) = \{w \in T^* | (q, S) \Rightarrow^* (p, w), q, p \in W\}$$

.

Pro každé $n \in I, n \geq 1$ dále definujeme

$$L(G, n) = \{w \in T^* | (q, S) \ {}_n \Rightarrow^* (p, w), q, p \in W\}$$

Derivace tvaru $(q, S)_n \Rightarrow^* (p, w)$, kde $p, q \in W, w \in T^*$ představuje úspěšné n-omezené generování řetězce w gramatikou G.

3.3 Hluboký zásobníkový automat

Jak již bylo zmíněno, hluboký zásobníkový automat (PDA_{deep}) vychází z klasického PDA. Stejně jako on v každém kroku výpočtu při derivaci shora dolů buďto smaže symbol z vrcholu zásobníku, nebo provede na zásobníku expanzi. Na rozdíl od klasického PDA však může expandovat symbol hlouběji v zásobníku, ne jen na vrcholu. Jinak se PDA_{deep} chová ve všech ohledech stejně jako klasický PDA.

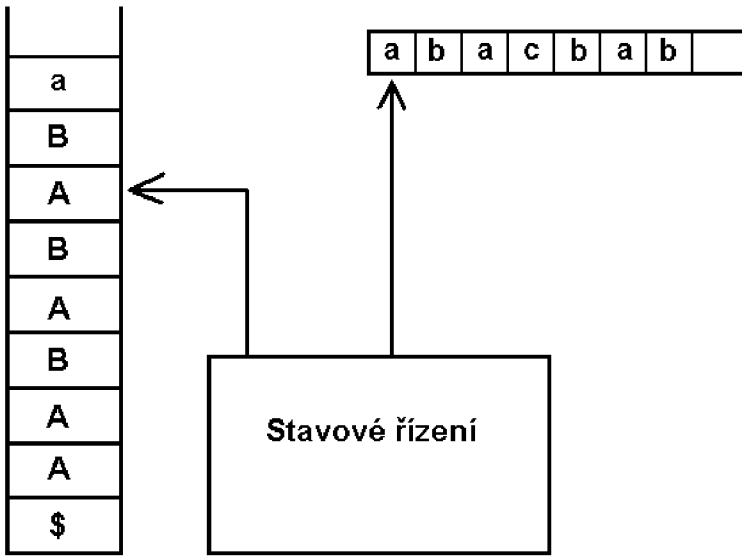
Definice tohoto automatu vychází z principu n-omezených stavových gramatik. Uvidíme, že je zde patrná korespondence: gramatice, která přepisovala nejvýše n-tý nejlevější neterminál, bude odpovídat automat, který přepisuje nejvýše n-tý neterminál pod vrcholem zásobníku.

Schematicky je hluboký automat znázorněn na obrázku 3.1.

3.3.1 Formální definice

Hluboký zásobníkový automat je sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$

- Q je konečná množina stavů
- Γ je konečná množina symbolů - zásobníková abeceda



Obrázek 3.1: Hluboký zásobníkový automat

- $\Sigma \subseteq \Gamma$ je vstupní abeceda
- množiny I (přirozená čísla),Q a Γ jsou po dvou disjunktní
- $\# \in (\Gamma \setminus \Sigma)$ je speciální symbol označující **dno zásobníku**
- $R \subseteq (I \times Q \times (\Gamma \setminus (\Sigma \cup \{\#\})) \times Q \times (\Gamma \setminus \{\#\})^+) \cup (I \times Q \times \{\#\} \times Q \times (\Gamma \setminus \{\#\})^*\{\#\})$ je konečná relace nazývaná **množina pravidel** automatu M.
- $s \in Q$ je počáteční stav
- $S \in \Gamma$ je startovací symbol zásobníku

Prvky relace R obvykle namísto $(m, q, A, p, v) \in R$ zapisujeme ve tvaru $mqA \to pv \in R$ a nazýváme je **pravidly**. Vidíme, že na rozdíl od klasického zásobníkového automatu v pravidlech přibylo číslo $m \in I$. Právě ono udává, kolikátý neterminál od vrcholu zásobníku pravidlo přepisuje.

3.3.2 Konfigurace hlubokého automatu

Konfigurace automatu M je trojice $x \in Q \times \Sigma^* \times (\Gamma \setminus \{\#\})^*\{\#\}$. Udává tedy, stejně jako u zásobníkového automatu, aktuální stav, obsah zásobníku a zbývající obsah vstupní pásky.

3.3.3 Typy přechodů

Označme χ množinu všech konfigurací automatu M a uvažujme dvě konfigurace $x, y \in \chi$. Potom říkáme, že automat M

- při přechodu z x do y **smaže** (*pop*) symbol ze zásobníku, zapsáno $x_p \Rightarrow y$, pokud $x = (q, au, az), y = (q, u, z)$, kde $a \in \Sigma, u \in \Sigma^*, z \in \Gamma^*$.
- při přechodu z x do y provede **expanzi** zásobníku, zapsáno $x_e \Rightarrow y$, pokud $x = (q, w, uAz)$, $y = (p, w, uvz), mqA \rightarrow pw \in R$, kde $q, p \in Q, w \in \Sigma^*, A \in \Gamma, u, v, z \in \Gamma^*$ a $occur(u, \Gamma \setminus \Sigma) = m - 1$

Podmínka $occur(u, \Gamma \setminus \Sigma) = m - 1$ u expanze zajišťuje, že na zásobníku přepisujeme právě m-tý neterminál.

Aby bylo jasné, že automat provede přechod $x_e \Rightarrow y$ podle pravidla $mqA \rightarrow pv$, zapisujeme $x_e \Rightarrow y[mqA \rightarrow pv]$.

Pravidlo $mqA \rightarrow pv$ nazveme **pravidlem hloubky** m. Obdobně $x_e \Rightarrow y[mqA \rightarrow pv]$ je **expanze hloubky m**.

Přechod

Automat M provede **přechod** mezi konfiguracemi x a y, psáno $x \Rightarrow y$, pokud $x_e \Rightarrow y$ nebo $x_p \Rightarrow y$.

3.3.4 Hloubka automatu

Protože množina R je konečná, je zřejmé, že vždy existuje přirozené číslo $n \in I$, které je nejmenší takové, že každé pravidlo z R je hloubky n nebo menší. Toto číslo pak nazýváme **hloubkou automatu** M. Automat M hloubky n označujeme ${}_nM$

Opět obvyklým způsobem rozšíříme přechody, tj. ${}_p \Rightarrow, {}_e \Rightarrow$ a $\Rightarrow$ pro nějaké $m \geq 0$ na sekvence přechodů délky m: ${}_p \Rightarrow^m, {}_e \Rightarrow^m$ a $\Rightarrow^m$. Zobecněním pak jsou sekvence přechodů obecné délky ${}_p \Rightarrow^*, {}_p \Rightarrow^+, {}_e \Rightarrow^*, {}_e \Rightarrow^+, \Rightarrow^*$ a $\Rightarrow^+$.

Jazyk hlubokého automatu

Nakonec definujme jazyk, přijímaný hlubokým automatem. Uvažujme automat M hloubky n, $n \in I$. Jazyk automatu ${}_nM$, $L({}_nM)$ je definován takto:

$$L({}_nM) = \{w | w \in \Sigma^*, (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#), f \in F\}$$

Jde tedy o všechny řetězce ze vstupní abecedy, se kterými automat provede výpočet z počátečního stavu s s počátečním obsahem zásobníku S a skončí v některém z koncových stavů f a s prázdným zásobníkem.

Vidíme, že zápis je prakticky stejný, jako u klasického zásobníkového automatu - liší se pouze označením hloubky. Podrobnějším zkoumáním rozšířeného modelu zjistíme, že hluboký zásobníkový automat hloubky 1 je prakticky totožný s klasickým zásobníkovým automatem - pouze u něj explicitně uvádíme hloubku.

3.3.5 Vlastnosti hlubokých automatů

V souladu s [2] zaveďme následující označení: $\mathbf{PD}_k$ označuje třídu jazyků, přijímaných hlubokým zásobníkovým automatem hloubky $k, k \geq 1$. **CF** (*context-free*) je označení třídy bezkontextových jazyků. **CS** (*context-sensitive*) označuje třídu jazyků kontextových.

Zdroj [2] uvádí několik zajímavých výsledků, které popisují vlastnosti hlubokých zásobníkových automatů.

1. ${}_{deep}\mathbf{PD}_1 = \mathbf{CF}$. Tedy, že hluboký zásobníkový automat hloubky 1 přijímá třídu bezkontextových jazyků. Tento výsledek je v souladu s tvrzením, jež bylo uvedeno výše, totiž že hluboký zásobníkový automat hloubky 1 je ekvivalentní klasickému zásobníkovému automatu (respektive jsou prakticky totožné).

2. ${}_{deep}\mathbf{PD}_k \subset_{deep} \mathbf{PD}_{k+1} \subset \mathbf{CF}$ pro libovolné $k \in I, k \geq 1$.

Důkazy obou vět je možné nalézt v [2].

Kombinace těchto výsledků popisuje zmiňovanou nekonečnou hierarchii jazyků, přijímaných hlubokými automaty. Vidíme, že třídu L_2 přijímá automat hloubky 1. Každý další krok v hloubce automatu pak jeho sílu zvýší (všimněme si, že inkluze ${}_{deep}\mathbf{PD}_k \subset_{deep} \mathbf{PD}_{k+1}$ je ostrá, tedy že vždy "přibude něco nového"), ale nikdy nedosáhneme toho, aby hluboký automat přijímal celou třídu kontextových jazyků.

