InterpRISe Project
Interregional Co-operation for Promoting Innovation Strategies
EFRE Article 10(1)(b) Nr.99.02.29.007.BG
Phare Contract No. 00-0047

EUROPEAN UNION
European Regional
Development Fund

Teoretická informatika

Učební texty

Prof. RNDr. Milan Češka, CSc.

FIT VUT v Brně
2002

Tento učební text je určen pro předměty Teoretická informatika 1 a Základy překladačů dobíhajícího oboru Informatika a výpočetní technika pro předměty Formální jazyky a překladače a Teoretická informatika nového bakalářského resp. magisterského studijního programu na fakultě informačních technologií VUT v Brně.

Text obsahuje základy teorie formálních jazyků, která má důležité aplikace v řadě oblastí informačních texnologií. Základní pojmy a poznatky teorie regulárních a bezkontextových jazyků, ústící v metody jejich syntaktické analýzy, jsou prerekvizicí pro studium překladačů programovacích jazyků, pro některé metody umělé inteligence i pro studium vyčíslitelnosti a algoritmické složitosti.

Způsob výkladu, použitý v učebním textu, předpokládá základní znalosti z diskrétní matematiky.

V Brně 1.9.2002

Autor

Obsah

Kapitola 1

Úvod

Teorie formálních jazyků představuje velmi důležitou oblast informatiky (computer science). Základy novodobé historie této disciplíny položil v roce 1956 americký matematik NOAM CHOMSKY, který v souvislosti se studiem přirozených jazyků vytvořil matematický model gramatiky jazyka.

Realizace původních představ, formalizovat popis přirozeného jazyka takovým způsobem, aby mohl být automatizován překlad z jednoho přirozeného jazyka do druhého nebo aby přirozený jazyk sloužil jako prostředek komunikace člověka s počítačem, se ukázala velmi obtížnou.

Začala se však vyvíjet vlastní teorie jazyků, která nyní obsahuje bohaté výsledky v podobě matematicky dokázaných tvrzení – teorémů. Tato teorie pracuje se dvěma duálními matematickými entitami, s gramatikou a s automatem, představující abstraktní matematický stroj. Zatímco gramatika umožňuje popsat strukturu vět formálního jazyka, automat dovede tuto strukturu identifikovat.

Poznatky teorie formálních jazyků mají význam pro mnohé oblasti informačních technologií. Dodávají algoritmy, jež jsou podkladem pro konstrukci reálných automatů zpracovávající informaci ve tvaru vět formálního jazyka. Stanovují však také možnosti a omezení algoritmických postupů řešení problémů; odhalují problémy, které jsou algoritmicky nerozhodnutelné, tj. problémy, jejichž řešení nelze dosáhnout v konečném čase.

Teorie formálních jazyků našla největší uplatnění v oblasti programovacích jazyků. Krátce po CHOMSKÉHO definici gramatiky formálního jazyka a klasifikaci formálních jazyků (CHOMSKÉHO hierarchii formálních jazyků), BACKUS a NAUER použili základních objektů gramatiky pro definici syntaxe programovacího jazyka *Algol 60* (ve tvaru formalismu, jež se nazývá Backus-Nauerova forma). Další vývoj pak přímočaře vedl k aplikacím teorie jazyků v oblasti překladačů programovacích jazyků. Stanovení principů syntaxí řízeného překladu a generátorů překladačů (programovacích systémů, které na základě formálního popisu syntaxe a sémantiky programovacího jazyka vytvoří jeho překladač) představuje kvalitativní skok při konstrukci překladačů umožňující automatizovat náročnou programátorskou práci spojenou s implementací programovacích jazyků. Teorie formálních nyní nachází také významné uplatnění v umělé inteligenci, v počítačové grafice, řídicí technice, verifikačním inženýrství a také v netechnických oborech, jako např. biologie a genetika.

Kapitola 2

Jazyky, gramatiky a jejich klasifikace

2.1 Jazyky

Jedním ze základních pojmů pro vymezení jazyka jsou pojmy *abeceda* a *řetězec*.

Definice 2.1 Abecedou rozumíme neprázdnou množinu prvků, které nazýváme *symboly abecedy*.

V některých teoretických případech je účelné pracovat i s abecedami nekonečnými, v našich aplikacích se však omezíme na abecedy *konečné*.

Příklad 2.1 Jako příklady abeced můžeme uvést *latinskou abecedu* obsahující 52 symbolů, které reprezentují velká a malá písmena, *řeckou abecedu*, dvouprvkovou *binární abecedu* reprezentovanou množinou $\{0, 1\}$ resp. $\{\text{true}, \text{false}\}$ nebo abecedy programovacích jazyků. Např. abeceda programovacího jazyka Algol 60 má 116 symbolů [**?**].

Definice 2.2 *Řetězcem* (také slovem nebo větou) nad danou abecedou rozumíme každou konečnou posloupnost symbolů abecedy. Prázdnou posloupnost symbolů, tj. posloupnost, která neobsahuje žádný symbol, nazýváme *prázdný řetězec*. Prázdný řetězec budeme označovat písmenem ϵ.

Formálně lze definovat řetězec nad abecedou Σ takto:

(1) prázdný řetězec ϵ je řetězec nad abecedou Σ,

(2) je-li x řetězec nad Σ a $a \in \Sigma$, pak xa je řetězec nad Σ,

(3) y je řetězec nad Σ, když a jen když lze y získat aplikací pravidel 1 a 2.

Příklad 2.2 Je-li $A = \{a, b, +\}$ abeceda, pak

$$\epsilon \qquad a \qquad b \qquad + \qquad aa \qquad a + b \qquad + ab$$

jsou některé řetězce nad abecedou A. Pořadí symbolů v řetězci je významné; řetězce $a + b$, $+ab$ jsou různé. Symbol b, například, je řetězec nad A, poněvadž $\epsilon b = b$.

Konvence 2.1 Budeme-li pracovat s obecnými abecedami, symboly, či řetězci, pak pro jejich značení použijeme:

- velkých řeckých písmen pro abecedy,
- malých latinských písmen $a, b, c, d, f, \ldots$ pro symboly,
- malých latinských písmen $t, u, v, w, \ldots$ pro řetězce.

Nyní uvedeme některé důležité operace nad řetězci.

Definice 2.3 Nechť x a y jsou řetězce nad abecedou Σ. *Konkatenací* (zřetězením) řetězce x s řetězcem y vznikne řetězec xy připojením řetězce y za řetězec x. Operace konkatenace je zřejmě asociativní, tj. $x(yz) = (xy)z$, ne však komutativní, $xy \neq yx$.

Příklad 2.3 Je-li $x = ab, y = ba$, pak $xy = abba, yx = baab, x\epsilon = \epsilon x = ab$, kde ϵ je prázdný řetězec.

Definice 2.4 Nechť $x = a_1 a_2 \ldots a_n$ je řetězec nad abecedou $\Sigma, a_i \in \Sigma$ pro $i = 1, \ldots, n$. *Reverzí* (zrcadlovým obrazem) řetězce x rozumíme řetězec $x^R = a_n a_{n-1} \ldots a_2 a_1$; tj. symboly řetězce x^R jsou vzhledem k řetězci x zapsány v opačném pořadí.

Příklad 2.4 Je-li $x = abbc$, pak $x^R = cbba$. Zřejmě $\epsilon^R = \epsilon$.

Definice 2.5 Nechť w je řetězec nad abecedou Σ. Řetězec z se nazývá *podřetězcem* řetězce w, jestliže existují řetězce x a y takové, že $w = xzy$. Řetězec x_1 se nazývá *prefixem* (předponou) řetězce w, jestliže existuje řetězec y_1 takový, že $w = x_1 y_1$. Analogicky, řetězec y_2 se nazývá *sufixem* (příponou) řetězce w, jestliže existuje řetězec x_2 takový, že $w = x_2 y_2$. Je-li $y_1 \neq \epsilon$, resp. $x_2 \neq \epsilon$, pak x_1 je *vlastní prefix*, resp. y_2 je *vlastní sufix* řetězce w.

Příklad 2.5 Je-li $w = abbc$, pak

$$\epsilon \qquad a \qquad ab \qquad abb \qquad abbc$$

jsou všechny prefixy řetězce w (první čtyři jsou vlastní),

$$\epsilon \qquad c \qquad bc \qquad bbc \qquad abbc$$

jsou všechny sufixy řetězce w (první čtyři jsou vlastní) a

$$a \qquad bb \qquad abb$$

jsou některé podřetězce řetězce w.

Zřejmě, prefix i sufix jsou podřetězce řetězce w, prázdný řetězec je podřetězcem, prefixem i sufixem každého řetězce.

Definice 2.6 *Délka řetězce* je nezáporné celé číslo udávající počet symbolů řetězce. Délku řetězce x značíme symbolicky $|x|$. Je-li $x = a_1 a_2 \ldots a_n$, $a_i \in \Sigma$ pro $i = 1, \ldots n$, pak $|x| = n$. Délka prázdného řetězce je nulová, tj. $|\epsilon| = 0$.

Konvence 2.2 Řetězec nebo podřetězec, který sestává právě z k výskytů symbolu a budeme symbolicky značit a^k. Např.

$$a^3 = aaa, \quad b^2 = bb, \quad a^0 = \epsilon.$$

Definice 2.7 Nechť Σ je abeceda. Označme symbolem Σ^* množinu všech řetězců nad abecedou Σ včetně řetězce prázdného, symbolem Σ^+ množinu všech řetězců nad Σ vyjma řetězce prázdného, tj. $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. Množinu L, pro níž platí $L \subseteq \Sigma^*$ (případně $L \subseteq \Sigma^+$, pokud $\epsilon \notin L$), nazýváme *jazykem* L nad abecedou Σ. Jazykem tedy může být libovolná podmnožina řetězců nad danou abecedou.

Příklad 2.6 Nechť jazyky L_1 a L_2 jsou definovány nad binární abecedou, tj. $L_1, L_2 \subseteq \{0,1\}^*$, takto:

$$\begin{aligned} L_1 &= \{0^n 1^n \mid n \geq 0\} \text{ tj.} \\ L_1 &= \{\epsilon, 01, 0011, 000111, \ldots\} \\ L_2 &= \left\{xx^R \mid x \in \{0,1\}^+\right\} \text{ tj.} \\ L_2 &= \{00, 11, 0000, 0110, 1001, 1111, \ldots\} \end{aligned}$$

Příklad 2.7 Uvažujeme abecedu programovacího jazyka Pascal. Jazyk Pascal je nekonečná množina řetězců (programů) nad jeho abecedou, které lze odvodit z grafů syntaxe jazyka Pascal (viz. [**?**]). Např. řetězec

```
program P; begin end.
```

patří do jazyka Pascal (přestože nepopisuje žádnou akci), kdežto řetězec

```
procedure S; begin a := a mod b end
```

nepatří do jazyka Pascal, protože reprezentuje pouze úsek možného programu.

