VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ČÁSTEČNĚ PARALELNÍ HLUBOKÉ ZÁSOBNÍKOVÉ AUTOMATY

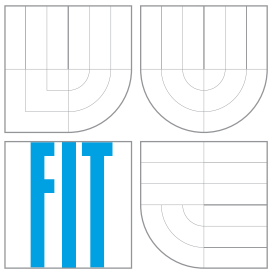
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MIROSLAV ŠEVČÍK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ČÁSTEČNĚ PARALELNÍ HLUBOKÉ ZÁSOBNÍKOVÉ AUTOMATY

SEMI-PARALLEL DEEP PUSHDOWN AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIROSLAV ŠEVČÍK
AUTHOR

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

BRNO 2012

Abstrakt

Tato práce zavádí částečně paralelní hluboké zásobníkové automaty jako rozšíření sekvenčních hlubokých zásobníkových automatů. Toto rozšíření spočívá v tom, že oproti sekvenční verzi je automat schopen provést expanzi n nejvýše položených nonterminálních symbolů na zásobníků současně. Tyto automaty jsou schopné rozpoznat třídu jazyků generovanou n-omezenými stavovými gramatikami stejně jako sekvenční hluboké zásobníkové automaty. Výhoda částečně paralelních hlubokých zásobníkových automatů je ale ve vyšší rychlosti. Dále je v této práci popsána implementace aplikace, která simuluje činnost těchto automatů.

Abstract

This work introduces semi-parallel deep pushdown automata as an extension of sequential deep pushdown automata. Unlike the sequential automaton, a semi-parallel one is able to make expansions of n nonterminal symbols on the pushdown top simultaneously. These automata, same as sequential deep pushdown automata, define the family of languages generated by n-limited state grammars. The advantage of semi-parallel deep pushdown automata lies in higher speed. This work also describes the implementation of an aplication that simulates operation of these automata.

Klíčová slova

gramatika, stavová gramatika, automat, zásobníkový automat, hluboký zásobníkový automat, částečně paralelní hluboký zásobníkový automat

Keywords

grammar, state grammar, automaton, pushdown automaton, deep pushdown automaton, semi-parallel deep pushdown automaton

Citace

Miroslav Ševčík: Částečně paralelní hluboké zásobníkové automaty, bakalářská práce, Brno, FIT VUT v Brně, 2012

Částečně paralelní hluboké zásobníkové automaty

Prohlášení

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

.....
Miroslav Ševčík
14. května 2012

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu, panu Prof. RNDr. Alexandru Medunovi CSc. za konzultace a pomoc poskytnutou při řešení této práce.

Obsah

1 Úvod **2**

2 Teoretický základ **4**

3 Gramatiky **5**
3.1 Chomského hierarchie . . . 5
3.2 Bezkontextová gramatika . . . 6
3.3 Stavová gramatika . . . 6

4 Automaty **8**
4.1 Konečný automat . . . 8
4.2 Zásobníkový automat . . . 9
4.3 Hluboký zásobníkový automat . . . 9
4.3.1 Příklad . . . 10

5 Částečně paralelní hluboké zásobníkové automaty **12**
5.1 Definice . . . 12
5.2 Konfigurace . . . 12
5.3 Pravidla . . . 13
5.4 Přechody . . . 13
5.5 Přijímané jazyky . . . 13
5.6 Síla . . . 14
5.6.1 Shrnutí . . . 15
5.7 Rychlost . . . 15
5.8 Příklady . . . 16
5.8.1 Příklad 1 . . . 16
5.8.2 Příklad 2 . . . 16
5.8.3 Příklad 3 . . . 17

6 Aplikace **18**
6.1 Analýza a návrh řešení . . . 18
6.2 Vstupní soubory . . . 19
6.3 Datové typy . . . 19
6.4 Implementace . . . 20

7 Závěr **22**
7.1 Praktická část . . . 22

A Obsah CD **24**

Kapitola 1

Úvod

Od dob prvních počítačů existuje potřeba, nějakým způsobem počítači sdělit, co má vykonat. V důsledku této potřeby vznikly programovací jazyky. Programovací jazyky jsou komunikačním nástrojem mezi programátorem, který používá programovací jazyk k zápisu algoritmů, a strojem (počítačem), který program provede (interpretuje). Avšak aby mohl být nějaký jazyk strojově zpracovatelný, musí být přesně matematicky popsán konečnou množinou pravidel. Takový jazyk nazýváme formální jazyk.

Abychom mohli s formálními jazyky pracovat v praxi, potřebujeme konečné prostředky pro jejich specifikaci. Konečné jazyky můžeme jednoduše popsat výčtem všech jejich řetězců (slov). Taková specifikace je ale nemožná pro nekonečné jazyky a prakticky nereálná i pro rozsáhlé konečné jazyky. Proto se nejčastěji pro popis (konečných i nekonečných) formálních jazyků využívají dva hlavní modely a sice gramatiky a automaty (pro regulární jazyky můžeme použít i regulární výrazy). Každý z těchto modelů definuje jazyk jiným způsobem. Gramatika generuje řetězce (slova) daného jazyka, zatímco automat je schopen rozpoznat, zda řetězec, který mu předáme na vstupu, patří do jazyka definovaného (přijímaného) tímto automatem. V kontextu programovacích jazyků gramatiky používáme jako model pro popis syntaxe programovacích jazyků. Naproti tomu automaty jsou nástoj, který provádí syntaktickou analýzu programů. Tedy je schopen rozpoznat, jestli vytvořený program splňuje syntaktická pravidla definovaná gramatikou. Automaty mají v oblasti překladačů velice významnou roli, neboť různé typy automatů často realizují části překladačů (viz konečný automat jako lexikální analyzátor, zásobníkový automat a jeho modifikace jako syntaktický analyzátor).

Podle Chomského hierarchie (viz sekce 3.1) existuje pro každý typ gramatiky, odpovídající typ automatu, který přijímá jazyk generovaný touto gramatikou. Tato práce zavádí částečně paralelní hluboké zásobníkové automaty, které jsou rozšířenou verzí sekvenčních hlubokých zásobníkových automatů zavedených v [2]. Tyto typy automatů jsou schopny rozpoznat jazyky, které tvoří nekonečnou hierarchii jazyků mezi jazyky bezkontextovými a kontextovými. Tyto jazyky jsou generovány tzv. stavovými gramatikami.

Kapitola 2 opakuje základní pojmy z oblasti formálních jazyků, nutné pro pochopení dalších částí této práce a vysvětluje pojmy a značení používané ve zbytku této práce.

V kapitole 3 je blíže popsána chomského hierarchie formálních gramatik (a jim odpovídajících automatů), aby čtenář lépe pochopil, kde se nachází v této hierarchii nachází automaty (a gramatiky), které jsou jádrem této práce. Dále je zde blíže popsána bezkontextová gramatika a její rozšíření – stavová gramatika.

Kapitola 4 popisuje stěžejní typy automatů používaných v oblasti formálních jazyků a sice konečný a zásobníkový automat. Dále popisuje hlukoký zásobníkový automat, ze

kterého částečně paralení hlukoký zásobníkový automat vychází.

V kapitole 5, která je jádrem této práce, je zaveden částečně paralení hlukoký zásobníkový automat. Najdeme zde jeho definicí, typ přijímaných jazyků, tvar pravidel a způsob jejich aplikace, srovnání síly a rychlosti se sekvenčními hlubokými zásobníkovými automaty a na závěr několik příkladů.

V kapitole 6 je popsán rozbor, návrh a implementace praktické části této práce. Jsou zde zejména popsány použité datové stuktury a algoritmy simulující činnost automatu.

Kapitola 2

Teoretický základ

V této kapitole budou zopakovány a rozebrány některé základní teoretické pojmy z oblasti formálních jazyků nutné pro pochopení dalších částí této práce. Bližší vysvětlení můžete nalézt např. v [4].

Je dána množina Q. Potom $card(Q)$ označuje počet symbolů množiny Q. Písmenem I budeme označovat množinu všech kladných celých čísel.

Definice 2.0.1. *Abeceda* je konečná, neprázdná množina prvků, které nazýváme symboly.

