

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODIFIKOVANÉ HLUBOKÉ ZÁSOBNÍKOVÉ AUTOMATY

BAKALÁŘSKÁ PRÁCE

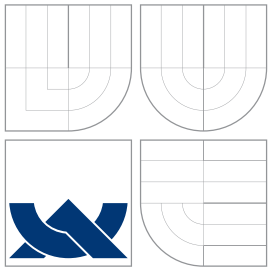
BACHELOR'S THESIS

AUTOR PRÁCE

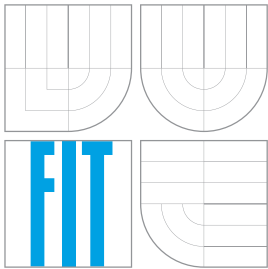
AUTHOR

RADKA ŠKVAŘILOVÁ

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODIFIKOVANÉ HLUBOKÉ ZÁSOBNÍKOVÉ AUTOMATY

MODIFIED DEEP PUSHDOWN AUTOMATA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

RADKA ŠKVAŘILOVÁ

VEDOUCÍ PRÁCE

SUPERVISOR

prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2013

Abstrakt

Tato práce představuje dvě nové modifikace hlubokých zásobníkových automatů – bezstavové hluboké zásobníkové automaty a paralelní bezstavové hluboké zásobníkové automaty. V teoretické části jsou zavedeny formální definice a také je zde zkoumána síla těchto automatů. V praktické části je ukázána na jednoduchém příkladu implementace těchto automatů.

Abstract

This thesis introduce two new modifications of deep pushdown automata – stateless deep pushdown automata and parallel deep pushdown automata. In theoretical part of this thesis is formal definition and research into power of these modification. Practical part consists of an implementation of simple automata program.

Klíčová slova

Formální jazyky, automaty, zásobníkového automaty, hluboké zásobníkové automaty, bezstavové hluboké zásobníkové automaty, paralelní hluboké zásobníkové automaty, gramatiky.

Keywords

Formal languages, automata, pushdown automata, deep pushdown automata, stateless deep pushdown automata, parallel deep pushdown automata, grammar.

Citace

Radka Škvařilová: Modifikované hluboké zásobníkové automaty, bakalářská práce, Brno, FIT VUT v Brně, 2013

Modifikované hluboké zásobníkové automaty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana prof. RNDr. Alexandra Meduny, CSc.

.....
Radka Škvařilová
14. května 2013

Poděkování

Zde bych ráda poděkovala mému vedoucímu prof. Alexandru Medunovi, za odborné vedení a konzultace při tvorbě této práce. Malé poděkování patří také mé rodině a příteli za psychickou podporu v posledním semestru studia.

© Radka Škvařilová, 2013.

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	3
2 Matematický základ	4
3 Jazyky	5
3.1 Chomského hierarchie	5
4 Gramatiky	6
4.1 Bezkontextová gramatika	6
4.1.1 Definice	6
4.2 Kontextová gramatika	6
4.2.1 Definice	7
4.3 N-omezená stavová gramatika	7
4.3.1 Definice	7
5 Automaty	8
5.1 Konečný automat	8
5.1.1 Definice	8
5.2 Zásobníkový automat	9
5.2.1 Zásobník	9
5.2.2 Definice	9
5.3 Hluboký zásobníkový automat	10
5.3.1 Definice	10
5.3.2 Přijmutí jazyka	11
5.4 Paralelní hluboký zásobníkový automat	11
5.4.1 Definice	12
5.4.2 Přijmutí jazyka	12
6 Modifikace	14
6.1 Bezstavový hluboký zásobníkový automat	14
6.1.1 Definice	14
6.1.2 Přijmutí jazyka	15
6.2 Paralelní bezstavový hluboký zásobníkový automat	16
6.2.1 Definice	16
6.2.2 Přijmutí jazyka	16

7	Síla	18
7.0.3	Síla bezstavového hlubokého zásobníkového automatu	18
7.0.4	Síla paralelního bezstavového hlubokého zásobníkového automatu	19
8	Ukázka	22
8.1	Převod hlubokého zásobníkového automatu na paralelní bezstavový hluboký zásobníkový automat	22
9	Struktura předkladačů	23
9.1	Lexikální analýza	23
9.2	Syntaktická analýza	23
9.3	Sémantická analýza	24
9.4	Generátor vnitřního kódu	24
9.5	Optimalizátor	24
9.6	Generátor cílového kódu	24
10	Návrh programu	25
10.1	SDPDA	25
10.2	PSDPDA	25
11	Implementace	26
11.1	Odlišnosti SDPDA a PSDPDA	26
12	Závěr	27
A	CD	29
A.1	Programy	29
A.2	Elektronická forma tohoto dokumentu	29

Kapitola 1

Úvod

Tato práce je zaměřena na oblast z teorie formálních jazyků. Budu se zde zabývat, jak již název napovídá, modifikacemi hlubokých zásobníkových automatů. Hluboké zásobníkové automaty poprvé zavedl v roce 2006 Alexander Meduna ve svém článku v časopise Acta Informatica [2]. Tyto automaty jsou v poslední době poměrně hodně zkoumané téma, jelikož je můžeme využít například při syntaktické analýze. I když si momentálně většina jazyků stále vystačí pouze se silou bezkontextových gramatik, nemusí tomu tak být stále. Hluboké zásobníkové automaty jsou schopny poměrně jednoduchým mechanismem sílu bezkontextových gramatik přesáhnout. Část této práce o modifikovaných hlubokých zásobníkových automatech byla již úspěšně prezentována na konferenci STUDENT EEICT 2013.

První z modifikací, která zde bude uvedena, je bezestavová verze hlubokých zásobníkových automatů. Možnost odstranění stavů je na poli formálních jazyků velmi oblíbené a také diskutované téma. Chtěla jsem proto zjistit, jaké důsledky to bude mít na hluboký zásobníkový automat. Při samotném zkoumání této modifikace jsem se rozhodla zavést paralelní verzi bezestavového hlubokého zásobníkového automatu, který svými vlastnostmi nejenže vylepšuje některé „špatné“ vlastnosti neparalelní verze, ale také nám umožňuje zrychlení. Některé výsledky této práce budou založeny na výsledcích Petra Solára, který ve své bakalářské práci [4] zavedl klasické paralelní hluboké zásobníkové automaty.

První část práce by měla sloužit čtenáři jako úvod do dané problematiky. Ač zde budou uvedeny matematické základy, úvod do gramatik a automatů, je text určen pro odborného čtenáře. Tyto části slouží především pro specifikaci symbolů v této práci. Inspirací pro tuto část práce, byly především knihy od Alexandra Meduny – Elements of Compile Design [3] a Automata and Languages [1].

Druhá část se bude zabývat formální i neformální definicí daných modifikací. Pro názornost bude u každé modifikace uveden i příklad přijímání konkrétního řetězce z určitého jazyka. V této části bude také zkoumána síla těchto automatů, především pak srovnání síly s klasickými hlubokými zásobníkovými automaty.

Poslední část bude zaměřena na praktické využití těchto modifikací. Jelikož se automaty využívají především v překladačích, konkrétně pak v syntaktické analýze, bude na začátku této části ve zkratce popsán překladač a všechny jeho části. Dále zde pak bude popsán samotný program a jeho implementace.

V samotném závěru této práce bude zmíněna možnost dalšího vývoje této práce.

Kapitola 2

Matematický základ

Uveďme si pro začátek některé základní matematické pojmy, které se vyskytují v této práci.

Množina Σ je kolekce elementů, které jsou prvky z předem definovaného prostoru. Pokud Σ obsahuje element a , pak symbolicky zapisujeme $a \in \Sigma$ a čteme a je prvkem Σ , naopak $a \notin \Sigma$ čteme a není prvkem Σ .

Kardinalita $card(\Sigma)$ značí počet prvků v množině Σ . Množina, která neobsahuje žádný prvek je nazývána prázdná množina, značí se \emptyset a platí $card(\emptyset) = 0$.

Abeceda Σ je konečná neprázdná množina elementů, které nazýváme symboly. Konečná množina symbolů nad Σ je řetězec nad Σ . ε je speciální řetězec nad abecedou Σ , který značí tzv. prázdný řetězec. To znamená, že neobsahuje žádný symbol. Σ^* značí množinu všech řetězců nad Σ ; $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Nechť x a y jdou dva řetězce nad abecedou Σ .

- Délka řetězce x , $|x|$, je definována pro $x = \varepsilon$, $|x| = 0$ a $x = a_1 \dots a_n$, pak $|x| = n$ pro $n \geq 1$ a $a_j \in \Sigma$ pro všechna $i = 1, \dots, n$.
- Konkatenace neboli zřetězení řetězců x a y je řetězec xy .
- x je podřetězec y , pokud existují řetězce z, z' nad abecedou Σ , přičemž platí $zxz' = y$.
- i -tá mocnina řetězce x , x^i , definována: $x^0 = \varepsilon$ a pro $i \geq 1$: $x^i = xx^{i-1}$.
- Pro množinu $W \subseteq \Sigma$ značí funkce $occur(x, W)$ počet výskytů symbolů z množiny W v řetězci x .