Povšimněme si nyní operací, které lze definovat nad jazyky. Tyto operace lze rozdělit do dvou skupin. Jednu skupinu představují obvyklé množinové operace plynoucí ze skutečnosti, že jazyk je množina. Má tedy smysl mluvit o *sjednocení*, *průniku*, *rozdílu* a *komplementu* jazyků. Je-li např. L_1 jazyk nad abecedou Σ, pak jeho komplement (doplněk) L_2 je jazyk, jenž je dán rozdílem $L_2 = \Sigma^* - L_1$.

Druhá skupina operací respektuje specifikum množin tvořících jazyky, tj. skutečnost, že jejich prvky jsou řetězce. Z této skupiny definujeme operaci součinu a iterace jazyků.

Definice 2.8 Nechť L_1 je jazyk nad abecedou Σ_1, L_2 jazyk nad abecedou Σ_2. *Součinem* (konkatenací) jazyků L_1 a L_2 je jazyk $L_1 \cdot L_2$ nad abecedou $\Sigma_1 \cup \Sigma_2$, jenž je definován takto:

$$L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

Operace součin jazyků je definována prostřednictvím konkatenace řetězců a má stejné vlastnosti jako konkatenace řetězců – je asociativní a nekomutativní.

Příklad 2.8 Nechť:

$$\begin{aligned} P &= \{A, B, \ldots, Z, a, b, \ldots, z\}, \\ C &= \{0, 1, \ldots, 9\} \quad \text{jsou abecedy} \\ L_1 &= P, \\ L_2 &= (P \cup C)^* \quad \text{jsou jazyky} \end{aligned}$$

Součinem jazyků $L_1 \cdot L_2$ je jazyk, který obsahuje všechny identifikátory, tak jak jsou obvykle definovány v programovacích jazycích.

Prostřednictvím součinu jazyka se sebou samým (mocniny jazyka) můžeme definovat důležitou operaci nad jazykem – iteraci jazyka.

Definice 2.9 Nechť L je jazyk nad abecedou Σ. *Iteraci* L^* jazyka L a *pozitivní iteraci* L^+ jazyka L definujeme takto:

(1) $$L^0 = \{\epsilon\}$$

(2) $$L^n = L \cdot L^{n-1} \text{ pro } n \geq 1$$

(3) $$L^* = \bigcup_{n \geq 0} L^n$$

(4) $$L^+ = \bigcup_{n \geq 1} L^n$$

Věta 2.1 Je-li L jazyk, pak platí:

(1) $$L^* = L^+ \cup \{\epsilon\} \quad \text{a}$$

(2) $$L^+ = L \cdot L^* = L^* \cdot L$$

Důkaz: Tvrzení 1 plyne z definice iterace a pozitivní iterace:

$$L^+ = L^1 \cup L^2 \cup \ldots, \qquad L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \qquad \text{tj.}$$

$$L^* - L^+ = L^0 = \epsilon$$

Tvrzení 2 dostaneme provedením součinu $L \cdot L^*$ resp. $L^* \cdot L$:

$$L \cdot L^* = L \cdot (L^0 \cup L^1 \cup L^2 \cup \ldots) = L^1 \cup L^2 \cup L^3 \cup \ldots = L^+,$$

s použitím distributivního zákona $L_1(L_2 \cup L_3) = L_1L_2 \cup L_1L_3$. Podobně, s využitím ekvivalentní definice mocniny $L^n = L^{n-1}L, n \geq 0$ obdržíme $L^* \cdot L = L^+$. □

Příklad 2.9 Je-li $L_1 = \{(p\}, L_2 = \{, p\}, L_3 = \{)\}$ potom jazyk $L_1 \cdot L_2^* \cdot L_3$ obsahuje řetězce:

$$(p) \qquad (p,p) \qquad (p,p,p) \qquad (p,p,p,p)\ldots$$

2.2 Gramatika

Pojem gramatika souvisí velmi úzce s problémem reprezentace jazyka. Triviální způsob reprezentace – výčet všech vět jazyka – je nepoužitelný nejen pro jazyky nekonečné, ale prakticky i pro rozsáhlé konečné jazyky. Také obvyklé matematické prostředky (použité v předchozích příkladech) jsou použitelné pouze pro reprezentaci jazyků s velmi jednoduchou strukturou.

Gramatika, jako nejznámější prostředek pro reprezentaci jazyků, splňuje základní požadavek kladený na reprezentaci konečných i nekonečných jazyků, požadavek konečnosti reprezentace. Používá dvou konečných disjunktních abeced:

(1) množiny N neterminálních symbolů

(2) množiny Σ terminálních symbolů

Nonterminální symboly, krátce *nonterminály*, mají roli pomocných proměnných označujících určité syntaktické celky – syntaktické kategorie.

Množina *terminálních symbolů*, krátce *terminálů*, je identická s abecedou, nad níž je definován jazyk. Sjednocení obou množin, tj. $N \cup \Sigma$, nazýváme *slovníkem gramatiky.*

Konvence 2.3 Pro zápis terminálních a nonterminálních symbolů a řetězců tvořených těmito symboly budeme používat této konvence:

(1) a, b, c, d reprezentují terminální symboly

(2) A, B, C, D, S reprezentují nonterminální symboly

(3) $U, V, \ldots, Z$ reprezentují terminální nebo nonterminální výrazy

(4) $\alpha, \beta, \ldots, \omega$ reprezentují řetězce terminálních a nonterminálních symbolů

(5) $u, v, \ldots, z$ reprezentují řetězce pouze terminálních symbolů

(6) řetězec uzavřený v úhlových závorkách $\langle, \rangle$ reprezentuje nonterminální symbol (konvence používaná v BNF).

Příklad 2.10 Slovník gramatiky pro definici jazyka identifikátorů může mít tvar:

$$N = \{\langle\text{identifikátor}\rangle, \langle\text{písmeno}\rangle, \langle\text{číslice}\rangle\}$$

$$\Sigma = \{A, B, \ldots, Z, \quad a, b, \ldots, z, \quad 0, 1, \ldots, 9\}$$

Obsahuje tedy 65 symbolů.

Gramatika představuje generativní systém, ve kterém lze z jistého vyznačeného nonterminálu generovat, aplikací tzv. přepisovacích pravidel, řetězce tvořené pouze terminálními symboly. Takové řetězce reprezentují právě věty gramatikou definovaného jazyka.

Jádrem gramatiky je tak konečná množina P *přepisovacích pravidel* (nazývaných také produkce). Každé přepisovací pravidlo má tvar uspořádané dvojice (α, β) řetězců; stanovuje možnou substitucí řetězce β namísto řetězce α, který se vyskytuje jako podřetězec generovaného řetězce. Řetězec α obsahuje alespoň jeden nonterminální symbol, řetězec βje prvek množiny $(N \cup \Sigma)^*$. Formálně vyjádřeno, množina P přepisovacích pravidel je podmnožinou kartézského součinu:

$$(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

Příklad 2.11 Uvažujme, že např. dvojice (AB, CDE) je jedním z přepisovacích pravidel gramatiky a předpokládejme, že řetězec $x = FGABH$ byl získán aplikací jiných pravidel gramatiky. Aplikujeme-li nyní na řetězec x pravidlo (AB, CDE), obdržíme řetězec $y = FGCDEH$ (nahrazením podřetězce AB řetězce x řetězcem CDE). Říkáme, že jsme řetězec y odvodili (derivovali) z řetězce x podle přepisovacího pravidla (AB, CDE).

Přistoupíme nyní k úplné definici gramatiky a formální definici pojmů, jejichž prostřednictvím lze definovat jazyk generovaný gramatikou.

Definice 2.10 Gramatika G je čtveřice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina nonterminálních symbolů

- Σ je konečná množina terminálních symbolů, $N \cap \Sigma = \emptyset$
- P je konečná podmnožina kartézského součinu $(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$,
- $S \in N$ je výchozí (také počáteční) symbol gramatiky

Prvek (α, β) množiny P nazýváme *přepisovacím pravidlem* (krátce *pravidlem*) a budeme jej zapisovat ve tvaru $\alpha \to \beta$. Řetězec α resp. β nazýváme *levou* resp. *pravou stranou* přepisovacího pravidla.

Příklad 2.12

$$G = (\{A, S\}, \{0, 1\}, P, S)$$
$$P = \{S \to 0A1, 0A \to 00A1, A \to \epsilon\}$$

Příklad 2.13 Gramatika definující jazyk identifikátorů může mít tvar:

$$\begin{aligned}
G &= (N, \Sigma, P, S), \text{ kde} \\
N &= \{\langle\text{identifikátor}\rangle, \langle\text{písmeno}\rangle, \langle\text{číslice}\rangle\} \\
\Sigma &= \{A, B, \ldots, Z, \quad a, b, \ldots, z, \quad 0, 1, \ldots, 9\} \\
P &= \{\langle\text{identifikátor}\rangle \to \langle\text{písmeno}\rangle, \\
&\quad \langle\text{identifikátor}\rangle \to \langle\text{identifikátor}\rangle\langle\text{písmeno}\rangle, \\
&\quad \langle\text{identifikátor}\rangle \to \langle\text{identifikátor}\rangle\langle\text{číslice}\rangle, \\
&\quad \langle\text{písmeno}\rangle \to A, \\
&\qquad \vdots \\
&\quad \langle\text{písmeno}\rangle \to Z, \\
&\quad \langle\text{číslice}\rangle \to 0, \\
&\qquad \vdots \\
&\quad \langle\text{číslice}\rangle \to 9\} \\
S &= \langle\text{identifikátor}\rangle
\end{aligned}$$

Množina P obsahuje 65 přepisovacích pravidel.

Konvence 2.4 Obsahuje-li množina pravidel P přepisovací pravidla tvaru $\alpha \to \beta_1, \alpha \to \beta_2, \ldots, \alpha \to \beta_n$, pak pro zkrácení lze použít zápisu $\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

Definice 2.11 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť λ a μ jsou řetězce z $(N \cup \Sigma)^*$. Mezi řetězci λ a μ platí relace $\Rightarrow_G$, nazývaná *přímá derivace*, jestliže můžeme řetězce λ a μ vyjádřit ve tvaru

$$\begin{aligned}
\lambda &= \gamma\alpha\delta \\
\mu &= \gamma\beta\delta
\end{aligned}$$

kde γ a δ jsou libovolné řetězce z $(N \cup \Sigma)^*$ a $\alpha \to \beta$ je nějaké přepisovací pravidlo z P.