3.3.6 Ekvivalence PDA_{deep} a m-omezených gramatik

Další z výsledků, uvedených v [2] tvrdí, že ke každému hlubokému zásobníkovému automatu hloubky n existuje ekvivalentní n-omezená stavová gramatika. Tedy formálně, že pro každé $n \in I, n \geq 1$ a každý hluboký zásobníkový automat ${}_nM$ existuje stavová gramatika G taková, že $L(G, n) = L({}_nM)$. Kromě formálního důkazu poskytuje [2] i praktický algoritmus převodu stavové gramatiky na hluboký automat. Hluboké automaty a omezené stavové gramatiky jsou tedy nástroji s ekvivalentní vyjadřovací silou, podobně jako zásobníkové automaty odpovídají bezkontextovým gramatikám.

3.3.7 Deterministické hluboké automaty

Stejně jako u zásobníkových automatů, i u jejich hlubokých variant je pro praktické využití vhodné zabývat se jejich deterministickými verzemi. Podle [2] můžeme u hlubokých automatů popsat hned několik variant determinismu. Základní, **striktní determinismus** je definován takto [2]:

Hluboký zásobníkový automat $M = (Q, \Sigma, \Gamma, R, s, S, F)$ je **deterministický**, pokud neexistují dvě pravidla se stejnou levou stranou. Tedy formálně, pro každé pravidlo $mqA \rightarrow pv \in R$ platí, že $card(\{mqA \rightarrow ow | mqA \rightarrow ow \in R, o \in Q, w \in \Gamma^+\} - \{mqA \rightarrow pv\}) = 0$.

3.3.8 Označení tříd jazyků hlubokých automatů

Pro zpřehlednění budou v dalším textu používána následující označení:

- L_2 v souladu se zažitou praxí označuje třídu bezkontextových jazyků
- $L_{PDAdeep}$ označuje třídu jazyků přijímaných hlubokými zásobníkovými automaty
- $L_{DPDAdeep}$ označuje třídu jazyků přijímaných striktně deterministickými hlubokými zásobníkovými automaty

3.3.9 Příklady

V následujících příkladech budeme demonstrovat, že výpočetní síla hlubokého zásobníkového automatu leží mezi třídami kontextových a bezkontextových jazyků.

Příklad 1 Uvažujme jazyk $L_1 = \{ww | w \in \{0,1\}^+\}$.
Dá se dokázat, že tento jazyk není bezkontextový. Hluboký automat, který jej přijímá, může vypadat například takto:

- $M = (Q, \Sigma, \Gamma, R, s, S, F)$
- $Q = \{s, q, j, n, f_j, f_n\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{\#, S, A, 0, 1\}$
- $F = \{f_j, f_n\}$

Přechodová funkce R je definována takto: R = { $1sS \to qAA$, $1qA \to n0A$, $2nA \to q0A$, $1qA \to j1A$, $2jA \to q1A$, $1qA \to f_j1$, $1f_jA \to f_j1$, $1qA \to f_n0$, $1f_nA \to f_n0$, } Automat přijímá koncovým stavem a prázdným zásobníkem. Demonstrujme přijetí řetězce 011011.

$(s, 011011, S\#)$
$_e \Rightarrow (q, 011011, AA\#)$ $[1sS \to qAA]$
$_e \Rightarrow (n, 011011, 0AA\#)$ $[1qA \to n0A]$
$_e \Rightarrow (q, 011011, 0A0A\#)$ $[2nA \to q0A]$
$_p \Rightarrow (q, 11011, A0A\#)$ $[pop]$
$_e \Rightarrow (j, 11011, 1A0A\#)$ $[1qA \to j1A]$
$_e \Rightarrow (q, 11011, 1A01A\#)$ $[2jA \to q1A]$
$_p \Rightarrow (q, 1011, A01A\#)$ $[pop]$
$_e \Rightarrow (f_j, 1011, 101A\#)$ $[1qA \to f_j1]$
$_e \Rightarrow (f_j, 1011, 1011\#)$ $[1f_jA \to f_j1]$
$_p \Rightarrow (f_j, 011, 011\#)$ $[pop]$
$_p \Rightarrow (f_j, 11, 11\#)$ $[pop]$
$_p \Rightarrow (f_j, 1, 1\#)$ $[pop]$
$_p \Rightarrow (f_j, \varepsilon, \#)$ $[pop]$

Příklad 2 V druhém příkladu předvedeme hluboký automat pro další klasický ne-bezkontextový jazyk, jakým je $L_2 = \{a^n b^m a^n | m \geq n \geq 1\}$.
Automat, který bude tento jazyk přijímat, může vypadat takto:

- $M = (Q, \Sigma, \Gamma, R, s, S, F)$
- $Q = \{s, q, p, f\}$
- $\Sigma = \{a, b\}$

- $\Gamma = \{\#, S, A, a, b\}$
- $F = \{f\}$

Přechodová funkce R je definována: R = { $1sS \rightarrow qAA$, $1qA \rightarrow paAb$, $2pA \rightarrow qAa$, $1qA \rightarrow faB$, $2fA \rightarrow fa$, $1fB \rightarrow fbB$, $1fB \rightarrow fb$ } Opět předvedeme funkci automatu na jednom řetězci, například $aabbbaa$.

$(s, aabbbaa, S\#)$
$_e \Rightarrow (q, aabbbaa, AA\#)$ $[1sS \rightarrow qAA]$
$_e \Rightarrow (p, aabbbaa, aAbA\#)$ $[1qA \rightarrow paAb]$
$_p \Rightarrow (p, abbbaa, AbA\#)$ $[pop]$
$_e \Rightarrow (q, abbbaa, AbAa\#)$ $[2pA \rightarrow qAa]$
$_e \Rightarrow (f, abbbaa, aBbAa\#)$ $[1qA \rightarrow faB]$
$_p \Rightarrow (f, bbbaa, BbAa\#)$ $[pop]$
$_e \Rightarrow (f, bbbaa, Bbaa\#)$ $[2fA \rightarrow fa]$
$_e \Rightarrow (f, bbbaa, bBbaa\#)$ $[1fB \rightarrow fbB]$
$_e \Rightarrow (f, bbbaa, bbbaa\#)$ $[1fB \rightarrow fb]$
$_p \Rightarrow (f, bbaa, bbaa\#)$ $[pop]$
$_p \Rightarrow (f, baa, baa\#)$ $[pop]$
$_p \Rightarrow (f, aa, aa\#)$ $[pop]$
$_p \Rightarrow (f, a, a\#)$ $[pop]$
$_p \Rightarrow (f, \varepsilon, \#)$ $[pop]$

Kapitola 4

Syntaktická analýza

4.1 Úvod

Hlavním cílem této práce je zkoumat možnosti využití hlubokých zásobníkových automatů v oblasti syntaktické analýzy. Z výsledků uvedených výše plyne, že schopnosti těchto automatů rozšiřují možnosti analyzovaných jazyků. Zároveň ale stále zachovávají poměrně striktní pravidla jejich tvorby. Jednou ze základních oblastí aplikace syntaktické analýzy vždy byla analýza programovacích jazyků, obvykle jako součást překladačů nebo interpretů. Proto i v této práci směřujeme k cíli sestavit syntaktický analyzátor takového programovacího jazyka, jenž nebude analyzovatelný běžnými bezkontextovými nástroji. Na tomto analyzátoru pak demonstrujeme vyšší sílu představovaného modelu.

4.2 Nástroje bezkontextové syntaktické analýzy

Stejně jako hluboké automaty vychází z automatů zásobníkových, vyjdeme i při tvorbě nástrojů syntaktické analýzy z již existujících prostředků, které jsou používány u bezkontextových jazyků.

Základním problémem zásobníkových automatů oproti automatům konečným je to, že neexistuje obecná převoditelnost obecného PDA na jeho deterministický ekvivalent. Deterministické zásobníkové automaty, tj. takové, které pro každou kombinaci stavu a vrcholu zásobníku obsahují nejvýše jedno pravidlo, přijímají pouze vlastní podtřídu třídy L_2. Většina prakticky použitelných bezkontextových jazyků bohužel do této třídy nepatří, a k jejich analýze je proto nutné použít jiné nástroje a metody.

Jednou ze základních takových metod je analýza shora dolů, řízená tzv. LL tabulkou. Její princip spočívá v tom, že analyzovaný jazyk popíšeme bezkontextovou gramatikou splňující jisté speciální požadavky, tzv. LL_1 gramatikou. Z pravidel této gramatiky algoritmicky odvodíme odpovídající zásobníkový automat, a také tzv. LL tabulku, která řídí činnost automatu. Toto řízení probíhá tak, že na základě aktuálního stavu automatu, obsahu zásobníku a známého příštího znaku na vstupu je jednoznačně zvoleno pravidlo pro příští krok výpočtu. Oproti klasickému PDA tedy musíme při výpočtu znát jednu informaci navíc; tou je příští znak na vstupu.

Uveďme nyní postup tvorby LL tabulky podle [1]; na jeho základě nakonec sestavíme obdobnou tabulku pro analýzu jazyků hlubokých automatů.

4.2.1 Množina $First$

Definice: Mějme bezkontextovou gramatiku $G = (N, T, P, S)$. Pro každý řetězec $x \in (N \cup T)^*$ definujeme množinu $First(x) = \{a | a \in T, x \Rightarrow^* ay; y \in (N \cup T)^*\}$

Množina $First$ tedy pro každý řetězec symbolů obsahuje všechny terminální symboly, jež je možné z řetězce x v dané gramatice vyderivovat na nejlevější pozici.

