VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

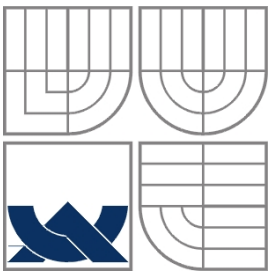
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAXÍ ŘÍZENÝ PŘEKLAD ZALOŽENÝ NA HLUBOKÝCH ZÁSOBNÍKOVÝCH AUTOMATECH

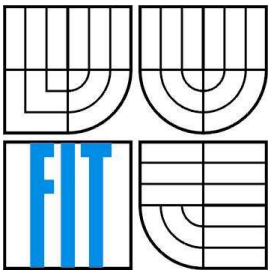
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Peter Solár
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTAXÍ ŘÍZENÝ PŘEKLAD ZALOŽENÝ NA HLUBOKÝCH ZÁSOBNÍKOVÝCH AUTOMATECH

SYNTAX-DIRECTED TRANSLATION BASED ON DEEP PUSHDOWN AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Peter Solár

VEDOUCÍ PRÁCE
SUPERVISOR

prof. RNDr. Alexander Meduna, CSc.

BRNO 2009

Abstrakt

Tato práce představuje zavádí syntaxí řízený překlad založený na použití hlubokých zásobníkových automatů. V teoretické části jde především o zavedení potřebných teoretických modelů. Nejdůležitějším modelem představeným v této práci je hluboký zásobníkový převodník, který lze jednoduše využít při nejdůležitější části překladu - syntaktické analýze. V praktické části je ilustrováno využití nově zavedených modelů při implementaci interpretu jednoduchého programovacího jazyka.

Klíčová slova

překladač, překladové schéma, syntaktická analýza, konečný automat, zásobníkový automat, hluboký zásobníkový automat, bezkontextová gramatika, stavová gramatika, překladová gramatika, konečný převodník, hluboký zásobníkový převodník, interpret, 3adresný kód.

Abstract

This thesis introduces syntax-directed translation based on deep pushdown automata. Necessary theoretical models are introduced in the theoretical part. The most important model, introduced in this thesis, is a deep pushdown transducer. The transducer should be used in syntax analysis, significant part of translation. Practical part consists of an implementation of simple-language interpret based on these models.

Keywords

parser, translation schema, syntax analysis, finite automata, pushdown automata, deep pushdown automata, context-free grammar, state grammar, translation grammar, finite transducer, deep pushdown transducer, interpret, 3-adress code

Citace

Bc. Peter Solár: Syntaxí řízený překlad založený na hlubokých zásobníkových automatech, diplomová práce, Brno, FIT VUT v Brně, 2009

Syntaxí řízený překlad založený na hlubokých zásobníkových automatech

Prohlášení

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

……
Bc. Peter Solár
26. 5. 2009

Poděkování

Rád bych poděkoval panu prof. Alexandru Medunovi za přínosné konzultace a pomoc při řešení této diplomové práce.
Dále bych rád poděkoval Bc. Jiřímu Viktorinovi za rady při řešení praktické části této práce.

Obsah

1 Úvod

Tato práci, věnovaná teorii formálních jazyků, je rozdělena do tří základních částí.

V první části budou představeny základní teoretické modely potřebné k definování syntaxí řízeného překladu založeného na hlubokých zásobníkových automatech. Tento typ syntaxí řízeného překladu je na rozdíl od běžného založen na silnějším typu gramatik (stavové gramatiky). Tato skutečnost zaručuje, že jeho možnosti přesahují běžně používaný, na bezkontextových gramatikách založený, překlad. Než však dojdeme k zavedení zmíněných modelů, bude potřeba připomenout některé základní vědomosti, které jsou potřebné pro lepší pochopení této problematiky.

Ve druhé kapitole se podíváme na potřebný teoretický základ. Připomeneme si použité matematické pojmy, dále uvedeme Chomského klasifikaci jazyků, která je v této teorii stěžejní. Následně si popíšeme dva základní modely pro popis jazyků – gramatiky a automaty. Detailně se zde budu věnovat hlubokým zásobníkovým automatům, na nichž bude dále stavěn model hlubokého zásobníkového převodníku.

Druhá část práce, odpovídající třetí a čtvrté kapitole, se věnuje syntaxí řízenému překladu založenému na hlubokých zásobníkových automatech. K tomu je samozřejmě potřeba uvést obecný úvod k překladačům a syntaxí řízeným modelům překladu.

Poslední část práce je praktická. Jejím cílem je návrh aplikace využívající nově zavedené poznatky. Po konzultacích bylo rozhodnuto pro implementaci interpretu jednoduchého programovacího jazyka.

2 Teoretický základ

V této kapitole bych rád připomněl základní teoretický základ potřebný k lepšímu pochopení dalších částí této práce.

2.1 Matematické pojmy

Nejprve se podívejme na základní matematické symboly a termíny.

Písmenem I budeme dále označovat množinu kladných celých čísel. Pro každou množinu Q můžeme zavést *kardinalitu* $card(Q)$, tedy počet prvků této množiny.

Abeceda je neprázdná konečná množina prvků, nazývaných *symboly*. Nechť Σ je abeceda. Potom ε je *prázdný řetězec* nad abecedou Σ (jedná se o řetězec délky 0). Pokud by x byl řetězec nad abecedou Σ a a je symbol $a \in \Sigma$, pak xa je také řetězec nad abecedou Σ. *Konkatenace* dvou řetězců je jejich spojení za vzniku nového řetězce. Příkladem může být výše uvedený řetězec xa (pozn. symbol a lze chápat také jako řetězec délky 1). Výsledkem konkatenace jakéhokoliv řetězce s prázdným řetězcem je původní řetězec ($x\varepsilon = \varepsilon x = x$).

Ze symbolů abecedy Σ můžeme konkatenací vytvořit množinu neprázdných řetězců Σ^+. Když k této množině přidáme i prázdný řetězec, získáme množinu Σ^*. Tedy platí $\Sigma^* = \Sigma^+ + \{\varepsilon\}$.

Pro jakýkoliv řetězec $x \in \Sigma^*$ zavedeme $|x|$ označující jeho délku (počet všech symbolů, které se v daném řetězci nacházejí) a funkci $alph(w)$ označující množinu symbolů vyskytujících se v řetězci w. Pro každou množinu $X \subseteq \Sigma$ funkce $occur(x, X)$ značí počet výskytů symbolů z množiny X v řetězci x. Pro $i = 1, \ldots, |x|$, $[x, i, X]$ označuje i-tý výskyt některého symbolu z množiny X v řetězci x. Pokud takový výskyt neexistuje, pak $[x, i, X] = 0$.

Pro každé $x \geq 0$ je $prefix(x, i)$ předpona řetězce x o délce i pokud je $|x| \geq i$ nebo $prefix(x, i) = x$ pokud $i > |x|$.

Pojmem *relace* nazýváme libovolný vztah mezi skupinou prvků množin. *Zobrazení* je předpis, jakým způsobem jednoznačně přiřazovat prvkům jedné množiny prvky jiné množiny. *Homomorfismus* je zobrazení z jedné algebraické struktury do jiné stejného typu. Příkladem může být například Morseova abeceda. Homomorfismus f nad Σ^* je takové zobrazení, kdy $f(A) \in \{A, \varepsilon\}$ pro všechna $A \in \Sigma$. *Bijektivní zobrazení* (bijekce) přiřazuje každému prvku první množiny vždy právě jeden prvek druhé množiny.

2.2 Jazyky

Jazyk je z formálního hlediska podmnožina množiny všech možných řetězců nad danou abecedou. Máme-li abecedu Σ a jazyk L, můžeme tuto skutečnost zapsat jako $L \subseteq \Sigma^*$. Jazyky zpravidla dělíme na konečné a nekonečné – podle toho, zda obsahují konečný počet řetězců či nikoliv. Příkladem nekonečného jazyka může být jazyk $L = \{a^n b^n | n \geq 1\}$ nad abecedou $\Sigma = \{a, b\}$. Mezi konečné jazyky patří i $L = \emptyset$ s kardinalitou 0 (jazyk neobsahuje žádný řetězec) a jazyk $L = \{\varepsilon\}$ s kardinalitou 1 (jeden prázdný řetězec).

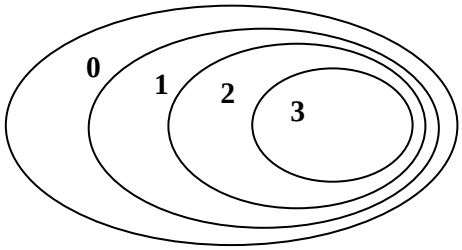
Jazyky můžeme popisovat dvěma základními modely. Prvním modelem jsou gramatiky. Ty podle svých gramatických pravidel generují (derivují) řetězce daného jazyka. Druhým modelem jsou automaty. Na rozdíl od gramatik však řetězce nevytvářejí, ale na základě svých pravidel rozhodují, zda vstupní řetězec patří do jazyka popisovaného tímto automatem.

2.2.1 Chomského klasifikace jazyků

Noam Chomsky v roce 1956 zavedl hierarchii formálních gramatik, generujících rekurzivně spočetné formální jazyky. Tyto gramatiky rozdělil do 4 základních typů.

typ	gramatika	automat	jazyk
0	rekurzivně vyčíslitelná	Turingův stroj (TS)	rekurzivně vyčíslitelný
1	kontextová (CSG)	lineárně ohraničený TS	kontextový (CS)
2	bezkontextová (CFG)	zásobníkový automat (PDA)	bezkontextový (CF)
3	regulární	konečný automat (FA)	regulární

Každý typ této klasifikace je přímou nadmnožinou dalšího typu (viz. Obrázek 1: Chomského hierarchie).



Obrázek 1: Chomského hierarchie

2.3 Gramatiky

Gramatiky jsou jedním ze základních stavebních prvků teorie formálních jazyků. Jádrem každé gramatiky jsou tzv. gramatické pravidla, jejichž pomocí dochází ke generování řetězců patřících do jazyka, který tato gramatika popisuje. V minulosti bylo u gramatik, na rozdíl od automatů, zavedeno velké množství modifikací. Proto uvedu jen dvě základní, které jsou pro tuto práci zajímavé.

2.3.1 Bezkontextové gramatiky

Bezkontextové gramatiky jsou schopny generovat bezkontextové jazyky, tj. typu 2 podle Chomského klasifikace. Do této skupiny jazyků lze zařadit například i programovací jazyky. Bezkontextové gramatiky jsou tedy v porovnání s regulárními gramatikami silnější. Mezi typické jazyky generované bezkontextovou gramatikou patří $L(G) = \{a^n b^n | n \geq 1\}$.

Formálně je bezkontextová gramatika čtveřice

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

kde

N je konečná množina neterminálních symbolů

T je konečná množina terminálních symbolů, $T \cap N = \emptyset$

$S \in N$ je počáteční neterminál

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

2.3.2 Stavové gramatiky

Tyto gramatiky jsou silnější než bezkontextové. Jejich významnou a velice zajímavou vlastností je to, že tvoří nekonečnou hierarchii tříd jazyků mezi jazyky bezkontextovými a kontextovými. To je způsobeno tím, že u stavových gramatik dochází při derivaci větné formy k situacím, kdy nemůžeme z důvodu neexistence vhodného pravidla přepsat nejlevější neterminál, ale můžeme přepsat neterminál umístěný hlouběji ve větné formě.

Formálně je stavová gramatika pětice

$$G = (V, W, T, P, S),$$

kde

V je úplná abeceda, $V = N \cup T$

W je konečná množina stavů

$T \subseteq V$ je abeceda terminálních symbolů

$S \in (V - T)$ je startující symbol

$P \subseteq \big(W \times (V - T)\big) \times (W \times V^+)$ je konečná relace. Přehledněji zapisujeme pravidla ve tvaru $(q, A) \rightarrow (p, v) \in P$ místo matematicky přesnějšího $(q, A, p, v) \in P$.

Každému řetězci $z \in V^*$ nastavíme množinu $states_G(z) = \{q |\ (q, B) \rightarrow (p, v) \in P\}$, kde $B \in (V - T) \cap alph(z)$, tedy množina neterminálů vyskytujících se v tomto řetězci, $v \in V^+$ je neprázdný řetězec a $q, p \in W$ jsou stavy. Pokud máme pravidlo $(q, A) \rightarrow (p, v) \in P$, řetězce $x, y \in V^*$ a množinu $states_G(x) \cap \{q\} = \emptyset$, pak gramatika G provede *derivační krok (derivaci)* z (q, xAy) do (p, xvz). Tuto skutečnost symbolicky zapisujeme $(q, xAy) \Longrightarrow (p, xvy)\ \ [(q, A) \rightarrow (p, v)]$. Dodáme-li kladné celé číslo n splňující podmínku $occur(xA, V - T) \leq n$, říkáme, že $(q, xAy) \Longrightarrow (p, xvy)\ \ [(q, A) \rightarrow (p, v)]$ je n-omezené, symbolicky zapisujeme $(q, xAy)\ {}_n\Longrightarrow (p, xvy)\ \ [(q, A) \rightarrow$

(p, v)]. Pokud nehrozí možnost záměny, můžeme zkráceně psát $(q, xAy) \Longrightarrow (p, xvy)$ a $(q, xAy)\ {}_n\Longrightarrow (p, xvy)$, tedy bez určení, podle kterého pravidla k derivaci došlo.