Kapitola 3

Jazyky

Nechť Σ^* značí množinu všech řetězců nad abecedou Σ . Nad každou abecedou je rovněž definována množina \emptyset a $\{\varepsilon\}$. Přičemž platí $\emptyset \neq \{\varepsilon\}$. \emptyset neobsahuje žádný prvek, zatímco $\{\varepsilon\}$ obsahuje ε . Jelikož Σ^* obsahuje všechny řetězce nad Σ , můžeme ji nazvat univerzální jazyk nad Σ . Jazyk je tedy definován jako množina řetězců nad abecedou Σ . Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad Σ .

Formální jazyky můžeme rozdělit na konečné a nekonečné. Jazyk je konečný, jestliže platí $\text{card}(L) = n$ pro $n \geq 1$, což znamená, že obsahuje konečný počet řetězců. V opačném případě se jedná o jazyk nekonečný.

Základními modely pro popis jazyků jsou gramatiky nebo automaty. Gramatiky generují dle svých pravidel řetězce daného jazyka, oproti tomu automaty pouze rozhodují o tom, jestli řetězec patří do jazyka, který definuje tento automat.

3.1 Chomského hierarchie

V roce 1956 byla vytvořena hierarchie tříd formálních gramatik generujících formální jazyky. Autorem této hierarchie byl Noam Chomsky, který se velmi úspěšně prosadil na poli lingvistiky. Gramatiky jsou v tomto modelu rozděleny hierarchicky do 4 skupin dle své síly.

Gramatika typu 0 – jedná se o nejsilnější typ. Zahrnuje všechny formální jazyky, které mohou být rozpoznávány Turingovým strojem. Jedná se o abstraktní výpočetní zařízení, které se skládá z konečného automatu, pravidel přechodové funkce a teoreticky nekonečné pásky pro zápis mezivýsledků. Těmto jazykům se někdy říká rekurzivně spočetné jazyky.

Gramatika typu 1 – tento typ je pomnožinou typu 0. Jedná se o kontextové jazyky (CS – z anglického context-sensitive). Tyto jazyky jsou rozpoznatelné lineárně ohraničeným Turingovým strojem. Jedná se o Turingův stroj, který má pomocí lineární funkce omezenou délku pásky.

Gramatika typu 2 – jde o podmnožinu typu 0 i typu 1. Tato gramatika zahrnuje bezkontextové jazyky (CF – z anglického context-free). Většinou se pomocí tohoto typu popisuje syntaxe programovacích jazyků. Gramatiky typu 2 můžeme tedy rozpoznat pomocí zásobníkového automatu.

Gramatika typu 3 – tento typ je pomnožinou všech předchozích typů. Gramatika typu 3 je schopna generovat jazyky regulární, které jsou přijímány konečným automatem.

Tato práce je především zaměřená na modely pohybující se na hranici gramatik typu 1 a 2, kam spadá velmi často zmiňovaný jazyk $a^n b^n c^n$ pro $n \geq 1$.

Kapitola 4

Gramatiky

Formální gramatiky jsou struktury, které popisují formální jazyky. Každá gramatika je založena na množině gramatických pravidel, která reprezentují terminální a neterminální symboly. Pomocí těchto pravidel může být vygenerován řetězec daného jazyka z předem daného počátečního symbolu. Programovací jazyky jsou většinou popsány pomocí specifických nástrojů založených na gramatikách.

Nyní si přiblížíme některé gramatiky, které budou později v práci korespondovat k daným formálním modelům.

4.1 Bezkontextová gramatika

Bezkontextové gramatiky mají velký praktický význam. Jak již bylo zmíněno, jsou ekvivalentní s 2. typem gramatik Chomského hierarchie a využívají se pro definování syntaxe většiny programovacích jazyků. Jsou definovány konečnou množinou neterminálních symbolů, konečnou množinou přepisovacích pravidel a startujícím terminálním symbolem. Typickým bezkontextovým jazykem je jazyk $a^n b^n$, kde $n \geq 1$.

Bezkontextová gramatika je speciálním případem kontextové gramatiky. Jedná se o prázdný kontext.

4.1.1 Definice

Bezkontextová gramatika je uspořádaná čtveřice $G = (N, T, P, S)$, kde

N je abeceda neterminálních symbolů

T je abeceda terminálních symbolů takových, že $(N \cap T) \neq \emptyset$

$S \in N$ je startující symbol

P je množina pravidel $A \rightarrow x$, kde $A \in N$, $x \in (N \cup T)^$*

4.2 Kontextová gramatika

Jedná se o gramatiku, jejíž levá i pravá strana pravidel jsou obklopeny kontextem terminálních a neterminálních symbolů. Síla těchto gramatik je ekvivalentní s 1. typem gramatik Chomského hierarchie.

4.2.1 Definice

Kontextová gramatika je uspořádaná čtveřice $G = (N, T, P, S)$, kde

N je abeceda neterminálních symbolů

T je abeceda terminálních symbolů takových, že $(N \cap T) \neq \emptyset$

$S \in N$ je startující symbol

P je množina pravidel $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, kde $A \in N$, $\alpha_1, \alpha_2, \beta \in (N \cup T)^+$

4.3 N-omezená stavová gramatika

N-omezené stavové gramatiky jsou velmi vhodné v situacích, kdy uživatel potřebuje využít některé vlastnosti kontextových jazyků a nechce v syntaktické analýze využít kontextových gramatik. Stačí jen využít n-omezenou stavovou gramatiku, která svojí silou překračuje sílu bezkontextových gramatik, avšak nepokrývá všechny kontextové jazyky. Tvoří tedy nekonečnou hierarchii tříd jazyků mezi jazyky bezkontextovými a kontextovými. Jelikož je tato gramatika ekvivalentní s hlubokým zásobníkovým automatem, který je hlavním předmětem této práce, blíže si ji představíme.

4.3.1 Definice

Stavová gramatika je uspořádaná šestice $G = (N, T, W, P, S, p_0)$, kde

N je abeceda neterminálních symbolů

T je abeceda terminálních symbolů takových, že $(N \cap T) \neq \emptyset$

$S \in N$ je startující symbol

W je konečná množina stavů

$p_0 \in W$ je startující stav

P je množina pravidel $(p, A) \rightarrow (q, v)$, kde $A \in N$, $p, q \in W$, $v \in (N \cup T)^+$

Jazyk definovaný touto gramatikou je pak definován: $L(G) = \{w : w \in T^+, (p_0, S) \Rightarrow^* (q, w) \text{ pro } q \in W\}$.

Nechť $V = N \cup T$ je totální abeceda. Automat definuje klasický derivační krok, provedený za pomoci pravidla $(q, A) \rightarrow (p, v)$, jako: $(q, xAy) \Rightarrow (p, xvy)$, kde $p, q \in W$, $x, y \in V^*$, $A \in (V - T)$, $v \in V^+$.

N-omezená derivace:

Nefornálně n-omezená derivace znamená, že v daném kroku derivace bude přepsán neterminální symbol maximálně hloubky n .

Formálně zapsáno jako $(q, xAy) \xrightarrow{n} (p, xvy)$, kde $p, q \in W$, $x, y \in V^*$, $A \in (V - T)$, $v \in V^+$. Přičemž musí platit $\text{occur}(xA, V - T) \geq n$.

Jazyk definovaný n-omezenou stavovou gramatikou je pak definován $L(G, n) = \{w : w \in T^+, (p_0, S) \xrightarrow{n}^* (q, w) \text{ pro } q \in W\}$.

Detailněji se n-omezenou stavovou gramatikou zabýval ve své bakalářské práci [5] Petr Zemek.

Kapitola 5

Automaty

Automaty obecně jsou výpočetní modely využívané při studiu formálních jazyků. Většinou kontrolují, zda vstupní řetězec splňuje pravidla daného jazyka. Pro lepší pochopení složitějších automatů si zde ukážeme hierarchii automatů od jednodušších ke složitějším. Většinou je totiž každý automat nějakým rozšířením již existujícího automatu.

5.1 Konečný automat

Konečný automat se skládá ze vstupní pásky, čtecí hlavy a konečného stavového zařízení. Vstupní páska je rozdělena do částí. Každá část obsahuje symbol vstupního řetězce/slova $a_1 \dots a_n$. Symboly jsou čteny pomocí čtecí hlavy. Konečné stavové zařízení obsahuje konečnou množinu stavů spolu s konečnou množinou pravidel. Automat zpracovává vstupní pásku pomocí sekvence přechodů. Každý přechod je proveden aplikací pravidla, které popisuje, jak se aktuální stav změní v následující, zatímco může být přečten symbol ze vstupní pásky. Jestliže je přečten symbol, čtecí hlava se posune o jednu část vstupní pásky doprava, jinak zůstává na místě. Aby automat mohl rozhodnout, zda přijme vstupní řetězec, je potřeba automatu nastavit počáteční stav a množinu koncových stavů. Řetězec je přijmut v případě, že automat se pomocí přechodů dostal z počátečního stavu do jednoho z koncových stavů.

Síla konečného automatu je ekvivalentní s regulárními jazyky.