4.2.2 LL gramatika

Intuitivně tedy můžeme uvažovat takto: pokud bude na vrcholu zásobníku nějaký neterminál A, porovnáme množiny $First$ všech pravých stran pravidel tvaru $A \to x$. Zvolíme pak to z pravidel, v jehož množině $First$ se nachází příští znak ze vstupu. Jinými slovy, zvolíme takové pravidlo, jehož pravá strana začíná žádaným terminálem - příštím znakem vstupu. Pokud takové pravidlo bude vždy nejvýše jedno, dostáváme jednoznačný postup výpočtu. Gramatiku, pro kterou toto platí pak nazveme LL gramatikou. Formálně tedy:

Definice: Buď $G = (N, T, P, S)$ bezkontextová gramatika bez ε-pravidel. G nazveme **LL gramatikou**, pokud platí: $\forall a \in T, \forall A \in N$ existuje nejvýše jedno pravidlo tvaru $A \Rightarrow X_1 X_2 \dots X_n \in P$ takové, že $a \in First(X_1 X_2 \dots X_n)$.

Jak je vidět z definice, tento postup je možno použít pouze u gramatik, v nichž se nevyskytují ε-pravidla, tj. pravidla tvaru $A \to \varepsilon$. Jak uvidíme, toto omezení má svoje následky.

4.2.3 Převod obecné bezkontextové gramatiky na LL

Třída jazyků LL gramatik je sice nadtřídou jazyků L_{DPDA}, stále ale zůstává vlastní podtřídou celé L_2. Přesto je ale možné alespoň některé z bezkontextových gramatik převést na LL gramatiky. K tomuto převodu existují dvě základní techniky, a to faktorizace a odstranění levé rekurze.

4.2.4 Faktorizace

Vzpomeňme si na základní požadavek na LL gramatiku: množiny $First$ dvou levých stran pravidel, jež mají stejnou pravou stranu, nesmí obsahovat společný prvek. Jinými slovy, nesmíme být schopni ze stejného neterminálu vyderivovat pomocí různých pravidel dva řetězce se stejným nejlevějším terminálem.

Pokud v gramatice existují pravidla tvaru

$$A \to xy_1, A \to xy_2 \dots A \to xy_n$$

je tento požadavek evidentně porušen. Proto tato pravidla nahradíme následujícími:

$$A \to xA', A' \to y_1, A' \to y_2, \dots, A' \to y_n$$

A' je nový neterminál, který přidáme do množiny N. Pokud tyto úpravy v gramatice zapříčinily vznik nových nevyhovujících pravidel, postup opakujeme.

4.2.5 Odstranění levé rekurze

Gramatiku nazveme **levě rekurzivní**, pokud obsahuje pravidla tvaru

$$A \to Ax, A \to y$$

. Tento typ pravidel je nevhodný pro konstrukci zásobníkového automatu: nejdříve totiž generuje všechny potřebné kopie řetězce x, a až poté umožní jejich případné mazání tím, že "odstraní z cesty" neterminál A. Z hlediska implementace je ale mnohem vhodné, pokud můžeme každou kopii x zpracovat samostatně a až poté generovat další. Proto pravidla zmíněného tvaru nahradíme takto:

$$A \to yA', A' \to xA', A' \to \varepsilon$$

kde A' je opět nový neterminál.

Vidíme, že tento proces nevyhnutelně zavádí do gramatiky ε-pravidla, jejichž výskyt byl v původní verzi LL gramatiky zakázán. Je proto nutné se s nimi nějakým způsobem vypořádat. K tomu slouží další nástroje, s jejichž pomocí budeme schopni sestavit řídicí LL tabulku i pro gramatiky s ε-pravidly.

4.2.6 Množina Follow

Definice: Buď $G = (N, T, P, S)$ bezkontextová gramatika. Pro každý neterminál $A \in N$ definujeme množinu

$$Follow(A) = \{a | a \in T, S \Rightarrow^* xAay, x, y \; in (N \cup T)^*\} \cup \$: S \Rightarrow^* xA, x \in (N \cup T)^*$$

Množina $Follow(A)$ tedy ke každému neterminálu A obsahuje terminální symboly, které se při derivaci mohou vyskytnout přímo za A. Intuitivně cítíme, že takováto množina se může hodit, pokud A bude na levé straně některého ε-pravidla.

Symbol \$ označuje koncovou značku. Pro zjednodušení přepokládáme, že tato značka se nachází za každým vstupním řetězcem (není tedy součástí vstupu, ale indikuje konec).

4.2.7 Množina Empty

Definice: Buď $G = (N, T, P, S)$ bezkontextová gramatika. Pro řetězec $x \in (N \cup T)^*$ definujeme množinu $Empty(x) = \{\varepsilon\}$, pokud $x \Rightarrow^* \varepsilon$. Jinak $Empty(x) = \emptyset$.

Množina $Empty$ tedy pro každý řetězec symbolů obsahuje ε, pokud je z tohoto řetězce možné vyderivovat prázdný řetězec. Prakticky tedy má smysl uvažovat o množině $Empty$ pouze u řetězců tvořených neterminály.

4.2.8 Množina Predict

Na základě předchozích dvou množin vytvoříme obdobu množiny $First$ pro gramatiky s ε-pravidly - množinu $Predict$.

Definice: Mějme bezkontextovou gramatiku $G = (N, T, P, S)$. Pro každé pravidlo $A \to x \in P$ definujeme množinu $Predict(A \to x)$ takto:

- pokud $Empty(x) = \varepsilon$, pak $Predict(A \to x) = First(x) \cup Follow(A)$
- pokud $Empty(x) = \emptyset$, pak $Predict(A \to x) = First(x)$

Množina *Predict* je tedy pro pravidla, jejichž pravá strana není redukovatelná až na ε, "zpětně kompatibilní" s množinou *First*. Pro ostatní pravidla přidává k "očekávaným" terminálům obsah *Follow* levé strany.

4.2.9 *LL* tabulka

Na základě takto sestavených množin můžeme vytvořit tzv. *LL* tabulku, pomocí níž bude řízena činnost zásobníkového automatu. V záhlaví tabulky bude seznam všech terminálů ze vstupní abecedy spolu s koncovým znakem \$. První sloupec bude obsahovat seznam všech neterminálů. Na průsečíky pak umístíme pravidla gramatiky podle následujícího algoritmu

Algoritmus

Mějme bezkontextovou gramatiku $G = (N, T, P, S)$. Pro každý neterminál $A \in N$ a terminál $a \in T$ umístíme do buňky (A, a) pravidlo $A \to X_1 X_2 \dots X_n \in P$, pokud
$a \in Predict(A \to X_1 X_2 \dots X_n)$. Jinak zůstane buňka (A, a) prázdná.

Příklad

Tabulka 4.1 ilustruje možnou podobu LL tabulky. Z důvodu přehlednosti a úspory místa se obvykle pravidla do tabulky nazapisují celá, pouze se uvádí nějaká forma odkazu, např. číslo pravidla.

Uvažujme blíže nespecifikovanou bezkontextovou gramatiku, kde $T = \{a, b, c\}$, $N = \{S, A, B\}$ a $card(P) = 4$.

	a	b	c	\$
S	1			
A		2		
B			3	4

Tabulka 4.1: Ukázka LL tabulky

Podle této tabulky se pro derivaci neterminálu S použije pravidlo 1, pokud je na vstupu symbol a. Neterminál A přepíšeme podle pravidla 2, pokud je na vstupu b. Konečně neterminál B přepisujeme podle pravidla 3, pokud je na vstupu c, a podle pravidla 4, pokud je na vstupu koncová značka \$.

4.3 Syntaktická analýza jazyků hlubokých automatů

V předchozí části byly připomenuty nástroje, s jejichž pomocí je možné realizovat syntaktickou analýzu shora dolů u bezkontextových jazyků. V následujícím textu se pokusím pomocí úprav těchto nástrojů navrhnout obdobnou metodu, již by bylo možno použít

k analýze jazyků přijímaných hlubokými automaty, resp. popisovaných stavovými gramatikami. Hlavním rozdílem mezi bezkontextovými a stavovými gramatikami je přidání množiny stavů; to také bude hlavním projevem následujících úprav.

4.3.1 Množina First

Definice: Mějme stavovou gramatiku $G = (V, W, T, P, S)$. Pro každou dvojici $(s, x), s \in W, x \in V^*$ definujeme množinu $First((s, x)) = \{a | a \in T, (s, x) \Rightarrow^* (q, ay); s, q \in W, y \in V^*\}$

Množina $First(s, x)$ je tedy nyní vázána nejen na řetězec, ale i na stav gramatiky. Obsahuje pak všechny terminály, jež mohou být vyderivovány z řetězce x na nejlevější pozici, ovšem pouze pokud derivace začíná ve stavu s.

Příklad

Uvažujme jednoduchou stavovou gramatiku s následujícími pravidly:

- $(s, S) \rightarrow (q, aA)$
- $(q, A) \rightarrow (s, aA)$
- $(s, A) \rightarrow (s, b)$

Pokud by šlo o klasickou bezkontextovou gramatiku (tj. odmyslíme si přechody mezi stavy), vypadala by např. množina $First(A)$ takto: $First(A) = a, b$. Protože ale jde o gramatiku stavovou, vypadají množiny $First$ takto:

- $First(s, S) = a$
- $First(q, A) = a$
- $First(s, A) = b$

Algoritmus

Množina všech řetězců $x \in V^*$ je obecně nekonečná, není tedy možné pro všechny takové řetězce množinu $First$ vypočítat. Protože ale uvažujeme gramatiku bez $\varepsilon - pravidel$, z každého symbolu na nejlevější pozici libovolného řetězce se časem stane terminál, a na zbytku řetězce tedy nezáleží. Počet symbolů v množině V je konečný, stejně jako počet stavů gramatiky. Můžeme tedy algoritmicky sestavit množiny $First(s, x)$ pro všechna $s \in W, x \in V$ takto:

- $\forall s \in W, a \in T : First(s, a) = a$
- opakuj následující postup, dokud je možné změnit některou množinu First:
- pokud existuje pravidlo $(s, A) \rightarrow (p, X_1 X_2 \dots Xn)$, přidej $First(p, X_1)$ k $First(s, A)$.