Obvyklým způsobem můžeme rozšířit derivační krok $\Longrightarrow$ na sérii m derivačních kroků $\Longrightarrow^m$, pro celé číslo $m \geq 0$ určující počet těchto kroků. Také můžeme zavést $\Longrightarrow^+$, kdy je proveden alespoň jeden derivační krok a $\Longrightarrow^*$, kdy nemusí být proveden žádný derivační krok (pokud není proveden žádný krok, musí být obě strany shodné).

Mějme $n \in I$ a $\nu, \varpi \in (\mathrm{W} \times \mathrm{V}^+)$. Abychom vyjádřili, že je každý derivační krok v $\nu \Longrightarrow^m \varpi$, $\nu \Longrightarrow^+ \varpi$ a $\nu \Longrightarrow^* \varpi$ n-omezený, píšeme často symboly derivací jako $\nu\ {}_n\Longrightarrow^m \varpi$, $\nu\ {}_n\Longrightarrow^+ \varpi$ a $\nu\ {}_n\Longrightarrow^* \varpi$. Zápisem $strings(\nu\ {}_n\Longrightarrow^* \varpi)$ vyjadřujeme množinu všech řetězců vyskytujících se v derivaci $\nu\ {}_n\Longrightarrow^* \varpi$.

Jazyk $L(G)$ generovaný gramatikou G je definován jako $L(G) = \{w \in T^* | \ (q, S) \Longrightarrow^* (p, w), p, q \in W\}$. Kromě toho můžeme definovat jazyk generovaný stavovou gramatikou stupně n pro všechna $n \geq 1$ jako $L(G, n) = \{w \in T^* | \ (q, S)\ {}_n\Longrightarrow^* (p, w),\ p, q \in W\}$. Derivace tvaru $(q, S)\ {}_n\Longrightarrow^* (p, w)$, kde $p, q \in W$ a $w \in T^*$, reprezentuje úspěšné n-omezené generování řetězce w v gramatice G.

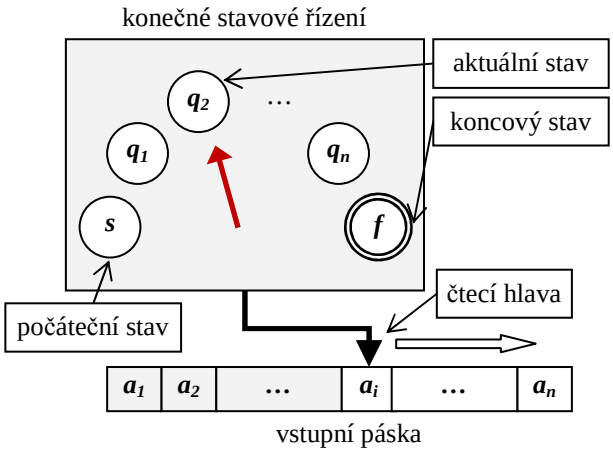
Derivaci nazýváme levou (pravou), pokud v každém jejím kroku nahrazujeme nejlevější (nejpravější) možný neterminál větné formy.

2.4 Automaty

Nyní se podíváme na vývoj automatů. Jak již bylo uvedeno dříve, automat na rozdíl od gramatiky slouží k rozhodování, zda daný řetězec patří do zadaného jazyka či nikoliv.

2.4.1 Konečný automat

Konečný automat je výpočetní model využívaný ke studiu formálních jazyků. Popisuje velice jednoduchý počítač přecházející mezi několika stavy pouze na základě aktuálního vstupu symbolu, získaného ze vstupní pásky. Tento automat tedy nepoužívá žádnou další paměť. Veškeré rozhodování je založeno pouze na znalosti aktuálního stavu a aktuálního vstupního symbolu. Vzhledem ke své jednoduchosti je schopen rozeznat pouze nejjednodušší,



Obrázek 2: Konečný automat

tzv. regulární, jazyky a je tedy vhodným nástrojem například pro zpracování regulárních výrazů. Pro syntaxí řízený překlad, jimž se budeme dále zabývat, se používá především v části lexikální analýzy.

2.4.1.1 Definice

Konečný automat je uspořádaná pětice

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

kde

Q je konečná množina stavů,

Σ je konečná množina vstupních symbolů (vstupní abeceda),

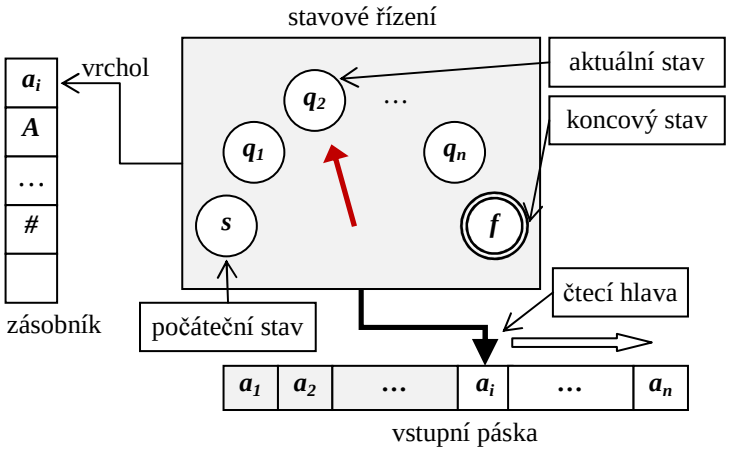
$s \in Q$ je startující (počáteční) stav,

$F \subseteq Q$ je konečná množina koncových stavů

a R je konečná množina pravidel. Čistě matematicky je R relací z $Q \times (\Sigma \cup \{\varepsilon\})$ do Q. Místo zápisu $(pa, q) \in R$ se častěji používá tvar $(pa \to q) \in R, p, q \in Q, a \in (\Sigma \cup \{\varepsilon\})$.

2.4.2 Zásobníkový automat

Zásobníkový automat je rozšířením konečného automatu. Toto rozšíření spočívá především v přidání zásobníku jako paměti. Zásobník dává tomuto automatu možnost rozhodovat se nejen na základě vstupního symbolu a stavu, ale také na základě symbolu umístěného na vrcholu tohoto zásobníku. Tímto rozšířením bylo dosaženo výrazného zvětšení množiny rozpoznávaných jazyků a to až na množinu tzv. bezkontextových jazyků. Podle Chomského klasifikace se jedná o jazyky typu 2.



Obrázek 3: Zásobníkový automat

Se zásobníkem mohou být prováděny dvě základní operace – expanze a vyjmutí. *Expanze* spočívá v nahrazení nevstupního zásobníkového symbolu nacházejícího se na vrcholu zásobníku řetězcem symbolů zásobníkové abecedy a přechodu do jiného stavu. Obě uvedené činnosti odpovídají právě použitému pravidlu. K *vyjmutí* může dojít pouze v situaci, kdy je na vrcholu zásobníku symbol ze vstupní abecedy a zároveň se stejný symbol nachází i na vstupu automatu. Daný symbol bude odebrán z vrcholu zásobníku a zároveň bude přečten ze vstupu. Přečtením se rozumí posun čtecí hlavy po vstupní pásce na následující symbol.

2.4.2.1 Definice

Zásobníkový automat je uspořádaná sedmice

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

kde

Q je konečná množina stavů,

Σ je konečná množina vstupních symbolů (vstupní abeceda),

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

$s \in Q$ je startující (počáteční) stav,

$S \in \Gamma$ je počáteční zásobníkový symbol,

$F \subseteq Q$ je konečná množina koncových stavů

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

2.5 Hluboký zásobníkový automat

Pojem hluboký zásobníkový automat zavedl v článku *Deep pushdown automata* prof. Meduna [7].

2.5.1 Definice

2.5.1.1 Neformální definice

Hluboký zásobníkový automat je rozšířenou variantou klasického zásobníkového automatu. Stejně jako on, používá ke své činnosti zásobník, nad nímž provádí dvě základní operace (expanzi a vyjmutí). Hlavním rozdílem je však fakt, že nekontroluje pouze nejhornější zásobníkový symbol, ale *může provádět kontrolovat i hlouběji umístěné zásobníkové symboly* (omezeno podle typu automatu). Tímto je dosaženo zvýšení tzv. síly. Tento automat je schopen generovat i některé kontextové jazyky. Na rozdíl od základních zásobníkových automatů schopných generovat jen třídu bezkontextových jazyků. Tato obecnější verze je nedeterministická a nepovoluje vymazávající pravidla (přepis neterminálního symbolu prázdným řetězcem).

2.5.1.2 Formální definice

Hluboký zásobníkový automat je uspořádaná sedmice

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

kde

$n \in I$ je maximální hloubka, v níž může dojít k nahrazení,

Q je konečná množina stavů,

Σ je konečná množina vstupních symbolů (vstupní abeceda),

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

$s \in Q$ je startující (počáteční) stav,

$S \in \Gamma$ je počáteční zásobníkový symbol,

$F \subseteq Q$ je konečná množina koncových stavů

a R je konečná množina pravidel tvaru $(mqA \to pv) \in R$, $A \in \Gamma$, $p, q \in Q$, $0 < m \leq n$, $v \in \Gamma^*$.

2.5.2 Konfigurace a přechody

Konfigurace hlubokého zásobníkového automatu obsahuje všechny potřebné údaje, se kterými automat pracuje. Těmito údaji jsou aktuální stav, nezpracovaná část vstupního řetězce a stav zásobníku (řetězec zásobníkových symbolů). Formálně je to trojice z $Q \times \Sigma^* \times (\Gamma - \{\#\})^*\{\#\}$.

Označme χ množinu všech možných konfigurací daného automatu. Dále pak označme $x, y \in \chi$. Přechody mezi jednotlivými konfiguracemi značíme $x \Longrightarrow y$ a pokud chceme zkrátit zápis, například přechod ze startovní do koncové konfigurace bez určení přesného počtu kroků, můžeme použít zápis $x \Longrightarrow^* y$. Přesný počet kroků můžeme zapsat obdobně, jen s tím rozdílem, že nahradíme znak $*$ číslem vyjadřujícím počet daných přechodů (např. $x \Longrightarrow^2 y$).

Obdobně jako u běžných zásobníkových automatů provádí i hluboké zásobníkové automaty expanzi $x \ {}_e\Longrightarrow y$ a vyjmutí $x \ {}_p\Longrightarrow y$.

2.5.3 Příklad

Uvažujme hluboký zásobníkový automat ${}_2M = (\{s, p, q, f\}, \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}, \{A, S, \#\}, R, s, S, \{f\})$, který má v R následující pravidla:

$1sS \to qAA$, [1]

$1qA \to paAb$, [2]

$1qA \to fab$, [3]

$2pA \to qAc$, [4]

$1fA \to fc$. [5]

Se vstupním řetězcem $aaabbbccc$ provede automat M tyto kroky:

$(s, aaabbbccc, S\#) \ {}_e\Longrightarrow (q,\ aaabbbccc,\ AA\#)$ [1]

${}_e\Longrightarrow (p,\ aaabbbccc,\ aAbA\#)$ [2]

${}_p\Longrightarrow (p,\ aabbbccc,\ AbA\#)$

${}_e\Longrightarrow (q,\ aabbbccc,\ AbAc\#)$ [4]

${}_e\Longrightarrow (p,\ aabbbccc,\ aAbbAc\#)$ [2]

${}_p\Longrightarrow (p,\ abbbccc,\ AbbAc\#)$

$$\underset{e}{\Longrightarrow}(q,\ abbbccc,\ AbbAcc\#) \quad [4]$$

$$\underset{e}{\Longrightarrow}(f,\ abbbccc,\ abbbAcc\#) \quad [3]$$

$$\underset{p}{\Longrightarrow}(f,\ bbbccc,\ bbbAcc\#)$$

$$\underset{e}{\Longrightarrow}(f,\ bbbccc,\ bbbccc\#) \quad [5]$$

$$\underset{p}{\Longrightarrow}(f,\ bbccc,\ bbccc\#)$$

$$\underset{p}{\Longrightarrow}(f,\ bccc,\ bccc\#)$$

$$\underset{p}{\Longrightarrow}(f,\ ccc,\ ccc\#)$$

$$\underset{p}{\Longrightarrow}(f,\ cc,\ cc\#)$$

$$\underset{p}{\Longrightarrow}(f,\ c,\ c\#)$$

$$\underset{p}{\Longrightarrow}(f,\ \varepsilon,\ \#)$$

Tedy platí $(s, aaabbbccc, S\#) \Longrightarrow^{*} (f, \varepsilon, \#)$ a zadaný řetězec byl přijat po 16 krocích (z nichž bylo 7 expanzí a 9 vyjmutí). Dále si můžeme všimnout, že jazyk přijímaný tímto automatem $L({}_2M) = \{a^n b^n c^n | n \geq 1\}$ patří do skupiny jazyků definovaných jako $CS - CF$.

2.5.4 Vlastnosti

Nyní si uvedeme některé základní vlastnosti hlubokých zásobníkových automatů. Případné důkazy a podrobnosti lze nalézt v [7].

Hluboké zásobníkové automaty a n-omezené stavové gramatiky jsou ekvivalentní modely. To vyplývá z následujícího teorému:

Teorém: Pro každé $n \geq 1$ a každý jazyk L platí, že je generovaný n-omezenou stavovou gramatikou G, $L = L(G, n)$ jen v případě, že je přijímán hlubokým zásobníkovým automatem ${}_nM$, $L = L({}_nM)$.