5.1.1 Definice

Konečný automat je uspořádaná pětice $M = (Q, \Sigma, R, s, F)$, kde

Q je konečná množina stavů

Σ je vstupní abeceda taková, že platí $\Sigma \cap Q = \emptyset$

$s \in Q$ je startující stav

$F \subseteq Q$ je konečná množina koncových stavů

R je konečná množina pravidel tvaru $(pa \rightarrow q) \in R$, $p, q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$

Konfigurace: $\chi \in Q\Sigma^*$

Přechod: Nechť pa a qx jsou dvě konfigurace konečného automatu, kde $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in \Sigma^*$. Nechť $r = pa \rightarrow q \in R$ je pravidlo. Potom může konečný automat provést přechod z pa do qx za použití r , zapsáno $pa \Rightarrow qx$.

Přijímaný jazyk: Nechť M je konečný automat. Jazyk přijímaný tímto automatem $L(M)$

je definován: $L(M) = \{w : w \in \Sigma^* | sw \Rightarrow^* f, f \in F\}$. Automat tedy musí dojít ze startujícího stavu do koncového a přečíst přitom ze vstupní pásky celý řetězec.

5.2 Zásobníkový automat

Zásobníkový automat je reprezentován konečným automatem, který je rozšířený o teoreticky nekonečný zásobník. Automat pracuje na mechanismu přechodů za pomoci konečné množiny pravidel. Přejít závisí na aktuálním symbolu, na nevstupním symbolu, který je na vrcholu zásobníku, a na vstupním symbolu. Během přechodu provede změnu stavu a prepíše nejvrchnější nevstupní symbol na zásobníku na řetězec zásobníkových symbolů, a pokud byl přečten vstupní symbol, posune vstupní hlavu o jednu pozici doprava.

Se zásobníkem mohou být prováděny dvě základní operace – vyjmutí a expanze. Vyjmutí se provádí v případě, že je na vrcholu zásobníku vstupní symbol a ten stejný je také na vstupní pásce. Expanze naopak vyžaduje nevstupní symbol na vrcholu zásobníku, ten je pak přepsán na řetězec zásobníkových symbolů.

Zásobníkový automat je díky využití zásobníku schopen zpracovat bezkontextové jazyky.

5.2.1 Zásobník

Zásobník je homogenní, lineární, obecně dynamický abstraktní datový typ. Používá se pro odkládání dat. Pracuje na principu „Last In – First Out“, což znamená, že prvek, který vložíme na zásobník jako poslední, z něj vybereme první. K manipulaci dat v zásobníku používáme především dvě operace – push a pop. Push uloží data na vrchol zásobníku, pop naopak vybere data z vrcholu zásobníku. Klasický zásobník si udržuje jen ukazatel na vrchol zásobníku, nemůže tedy operovat s položkami uloženými hlouběji v zásobníku.

5.2.2 Definice

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

Q je konečná množina stavů

Σ je vstupní abeceda taková, že platí $\Sigma \cap Q = \emptyset$

Γ je konečná množina zásobníkových symbolů, nazývaná zásobníková abeceda kde, $\Sigma \subset \Gamma$

$s \in Q$ je startující stav

$S \in \Gamma$ je počáteční zásobníkový symbol

$F \subseteq Q$ je konečná množina koncových stavů

R je konečná množina pravidel tvaru $(Aqa \rightarrow pw) \in R, A \in (\Gamma - \Sigma), p, q \in Q, a \in (\Sigma \cup \{\varepsilon\}), w \in \Gamma^*$

Konfigurace: $\chi \in \Gamma^*Q\Sigma^*$

Přejít: Necht $xApay$ a $xwqy$ jsou dvě konfigurace zásobníkového automatu, kde $x, w \in \Gamma^*, A \in \Gamma, p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, a \in \Sigma^*$ a $y \in \Sigma^*$. Necht $r = Apa \rightarrow wq \in R$ je pravidlo. Potom může zásobníkový automat provést přechod z $xApay$ do $xwqy$ za použití r , zapsáno $xApay \Rightarrow xwqy$.

Přijímaný jazyk: Jazyk přijímaný zásobníkovým automatem může být buď přijímaný přechodem do koncového stavu: $L(M) = \{w : w \in \Sigma^*, Ssw \Rightarrow^* zf, z \in \Gamma^*, f \in F\}$, nebo vyprázdněním zásobníku: $L(M) = \{w : w \in \Sigma^*, Ssw \Rightarrow^* zf, z = \varepsilon, f \in Q\}$, nebo

přechodem do koncového stavu a zároveň vyprázdněním zásobníku: $L(M) = \{w : w \in \Sigma^*, Ssw \Rightarrow^* zf, z = \varepsilon, f \in F\}$.

5.3 Hluboký zásobníkový automat

Hluboký zásobníkový automat představuje úpravu klasického zásobníkového automatu o možnost zasahovat do hloubky. Síla tohoto automatu je srovnatelná s generativní silou řízených bezkontextových gramatik bez zkracujících pravidel. Ty jsou totiž silnější než klasické zásobníkové automaty, ale zároveň slabší než kontextové gramatiky. Přesněji síla těchto automatů hierarchicky roste, stejně jako síla n -omezených stavových gramatik, což bylo dokázáno v již zmiňovaném článku [2]. Automat rovněž jako zásobníkový automat funguje na principu expanze a vyjmutí. Například pokud máme na vstupní pásce symbol a a na vrcholu zásobníku je rovněž a , můžeme provést vyjmutí tohoto symbolu a posuneme se na další vstupní symbol. Vyjmutí je tedy shodné jako u zásobníkového automatu. Expanze je poněkud odlišná. Jestliže se mezi nejvchnějšími n neterminálními symboly zásobníku nachází neterminální symbol, pro který existuje pravidlo, které umožňuje expanzi daného symbolu, můžeme symbol nahradit jiným řetězcem dle daného pravidla. Pokud omezíme hloubku n zásobníkového automatu na 1, dostaneme se zpět na sílu klasického zásobníkového automatu. Zde popsána verze nedovoluje užívání vymazávacích pravidel a je nedeterministická.

Uvedu znovu formálně některé vlastnosti, které budeme později v práci využívat:

- Pro každé $n \geq 1$ a pro každý jazyk n -omezené stavové gramatiky L , $L(G, n)$, existuje jazyk, který přijímá hluboký zásobníkový automat K , $K(nM)$, přijímající stejný jazyk a naopak. Platí tedy $L(G, n) = K(nM)$.
- ${}_nM \subset_{n+1} M$ pro každé $n \geq 1$.
- ${}_1M = CF$.
- Pro každé $n \geq 1$ platí ${}_nM \subset CS$.

5.3.1 Definice

Hluboký zásobníkový automat je uspořádaná sedmice ${}_{deep}M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

deep je maximální hloubka, v níž může dojít k nahrazení

Q je konečná množina stavů

Σ je vstupní abeceda

Γ je konečná množina zásobníkových symbolů, nazývaná zásobníková abeceda kde, $\Sigma \subset \Gamma$, $\# \in (\Gamma - \Sigma)$, $\#$ je speciální pomocný symbol značící dno zásobníku

stav $s \in Q$ je startující stav

$S \in \Gamma$ je počáteční zásobníkový symbol

$F \subseteq Q$ je konečná množina koncových stavů

a R je konečná množina pravidel tvaru $(mqA \rightarrow pv) \in R$, $A \in \Gamma - \Sigma$, $p, q \in Q$, $0 < m \leq deep$ (m značí v jaké hloubce zásobníku budeme nahrazovat), $v \in \Gamma^*$

Konfigurace: $\chi \in Q\Sigma^*(\Gamma - \{\#\})^*\{\#\}$

Přechod: Nechtě x, y jsou dvě konfigurace.

Automat provede vyjmutí a přejde z x do y , pokud $x = (q, au, az)$ a $y = (q, u, z)$, kde $q \in Q$,

$a \in \Sigma, u \in \Sigma^*, z \in \Gamma^*$. Zapišeme jako $x \xrightarrow{p} y$.

Automat provede expanzi a přejde z x do y , pokud $x = (q, w, uAz)$ a $y = (p, w, uvz)$ a zároveň $(mqA \rightarrow pv) \in R$, kde $q, p \in Q, w \in \Sigma^*, A \in \Gamma, u, v, z \in \Gamma^*$. Zapišeme jako $x \xrightarrow{e} y$.

Přijímaný jazyk: Nechť ${}_nM$ je hluboký zásobníkový automat. Jazyk přijímaný tímto automatem $L({}_nM)$ je definován: $L({}_nM) = \{w : w \in \Sigma^*, (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#) \text{ v } {}_nM, \text{ kde } f \in F\}$, nebo $E({}_nM) = \{w : w \in \Sigma^*, (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#) \text{ v } {}_nM, \text{ kde } f \in Q\}$. První definice značí jazyk, který je přijímán prázdným automatem a koncovým stavem. Druhá definice udává, že jazyk je přijmut i v případě, že se automat nenachází v koncovém stavu.