4.3.2 LL_d gramatika

Definice: Buď $G = (V, W, T, P, S)$ stavová gramatika bez ε-pravidel. G nazveme LL_d gramatikou, pokud platí: $\forall s \in W, \forall a \in T, \forall A \in (V \setminus T)$ existuje nejvýše jedno pravidlo tvaru $s, A \Rightarrow p, X_1X_2 \dots X_n \in P$ takové, že $a \in First((p, X_1X_2 \dots X_n))$.

Příklad

Gramatika, popsaná v odstavci 4.3.1, tuto definici splňuje.

4.3.3 Faktorizace

Stejně jako u bezkontextových gramatik, i nyní potřebujeme nástroje pro převod obecných stavových gramatik na LL_d. Faktorizaci u těchto gramatik budeme provádět takto:

Pokud v gramatice existují pravidla tvaru

$$(s, A) \to (p, xy_1), (s, A) \to (p, xy_2) \dots (s, A) \to (p, xy_n)$$

nahradíme je následujícími pravidly:

$$(s, A) \to (p', xA'), (p', A') \to (p, y_1), (p', A') \to (p, y_2), \dots, (p', A') \to (p, y_n)$$

A' je nový neterminál, který přidáme do množiny N. Zavedeme také nový stav $p' \in W$. Pokud by tyto úpravy v gramatice zapříčinily vznik nových nevyhovujících pravidel, postup opět opakujeme.

Základní myšlenka je tedy obdobná jako u bezkontextové gramatiky: zbytečně časné rozvětvení derivace na začátku spojíme pomocí nového neterminálu a stavu. Rozvětvení pravidel pak vedeme až v nezbytně nutném místě, tj. z tohoto nového stavu a neterminálu.

Příklad

Uvažujme následující pravidla stavové gramatiky:

- $(s, S) \to (q, aA)$
- $(s, S) \to (q, aB)$
- $(q, A) \to (f, a)$
- $(q, B) \to (f, b)$

Faktorizujeme první dvě pravidla, výsledek bude vypadat takto:

- $(s, S) \to (q', a\langle AB \rangle)$
- $(q', \langle AB \rangle) \to (q, A)$
- $(q', \langle AB \rangle) \to (q, B)$
- $(q, A) \to (f, a)$
- $(q, B) \to (f, b)$

4.3.4 Odstranění levé rekurze

Stavovou gramatiku nazveme levě rekurzivní, pokud obsahuje pravidla tvaru

$$(p, A) \rightarrow (p, Ax), (p, A) \rightarrow (q, y)$$

Levé rekurze se opět potřebujeme "zbavit" s ohledem na budoucí převod na automat. Proto zmíněná pravidla nahradíme takto:

$$(p, A) \rightarrow (p', yA'), (p', A') \rightarrow (p', xA'), (p', A' \rightarrow (q, \varepsilon)$$

kde A' je opět nový neterminál a p' nový stav gramatiky.

Příklad

Uvažujme následující pravidla stavové gramatiky:

- $(s, S) \rightarrow (s, A)$
- $(s, A) \rightarrow (s, Aa)$
- $(s, A) \rightarrow (f, a)$

Po odstranění levé rekurze bude gramatika vypadat takto:

- $(s, S) \rightarrow (s, A)$
- $(s, A) \rightarrow (s', aA')$
- $(s', A') \rightarrow (s', aA')$
- $(s', A') \rightarrow (f, \varepsilon)$

Opět jsme tedy ve stavu, kdy převodem směrem k LL_d gramatice v množině pravidel vznikají ε-pravidla. Ta v ní mohou samozřejmě být i z jiného důvodu, například pro snadnější pochopení gramatiky nebo proto, že bez nich je tvorba gramatiky pro daný jazyk obtížná či nemožná. Proto pokračujeme v úpravách směrem k obecnější LL_d gramatice, která umožní existenci ε-pravidel.

4.3.5 Množina Follow

Definice: Buď $G = (V, W, T, P, S)$ stavová gramatika. Pro každý neterminál $A \in N$ definujeme množinu

$$Follow(A) = a | a \in T, (q_0, S) \Rightarrow^* (p, xAay), x, y\ in V^*, p \in W \cup \{\$: (s, S) \Rightarrow^* (p, xA), x \in V^*,$$

$$p \in W\} \cup \{\$: (s, S) \Rightarrow^* (p, xAy), x \in V^*, p \in W, Empty(p, y) = \{\varepsilon\}\}$$

Množina $Follow(A)$ tedy opět pro každý neterminální symbol A obsahuje ty terminální symboly, jež jej mohou bezprostředně následovat. Pokud se neterminál A vyskytuje v některém pravidle zcela vpravo, může být následován také koncovou značkou \$. Oproti

bezkontextovým gramatikám je nutné také přidat třetí podmínku, a tou je možnost vymazání řetězce za neterminálem A. Ve stavových gramatikách je totiž možné, že při výpočtu bude pravidly vynuceno takové pořadí kroků, jež nejdříve přepíše či vymaže neterminál A, a až poté smaže řetězec y. V takovém případě evidentně symbol \$ také patří do množiny $Follow(A)$.

4.3.6 Množina Empty

Definice: Buď $G = (V, W, T, P, S)$ stavová gramatika. Pro dvojici $(p, x), p \in W, x \in V^*$ definujeme množinu $Empty(p, x) = \varepsilon$, pokud $(p, x) \Rightarrow^* (q, \varepsilon), q \in W$. Jinak $Empty(x) = \emptyset$. Intuitivně tedy množina $Empty(p, x)$ udává, zda je možné při derivaci ze stavu p řetězec x zredukovat až na prázdný.

Příklad

Uvažujme následující pravidla stavové gramatiky:

- $(s, S) \rightarrow (q, A)$
- $(q, A) \rightarrow (p, B)$
- $(q, A) \rightarrow (r, B)$
- $(p, B) \rightarrow (f, \varepsilon)$
- $(r, B) \rightarrow (q, aA)$

Uveďme množiny $Empty(p, x)$ pro všechna $x, |x| = 1$

- $Empty(r, B) = \emptyset$
- $Empty(p, B) = \varepsilon$
- $Empty(q, A) = \varepsilon$
- $Empty(s, S) = \varepsilon$

Vidíme, že pro neterminál B už obsah množiny $Empty$ závisí na stavu gramatiky: ze stavu r vede přechod pouze s generováním terminálu a, proto je $Empty(r, B) = \emptyset$ ale $Empty(p, B) = \varepsilon$.

4.3.7 Množina *Predict*

Nakonec opět všechny připravené množiny složíme dohromady a získáme tak poslední nástroj potřebný k syntaktické analýze - množinu $Predict$ a následně pak LL_d tabulku.

Definice: Mějme stavovou gramatiku $G = (V, W, T, P, S)$. Pro každé pravidlo $(p, A) \rightarrow (q, x) \in P$ definujeme množinu $Predict((p, A) \rightarrow (q, x))$ takto:

- pokud $Empty(q, x) = \varepsilon$, pak $Predict((p, A) \rightarrow (q, x)) = First(q, x) \cup Follow(A)$
- pokud $Empty(q, x) = \emptyset$, pak $Predict((p, A) \rightarrow (q, x)) = First(x)$

4.3.8 LL_d tabulka

Analogicky jako u bezkontextových gramatik užijeme vytvořených množin *Predict* k sestavení tabulky, která bude řídit výpočet hlubokým automatem.

V záhlaví tabulky jsou uvedeny všechny vstupní symboly včetně koncové značky \$. Každý řádek pak odpovídá jedné kombinaci stav-neterminál, tj. $(s, A), s \in W, A \in (V \setminus T)$. Na průsečík $(s, A) \times a$ umístíme odkaz na pravidlo $(s, A) \to (p, x)$, pokud $a \in Predict((s, A) \to (p, x))$. Jinak zůstane buňka nevyplněna.

Ukázka LL_d tabulky

LL_d tabulka tedy může mít například podobu tabulky 4.2 (uvažujme obecnou nespecifikovanou stavovou gramatiku).

	a	b	c	\$
(q,S)	1			
(q,A)		2		
(q,B)			3	4

Tabulka 4.2: Ukázka LLd tabulky

Tabulka udává, že při vstupu a se má ve stavu q pro přepis neterminálu S použít pravidlo 1, ostatní vstupy vedou k chybě. Další řádek je obdobný. Poslední řádek udává, že ve stavu q se pro přepis neterminálu B použije pravidlo 3, pokud je na vstupu symbol c, a pravidlo 4, pokud je na vstupu symbol \$, tj. konec vstupu.

4.3.9 Omezení použití LL_d tabulky

Klasická LL tabulka v bezkontextové syntaktické analýze předpokládá, že vždy, když přepisujeme na zásobníku neterminál, můžeme se "podívat" na první znak vstupního řetězce a podle jeho příslušnosti do množin *Predict* daného neterminálu rozhodnout, které pravidlo použijeme. V případě hlubokého automatu ale narážíme na problém v případech, kdy přepisujeme neterminál hlouběji v zásobníku. Nad ním může být obecně libovolné množství terminálních symbolů, "blokovaných" jedním či více neterminály. Tyto terminály tak nemůžeme ze zásobníku odstranit, a tedy ani dočíst vstup až do místa, kde bychom se potřebovali podívat na vstupní symbol rozhodující pro aktuálně přepisovaný neterminál. Proto při této metodě syntaktické analýzy zavedeme následující omezení: LL_d tabulku použijeme pouze v případech, kdy přepisujeme první neterminál na zásobníku. Z toho plyne, že automat ${}_nM$ musí být pro expanze hloubky $2 \ldots n$ deterministický.