Platí-li mezi řetězci λ a μ relace přímé derivace, pak píšeme $\lambda \Rightarrow_G \mu$ a říkáme, že řetězec μ lze přímo generovat z řetězce λ v gramatice G. Je-li z kontextu zřejmé, že jde o derivaci v gramatice G, pak nemusíme specifikaci gramatiky pod symbolem $\Rightarrow$ uvádět.

Příklad 2.14 Uvažujme gramatiku z příkladu 2.12 a řetězce $\lambda = 000A111$ a $\mu = 0000A1111$. Položíme-li

$$\begin{aligned} \lambda &= \underbrace{00}_{\gamma}\underbrace{0A}_{\alpha}\underbrace{111}_{\delta} \\ \mu &= \underbrace{00}_{\gamma}\underbrace{00A1}_{\beta}\underbrace{111}_{\delta} \end{aligned}$$

vidíme, že platí $000A111 \Rightarrow 0000A1111$, protože $0A \to 00A1$ je pravidlem v této gramatice.

Příklad 2.15 Je-li $\alpha \to \beta$ pravidlo v gramatice G, pak v této gramatice platí $\alpha \Rightarrow \beta$, jak plyne z definice 2.11, položíme-li $\gamma = \delta = \epsilon$.

Definice 2.12 Nechť $G = (N, \Sigma, P, S)$ je gramatika a λ a μ jsou řetězce z $(N \cup \Sigma)^*$. Mezi řetězci λ a μ platí relace $\Rightarrow^+$ nazývaná *derivace*, jestliže existuje posloupnost přímých derivací $\nu_{i-1} \Rightarrow \nu_i \quad i = 1, \ldots, n, \quad n \geq 1$ taková, že platí:

$$\lambda = \nu_0 \Rightarrow \nu_1 \Rightarrow \ldots \Rightarrow \nu_{n-1} \Rightarrow \nu_n = \mu$$

Tuto posloupnost nazýváme *derivací délky n*. Platí-li $\lambda \Rightarrow^+ \mu$, pak říkáme, že řetězec μ lze *generovat* z řetězce λ v gramatice G. Relace $\Rightarrow^+$ je zřejmě tranzitivním uzávěrem relace přímé derivace $\Rightarrow$. Symbolem $\Rightarrow^n$ značíme n-tou mocninu relace $\Rightarrow$.

Příklad 2.16 V gramatice z příkladu 2.12 platí (v důsledku pravidla $0A \to 00A1$, viz příklad 2.14) relace

$$0^n A 1^n \Rightarrow 0^{n+1} A 1^{n+1} \qquad n > 0$$

a tudíž také $0A1 \Rightarrow^+ 0^n A 1^n$ pro libovolné $n > 1$.

Definice 2.13 Jestliže v gramatice G platí pro řetězce λ a μ relace $\lambda \Rightarrow^+ \mu$ nebo identita $\lambda = \mu$, pak píšeme $\lambda \Rightarrow^{\mu}_{*}$. Relace $\Rightarrow^*$ je tranzitivním a reflexivním uzávěrem relace přímé derivace $\Rightarrow$.

Příklad 2.17 Relaci $0A1 \Rightarrow^+ 0^n A 1^n, \quad n > 1$ z příkladu 2.16 lze rozšířit na relaci

$$0A1 \overset{*}{\Rightarrow} 0^n A 1^n, \quad n > 0$$

Poznámka 2.1 Dojdeme-li v posloupnosti přímých derivací k řetězci, který obsahuje pouze terminální symboly, pak již nelze aplikovat žádné přepisovací pravidlo a proces generování *končí*. Z této skutečnosti, jež vyplývá z definice pravidla, je odvozen název množiny Σ jako množiny *terminálních* symbolů.

Definice 2.14 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Řetězec $\alpha \in (N \cup \Sigma)^*$ nazýváme *větnou formou*, jestliže platí $S \Rightarrow^* \alpha$, tj. řetězec α je generovatelný z výchozího symbolu S. Větná forma, která obsahuje pouze terminální symboly, se nazývá *věta*. *Jazyk* $L(G)$, generovaný gramatikou G, je definován množinou všech vět

$$L(G) = \{w \mid S \overset{*}{\Rightarrow} w \wedge w \in \Sigma^*\}$$

Příklad 2.18 Jazyk generovaný gramatikou G z příkladu 2.12 je množina

$$L(G) = \{0^n 1^n \mid n > 0\},$$

protože platí

$$\begin{array}{lcll} S & \Rightarrow & 0A1 & \\ S & \Rightarrow^* & 0^n A 1^n \quad n > 0 & \text{(viz příklad 2.16 a 2.17)} \\ S & \Rightarrow^* & 0^n 1^n \quad n > 0 & \text{(po aplikaci pravidla } A \to \epsilon) \end{array}$$

2.3 Chomského klasifikace gramatik

Chomského klasifikace gramatik (a jazyků), známá také pod názvem Chomského hierarchie jazyků, vymezuje čtyři typy gramatik, podle tvaru přepisovacích pravidel, jež obsahuje množina přepisovacích pravidel P. Tyto typy se označují jako typ 0, typ 1. typ 2 a typ 3.

2.3.1 Typ 0

Gramatika typu 0 obsahuje pravidla v nejobecnějším tvaru, shodným s definicí 2.10 gramatiky:

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

Z tohoto důvodu se gramatiky typu 0 nazývají také gramatikami *neomezenými*.

Příklad 2.19 Příklad neomezené gramatiky:

$$\begin{array}{rcl} G & = & (\{A, B\}, \{a, b\}, P, A) \text{ s pravidly} \\ A & \to & AbB \mid a \\ AbB & \to & baB \mid BAbB \\ B & \to & b \mid \epsilon \end{array}$$

2.3.2 Typ 1

Gramatika typu 1 obsahuje pravidla tvaru:

$$\alpha A \beta \to \alpha \gamma \beta, \quad A \in N, \quad \alpha, \beta \in (N \cup \Sigma)^*, \quad \gamma \in (N \cup \Sigma)^+ \text{ nebo } S \to \epsilon$$

Gramatiky typu 1 se nazývají také *gramatikami kontextovými*, poněvadž tvar pravidla této gramatiky implikuje, že nonterminál A může být nahrazen řetězcem γ pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α.

Kontextové gramatiky neobsahují pravidla tvaru $\alpha A \beta \to \alpha \beta$, tj. nepřipouštějí, aby byl nonterminál nahrazen prázdným řetězcem. Jedinou přípustnou výjimkou je pravidlo $S \to \epsilon$ (S je výchozí symbol), které umožňuje popsat příslušnost prázdného řetězce k jazyku, který je danou gramatikou generován. V důsledku této vlastnosti pravidel gramatiky typu 1 nemůže při generování věty dojít ke zkráceni generovaných řetězců, tj. platí-li $\lambda \Rightarrow \mu$, pak $|\lambda| \leq |\mu|$ (s výjimkou $S \Rightarrow \epsilon$).

Příklad 2.20 Příklad kontextové gramatiky:

$$\begin{aligned} G &= (\{A,S\},\{0,l,c\},P,S) \quad \text{s pravidly} \\ S &\to OA1 \\ OA &\to OOA1 \qquad (\alpha = 0, \beta = \epsilon, \gamma = OA1) \\ A &\to c \end{aligned}$$

2.3.3 Typ 2

Gramatika typu 2 obsahuje pravidla tvaru:

$$A \to \gamma, \quad A \in N, \quad \gamma \in (N \cup \Sigma)^*$$

Gramatiky typu 2 se nazývají také *bezkontextovými gramatikami*, protože substituci levé strany γ pravidla za nonterminál A lze provádět bez ohledu na kontext, ve kterém je nonterminál A uložen. Na rozdíl od kontextových gramatik, bezkontextové gramatiky smí obsahovat pravidla tvaru $A \to \epsilon$. V kapitole 4 však ukážeme, že každou bezkontextovou gramatiku lze transformovat, aniž by se změnil jazyk generovaný touto gramatikou tak, že obsahuje nejvýše jedno pravidlo s prázdným řetězcem na levé straně tvaru $S \to \epsilon$. V takovém případě, stejně jako v případě kontextových gramatik, nesmí se výchozí symbol S objevit v žádné pravé straně přepisovacího pravidla gramatiky.

Příklad 2.21 Příklad bezkontextové gramatiky:

$$\begin{aligned} G &= (\{S\},\{0,1,c\},P,S) \quad \text{s pravidly} \\ S &\to 0S1 \mid c \end{aligned}$$

Poznamenejme, že jazyk generovaný touto gramatikou je stejný jako jazyk generovaný bezkontextovou gramatikou z příkladu 2.20:

$$L(G) = \{0^n c 1^n\}, \qquad n \geq 1$$

2.3.4 Typ 3

Gramatika typu 3 obsahuje pravidla tvaru:

$$A \to xB \quad \text{nebo} \quad A \to x; \quad A,B \in N, \quad x \in \Sigma^*$$

Gramatika s tímto tvarem pravidel se nazývá *pravá lineární gramatika* (jediný možný nonterminál pravé strany pravidla stojí úplně napravo). V následující kapitole ukážeme, že k uvedené gramatice lze sestrojit ekvivalentní speciální pravou lineární gramatiku s pravidly tvaru

$$A \to aB \quad \text{nebo} \quad A \to a; \quad A,B \in N, \quad x \in \Sigma \quad \text{nebo} \quad S \to \epsilon$$

Tuto gramatiku budeme nazývat regulární gramatikou, přesněji pravou regulární gramatikou. Gramatiky typu 3 se proto nazývají také *regulárními gramatikami.*

Příklad 2.22 Příklady gramatiky typu 3:

$$\begin{aligned} G &= (\{A,B\},\{a,b,c\},P,A) \quad \text{s pravidly} \\ A &\to aaB \mid ccB \\ B &\to bB \mid \epsilon \end{aligned}$$

Definice 2.15 Jazyk generovaný gramatikou typu $i, i = 0, 1, 2, 3$, nazýváme jazykem typu i. Podle anonymních názvů gramatik mluvíme také o jazycích neomezených ($i = 0$), kontextových ($i = 1$), bezkontextových ($i = 2$) a regulárních ($i = 3$).

Věta 2.2 Nechť $L_i, i = 0, 1, 2, 3$ značí třídu všech jazyků typu i. Pak platí $L_0 \supseteq L_1 \supseteq L_2 \supseteq L_3$.