Řetězcem nad danou abecedou rozumíme konečnou posloupnost symbolů této abecedy. Prázdnou posloupnost symbolů, tedy posloupnost, která neobsahuje žádný symbol, nazýváme prázdný řetězec a označujeme ho písmenem ε. Prázdný řetězec je řetězec nad každou abecedou. Formálně řetězec definujeme takto:

Definice 2.0.2. Nechť Σ je abeceda.

1. ε je řetězec nad abecedou Σ
2. pokud x je řetězec nad Σ a $a \in \Sigma$, potom xa je řetězec nad abecedou Σ

Symbol Σ^* značí množinu všech řetězců nad abecedou Σ včetně prázdného řetězce ε a symbol Σ^+ množinu všech všech řetězců nad abecedou Σ vyjma prázdného řetězce.

Nechť $w \in \Sigma^*$ je řetězec, potom $|w|$ značí délku, neboli počet symbolů, řetězce w, $alph(w)$ je množina symbolů vyskytujících se v w. Nechť $W \subset \Sigma$, potom $occur(w, W)$ označuje počet výskytů symbolů z W v řetězci w.

Definice 2.0.3. Nechť Σ^* značí množinu všech řetězců nad Σ, potom každá podmnožina $L \in \Sigma^*$ je *jazyk* nad Σ. Jazykem může být libovolná podmnožina řetězců nad abecedou.

Kapitola 3

Gramatiky

Gramatika jako prostředek pro reprezentaci jazyků splňuje základní požadavek pro reprezentaci jazyků – konečnost reprezentace. Každá gramatika využívá dvou navzájem disjunktních abeced: množiny N nonterminálních a množiny T terminálních symbolů. Dále obsahuje konečnou množinu pravidel, pomocí kterých generuje daný jazyk. Jednotlivé řetězce se pak vytvářejí postupnou aplikací těchto pravidel. Podklady pro tuto kapitolu byly čerpány z [4, 3].

3.1 Chomského hierarchie

Vytvořil ji v roce 1956 americký lingvista Noam Chomsky. Chomského hierarchie formálních gramatik je hierarchie tříd formálních gramatik generujících formální jazyky. Čerpáno z [1]. Chomského hierarchie dělí gramatiky na následující typy:

Gramatiky typu 0 zahrnují všechny formální gramatiky. Tyto gramatiky generují jazyky, které mohou být rozpoznané nějakým Turingovým strojem[1]. Jazyky definované těmito gramatikymi jsou rekurzivně spočetné.

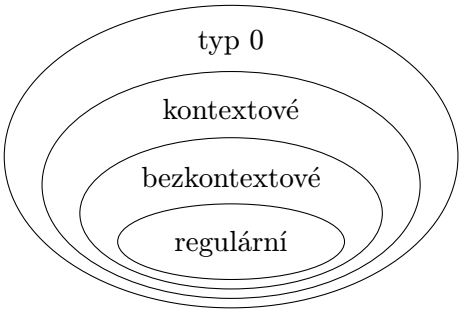
Gramatiky typu 1 (kontextové gramatiky) generují kontextové jazyky. Tyto jazyky jsou rozpoznatelné lineárně ohraničeným Turingovým strojem.

Gramatiky typu 2 (bezkontextové gramatiky) generují bezkontextové jazyky. Tento typ jazyků je rozpoznatelný nedeterministickými zásobníkovými automaty.

Gramatiky typu 3 (regulární gramatiky) generují regulární jazyky. Tyto jazyky jsou rozpoznatelné konečnými automaty.

Jak je naznačeno na obrázku 3.1, každý typ je nadmnožinou typu následujícího.

[1] Turingův stroj je teoretický model vytvořevý matematikem Alanem Turingem. Skládá se z konečné stavové řídící jednotky, čtecí/zapisovací hlavy, vstupní pásky, která je z jedné strany neohraničená a množiny pravidel – přechodové funkce, která je součástí řídící jednotky.



Obrázek 3.1: Chomského hierarchie formálních gramatik

3.2 Bezkontextová gramatika

Bezkontextové gramatiky jsou dnes velice častým nástrojem využívaným pro popis syntaxe programovacích jazyků. Tyto gramatiky (resp. ekvivalentní zásobníkové automaty, viz níže) jsou základem syntaktických analyzátorů mnoha překladačů programovacích jazyků. Bezkontextové gramatiky jsou také využívány při formální specifikaci a verifikaci počítačových systémů.

Definice 3.2.1. *Gramatika* G je čtveřice $G = (N, T, P, S)$, kde

N je abeceda nonterminálních symbolů,
T je abeceda terminálních symbolů, přičemž $N \cap T = \varnothing$
P je konečná množina *pravidel* tvaru $A \to x \in P$, popř. relační zápis $(A, x) \in P$, kde $A \in N$, $x \in (N \cup T)^*$. Matematicky zapsáno je P konečná relace $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$
$S \in N$ je počáteční nonterminál

3.3 Stavová gramatika

Podklady pro tuto sekci jsem čerpal z [2]. Stavová gramatika je rozšířením bezkonextové gramatiky o množinu stavů. V každém kroku derivace se derivovaný řetězec nachází v určitém stavu. Každé z pravidel obsahuje stav a toto pravidlo můžeme aplikovat pouze tehdy, když se řetězec nachází ve stavu požadovaném tímto pravidlem. Důsledkem je, že ne vždy jsou při derivaci řetězce přepisovány nejlevější nonterminály jako u bezkontextové gramatiky, ale nonterminály hlouběji v řetězci, protože pro žádný z předchozích nonterminálů v řetězci neexistuje za daného stavu vyhovující pravidlo.

Definice 3.3.1. *Stavová Gramatika* G je pětice $G = (V, W, T, P, S)$, kde

V je totální abeceda,
W je konečná množina stavů,
T je abeceda terminálních symbolů, přičemž $T \subseteq V$
P je konečná množina *pravidel* tvaru $(q, A) \to (p, v) \in P$, popř. relační zápis $(q, A, p, v) \in P$. Matematicky zapsáno je P konečná relace $P \subseteq (W \times (V - T)) \times (W \times V^+)$,
S je počáteční nonterminál, přičemž platí, že $S \in (V - T)$

Pro každý řetězec $z \in V^*$, množina ${}_G states(z) = \{q|(q,B) \rightarrow (p,v) \in P\}$, kde $B \in (V-T) \cap alph(z)$, značí množinu nonterminálů vyskytujících se v tomto řetězci. $v \in V^+$ je neprázdný řetězec nad abecedou V a $q,p \in W$ jsou stavy. Pokud máme pravidlo $(q,A) \rightarrow (p,v)$ a dva řetězce $x,y \in V^*$, ${}_G states(x) \cap \{q\} = \varnothing$, potom G udělá derivační krok z (q,xAy) do (p,xvy), symbolicky zapíšeme jako $(q,xAy) \Rightarrow (p,xvy)[(q,A) \rightarrow (p,v)]$. Pokud navíc máme kladné celé číslo n, pro které platí, že $occur(xA, V-T) \leq n$ neboli počet nonterminálů v řetězci xA je menší nebo roven n, říkáme, že derivační krok $(q,xAy) \Rightarrow (p,xvy)[(q,A) \rightarrow (p,v)]$ je n-omezený. Symbolicky píšeme jako $(q,xAy)\ {}_n\Rightarrow (p,xvy)[(q,A) \rightarrow (p,v)]$. Pokud může být použito pouze jedno pravidlo a nehrozí tedy záměna, můžeme zjednodušeně psát $(q,xAy) \Rightarrow (p,xvy)$ resp. $(q,xAy)\ {}_n\Rightarrow (p,xvy)$. Standardním způsobem rozšíříme derivační krok $\Rightarrow$ na posloupnost derivačních kroků $\Rightarrow^m$, kde $m \geq 0$. Potom na základě $\Rightarrow^m$ definujeme $\Rightarrow^+$ a $\Rightarrow^*$. Nechť n je kladné celé číslo a $v,w \in (W \times V^+)$. Abychom vyjádřili, že každý derivační krok v posloupnosti derivací $v \Rightarrow^m w$, $v \Rightarrow^+ w$ a $v \Rightarrow^* w$ je n-omezený, používáme zápis $v\ {}_n\Rightarrow^m w$, $v\ {}_n\Rightarrow^+ w$ a $v\ {}_n\Rightarrow^* w$. Pomocí zápisu $strings(v\ {}_n\Rightarrow^* w)$ označujeme množinu všech řetězců vyskytujících se v derivaci $v\ {}_n\Rightarrow^* w$. Jazyk definovaný gramatikou $G, L(G)$ je definovaný jako $L(G) = \{w \in T^*|(q,S) \Rightarrow^* (p,w), p,q \in W\}$. Mimo jiné, pro každé $n \geq 1$, definujeme jazyk $L(G,n) = \{w \in T^*|(q,S)\ {}_n\Rightarrow^* (p,w), p,q \in W\}$. Derivace ve tvaru $(q,S)\ {}_n\Rightarrow^* (p,w)$, kde $p,q \in W$ a $w \in T^*$, představuje úspěšné n-omezené vygenerování řetězce w z gramatiky G. Jazyky definované stavovými gramatikami leží mezi bezkontextovými a kontextovými jazyky a budeme je nazývat jako stavové jazyky.