4.4 Tvorba LLd tabulky pro jazyk $a^n b^n c^n$

Ukažme nyní prakticky průběh tvorby LL_d tabulky pro analýzu jazyka $a^n b^n c^n$. Nejdříve nalezneme stavovou gramatiku, která tento jazyk popisuje. Jednou takovou gramatikou je tato:

4.4.1 Stavová gramatika jazyka $a^n b^n c^n$

$G_1 = (V, W, T, P, S)$
$V = \{S, A, C, a, b, c\}$
$W = \{s, q, f\}$
$T = \{a, b, c\}$
$P = \{$

- $(s, S) \rightarrow (s, AC)$
- $(s, A) \rightarrow (q, aAb)$
- $(q, C) \rightarrow (s, cC)$
- $(s, A) \rightarrow (f, \varepsilon)$
- $(f, C) \rightarrow (f, \varepsilon)$

$\}$

Gramatiku ponecháme bez důkazu, že opravdu generuje jazyk $a^n b^n c^n$. Postupně sestrojíme množiny $First$, $Follow$, $Predict$ a nakonec i vlastní LL_d tabulku.

4.4.2 Množiny $First$ gramatiky G_1

$First(s, a) = \{a\}$ $First(q, a) = \{a\}$ $First(f, a) = \{a\}$	$First(s, b) = \{b\}$ $First(q, b) = \{b\}$ $First(f, b) = \{b\}$	$First(s, c) = \{c\}$ $First(q, c) = \{c\}$ $First(f, c) = \{c\}$
$First(s, S) = \{a\}$	$First(s, A) = \{a\}$	$First(q, C) = \{c\}$
$First(s, AC) = a$		

Tabulka 4.3: Množiny $First$ gramatiky G_1

4.4.3 Množiny $Empty$ gramatiky G_1

$Empty(s, S) = \{\varepsilon\}$ $Empty(q, C) = \emptyset$ $Empty(s, AC) = \{\varepsilon\}$	$Empty(s, A) = \{\varepsilon\}$ $Empty(f, C) = \varepsilon$

Tabulka 4.4: Množiny $Empty$ gramatiky G_1

4.4.4 Množiny $Follow$ gramatiky G_1

$Follow(S) = \{\$\}$ $Follow(C) = \{\$\}$	$Follow(A) = \{b, \$\}$

Tabulka 4.5: Množiny $Follow$ gramatiky G_1

4.4.5 Množiny Predict gramatiky G_1

Na základě přechozích výsledků sestrojíme množiny $Predict$ pro všechna pravidla gramatiky, tj.

1. $(s, S) \rightarrow (s, AC)$
2. $(s, A) \rightarrow (q, aAb)$
3. $(q, C) \rightarrow (s, cC)$
4. $(s, A) \rightarrow (f, \varepsilon)$
5. $(f, C) \rightarrow (f, \varepsilon)$

1. $(s, S) \rightarrow (s, AC)$: $Empty(s, AC) = \{\varepsilon\} \Rightarrow$
   $Predict((s, S) \rightarrow (s, AC)) = First(s, AC) \cup Follow(S) = \{a, \$\}$
2. $(s, A) \rightarrow (q, aAb)$: $Empty(s, aAb) = \emptyset \Rightarrow$
   $Predict((s, A) \rightarrow (q, aAb)) = First(q, aAb) = \{a\}$
3. $(q, C) \rightarrow (s, cC)$: $Empty(s, cC) = \emptyset \Rightarrow$
   $Predict((q, C) \rightarrow (s, cC)) = First(s, cC) = \{c\}$
4. $(s, A) \rightarrow (f, \varepsilon)$: $Empty(f, \varepsilon) = \{\varepsilon\} \Rightarrow$
   $Predict((s, A) \rightarrow (f, \varepsilon)) = First(f, \varepsilon) \cup Follow(A) = \{b, \$\}$
5. $(f, C) \rightarrow (f, \varepsilon)$: $Empty(f, \varepsilon) = \{\varepsilon\} \Rightarrow$
   $Predict((f, C) \rightarrow (f, \varepsilon)) = First(f, \varepsilon) \cup Follow(C) = \{\$\}$

Pravidlo	$Predict$
$(s, S) \rightarrow (s, AC)$	$\{a, \$\}$
$(s, A) \rightarrow (q, aAb)$	$\{a\}$
$(q, C) \rightarrow (s, cC)$	$\{c\}$
$(s, A) \rightarrow (f, \varepsilon)$	$\{b, \$\}$
$(f, C) \rightarrow (f, \varepsilon)$	$\{\$\}$

Tabulka 4.6: Množiny $Predict$ gramatiky G_1

4.4.6 LL_d tabulka pro gramatiku G_1

Získané výsledky zaneseme do tabulky 4.7. Ta se svou formou podobá klasické LL tabulce, pouze v prvním sloupci namísto neterminálů uvádíme dvojice (stav, neterminál). Opět ale jde o levé strany pravidel gramatiky.

	a	b	c	$
(s,S)	1			1
(s,A)	2	4		4
(q,C)			3	
(f,C)				5

Tabulka 4.7: LL_d tabulka granatiky G_1

4.4.7 Převod stavové gramatiky na hluboký automat

Posledním krokem před implementací je převod pravidel gramatiky na pravidla automatu a jejich "spárování" v LL_d tabulce. Prakticky postup předvedeme v následující kapitole, věnované implementaci. Na tomto místě uveďme převodní algoritmus, převzatý z [2].

Mějme stavovou gramatiku $G = (V, W, T, P, S)$ a číslo $n \geq 1$. Vytvořme množinu neterminálních symbolů $N = V \setminus T$. Nakonec definujme homomorfismus f nad $(\{\#\} \cup V)^*$ takto: $\forall A \in N \cup \{\#\}, fA = A. \forall a \in T, f(a) = \varepsilon$.

Zavedeme hluboký zásobníkový automat M hloubky n:

$$_nM = (Q, T, \{\#\} \cup V, R, s, S, \{\$\})$$

kde $Q = \{S, \$\} \cup \{\langle P, U\rangle | p \in W, u \in N^*\{\#\}^n, |u| \leq n\}$ a množinu pravidel R sestavíme pomocí následujících kroků:

1. pro každé pravidlo gramatiky G tvaru $(p, S) \to (q, x) \in P, p, q \in W, x \in V^+$ přidáme do množiny R pravidlo $1sS \to \langle p, S\rangle S$.

2. pokud v G existuje pravidlo $(p, A) \to (q, x) \in P, \langle p, uAv\rangle \in Q, p, q \in W, A \in N, x \in V^+, u \in N^*, v \in N^*\{\#\}^*, |uAv| = n, p \notin G_s tates(u)$, pak k množině R přidáme pravidlo $|uA|\langle p, uAv\rangle A \to \langle q, \mathbf{prefix}(uf(x)v, n)\rangle x$.

3. pro $A \in N, p \in W, u \in N^*, v \in \{\#\}^*, |uv| \leq n-1, p \notin G_s tates(u)$ přidáme do R pravidla $|uA|\langle p, uv\rangle A \to \langle p, uAv\rangle A$ a $|uA|\langle p, uv\rangle \# \to \langle p, uv\#\rangle \#$.

4. pro každý stav $q \in W$ přidáme do množiny R pravidlo $1\langle q, \#^n\rangle \# \to \$\#$.

Základní myšlenka převodního algoritmu spočívá v tom, že vytvářený automat si ve svých stavech "zapamatuje" potřebný počet neterminálů na vrcholu zásobníku. Při výpočetním kroku simuluje příslušnou derivaci, a přejde do stavu odpovídajícího novým neterminálům na vrcholu. Poté co je vygenerován terminální řetězec, automat vyprázdní zásobník a přejde do koncového stavu.

Kapitola 5

Implementace

V této kapitole popíšeme způsob, jakým byl implementován syntaktický analyzátor příjímající jazyky hlubokých automatů, resp. stavových gramatik. Implementujeme automat pro konkrétní jazyk, vycházející z konstrukcí použitelných v zápisu zdrojového kódu programovacích jazyků.

5.1 Jazykové konstrukce

Jako inspiraci použijeme modelový programovací jazyk IFJ06, uvedený v [3]. Tento jazyk je ve své původní podobě bezkontextový. Na jeho základě ale můžeme popsat některé jazykové konstrukce, jež jej posunou ze třídy bezkontextových jazyků. Rozšířený jazyk nazveme ΩΘ3.

5.1.1 Deklarace a inicializace proměnných

V programovacím jazyce ΩΘ3 lze deklarovat globální proměnné. Deklarace je nepovinná, pokud je uvedena, vyskytuje se na začátku programu. Oproti jazyku IFJ06 rozšíříme deklaraci o povinnnou inicializaci hodnot. Deklarace tedy bude mít tvar

```
\'var\' typ^{n} id^{n} ival^{n} \';\'
```

Za klíčovým slovem `var` následuje seznam datových typů `typ` jednotlivých proměnných, po něm seznam identifikátorů `id` a nakonec seznam inicializačních hodnot `ival`. Všechny tři seznamy musí být shodné délky.