5.3.2 Přijmutí jazyka

${}_2M = (\{s, q, p, f\}, \{a, b, c, \}, \{a, b, c, A, S, \#\}, R, s, S, \{f\})$
s pravidly

1. $1sS \rightarrow qAA$
2. $1qA \rightarrow paAb$
3. $1qA \rightarrow fab$
4. $2pA \rightarrow qAc$
5. $1fA \rightarrow fc$

Na vstupu máme řetězec $aaabbbccc$, který patří do jazyka $a^n b^n c^n$:

$$\begin{aligned} (s, aaabbbccc, S\#) &\xrightarrow{e} (q, aaabbbccc, AA\#) [1] \\ &\xrightarrow{e} (p, aaabbbccc, aAbA\#) [2] \\ &\xrightarrow{p} (p, aabbbccc, AbA\#) \\ &\xrightarrow{e} (q, aabbbccc, AbAc\#) [4] \\ &\xrightarrow{e} (p, aabbbccc, aAbbAc\#) [2] \\ &\xrightarrow{p} (p, abbbccc, AbbAc\#) \\ &\xrightarrow{e} (q, aaabbbccc, aAbbAcc\#) [4] \\ &\xrightarrow{e} (f, abbbccc, abbbAcc\#) [3] \\ &\xrightarrow{p} (f, bbccc, bbbAcc\#) \\ &\xrightarrow{p} (f, bbccc, bbAcc\#) \\ &\xrightarrow{p} (f, bccc, bAcc\#) \\ &\xrightarrow{p} (f, ccc, Acc\#) \\ &\xrightarrow{e} (f, ccc, ccc\#) [5] \\ &\xrightarrow{p} (f, cc, cc\#) \\ &\xrightarrow{p} (f, c, c\#) \\ &\xrightarrow{p} (\varepsilon, \#) \end{aligned}$$

Tento jazyk byl přijat po 16ti krocích, z toho bylo 7 expanzí a 9 vyjmutí.

5.4 Paralelní hluboký zásobníkový automat

Paralelní hluboký zásobníkový automat rozšiřuje klasickou verzi hlubokého zásobníkového automatu o možnost expanze několika nejvrchnějších neterminálních symbolů na zásobníku během jediného kroku. Tento rozdíl je způsoben změnou tvarů pravidel. Každé pravidlo je

složeno z několika jednodušších akcí, které nahrazují jednotlivé neterminální symboly v dané hloubce řetězcem. První neterminální symbol v pravidle vždy odpovídá nejvrchnějšímu neterminálnímu symbolu na zásobníku. Maximální velikost levé strany pravidla je omezena hloubkou, do které můžeme zasahovat. Pravidlo můžeme použít pouze v případě, že je na zásobníku dostatečný počet neterminálních symbolů. Tyto symboly se na zásobníku musí nacházet v pořadí, které je dáno konkrétním pravidlem.

Síla tohoto automatu je shodná se silou hlubokého zásobníkového automatu. Hlavní předností tohoto automatu je především jeho rychlost. Toto tvrzení ovšem platí jen pro některé případy. Musí se jednat o automat větší hloubky než 1, jehož pravidla jsou vhodně definována. Navíc ke zrychlení dochází pouze při expanzi. Vyjmutí je definováno stejně jako u hlubokého zásobníkového automatu.

Stejně jako již zmíněný hluboký zásobníkový automat, tento automat pracuje nedeterministicky a nedovoluje vymazávací pravidla.

5.4.1 Definice

Paralelní hluboký zásobníkový automat je uspořádaná sedmice ${}_{deep}M_{par} = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

$deep$ je maximální hloubka, v níž může dojít k nahrazení

Q je konečná množina stavů

Σ je vstupní abeceda

Γ je konečná množina zásobníkových symbolů, nazývaná zásobníková abeceda

kde, $\Sigma \subset \Gamma$, $\# \in (\Gamma - \Sigma)$, $\#$ je speciální pomocný symbol značící dno zásobníku stav $s \in Q$ je startující stav

$S \in (\Gamma - \Sigma)$ je počáteční zásobníkový symbol

$F \subseteq Q$ je konečná množina koncových stavů

R je konečná množina pravidel tvaru $p(A_1, A_2 \dots A_n) \rightarrow q(v_1, v_2 \dots v_n) \in R$, $p, q \in Q$, $A_j \in \Gamma - (\Sigma \cap \#)$, $1 < j \leq deep$, $n \leq deep$, $v_j \in \Gamma^$*

Konfigurace: $\chi \in Q\Sigma^*(\Gamma - \{\#\})^*\{\#\}$

Přechod: Nechť x, y jsou dvě konfigurace.

Automat provede vyjmutí a přejde z x do y , pokud $x = (q, au, az)$ a $y = (q, u, z)$, kde $q \in Q$, $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$. Zapišeme jako $x \xrightarrow{p} y$.

Automat provede expanzi a přejde z x do y , pokud $x = (q, w, uABCz)$ a $y = (p, w, ucdez)$ a zároveň $q(A, B, C) \rightarrow p(c, d, e) \in R$, kde $q, p \in Q$, $w \in \Sigma^*$, $A, B, C \in \Gamma$, $u, c, d, e, z \in \Gamma^*$. Zapišeme jako $x \xrightarrow{e} y$.

Přijímaný jazyk: Nechť ${}_nM$ je paralelní hluboký zásobníkový automat. Jazyk přijímaný tímto automatem $L({}_nM)$ je definován stejně jako jazyk přijímaný neparalelní verzí: $L({}_nM) = \{w : w \in \Sigma^*, (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#) \vee {}_nM, \text{ kde } f \in F\}$, nebo $E({}_nM) = \{w : w \in \Sigma^*, (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#) \vee {}_nM, \text{ kde } f \in Q\}$.

5.4.2 Přijmutí jazyka

${}_2M = (\{s, q, f\}, \{a, b, c\}, \{a, b, c, A, S, \#\}, R, s, S, \{f\})$
s pravidly

1. $1sS \rightarrow qAA$

$$2. 1q(A, A) \rightarrow p(aAb, Ac)$$

$$3. 1q(A, A) \rightarrow f(ab, c)$$

Na vstupu máme řetězec aaabbbccc, který patří do jazyka $a^n b^n c^n$:

$$\begin{aligned} (s, aaabbbccc, S\#) & \xrightarrow{e} (q, aaabbbccc, AA\#) [1] \\ & \xrightarrow{e} (q, aaabbbccc, aAbAc\#) [2] \\ & \xrightarrow{p} (q, aabbbccc, AbAc\#) \\ & \xrightarrow{e} (q, aabbbccc, aAbbAcc\#) [2] \\ & \xrightarrow{p} (q, abbbccc, AbbAcc\#) \\ & \xrightarrow{e} (f, abbbccc, abbbccc\#) [3] \\ & \xrightarrow{p} (f, bbccc, bbccc\#) \\ & \xrightarrow{p} (f, bbccc, bbccc\#) \\ & \xrightarrow{p} (f, bccc, bccc\#) \\ & \xrightarrow{p} (f, ccc, ccc\#) \\ & \xrightarrow{p} (f, cc, cc\#) \\ & \xrightarrow{p} (f, c, c\#) \\ & \xrightarrow{p} (\varepsilon, \#) \end{aligned}$$

Tento jazyk byl přijat po 13ti krocích, z toho byly 4 expanze a 9 vyjmutí.

Kapitola 6

Modifikace

V této části práce představím doposud nedefinované modifikace – bezstavový hluboký zásobníkový automat a paralelní bezstavový hluboký zásobníkový automat. U těchto verzí si ukážeme, jakým způsobem přijímají řetězec jazyka, uvedeme si některé jejich vlastnosti a také možnost využití v praxi.

6.1 Bezstavový hluboký zásobníkový automat

Zavedení bezstavového hlubokého zásobníkového automatu je modifikací klasického hlubokého zásobníkového automatu. Jak již název napovídá, bude se jednat o eliminaci stavů, což je v poslední době velmi aktuální oblast výzkumu. Tento automat je prakticky ekvivalentní ke klasickému hlubokému zásobníkovému automatu, který má pouze jeden stav. Jelikož se tedy stav nemůže nikdy změnit na jiný, můžeme jej odstranit. Výsledkem toho je, že každý výpočetní krok závisí pouze na obsahu zásobníku a aktuálním symbolu na čtecí pásce. Stejně jako hluboký zásobníkový automat jsou i zde dvě klasické operace – expanze a vyjmutí. Expanze opět znamená rozgenerování nejvrchnějšího neterminálního symbolu na zásobník na řetězec. A vyjmutí znamená odstranění terminálního symbolu z vrcholu zásobníku a zároveň přečtení shodného symbolu ze vstupní pásky.

V této práci budeme uvažovat nedeterministickou verzi tohoto automatu, která nepovoluje vymazávací pravidla.

Některé části této práce označují tento automat jako *SDPDA*.

6.1.1 Definice

Bezstavový hluboký zásobníkový automat je uspořádaná čtveřice ${}_{deep}M = (\Sigma, \Gamma, R, S)$, kde

$deep$ je maximální hloubka, v níž může dojít k nahrazení

Σ je vstupní abeceda

Γ je konečná množina zásobníkových symbolů, nazývaná zásobníková abeceda,

$\Sigma \subset \Gamma$, $\# \in (\Gamma - \Sigma)$, $\#$ je speciální pomocný symbol značící dno zásobníku

$S \in \Gamma$ je počáteční zásobníkový symbol

R je konečná množina pravidel tvaru $(mA \rightarrow v) \in R$, $A \in \Gamma$, $0 < m \leq deep$ (m značí v jaké hloubce zásobníku budeme nahrazovat), $v \in \Gamma^$*

Konfigurace: $\chi \Sigma^* (\Gamma - \{\#\})^* \{\#\}$

Přechod: Nechtě x, y jsou dvě konfigurace.