Kapitola 4

Automaty

Automaty obecně jsou tvořeny konečně stavovou řídící jednotkou a čtecí hlavou. Čtecí hlava načítá ze vstupní pásky se vstupním řetězcem jednotlivé vstupní symboly. Dále má tedy každý automat jednoznačně danou abecedu vstupních symbolů a konečnou množinu pravidel (přechodovou funkci). Automat vlastně pracuje jako algoritmus, který je pro libovolný řetězec nad vstupní abecedou schopen rozhodnout, jestli řetězec patří do jazyka přijímaného tímto automatem, nebo ne. Podklady pro tuto kapitolu jsem čerpal z [4, 3, 6], kde může čtenář v případě zájmu najít bližší popis těchto automatů.

4.1 Konečný automat

Konečné automaty jsou v oblasti formálních jazyků a překladačů velice důležité, neboť jsou základem takzvaných konečných převodníků. Pomocí konečných automatů (resp. konečných převodníků) se realizují v překladačích lexikální analyzátory, které po znacích načítají vstupní soubor a načítané znaky převádí na jednotlivé lexikální symboly (terminály).

Neformálně, konečný automat se skládá z konečné množiny stavů, z nichž jeden stav je počáteční a minimálně jeden koncový. Z abecedy vstupních symbolů a konečné množiny pravidel (přechodové funkce). Konečný automat pracuje tak, že v každém kroku načte ze vstupní pásky symbol a popřípadě přejde do dalšího stavu (viz tvar pravidel v definici). Automat obecně může vykonat přechod i bez toho, že by přečetl vstupní symbol (čtecí hlava se neposune). Automat začíná v počátečním stavu a čte první symbol ze vstupní pásky. Pokud načte všechny symboly na vstupní pásce a skončí v jednom z koncových stavů, tak řetězec byl úspěšně přijat.

Definice 4.1.1. *Konečný automat* M je pětice $M = (Q, \Sigma, R, s, F)$, kde

Q je konečná množina stavů
Σ je vstupní abeceda
R je konečná množina pravidel tvaru $pa \to q$, kde $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$. Matematicky je R relace $(Q \times (\Sigma \cup \{\varepsilon\})) \times Q$.
$s \in Q$ je počáteční stav
$F \subseteq Q$ je množina koncových stavů

4.2 Zásobníkový automat

Zásobníkové automaty jsou základním modelem pro syntaktickou analýzu bezkontextových jazyků. Oproti konečným automatům jsou rozšířeny o teoreticky nekonečnou paměť – zásobník, jehož vrchol ovlivňuje každý přechod automatu (viz tvar pravidel níže).

Zásobníkový automat v každém kroce může provést buď expanzi, nebo vyjmutí symbolu na vrcholu zásobníku. Pokud se na vrcholu nachází nevstupní symbol (nonterminál) automat provede jeho expanzi, tzn. že ho přepíše na řetězec vstupních i nevstupních symbolů (popř. i přečte další symbol ze vstupního řetězce) a přejde do dalšího stavu. Pokud se na vrcholu nachází vstupní symbol (terminál), a tento symbol je shodný s aktuálním symbolem na vstupní pásce, provede se operace vyjmutí a posune se čtecí hlava. Obecně může zásobníkový automat vykonat krok i bez přečtení vstupního symbolu (čtecí hlava se neposune) a nemusí ani číst nebo zapisovat na zásobník.

Vstupní řetězec patří do jazyka přijímaného zásobníkovým automatem, pokud automat provede sekvenci přechodů, během které načte celý vstupní řetězec, vyprázdní zásobník a přejde do koncového stavu. Pro všechny typy zásobníkových automatů platí, že pokud automat nemá specifikovány žádné koncové stavy (je tedy definován jako šestice), potom vstupní řetězec patří do jazyka přijímaného tímto automatem, pokud ho automat celý načte a současně vyprázdní zásobník. Potom říkáme, že automat přijímá prázdným zásobníkem.

Definice 4.2.1. *Zásobníkový automat* je sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

Q je konečná množina stavů
$\Sigma \subset \Gamma$ je vstupní abeceda
Γ je zásobníková abeceda
$\# \in (\Gamma - \Sigma)$ je speciální symbol značící dno zásobníku
R je konečná množina pravidel tvaru $Apa \rightarrow wq$, kde $A \in (\Gamma - (\Sigma \cup \{\#\}))$, $p, q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $w \in (\Gamma - \{\#\})^*$. Matematicky je R relace $((\Gamma - (\Sigma \cup \{\#\})) \times Q \times (\Sigma \cup \{\varepsilon\})) \times (Q \times (\Gamma - \{\#\})^*)$.
$s \in Q$ je počáteční stav
$S \in \Gamma$ je počáteční zásobníkový symbol
$F \subseteq Q$ je množina koncových stavů

U automatů které obsahují zásobník, bude zásobník reprezentován řetězcem, jehož nejlevější symbol odpovídá vrcholu zásobníku.

4.3 Hluboký zásobníkový automat

Podklady pro tuto sekci jsem čerpal [2], kde je tento typ automatů je zaveden a blíže popsán. Tento typ automatů je rozšířenou verzí zásobníkových automatů. Stejně jako zásobníkové automaty provádí v každém kroce vyjmutí nebo expanzi, ale s tím rozdílem, že hluboký zásobníkový automat je schopen provádět expanzi nonterminálních symbolů ne pouze na vrcholu, ale i hlouběji na zásobníku (hloubka je omezena typem automatu). Automat může provádět expanze v hloubce m tak, že nahradí m-tý nonterminální symbol na zásobníku řětězcem, kde $m \geq 1, m \in I$. Jinak pracuje stějně jako zásobníkový automat. Díky tomuto rozšíření se zvětší síla automatu, neboť je schopen rozpoznat i stavové jazyky (generované stavovými gramatikami popsanými v sekci 3.3).

Definice 4.3.1. *Hluboký zásobníkový automat* je sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

Q je konečná množina stavů
$\Sigma \subset \Gamma$ je vstupní abeceda
Γ je zásobníková abeceda
$\# \in (\Gamma - \Sigma)$ je speciální symbol značící dno zásobníku
R je konečná množina pravidel tvaru $mqA \to pv$, kde $A \in (\Gamma - (\Sigma \cup \{\#\}))$, $p, q \in Q$, $v \in (\Gamma - \{\#\})^+$, $m \geq 1$, $m \in I$ je hlubka ve které se nachází nonterminál, který chceme expandovat. Matematicky je R relace $(I \times Q \times (\Gamma - (\Sigma \cup \{\#\}))) \times (Q \times (\Gamma - \{\#\})^+)$
$s \in Q$ je počáteční stav
$S \in \Gamma$ je počáteční zásobníkový symbol
$F \subseteq Q$ je množina koncových stavů