Důkaz: Z definice gramatiky typu i plyne, že každá gramatika typu 1 je zároveň gramatikou typu 0 a každá gramatika typu 3 je zároveň bezkontextovou gramatikou. Inkluze $L_1 \supseteq L_2$ plyne ze skutečnosti, že každá bezkontextová gramatika může být převedena na bezkontextovou gramatiku, jež neobsahuje, s výjimkou pravidla $S \to \epsilon$ (S je výchozí symbol), žádné pravidlo s pravou stranou totožnou s prázdným řetězcem ϵ. □

K poznatkům teorie formálních jazyků patří další tvrzení, jež je zesílením věty 2.2 a jež definuje Chomského hierarchii formálních jazyků. Toto tvrzení uvádíme bez důkazu.

Věta 2.3 Nechť $L_i, i = 0, 1, 2, 3$ jsou třídy jazyků typu i. Pak platí $L_0 \supset L_1 \supset L_2 \supset L_3$.

2.4 Cvičení

Cvičení 2.4.1 Vytvořte gramatiku typu 3, která generuje identifikátory jež mohou mít maximálně 6 znaků a začínají písmenem I, J, K, L, M nebo N(celočíselné proměnné v jazyce Fortran).

Cvičení 2.4.2 Vytvořte gramatiku typu 3, která generuje čísla jazyka Pascal.

Cvičení 2.4.3 Vytvořte bezkontextovou gramatiku, která generuje všechny řetězce nul a jedniček takové, že počet nul je v každém řetězci shodný s počtem jedniček.

Cvičení 2.4.4 Ukažte, že neomezená gramatika $G = (\{S, B, C\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla $S \to SaBC, Ca \to aC, S \to aBC, BC \to CB, aB \to Ba, CB \to BC, Ba \to aB, B \to b, aC \to Ca, C \to c$ generuje věty ve kterých je počet výskytů symbolů a, b, c navzájem roven.

Kapitola 3

Jazyky typu 3 Chomského klasifikace

V této kapitole ukážeme ekvivalenci jazyků generovaných gramatikou typu 3 a regulárními množinami s jazyky přijímanými konečnými automaty.

3.1 Regulární množiny a jazyky typu 3

Definice 3.1 Nechť Σ je konečná abeceda. *Regulární množinu* nad abecedou Σ definujeme rekurzívně takto:

(1) $\emptyset$ (prázdná množina) je regulární množina nad Σ

(2) $\{\epsilon\}$ (množina obsahující pouze prázdny řetězec) je regulární množina nad Σ

(3) $\{a\}$ pro všechna $a \in \Sigma$ je regulární množina nad Σ

(4) jsou-li P a Q regulární množiny nad Σ, pak také $P \cup Q$, $P \cdot Q$ a P^* jsou regulární množiny nad Σ

(5) regulárními množinami jsou právě ty množiny, které lze získat aplikací 1–4.

Třída regulárních množin je tedy nejmenší třída jazyků, která obsahuje $\emptyset, \{\epsilon\}, \{a\}$ pro všechny symboly a a je uzavřena vzhledem k operacím sjednocení, součinu a iterace.

Ukážeme, že tato třída tvoří právě třídu L_3, tj. třídu jazyka typu 3 Chomského hierarchie.

Poznámka 3.1 Obvyklou notací pro reprezentaci regulárních množin jsou regulární výrazy [**?**].

Věta 3.1 Nechť Σ je konečná abeceda. Pak

(1) $\emptyset$

(2) $\{\epsilon\}$

(3) $\{a\}$ pro všechna $a \in \Sigma$

jsou jazyky typu 3 nad abecedou Σ.

Důkaz:

(1) $G = (\{S\}, \Sigma, \emptyset, S)$ je gramatika typu 3 pro kterou $L(G) = \emptyset$

(2) $G = (\{S\}, \Sigma, \{S \to \epsilon\}, S)$ je gramatika typu 3 pro kterou $L(G) = \{\epsilon\}$

(3) $G_a = (\{S\}, \Sigma, \{S \to a\}, S)$ je gramatika typu 3, pro kterou $L(G_a) = \{a\}$

□

Věta 3.2 Nechť L_1 a L_2 jsou jazyky typu 3 nad abecedou Σ. Pak

(1) $L_1 \cup L_2$

(2) $L_1 \cdot L_2$

(3) L_1^*

jsou jazyky typu 3 nad abecedou Σ.

Důkaz: Protože L_1 a L_2 jsou jazyky typu 3, můžeme předpokládat existenci gramatik $G_1 = (N_1, \Sigma, P_1, S_1)$ a $G_2 = (N_2, \Sigma, P_2, S_2)$ typu 3 takových, že $L(G_1) = L_1$ a $L(G_2) = L_2$. Bez újmy na obecnosti dále předpokládejme, že množiny N_1 a N_2 jsou disjunktní.

(1) Nechť $G_3 = (N_1 \cup N_2 \cup \{S_3\}, \Sigma, P_1 \cup P_2 \cup \{S_3 \to S_1, S_3 \to S_2\}, S_3)$ je gramatika typu 3, kde S_3 je nový nonterminál, $S_3 \notin N_1$, $S_3 \notin N_2$. Zřejmě $L(G_3) = L(G_1) \cup L(G_2)$, poněvadž pro každou derivaci $S_3 \Rightarrow^+_{G_3} w$ existuje buď derivace $S_1 \Rightarrow^+_{G_1} w$, nebo $S_2 \Rightarrow^+_{G_2} w$ a naopak. Protože G_3 je gramatika typu 3, je $L(G_3)$ jazyk typu 3.

(2) Nechť $G_4 = (N_1 \cup N_2, \Sigma, P_4, S_1)$ je gramatika typu 3, jejíž množina přepisovacích pravidel P_4 je definována takto:

   (a) je-li $A \to xB$ v P_1, pak $A \to xB$ je v P_4

   (b) je-li $A \to x$ v P_1, pak $A \to xS_2$ je v P_4

   (c) všechna pravidla z P_2 jsou v P_4.

   Nyní, jestliže $S_1 \Rightarrow^+_{G_1} w$, pak $S_1 \Rightarrow^+_{G_4} wS_2$. Je-li dále $S_2 \Rightarrow^+_{G_2} x$, pak $S_2 \Rightarrow^+_{G_4} x$ a tedy $S_1 \Rightarrow^+_{G_4} wx$ pro libovolné w a x. Z toho plyne $L(G_1) \cdot L(G_2) \subseteq L(G_4)$.

   Předpokládejme, že platí $S_1 \Rightarrow^+_{G_4} w$. Protože v P_4 nejsou žádná pravidla tvaru $A \to x$ z množiny P_1, můžeme tuto derivaci zapsat ve tvaru $S_1 \Rightarrow^+_{G_4} x \quad S_2 \Rightarrow^+_{G_4} xy$, kde $w = xy$, přičemž všechna pravidla použitá v derivaci $S_1 \Rightarrow^+_{G_4} xS_2$ byla odvozena podle (a) a (b). Pak ale musí existovat derivace $S_1 \Rightarrow^+_{G_1} x$ a $S_2 \Rightarrow^+_{G_2} y$ a tudíž $L(G_4) \subseteq L(G_1) \cdot L(G_2)$. Z toho však plyne $L(G_4) = L(G_1) \cdot L(G_2)$.

(3) Nechť $G_5 = \{N_1 \cup \{S_5\}, \Sigma, P_5, S_5\}$, $S_5 \notin N_1$ a množina P_5 je konstruována takto:

   (a) je-li $A \to xB$ v P_1, pak $A \to xB$ je v P_5

   (b) je-li $A \to x$ v P_1, pak $A \to xS_5$ a $A \to x$ jsou v P_5

   (c) $S_5 \to S_1$ a $S_5 \to \epsilon$ jsou v P_5.

Nyní je třeba dokázat tvrzení:

Derivace

$$S_5 \overset{+}{\underset{G_5}{\Rightarrow}} x_1 S_5 \overset{+}{\underset{G_5}{\Rightarrow}} x_1 x_2 S_5 \overset{+}{\underset{G_5}{\Rightarrow}} \ldots \overset{+}{\underset{G_5}{\Rightarrow}} x_1 x_2 \ldots x_{n-1} S_5 \overset{+}{\underset{G_5}{\Rightarrow}} x_1 x_2 \ldots x_{n-1} x_n$$

existuje tehdy a jen tehdy, když existují derivace

$$S_1 \overset{+}{\underset{G_1}{\Rightarrow}} x_1, \;\; S_1 \overset{+}{\underset{G_1}{\Rightarrow}} x_2, \ldots, S_1 \overset{+}{\underset{G_1}{\Rightarrow}} x_n.$$

Z tohoto tvrzení, jehož důkaz ponecháme jako cvičení, bezprostředně plyne $L(G_5) = (L(G_1))^*$. Protože G_5 je gramatika typu 3, je i jazyk $L(G_5)$ typu 3. □

Věta 3.3 Jazyk L je regulární množinou když a jen když je generován gramatikou typu 3.

Důkaz: Část „jen když“ plyne z vět 3.1 a 3.2, důkaz části „když“ využívá teorie regulárních výrazů a lze jej nalézt v [**?**]. □

3.2 Jazyky přijímané konečnými automaty a jazyky typu 3

Definice 3.2 Nedeterministický konečný automat je 5-tice $M = (Q, \Sigma, \delta, q_0, F)$, kde

(1) Q je konečná množina stavů

(2) Σ je konečná vstupní abeceda

(3) δ je zobrazení $Q \times \Sigma \to 2^Q$ (2^Q je množina podmnožin množiny Q)

(4) $q_0 \in Q$ je počáteční stav

(5) $F \subseteq Q$ je množina koncových stavů

Zobrazení δ nazýváme *funkcí přechodu*, je-li $\delta : Q \times \Sigma \to Q$, pak automat M je *deterministický konečný automat.*

Činnost konečného automatu M je dána posloupností přechodů; přechod z jednoho stavu do druhého je řízen funkcí přechodu δ, která na základě přítomného stavu q_i a právě přečteného symbolu $a \in \Sigma$ vstupního řetězce předepisuje budoucí stav q_j automatu. Je-li M deterministický konečný automat, δ předepisuje vždy jediný budoucí stav q_j; v případě nedeterministického konečného automatu δ předepisuje množinu budoucích stavů Q_j, $Q_j \in 2^Q$.