Automat provede vyjmutí a přejde z x do y , pokud $x = (au, az)$ a $y = (u, z)$, kde $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$. Zapišeme jako $x_p \Rightarrow y$.

Automat provede expanzi a přejde z x do y , pokud $x = (w, uAz)$ a $y = (w, uc z)$ a zároveň $(A \rightarrow c) \in R$, kde $w \in \Sigma^*$, $A \in \Gamma$, $u, c, z \in \Gamma^*$. Zapišeme jako $x_e \Rightarrow y$.

Přijímaný jazyk: Necht' ${}_nM$ je bezstavový hluboký zásobníkový automat. Jazyk přijímaný tímto automatem $L({}_nM)$ je definován jako jazyk: $L({}_nM) = \{w : w \in \Sigma^*, (w, S\#) \Rightarrow^* (\varepsilon, \#) \text{ v } {}_nM\}$. Jazyk je tedy přijímán vyprázdněním zásobníku.

6.1.2 Přijmutí jazyka

${}_2M = (\{a, b, c, \}, \{a, b, c, A, S, \#\}, R, S)$

s pravidly

1. $1S \rightarrow AB$
2. $1A \rightarrow aAb$
3. $1A \rightarrow ab$
4. $2B \rightarrow Bc$
5. $1B \rightarrow Bc$
6. $1B \rightarrow c$

Jelikož tento automat neumí přijímat řetězec $a^n b^n c^n$, ukážeme si přijímání jazyka bezkontextového $a^n b^n c^m$ pro $n \geq 1$ a $m \geq 1$.

Na vstupu máme řetězec aaabbbccccc:

$$\begin{aligned} (aaabbbccccc, S\#) & \xrightarrow{e} (aaabbbccccc, AB\#) [1] \\ & \xrightarrow{e} (aaabbbccccc, aAbB\#) [2] \\ & \xrightarrow{p} (aabbccccc, AbB\#) \\ & \xrightarrow{e} (aabbccccc, AbBc\#) [4] \\ & \xrightarrow{e} (aabbccccc, aAbbBc\#) [2] \\ & \xrightarrow{p} (abbccccc, AbbBc\#) \\ & \xrightarrow{e} (abbccccc, abbbBc\#) [3] \\ & \xrightarrow{p} (bbccccc, bbbBc\#) \\ & \xrightarrow{p} (bbccccc, bbBcc\#) \\ & \xrightarrow{p} (bccccc, bBcc\#) \\ & \xrightarrow{p} (cccc, Bcc\#) \\ & \xrightarrow{e} (cccc, Bccc\#) [5] \\ & \xrightarrow{e} (cccc, cccc\#) [6] \\ & \xrightarrow{p} (ccc, ccc\#) \\ & \xrightarrow{p} (cc, cc\#) \\ & \xrightarrow{p} (c, c\#) \\ & \xrightarrow{p} (\varepsilon, \#) \end{aligned}$$

Tento jazyk byl přijat po 17ti krocích, z toho bylo 7 expanzí a 10 vyjmutí. Tento automat je těžce nedeterministický, proto jsme jeho chování museli řídit osobně.

6.2 Paralelní bezestavový hluboký zásobníkový automat

Paralelní bezestavový hluboký zásobníkový automat spojuje dohromady vlastnosti paralelního a bezestavového hlubokého zásobníkového automatu. To v praxi znamená, že tento automat nemá stavy a jeho pravidla mohou přepisovat zároveň i více symbolů než jen jeden. Jako u klasické paralelní verze jsme zde však opět omezení maximální hloubkou *deep*. Spojení těchto dvou vlastností tento typ automatu vyzdvihuje nad ostatní. Nejenže si nemusíme udržovat informaci o tom, v jakém stavu se automat nachází, ale také můžeme využít paralelního zpracování pravidel, což samo o sobě může vést ke zrychlení.

V této práci budeme uvažovat nedeterministickou verzi tohoto automatu, která nepovoluje vymazávací pravidla.

Některé části této práce označují tento automat jako *PSDPDA*.

6.2.1 Definice

Paralelní bezestavový hluboký zásobníkový automat je uspořádaná čtveřice ${}_{deep}M_{par} = (\Sigma, \Gamma, R, S)$, kde

$deep$ je maximální hloubka, v níž může dojít k nahrazení

Σ je vstupní abeceda

Γ je konečná množina zásobníkových symbolů, nazývaná zásobníková abeceda,

$\Sigma \subset \Gamma$, $\# \in (\Gamma - \Sigma)$, $\#$ je speciální pomocný symbol značící dno zásobníku

$S \in \Gamma$ je počáteční zásobníkový symbol

R je konečná množina pravidel tvaru $(A_1, A_2 \dots A_n) \rightarrow (v_1, v_2 \dots v_n) \in R$,

$A_j \in \Gamma - (\Sigma \cap \#)$, $1 < j \leq deep$, $n \leq deep$, $v_j \in \Gamma^$*

Konfigurace: $\chi \Sigma^* (\Gamma - \{\#\})^* \{\#\}$

Přechod: Necht x, y jsou dvě konfigurace.

Automat provede vyjmutí a přejde z x do y , pokud $x = (au, az)$ a $y = (u, z)$, kde $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$. Zapišeme jako $x_p \Rightarrow y$.

Automat provede expanzi a přejde z x do y , pokud $x = (w, uABCz)$ a $y = (w, ucdez)$ a zároveň $(A, B, C) \rightarrow (c, d, e) \in R$, kde $w \in \Sigma^*$, $A, B, C \in \Gamma$, $u, c, d, e, z \in \Gamma^*$. Zapišeme jako $x_e \Rightarrow y$.

Přijímaný jazyk: Necht ${}_nM$ je paralelní bezestavový hluboký zásobníkový automat. Jazyk přijímaný tímto automatem $L({}_nM)$ je definován jako jazyk: $L({}_nM) = \{w : w \in \Sigma^*, (w, S\#) \Rightarrow^* (\varepsilon, \#) \text{ v } {}_nM\}$. Je tedy přijímán pouze vyprázdněním zásobníku.

6.2.2 Přijmutí jazyka

${}_2M = (\{a, b, c, \}, \{a, b, c, A, S, \#\}, R, S)$

s pravidly

1. $(S) \rightarrow (AA)$
2. $(A, A) \rightarrow (aAb, Ac)$
3. $(A, A) \rightarrow (ab, c)$

Na vstupu máme řetězec aaabbbccc, který patří do jazyka $a^n b^n c^n$:

$$\begin{aligned}
 (aaabbbccc, S\#) & \xrightarrow{e} (aaabbbccc, AA\#) [1] \\
 & \xrightarrow{e} (aaabbbccc, aAbAc\#) [2] \\
 & \xrightarrow{p} (aabbbccc, AbAc\#) \\
 & \xrightarrow{e} (aabbbccc, aAbbAcc\#) [2] \\
 & \xrightarrow{p} (abbbccc, AbbAcc\#) \\
 & \xrightarrow{e} (abbbccc, abbbccc\#) [3] \\
 & \xrightarrow{p} (bbbbccc, bbbccc\#) \\
 & \xrightarrow{p} (bbccc, bbccc\#) \\
 & \xrightarrow{p} (bbccc, bbccc\#) \\
 & \xrightarrow{p} (bccccc, bccccc\#) \\
 & \xrightarrow{p} (ccc, ccc\#) \\
 & \xrightarrow{p} (cc, cc\#) \\
 & \xrightarrow{p} (c, c\#) \\
 & \xrightarrow{p} (\varepsilon, \#)
 \end{aligned}$$

Tento jazyk byl přijat po 13ti krocích, z toho byly 4 expanze a 9 vyjmutí. Jak je vidět z příkladu, automat je stejně rychlý jako je klasická pralelní verze a navíc jsme nemuseli udržovat informaci o stavu, ve kterém se automat nachází.

Kapitola 7

Síla

7.0.3 Síla bezstavového hlubokého zásobníkového automatu

Lemma 1:

Pro každý zásobníkový automat X , existuje bezstavový hluboký zásobníkový automat hloubky n , ${}_nY$, který přijímá stejný jazyk jako zásobníkový automat, platí tedy $L(X) = L({}_nY)$.

Důkaz:

Je obecně známo, že zásobníkové automaty jsou stejně silné jako bezstavové zásobníkové automaty. Toto je uvedeno i v knize Elements of Compiler Design [3]. Bezstavový zásobníkový automat je snadno převeditelný na bezstavový hluboký zásobníkový automat. Stačí jen bezstavový hluboký zásobníkový automat omezit na hloubku 1, čímž dostáváme ekvivalentní bezstavový zásobníkový automat.

Věta 1:

Pro každý bezstavový hluboký zásobníkový automat hloubky n , ${}_nX$, existuje zásobníkový automat Y , který přijímá stejný jazyk jako zásobníkový automat, platí tedy $L({}_nX) = L(Y)$.