Konfigurací hlubokého zásobníkového automatu M je trojice qxw, kde $q \in Q$, $x \in \Sigma^*$, $w \in (\Gamma - \{\#\})^*$. Nechť χ označuje množinu všech konfigurací M. Nechť $x, y \in \chi$ jsou dvě konfigurace. M provede přechod z x do y, zapisujeme jako $x \vdash y$. M provede operaci vyjmutí a přejde z konfigurace x do y, symbolicky zapsáno jako $x \ {}_p\vdash \ y$, pokud $x = (q, au, az)$ a $y = (q, u, z)$, kde $q \in Q$, $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$. M provede expanzi nontermínálu na zásobníku z konfigurace x do y, symbolicky zapsáno jako $x \ {}_e\vdash \ y$, pokud $x = (q, w, uAz)$ a $y = (p, w, uvz)$ a existuje pravidlo $mqA \to pv \in R$, kde $p, q \in Q$, $w \in \Sigma^*$, $A \in (\Gamma - (\Sigma \cup \{\#\}))$, $u, v, z \in \Gamma^*$, $occur(u, \Gamma - \Sigma) = m - 1$, neboli počet nonterminálů v řetězci u je roven $m - 1$ a A je tedy m-tým nonterminálem na zásobníku, takže pravidlo může být použito (nonterminály jsou na zásobníku číslovány od vrcholu zásobníku od jedničky). Pro vyjádření, že M provede expanzi nonterminálu na zásobníku z konfigurace x do y za použití pravidla $mqA \to pv$ používáme zápis $x \ {}_e\vdash \ y[mqA \to pv]$. Říkáme, že pravidlo $mqA \to pv$ je hloubky m. Zápis $x \ {}_e\vdash \ y[mqA \to pv]$ tedy označuje expanzi v hloubce m. Nechť $n \in I$ je co nejmenší kladné celé číslo takové, že všechna pravidla z R mají hloubku menší nebo rovnu n, pak říkáme, že automat M je hloubky n, což zapisujeme jako ${}_nM$. Standardním způsobem rozšíříme zápis pro výpočetní krok ${}_p\vdash$, ${}_e\vdash$ resp $\vdash$ na sekvenci $m, m \geq 0$, kroků ${}_p\vdash^m$, ${}_e\vdash^m$ a $\vdash^m$ a na jejich základě definujeme ${}_p\vdash^+$, ${}_p\vdash^*$, ${}_e\vdash^+$, ${}_e\vdash^*$, $\vdash^+$ a $\vdash^*$. Mějme ${}_nM, n \in I$, pak jazyk přijímaný tímto automatem definujeme jako $L({}_nM) = \{w \in \Sigma^* | (s, w, S\#) \vdash^* (f, \varepsilon, \#), f \in F\}$. Jazyk, který ${}_nM$ přijímá prázdným zásobníkem definujeme jako $E({}_nM) = \{w \in \Sigma^* | (s, w, S\#) \vdash^* (q, \varepsilon, \#)$ kde $q \in Q\}$.

4.3.1 Příklad

Mějme hluboký zásobníkový automat ${}_2M = (\{s, p, q, f\}, \{a, b, c\}, \{A, S, \#\}, R, s, S, \{f\})$, množina R obsahuje pravidla $1sS \to qAA, 1qA \to paAb, 1qA \to fab, 2pa \to qAc, 1fA \to fc$. Tento automat přijímá jazyk $L({}_2M) = \{a^n b^n c^n | n \geq 1\}$. Se vstupním řetězcem *aabbcc* provede ${}_2M$ následující přechody:

$(s, aabbcc, S\#) \ {}_e\vdash \ (q, aabbcc, AA\#)$ $[1sS \to qAA]$
$\ {}_e\vdash \ (p, aabbcc, aAbA\#)$ $[1qA \to paAb]$
$\ {}_p\vdash \ (p, abbcc, AbA\#)$
$\ {}_e\vdash \ (q, abbcc, AbAc\#)$ $[2pA \to qAc]$
$\ {}_e\vdash \ (f, abbcc, abbAc\#)$ $[1qA \to fab]$
$\ {}_p\vdash \ (f, bbcc, bbAc\#)$
$\ {}_p\vdash \ (f, bcc, bAc\#)$
$\ {}_p\vdash \ (f, cc, Ac\#)$

$$
\begin{aligned}
&{}_{e}\vdash (f, cc, cc\#)\ [1fA \to fc] \\
&{}_{p}\vdash (f, c, c\#) \\
&{}_{p}\vdash (f, \varepsilon, \#)
\end{aligned}
$$

Vidíme, že automat potřeboval 11 kroků, aby přijal vstupní řetězec *aabbcc*.

Kapitola 5

Částečně paralelní hluboké zásobníkové automaty

Částečně paralelní hluboký zásobníkový automat je rozšířením sekvenčního hlubokého zásobníkového automatu popsaného v sekci 4.3. Liší se v tom, že může v jednom kroce provést expanzi několika nonterminálů na zásobníku současně (paralelně) (viz tvar pravidel níže). Automat provádí expanze v hloubce m tak, že expanduje všechny nonterminální symboly na zásobníku od m-tého, až po první.

5.1 Definice

Definice 5.1.1. *Částečně paralelní hluboký zásobníkový automat* je sedmice $M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

Q je konečná množina stavů
$\Sigma \subset \Gamma$ je vstupní abeceda
Γ je zásobníková abeceda
$\# \in (\Gamma - \Sigma)$ je speciální symbol značící dno zásobníku
R je konečná množina pravidel tvaru $mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m)$, kde $p, q \in Q$, $A_1, A_2, \ldots, A_m \in (\Gamma - (\Sigma \cup \{\#\}))$, $v_1, v_2, \ldots, v_m \in (\Gamma - \{\#\})^+, m \geq 1, m \in I$. Matematicky je R relace $(I \times Q \times ((\Gamma - (\Sigma \cup \{\#\}))^m)) \times (Q \times ((\Gamma - \{\#\})^+)^m)$
$s \in Q$ je počáteční stav
$S \in \Gamma$ je počáteční zásobníkový symbol
$F \subseteq Q$ je množina koncových stavů

5.2 Konfigurace

Konfiguraci částečně paralelního hlubokého zásobníkového automatu budeme zapisovat jako

$$(q, w, z) \in Q \times \Sigma^* \times (\Gamma - \{\#\})^*\{\#\}$$

q je aktuální stav řídící jednotky.
w je nepřečtená část vstupního řetězce. Pokud $w = \varepsilon$, pak byl celý vstupní řetězec přečten.
z je aktuální obsah zásobníku. Pokud $z = \varepsilon$, pak je zásobník prázdný.

5.3 Pravidla

Pravidla budeme zapisovat ve tvaru

$$mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m) \in R$$

kde $p, q \in Q, A_1, A_2, \ldots, A_m \in (\Gamma - (\Sigma \cup \{\#\})), v_1, v_2, \ldots, v_m \in (\Gamma - \{\#\})^+, m \geq 1, m \in I$. Stav q odpovídá aktuálnímu stavu automatu. Nonterminální symboly na levé straně odpovídají nonterminálům na zásobníku v přirozeném pořadí. Tedy první nonterminál uvedený na levé straně pravidla, označený jako A_1, odpovídá prvnímu nonterminálnímu symbolu na zásobníku (první nonterminál na zásobníku musí být A_1). A_2 odpovídá druhému nonterminálu, až A_m odpovídá m-tému nonterminálu na zásobníku. Pokud je automat ve stavu q a nonterminály v pravidle se vyskytují v odpovídajícím pořadí na zásobníku, pravidlo můžeme použít. Číslo m tedy označuje počet nonterminálů na levé straně pravidla. Říkáme, že pravidlo je hloubky m.

Nechť $n \in I$ je co nejmenší kladné celé číslo takové, že hloubka všech pravidel z R je menší nebo rovna n. Pak říkáme, že $M^{semi\text{-}par}$ je hloubky n, což zapisujeme jako ${}_nM^{semi\text{-}par}$.

5.4 Přechody

Nechť $M^{semi\text{-}par}$ je částečně paralelní hlubký zásobníkový automat. Nechť χ označuje množinu všech konfigrací $M^{semi\text{-}par}$. Nechť $x, y \in \chi$ jsou dvě konfigurace. $M^{semi\text{-}par}$ přejde z konfigurace x do y, zapsáno jako $x \vdash y$. $M^{semi\text{-}par}$ provede operaci vyjmutí a přejde z konfigurace x do y, zapsáno jako $x \ {}_p\vdash y$, pokud $x = (q, au, az)$, $y = (q, u, z)$, kde $a \in \Sigma$, $u \in \Sigma^*$, $z \in (\Gamma - \{\#\})^*$. $M^{semi\text{-}par}$ expanduje prvních m nonterminálů na zásobníku a přejde z konfigurace x do y, zapsáno jako $x \ {}_e\vdash y$, pokud $x = (q, w, uv)$, $y = (p, w, zv)$, $mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m) \in R$, kde $p, q \in Q$, $w \in \Sigma^*$, $u \in (\Sigma^*(\Gamma - (\Sigma \cup \{\#\})))^m$, $v \in \Gamma^*$, $z \in ((\Gamma - \{\#\})^+)^m$, $A_1, A_2, \ldots, A_m \in (\Gamma - (\Sigma \cup \{\#\}))$, $v_1, v_2, \ldots, v_m \in (\Gamma - \{\#\})^+$, $m \geq 1$, $m \in I$. A platí, že $occur(u, \{A_1\}) = 1$, $occur(u, \{A_2\}) = 1, \ldots, occur(u, \{A_m\}) = 1$, tedy na zásobníku se vyskytují nonterminály z levé strany pravidla a jsou v požadovaném pořadí.