Příklad

Ukázka korektní deklarace globálních proměnných v jazyce ΩΘ3:

```
var int int real gwen pavka rada 1 42 0.56
```

Jak vidíme, jde o obdobu jazyka $L_a = \{a^n b^n c^n | n \in I, n \geq 1\}$, což je jeden z klasických kontextových (ale ne bezkontextových) jazyků. Seznam inicializačních hodnot není zcela čisté c^n - dva literály jsou celočíselné, třetí desetinný. Pro účely syntaktické analýzy tedy pravděpodobě bude nutné zavést obecný typ tokenu "literál" a podrobnější kontrolu

provádět buďto zanořením syntaktického analyzátoru, nebo ji ponechat v rukou analýzy sémantické.

5.1.2 Příkaz print_fancy

Druhou úpravou jazyka IFJ06 je zavedení nového příkazu `print_fancy`. Tento příkaz umožňuje mírně formátovaný tisk hodnot proměnných na výstup. Jeho syntaxe je definována takto:

`'print_fancy' id^{n} coords^{m} color^{n}`, $m \geq n \geq 1$

Z klíčovým slovem `print_fancy` následuje seznam identifikátorů proměnných `id`. Druhou částí je seznam `coords`, který obsahuje seznam dvojic celočíselných souřadnic ve tvaru `[x,y]`. Délka tohoto seznamu musí být minimálně stejná jako délka seznamu identifikátorů. Sémantika je následující: prvních n položek seznamu souřadnic určuje souřadnice vykreslení hodnot odpovídajících proměnných. Pokud je seznam delší, vykreslují se další hodnoty opět od začátku seznamu proměnných, až do konce seznamu souřadnic. Poslední částí je seznam barev ve tvaru `#00ff00`, kterými budou jednotlivé hodnoty vykresleny. Jeho délka musí odpovídat délce seznamu proměnných.

Příklad

Ukázka korektního použití příkazu `print_fancy` v jazyce ΩΘ3:

```
print_fancy gwen pavka [1,4] [4,5] [4,12] [6,8] #32FE56 #FFCEC9
```

Tento zápis způsobí dvojí výpis proměnné `gwen` na souřadice `[1,4]` a `[4,12]` v barvě `#32FE56`. Proměnná `pavka` bude vypsána na souřadnice `[4,5]` a `[6,8]` v barvě `#FFCEC9`.

Tento jazyk je (až na samotný identifikátor `print_fancy`) izomorfní s jazykem $L_b = \{a^n b^m c^n | m, n \in I, m \geq n \geq 1\}$. Připomeňme, že v kapitole o hlubokých automatech byl představen automat, který právě tento jazyk přijímá.

5.2 Důkaz ne-bezkotextovosti představených jazyků

V této sekci ukážeme, že žádný z jazyků není bezkontextový, a má tedy smysl se jimi zabývat. K důkazu užijme tzv. Pumping lemma neboli lemma o vkládání.

5.2.1 Pumping lemma pro bezkontextové jazyky

Nechť L je bezkontextový jazyk. Pak existuje konstanta k taková, že libovolný řetězec $z, z \in L, |z| \geq k$ je možno napsat ve tvaru

$$z = uvwxy, vx \neq \varepsilon, |vwx| \leq k$$

a pro všechna $i \geq 0$ platí $uv^i wx^i y \in L$.

Uvedeného lemmatu nyní využijeme k vlastnímu důkazu tvrzení, že představené jazyky nejsou bezkontextové, a má tedy smysl zkoumat jejich přijímání hlubokými zásobníkovými automaty.

5.2.2 Důkaz nebezkontextovosti jazyka $a^n b^n c^n$

Tvrzení: $a^n b^n c^n \notin L_2$ Důkaz: Zvolme řetězec $z = a^k b^k c^k$. Je zřejmé, že $z \in L$, i že $|z| \geq k$. Ukážeme, že neexistuje dělení $z = uvwxy, vx \neq \varepsilon, |vwx| \leq k$ takové, aby pro všechna $i \geq 0$ platilo $z' = uv^i wx^i y \in L$.

Možná dělení:

- $v = a^m, w = a^*, x = a^n$. Zvolme i = 2. Pak $z' = a^{k+m+n} b^k c^k \notin L$,
  protože $|vx| \geq 1 \Rightarrow m + n \geq 1 \Rightarrow |a^{k+m+n}| \neq |b^k|$
- $v = a^m, w = a^* b^*, x = b^n$. Zvolme i = 2. Pak $z' = a^{k+m} b^{k+n} c^k \notin L$,
  protože $|vx| \geq 1 \Rightarrow m \geq 1 \vee n \geq 1 \Rightarrow |a^{k+m}| \neq |c^k| \vee |b^{k+n}| \neq |c^k|$.
- $v = b^m, w = b^*, x = b^n$. Zvolme i = 2. Pak $z' = a^k b^{k+m+n} c^k \notin L$,
  protože $|vx| \geq 1 \Rightarrow m + n \geq 1 \Rightarrow |b^{k+m+n}| \neq |a^k|$
- $v = b^m, w = b^* c^*, x = c^n$. Zvolme i = 2. Pak $z' = a^k b^{k+m} c^{k+n} \notin L$,
  protože $|vx| \geq 1 \Rightarrow m \geq 1 \vee n \geq 1 \Rightarrow |b^{k+m}| \neq |a^k| \vee |c^{k+n}| \neq |a^k|$.
- $v = c^m, w = c^*, x = c^n$. Zvolme opět i = 2. Pak $z' = a^k b^k c^{k+m+n} \notin L$,
  protože $|vx| \geq 1 \Rightarrow m + n \geq 1 \Rightarrow |c^{k+m+n}| \neq |a^k|$

Ve všech případech dojdeme ke sporu s tvrzením PL, jazyk tedy není bezkontextový.

5.2.3 Důkaz nebezkontextovosti jazyka $a^n b^m c^n$

Tvrzení: $a^n b^m c^n, m \geq n \geq 1 \notin L_2$ Důkaz: Zvolme řetězec $z = a^k b^k c^k$. Je zřejmé, že $z \in L, |z| \geq k$. Ukážeme, že neexistuje dělení $z = uvwxy, vx \neq \varepsilon, |vwx| \leq k$ takové, aby pro všechna $i \geq 0$ platilo $z^{'} = uv^i wx^i y \in L$.

Možná dělení:

- $v = a^m, w = a^*, x = a^n$. Zvolme i = 2. Pak $z^{'} = a^{k+m+n} b^k c^k \notin L$,
  protože $|vx| \geq 1 \Rightarrow m + n \geq 1 \Rightarrow |a^{k+m+n}| \neq |c^k|$
- $v = a^m, w = a^* b^*, x = b^n$. Zvolme i = 0. Pak $z^{'} = a^{k-m} b^{k-n} c^k \notin L$,
  protože $|vx| \geq 1 \Rightarrow m \geq 1 \vee n \geq 1 \Rightarrow |a^{k-m}| \neq |c^k| \vee |b^{k-n}| < |c^k|$.
- $v = b^m, w = b^*, x = b^n$. Zvolme i = 0. Pak $z^{'} = a^k b^{k-m-n} c^k \notin L$,
  protože $|vx| \geq 1 \Rightarrow m + n \geq 1 \Rightarrow |b^{k-m-n}| < |a^k|$
- $v = b^m, w = b^* c^*, x = c^n$. Zvolme i = 0. Pak $z^{'} = a^k b^{k-m} c^{k-n} \notin L$,
  protože $|vx| \geq 1 \Rightarrow m \geq 1 \vee n \geq 1 \Rightarrow |b^{k-m}| < |a^k| \vee |c^{k-n}| \neq |a^k|$.
- $v = c^m, w = c^*, x = c^n$. Zvolme i = 2. Pak $z^{'} = a^k b^k c^{k+m+n} \notin L$,
  protože $|vx| \geq 1 \Rightarrow m + n \geq 1 \Rightarrow |c^{k+m+n}| \neq |a^k|$

Ve všech případech opět dojdeme ke sporu s tvrzením PL, ani tento jazyk tedy není bezkontextový.

5.3 Postup implementace

K implementaci syntaktického analyzátoru použijeme tento postup: nejprve pomocí metod uvedených v kapitole o syntaktické analýze sestrojíme LL_d tabulku pro daný jazyk. Užitím algoritmu uvedeného v odstavci 4.4.7 převedeme gramatiky na hluboké automaty. Spojením automatu a příslušející LL_d tabulky získáme deterministický, a tedy implementovatelný, nástroj.

Vytvoříme program, který načte ze standardního vstupu textový soubor, obsahující fragment zdrojového kódu. Analyzátor se pokusí tento kód přijmout jako řetězec z příslušného jazyka, tj. buďto $a^n b^n c^n$ nebo $a^n b^m c^n$. Poté analyzátor na standardní výstup vypíše informaci, zda vstupní řetězec patřil do daného jazyka.

Součástí programu bude jednoduchý lexikální analyzátor vstupních symbolů.

5.4 Analyzátor jazyka deklarace globálních proměnných

Upravená deklarace globálních proměnných v představeném jazyce je izomorfní s jazykem $L_{nnn} = a^n b^n c^n$. Můžeme tedy využít dříve uvedené výsledky, zejména gramatiku G_1, popisující tento jazyk, a příslušnou LL_d tabulku.