Úvaha:

Bohužel tato věta se mi ještě nepodařila dokázat. Tento důkaz je poměrně složitý. Zásadním problémem je právě ona hloubka bezstavového hlubokého zásobníkového automatu. I po značném úsilí jsem nenašla jazyk, který by byl silnější než bezkontextový, který by tuto větu vyvracel.

Obecně používanou metodou v takovém případě je možnost uložení hloubky či neterminálních symbolů do stavů, které zásobníkový automat má. Zatím jsem nenalezla přesný mechanismus, při kterém by nevznikla nekonečná množina stavů. Definice zásobníkového automatu totiž vyžaduje konečnou množinu stavů.

Věta 2:

Pro každý hluboký zásobníkový automat hloubky n , ${}_nX$, existuje zásobníkový automat Y , který přijímá stejný jazyk jako zásobníkový automat, platí tedy $L({}_nX) = L(Y)$.

Úvaha:

Dovoluji si tvrdit, že tato věta neplatí. Ukažme si toto tvrzení na ukázkovém jazyce $a^n b^n c^n$, který spadá do kategorie kontextových jazyků, které nejsou bezkontextové. Tento jazyk zvládá přijímat hluboký zásobníkový automat:

$${}_2M = (\{s, q, p, f\}, \{a, b, c, \}, \{a, b, c, A, S, \#\}, R, s, S, \{f\})$$

s pravidly

1. $1sS \rightarrow qAA$

2. $1qA \rightarrow paAb$
3. $1qA \rightarrow fab$
4. $2pA \rightarrow qAc$
5. $1fA \rightarrow fc$

Jelikož tento jazyk dokáže zpracovat hluboký zásobníkový automat, měl by jej umět zvládat i bezstavový hluboký zásobníkový automat, pokud by tyto automaty byly stejně silné.

Když bychom chtěli tento jazyk přijímat bezstavovým hlubokým zásobníkovým automatem vyvstane několik problémů. Řetězce tohoto jazyka obsahují vždy stejný počet písmen a , b i c a to tak, že nejprve jsou v řetězci umístěna všechna písmena a , pak b a nakonec c . Využijeme-li pravidla, která v tomto případě používal hluboký zásobníkový automat, a odstraníme z nich stavy, nastane problém – například při aplikaci pravidla $1A \rightarrow aAb$, po kterém by mělo bezprostředně následovat pravidlo $2A \rightarrow Ac$. Bez stavů však tuto posloupnost nemusíme dodržovat. Dostaneme proto i různé další řetězce než jen $a^n b^n c^n$.

Shrnu zde jistá fakta, která nám naznačují, že tento jazyk bezstavovým hlubokým zásobníkovým automatem přijímat nepůjde:

- z jednoho neterminálního symbolu nikdy po aplikování jednoho pravidla nedostaneme řetězec $a^n b^n c^n$,
- z jednoho neterminálního symbolu dostaneme maximálně shodnou posloupnost $a^n b^n$, například pomocí pravidla $1A \rightarrow aAb$,
- kdybychom se snažili generovat znaky, například pomocí pravidel $1A \rightarrow aAbC$ a $1C \rightarrow c$, nastane problém, kdy na zásobníku dostaneme $aaabCbCbC$, tedy c nebude na správném místě,
- hloubka nám v tomto případě brání možnosti řízení aplikace pravidel.

Theorem 1:

Bezstavový hluboký zásobníkový automat je minimálně stejně silný jako zásobníkový automat, což je viditelné z prvního lemmatu.

Otevřený problém:

Základní otázkou k dalšímu zkoumání je, zda síla bezstavového zásobníkového automatu končí na síle bezkontextových gramatik nebo je schopna tuto sílu překročit.

7.0.4 Síla paralelního bezstavového hlubokého zásobníkového automatu

Lemma 1:

Pro každý hluboký zásobníkový automat hloubky $n \geq 1$, ${}_nX$, existuje paralelní bezstavový hluboký zásobníkový automat hloubky $n + 1$, ${}_{n+1}Y$, který přijímá stejný jazyk jako hluboký zásobníkový automat, platí tedy $L({}_nX) = L({}_{n+1}Y)$.

Důkaz:

Mějme hluboký zásobníkový automat ${}_nX = (Q_1, \Sigma_1, \Gamma_1, R_1, s_1, S_1, F_1)$, kde $n \geq 1$ a množina neterminálních symbolů je $N_1 = \Gamma - (\Sigma + \#)$.

Zavedme paralelní bezstavový hluboký zásobníkový automat hloubky $n + 1$

${}_{n+1}Y = (\Sigma_2, \Gamma_2, R_2, S_2)$, kde

$\Sigma_2 = \Sigma_1$

$\Gamma_2 = \Gamma_1 \cup Q_1$, množinou neterminálních symbolů pak je $N_2 = \Gamma - (\Sigma + \#)$

$S_2 \in \Gamma_2$

a R_2 je konstruováno pomocí následujících pravidel

1. Vytvoříme množinu R_2 .
2. U pravidel z R_1 , která na levé straně obsahují počáteční zásobníkový symbol S_1 a zároveň počáteční stav s_1 , $1s_1S_1 \rightarrow qu$, kde $s, q \in Q_1, A, S_1 \in N_1, u \in \Gamma_1^*$ odstraníme stav s_1 a dostaneme pravidlo $1S \rightarrow qu$, kde $q \in Q_1, A \in N_1, u \in \Gamma_1^*$, které prozatím přidáme do původní množiny pravidel R_1 .
3. Pro každé pravidlo hloubky 1 z R_1 , $1qA \rightarrow pu$, kde $q, p \in Q_1, A \in N_1, u \in \Gamma_1^*$, zavedeme nové pravidlo $QA \rightarrow Pu$, kde $Q, A, P \in N_2, u \in \Gamma_2^*$ a to přidáme do R_2 .
4. Pro každé pravidlo hloubky větší než 1, $3pA \rightarrow qu$, kde $p, q \in Q_1, A \in N_1, u \in \Gamma_1^*$ zavedeme množinu pravidel $pZA \rightarrow qZAu$ takovou, že $Z \subseteq (N_2 - Q_1)^*$, $A \in N_2, p, q \in Q_1, u \in \Gamma_1^*$ a zároveň platí na levé straně pravidla $|ZA| = \text{hloubce pravidla}$. Stavů těchto pravidel upravíme stejně jako v pravidle 3, dostaneme tedy $PZA \rightarrow QZAc$, $P, A, Q \in N_2, Z \in (N_2 - Q_1)^*, u \in \Gamma_2^*$ a tato pravidla umístíme do R_2 .
5. Pro optimalizaci můžeme odstranit pravidla, která nikdy nebudou aplikovatelná, tento krok je pro samotný důkaz zbytečný.

Algoritmus postupně bere pravidla hlubokého zásobníkového automatu a odstraňuje z nich stavy.

Pravidla, ve kterých se nacházel počáteční symbol a zároveň i počáteční stav, jsou po odstranění stavu přiřazena zpět do původní množiny, jelikož budou dále ještě upravována. Tato pravidla nám umožní začít přijímat jazyk ze stavu, kdy je v zásobníku uložen pouze symbol dna zásobníku a startující symbol, nemáme tedy nikde uveden stav. Do takovéto situace se automat dostane pouze před použitím prvního pravidla. Ostatní pravidla nám již na zásobník uloží svůj stav.

Všechna pravidla hloubky 1 (tedy i pravidla, ze kterých jsme již odstranili stav na levé straně – tato pravidla mají stav na pravé straně) uchovávají svůj stav do neterminálního symbolu, který se nenacházel v neterminálních symbolech hlubokého zásobníkového automatu. To v praxi znamená, že stavy z hlubokého zásobníkového automatu jsou nyní v paralelním bezstavovém hlubokém zásobníkovém automatu přidány k množině zásobníkové abecedy. Nyní na levé straně každého nově přidaného pravidla dostáváme dva neterminální symboly – jeden značící stav a druhý je neterminálním symbolem, který má být přepsán. Tuto modifikaci můžeme použít jen při paralelní verzi, která je schopna přepsat více zásobníkových symbolů zároveň.

Pravidla hloubky větší než 1 upravíme opět pomocí převedení stavů na neterminální symboly. Na příkladu si ukážeme, jak dále postupovat. Máme například pravidlo $3pA \rightarrow qu$, pokud bychom jen uložili stav na zásobník, mohlo by se stát $PA \rightarrow Qu$. To by ovšem znamenalo, že neterminální symbol A , který se měl nacházet v hloubce 3 nyní bude v hloubce 2. Pokud se na to podíváme tak, že na prvním místě zásobníku jsme přidali stav, jedná se dokonce o zásobníkový symbol, který se původně nacházel v hloubce 1. Tím bychom ovšem dostali úplně jiné pravidlo. Musíme proto provést další úpravu. A to použít řetězec, který je kombinací neterminálních symbolů, který doplníme mezi P a A tak, abychom A dostali do potřebné hloubky. Předpokládejme, že máme množinu neterminálních symbolů A, B, C, P, Q a předpokládejme, že P a Q jsou neterminální symboly nahrazující stavy, tedy

mohou se nacházet v zásobníku jen na prvním místě, pak můžeme stavy P a Q vynechat. Zbyly nám jen tři neterminální symboly A, B, C . Naše pravidlo bylo původně hloubky 1, tedy délka řetězce musí být o jedno menší než je hloubka. Možné řetězce z neterminálů tedy jsou $AA, BB, CC, AB, AC, BA, BC, CA, CB$. Zavedeme nová pravidla s těmito řetězci. Dostaneme tedy například nové pravidlo $PAAA \rightarrow QAAu$. Tím zachováme původní stav zásobníku a zaměníme jen neterminální symbol v určité hloubce.