Na základě uvedeného teorému a vlastností n-omezených stavových gramatik lze dojít k závěru, že třídy jazyků přijímaných hlubokými zásobníkovými automaty tvoří taktéž nekonečnou hierarchii mezi třídami kontextových a bezkontextových jazyků. Jedná se o nejdůležitější závěr článku o hlubokých zásobníkových automatech [7].

Při omezení maximální hloubky pravidel hlubokého zásobníkového automatu na hodnotu 1 se z něj stane klasický zásobníkový automat, jehož síla bude stačit pouze na bezkontextové jazyky.

2.5.5 Determinismus

Determinismus je vlastnost automatu (algoritmu), kdy daný automat (algoritmus) zareaguje na zadané výchozí podmínky vždy stejným, a tedy předvídatelným, způsobem.

Doposud uvedené definice a vlastnosti odpovídají nedeterministické variantě hlubokých zásobníkových automatů. Na rozdíl od klasických zásobníkových automatů můžeme determinismus v tomto případě chápat dvěma různými způsoby.

2.5.5.1 Striktní determinismus

Prvním způsobem, který nás může napadnout, je *striktní determinismus*. Jeho základem je předpoklad, že hluboký zásobníkový automat ${}_nM = (Q, \Sigma, \Gamma, R, s, S, F)$ může v každém stavu použít nejvýše jedno pravidlo (platí celkově pro všechny přípustné hloubky). Formálně pro všechna $(mqA \to pv) \in R$, $card(\{mqA \to ow \mid mqA \to ow \in R, o \in Q, w \in \Gamma^+\} - \{mqA \to pv\}) = 0$.

Není dokázáno, zda je striktně deterministický hluboký zásobníkový automat stejně silný jako nedeterministická varianta.

2.5.5.2 Determinismus s ohledem na hloubku

Determinismus s ohledem na hloubku je slabší formou determinismu. Zde je opět předpokladem možnost použití maximálně jediného pravidla, ale s platností pro každou přípustnou hloubku. Pro všechna $q \in Q$, $card(\{m|mqA \to pv \ \in R, p \in Q, A \in \Gamma, v \in \Gamma^+\}) \leq 1$, protože z jednoho stavu mají všechny expanze, které automat M může udělat, stejnou hloubku.

Na základě konstrukce automatu ${}_nM$ v konstrukci důkazu Lemmatu 1, který je deterministický s ohledem na hloubku pravidel lze odvodit následující vztah:

Pro každé $n \geq 1$ a každou stavovou gramatiku G existuje hluboký zásobníkový automat ${}_nM$, takový, že $L({}_nM) = L(G, n)$ a ${}_nM$ je deterministický s ohledem na hloubku expanze.

3 Překladače

V současné době téměř každý programátor programuje v některém z vyšších programovacích jazyků. Těmto jazykům však procesor počítače nerozumí a proto je potřeba transformovat programy napsané programátorem do strojového jazyka. Při této transformaci samozřejmě požadujeme, aby program ve strojovém jazyce prováděl přesně to, co původní verze napsaná programátorem. Tato transformace se nazývá *překlad* a má ji na starosti program zvaný *překladač*.

Překladač je program, který na svém vstupu přijímá zdrojový program (ve zdrojovém jazyce) a na výstupu produkuje cílový program (v cílovém jazyce).

Kompilátor je speciální druh překladače, překládající programy z vyšších programovacích jazyků na ekvivalentní programy v nižších programovacích jazycích.

Assembler je program, překládající program ze strojového jazyka na ekvivalentní strojový kód.

V praxi se také můžeme setkat i s pojmy *dekompilátor* a *disassembler*. Jejich činnost je opačná k výše uvedeným pojmům.

Interpret je program, který je schopen přímo vykonávat program napsaný ve zdrojovém jazyce.

Překlad se obvykle skládá z následujících fází:

- lexikální analýza
- syntaktická analýza
- sémantická analýza
- generování vnitřní formy programu
- optimalizace
- generování cílového programu

Podle jednotlivých fází se nazývají i jednotlivé části kompilátoru.

Lexikální analyzátor je první částí kompilátoru. Jeho úkolem je projít kód zdrojového programu a rozdělit jej na posloupnost lexikálních symbolů a jednotlivým symbolům přiřadit typ (identifikátor, klíčové slovo, konstanta, …). Výstup lexikální analýzy se předá *syntaktickému analyzátoru*. Ten má zjistit, zda je zdrojový program napsán syntakticky správně (tedy uvedená posloupnost lexikálních symbolů patří do daného jazyka). Výstupem syntaktického analyzátoru je obvykle *derivační strom*. *Sémantický analyzátor* kontroluje celou řadu dalších, tzv. sémantických aspektů (jsou proměnné deklarovány?, jsou proměnné správného typu?, apod.).

Generátor kódu vytvoří na základě syntaktické struktury cílový program. Součástí tohoto generování mohou být provedeny různé *optimalizace*.

V praxi se však první 3 fáze (lexikální, syntaktická a sémantická analýza) zpravidla prolínají a jednotlivé části kompilátoru spolupracují.

3.1 Formální překlad

Mějme abecedy Σ_I, Σ_O a jazyky $L_I \subseteq \Sigma_I^*$, $L_O \subseteq \Sigma_O^*$. Překladem z jazyka L_I do jazyka L_0 nazveme relaci

$$P \subseteq L_I \times L_O$$

Σ_I je vstupní abeceda

Σ_O je výstupní abeceda

Jestliže $(x, z) \in P$, pak slovo z nazveme překladem věty x.

Překlad je relací, tedy může existovat více výstupů pro jednu konkrétní vstupní větu. To však v praxi může vést k problémům. Protože je překlad tvořen nekonečnou množinou dvojic slov, potřebujeme mít prostředky, které by nám takovéto nekonečné množiny umožnily specifikovat.

3.2 Syntaxí řízené překladové schéma

Je jedním z nejznámějších prostředků specifikujících nekonečné množiny překladu.

Formálně se jedná o pětici

$$T = (N, \Sigma_I, \Sigma_O, R, S),$$

kde

N je abeceda neterminálů,

Σ_I je vstupní abeceda,

Σ_O je výstupní abeceda,

$S \in N$ je počáteční neterminál,

a R je konečná množina pravidel tvaru $A \to x, y,\ A \in N,\ x \in (N \cup \Sigma_I)^*,\ y \in (N \cup \Sigma_O)^*$. Přitom neterminály z y jsou permutací neterminálů z x.

Překladová forma schématu T je definována:

i, (S, S) je překladová forma, v níž jsou výskyty neterminálů S přidruženy

ii, je-li (uAv, wAz) překladová forma, uvedené výskyty A jsou přidružené neterminály, $u, v \in (N \cup \Sigma_I)^*,\ w, z \in (N \cup \Sigma_O)^*,\ A \to x$ a $y \in R$, potom (uxv, wyz) je překladová forma, kde přidružení výskytů symbolů v x a y je dáno jejich přidružením v pravidle $A \to x, y$. Pak říkáme, že překladová forma (uAv, xAz) *přímo derivuje* překladovou formu (uxv, vyz). Symbolicky

$$(uAv, wAz) \Rightarrow (uxv, wyz).$$

Relaci ⇒ nazýváme *překladová derivace*.

Překlad definovaný schématem T označujeme $P(T)$.

$$P(T) = \{(x, y); (S, S) \Rightarrow^* (x, y), x \in \Sigma_I^*, y \in \Sigma_O^*\}.$$

4 Syntaxí řízený překlad založený na hlubokých zásobníkových automatech

4.1 Lexikální analýza

Lexikální analýza je první část překladu. Jejím hlavním úkolem nalezení a rozpoznání jednotlivých lexikálních symbolů ve zdrojovém programu. Mimo to však musí lexikální analyzátor jednotlivé lexikální symboly zakódovat tak, aby s nimi mohl následně syntaktický analyzátor lépe pracovat. Mezi další, méně významné, úkoly patří také např. vynechání komentářů a jiných textů, sloužících k zpřehlednění kódu.

Teoreticky je lexikální analýza samostatnou částí překladu, jejíž výstup je pak předložen na vstup syntaktického analyzátoru. V praxi se však mnohem častěji setkáme s tím, že je lexikální analyzátor podprogramem syntaktického analyzátoru, který jej zavolá vždy, když potřebuje další lexikální symbol.

4.1.1 Regulární syntaxí řízené překladové schéma

Pro začátek si zavedeme speciální případ řízených překladových schémat.

Mějme syntaxí řízené přechodové schéma $T = (N, \Sigma_I, \Sigma_O, R, S)$ a každé pravidlo z množiny R je v jednom z těchto tvarů:

1, $A \to aB, xB$,

2, $A \to a, x$,

$a \in \Sigma_I \cup \{\varepsilon\}$, $x \in \Sigma_O^*$.

Pak T nazýváme *regulární překladové schéma*. Překlad $P(T)$ pak nazveme *regulární překlad*.

4.1.2 Konečný převodník

Jednoduchým rozšířením konečného automatu o možnost výstupu řetězu symbolů nad výstupní abecedou získáme tzv. *konečný převodník*.

Konečný převodník M je šestice

$$M = (Q, \Sigma_I, \Sigma_O, g, q_0, F),$$

kde

Q je konečná množina stavů,

Σ_I je vstupní abeceda,

Σ_O je výstupní abeceda,

g je zobrazení $Q \times (\Sigma \cup \{\varepsilon\})$ do množiny konečných podmnožin $Q \times \Sigma_O^*$,

$q_0 \in Q$ je počáteční stav,

F je konečná množina koncových stavů.

Obdobně jako u konečného automatu můžeme i zde definovat konfiguraci konečného převodníku.

Konfigurace konečného *převodníku M* je trojice

$$(q, x, y) \in Q \times \Sigma_I^* \times \Sigma_O^*,$$

kde

q je stav konečného převodníku

$x \in \Sigma_I^*$ je dosud nepřečtená část vstupního řetězce (sufix)

$y \in \Sigma_O^*$ je dosud vytvořená část výstupního řetězce (prefix)

Mějme dvě konfigurace konečného převodníku M, (q, ax, y) a (r, x, yz), $q, r \in Q$, $x \in \Sigma_I^*$, $y, z \in \Sigma_O^*$, $a \in \Sigma_I \cup \{\varepsilon\}$. M přímo přejde z konfigurace (q, ax, y) do konfigurace (r, x, yz), $(q, ax, y) \Rightarrow (r, x, yz)$, když a jen když $g(q, a)$ obsahuje (r, z).

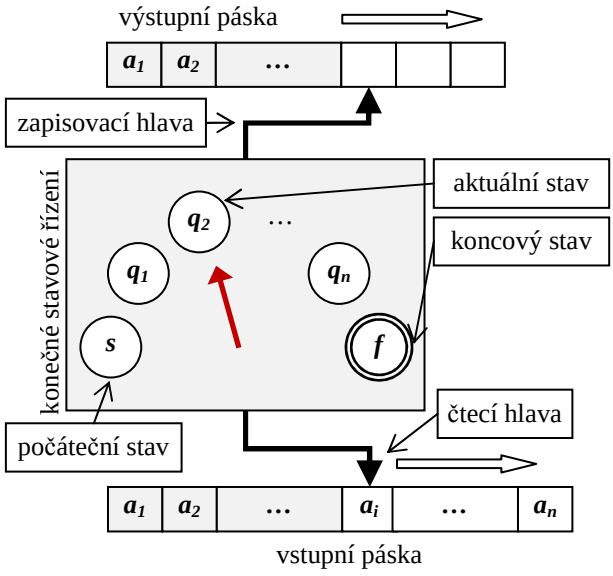
Pokud chceme zkrátit zápis, například přechod ze startovní do koncové konfigurace bez určení přesného počtu kroků, můžeme použít $(q, ax, y) \Longrightarrow^* (r, x, yz)$. Přesný počet kroků můžeme zapsat obdobně, jen s tím rozdílem, že nahradíme znak $*$ číslem vyjadřujícím počet daných přechodů (např. $(q, ax, y) \Longrightarrow^2 (r, x, yz)$).

Konfiguraci (q_0, x, ε), $x \in \Sigma_I^*$ nazýváme *počáteční*, konfiguraci (q, ε, y), $q \in F$, $y \in \Sigma_O^*$ nazýváme *koncovou*.

$$(q_0, x, \varepsilon) \Longrightarrow^* (q, \varepsilon, y), q \in F$$

Překlad $P(M)$ definovaný konečným převodníkem M je množina dvojic

$$P(M) = \{(x, y); (q_0, x, \varepsilon) \Longrightarrow^* (q, \varepsilon, y), q \in F, x \in \Sigma_I^*, y \in \Sigma_O^*\}$$



Obrázek 4: Konečný převodník

4.1.3 Vztah regulárních syntaxí řízených překladových schémat a konečných převodníků

Podobně jako je tomu u konečných automatů a regulárních gramatik (nebo regulárních výrazů), i mezi regulárními syntaxí řízenými překladovými schématy a konečnými převodníky existuje velice těsná souvislost. Tuto vazbu vystihuje následující věta:

Věta: Nechť T je regulární syntaxí řízené schéma. Pak existuje konečný převodník M takový, že $P(M) = P(T)$.

Důkaz lze nalézt například v kapitole 3.2.4 studijní opory předmětu IFJ [9].

4.2 Syntaktická analýza

Syntaktická analýza je druhou a zároveň nejdůležitější fází překladu. Úkolem této fáze je určení syntaktické struktury zdrojového kódu. Syntaktický analyzátor se obecně snaží rozhodnout, zda daný vstupní řetězec (zdrojový program) patří do daného zdrojového jazyka a následně se snaží vytvořit pro danou posloupnost lexikálních symbolů zdrojového kódu derivační strom. Vzhledem ke známému vztahu derivačních stromů s levými/pravými derivacemi bývá často vhodnější hledat pro danou posloupnost levou/pravou derivaci (viz 2.3.2).