5.4.1 Hluboký zásobníkový automat

Použitím algoritmu z 4.4.7 převedeme gramatiku G_1 na automat. Připomeňme, že gramatika vypadá takto: $G_1 = (V, W, T, P, S)$
$V = \{S, A, C, a, b, c\}$
$W = \{s, q, f\}$
$T = \{a, b, c\}$
$P = \{$

- $(s, S) \rightarrow (s, AC)$
- $(s, A) \rightarrow (q, aAb)$
- $(q, C) \rightarrow (s, cC)$
- $(s, A) \rightarrow (f, \varepsilon)$
- $(f, C) \rightarrow (f, \varepsilon)$

$\}$ Výsledkem převodu je následující automat

- ${}_{deep}M = (Q, \Sigma, \Gamma, R, s, S, F)$
- $Q = \{s, \langle s, S\# \rangle, \langle s, AC \rangle, \langle q, AC \rangle, \langle f, C\# \rangle, \langle f, \#\# \rangle, \$\}$
- $\Sigma = \{a, b, c\}$
- $\Gamma = \{\#, S, A, C, a, b, c\}$
- $F = \{\$\}$

Přechodová funkce R je definována:

R = {
1. $1sS \rightarrow \langle s, S\#\rangle S$,
2. $1\langle s, S\#\rangle S \rightarrow \langle s, AC\rangle AC$,
3. $1\langle s, AC\rangle A \rightarrow \langle q, AC\rangle aAb$,
4. $2\langle q, AC\rangle C \rightarrow \langle s, AC\rangle cC$,
5. $1\langle s, AC\rangle A \rightarrow \langle f, C\#\rangle \varepsilon$,
6. $1\langle f, C\#\rangle C \rightarrow \langle f, \#\#\rangle \varepsilon$,
7. $1\langle f, \#\#\rangle \# \rightarrow \$\#$,
}

Vidíme, že pravidla automatu na druhé až šesté pozici jsou analogiemi pravidel gramatiky. První pravidlo slouží pouze k "odstartování" výpočtu a je vždy provedeno jako první. Posledním pravidlem pak automat přejde do koncového stavu.

5.4.2 LL_d tabulka

Dále budeme potřebovat LL_d tabulku pro tento jazyk, vytvořenou v odstavci 4.4.6. Přečíslujeme pravidla a přejmenujeme stavy, aby tabulka odpovídala automatu.

	a	b	c	$
($\langle s, S\#\rangle$,S)	2			2
($\langle s, AC\rangle$,A)	3	5		5
($\langle q, AC\rangle$,C)			4	
($\langle f, C\rangle$, C)				6

Tabulka 5.1: LL_d tabulka pro jazyk L_{nnn}

5.4.3 Demonstrace analýzy s využitím LL_d tabulky

Demonstrujme nyní prakticky přijetí řetězce z jazyka $L = \{a^n b^n c^n\}$ pomocí uvedené tabulky a hlubokého automatu.

Zvolme například řetězec *aabbcc*. Ten zřejmě náleží do jazyka L.

$(s, aabbbaa, S\#)$
$_e \Rightarrow (\langle s, S\#\rangle, aabbcc\$, S\#)$ [1]
$_e \Rightarrow (\langle s, AC\rangle, aabbcc\$, AC\#)$ $[2 = LL_d(\langle s, S\#\rangle, a)]$
$_e \Rightarrow (\langle q, AC\rangle, aabbcc\$, aAbC\#)$ $[3 = LL_d(\langle s, AC\rangle, a)]$
$_p \Rightarrow (\langle q, AC\rangle, abbcc\$, AbC\#)$ [*pop*]
$_e \Rightarrow (\langle s, AC\rangle, abbcc\$, AbcC\#)$ [4]
$_e \Rightarrow (\langle q, AC\rangle, abbcc\$, aAbbcC\#)$ $[3 = LL_d(\langle s, AC\rangle, a)]$
$_p \Rightarrow (\langle q, AC\rangle, bbcc\$, AbbcC\#)$ [*pop*]
$_e \Rightarrow (\langle s, AC\rangle, bbcc\$, AbbccC\#)$ [4]
$_e \Rightarrow (\langle f, C\#\rangle, bbcc\$, bbccC\#)$ $[5 = LL_d(\langle s, AC\rangle, b)]$

$_p \Rightarrow (\langle f, C\# \rangle, bcc\$, bccC\#)$ [*pop*]
$_p \Rightarrow (\langle f, C\# \rangle, cc\$, ccC\#)$ [*pop*]
$_p \Rightarrow (\langle f, C\# \rangle, c\$, cC\$)$ [*pop*]
$_p \Rightarrow (\langle f, C\# \rangle, \$, C\$)$ [*pop*]
$_e \Rightarrow (\langle f, \#\# \rangle, \$, \#)$ $[6 = LL_d(\langle f, C\# \rangle, \$)]$
$_e \Rightarrow (\$, \$, \#)$ [7]

Automat ukončil výpočet v koncovém stavu, a řetězec tedy byl přijat.

5.5 Požadavky na program

Program bude provádět analýzu syntaktických konstrukcí odpovídajících jazyku L_{nnn}, a to pomocí implementace hlubokého zásobníkového automatu a LL_d tabulky. Nebude obsahovat grafické rozhraní, program bude spouštěn v prostředí příkazového řádku. Vstupní text bude načítán ze standardního vstupu. Na standardní výstup program vypíše řetězec "ANO", pokud vstup patří do jazyka L_{nnn}. Pokud vstup do tohoto jayzka nepatří, vypíše program na standardní výstup řetězec "NE". V případě chyby bude vypsáno vhodné hlášení na standardní chybový výstup a běh programu skončí. Každý řádek je považován za samostatný vstup, výstup tedy bude obsahovat tolik řetězců "ANO" nebo "NE", kolik bylo na vstupu řádků.

Program bude dále průběžně demonstrovat průběh výpočtu tak, že bude na standardní chybový výstup zapisovat vhodně formátované informace o stavu zásobníku, aktuálním vstupním symbolu, prováděném pravidlu aj. Zobrazování těchto informací bude možno zakázat pomocí parametru na příkazovém řádku.

5.6 Analýza požadavků

Vyvíjená aplikace má pracovat v jednoduchém textovém prostředí. Veškerá její interakce s okolím se bude odehrávat prostřednictvím příkazového řádku, resp. standardního vstupu a výstupu. Zároveň z požadavků přirozeně vyplývá, že bude vhodné použít objektově orientovaný přístup, neboť program se bude skládat z několika logicky oddělených komponent (automat, lexikální analyzátor, ...). V důsledku toho zvolíme jako implementační jazyk C++. Jde o jazyk, který podporuje objektový přístup. Zároveň jde o kompilovaný jazyk, a konzolové aplikace jsou jedním z jeho obvyklých prostředí.

Program bude provádět tyto základní operace

- čtení vstupu
- lexikální analýza vstupního řetězce
- syntaktická analýza pomocí hlubokého automatu a LL_d tabulky
- prezentace výpočtu a výpis výsledku

Vidíme, že problém se v zásadě rozpadá na dvě hlavní části, tj. lexikální a syntaktickou analýzu. Proto bude vhodné implementovat odděleně lexikální analyzátor, a teprve jeho výstupy použít jak vstup hlavního analyzátoru.

5.7 Návrh

Při implementaci využijeme objektový přístup; program se tedy bude skládat z několika tříd.

Název třídy	**Popis**
Automaton	Hluboký automat. Obsahuje zásobník a stavové řízení.
Lex	Lexikální analyzátor. Čte textový standardní vstup, rozpoznává jednotlivé tokeny.
Parser	Vlastní syntaktický analyzátor. Jako svou součást implementuje LLd tabulku, od objektu třídy Lex přebírá informace o tokenech a řídí výpočet, probíhající v objektu třídy Automaton.
Presenter	Prezentační třída, vypisující požadované formátované údaje na výstup.

Tabulka 5.2: Přehled tříd

5.7.1 Lex

Třída reprezentující jednoduchý lexikální analyzátor. Bude implementovat jednoduchý konečný automat, který bude rozpoznávat tři typy vstupních řetězců podle následující tabulky.

Token	**Vstupní řetězec**
a	`real, int nebo bool`
b	identifikátor, $[a-zA-Z][a-zA-Z0-9]^*$
c	`true, false nebo číselný literál` $[0-9]^+[.]^*[0-9]^*$
x	chybný vstup (řetězec neodpovídající žádném z výše uvedených)
$	konec vstupního souboru

Tabulka 5.3: Rozpoznávané řetězce

Čtení bude probíhat přímo ze standardního vstupu. Výstupním rozhraním bude veřejná metoda poskytující při každém zavolání jeden token.

Název metody	Popis
GetNextToken	Vrátí token odpovídající vstupnímu řetězci. Řetězce jsou na vstupu odděleny bílými znaky (mezera, tabulátor, konec řádku).

Tabulka 5.4: Metody třídy Lex

5.7.2 Automaton

Třída bude reprezentovat hluboký zásobníkový automat. Bude implementovat zásobník a potřebné operace nad ním. Dále bude obsahovat rozhraní pro stavové řízení, pomocí kterého bude možné provádět výpočetní krok automatu, tj. měnit jeho stav a přepisovat neterminální symbol na zásobníku. Požadované rozhraní a jeho vlastnosti shrnuje následující tabulka.