Definice 3.3 Je-li $M = (Q, \Sigma, \delta, q_0, F)$ konečný automat, pak dvojici $C = (q, w)$ z $Q \times \Sigma^*$ nazýváme *konfigurací* automatu M. Konfigurace tvaru (q_0, w) je *počáteční konfigurace*, konfigurace tvaru (q, ϵ), $q \in F$ je *koncová konfigurace. Přechod* automatu M je reprezentován binární relací $\vdash_M$ na množině konfigurací C. Jestliže $\delta(q, a)$ obsahuje q' (tj. $\delta(q, a) = Q_j$, $Q_j \in 2^Q$, $q' \in Q_j$), pak $(q, aw) \vdash_M (q', w)$ pro všechna $w \in \Sigma^*$. Označíme symbolem $\vdash_M^k$, $k \geq 0$, k-tou mocninu ($C \vdash^0 C'$ právě když $C = C'$), symbolem $\vdash_M^+$ tranzitivní uzávěr a symbolem $\vdash_M^*$ tranzitivní a reflexivní uzávěr relace $\vdash_M$. Bude-li zřejmé, že jde o automat M, pak uvedené relace zapíšeme pouze ve tvaru $\vdash$, $\vdash^k$, $\vdash^+$, $\vdash^*$.

δ	**z**	**c**	·	**10**	♯
$\mathbf{q_0}$	q_8	q_7	q_6	q_4	
$\mathbf{q_1}$					
$\mathbf{q_2}$		q_2			q_1
$\mathbf{q_3}$		q_2			
$\mathbf{q_4}$	q_3	q_2			
$\mathbf{q_5}$		q_5		q_4	q_1
$\mathbf{q_6}$		q_5			
$\mathbf{q_7}$		q_7	q_6	q_4	q_1
$\mathbf{q_8}$		q_7	q_6	q_4	

Tabulka 3.1: Funkce přechodů automatu M

Definice 3.4 Říkáme, že vstupní řetězec w je *přijímán* konečným automatem M, jestliže $(q_0, w) \vdash^* (q, \epsilon),\ q \in F$. Jazyk přijímaný konečným automatem M označujeme symbolem $L(M)$ a definujeme ho jako množinu všech řetězců přijímaných automatem M:

$$L(M) = \{w \mid (q_0, w) \overset{*}{\vdash} (q, \epsilon) \wedge q \in F\}$$

Příklad 3.1 Konečný deterministický automat

$$M = (\{g_0, q_1, q_3, q_4, q_5, q_6, q_7, q_8\},\ \{c, z, _{10}, \cdot, \sharp\},\ \delta, q_0, \{q_1\}),$$

jehož funkce přechodu δ je definována tabulkou 3.1, přijímá jazyk algolovských zápisů čísel. Symbolem c značíme prvek množiny $\{0, 1, \ldots, 9\}$, symbolem z prvek množiny $\{+, -\}$; znak $\sharp$ ukončuje zápis čísla.

Např. vstupnímu řetězci $zc.c_{10}zc\sharp$ (např. algolovskému číslu $+3.1_{10} - 5$) bude odpovídat tato posloupnost konfigurací:

$$\begin{aligned}
(q_0, zc.c_{10}\sharp) \quad \vdash \quad & (q_8, c.c_{10}zc\sharp) \\
& (q_7, .c_{10}zc\sharp) \\
& (q_6, c_{10}zc\sharp) \\
& (q_{5,10}\, zc\sharp) \\
& (q_4, zc\sharp) \\
& (q_3, c\sharp) \\
& (q_2, \sharp) \\
& (q_1, \epsilon)
\end{aligned}$$

Poněvadž $(q_0, zc.c_{10}zc\sharp) \vdash^* (q_1, \epsilon)$, patří $zc.c_{10}zc\sharp$ do $L(M)$.

Poznámka 3.2 K základům teorie automatů patří poznatek, *že každý konečný nedeterministický automat M může být převeden na „ekvivalentní“ konečný deterministický automat M',* tj. pro automaty. M a M' platí $L(M') = L(M)$, [**?**], [**?**]. Tato skutečnost je velmi důležitá také v našich aplikacích.

Dříve, než přistoupíme k důkazu ekvivalence třídy $\mathcal{L}_3$ a třídy jazyků přijímaných konečnými automaty, ukážeme, že každý jazyk typu 3 může být generován speciálnějším typem gramatiky, než je pravá lineární gramatika.

Věta 3.4 Každá pravá lineární gramatika $G = (N, \Sigma.P, S)$, tj. gramatika obsahující pouze pravidla typu $A \to xB$ nebo $A \to x$, kde $A, B \in N$, $x \in \Sigma^*$, může být transformována na gramatiku $G' = (N', \Sigma, P', S')$, která obsahuje pouze pravidla tvaru $A \to aB$ nebo $A \to \epsilon$, přičemž $L(G) = L(G')$.

Důkaz: Množinu přepisovacích pravidel P' gramatiky G' konstruujeme takto:

(1) všechna pravidla z P tvaru $A \to aB$ a $A \to \epsilon$ kde $A, B \in N$, $a \in \Sigma$ zařadíme do množiny P'

(2) každé pravidlo tvaru $A \to a_1 a_2 \dots a_n B$; $A, B \in N$, $a_i \in \Sigma$, $n \geq 2$ z množiny P nahradíme v množině P' soustavou pravidel:

$$\begin{aligned} A &\to a_1 A_1 \\ A_1 &\to a_2 A_2 \\ &\vdots \\ A_{n-1} &\to a_n B \end{aligned}$$

Nově zavedené nonterminály $A_1, \dots, A_{n-1}$ přidáme k množině N'. Derivaci $A \Rightarrow_G a_1 \dots a_n B$ zřejmě odpovídá právě derivace $A \Rightarrow^n_{G'} a_1 a_2 \dots a_n B$.

(3) Každé pravidlo tvaru $A \to a_1 \dots a_n$, $a_i \in \Sigma$, $n \geq 2$ z množiny P nahradíme v množině P' soustavou pravidel:

$$\begin{aligned} A &\to a_1 A'_1 \\ A'_1 &\to a_2 A'_2 \\ &\vdots \\ A'_{n-1} &\to a_n A'_n \\ A'_n &\to \epsilon \end{aligned}$$

Derivaci $A \Rightarrow_G a_1 a_2 \dots a_n$ zřejmě odpovídá právě derivace $A \Rightarrow^{n+1}_{G'} a_1 a_2 \dots a_n$.

(4) zbývající, tzv. jednoduchá pravidla tvaru $A \to B, A, B \in N$, nahradíme takto:

(a) určíme množinu $N_A = \{C \mid C = C_1 \Rightarrow C_2 \Rightarrow \dots \Rightarrow C_n = A\}$; $A, B, C_i \in N$, tj. množinu nonterminálů, z nichž lze aplikací jednoduchých pravidel generovat nonterminál A.

(b) ke každému pravidlu typu $A \to aB$ přidáme do P' všechna pravidla tvaru $C \to aB$ pro všechna $C \in N_A$.

Bod 4b aplikuje obecný algoritmus odstranění jednoduchých pravidel bezkontextové gramatiky, který je uveden a dokázán v kapitole 4 (algoritmus 4.3). Jeho součástí je také efektivní výpočet množiny N_A. □

Příklad 3.2 Na základě předchozí věty budeme transformovat pravou lineární gramatiku $G = (\{X, Y\}, \{a, b, c\}, P, X)$, kde P obsahuje pravidla:

$$\begin{aligned} X &\to abc \mid Y \mid \epsilon \\ Y &\to aY \mid cbX \end{aligned}$$

Podle 1 budou v P' pravidla:

$$X \to \epsilon, Y \to aY$$

Podle 2 nahradíme pravidlo $Y \to cbX$ pravidly

$$Y \to cZ, Z \to bX$$

Podle 3 nahradíme pravidlo $Y \to abc$ pravidly

$$Y \to aU, U \to bV, V \to cW, W \to \epsilon$$

Podle 3 nahradíme pravidlo $X \to Y$. Protože $N_Y = \{X\}$ přidáme k P' pravidla

$$X \to aY, X \to cZ$$

Výsledná gramatika má pak tvar $G' = (\{X, Y, Z, U, V, W\}, \{a, b, c\}, P', X)$ kde P' obsahuje pravidla:

$$\begin{aligned} X &\to \epsilon \mid aY \mid cZ \mid aU \\ Y &\to aY \mid cZ \\ Z &\to bX \\ U &\to bV \\ V &\to cW \\ W &\to \epsilon \end{aligned}$$

Definice 3.5 Gramatika $G' = (N, \Sigma, P, S)$ se nazývá *regulární* (pravá regulární) jestliže množina přepisovacích pravidel P obsahuje pravidla pouze tvaru $A \to aB$ nebo $A \to a$ kde $A, B \in N, a \in \Sigma$. V případě, že $L(G)$ obsahuje prázdný řetězec, pak regulární gramatika obsahuje jediné pravidlo s prázdným řetězcem na pravé straně ve tvaru $S \to \epsilon$. Výchozí symbol S se pak nesmí objevit v žádné pravé straně pravidla.

Věta 3.5 Každý jazyk typu 3 lze generovat regulární gramatikou, tj. je *regulárním jazykem*.

Důkaz: Regulární gramatiku získáme z gramatiky zkonstruované podle věty 3.4 odstraněním pravidel s prázdným řetězcem na pravé straně. Systematicky tento postup popisuje algoritmus 4.4 v kapitole 4; zatím jej ilustrujme na příkladě. □

Příklad 3.3 Gramatiku z příkladu 3.2 převedeme na regulární odstraněním pravidla $W \to \epsilon$ (vznikne pravidlo $V \to c$) a pravidla $X \to \epsilon$ (vznikne pravidlo $Z \to b$). Protože $\epsilon \in L(G)$ a X je na pravé straně pravidla $Z \to bX$, musíme zavést nový výchozí symbol X' a odstranit jednoduché pravidlo $X' \to X$. Výsledná gramatika bude mít tvar:

$$G'' = (\{X', X, Y, Z, U, V\}, \{a, b, c\}, P'', X'),$$

P'' obsahuje pravidla:

$$\begin{aligned} X' &\to \epsilon \mid aY \mid cZ \mid aU \\ Y &\to aY \mid cZ \\ Z &\to bX \mid b \\ X &\to aY \mid cZ \mid aU \\ U &\to bV \\ V &\to c \end{aligned}$$

Věta 3.6 Nechť L je jazyk typu 3. Pak existuje konečný automat M takový, že $L = L(M)$. Označíme-li $\mathcal{L}_M$ třídu jazyků přijímaných konečnými automaty, pak ekvivalentní tvrzení má tvar $\mathcal{L}_3 \subseteq \mathcal{L}_M$.