Pro vyjádření, že expanze a přechod z konfigurace x do y byla provedena pomocí pravidla $mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m) \in R$ používáme zápis $x \ {}_e\vdash y[mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m)]$. Říkáme, že pravidlo $mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m)$ je hloubky m, potom $x \ {}_e\vdash y[mq(A_1, A_2, \ldots, A_m) \to p(v_1, v_2, \ldots, v_m)]$ je expanze hloubky m.

Standardním způsobem rozšíříme zápis pro přechod ${}_p\vdash, {}_e\vdash$ resp $\vdash$ na sekvenci $m, m \geq 0$ přechodů ${}_p\vdash^m, {}_e\vdash^m$ a $\vdash^m$ a na jejich základě definujeme ${}_p\vdash^+, {}_p\vdash^*, {}_e\vdash^+, {}_e\vdash^*, \vdash^+$ a $\vdash^*$.

5.5 Přijímané jazyky

Jazyk přijímaný automatem ${}_nM^{semi\text{-}par}$ definujeme jako:

$$L({}_nM^{semi\text{-}par}) = \{w \in \Sigma^* | (s, w, S\#) \vdash^* (f, \varepsilon, \#) \text{ je konfigurace } {}_nM^{semi\text{-}par} \text{ a } f \in F\}$$

.

Jazyk, který ${}_nM^{semi\text{-}par}$ přijímá prázdným zásobníkem definujeme jako:

$$E({}_nM^{semi\text{-}par}) = \{w \in \Sigma^* | (s, w, S\#) \vdash^* (f, \varepsilon, \#) \text{ je konfigurace } {}_nM^{semi\text{-}par} \text{ a } f \in Q\}$$

5.6 Síla

Pro každé $k \geq 1$, ${}_{semi\text{-}par\ deep}\mathbf{PD}_k$ označuje třídu jazyků definovanou částečně paralelními hlubokými zásobníkovými automaty hloubky i, kde $0 \leq i \leq k$. Podobně ${}^{empty}_{semi\text{-}par\ deep}\mathbf{PD}_k$ popisuje třídu jazyků, kterou částečně paralelní hluboké zásobníkové automaty hloubky i, kde $0 \leq i \leq k$, přijímají prázdným zásobníkem. **CS** a **CF** označují kontextové, repsektive bezkontextové jazyky.

Lemma 1. Pro každou stavovou gramatiku G a pro každé $n \geq 1$, existuje částečně paralelní hluboký zásobníkový automat hloubky n, ${}_nM^{semi\text{-}par}$ takový, že je schopen rozpoznat řetězce jazyka generovaného gramatikou G, $L(G,n) = L({}_nM^{semi\text{-}par})$.

Základní myšlenka. Automat ${}_nM^{semi\text{-}par}$ simuluje n-omezené derivace gramatiky G tak, že si vždy do svého stavu zaznamená prvních n nonterminálů vyskytujících se v aktuální větné formě (pokud se v aktuálním řetězci vyskytuje méně než n nonterminálů doplní na konec řetězce znaky # tak, aby měl řetězec délku n). ${}_nM^{semi\text{-}par}$ simuluje derivační krok ve svém zásobníku a současně si nonteminály z nově vygenerovaného řetězce zaznamená výše popsaným způsobem do svého stavu. Jakmile gramatika G dokončí generování řetězce (to znamená, že řetězec se skládá už pouze z terminálních symbolů), tak ${}_nM^{semi\text{-}par}$ dokončí čtení řetězce, vyprázdní zásobník a přejde do koncového stavu.

Lemma 2. Pro každé $n \geq 1$ a pro každý částečně paralelní hluboký zásobníkový automat ${}_nM^{semi\text{-}par}$, existuje stavová gramatika G taková, že generuje jazyk, který tento automat přijímá $L({}_nM^{semi\text{-}par}) = L(G,n)$.

Základní myšlenka. Gramatika G simuluje aplikaci pravidla $mq(A_1, A_2, \ldots, A_m) \rightarrow p(v_1, v_2, \ldots, v_m) \in R$ tak, že z leva do prava prochází větnou formu. Počítá výskyt nonterminálů, dokud nenarazí na m-tý nonterminál (zleva). Tento nonterminál porovná s nonterminálem A_m z pravidla a pokud jsou shodné, nahradí ho ve větné formě řetězcem v_m. Potom se ve větné formě posune o jeden nonterminál doleva a ten porovná s A_{m-1} z pravidla a v případě rovnosti jej opět nahradí odpovídajícím řetězcem v_{m-1}. Stejným způsobem postupuje až po první nonterminál ve větné formě a po té simuluje přechod do stavu p. V průběhu simulace přechodů automatu gramatikou je vždy poslední symbol větné formy z množiny W (uvažujme, že je zavedená bijekce h z T do W). G dokončí simulaci přijetí řetězce automatem ${}_nM^{semi\text{-}par}$ tak, že použije pravidlo, které nahradí poslední symbol ve větné formě (symbol z množiny W) terminálním symbolem ($h(W) \rightarrow T$) a tím dokončí vygenerování řetězce.

Theorém 1. Pro každé $n \geq 1$ a pro každý stavový jazyk L takový, že $L = L(G,n)$ je generován stavovou gramatikou G, tehdy a jen tehdy, když existuje částečně paralelní hluboký zásobníkový automat $L = L({}_nM^{semi\text{-}par})$, který přijímá tento jazyk.

Theorém 2. Pro každé $n \geq 1$ a pro každý stavový jazyk L takový, že $L = L(G,n)$ je generován stavovou gramatikou G, tehdy a jen tehdy, když existuje částečně paralelní hluboký zásobníkový automat $L = E({}_nM^{semi\text{-}par})$, který přijímá tento jazyk prázdným zásobníkem.

Důkaz Theorém 1 a 2 vyplívá z Lemmat 1 a 2.

Theorém 3.

$$ {}^{empty}_{semi\text{-}par\ deep}\mathbf{PD}_n = {}_{semi\text{-}par\ deep}\mathbf{PD}_n \subset {}^{empty}_{semi\text{-}par\ deep}\mathbf{PD}_{n+1} = {}_{semi\text{-}par\ deep}\mathbf{PD}_{n+1} $$

, pro každé $n \geq 1$

Důkaz Toto tvrzení vychází z teorému 1 a 2 a ze znalosti, že m-omezené stavové gramatiky generují třídu jazyků, která je podtřídou jazyků generovaných gramatikami, které jsou $m+1$-omezené, kde $m \geq 1$.

Theorém 4.

$$_{semi\text{-}par\ deep}\mathbf{PD}_1 = {}^{empty}_{semi\text{-}par\ deep}\mathbf{PD}_1 = \mathbf{CF}$$

Důkaz Theorém 4 vyplívá z Lemmat 1 a 2.

Theorém 5. Pro každé $n \geq 1$ platí,

$$^{empty}_{semi\text{-}par\ deep}\mathbf{PD}_n = {}_{semi\text{-}par\ deep}\mathbf{PD}_n \subset \mathbf{CS}$$

Důkaz Víme, že n-omezené stavové gramatiky generují třídu jazyků, která je podtřídou **CS**, potom Theorém 5 vychází z Lemmat 1 a 2 a theorému 1 a 2.

Theorém 6.

$$_{semi\text{-}par\ deep}\mathbf{PD}_n = {}_{deep}\mathbf{PD}_n$$

Důkaz 6 vychází z theorému 1 a 2 a z theorémů 1 a 2 v článku [2].