Definice syntaktické analýzy je následující:

Mějme gramatiku $G = (N, T, P, S)$, která má n pravidel číslovaných $1, \dots, n$ a mějme řetězec $w \in L(G)$.

Obecná syntaktická analýza je proces vedoucí k nalezení posloupnosti čísel pravidel gramatiky G použitých při libovolné derivaci věty w.

Syntaktická analýza metodou shora dolů je proces vedoucí k nalezení posloupnosti čísel pravidel použitých při levé derivaci věty w.

Syntaktická analýza metodou zdola nahoru je proces vedoucí k nalezení obrácené posloupnosti čísel pravidel použitých při pravé derivaci.

4.2.1 Jednoduché syntaxí řízené překladové schéma

Základním formálním prostředkem, jehož pomocí můžeme přeložit věty zadaného stavového jazyka na jím odpovídající levé/pravé rozbory, je speciální případ syntaxí řízeného překladového schématu.

Mějme syntaxí řízené překladové schéma ve tvaru šestice

$$T = (V, W, \Sigma_I, \Sigma_O, R, S),$$

kde

V je úplná abeceda, $N = V - \Sigma_I - \Sigma_O$

W je konečná množina stavů

Σ_I je vstupní abeceda

Σ_O je výstupní abeceda

$S \in N$ je počáteční neterminál,

a R je konečná množina pravidel tvaru $pA \to qx, y,\ p, q \in W,\ A \in N,\ x \in (N \cup \Sigma_I)^*,$ $y \in (N \cup \Sigma_O)^*$. Přitom neterminály z y jsou permutací neterminálů z x.

Pak T nazýváme jednoduché syntaxí řízené schéma.

Překlad $P(T)$ definovaný jednoduchým syntaxí řízeným překladovým schématem se nazývá jednoduchý syntaxí řízený překlad.

4.2.2 Hluboký zásobníkový převodník

Hluboký zásobníkový převodník může vzniknout rozšířením hlubokého zásobníkového automatu. Rozšíření spočívá obdobně jako u konečného převodníku především v možnosti výstupu řetězce nad výstupní abecedou.

Hluboký zásobníkový převodník je osmice

$$_nM = (Q, \Sigma_I, \Gamma, \Sigma_O, \delta, q_0, S, F),$$

kde

$n \in I$ je maximální hloubka, v níž může dojít k nahrazení

Q je konečná množina stavů

Σ_I je vstupní abeceda

Γ je zásobníková abeceda, $\Sigma_I, \Sigma_O \subset \Gamma,\ \# \in \Gamma - \Sigma_I - \Sigma_O$, # je speciální pomocný symbol značící dno zásobníku

Σ_O je výstupní abeceda

δ je zobrazení z $I \times Q \times (\Sigma_I \cup \{\varepsilon\}) \times \Gamma$ do množiny konečných podmnožin množiny $Q \times \Gamma \times \Sigma_O^*$

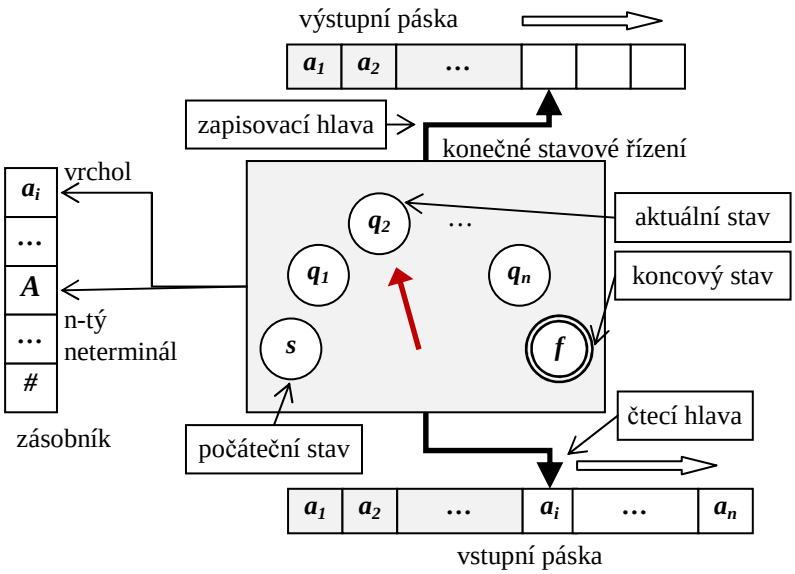
$q_O \in Q$ je počáteční stav

$S \in \Gamma$ je počáteční zásobníkový symbol

$F \subseteq Q$ je konečná množina koncových stavů

4.2.2.1 Konfigurace a přechody

Konfigurace hlubokého zásobníkového převodníku obsahuje všechny potřebné údaje, se kterými převodník pracuje. Obdobně jako u hlubokého zásobníkového automatu jsou těmito údaji aktuální stav, nezpracovaná část vstupního řetězce a stav zásobníku (řetězec zásobníkových symbolů). Dále potřebujeme znát již vytvořený prefix výstupního řetězu.



Obrázek 5: Hluboký zásobníkový převodník

Formálně je tedy konfigurace tohoto převodníku čtveřice

$$(q, x, z, y),$$

kde

$q \in Q$ je aktuální stav převodníku

$x \in \Sigma_I^*$ je nepřečtená část vstupního řetězce (sufix)

$z \in (\Gamma - \{\#\})^*\{\#\}$ je stav zásobníku

$y \in \Sigma_O^*$ je dosud vygenerovaná část výstupního řetězce (prefix)

Označme χ množinu všech možných konfigurací daného převodníku. Dále pak označme $a, b \in \chi$. Přechody mezi jednotlivými konfiguracemi značíme $a \Longrightarrow b$ a pokud chceme zkrátit zápis, například přechod ze startovní do koncové konfigurace bez určení přesného počtu kroků, můžeme použít $a \Longrightarrow^* b$ (tranzitivní uzávěr relace přechodu). Přesný počet kroků můžeme zapsat nahrazením znaku $*$ číslem vyjadřujícím počet daných přechodů (např. $a \Longrightarrow^2 b$).

Podobně jako jiných převodníků a automatů, i zde můžeme nalézt dvě významné konfigurace – počáteční (q_0, x, S, ε) a koncovou (q, ε, z, y), $q \in F$.

Překlad $P(M)$ definovaný hlubokým zásobníkovým převodníkem M je množina

$$P(M) = \{(x, y); (q_0, x, S, \varepsilon) \Rightarrow^* (q, \varepsilon, z, y), pro\, q \in Q, z \in (\Gamma - \{\#\})^*\{\#\}\}$$

Překlad $P_e(M)$ definovaný hlubokým zásobníkovým převodníkem M s prázdným zásobníkem je množina dvojic:

$$P_e(M) = \{(x, y); (q_0, x, S, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon, y), \text{pro } q \in Q\}$$

4.2.2.2 Deterministický hluboký zásobníkový převodník s ohledem na hloubku expanzí

V praxi je vždy výhodnější mít jako model syntaktického analyzátoru deterministický hluboký zásobníkový převodník. Jak již bylo uvedeno u hlubokých zásobníkových automatů, není dokázáno, zda je striktně deterministický hluboký zásobníkový automat stejně silný jako nedeterministický, a proto se zaměříme na slabší formu determinismu.

Hluboký zásobníkový převodník

$$_nM = (Q, \Sigma_I, \Gamma, \Sigma_O, \delta, q_0, S, F)$$

se nazývá deterministický s ohledem na hloubku expanzí, pokud platí následující podmínky:

1, $\forall q \in Q, a \in \Sigma_I \cup \{\varepsilon\}, B \in \Gamma$: $card(\delta(i, q, a, B)) \leq 1$

2, $\forall q \in Q, a \in \Sigma_I, B \in \Gamma$: jestliže $\delta(i, q, \varepsilon, B) \neq \emptyset$, pak $\delta(i, q, a, B) = \emptyset$

4.2.3 Vztah jednoduchých syntaxí řízených překladových schémat a hlubokých zásobníkových převodníků

Věta: Nechť $T = (V, W, \Sigma_I, \Sigma_O, R, S)$ je jednoduché syntaxí řízené překladové schéma. Pak existuje hluboký zásobníkový převodník M takový, že $P(M) = P(T)$.

Důkaz: Mějme jednoduché syntaxí řízené překladové schéma $T = (V, W, \Sigma_I, \Sigma_O, R, S)$. Bez újmy na obecnosti můžeme předpokládat, že $\Sigma_I \cap \Sigma_O = \emptyset$. Množina $N = V - \Sigma_I - \Sigma_O$, viz definice v kapitole 4.2.1. Dále nadefinujme homomorfismus h nad $(\{\#\} \cup V)^*$ jako $h(A) = A$ pro všechna $A \in \{\#\} \cup N$ a $h(a) = \varepsilon$ pro všechna $a \in V - N$.

Sestrojíme hluboký zásobníkový převodník

$$_nM = (Q, \Sigma_I, V \cup \{\#\}, \Sigma_O, \delta, s, S, F),$$

kde

$Q = \{s\} \cup \{\langle p, u\rangle | p \in W, u \in prefix(N^*\{\#\}^n, n), |u| \leq n\}$

$F = \{\langle p, u\rangle | p \in W, u \in prefix(\{\#\}^n, n), |u| \leq n\}$

a množinu δ sestrojíme následovně:

1. pro každé pravidlo $pS \to qx, y \in R,\ p, q \in W, x \in \Sigma_I^+, y \in \Sigma_O^*$ přidáme
   $\delta(1, p, \varepsilon, S) = (\langle p, S\rangle, S, \varepsilon)$
2. jestliže $pA \to qx, y \in R, x = x_0 B_1 x_1 \dots B_k x_k,\ y = y_0 B_1 y_1 \dots B_k y_k,\ p, q \in W,\ k \geq 0,$
   $1 \leq j \leq k, 0 \leq i \leq k,\ A, B_j \in N,\ \langle p, uAv\rangle \in Q,\ \ x_i \in \Sigma_I^*, y_i \in \Sigma_O^*,\ u \in N^*,\ v \in N^*\{\#\}^*,\ |uAv| = n,\ p \notin states_G(u),$
   potom $(\langle q, prefix(u\ h(x)\ v, n)\rangle, x_0 y_0 B_1 x_1 y_1 \dots B_k x_k y_k, \varepsilon) \in \delta(|uA|, \langle p, uAv\rangle, \varepsilon, A)$
3. jestliže $A \in N, p \in W, u \in N^*, v \in \{\#\}^*, |uv| = n - 1,\ p \notin states_G(u),$
   potom $(\langle p, uAv\rangle, A, \varepsilon) \in \delta(|uA|, \langle p, uv\rangle, \varepsilon, A)$
   a $(\langle p, uv\#\rangle, \#, \varepsilon) \in \delta(|uA|, \langle p, uv\rangle, \varepsilon, \#)$

4. $\delta(1, q, a, a) = \{(q, \varepsilon, \varepsilon)\}$ pro všechna $a \in \Sigma_I$ a všechna $q \in Q$
5. $\delta(1, q, \varepsilon, b) = \{(q, \varepsilon, b)\}$ pro všechna $b \in \Sigma_O$ a všechna $q \in Q$

Příklad 4.2.3.1: Mějme jednoduché syntaxí řízené překladové schéma $T = (V, \{p\}, \Sigma_I, \Sigma_O, R, E),\ N = \{A, E, F\}, \Sigma_I = \{+, *,), (, a\},\ \Sigma_O = \{\ddot{+}, \ddot{*}, \ddot{a}\}$. Symboly v množině Σ_O jsou označeny ¨ z důvodu zajištění předpokladu $\Sigma_I \cap \Sigma_O = \emptyset$. Uvedené schéma je obdobou překladového schématu pro převod výrazů do postfixové (polské) notace z příkladu 3.3.10 Studijní opory IFJ [9].[1]

Kde R obsahuje pravidla:

$p\ E \rightarrow p\ E + A, EA\ddot{+}$

$p\ E \rightarrow p\ A, A$

$p\ A \rightarrow p\ A * F, AF\ \ddot{*}$

$p\ A \rightarrow p\ F, F$

$p\ F \rightarrow p\ (E), E$

$p\ F \rightarrow p\ a, \ddot{a}$

Hluboký zásobníkový převodník sestrojený podle uvedeného postupu vypadá následovně:

${}_1M = (Q, \Sigma_I, \mathrm{V} \cup \{\#\}, \Sigma_O, \delta, p, E, \{\langle p, \# \rangle\}),$

$Q = \{p, \langle p, \varepsilon \rangle, \langle p, A \rangle, \langle p, E \rangle, \langle p, F \rangle, \langle p, \# \rangle\}$

$\delta(1, p, \varepsilon, E) = (\langle p, E \rangle, E, \varepsilon)$ [1]

$\delta(|E|, \langle p, E \rangle, \varepsilon, E) = \{(\langle p, E \rangle, E + A\ddot{+}, \varepsilon), (\langle p, A \rangle, A, \varepsilon)\}$ [2], [3]

$\delta(|A|, \langle p, A \rangle, \varepsilon, A) = \{(\langle p, A \rangle, A * F\ \ddot{*}, \varepsilon), (\langle p, F \rangle, F, \varepsilon)\}$ [4], [5]