Důkaz:

Podle věty 3.5 můžeme jazyk L generovat gramatikou $G = (N, \Sigma, P, S)$ typu 3, jejíž pravidla mají tvar $A \to aB$ nebo $A \to \epsilon$ $(A, B \in N, a \in \Sigma)$. Sestrojme konečný (nedeterministický) automat M

$$M = (Q, \Sigma, \delta, q_0, F) \text{ kde}$$

(1) $Q = N$

(2) $\Sigma = \Sigma$

(3) $\delta : \delta(A, a)$ obsahuje B pro každé pravidlo $A \to aB$ z P

(4) $q_0 = S$

(5) $F = \{A \mid A \to \epsilon \text{ je pravidlo z } P\}$

Ukážeme, že $L(G) = L(M)$.

Důkaz provedeme indukcí. Dokazovaná induktivní hypotéza nechť má tvar:

> pro každé $A \in N$ platí $A \Rightarrow_G^{i+1} w, w \in \Sigma^*$, právě když $(A, w) \vdash_M^i (C, \epsilon)$ pro nějaké $C \in F$

Nejdříve dokážeme tuto hypotézu pro $i = 0$. Zřejmě platí

$$A \Rightarrow \epsilon \text{ právě když } (A, \epsilon) \overset{0}{\vdash} (A, \epsilon) \text{ pro } A \in F$$

Nyní předpokládejme, že dokazovaná hypotéza platí pro i a položíme $w = ax, a \in \Sigma, |x| = i$. Pak platí $A \Rightarrow^{i+1} w$, právě když $A \Rightarrow aB \Rightarrow^i x$. Podle definice funkce δ pak $\delta(a, A)$ obsahuje B (v důsledku přímé derivace $A \Rightarrow aB$). Na základě induktivní hypotézy platí $B \Rightarrow^i x$, právě když $(B, x) \vdash^{i-1} (C, \epsilon), C \in F$. Shrneme-li tyto skutečnosti, platí:

$$A \Rightarrow aB \overset{i}{\Rightarrow} ax = w, \text{ právě když } (A, aX) \vdash (B, x) \overset{i-1}{\vdash} (C, \epsilon), C \in F$$

$$\text{tj.} \quad A \overset{i+1}{\Rightarrow} w, \text{ právě když } (A, w) \overset{i}{\vdash} (C, \epsilon), C \in F;$$

tím jsme platnost induktivního předpokladu dokázali pro všechna $i \geq 0$.

Speciálně pak platí

$$S \overset{*}{\Rightarrow} w, \text{ právě když } (S, w) \overset{*}{\vdash} (C, \epsilon), C \in F$$

a tedy $L(G) = L(M)$. □

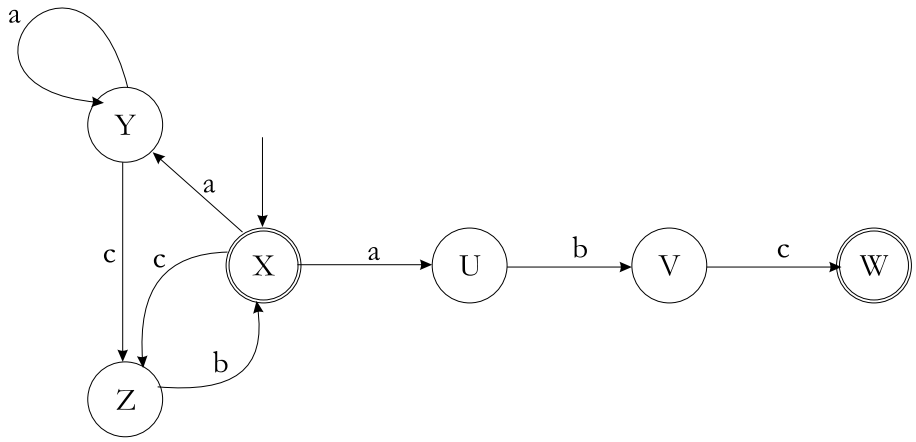
Příklad 3.4 Ke gramatice $G = (\{X, Y, Z, U, V, W\}, \{a, b, c\}, P, X)$, kde P obsahuje pravidla

$$\begin{aligned} X &\to \epsilon \mid aY \mid cZ \mid aU \\ Y &\to aY \mid cZ \\ Z &\to bX \\ U &\to bV \\ V &\to cW \\ W &\to \epsilon \end{aligned}$$

sestrojíme konečný automat, který přijímá jazyk $L(G)$.

Budeme postupovat podle věty 3.6. Funkci přechodů δ reprezentujeme diagramem přechodů (obr. 3.1), v němž koncové stavy jsou vyznačeny dvojitým kroužkem.

$$\begin{aligned} M &= (Q, \Sigma, \delta, q_0, F) \text{ kde} \\ Q &= \{X, Y, Z, U, V, W\} \\ \Sigma &= \{a, b, c\} \\ \delta &: \text{ viz obr. 3.1} \\ q_0 &= x \\ F &= \{X, W\} \end{aligned}$$



Obrázek 3.1: Diagram přechodů automatu M

Věta 3.7 Nechť $L = L(M)$ pro nějaký konečný automat M. Pak existuje gramatika G typu 3 taková, že $L = L(G)$, tj. $\mathcal{L}_M \subseteq L_3$.

Důkaz: Nechť $M = (Q, \Sigma, \delta, q_0, F)$. Protože každý nedeterministický automat může být převeden na deterministický automat přijímající stejný jazyk, předpokládejme, že M je deterministický automat. Nechť G je gramatika typu 3, $G = (Q, \Sigma, P, q_0)$, jejíž množina P přepisovacích pravidel je definována takto:

(1) je-li $\delta(q, a) = r$, pak P obsahuje pravidlo $q \to ar$

(2) je-li $p \in F$, pak P obsahuje pravidlo $p \to \epsilon$

V další části probíhá důkaz zcela analogicky důkazu věty 3.6. □

Příklad 3.5 Na základě příkladu 3.1 sestrojíme gramatiku typu 3, která generuje jazyk algolovských zápisů čísel. Přechodová funkce deterministického konečného automatu, který přijímá tento jazyk, je v tabulce 3.1. Výsledná gramatika bude tvaru:

$$G = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}, \{c, z, {}_10, ., \sharp\}, P, q_0)$$

P obsahuje pravidla

$$\begin{aligned}
q_0 &\to zq_8 \mid cq_7 \mid \cdot q_6 \mid 10^{q_4} \\
q_1 &\to \epsilon \\
q_2 &\to cq_2 \mid \sharp q_1 \\
q_3 &\to cq_2 \\
q_4 &\to zq_3 \mid cq_2 \\
q_5 &\to cq_5 \mid 10^{q_4} \mid \sharp q_1 \\
q_6 &\to cq_5 \\
q_7 &\to cq_7 \mid \cdot q_6 \mid 10^{q_4} \mid \sharp q_1 \\
q_8 &\to cq_7 \mid \cdot q_6 \mid 10^{q_4}
\end{aligned}$$

Poznamenejme, že převod této gramatiky na gramatiku regulární je velice snadný, stačí dosadit za q_1 prázdný řetězec a odstranit pravidlo $q_1 \to \epsilon$.

Věta 3.8 Třída jazyků, jež jsou přijímány konečnými automaty, je totožná s třídou jazyků typu 3 Chomského hierarchie.

Důkaz: Tvrzení bezprostředně plyne z vět 3.6 a 3.7. □

Definice 3.6 Gramatika $G = (N, \Sigma, P, S)$ se nazývá levá lineární gramatika, jestliže množina P obsahuje pouze pravidla typu $A \to Bx$ nebo $A \to x$ kde $A, B \in N$, $x \in \Sigma^*$.

Věta 3.9 Každý jazyk typu 3 může být generován levou lineární gramatikou.

Důkaz: Úplný důkaz ponecháme na cvičení. Lze postupovat tak, že nejprve z definice regulární množiny dokážeme tvrzení: je-li L regulární množina, pak L^R je také regulární množina. Dále ukážeme: je-li $G = (N, \Sigma, P, S)$ pravá lineární gramatika, pak levá lineární gramatika G', jejíž množina pravidel má tvar $P' = \{A \to \alpha^R \mid A \to \alpha \text{ je v } P\}$, generuje jazyk $(L(G))^R$. □

Příklad 3.6 Gramatika $G' = (\{X, Y\}, \{a, b, c\}, P', X)$ kde P' obsahuje pravidla

$$\begin{aligned}
X &\to cba \mid Y \mid \epsilon \\
Y &\to Ya \mid Xbc
\end{aligned}$$

je levá lineární gramatika, pro kterou $L(G') = (L(G))^R$, G je pravá lineární gramatika z příkladu 3.2.

Definice 3.7 Levá lineární gramatika G = (N, Σ, P, S) se nazývá *regulární* (přesněji levá regulární), jestliže pravidla v P mají tvar $A \to Ba$ nebo $A \to a$. Pokud L(G) obsahuje prázdný řetězec ϵ, pak jediné přípustné pravidlo s pravou stranou obsahující prázdný řetězec je $S \to \epsilon$ a S se v takovém případě nesmí objevit na pravé straně žádného pravidla.

Poznamenejme, že konstrukce levé regulární gramatiky k dané levé lineární gramatice je zcela analogická konstrukci pravé regulární gramatiky k dané pravé lineární gramatice. Tuto konstrukci ilustruje příklad:

Příklad 3.7 Uvažujme gramatiku z příkladu 3.6, jež má pravidla:

$$\begin{aligned} X &\to cba \mid Y \mid \epsilon \\ Y &\to Ya \mid Xbc \end{aligned}$$

1. krok zavádí pouze pravidla typu $A \to Ba$ nebo $A \to \epsilon$:

$$\begin{aligned} X &\to Ua \mid Ya \mid Zc \mid \epsilon \\ Y &\to Ya \mid Zc \\ U &\to Vb \\ V &\to Wc \\ W &\to \epsilon \\ Z &\to Xb \end{aligned}$$

2. krok odstraňuje pravidla typu $A \to \epsilon$ a upravuje gramatiku na konečný tvar:

$$\begin{aligned} X' &\to Ua \mid Ya \mid Zc \mid \epsilon \\ Y &\to Ya \mid Zc \\ U &\to Vb \\ V &\to c \\ Z &\to Xb \mid b \\ X &\to Ua \mid Ya \mid Zc \end{aligned}$$

Povšimněme si, v čem se liší pravidla této gramatiky od pravidel pravé regulární gramatiky z příkladu 3.3, jež byla získána z pravé lineární gramatiky (příklad 3.2)

Algoritmus 3.1 Konstrukce nedeterministického konečného automatu k pravé regulární gramatice.

Vstup: Pravá regulární gramatika $G = (N, \Sigma, P, S)$, jejíž pravidla mají tvar $A \to aB$ a $A \to a$ kde $A, B \in N$ a $a \in \Sigma$, případně $S \to \epsilon$, je-li $\epsilon \in L(G)$.