5.6.1 Shrnutí

Díky tomu, že částečně paralelní hluboké zásobníkové automaty mohou provádět expanze hlouběji v zásobníku, jsou schopny rozpoznávat jazyky generované stavovými gramatikami (popsány v sekci 3.3). Stavové gramatiky generují nekonečnou hierarchii tříd stavových jazyků (mezi bezkontextovými a kontextovými jazyky). Automaty, které jsou schopny tyto jazyky rozpoznat mají tedy větší sílu, než klasické zásobníkové automaty. Důsledkem je zvýšení síly syntaktické analýzy, která na těchto modelech staví, neboť máme možnost více monitorovat kontext vstupního řetězce.

Výše uvedené poznatky ale platí jen pro automaty, které jsou hloubky 2 a větší. Automaty hloubky 1 se stávají klasickými zásobníkovými automaty a jsou tedy schopny přijímat pouze bezkontextové jazyky.

Jak vyplívá z výše uvedených vět je síla částečně paralelních a sekvenčních hlubokých zásobníkových automatů stejná.

5.7 Rychlost

V příkladu 4.3.1 je uveden sekvenční hluboký zásobníkový automat, který přijme vstupní řetězec *aabbcc* po 11 krocích. V příkladu 5.8.1 je ekvivalentní částečně paralelní automat, který přijme stejný řetězec po 9 krocích. Zde je tedy důkaz toho, že i když paralelismus nezvětší sílu těchto automatů, zvýší se rychlost automatu neboli sníží se počet kroků, které musí automat vykonat, aby přijmul řetězec. Urychlení je dosaženo díky tomu, že v jednom kroce automat expanduje současně několik nonterminálních symbolů na zásobníku.

Urychlení oproti sekvenční verzi je tedy závislé na konkrétním automatu (resp. jeho pravidlech) a na délce vstupního řetězce. Čím delší je vstupní řetězec, tím větší je urychlení.

Také je nutné si uvědomit, že urychlení se týká pouze expanze. Operace vyjmutí probíhá stejně jako u sekvenčního automatu po jednom symbolu.

5.8 Příklady

V této sekci uvedu několik případů stavových jazyků a k nim odpovídajících automatů, které jsou schopny tyto jazyky rozpoznat.

5.8.1 Příklad 1

V tomto příkladě s částečně paralelním automatem, který přijímá stejný jazyk jako sekvenční automat definovaný v příkladě 4.3.1, zkusíme přijmout stejný řetězec.

Mějme částečně paralelní hluboký zásobníkový automat
${}_2M^{semi\text{-}par} = (\{s, p, f\}, \{a, b, c\}, \{A, S, a, b, c, \#\}, R, s, S, \{f\})$, množina R obsahuje pravidla $1s(S) \to p(AA), 2p(A, A) \to p(aA, bAc), 2p(A, A) \to f(a, bc)$ Se vstupním řetězcem *aabbcc* provede ${}_2M^{semi\text{-}par}$ následující přechody:

$$
\begin{aligned}
(s, aabbcc, S\#) &\ {}_e\vdash (p, aabbcc, AA\#)\ [1s(S) \to p(AA)\\
&\ {}_e\vdash (p, aabbcc, aAbAc\#)\ [2p(A, A) \to p(aA, bAc)]\\
&\ {}_p\vdash (p, abbcc, AbAc\#)\\
&\ {}_e\vdash (f, abbcc, abbcc\#)\ [2p(A, A) \to f(a, bc)]\\
&\ {}_p\vdash (f, bbcc, bbcc\#)\\
&\ {}_p\vdash (f, bcc, bcc\#)\\
&\ {}_p\vdash (f, cc, cc\#)\\
&\ {}_p\vdash (f, c, c\#)\\
&\ {}_p\vdash (f, \varepsilon, \#)
\end{aligned}
$$

5.8.2 Příklad 2

Nyní uvažujme automat ${}_3M^{semi\text{-}par}$, který přijímá jazyk $L({}_3M^{semi\text{-}par}) = \{a^m b^n a^m b^n | m, n \geq 1\}$. ${}_3M^{semi\text{-}par} = (\{s, p, f\}, \{a, b\}, \{A, B, S, a, b, \#\}, R, s, S, \{f\})$, množina R obsahuje pravidla $1s(S) \to p(ABAB), 3p(A, B, A) \to p(aA, B, aA), 3p(A, B, A) \to p(a, B, a), 2p(B, B) \to p(bB, bB), 2p(B, B) \to f(b, b)$. Se vstupním řetězcem *aaabaaab* provede ${}_3M^{semi\text{-}par}$ následující přechody:

$$
\begin{aligned}
(s, aaabaaab, S\#) &\ {}_e\vdash (p, aaabaaab, ABAB\#)\ [1s(S) \to p(ABAB)]\\
&\ {}_e\vdash (p, aaabaaab, aABaAB\#)\ [3p(A, B, A) \to p(aA, B, aA)]\\
&\ {}_p\vdash (p, aabaaab, ABaAB\#)\\
&\ {}_e\vdash (p, aabaaab, aABaaAB\#)\ [3p(A, B, A) \to p(aA, B, aA)]\\
&\ {}_p\vdash (p, abaaab, ABaaAB\#)\\
&\ {}_e\vdash (p, abaaab, aBaaaB\#)\ [3p(A, B, A) \to p(a, B, a)]\\
&\ {}_p\vdash (p, baaab, BaaaB\#)\\
&\ {}_e\vdash (f, baaab, baaab\#)\ [2p(B, B) \to f(b, b)]\\
&\ {}_p\vdash (f, aaab, aaab\#)\\
&\ {}_p\vdash (f, aab, aab\#)\\
&\ {}_p\vdash (f, ab, ab\#)\\
&\ {}_p\vdash (f, b, b\#)\\
&\ {}_p\vdash (f, \varepsilon, \#)
\end{aligned}
$$

5.8.3 Příklad 3

Mějme automat ${}_2M^{semi\text{-}par}$, který přijímá jazyk $L({}_2M^{semi\text{-}par}) = \{ww|w \in \{0,1\}^+\}$. ${}_2M^{semi\text{-}par} = (\{s,p,f\},\{0,1\},\{S,A,0,1,\#\},R,s,S,\{f\})$, množina R obsahuje pravidla $1s(S) \rightarrow p(AA), 2p(A,A) \rightarrow p(0A,0A), 2p(A,A) \rightarrow p(1A,1A), 2p(A,A) \rightarrow p(0,0)$, $2p(A,A) \rightarrow p(1,1)$. Se vstupním řetězcem 00100010 provede ${}_2M^{semi\text{-}par}$ následující přechody:

$$
\begin{aligned}
(s, 00100010, S\#) &\ {}_e\vdash\ (p, 00100010, AA\#)\ [1s(S) \rightarrow p(AA)] \\
&\ {}_e\vdash\ (p, 00100010, 0A0A\#)\ [2p(A,A) \rightarrow p(0A,0A)] \\
&\ {}_p\vdash\ (p, 0100010, A0A\#) \\
&\ {}_e\vdash\ (p, 0100010, 0A00A\#)\ [2p(A,A) \rightarrow p(0A,0A)] \\
&\ {}_p\vdash\ (p, 100010, A00A\#) \\
&\ {}_e\vdash\ (p, 100010, 1A001A\#)\ [2p(A,A) \rightarrow p(1A,1A)] \\
&\ {}_p\vdash\ (p, 00010, A001A\#) \\
&\ {}_e\vdash\ (p, 00010, 00010\#)\ [2p(A,A) \rightarrow p(0,0)] \\
&\ {}_p\vdash\ (p, 0010, 0010\#) \\
&\ {}_p\vdash\ (f, 010, 010\#) \\
&\ {}_p\vdash\ (f, 10, 10\#) \\
&\ {}_p\vdash\ (f, 0, 0\#) \\
&\ {}_p\vdash\ (f, \varepsilon, \#)
\end{aligned}
$$

Kapitola 6

Aplikace

Podle zadání a upřesnění na konzultacích s vedoucím práce jsem automaty definované v části 5 aplikoval v syntaktické analýze na stavové jazyky (které jsou generovány stavovými gramatikami popsanými v části 3.3). Aplikace byla implementována v jazyce C++.