$\delta(|F|, \langle p, F \rangle, \varepsilon, F) = \{(\langle p, E \rangle, (E), \varepsilon), (\langle p, \varepsilon \rangle, a\ \ddot{a}, \varepsilon)\}$ [6], [7]

$\delta(|E|, \langle p, \varepsilon \rangle, \varepsilon, E) = \{(\langle p, E \rangle, E, \varepsilon)\}$ [8]

$\delta(|A|, \langle p, \varepsilon \rangle, \varepsilon, A) = \{(\langle p, A \rangle, A, \varepsilon)\}$ [9]

$\delta(|F|, \langle p, \varepsilon \rangle, \varepsilon, F) = \{(\langle p, F \rangle, F, \varepsilon)\}$ [10]

$\delta(|E|, \langle p, \varepsilon \rangle, \varepsilon, \#) = \delta(|A|, \langle p, \varepsilon \rangle, \varepsilon, \#) = \delta(|F|, \langle p, \varepsilon \rangle, \varepsilon, \#) = \{(\langle p, \# \rangle, \#, \varepsilon)\}$ [11]

$\delta(1, q, c, c) = \{(q, \varepsilon, \varepsilon)\}$ pro všechna $c \in \Sigma_I$ a všechna $q \in Q$ [pop]

$\delta(1, q, \varepsilon, d) = \{(q, \varepsilon, d)\}$ pro všechna $d \in \Sigma_O$ a všechna $q \in Q$ [out]

Pro vstupní posloupnost $a * (a + a)$ provede hluboký zásobníkový převodník následující posloupnost taktů:

$(p, a * (a + a), E\#, \varepsilon) \Rightarrow (\langle p, E \rangle, a * (a + a), E\#, \varepsilon)$ [1]

$\Rightarrow (\langle p, A \rangle, a * (a + a), A\#, \varepsilon)$ [3]

$\Rightarrow (\langle p, A \rangle, a * (a + a), A * F\ \ddot{*}\ \#, \varepsilon)$ [4]

$\Rightarrow (\langle p, F \rangle, a * (a + a), F * F\ \ddot{*}\ \#, \varepsilon)$ [5]

[1] Pro přehlednost byl zvolen příklad nevyužívající všech vlastností hlubokých zásobníkových převodníků.

$$\Longrightarrow (\langle p, \varepsilon\rangle, a * (a + a), a\ \ddot{a} * F \ddot{*} \#, \varepsilon) \quad [7]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, * (a + a), \ddot{a} * F \ddot{*} \#, \varepsilon) \quad [pop]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, * (a + a), * F \ddot{*} \#, \ddot{a}) \quad [out]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, (a + a), F \ddot{*} \#, \ddot{a}) \quad [pop]$$
$$\Longrightarrow (\langle p, F\rangle, (a + a), F \ddot{*} \#, \ddot{a}) \quad [10]$$
$$\Longrightarrow (\langle p, E\rangle, (a + a), (E) \ddot{*} \#, \ddot{a}) \quad [6]$$
$$\Longrightarrow (\langle p, E\rangle, a + a), E) \ddot{*} \#, \ddot{a}) \quad [pop]$$
$$\Longrightarrow (\langle p, E\rangle, a + a), E + A\ddot{+}) \ddot{*} \#, \ddot{a}) \quad [2]$$
$$\Longrightarrow (\langle p, A\rangle, a + a), A + A\ddot{+}) \ddot{*} \#, \ddot{a}) \quad [3]$$
$$\Longrightarrow (\langle p, F\rangle, a + a), F + A\ddot{+}) \ddot{*} \#, \ddot{a}) \quad [5]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, a + a), a\ \ddot{a} + A\ddot{+}) \ddot{*} \#, \ddot{a}) \quad [7]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, +a), \ddot{a} + A\ddot{+}) \ddot{*} \#, \ddot{a}) \quad [pop]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, +a), +A\ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}) \quad [out]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, a), A\ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}) \quad [pop]$$
$$\Longrightarrow (\langle p, A\rangle, a), A\ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}) \quad [9]$$
$$\Longrightarrow (\langle p, F\rangle, a), F\ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}) \quad [5]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, a), a\ \ddot{a}\ \ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}) \quad [7]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle,), \ddot{a}\ \ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}) \quad [pop]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle,), \ddot{+}) \ddot{*} \#, \ddot{a}\ \ddot{a}\ \ddot{a}\ ) \quad [out]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle,),) \ddot{*} \#, \ddot{a}\ \ddot{a}\ \ddot{a}\ \ddot{+}) \quad [out]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, \varepsilon, \ddot{*} \#, \ddot{a}\ddot{a}\ddot{a}\ddot{+}) \quad [pop]$$
$$\Longrightarrow (\langle p, \varepsilon\rangle, \varepsilon, \#, \ddot{a}\ddot{a}\ddot{a}\ddot{+}\ \ddot{*}) \quad [out]$$
$$\Longrightarrow (\langle p, \#\rangle, \varepsilon, \#, \ddot{a}\ddot{a}\ddot{a}\ddot{+}\ \ddot{*}) \quad [11]$$

Věta: Nechť ${}_nM = (Q, \Sigma_I, \Gamma, \Sigma_O, \delta, q_0, S, F)$ je hluboký zásobníkový převodník. Pak existuje jednoduché syntaxí řízené překladové schéma T takové, že $P(T) = P(M)$.

4.2.4 Překladová gramatika

Překladové gramatiky jsou dalším, jednodušším prostředkem, než dříve zmíněná překladová schémata.

Překladová gramatika je stavová gramatika $G = (V, W, \Sigma_I \cup \Sigma_O, P, S)$, která má množinu terminálních symbolů rozdělenu na dvě disjunktní podmnožiny - Σ_I (množina vstupních symbolů) a Σ_O (množina výstupních symbolů).

Pro každou překladovou gramatiku definujme:

- Vstupní homomorfismus h_I jako zobrazení z $N \cup \Sigma_I \cup \Sigma_O$ do $N \cup \Sigma_I \cup \{\varepsilon\}$ definované:
  1. jestliže $X \in N \cup \Sigma_I$, pak $h_I(X) = X$
  2. jestliže $X \in \Sigma_O$, pak $h_I(X) = \varepsilon$
- Výstupní homomorfismus h_O jako zobrazení z $N \cup \Sigma_I \cup \Sigma_O$ do $N \cup \Sigma_I \cup \{\varepsilon\}$ definované:
  1. jestliže $X \in N \cup \Sigma_O$, pak $h_O(X) = X$
  2. jestliže $X \in \Sigma_I$, pak $h_O(X) = \varepsilon$

Dále tyto homomorfismy rozšíříme na definiční obor $(N \cup \Sigma_I \cup \Sigma_O)^*$

$h_I(\varepsilon) = \varepsilon$

$h_I(Xx) = h_I(X)h_I(x)$

$h_O(\varepsilon) = \varepsilon$

$h_O(Xx) = h_O(X)h_O(x)$

pro $X \in N \cup \Sigma_I \cup \Sigma_O$ a $x \in (N \cup \Sigma_I \cup \Sigma_O)^*$.

Na základě uvedených homomorfismů h_I a h_O lze zavést překlad $P(G)$ definovaný gramatikou G jako $P(G) = \{(h_I(w), h_O(w)); w \in L(G)\}$

Nechť $w \in L(G)$. Pak $x \in h_I(w)$ se nazývá *vstupní řetězec* a $y \in h_O(w)$ *výstupní řetězec*. Řetězec w je *charakteristická věta* dvojice (x, y).

V uvedeném překladu však může pro jeden vstupní řetězec existovat více výstupních řetězců, což je v kompilátorech nepřípustné. Proto zavedeme pojem *jednoznačný překlad*, $P \in \Sigma_I^* \times \Sigma_O^*$, pro nějž platí, že libovolné vstupní větě x odpovídá právě jedna výstupní věta y, tj. $y = P(x)$.

Jednoznačný překlad je definován *jednoznačnou gramatikou*.

Dalším stupněm v jednoznačnosti překladových gramatik je sémanticky jednoznačná gramatika.

Překladovou gramatiku $G = (V, W, \Sigma_I \cup \Sigma_O, P, S)$ nazýváme *sémanticky jednoznačná*, pokud splňuje podmínku:

jestliže $(p, A) \to (q, x) \in P \wedge (p, A) \to (q, y) \in P \wedge x \neq y$, pak $h_I(x) \neq h_I(y)$.

Mějme překladovou gramatiku $G = (V, W, \Sigma_I \cup \Sigma_O, P, S)$, potom *vstupní gramatikou* nazveme stavovou gramatiku $G_I = (V, W, \Sigma_I, P_I, S)$ $P_I = \{(p, A) \to (q, h_I(x)); (p, A) \to (q, x) \in P,\ p, q \in W\}$. Obdobně můžeme nazvat *výstupní gramatikou* stavovou gramatiku $G_O = (V, W, \Sigma_O, P_O, S)$ $P_O = \{(p, A) \to (q, h_O(x)); (p, A) \to (q, x) \in P,\ p, q \in W\}$.

Překladové gramatiky G_1 a G_2 nazýváme *ekvivalentní*, pokud $P(G_1) = P(G_2)$.

4.2.5 Vztah jednoduchých syntaxí řízených překladových schémat a překladových gramatik

Věta: Nechť T je jednoduché syntaxí řízené překladové schéma. Potom existuje překladová gramatika G taková, že

$$P(G) = P(T).$$

Důkaz: Nechť $T = (V, W, \Sigma_I, \Sigma_O, R, S)$ je jednoduché syntaxí řízené překladové schéma. Potom můžeme definovat gramatiku

$$G = (V, W, \Sigma_I \cup \Sigma_O, P, S),$$

kde $P = \{(p, A) \to (q, x_0 y_0 B_1 x_1 y_1 \dots B_n x_n y_n) \mid (p, A) \to (x_0 B_1 x_1 \dots B_n x_n, y_0 B_1 y_1 \dots B_n y_n) \in R, xi \in \Sigma I*, yi \in \Sigma O*, Bj \in N, 0 \leq i \leq n, 1 \leq j \leq n$

4.2.6 Syntaktická analýza shora dolů

Syntaktická analýza shora dolů je prvním ze dvou základních systematických pohledů na realizaci syntaktické analýzy. Jak již bylo uvedeno dříve (kapitola 4.2), cílem je nalézt posloupnost očíslovaných pravidel použitých při levé derivaci vstupní věty, což odpovídá konstrukci derivačního stromu od kořene směrem dolů k vstupní větě. Nejprve si ukážeme, jak pro libovolnou stavovou gramatiku sestrojíme jednoduché syntaxí řízené překladové schéma provádějící syntaktickou analýzu shora dolů.

Věta: Nechť G je stavová gramatika. Potom existuje jednoduché syntaxí řízené překladové schéma T takové, že

$$P(T) = \{(x, v); x \in L(G), v \; je \; levý \; rozbor \; x\}.$$

Důkaz: Nechť $G = (V, W, T', P, S)$ je stavová gramatika s m pravidly ($m \geq 1$) očíslovanými $1 \dots m$.

Potom definujme jednoduché syntaxí řízené překladové schéma

$$T = (V, W, \Sigma_I, \Sigma_O, R, S),$$

$\Sigma_I = T'$,

$\Sigma_O = \{1, \dots, m\}$,

a pravidla z množiny R definujeme:

pokud $(p,\ A) \to (q,\ x) \in P$ je pravidlo s číslem i, $i \in \{1, \dots, m\}$, $A \in N$, $p, q \in W$, $x \in (N \cup T)^*$, potom $(p, A) \to (q,\ x, i\ @\ x') \in R$, kde x' je slovo x, z nějž byly eliminovány všechny terminální symboly a $@ \in \bigcup_{i=0}^{n-1} (N - \{A\})^i$.

Důkaz $P(T) = \{(x, v); x \in L(G), v \; je \; levý \; rozbor \; x\}$ lze provést indukcí, viz [1].

Příklad 4.2.6.1: Mějme stavovou gramatiku $G = (V, W, T, P, S)$.

$V = \{S, A, A', a, +, =, .\}$

$W = \{s, p, q, r, f\}$

$T = \{a, +, =, .\}$

$S = \{S\}$

$P = \{ (s,\ S) \rightarrow (p,\ AA'),$ [1]

$(p,\ A) \rightarrow (q,\ a.A.a),$ [2]

$(q,\ A') \rightarrow (p,\ A'.a),$ [3]

$(p,\ A) \rightarrow (r,\ a = a),$ [4]

$(r,\ A') \rightarrow (f,\ +a)\}$ [5]

Dále uvažujme vstupní větu $a.a.a = a.a.a + a.a.a$. Tuto větu můžeme v uvedené gramatice G odvodit levou derivací (viz 2.3.2):

$(s,\ S) \Rightarrow (p,\ AA')$ [1]

$\Rightarrow (q,\ a.A.aA')$ [2]

$\Rightarrow (p,\ a.A.aA'.a)$ [3]

$\Rightarrow (q,\ a.a.A.a.aA'.a)$ [2]

$\Rightarrow (p,\ a.a.A.a.aA'.a.a)$ [3]

$\Rightarrow (r,\ a.a.a = a.a.aA'.a.a)$ [4]

$\Rightarrow (f,\ a.a.a = a.a.a + a.a.a)$ [5]

Levý rozbor uvedené věty je tedy 1 2 3 2 3 4 5.