Výstup: Nedeterministický konečný automat $M = (N, \Sigma, \delta, q_0, F)$, pro který je $L(M) = L(G)$.

Metoda:

(1) Položíme $Q = N \cup \{q_F\}$, kde q_F reprezentuje koncový stav.

(2) Množina vstupních symbolů automatu M je identická s množinou terminálů gramatiky G.

(3) Funkci přechodů δ definujeme takto:

(a) Je-li $A \to aB$ pravidlo z P, pak $\delta(A, a)$ obsahuje stav B

(b) Je-li $A \to a$ pravidlo z P, pak $\delta(A, a)$ obsahuje q_F

(4) $q_0 = S$

(5) Je-li $S \to \epsilon$ pravidlo z P, pak $F = \{S, q_F\}$, v opačném případě $F = \{q_F\}$.

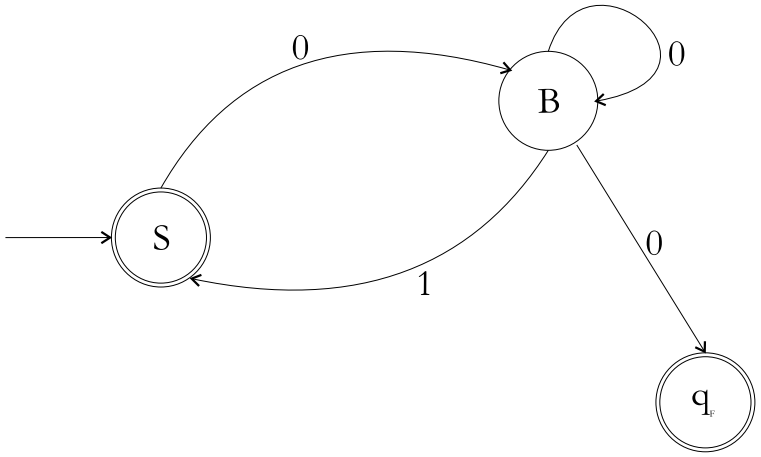
Věta 3.10 Nechť G je pravá regulární gramatika a M konečný automat z algoritmu 3.1. Pak $L(M) = L(G)$.

Důkaz: Důkaz této věty je modifikací důkazu věty 3.6; přenecháme ho jako cvičení. □

Příklad 3.8 Uvažujeme gramatiku $G = (\{S, B\}, \{0, 1\}, P, S)$ s pravidly

$$\begin{aligned} S &\to 0B \mid \epsilon \\ B &\to 0B \mid 1S \mid 0 \end{aligned}$$

Diagram přechodů automatu přijímajícího jazyk $L(G)$ má, na základě konstrukce podle algoritmu 3.1, tvar:



Obrázek 3.2: Diagram přechodů

Algoritmus 3.2 Konstrukce nedeterministického konečného automatu k levé regulární gramatice.

Vstup: Levá regulární gramatika $G = (N, \Sigma, P, S)$ jejíž pravidla mají tvar $A \to Ba$ a $A \to a$ kde $A, B \in N$ a $a \in \Sigma$, případně $S \to \epsilon$, je-li $\epsilon \in L(G)$.

Výstup: Nedeterministický konečný automat $M = (N, \Sigma, \delta, q_0, F)$, pro který je $L(M) = L(G)$.

Metoda:

(1) Položíme $Q = N \cup \{q_0\}$

(2) Množina vstupních symbolů automatu M je identická s termální abecedou gramatiky G.

(3) Funkci přechodu δ definujeme takto:

(a) Je-li $A \to Ba$ pravidlo z P, pak $\delta(B, a)$ obsahuje stav A.

(b) Je-li $A \to a$, pak $\delta(q_0, a)$ obsahuje stav A.

(4) q_0 je počáteční stav automatu M

(5) Je-li $S \to \epsilon$ pravidlo z P pak $F = \{S, q_0\}$, v opačném případě $F = \{S\}$

Věta 3.11 Nechť G je levá lineární gramatika a M konečný automat z algoritmu 3.2. Pak $L(M) = L(G)$.

Důkaz: Nejprve dokážeme, že $\epsilon \in L(G)$, právě když $\epsilon \in L(M)$. Podle konstrukce automatu M, $q_0 \in F$ právě když v P je pravidlo $S \to \epsilon$ a tedy

$$S \Rightarrow \epsilon, \text{ právě když } (q_0, \epsilon) \overset{0}{\vdash} (q_0, \epsilon), \ q_0 \in F$$

Nyní dokážeme, že pro libovolné $w \in \Sigma^+$ platí

$$S \overset{+}{\Rightarrow} w, \text{ právě když } (q_0, w) \overset{+}{\vdash} (S, \epsilon), \ S \in F$$

Položme $w = ax, \ a \in \Sigma, \ x \in \Sigma^*$. Induktivní hypotéza nechť má tvar:

$$S \overset{i}{\Rightarrow} Ax \Rightarrow ax, \text{ právě když } (q_0, ax) \vdash (A, x) \overset{i}{\vdash} (S, \epsilon), \ A \in N, \ S \in F, \ |x| = i$$

Pro $i = 0$ dostaneme:

$$S \Rightarrow a, \text{ právě když } (q_0, a) \vdash (S, \epsilon),$$

což plyne přímo z definice funkce přechodů automatu M ($\delta(q_0, a)$ obsahuje S, právě když $S \to a$ je v P).

Pro $i \geq 1$ je třeba dokázat:

(a) $Ax \Rightarrow ax$, právě když $(q_0, ax) \vdash (A, x)$

(b) $S \Rightarrow^i Ax$ právě když $(A, x) \vdash^i (S, \epsilon)$

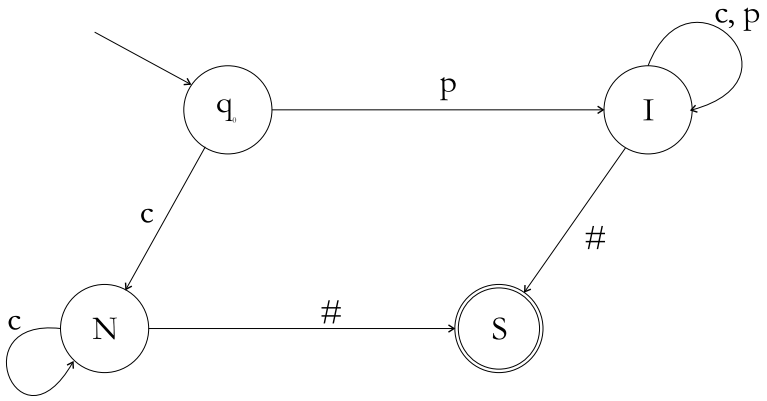
Důkaz tvrzení (a) a (b) ponecháme jako cvičení. □

Příklad 3.9 Uvažujme gramatiku $G = (\{S, I, N\}, \{c, p, \sharp\}, P, S)$ s pravidly:

$$\begin{aligned} E &\to I\sharp \mid N\sharp \\ I &\to p \mid Ip \mid Ic \\ N &\to c \mid Nc \end{aligned}$$

Značí-li c arabskou číslici a p písmeno, pak $L(G)$ je jazyk identifikátorů (nonterminál I) a celých čísel bez znaménka (nonterminál N). Každá věta jazyka $L(G)$ končí koncovým znakem $\sharp$.

Diagram přechodů automatu, který přijímá jazyk $L(G)$, má na základě algoritmu 3.2 tvar:



Obrázek 3.3: Diagram přechodů

3.3 Cvičení

Cvičení 3.3.1 Ke konečnému nedeterministickému automatu

$$M = (\{q_0, q_1, q_2, q_3, q_F\}, \{1, 2, 3\}, \delta, q_0, \{q_F\}),$$

kde zobrazení δ je definováno touto tabulkou

	Σ		
Q	1	2	3
$\mathbf{q_0}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_3\}$
$\mathbf{q_1}$	$\{q_1, q_F\}$	$\{q_1\}$	$\{q_1\}$
$\mathbf{q_2}$	$\{q_2\}$	$\{q_2, q_F\}$	$\{q_1\}$
$\mathbf{q_3}$	$\{q_3\}$	$\{q_3\}$	$\{q_3, q_F\}$
$\mathbf{q_F}$	$\emptyset$	$\emptyset$	$\emptyset$

sestrojte deterministický automat M', pro který platí $L(M') = L(M)$.

Cvičení 3.3.2 Nechť $L_1 = L(M_1)$ a $L_2 = L(M_2)$ jsou jazyky přijímané konečnými automaty $M_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$ a $M_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2)$. Analogicky větě 3.2 ukažte konstrukci automatů M_3, M_4 a M_5, pro které platí:

$$L(M_3) = L_1 \cup L_2, \;\; L(M_4) = L_1 \cdot L_2 \text{ a } L(M_5) = L_1^*.$$

Cvičení 3.3.3 K pravé lineární gramatice, která osahuje pravidla

$$\begin{aligned} A &\to B \mid C \\ B &\to 0B \mid 1B \mid 011 \\ C &\to 0D \mid 1C \mid \epsilon \\ D &\to 0C \mid 1D \end{aligned}$$

vytvořte pravou lineární gramatiku, jež generuje stejný jazyk. Nonterminál A je výchozím symbolem gramatiky.

Cvičení 3.3.4 Vytvořte regulární gramatiku, která generuje jazyk přijímaný automatem M ze cvičení 3.3.1.

Cvičení 3.3.5 Vytvořte konečný automat, který přijímá jazyk generovaný gramatikou ze cvičení 3.3.3.

Cvičení 3.3.6 Na základě algoritmu, jenž je „inverzní“ k algoritmu 3.2, sestrojte levou regulární gramatiku pro jazyk algolovských čísel. Automat přijímající tento jazyk je uveden v příkladě 3.1.

Cvičení 3.3.7 Na základě gramatiky ze cvičení 2.4.2 vytvořte konečný deterministický automat, který přijímá jazyk čísel programovacího jazyka Pascal.

Kapitola 4

Bezkontextové jazyky

Význam bezkontextových gramatik a jazyků je dán dvěma důležitými faktory:

(1) Bezkontextovými gramatikami jsme schopni popsat převážnou většinu rysů současných programovacích jazyků.

(2) Jsou známé algoritmy, které umožňují efektivně analyzovat věty bezkontextových jazyků, tj. jazyků generovaných bezkontextovými gramatikami. Tyto algoritmy tvoří základní část reálných překladačů – tzn. syntaktický analyzátor.