Vytvořil jsem konzolovou aplikaci, která je pro zadaný částečně paralelní hluboký zásobníkový automat schopna provést syntaktickou analýzu vstupního řetězce. Jinými slovy zjistí, jestli vstupní řetězec patří do jazyka přijímaného zadaným automatem.

Uživatel může při spuštění programu zadat jestli chce automat po načtení z XML souboru vytisknout na výstup, což může dobře posloužit pro kontrolu spravného zadání. Dále může nechat vytisknout, v případě úspěšné syntaktické analýzy, sekvenci použitých pravidel.

6.1 Analýza a návrh řešení

Jak bylo popsáno výše, výhoda částečně paralelních automatů oproti sekvenční verzi spočívá v tom, že mohou v jednom kroce přepsat více nonterminálních symbolů na zásobníku současně. V mé aplikaci budu tento paralelismus simulovat sekvenčně, kdy budu postupně přepisovat nonterminály od nejhlouběji zanořeného, který se má přepsat, až po první.

Vzhledem k tomu, že automaty, které moje aplikace musí umět simulovat pracují nedeterministicky (neboli více pravidel může mít stejnou levou stranu), bylo nutné navrhnout takový algoritmus, který bude schopen, v případě neúspěchu po aplikaci jednoho pravidla, načíst zpět konfiguraci před použitím tohoto pravidla a aplikovat další pravidlo v pořadí. Tento popis nás přímo navádí na známý algoritmus prohledávání stavového prostoru zvaný backtracking (česky metoda zpětného vyhledávání). Bylo tedy nutné vytvořit zásobník historie, na který se budou ukládat konfigurace, ve kterých je možné aplikovat více než jedno pravidlo. Každá konfigurace podle očekávání uchovává pozici ve vstupním řetězci, aktuální stav automatu a aktuální obsah zásobníku. Navíc uchovává i identifikátor pravidla, které se má v případě jejího načtení použít (díky tomu nebudeme muset po načtení konfigurace znovu prohledávat množinu pravidel). A nakonec sekvenci dosud použitých pravidel.

Po konzultaci s vedoucím práce jsem se rozhodl implementovat další algoritmus. Tento se od předchozího liší v tom, že výpočet provádí více paralelně běžících vláken. Tento přístup spočívá v tom, že pokaždé, kdy je možné použít v jedné konfiguraci n pravidel, kde $n > 1$, tak výpočet, kdy se použije první nalezené pravidlo pokračuje v aktuálním vlákně a pro každé další vyhovující pravidlo se spustí nové vlákno a aplikace tohoto pravidla a další výpočet už pokračuje v tomto novém vláknu. Jakmile některé vlákno úspěšně přijme

vstupní řetězec, ostatní vlákna se ukončí a výpočet končí. Výhoda tohoto přístupu je ve vyšší rychlosti výpočtu. Po implementování a testování jsem ale zjistil, že výpočet naopak trvá déle, než při použití backtrackingu. To může být způsobeno tím, že pro vstupní rětězce, obsahující stovky symbolů se souští stovky i tisíce nových vláken. Záleží samozřejmě na zadaném automatu, ale vzrůstá tím velice režie při vytváření, rušení a přepínání vláken. Pro některé automaty navíc vlákno po spuštění provede aplikaci jen několika málo pravidel a zjistí, že nemůže pokračovat a tedy se ukončí. Proto je otázka, jestli je ve výsledku vzhledem k pořebné režii na spouštění a ukončení vlákna vůbec nějaké urychlení.

Vzhledem k tomu, že výše uvedený algoritmus neměl žádný přínos, jsem se po konzultaci s vedoucím práce a s panem Dr. Peringerem rozhodl zvolit jiný přístup, kdy se spustí pouze dvě paralelně bežící vlákna. Každé z těchto vláken bude pracovat jako v případě backtrackingu, ale konfigurace se budou ukládat na sdílený zásobník. Pokud se vlákno (resp. automat) dostane do konfigurace, kdy nemůže pokračovat, načte ze sdíleného zásobníku poslední uloženou konfiguraci. Naopak pokud se dostane do konfigurace, kdy může použít n pravidel, kde $n > 1$, první vyhovující pravidlo si uloží a pro ostatní uloží na zásobník historie odpovídající konfigurace. Tímto se eliminuje režie nutná na správu velkého množství vláken. Na druhou stranu je ale nutné zajistit výlučný přístup ke sdíleným zdrojům, což je oproti předchozímu přístupu režie navíc.

Po skončení syntaktické analýzy program vypíše na výstup hlášení, jestli řetězec automatem byl nebo nebyl přijat.

6.2 Vstupní soubory

Soubor se vstupním řetězcem je textový soubor obsahující pouze vstupní řetězec. Pokud jsou vstupní symboly delší než jednoznakové, musí být odděleny bílými znaky, jinak automat nemusí fungovat správně.

Automat je zadán skrze XML soubor. Jak se automat zadává zde popsáno není. Podrobný návod naleznete v dokumentaci k aplikaci na přiloženém CD. Pro lepší pochopení jsou na CD i příklady několika XML souborů s automaty.

6.3 Datové typy

Pro reprezentaci symbolů slouží třída `Symbol`, která v proměnné typu `string` uchovává hodnotu symbolu a druhá proměnná výčtového typu `SymbolType` (resp. `int`) uchovává o jáký typ symbolu se jedná – terminál, nonterminál, nebo symbol značící dno zásobníku automatu.

Zásobník automatu představuje třída `DeepStack`. Tato třída obsahuje ukazatel na vrchol zásobníku a uchovává si symbol značící dno zásobníku (aby automat bezpečně poznal, kdy je na vrcholu zásobníku právě symbol značící dno, tedy kdy je zásobník prázdný). Jednotlivé položky na zásobníku obsahují proměnnou typu `Symbol` a ukazatel na další položku. Klíčová metoda v této třídě je `DeepStack::expand()`, která expanduje obsah zásobníku podle předaného pravidla.

K uchovávání pravidel slouží třída `Rule`. Tato třída má dvě členské proměnné typu `string`, které uchovávají výchozí a následující stav automatu pro toto pravidlo. Dále obsahuje proměnnou typu `unsigned int` uchovávající hloubku pravidla a pole prvků typu `unsigned int`, které obsahuje počty symbolů, na které se každý z nonterminálů na levé

straně expanduje. Symboly na levé a pravé straně pravidla jsou uloženy v polích obsahujících položky typu `Symbol`.

Zásobník pro metodu backtracking představuje třída `BacktrStack`. Pro uložení kofigurací ukládaných na tento zásobník slouží třída `Configuration`. `Configuration` uchovává v proměnné typu `string` stav automatu. Dvě proměnné typu `int`, které obsahují pozici čtecí hlavy ve vstupním souboru a identifikátor pravidla, které se má v této konfiguraci použít. Proměnnou typu `DeepStack`, která uchovává obsah zásobníku automatu a proměnnou se sekvencí dosud použitých pravidel, pro kterou jsem použil datový kontejner `vector<>` z knihovny STL, který bude uchovávat položky typu `int`, které reprezentují identifikátory pravidel.

Nyní se dostávám k hlavní třídě `Automaton`. Ta obsahuje proměnné typu `string` pro uložení počátečního a aktuálního stavu, zásobník automatu typu `DeepStack` a zásobník pro metodu backtracking typu `BacktrStack`. Proměnnou typu `int` udávající pozici čtecí hlavy. Maximální hloubku automatu typu `unsigned int`, sekvenci použitých pravidel stejně jako v `Configuration` v datové struktuře typu `vector<>`. Proměnná typu `Symbol` pro uložení počátečního zásobníkového symbolu. Dále je zde několik množin pro něž jsem použil další šablonu, opět z knihovny STL a sice `set<>`. Jsou to množiny stavů a koncových stavů, množina pravidel, vstupní abeceda a zásobníková abeceda. Automat obsahuje i instanci třídy `ReadingHead`. Jak název napovídá jedná se o čtecí hlavu automatu. Ta uchovává ukazatel na soubor se vstupním řetězcem.

6.4 Implementace

Po spuštění se pomocí funkce `getopt()` zpracují parametry příkazové řádky. Poté se zavolá funkce `loadAutomaton`, která z xml souboru načte automat. K načítání dat ze souboru ve formátu XML využívám knihovnu RapidXml zejména z důvodu její rychlosti a jednoduchosti použití. Následně se spustí parsování zavoláním metody `Automaton::parse()`. Podle zadaných parametrů při spustění se poté zavolá odpovídající metoda parsování.