Podle výše uvedeného algoritmu můžeme sestrojit jednoduché syntaxí řízené překladové schéma

$$T = (V, W, \Sigma_I, \Sigma_O, R, S),$$

$V = \{S, A, A', a, +, =, .\}$

$W = \{s, p, q, r, f\}$

$\Sigma_I = \{a, +, =, .\}$

$\Sigma_O = \{1, 2, 3, 4, 5\}$

$R = \{ (s,\ S) \rightarrow (p,\ AA',\ 1\ @\ AA'),$

$(p,\ A) \rightarrow (q,\ a.A.a,\ 2\ @\ A),$

$(q,\ A') \rightarrow (p,\ A'.a,\ 3\ @\ A'),$

$(p,\ A) \rightarrow (r,\ a = a,\ 4),$

$(r,\ A') \rightarrow (f,\ +a\ ,\ 5\ )\}$

Takto získané jednoduché syntaxí řízené překladové schéma můžeme použít na stejnou vstupní větu jako v případě stavové gramatiky, $a.a.a = a.a.a + a.a.a$. Výsledná derivace vypadá následovně:

$$
\begin{aligned}
(s,\ S,\ S) &\Rightarrow (p, \quad AA', && 1\ AA')\\
&\Rightarrow (q, \quad a.A.aA', && 1\ 2\ AA')\\
&\Rightarrow (p, \quad a.A.aA'.a, && 1\ 2\ 3\ AA')\\
&\Rightarrow (q, \quad a.a.A.a.aA'.a, && 1\ 2\ 3\ 2\ AA')\\
&\Rightarrow (p, \quad a.a.A.a.aA'.a.a, && 1\ 2\ 3\ 2\ 3\ AA')\\
&\Rightarrow (r, \quad a.a.a = a.a.aA'.a.a, && 1\ 2\ 3\ 2\ 3\ 4\ A')\\
&\Rightarrow (f, \quad a.a.a = a.a.a + a.a.a, && 1\ 2\ 3\ 2\ 3\ 4\ 5)
\end{aligned}
$$

Vidíme, že levý rozbor uvedené věty je 1 2 3 2 3 4 5, což odpovídá levému rozboru, který jsme získali derivací podle gramatiky G.

Věta: Nechť G je stavová gramatika a n je libovolné kladné celé číslo. Potom existuje hluboký zásobníkový převodník M takový, že

$$P_e({}_nM) = \{(x, v); x \in L(G, n), v\ je\ levý\ rozbor\ x\}.$$

Důkaz: Nechť $G = (V, W, T, P, S)$ je stavová gramatika s m pravidly ($m \geq 1$) očíslovanými $1 \ldots m$ a n je libovolné kladné celé číslo, $n \geq 1$. Bez újmy na obecnosti můžeme předpokládat, že $T \cap \{1, \ldots, m\} = \emptyset$. Dále nadefinujme homomorfismus h nad $(\{\#\} \cup V)^*$ jako $h\ (A) = A$ pro všechna $A \in \{\#\} \cup N$ a $h(a) = \varepsilon$ pro všechna $a \in V - N$.

Potom můžeme definovat hluboký zásobníkový převodník

$${}_nM = (Q, \Sigma_I, \Gamma, \Sigma_O, \delta, q_0, S, F)$$

$\Sigma_I = T$,

$\Sigma_O = \{1, \ldots, m\}$,

$N = V - T$,

$\Gamma = \{\#\} \cup V$,

$Q = \{s_0, \$\} \cup \{\langle p, u\rangle |\ p \in W, u \in prefix(N^*\{\#\}^n, n), |u| \leq n\ \}$,

$F = \{\$\}$,

$q_0 = s_0$.

Množinu δ sestrojíme následovně:

1. pro každé pravidlo $(p, S) \to (q, x) \in P,\ p, q \in W, x \in \Sigma_I^+$
   přidáme $\delta(1, s_0, \varepsilon, S) = (\langle p, S\rangle, S, \varepsilon)$
2. jestliže $(p, A) \to (q, x) \in P$ je pravidlo s číslem i, $i \in \{1, \ldots, m\}$ a $\langle p, uAv\rangle \in Q,\ p, q \in W,\ A \in N, u \in N^*, v \in N^*\{\#\}^*, |uAv| \leq n, p \notin\ states_G(u)$
   potom $(\langle q, prefix(u\ h(x)\ v, n)\rangle, x, i) \in \delta(|uA|, \langle p, uAv\rangle, \varepsilon, A)$
3. jestliže $A \in N, p \in W, u \in N^*, v \in \{\#\}^*, |uv| \leq n - 1,\ p \notin\ states_G(u)$,
   potom $(\langle p, uAv\rangle, A, \varepsilon) \in \delta(|uA|, \langle p, uv\rangle, \varepsilon, A)$
   a $(\langle p, uv\#\rangle, \#, \varepsilon) \in \delta(|uA|, \langle p, uv\rangle, \varepsilon, \#)$
4. $\delta(1, \langle q, \#^n\rangle, \#, \varepsilon) =\ \{(\$, \#, \varepsilon)\}$ pro všechna $q \in W$

Příklad 4.2.6.2: Mějme stavovou gramatiku G z příkladu 4.2.6.1.

Podle uvedeného algoritmu můžeme definovat hluboký zásobníkový převodník

$$_nM = (Q, \Sigma_I, \Gamma, \Sigma_O, \delta, q_0, S, F)$$

$m = 5, n = 2,$

$\Sigma_I = \{a, +, =, \ ,\}, \Sigma_O = \{1, \dots, m\},$

$\Gamma = \{\#, S, A, A', a, +, =, \ .\},$

$N = \{S, A, A'\},$

$Q = \{s_0, \$\} \cup \{\langle z\rangle,\ \langle z, S\rangle,\ \langle z, A\rangle,\ \langle z, A'\rangle,\ \langle z, \#\rangle,\ \langle z, \#\#\rangle,\ \langle z, SS\rangle,\ \langle z, SA\rangle,\ \langle z, SA'\rangle,\ \langle z, S\#\rangle,$
$\langle z, AS\rangle,\ \langle z, AA\rangle,\ \langle z, AA'\rangle,\ \langle z, A\#\rangle,\ \langle z, A'S\rangle,\ \langle z, A'A\rangle,\ \langle z, A'A'\rangle,\ \langle z, A'\#\rangle \mid z \in \{s, p, q, r, f\}\},$

$q_0 = s_0,$

$F = \{\$\},$

1,

$\delta(1, s_0, \varepsilon, S) = (\langle s, S\rangle, S, \varepsilon)$

2,

$\delta(2, \langle s, AS\rangle, \varepsilon, S) = (\langle p, AA\rangle,\ AA',\ 1)$
$\delta(1, \langle s, SA\rangle, \varepsilon, S) = (\langle p, AA'\rangle,\ AA',\ 1)$
$\delta(1, \langle s, S\#\rangle, \varepsilon, S) = (\langle p, AA'\rangle,\ AA',\ 1)$

$\delta(2, \langle s, A'S\rangle, \varepsilon, S) = (\langle p, A'A\rangle,\ AA',\ 1)$
$\delta(1, \langle s, SA'\rangle, \varepsilon, S) = (\langle p, AA'\rangle,\ AA',\ 1)$
$\delta(1, \langle s, S\rangle, \varepsilon, S) = (\langle p, AA'\rangle,\ AA',\ 1)$

$\delta(2, \langle p, SA\rangle, \varepsilon, A) = \{(\langle q, SA\rangle,\ a.A.a,\ 2),\ (\langle r, S\rangle,\ a = a,\ 4)\}$
$\delta(2, \langle p, A'A\rangle, \varepsilon, A) = \{(\langle q, A'A\rangle,\ a.A.a,\ 2),\ (\langle r, A'\rangle,\ a = a,\ 4)\}$
$\delta(1, \langle p, AS\rangle, \varepsilon, A) = \{(\langle q, AS\rangle,\ a.A.a,\ 2),\ (\langle r, S\rangle,\ a = a,\ 4)\}$
$\delta(1, \langle p, AA'\rangle, \varepsilon, A) = \{(\langle q, AA'\rangle,\ a.A.a,\ 2),\ (\langle r, A'\rangle,\ a = a,\ 4)\}$
$\delta(1, \langle p, A\#\rangle, \varepsilon, A) = \{(\langle q, A\#\rangle,\ a.A.a,\ 2),\ (\langle r, \#\rangle,\ a = a,\ 4)\}$
$\delta(1, \langle p, A\rangle, \varepsilon, A) = \{(\langle q, A\rangle,\ a.A.a,\ 2),\ (\langle r, \#\rangle,\ a = a,\ 4)\}$

$\delta(2, \langle q, SA'\rangle, \varepsilon, A') = (\langle p, SA'\rangle,\ A'.a,\ 3)$
$\delta(1, \langle q, A'S\rangle, \varepsilon, A') = (\langle p, A'S\rangle,\ A'.a,\ 3)$
$\delta(1, \langle q, A'\#\rangle, \varepsilon, A') = (\langle p, A'\#\rangle,\ A'.a,\ 3)$

$\delta(2, \langle q, AA'\rangle, \varepsilon, A') = (\langle p, AA'\rangle,\ A'.a,\ 3)$
$\delta(1, \langle q, A'A\rangle, \varepsilon, A') = (\langle p, A'A\rangle,\ A'.a,\ 3)$
$\delta(1, \langle q, A'\rangle, \varepsilon, A') = (\langle p, A'\rangle,\ A'.a,\ 3)$

$\delta(2, \langle r, SA'\rangle, \varepsilon, A') = (\langle f, S\rangle,\ +a,\ 5)$
$\delta(1, \langle r, A'S\rangle, \varepsilon, A') = (\langle f, S\rangle,\ +a,\ 5)$
$\delta(1, \langle r, A'\#\rangle, \varepsilon, A') = (\langle f, \#\rangle,\ +a,\ 5)$

$\delta(2, \langle r, AA'\rangle, \varepsilon, A') = (\langle f, A\rangle,\ +a,\ 5)$
$\delta(1, \langle r, A'A\rangle, \varepsilon, A') = (\langle f, A\rangle,\ +a,\ 5)$
$\delta(1, \langle r, A'\rangle, \varepsilon, A') = (\langle f, \varepsilon\rangle,\ +a,\ 5)$

3, pro všechna $X \in N$

$$u = \varepsilon, v = \ \varepsilon, |uA| = 1$$

$\delta(1, \langle s, \varepsilon\rangle, \varepsilon, X)$	$= (\langle s, X\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle s, \varepsilon\rangle, \varepsilon, \#)$	$= (\langle s, \#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle p, \varepsilon\rangle, \varepsilon, X)$	$= (\langle p, X\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle p, \varepsilon\rangle, \varepsilon, \#)$	$= (\langle p, \#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle q, \varepsilon\rangle, \varepsilon, X)$	$= (\langle q, X\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle q, \varepsilon\rangle, \varepsilon, \#)$	$= (\langle q, \#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle r, \varepsilon\rangle, \varepsilon, X)$	$= (\langle r, X\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle r, \varepsilon\rangle, \varepsilon, \#)$	$= (\langle r, \#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle f, \varepsilon\rangle, \varepsilon, X)$	$= (\langle f, X\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle f, \varepsilon\rangle, \varepsilon, \#)$	$= (\langle f, \#\rangle,\ \#,\ \varepsilon)$

$$u = \varepsilon, v = \#, |uA| = 1$$

$\delta(1, \langle s, \#\rangle, \varepsilon, X)$	$= (\langle s, S\#\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle s, \#\rangle, \varepsilon, \#)$	$= (\langle s, \#\#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle p, \#\rangle, \varepsilon, X)$	$= (\langle p, S\#\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle p, \#\rangle, \varepsilon, \#)$	$= (\langle p, \#\#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle q, \#\rangle, \varepsilon, X)$	$= (\langle q, S\#\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle q, \#\rangle, \varepsilon, \#)$	$= (\langle q, \#\#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle r, \#\rangle, \varepsilon, X)$	$= (\langle r, S\#\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle r, \#\rangle, \varepsilon, \#)$	$= (\langle r, \#\#\rangle,\ \#,\ \varepsilon)$
$\delta(1, \langle f, \#\rangle, \varepsilon, X)$	$= (\langle f, S\#\rangle,\ X,\ \varepsilon)$	$\delta(1, \langle f, \#\rangle, \varepsilon, \#)$	$= (\langle f, \#\#\rangle,\ \#,\ \varepsilon)$

$$u \neq \varepsilon, v \neq \varepsilon, |uA| = 2$$

$\delta(2, \langle s, A\rangle, \varepsilon, X)$	$= (\langle s, AX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle s, A\rangle, \varepsilon, \#)$	$= (\langle s, A\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle s, A'\rangle, \varepsilon, X)$	$= (\langle s, A'X\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle s, A'\rangle, \varepsilon, \#)$	$= (\langle s, A'\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle p, S\rangle, \varepsilon, X)$	$= (\langle p, SX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle p, S\rangle, \varepsilon, \#)$	$= (\langle p, S\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle p, A'\rangle, \varepsilon, X)$	$= (\langle p, A'X\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle p, A'\rangle, \varepsilon, \#)$	$= (\langle p, A'\#\rangle, \#,\ \varepsilon)$
$\delta(2, \langle q, S\rangle, \varepsilon, X)$	$= (\langle q, SX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle q, S\rangle, \varepsilon, \#)$	$= (\langle q, S\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle q, A\rangle, \varepsilon, X)$	$= (\langle q, AX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle q, A\rangle, \varepsilon, \#)$	$= (\langle q, A\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle r, S\rangle, \varepsilon, X)$	$= (\langle r, SX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle r, S\rangle, \varepsilon, \#)$	$= (\langle r, S\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle r, A\rangle, \varepsilon, X)$	$= (\langle r, AX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle r, A\rangle, \varepsilon, \#)$	$= (\langle r, A\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle f, S\rangle, \varepsilon, X)$	$= (\langle f, SX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle f, S\rangle, \varepsilon, \#)$	$= (\langle f, S\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle f, A\rangle, \varepsilon, X)$	$= (\langle f, AX\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle f, A\rangle, \varepsilon, \#)$	$= (\langle f, A\#\rangle,\ \#,\ \varepsilon)$
$\delta(2, \langle f, A'\rangle, \varepsilon, X)$	$= (\langle f, A'X\rangle,\ X,\ \varepsilon)$	$\delta(2, \langle f, A'\rangle, \varepsilon, \#)$	$= (\langle f, A'\#\rangle,\ \#,\ \varepsilon)$