Mezi různé optimalizace tohoto algoritmu by mohlo patřit vynechání pravidel, u kterých by bylo jasné vidět, že se automat nikdy nedostane do situace, kdy by mohly být aplikovatelné. Nebo můžeme spojit pomocí stavů několik pravidel do jednoho.

Lemma 2:

Pro každé $n \geq 1$ a každý paralelní bezstavový hluboký zásobníkový automat ${}_nX_{par}$ existuje paralelní hluboký zásobníkový automat ${}_nY_{par}$, který přijímá stejný jazyk jako paralelní bezstavový hluboký zásobníkový automat. Platí tedy $L({}_nX_{par}) = L({}_nY_{par})$.

Důkaz:

Mějme paralelní bezstavový hluboký zásobníkový automat ${}_nX_{par} = (\Sigma_1, \Gamma_1, R_1, S_1)$, kde $n \geq 1$. Zavedme paralelní hluboký zásobníkový automat hloubky n

${}_nX_{par} = (Q, \Sigma_2, \Gamma_2, R_2, s, S_2, F)$, kde

$$Q = \{s\}$$

$$\Sigma_2 = \Sigma_1$$

$$\Gamma_2 = \Gamma_1$$

$$S_2 \in \Gamma_2$$

$$F = \{s\}$$

a R_2 je konstruováno následovně

1. Vytvoříme množinu R_2 .
2. Pro každé pravidlo R_1 , zavedeme nové pravidlo, kde na levou i pravou stranu přidáme stav s . Takže z pravidla $PAAA \rightarrow QAAu$ dostaneme $sPAAA \rightarrow sQAAu$. Takové pravidlo vložíme do R_2 .

Algoritmus jednoduše doplní ke všem pravidlům jediný stav. Paralelní bezstavový hluboký zásobníkový automat je schopen samotného řízení, přidání stavu proto nemá žádný vliv na jeho sílu.

Theorem 1:

Paralelní hluboký zásobníkový automat je stejně silný jako neparalelní.

Toto tvrzení vychází z článku [4].

Theorem 2:

Pro každé $n \geq 1$ a každý hluboký zásobníkový automat existuje paralelní bezstavový hluboký zásobníkový automat hloubky maximálně $n + 1$.

Toto tvrzení je založeno na předchozím theoremu a výše uvedených lemmat.

Theorem 3:

$${}_nPSDPDA \subseteq_n PSDPDA \subseteq CS$$

Toto tvrzení vychází z vlastností hlubokých zásobníkových automatů, jejichž síla roste s hloubkou.

Theorem 4:

$${}_1PSDPDA = CF$$

Omezení hloubky na velikost 1 snižuje sílu na sílu bezkontextových jazyků. Je tedy ekvivalentní zásobníkovému automatu.

Kapitola 8

Ukázka

8.1 Převod hlubokého zásobníkového automatu na paralelní bezestavový hluboký zásobníkový automat

Nyní si ukážeme, jak se dá klasický hluboký zásobníkový automat převést na paralelní bezestavovou verzi s klasickými pravidly. Nechť ${}_2M$ je hluboký zásobníkový automat, pro který platí:

${}_2M = (\{s, q, p, f\}, \{a, b, c, \}, \{a, b, c, A, S, \#\}, R, s, S, \{f\})$. Pravidla tohoto automatu jsou: $1sS \rightarrow qAA$, $1qA \rightarrow paAb$, $1qA \rightarrow fab$, $2pA \rightarrow qAc$, $1fA \rightarrow fc$.

Převod:

Definujme si nyní paralelní bezestavový hluboký zásobníkový automat ${}_2X$ jako:

${}_2X = (\{a, b, c, \}, \{S_q, Q_q, P_q, f_q, a, b, c, A, S, \#\}, R, S, \})$,
jehož pravidla sestrojíme pomocí následujícího postupu.

1. Vyvoříme množinu R_2 .
2. Přidáme do původní množiny pravidlo $1S \rightarrow qAA$.
3. Nyní do množiny R_2 přidáme upravená pravidla, která byla původně hloubky 1 ($S \rightarrow (Q_qAA)$, $(S_q, S) \rightarrow (Q_q, AA)$, $(Q_q, A) \rightarrow (P_q, aAb)$, $(Q_q, A) \rightarrow (F_qab)$, $(F_qA) \rightarrow (F_qc)$).
4. Upravíme pravidla hloubky větší než jedna. $2pA \rightarrow qAc$ upravíme nejprve na $(P_q, A) \rightarrow (Q_q, Ac)$ a nyní doplníme dostatečný počet neterminálních symbolů dle hloubky, tedy jeden $(P_q, A, A) \rightarrow (Q_q, A, Ac)$ a $(P_q, S, A) \rightarrow (Q_q, S, Ac)$.

Nová pravidla jsou tedy: $(S) \rightarrow (Q_qAA)$, $(S_q, S) \rightarrow (Q_q, AA)$, $(Q_q, A) \rightarrow (P_q, aAb)$, $(Q_q, A) \rightarrow (F_qab)$, $(F_qA) \rightarrow (F_qc)$, $(P_q, A, A) \rightarrow (Q_q, A, Ac)$ a $(P_q, S, A) \rightarrow (Q_q, S, Ac)$.

Kapitola 9

Struktura předkladačů

Překladač je program, který čte zdrojový program a překládá ho na cílový program. Tyto programy jsou vzájemně funkčně ekvivalentní. Obvykle je zdrojový program vyšší programovací jazyk a je překládán do strojového kódu či assembleru. Překladač nejprve zkontroluje korektnost vstupu pomocí lexikálního, syntaktického a sémantického analyzátoru. Pokud analýza kódu proběhne úspěšně, provede se transformace kódu za pomoci generátoru vnitřního kódu, optimalizátoru a generátoru cílového kódu. Jednotlivé části si nyní ještě více rozebereme.

9.1 Lexikální analýza

Lexikální analyzátor rozdělí zdrojový program na lexémy, což jsou logicky oddělené lexikální jednotky jako například identifikátory, čísla, klíčová slova, operátory atd. Jestliže jsou lexémy ve správném tvaru, jsou dále reprezentovány pomocí tokenů, přičemž některé z nich mohou nést další atributy, které token podrobněji specifikují. Tokeny jsou dále poslány syntaktickému analyzátoru.

Lexikální analýza rozpoznává jednotlivé lexémy pomocí tzv. scanneru, který čte sekvenci znaků ze zdrojového programu. K tomuto se využívá především konečný automat.

9.2 Syntaktická analýza

Syntaktická analýza kontroluje syntaktickou strukturu programu, který je nyní reprezentován pomocí tokenů. Pro kontrolu syntaxe se většinou využívají moderní matematické postupy. Syntaxe jazyka je obvykle specifikována pomocí gramatických pravidel, na kterých je založen analyzátor. Jestliže je řetězec tokenů správný, je k němu nalezen odpovídající derivační strom, jehož listy obsahují tokeny.

K analýze můžeme využít princip shora dolů, který vytváří strom od kořene směrem dolů ke koncovým listům a nebo princip zdola nahoru, který začíná budovat strom od koncových listů a pokračuje nahoru směrem ke kořeni.

Program je syntakticky správně jedině v případě, že nalezneme kompletní derivační strom. V opačném případě překladač oznámí uživateli hlášení o syntaktické chybě.

Typickým mechanismem pro kontrolu syntaxe je zásobníkový automat a různé jeho modifikace.

9.3 Sémantická analýza

Sémantická analýza kontroluje, zda derivační strom odpovídá sémantickým pravidlům zdrojového jazyka. Mezi typické kontroly patří kontrola deklarací proměnných nebo kontrola typů (zde může být prováděna i implicitní konverze). Výstupem sémantického analyzátoru je abstraktní syntaktický strom.

Většina překladačů využívá syntaxí řízený překlad, při kterém je provádění sémantických akcí a generování abstraktního syntaktického stromu řízeno již zmíněným syntaktickým analyzátozem.

9.4 Generátor vnitřního kódu

Generátor vnitřního kódu mění abstraktní syntaktický strom na vnitřní reprezentaci programu nazývanou vnitřní kód. Jedná se o mezikrok mezi zdrojovým a cílovým kódem. Vnitřní kód umožňuje lepší adaptaci překladače na různé stroje. Jedná se obvykle o 3adresný kód, který reprezentuje vstupní program v sekvenci jednoduchých instrukcí. Tato sekvence se pak snadněji optimalizuje.

9.5 Optimalizátor

Optimalizace vnitřního kódu většinou obsahuje několik podfází, které jsou aplikovány i opakovaně. Optimalizace můžeme obecně rozdělit na dva typy – závislé na stroji a nezávislé na stroji. Příkladem optimalizace je třeba šíření konstanty, šíření kopírováním či eliminace mrtvého kódu.