Pokud chceme provést syntaktickou analýzu za použití metody backtarcking, zavolá se metoda `Automaton::parseUsingBacktracking()`. Algoritmus této metody je následující. Nejdříve se otestuje jaký typ symbolu je na vrcholu zásobníku. Pokud terminál, zavolá se metoda `ReadingHead::getNextSymbol()` (blíže popsána níže), která vrátí další vstupní symbol. Symbol na vrcholu zásobníku se porovná se symbolem na vstupu a pokud se rovnají, provede se operace `DeepStack::pop()`. Pokud jsou rozdílné, automat načte předchozí konfiguraci ze zásobníku historie. Je-li na vrcholu zásobníku nonterminál, postupně procházíme množinu pravidel a pokud pravidlo má stejný výchozí stav jako je aktuální stav automatu otestujeme pomocí metody `DeepStack::testRule()` jestli můžeme pravidlo aplikovat podle aktuálního obsahu zásobníku. Pokud ano uschováme si na něj ukazatel. Pokračujeme v testování dalších pravidel a pro každé které může být použito si na zásobník historie uložíme aktuální konfiguraci (ve které bude i id pravidla, které se má použít, viz výše). Po otestování všech pravidel zavoláme metodu `DeepStack::expand()` s parametrem typu `Rule*` ukazujícím na první nalezené pravidlo. Pokud nebylo nalezeno žádné vyhovující pravidlo, opět se načte předchozí konfigurace. Pokud se podaří vyprázdnit zásobník automatu, otestuje se, jestli už ve vstupním souboru není žádný symbol a jestli se automat nachází v koncovém stavu. Pokud jsou obě podmínky splněny, znamená to, že automat úspěšně přijal vstupní řetězec a výpočet končí. Pokud některá z podmínek neplatí, automat načte předchozí konfiguraci. Pokud se má načíst přechozí konfigurace, ale zásobník

historie je prázdný, znamená to, že vstupní řetězec nepatří do jazyka přijímaného zadaným automatem a výpočet končí.

Duhou možností je provést syntaktickou analýzu pomocí dvou paralelně běžících vlákem. V tomto případě se spustí metoda `Automaton::parseUsingThreads()`. Tato metoda vytvoří dvě vláka, z nichž každé má svou vlastní množinu proměnných, které se sice vyskytují ve třídě `Automaton`, ale protože se v průběhu výpočtu jejich obsah mění, každé vláno musí mít své vlastní proměnné. Každé vlákno má tedy svůj vlastní zásobník, proměnnou uchovávající aktuální stav, pozici ve vstupním souboru, a sekvenci dosud použitých pravidel. Druhou možností by bylo vytvořit kopii třídy Automaton a mít sdílený zásobník historie, ale v tomto případě by se zbytečně v paměti nacházely dvakrát stejná data (např. množina pravidel, symbolů, atd.). Algoritmus práce vláken je stejný jako u metody backtracking, ale vlákna pracují se sdíleným zásobníkem historie. Přístup k zásobníku historie je řízen pomocí sdíleného semaforu, který je globální proměnnou v rámci modulu. Pokud vlákno chce načíst konfiguraci, nejdříve zjistí, jestli už na semaforu nečeká i druhé vlákno, což by znamenalo, že vstupní řetězec nepatří do jazyka přijímaného zadaným automatem. V takovém případě se obě vlákna ukončí. V případě, že některé z vláken úspěšně přijme vstupní řetězec nastaví hodnotu sdílené globální proměnné `parsingResult`, kterou vlákna v každém cyklu výpočtu testují. Jakmile vlákno zjistí, že je tato proměnná nastavena, ukončí svoji činnost.

Hlavní metoda třídy `ReadingHead`, kterou je `ReadingHead::getNextSymbol()` čte vstupní soubor znak po znaku. Pokud narazí na bílý znak, považuje ho za oddělovač symbolů a aktuálně načtený řetezec porovnává se symboly ze vstupní abecedy automatu. Pokud vstupní abeceda obsahuje symbol (řetězec), se kterým se aktuálně načítaný symbol shoduje, tak metoda končí úspěšně. Pokud při čtení ze souboru na bílý znak nenarazí, tak po načtení jednoho znaku do symbolu tento symbol vždy porovná se symboly ze vstupní abecedy. V případě nalezení shody opět končí (pokud shodu nikdy nenajde, tak bude načítat a porovnánat dokud nenarazí na bílý znak, nebo konec souboru). Podle návratového kódu této metody poznáme, jestli byl úspěšně načten vstupní symbol nebo jestli narazil na neznámý symbol popř. konec souboru. Vidíme tedy, že čtecí hlava pracuje jako jednoduchý lexikální analyzátor (konečný automat resp. konečný převodník), který načítá znak po znaku vstupní soubor a na výstup dodává symboly ze vstupní abecedy zadaného automatu.

Kapitola 7

Závěr

V této práci byly představeny částečně paralelní hluboké zásobníkové automaty, které jsou rozšířením sekvenčních hlubokých zásobníkových automatů. Zjistili jsme, že oba tyto typy automatů přijímají stejnou třídu jazyků, které jsou definované stavovými gramatikami. Oproti sekvenčním jsou ale částečně paralelní automaty schopny přijmout vstupní řetězec rychleji (potřebují menší počet kroků).

V této práci byla rozebrána obecná verze těchto automatů, které pracují nedeterministicky. Z pohledu budoucího výzkumu nás ale bude zcela jistě zajímat deterministická verze, která je klíčová pro využití těchto automatů v praxi. Dále jsem v této práci uvažoval pouze tvar pravidel, kdy se nonterminály na zásobníku vždy přepíší na neprázdný řetězec. Další otázkou tedy je, jak se změní vlastnosti automatů, pokud bych uvažoval i možnost přepsat nonterminály na prázdný řetězec (ε), popř. jak se změní třída přijímaných jazyků.

7.1 Praktická část

V praktické části jsem podle zadání aplikoval částečně paralelní hluboké zásobníkové automaty v syntaktické analýze. Výsledkem je aplikace, která je pro libovolný nedeterministický částečně paralelní hluboký zásobníkový automat schopna provést syntaktickou analýzu vstupního řetězce. Implementoval jsem dva různé algoritmy simulující činnost automatu. Ukázalo se, že algoritmus provádějící výpočet pomocí dvou paralelně běžících vláken je skutečně v některých případech rychlejší, tudíž použití paralelismu zde má význam, i když paralelismus nebyl využit tak, jak bylo zamýšleno původně (viz sekce 6.1).

Aplikace může najít svoje využití například při výuce, jako podpora pro studenty. Jedním ze způsobů jak aplikaci přiblížit použití v praxi, by bylo její využití s programem, který by byl schopen pro zadanou stavovou gramatiku vytvořit odpovídající částečně paralelní hluboký zásobníkový automat a tento automat exportovat do XML souboru (který má samozřejmě stejnou strukturu, jako XML soubor pro můj program). Tento soubor s automatem by potom sloužil jako vstup mnou vytvořené aplikace.

Literatura

[1] Chomského hierarchie [online]. 22. července 2011. [cit. 2012-04-13].
URL http://cs.wikipedia.org/wiki/Chomského_hierarchie

[2] Meduna, A.: Deep Pushdown Automata. *Acta Informatica*, ročník 2006, č. 98, 2006: s. 114–124, ISSN 0001-5903.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8000

[3] Meduna, A.; Lukáš, R.: *Výstavba překladačů*. FIT VUT v Brně, studijní opora.

[4] Meduna, A.; Lukáš, R.: *Formální jazyky a překladače*. FIT VUT v Brně, 2006, studijní opora.

[5] Solár, P.: *Paralelní hluboké zásobníkové automaty*. Bakalářská práce, FIT VUT v Brně, Brno, 2007.

[6] Češka, M.; Vojnar, T.; Smrčka, A.: *Teoretická informatika*. FIT VUT v Brně, 2011, studijní opora.

Příloha A

Obsah CD

Přiložené CD obsahuje elektronickou verzi tohoto dokumentu, aplikaci, zdrojové kódy aplikace a dokumentaci.