4, pro všechna $z \in \{s, p, q, r, f\}$

$\delta(1, \langle z, \#\#\rangle, \#, \varepsilon) \quad = (\$,\ \#,\ \varepsilon)$

Pro vstupní řetězec $a.a.a\ =\ a.a.a\ +\ a.a.a$ provede tento převodník tuto posloupnost kroků:

$(s,\ a.a.a\ =\ a.a.a\ +\ a.a.a,\ S\#,\ \varepsilon)$

$\Longrightarrow (\langle s, S\rangle,\quad a.a.a\ =\ a.a.a + a.a.a,\quad S\#,\qquad \varepsilon)$

$\Longrightarrow (\langle p, AA'\rangle,\ a.a.a\ =\ a.a.a + a.a.a,\quad AA'\#,\qquad 1)$

$\Longrightarrow (\langle q, AA'\rangle,\ a.a.a\ =\ a.a.a + a.a.a,\quad a.A.aA'\#,\qquad 1\ 2)$

$\Longrightarrow (\langle q, AA'\rangle,\ .a.a\ =\ a.a.a + a.a.a,\quad .A.aA'\#,\qquad 1\ 2)$

$\Longrightarrow (\langle q, AA'\rangle,\ a.a\ =\ a.a.a + a.a.a,\quad A.aA'\#,\qquad 1\ 2)$

$\Rightarrow (\langle p, AA' \rangle,\ a.a\ =\ a.a.a + a.a.a,\quad A.aA'.a\#,\quad 1\ 2\ 3)$

$\Rightarrow (\langle q, AA' \rangle,\ a.a\ =\ a.a.a + a.a.a,\quad a.A.a.aA'.a\#,\quad 1\ 2\ 3\ 2)$

$\Rightarrow (\langle q, AA' \rangle,\ .a\ =\ a.a.a + a.a.a,\quad .A.a.aA'.a\#,\quad 1\ 2\ 3\ 2)$

$\Rightarrow (\langle q, AA' \rangle,\ a\ =\ a.a.a + a.a.a,\quad A.a.aA'.a\#,\quad 1\ 2\ 3\ 2)$

$\Rightarrow (\langle p, AA' \rangle,\ a\ =\ a.a.a + a.a.a,\quad A.a.aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3)$

$\Rightarrow (\langle r, A' \rangle,\ a\ =\ a.a.a + a.a.a,\quad a = a.a.aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ =\ a.a.a + a.a.a,\quad = a.a.aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ a.a.a + a.a.a,\quad a.a.aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ .a.a + a.a.a,\quad .a.aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ a.a + a.a.a,\quad a.aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ .a + a.a.a,\quad .aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ a + a.a.a,\quad aA'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A' \rangle,\ +a.a.a,\quad A'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle r, A'\# \rangle,\ +a.a.a,\quad A'.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4)$

$\Rightarrow (\langle f, \# \rangle,\ +a.a.a,\quad +a.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ a.a.a,\quad a.a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ .a.a,\quad .a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ a.a,\quad a.a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ .a,\quad .a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ a,\quad a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ a,\quad a\#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \# \rangle,\ \varepsilon,\quad \#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\langle f, \#\# \rangle,\ \varepsilon,\quad \#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

$\Rightarrow (\$,\ \varepsilon,\quad \#,\quad 1\ 2\ 3\ 2\ 3\ 4\ 5)$

Vidíme, že levý rozbor uvedené věty je 1 2 3 2 3 4 5 a je shodný s levým rozborem, který jsme získali derivací podle gramatiky G.

4.2.7 Syntaktická analýza zdola nahoru

Věta: Nechť G je stavová gramatika. Potom existuje jednoduché překladové schéma T takové, že

$$P(T) = \{(x, v); x \in L(G), v\ je\ pravý\ rozbor\ x\}.$$

Důkaz: Nechť $G = (V, W, T', P, S)$ je stavová gramatika s m pravidly ($m \geq 1$) očíslovanými $1 \ldots m$.

Potom definujme jednoduché syntaxí řízené překladové schéma

$$T = (V, W, \Sigma_I, \Sigma_O, R, S),$$

$\Sigma_I = T'$,

$\Sigma_O = \{1, \ldots, m\}$,

a pravidla z množiny R definujeme:

pokud $pA \to qx \in P$ je pravidlo s číslem i, $i \in \{1, \ldots, m\}$, $A \in N$, $p, q \in W$, $x \in (N \cup T)^*$, potom $pA \to qx, x'@i \in R$, kde x' je slovo x, z nějž byly eliminovány všechny terminální symboly a $@ \in \bigcup_{i=0}^{n-1}(N - \{A\})^i$.

Důkaz $P(T) = \{(x, v); x \in L(G), v\ je\ pravý\ rozbor\ x\}$ lze provést indukcí, viz [1].

Příklad 4.2.7.1: Mějme stavovou gramatiku $G = (V, W, T, P, S)$.

$V = \{S, A, A', B, B'a, -, =, .\}$

$W = \{s, p, q, r, f\}$

$T = \{a, -, =, .\}$

$S = \{S\}$

$P = \{ (s, S) \to (p, AA'),$ [1]

$(p, A) \to (q, a.A.a),$ [2]

$(q, A') \to (p, A'.a),$ [3]

$(p, A) \to (r, BB'),$ [4]

$(r, A') \to (f, -a)$ [5]

$(f, B) \to (f, a)\}$ [6]

$(f, B') \to (f, = a)\}$ [7]

Dále uvažujme vstupní větu $a.a = a.a - a.a$. Tuto větu můžeme v uvedené gramatice G odvodit pomocí pravé derivace (viz 2.3.2) :

$(s, S) \Rightarrow (p, AA')$ [1]

$\Rightarrow (q, a.A.aA')$ [2]

$\Rightarrow (p, a.A.aA'.a)$ [3]

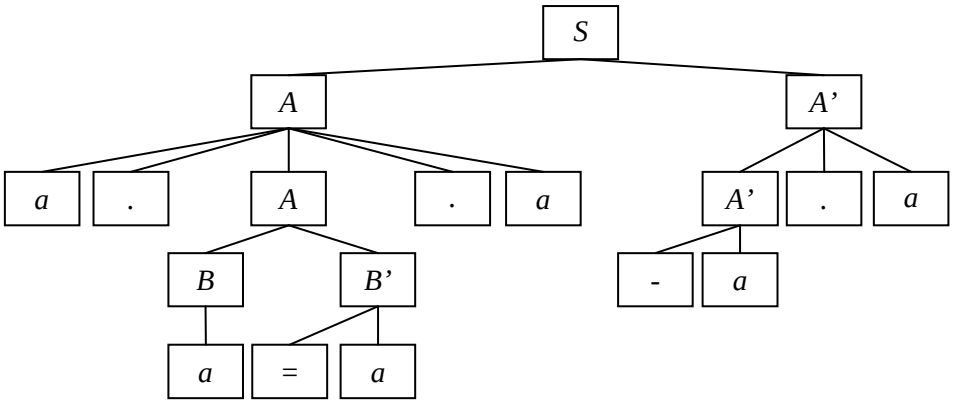
$\Rightarrow (r, a.BB'.aA'.a)$ [4]

$\Rightarrow (f, a.BB'.a - a.a)$ [5]

$\Rightarrow (f, a.B = a.a - a.a)$ [7]

$\Rightarrow (f, a.a = a.a - a.a)$ [6]

Pravý rozbor uvedené věty je tedy 6 7 5 4 3 2 1.



Obrázek 6: obrázek k příkladu 4.2.7.1

Na základě uvedeného algoritmu můžeme sestrojit jednoduché syntaxí řízené překladové schéma

$$T = (V, W, \Sigma_I, \Sigma_O, R, S),$$

$V = \{S, A, A', B, B', a, -, =, .\}$

$W = \{s, p, q, r, f\}$

$\Sigma_I = \{a, -, =, .\}$

$\Sigma_O = \{1, 2, 3, 4, 5\}$

$R = \{ (s,\ S) \rightarrow (p,\ AA',\ AA'\ @\ 1\ ),$

$(p,\ A) \rightarrow (q,\ a.A.a,\ A\ @\ 2),$

$(q,\ A') \rightarrow (p,\ A'.a,\ A'\ @\ 3),$

$(p,\ A) \rightarrow (r,\ BB',\ 4),$

$(r,\ A') \rightarrow (f, -a\ ,\ 5\ )$

$(f,\ B) \rightarrow (f,\ a\ ,\ 6)$

$(f,\ B') \rightarrow (f,\ =a\ ,\ 7)\}$

$(s,\ S,\ S) \Rightarrow (p,\ \ AA',\ \ AA'\ 1)$

$\Rightarrow (q,\ \ a.A.aA',\ \ AA'2\ 1)$

$\Rightarrow (p,\ \ a.A.aA'.a,\ \ AA'\ 3\ 2\ 1)$

$\Rightarrow (r,\ \ a.BB'.aA'.a,\ \ BB'A'\ 4\ 3\ 2\ 1)$

$\Rightarrow (f,\ \ a.BB'.a - a.a,\ \ BB'\ 5\ 4\ 3\ 2\ 1)$

$\Rightarrow (f,\ \ a.B = a.a - a.a,\ \ B\ 7\ 5\ 4\ 3\ 2\ 1)$

$\Rightarrow (f,\ \ a.a = a.a - a.a,\ \ 6\ 7\ 5\ 4\ 3\ 2\ 1)$

Vidíme, že pravý rozbor je 6 7 5 4 3 2 1.

Věta: Nechť G je stavová gramatika. Potom existuje hluboký zásobníkový převodník M, expandující nejhlubší možný neterminál, takový, že

$$P_e(M) = \{(x, v); x \in L(G), v\ je\ pravý\ rozbor\ x\}.$$

Důkaz: Nechť $G = (V, W, T, P, S)$ je stavová gramatika s m pravidly ($m \geq 1$) očíslovanými $1 \ldots m$ a n je libovolné kladné celé číslo, $n \geq 1$. Bez újmy na obecnosti můžeme předpokládat, že $T \cap \{1, \ldots, m\} = \emptyset$. Dále nadefinujme homomorfismus h nad $(\{\#\} \cup V)^*$ jako $h\ (A) = A$ pro všechna $A \in \{\#\} \cup N$ a $h(a) = \varepsilon$ pro všechna $a \in V - N$.

Potom můžeme definovat hluboký zásobníkový převodník, expandující nejhlubší možný neterminál,

$$_nM = (Q, \Sigma_I, \Gamma, \Sigma_O, \delta, q_0, S, F)$$

$\Sigma_I = T$,

$\Sigma_O = \{1, \ldots, m\}$,

$N = V - T$,

$\Gamma = \{\#\} \cup V$,

$Q = \{s, \$\} \cup \{\langle p, u\rangle |\ p \in W, u \in prefix(N^*\{\#\}^n, n), |u| \leq n\ \}$,

$F = \{\$\}$,

$q_0 = s$.

Množinu δ sestrojíme následovně:

1. pro každé pravidlo $(p, S) \to (q, x) \in P,\ p, q \in W, x \in \Sigma_I^+$
   přidáme $\delta(1, p, \varepsilon, S) = (\langle p, S\rangle, S, \varepsilon)$
2. jestliže $(p, A) \to (q, x) \in P$ je pravidlo s číslem i, $i \in \{1, \ldots, m\}$ a $\langle p, uAv\rangle \in Q,\ p, q \in W,\ A \in N, u \in N^*, v \in N^*\{\#\}^*, |uAv| \leq n, p \notin\ states_G(u)$
   potom $(\langle q, prefix(u\ h(x)\ v, n)\rangle, x, i) \in \delta(|uA|, \langle p, uAv\rangle, \varepsilon, A)$
3. jestliže $A \in N, p \in W, u \in N^*, v \in \{\#\}^*, |uv| \leq n - 1,\ p \notin\ states_G(u)$,
   potom $(\langle p, uAv\rangle, A, \varepsilon) \in \delta(|uA|, \langle p, uv\rangle, \varepsilon, A)$
   a $(\langle p, uv\#\rangle, \#, \varepsilon) \in \delta(|uA|, \langle p, uv\rangle, \varepsilon, \#)$
4. $\delta(1, \langle q, \#^n\rangle, \#, \varepsilon) =\ \{(\$, \#, \varepsilon)\}$ pro všechna $q \in W$

4.3 Generování kódu

Generování kódu je další důležitou fází překladu. Jak již bylo uvedeno v předchozí kapitole, výstupem syntaktické analýzy je řetěz reprezentující derivační strom (např. levý rozbor nebo pravý rozbor). Generování kódu se zpravidla rozpadá do 3 samostatných etap – generování vnitřní formy, optimalizace a nakonec generování cílového programu.