Tato část může samotný překlad zpomalit, některé překladače proto povolují tuto fázi vypnout nebo ji dokonce vůbec nemají.

9.6 Generátor cílového kódu

Generátor cílového kódu převede optimalizovaný vnitřní kód do cílového jazyka, například assembleru. To znamená, že jednotlivé konstrukce zdrojového kódu jsou převedeny na sekvenci jednoduchých instrukcí cílového jazyka, který je funkcionálně shodný se zdrojovým kódem. Cílový kód je závislý na stroji, na kterém je prováděn. Různé stroje mohou mít různé umístění například proměnných.

Kapitola 10

Návrh programu

V předchozí kapitole je popsána struktura překladače. V této části nás zajímá především syntaktická analýza, kde se, jak již bylo zmíněno, využívají zásobníkové automaty a jejich případné modifikace. Laicky řečeno syntaktická analýza kontroluje, zda je program, nebo některé jeho části, prvkem zadaného jazyka. Jazyk je definován pomocí posloupnosti pravidel. Tato pravidla se postupně aplikují, a pokud se za pomoci těchto pravidel přečte celý vstupní řetězec a zůstane prázdný zásobník, je řetězec přijat. Klasický syntaktický analyzátor by k danému řetězci i vygeneroval odpovídající derivační strom, avšak pro naši ukázkou je tato část zbytečná.

Jelikož v této práci byly zavedeny dva automaty, které mají odlišné vlastnosti, budou také naimplementovány dva programy. Pro bezstavový hluboký zásobníkový automat to bude program SDPDA(Stateless Deep Pushdown Automaton) a pro paralelní verzi PSDPDA(Parallel Stateless Deep Pushdown Automaton). Zásadním rozdílem těchto dvou programů je expanze. Zatímco SDPDA jednoduše rozgeneruje jeden symbol na zásobníku, PSDPDA jich musí umět v jednom kroku rozgenerovat více.

10.1 SDPDA

Automat SDPDA má problémy se zvládnutím jazyků, které jsou silnější než bezkontextové. Ukážeme si proto jeho využití na jazyce bezkontextovém. Využijeme příklad z definice tohoto automatu – tedy jazyk $a^n b^n c^m$.

Program dostane na standardní vstup řetězec. Pomocí pravidel bude zkontrolováno, zda tento řetězec patří do daného jazyka. Jelikož je řízení tohoto automatu nedeterministické, bude pravidla vybírat uživatel. Uživatel bude jasně vidět, co se v automatu děje, jelikož bude po každém kroku vypsan obsah zásobníku a zůstatek vstupního řetězce, který ještě nebyl přijat.

10.2 PSDPDA

Automat PSDPDA je dostatečně vhodný pro zvládnutí jazyků silnějších než bezkontextových. Opět byl v tomto případě vybrán jazyk, který je uvedený už v teoretické části u tohoto jazyka $a^n b^n c^n$.

Princip programu je stejný jako u SDPDA. Uživatel zadá vstupní řetězec a vybírá pravidla, pomocí kterých by mohl být řetězec akceptován. Opět bude po každém kroku vypsan aktuální stav zásobníku i vstupního řetězce.

Kapitola 11

Implementace

Programy jsou napsány v jazyce C++. Testování proběhlo na školním serveru merlin. Jedná se o konzolové programy, jelikož jde pouze o práci s textovými řetězci. Programy se spouští bez parametrů, protože jazyky jsou již předem specifikovány uvnitř programu.

Oba programy obsahují třídu zastupující automat. Tento automat má v sobě uložený zásobník, v našem případě zastoupený vektorem. Abstraktní datový typ zásobník totiž neumožňuje zasahování do hloubky.

U každého symbolu, který bude uložen v zásobníku, bude uvedeno, zda se jedná o terminální symbol či neterminální symbol. Programy automaticky předpokládají, že symboly A-Z jsou neterminální symboly. Všechno ostatní jsou terminální symboly.

Před každou žádostí o zadání čísla pravidla je vypsán i seznam pravidel s jejich očíslováním. Z tohoto seznamu uživatel vybírá pravidla, která budou aplikována. Tímto řídí uživatel expanzi. Operace odstranění terminálního symbolu z vrcholu zásobníku a odstranění ze vstupní pásky je prováděno automaticky.

11.1 Odlišnosti SDPDA a PSDPDA

Hlavní odlišností těchto dvou programů je implementace metody expanze. Tato metoda má jeden parametr – číslo pravidla. Jestliže je pravidlo možné provést, provede se. Pokud se neprovede, tak zásobník zůstává v původní podobě.

Expanze u SDPDA zahrnuje rozgenerování jednoho zásobníkového symbolu na řetězec. Stačí procházet zásobník od vrcholu. Pokud narazíme na správný neterminální symbol, který bude navíc umístěn ve správné hloubce, můžeme jej rozgenerovat. V praxi to znamená, že jednoduše odstraníme daný symbol a na jeho místo postupně vkládáme dané symboly z pravé strany pravidla.

Expanze u PSDPDA je o trošku složitější. Než provedeme samotné nahrazení, musíme zkontrolovat, zda se na zásobníku nachází všechny symboly ve správné hloubce. Pokud symboly odpovídají, jsou postupně odstraňovány. Musíme začít od nejhlubšího symbolu a rovnou ho rozgenerujeme. Pokud bychom začali shora, mohlo by se stát, že po nahrazení jednoho symbolu řetězcem se změní hloubka všech ostatních symbolů.

Kapitola 12

Závěr

V této práci byly prezentovány dvě nové modifikace hlubokých zásobníkových automatů – bezstavový hluboký zásobníkový automat a paralelní bezstavový hluboký zásobníkový automat. Ukázali jsme si, jak tyto automaty přijímají řetězec z určitého jazyka. Uvedli jsme si některé vlastnosti týkající se síly a ukázali jsme si na jednoduchém příkladu praktické použití těchto automatů. Nyní zde shrneme základní poznatky o daných modifikacích, které vyplývají z této práce.

Bezstavový hluboký zásobníkový automat nám svými vlastnostmi nepřináší nic nového, spíše naopak. Jeho přesnou sílu se bohužel nedokázalo přesně dokázat, ale dle ukázky některých vlastností tušíme, že sílu hlubokých zásobníkových automatů rozhodně nepřesáhne. Nejspíše se této síle ani nevyrovná. Navíc je tato verze hluboce nedeterministická a obtížně říditelná. Zasahování do hloubky zde spíše než zvýšení síly přináší nevýhodu v řízení chodu automatu. Tyto vlastnosti nejspíše předurčují bezstavový hluboký zásobníkový automat k zániku. V praxi je daleko snadnější použít klasický zásobníkový automat, který má navíc jasně definovatelný determinismus.

Paralelní bezstavový hluboký zásobníkový automat naopak oproti neparalelní bezstavové verzi je schopen dosáhnout síly hlubokých zásobníkových automatů, a to v nejhorším případě pro hloubku o 1 větší než u klasického automatu. Stejně jako stavová verze toho automatu je i tento automat rychlejší v přijímání řetězce a navíc zde ani nemusíme kontrolovat stavy, čímž odpadá jedna operace navíc. Nevýhodou se může jevit snad jen nutnost kontrolovat při výběru pravidla stav zásobníku, a to například až do hloubky *deep*, přičemž se jedná o hloubku neterminálních symbolů, nikoliv všech symbolů na zásobníku. Pokud by jedno paralelní pravidlo obsahovalo několik pravidel z klasického automatu, bude se stav zásobníku kontrolovat jen jednou a ne několikrát, tím je kontrola zásobníku prakticky ekvivalentní, ne-li rychlejší.

Další vývoj práce by mohl směřovat k dosažení výsledku o přesné síle bezstavových hlubokých zásobníkových automatů. Z tohoto důkazu by pak mohlo vyplynout, že paralelismus má opravdu schopnost řízení podobně jako například použití stavů automatu. Dále by pak bylo vhodné vyzkoušet využití paralelismu na strojích s paralelní architekturou, kde by se paralelismus nemusel již pouze simulovat, ale mohl by vést k opravdu velkému zrychlení. Mezi zajímavé problematiky také patří to, jaký vliv by mohly na tyto automaty mít zkracující pravidla.

Literatura

- [1] MEDUNA, A.: *Automata and languages*. Springer, 2000, ISBN 1-85233-074-0.
- [2] MEDUNA, A.: Deep pushdown automata. *Acta Informatica*, ročník 2006, č. 98, 2006: s. 114–124, ISSN 0001-5903.
- [3] MEDUNA, A.: *Elements of Compiler Design*. Auerbach Publications, 2008, ISBN 1-4200-6323-5.
- [4] SOLÁR, P.: *Paralelní hluboké zásobníkové automaty*. Diplomová práce, Brno: Vysoké učení technické. Fakulta informačních technologií, 2007.
- [5] ZEMEK, P.: *Canonical Derivations in Programmed Grammars*. Diplomová práce, Brno: Vysoké učení technické. Fakulta informačních technologií, 2008.

Dodatek A

CD

A.1 Programy

A.2 Elektronická forma tohoto dokumentu