Název metody	Popis
MakeStep(zdrojový stav, cílový stav, n, neterminál, řetězec)	Provede jeden výpočetní krok automatu. Změní jeho stav ze *zdrojový stav* na *cílový stav* a přepíše *n*-tý *neterminál* na zásobníku na *řetězec*
CanPop()	Test, zda je možné v aktuální konfiguraci automatu provést operaci *pop*
DoPop()	Provedení operace *pop*, tj. smazání terminálu z vrcholu zásobníku a přečtení tohoto terminálu ze vstupu
GetStateNterm(stav, neterminál)	Vrátí aktuální stav automatu a první neterminál na zásobníku.
SetInchar(terminál)	Nastaví *terminál* jako vstupní symbol automatu. Protože operace čtení tokenu od lexikálního analyzátoru pomocí jeho metody GetNextToken je jednorázová, nebude ji automat provádět. Terminál mu na vstup bude vkládat nadřazený objekt syntaktického analyzátoru.
Accepted()	Test, zda je automat ve stavu, kdy přijal vstupní řetězec.
GetStack()	Vrátí řetězec reprezentující obsah zásobníku

Tabulka 5.5: Metody třídy Automaton

5.7.3 Presenter

Název metody	Popis
PrintResult(výsledek)	Vypíše výsledek výpočtu - řetězec ANO nebo NE
PrintConfig(automat)	Vypíše aktuální konfiguraci automatu - stav, obsah zásobníku

Tabulka 5.6: Metody třídy Presenter

5.7.4 Parser

Zastřešující třída. Parser bude vytvářet objekty Lex, Automaton a Presenter. Od lexikálního analyzátoru bude číst tokeny, od automatu zjistí jeho aktuální konfiguraci. Pomocí LL tabulky, která je jeho atributem, rozhodne o dalším kroku výpočtu automatu. V okamžiku, kdy je dočten vstup do konce, voláním třídy Presenter vypisuje výsledek. Parser implementuje metodu, pomocí níž rozhoduje o dlaším kroku.

Název metody	Popis
ChooseNextRule()	Zvolí další pravidlo na základě konfihurace automatu, vstupu a LL_{deep} tabulky

Tabulka 5.7: Metody třídy Parser

Činnost Parseru by se dala popsat následujícím pseudokódem:

```
begin
    do
        Lex.GetNextToken();
        Parser.ChooseNextRule();
        if rule not found then
            Presenter.PrintResult(false);
            exit;
        else
            Automaton.PerformRule();
            while ( Automaton.CanPop()) do
                    Automaton.DoPop();
            endwhile
        endif
    while not end of input

    Presenter.PrintResult(Automaton.Accepted())
end
```

5.8 Testování

Činnost programu byla ověřena na testovacích vstupních datech. Výsledky shrnuje tabulka 5.8.

Vstup	**Výstup**
`int int int aa bb cc 1 2 3`	ANO
`int int aa bb cc 1 2 3`	NE
`int int int aa bb cc dd 1 2 3`	NE
`int real bool alfa beta gama 1 2.9 true`	ANO

Tabulka 5.8: Výsledky testů

5.9 Vývojové prostředí

K tvorbě zdrojového textu nebylo použito žádné integrované prostředí. Program byl překládán v operačním systému GNU/Linux pomocí kompilátoru g++ z kolekce GNU Compiler Collection. Pro implementaci bylo použito pouze prostředků standardního C++, tedy žádné doplňující knihovny nebo komponenty.

Kapitola 6

Závěr

V této práci jsem navrhl funkční metodu pro syntaktickou analýzu jazyků hlubokých zásobníkových automatů. Metoda ve své podstatě vychází z obdobné metody používané u jazyků bezkontextových. Pravděpodobně jde o rozšíření možností analýzy jazyků $L_{PDAdeep}$ oproti deterministickým hlubokým automatům. Formální důkaz, že třída jazyků analyzovatelných s pomocí LL_d tabulky je vlastní nadtřídou třídy jazyků deterministických DPDA, ale chybí. Postup takového důkazu by patrně musel zahrnout vyšetření (alespoň některých) uzávěrových vlastností jak jazyků z $LPDAdeep$, tak $L_{DPDAdeep}$. Na jejich základě by pak bylo možno ukázat existenci jazyka, který je analyzovatelný s pomocí LL_d tabulky, ale nepatří do $L_{DPDAdeep}$.

Další kroky v oblasti syntaktické analýzy jazyků z $LPDAdeep$ by mohly být vedeny snahou o nalezení silnějšího nástroje, než je metoda popsaná v této práci. Jak je uvedeno v odstavci 4.3.9, LL_d tabulku, tak jak byla definována, je možné použít k rozhodování pouze při expanzích hloubky 1, tj. na úrovni klasických zásobníkových automatů. Ve větší hloubce na zásobníku se musíme spokojit s tím, že po automatu budeme pro tyto expanze požadovat striktní determinismus. Domnívám se, že toto omezení bude mít vliv na sílu popsaného modelu. Tedy že třída takto analyzovatelných jazyků sice velmi pravděpodobně je vlastní natřídou $L_{DPDAdeep}$, ale nástroj, který by umožňoval rozhodování i při hlubších expanzích, pravděpodobně bude mít sílu ještě vyšší.

Hluboké zásobníkové automaty sice rozšiřují možnosti expanzí na zásobníku, ale stále zachovávají princip čtení pouze jednoho znaku vstupu. Stejně tak v LL_d tabulce máme možnost podívat se pouze na první znak vstupu, a na jeho základě určit použité pravidlo. Bylo by tedy přirozeným rozšířením snažit se nalézt způsob, jakým rozšířit možnosti analýzy tak, aby i expanze prováděné hlouběji v zásobníku bylo možné nějakým způsobem řídit v závislosti na vstupu. Jedním ze způsobů by mohlo být použití vícehlavého automatu, který má více čtecích hlav a může je nastavovat na různá místa vstupního řetězce nezávisle na sobě. Problémem by ovšem stále zůstalo, jakým způsobem určit (nebo alespoň odhadnout), která část vstupního řetězce má "řídit" expanzi daného neterminálu v daném stavu.

Příbuzný výzkum by tedy podle mého názoru bylo vhodné směřovat těmito směry:

- vyšetření uzávěrových vlastností $LPDAdeep$ a $L_{DPDAdeep}$
- zjištění síly představeného modelu

- pokus o nalezení silnější alternativy

V implementační části své práce jsem předvedl praktickou realizovatelnost navržené metody syntaktické analýzy. Analyzátor je ve své současné podobě sice pouze jednoúčelový, neboť přijímá pouze jediný jazyk, ale jeho návrh umožňuje snadné rozšíření. Pravidla automatu jsou uložena v datové struktuře a je možné je libovolně upravovat, stejně tak LL_d tabulku. Proto by nemělo být obtížné vytvořit např. modul, který nastaví automat podle textového popisu v externím souboru, nebo přidat k programu grafické rozhraní umožňující interaktivní tvorbu a ovládání analyzátoru.

Praktickým důsledkem této práce je zjištění, že existuje poměrně jednoduchý nástroj implementující analýzu jazyků ležících nad jazyky bezkontextovými. To umožňuje použití složitějších jazykových konstrukcí, např. v syntaxi programovacích jazyků. Představený modelový způsob deklarace proměnných sice pravděpodobně v praxi nebude příliš využíván, ale nepochybně se najdou syntaktické konstrukce, jejichž praktická užitečnost bude vyšší a k jejich analýze bude použito právě hlubokých zásobníkových automatů.

Literatura

[1] Meduna A. *Automata and Languages*. Springer, London, 2000.

[2] Meduna A. Deep pushdown automata. *Acta Informatica*, 2006.

[3] Schoenecker R., Lukáš R. Zadání projektu z předmětů ifj a ial. `https://www.fit.vutbr.cz/study/courses/IFJ/private/projekt/ifj2006.pdf`.

Dodatek A

Návod k obsluze programu

A.1 Kompilace

Program je dodáván ve formě zdrojových kódů jazyka C++, spolu se souborem `Makefile`. V prostředí podporujícím příkaz `make` je program zkompilovatelný jeho spuštěním v adresáři se zdrojovými kódy.

A.2 Vstupní data

Program očekává vstupní řetězce na standardním vstupu. Je tedy možné např. přesměrovat vstup uložený v textovém souboru takto: `deepparser >vstup.txt`. Obsahem mohou být libovolné textové řetězce. Program ovšem přijme (vypíše "ANO") pouze pro vstupy splňující následující omezení:

- jde o jazyk $a^n b^n c^n$, kde
- a je klíčové slovo `real, int` nebo `bool`
- b je identifikátor vyhovující regulárnímu výrazu $[a-zA-Z][a-zA-Z0-9]^*$
- c je `true`, `false` nebo číselný literál $[0-9]^+[.]^*[0-9]^*$

A.2.1 Přijaté vstupy

Analyzátor přijme například následující vstupní řetězce:

- `int int int aa bb cc 1 2 3`
- `int real bool alfa beta gama 1 2.9 true`

Každý řádek vstupního souboru je považován za samostatný vstup.

A.3 Výstup

Na standardní textový výstup je zobrazován pouze výsledek analýzy, tj "ANO" nebo "NE". Případné chyby jsou hlášeny na stanradní chybový výstup. Na ten jsou také ve výchozím nastavení vypisovány rozšiřující informace. Jejich výpis je možné potlačit paramatrem `-s`.

Seznam tabulek

2.1 Chomského hierarchie jazyků 8

4.1 Ukázka LL tabulky 22
4.2 Ukázka LLd tabulky 27
4.3 Množiny *First* gramatiky G_1 28
4.4 Množiny *Empty* gramatiky G_1 28
4.5 Množiny *Follow* gramatiky G_1 29
4.6 Množiny *Predict* gramatiky G_1 29
4.7 LL_d tabulka granatiky G_1 30

5.1 LL_d tabulka pro jazyk L_{nnn} 35
5.2 Přehled tříd 37
5.3 Rozpoznávané řetězce 37
5.4 Metody třídy Lex 38
5.5 Metody třídy Automaton 38
5.6 Metody třídy Presenter 39
5.7 Metody třídy Parser 39
5.8 Výsledky testů 40