4.3.1 Generování vnitřní formy

U této etapy se také často můžeme setkat s názvem generování mezikódu. Jejím vstupem je obecně derivační strom určující syntaktickou strukturu zdrojového programu. V praxi se však, obdobně jako je tomu u lexikálního analyzátoru, setkávám se situací, kdy je generátor vnitřní formy programu velice úzce provázán se syntaktickým analyzátorem. V takovém případě můžeme říci, že řetěz lexikálních symbolů je po syntaktické kontrole přímo přeložen na řetězec, který tuto vnitřní formu reprezentuje.

Hlavním důvodem generování vnitřní formy především její nezávislost na konkrétním strojovém jazyce. To má za následek zjednodušení přenosu kompilátoru mezi různými operačními systémy nebo typy počítačů. Dalším důvodem může být také možnost optimalizovat daný program. Využití mezikódu nalezneme také v interpretech.

4.3.1.1 3adresný kód (3AC)

Typickým zástupcem zápisu mezikódu je například 3adresný (3adresový) kód. Tento kód se skládá z posloupnosti instrukcí s nejvýše třemi operandy. Symbolicky můžeme tuto skutečnost zapsat jako:

$$(operátor, operand\ 1, operand\ 2, výsledek).$$

Pro lepší pochopení je vhodnější si použití ukážeme na příkladě.

Příklad:

zdrojový kód	3AC
$d = a$	$(=, a, , d)$
$c = a * b$	$(*, a, b, c)$

4.3.2 Optimalizace

Etapa optimalizace kódu se snaží vylepšit mezikód s cílem získání rychlejšího výsledku či kratšího strojového kódu. Optimalizace mohou zefektivnit kód nejen na obecné úrovni (bez ohledu na konkrétní strojový jazyk), ale také mohou upravit mezikód pro efektivnější převod do cílového strojového jazyka.

Cílem optimalizací zpravidla není nalezení nejlepší varianty. V některých situacích může dojít naopak ke zhoršení vlastností původního kódu.

Optimalizace jsou zcela volitelné a překladače se v jejich použití velmi liší.

4.3.3 Generování cílového programu

Tato poslední etapa se stará o vytvoření výsledného strojového kódu nebo programu v jazyce asembler. Mezi základní kroky patří přidělení místa v paměti, případně registru, jednotlivým proměnným a následné převedení instrukcí mezikódu na odpovídající strojové instrukce.

5 Návrh aplikace

Za praktickou část této práce byl po několika konzultacích s vedoucím práce, prof. Alexandrem Medunou, zvolen interpret jednoduchého programovacího jazyka.

5.1 Návrh programovacího jazyka

5.1.1 Obecné vlastnosti a datové typy

Vzorem pro návrh jazyka se stal programovací jazyk Basic. Navrhovaný jazyk je netypovaný a nerozlišuje velká a malá písmena (není case-sensitive). V netypovaných jazycích nemají jednotlivé proměnné předem určený typ, a tedy mohou uchovávat hodnotu libovolného typu.

Identifikátor je posloupnost začínající jedním písmenem volitelně následovaným libovolnou posloupností písmen, číslic a znaku _. Pomocí regulárního výrazu $[a-z][a-z0-9\_]*$.

Literály (konstanty) mohou být tří typů:

- Celočíselný literál definujeme jako posloupnost číslic, $[0-9]+$
- Desetinný literal se skládá ze dvou posloupností číslic oddělených tečkou, $[0-9]+\.[0-9]+$
- Řetězcový literál je posloupnost znaků (s ASCII hodnotou vyšší než 20_H) ohraničená uvozovkami. Pro případ potřeby je možné použít i speciální dvojice znaků, tzv. escape sekvence:
  - \n konec řádku
  - \“ znak "
  - \\ znak \

Komentáře do konce řádku se nacházejí za symbolem ‘ nebo za klíčovým slovem rem. Blokové komentáře jsou ohraničeny znaky ‘* na začátku bloku a znaky *’ na konci bloku.

5.1.2 Příkazy a návěští

Program je tvořen neprázdnou posloupností příkazů a návěští v libovolném pořadí. Příkaz je ukončen vždy středníkem (;). Návěští je definováno jako identifikátor ukončený dvojtečkou (:). Názvy identifikátorů a návěští nesmí být shodné.

Příkazy:

- Definice a redefinice proměnné

$$< identifikátor > = < výraz >;$$

  Proměnná vlevo nemusí být předem definována. Výraz musí mít známou hodnotu.

- Skoky na předem definované návěští (zde není následováno dvojtečkou)

$$\text{goto } < návěští >;$$

- Tisk libovolné proměnné na výstup

$$\text{print } < identifikátor >;$$

- Čtení ze vstupu do proměnné (nepředpokládá se chyba)

$$\text{input } < identifikátor >;$$

- Podmíněné příkazy

$$\text{on} < výraz > \text{ goto } < návěští >;$$

5.1.3 Výrazy

Výrazy mohou být tvořeny konstantami, předem definovanými proměnnými, speciálním oddělovačem – čárkou $(,)$ a binárními operátory pro číselné proměnné a konstanty: $+, -, *, /,$ $\%\ (modulo), <, >, ==, !=, <=, >=$. Výrazy mohou být pouze jednoduchého typu – s jediným operátorem.

Operace porovnávání mají výslednou hodnotu 1 při splnění podmínky, jinak 0.

5.1.3.1 Oddělovač $',$

Tento oddělovač je zaveden pro demonstraci použití hlubokého zásobníkového automatu. Díky těmto automatům můžeme provádět vícenásobné přiřazení:

$$x, y, z = 1, 2, 5;$$
$$a, b, c = 1, 2, 3 + x, y, z;$$
$$d, e = 2, 3 * x, c;$$

5.1.4 Typová kontrola

Navrhovaný jazyk je netypovaný, ale interpret za běhu provádí typování všech proměnných s využitím datových typů:

- celé číslo,
- desetinné číslo,
- řetězec.

Typ proměnné se určuje podle typu výrazu, který je do proměnné přiřazen. To se řídí následujícími pravidly:

- podle tvaru literálu (konstanty)
- podle definice
- u vstupu platí stejné pravidla jako u literálů
- výsledek operace porovnávání je vždy celočíselný
- u ostatních operací
    - operandy stejného typu mají výsledek téhož typu
    - operandy různých typů mají za výsledek desetinné číslo

5.1.5 Ukázka

```
prog prvniprogram;
  start:                                rem návěští
      print “zadejte první číslo“;      ' výpis
      input a;                          '* vstup *'
  navesti1:
      print “zadejte druhé číslo“;
      input b;
      on a == b goto navesti1;          ' porovnání
      c, d = a, b + 2, 3;               '* vícenásobné přiřazení,
                                        využití hlubokého zásobníku *'
      print c;
      on c > d goto konec;
      goto start;
  konec:
```

6 Implementace

Interpret navrženého programovacího jazyka byl implementován v prostředí Microsoft Visual Studio 2008, konkrétně v programovacím jazyce C#. Z tohoto důvodu je potřeba mít pro jeho spuštění v počítači nainstalován .NET Framework 3.0.

6.1 Program

Aplikace je vzhledem k požadované funkčnosti navržena jako konzolová. Jediným povinným parametrem je cesta k souboru se zdrojovým kódem. Druhým (tentokrát volitelným) parametrem je cesta k souboru obsahujícímu syntaktická pravidla pro syntaktickou analýzu. Není-li tento parametr uveden, použije se soubor syntax.xml ze složky s programem.

```
interpret.exe  zdrojovykod.txt  [syntax.xml]
```

6.2 Popis činnosti

Vzhledem k možnému nedeterminismu pracuje aplikace ve třech fázích.

V první fázi je cílem vygenerování posloupnosti pravidel vedoucích k úspěšnému přijetí zdrojového kódu. Současně je vytvářen abstraktní syntaktický strom.

Druhá fáze přebírá výsledky první fáze a na jejich základě generuje posloupnost prováděných instrukcí v 3-adresném kódu.

Poslední fází je již samotné provádění získaných instrukcí.

6.3 Dokumentace

Na přiloženém CD naleznete kompletní dokumentaci k aplikaci:

- popis zdrojových souborů, tříd a funkcí
- popis konfiguračního souboru syntax.xml
- přehled pravidel syntaktického analyzátoru

7 Závěr

Po dlouhou dobu byly základním přístupem k syntaktické analýze programovacích jazyků bezkontextové gramatiky a zásobníkové automaty. V posledních desetiletích byla prezentována celá řada modifikací bezkontextových gramatik, jako například maticové gramatiky. Z pohledu této práce bylo však důležitější zavedení *stavových gramatik*. Stavové gramatiky mají, na rozdíl od bezkontextových, přidánu množinu stavů a, v souladu s touto změnou, upravená pravidla. Stavové gramatiky tak definují nekonečnou hierarchii jazyků mezi třídou bezkontextových a kontextových jazyků. Automatům byla věnována mnohem menší pozornost. Zavedení *hlubokých zásobníkových automatů* se pak zdá být velkým přínosem do teorie formálních jazyků. Tyto automaty rozšířením zásobníkových automatů a jsou ekvivalentním modelem ke stavovým gramatikám. Výhodou těchto automatů je možnost nahrazovat nevstupní zásobníkové symboly nejen na vrcholu, ale i hlouběji na zásobníku. Podle maximální hloubky pravidel pak tvoří nekonečnou hierarchii odpovídající hierarchii stavových gramatik.

Stavové gramatiky a hluboké zásobníkové modely společně s teorií překladu se staly základními kameny této práce. V teoretické části práce byly prezentovány jednotlivé části překladu společně se zavedením nových modelů syntaktické analýzy. Těmito modely jsou jednoduchá stavová překladová schémata, stavové překladové gramatiky a především hluboký zásobníkový převodník. Převodníky mají ve svých pravidlech, na rozdíl od automatů, možnost generovat výstupní řetězec. To je předurčuje k využití v situacích, jako je právě překlad mezi dvěma jazyky. Velká pozornost byla věnována převodům mezi uvedenými modely a ukázkám na příkladech.

V praktické části byl implementován interpret jednoduchého programovacího jazyka. Tento jazyk byl vytvořen pouze pro demonstraci vyšší síly hlubokých zásobníkových automatů (a tedy i převodníků).

Možnosti využití vlastností hlubokých zásobníkových automatů v kompilátorech není pro současné programovací jazyky, a jejich případná rozšíření, mnoho. Nejviditelnějším využitím je možnost zkrácení zápisu některých konstrukcí, jako například využití vícenásobného přiřazení, které je použité v uvedeném programovacím jazyce. Dalším rozumným využitím v kompilátorech by mohla být kontrola deklarací a typů proměnných. Mnohem lepší uplatnění by však tyto automaty mohly mít pro zpracování (syntaktickou analýzu nebo překlady) přirozeného jazyka. U českého jazyka lze za příklad takového využití uvést shodu podmětu s přísudkem. V neposlední řadě by mohly hluboké zásobníkové automaty nalézt uplatnění v jiných oblastech, například v bioinformatice.

Literatura

[1] **Aho, A. V., Ullman, J. D.:** *The Theory of Parsing, Translation and Compiling - Volume I: Parsing*, Englewood Cliffs, New Jersey, Prentice-Hall, 1972, ISBN: 0-13-914556-7.

[2] **Aho, A. V., Ullman, J. D.:** *The Theory of Parsing, Translation and Compiling - Volume II: Compiling*, Englewood Cliffs, New Jersey, Prentice-Hall, 1973, ISBN: 0-13-914564-8.

[3] **Češka, M.:** *Teoretická informatika*, učební texty, Brno, FIT VUT v Brně, 2002.

[4] **Češka, M., Vojnar, T., Smrčka, A.:** *Teoretická informatika*, studijní opora, Brno, FIT VUT v Brně, 2007.

[5] **Lewis, P. M. II, Stearns, R. E.:** Syntax-Directed Transduction, *Journal of the Association for Computing Machinery*, July 1968, Vol. 15, No. 3, pp. 465-488.

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

[7] **Meduna, A.:** Deep pushdown automata, *Acta informatica*, 2006, Vol. 98, pp. 114-124.

[8] **Meduna, A.:** *Elements of Compiler Design*, New York, Taylor & Francis, 2008, ISBN: 1-4200-6323-5.

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

[10] **Meduna, A.:** *Výstavba překladačů*, studijní opora, Brno, FIT VUT v Brně, 2006.

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

[12] **Viktorin, J.:** *Aplikace hlubokých zásobníkových automatů v kompilátorech*, diplomová práce, Brno, FIT VUT v Brně, 2009.

[13] **Wikipedia contributors:** Chomsky hierarchy, [Online] Wikipedia, The Free Encyclopedia, 1. květen 2009 [Citace: 8. květen 2009] http://en.wikipedia.org/w/index.php?title=Chomsky_hierarchy&oldid=287307866.

[14] **Wikipedia contributors:** Regular expression. [Online] Wikipedia, The Free Encyclopedia, 12. květen 2009 [Citace: 16. květen 2009] http://en.wikipedia.org/w/index.php?title=Regular_expression&oldid=289365504.

Seznam příloh

Příloha 1: Seznam obrázků

Příloha 2: CD s programem, dokumentací, zdrojovými kódy a elektronickou verzí tohoto dokumentu

Příloha 1: Seznam obrázků