Definice 4.1 Bezkontextová gramatika G je čtveřice $G = (N, \Sigma, P, S)$

(1) N je konečná množina nonterminálních symbolů

(2) Σ je konečná množina terminálních symbolů

(3) P je konečná množina přepisovacích pravidel (pravidel) tvaru $A \to \alpha$, $A \in N$ a $\alpha \in (N \cup \Sigma)^*$

(4) $S \in N$ je výchozí symbol gramatiky.

Dále budeme gramatikou bez další specifikace rozumět bezkontextovou gramatikou.

Příklad 4.1 Gramatika $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla

$$\begin{aligned} S &\to AB \\ A &\to aAb \mid ab \\ B &\to cBd \mid cd \end{aligned}$$

generuje bezkontextový jazyk $L(G) = \{a^n b^n c^m d^m \mid \ n \leq 1, \ m \leq 1\}$

Jazyk $L(G)$ je součinem dvou bezkontextových jazyků generovaných gramatikami

$$\begin{aligned} G_1 &= (\{A\}, \{a, b\}, \{A \to aAb, A \ \to ab\}, A) \\ G_2 &= (\{B\}, \{c, d\}, \{B \to cBd, B \to cd\}, B) \end{aligned}$$

Poznamenejme, že jazyk $L(G_1) = \{a^n b^n \mid n \leq 1\}$ a tudíž ani jazyk $L(G)$ není jazykem regulárním, protože konečný automat nemůže pro libovolné n „počítat“ stejný počet výskytů symbolů a a b. Na druhé straně, jazyk $\{a^n b^n c^n d^m \mid n \leq 1\}$, dokonce ani jazyk $\{a^n b^n c^n \mid n \leq 0\}$ není jazykem bezkontextovým.

4.1 Derivační strom

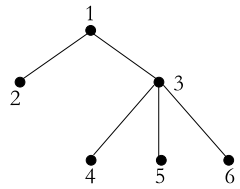
Důležitým prostředkem pro grafické vyjádření struktury věty (její derivace) je strom, který se nazývá derivačním nebo syntaktickým stromem.

Připomeňme, že strom je orientovaný acyklický graf s těmito vlastnostmi:

(1) Existuje jediný uzel, tzv. *kořen stromu*, do něhož nevstupuje žádná hrana.

(2) Do všech ostatních uzlů grafu vstupuje právě jedna hrana.

Uzly, z nichž žádná hrana nevystupuje, se nazývají *koncové uzly stromu* (listy). Při kreslení stromu je obyčejně dodržována tato konvence: kořen leží nejvýše, všechny hrany jsou orientovány směrem dolů. Budeme-li tuto konvenci dodržovat, pak můžeme vynechat šipky, které označují orientaci hran. Kořenem stromu na 4.1 je uzel 1, uzly 2, 4, 5, 6 jsou koncovými uzly stromu.



Obrázek 4.1: Příklad stromu

Definice 4.2 Nechť δ je věta nebo větná forma generovaná v gramatice $G = (N, \Sigma, P, S)$ a nechť $S = \nu_0 \Rightarrow \nu_1 \Rightarrow \nu_2 \ldots \Rightarrow \nu_k = \delta$ její derivace v G. *Derivační strom* příslušející této derivaci je strom s těmito vlastnostmi:

(1) Uzly derivačního stromu jsou označeny symboly z množiny $N \cup \Sigma$; kořen stromu je označen výchozím symbolem S.

(2) Přímé derivaci $\nu_{i-1} \Rightarrow \nu_i,\ i = 0, 1, \ldots, k$, kde

$$\begin{aligned} &\nu_{i-1} = \mu A \lambda,\ \mu, \lambda \in (N \cup \Sigma)^*,\ A \in N \\ &\nu_i = \mu \alpha \lambda \text{ a} \\ &A \to \alpha,\ \alpha = X_1 \ldots X_n \text{ je pravidlo z } P, \end{aligned}$$

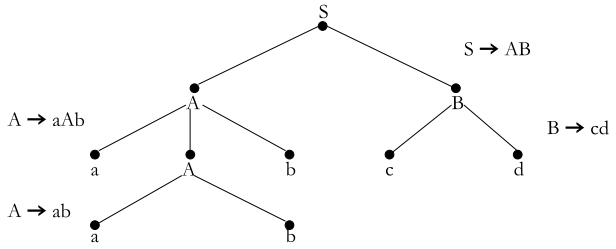
odpovídá právě n hran $(A, X_j), j = 1, \ldots, n$ vycházejících z uzlu A jež jsou uspořádány zleva doprava v pořadí $(A, X_1), (A, X_2), \ldots (A, X_n)$.

(3) Označení koncových uzlů derivačního stromu vytváří zleva doprava větnou formu nebo větu δ (plyne z (1) a (2)).

Příklad 4.2 V Gramatice z příkladu 4.1 můžeme generovat řetězec *aabbcd* např. derivací:

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcd$$

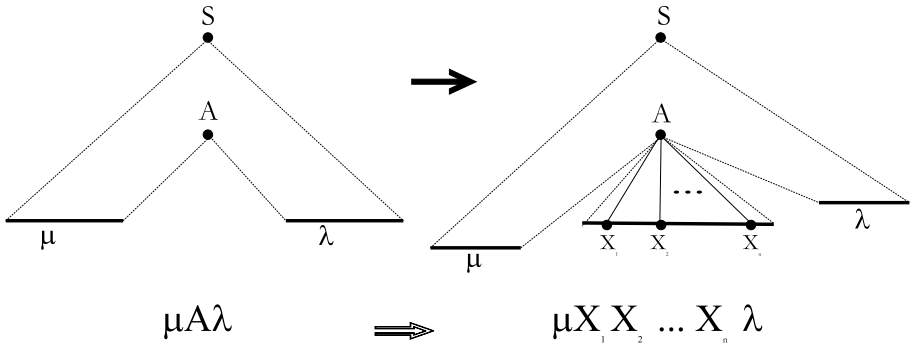
Derivační strom odpovídající této derivaci je na obrázku 4.2. Po stranách jsou uvedena použitá pravidla.



Obrázek 4.2: Derivační strom

Poznámka 4.1 Uzel derivačního stromu, který je označen terminálním symbolem, musí být zřejmě koncovým uzlem derivačního stromu.

Při rekonstrukci derivačního stromu k dané derivaci opakovaně aplikujeme bod (2) z definice 4.2. Tuto aplikaci ilustruje obr. 4.3.



Obrázek 4.3: Konstrukce derivačního stromu

Příklad 4.3 Uvažujme gramatiku

$$G = (\{\langle\text{výraz}\rangle, \langle\text{term}\rangle, \langle\text{faktor}\rangle\}, \{+, -, *, /, (,), i\}, P, \langle\text{výraz}\rangle),$$

které se často používá pro popis aritmetického výrazu s binárními operacemi $+, -, *, /$. Terminální symbol i odpovídá identifikátoru. Množina P obsahuje tato přepisovací pravidla:

$$\begin{aligned}
\langle\text{výraz}\rangle &\rightarrow \langle\text{term}\rangle \mid \langle\text{výraz}\rangle + \langle\text{term}\rangle \mid \langle\text{výraz}\rangle - \langle\text{term}\rangle \\
\langle\text{term}\rangle &\rightarrow \langle\text{faktor}\rangle \mid \langle\text{term}\rangle * \langle\text{faktor}\rangle \mid \langle\text{term}\rangle / \langle\text{faktor}\rangle \\
\langle\text{faktor}\rangle &\rightarrow (\langle\text{výraz}\rangle) \mid i
\end{aligned}$$

Jak lze snadno ukázat, větami jazyka $L(G)$ jsou například řetězce i, (i), $i * i$, $i * i + i$, $i * (i + i)$.

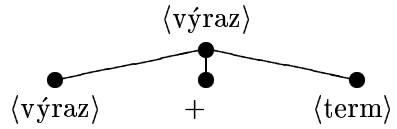
Předpokládejme, že máme zkonstruovat derivační strom pro větnou formu $\langle\text{výraz}\rangle + \langle\text{term}\rangle * \langle\text{faktor}\rangle$. Nejprve ukažme, že tento řetězec je skutečně větnou formou:

$$\langle\text{výraz}\rangle \Rightarrow \langle\text{výraz}\rangle + \langle\text{term}\rangle \Rightarrow \langle\text{výraz}\rangle + \langle\text{term}\rangle * \langle\text{faktor}\rangle$$

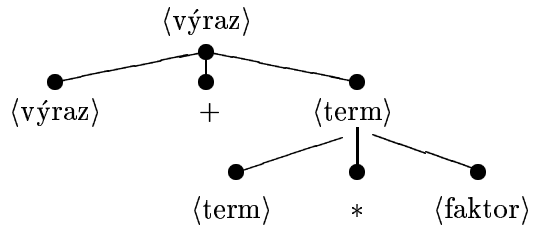
Derivační strom začínáme vytvářet od výchozího symbolu:

$\langle\text{výraz}\rangle$
●

První přímé derivaci odpovídá konstrukce:



Po znázornění druhé přímé derivace obdržíme výsledný derivační strom.



Je-li tedy dána derivace větné formy, pak této derivaci přísluší právě jeden derivační strom. Důkaz vyplývá z konstrukce derivačního stromu. Podívejme se nyní, zda uvedené tvrzení platí také obráceně: K derivačnímu stromu větné formy přísluší pouze jedna derivace. Uvažujme gramatiku G z příkladu 4.1.

$$\begin{aligned} S &\rightarrow A\ B \\ A &\rightarrow aAb \mid ab \\ B &\rightarrow cBd \mid cd \end{aligned}$$

Na obr. 4.2 jsme znázornili derivační strom k derivaci

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcd$$

Ukážeme nyní, že existují i jiné derivace věty *aabbcd*.

$$S \Rightarrow AB \Rightarrow Acd \Rightarrow aAbcd \Rightarrow aabbcd$$

nebo

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aAbcd \Rightarrow aabbcd$$

Uvedené derivace věty *aabbcd* se liší v pořadí, v němž byly vybírány nonterminály pro přímé derivace. Toto pořadí však ve výsledném derivačním stromě není postiženo, a proto budou derivační stromy příslušející k 2. a 3. z uvedených derivací věty *aabbcd* totožné s derivačním stromem na obr. 4.2. Neplatí tedy tvrzení, že k derivačnímu stromu přísluší pouze jedna derivace.

Poznamenejme, že jiné derivace věty *aabbcd* v gramatice z příkladu 4.2 neexistují. V první a druhé z uvedených derivací byla přepisovací pravidla aplikována určitým kanonickým způsobem, který je charakteristický pro nejužívanější syntaktické analyzátory překladačů programovacích jazyků. Zavedeme si pro tyto speciální derivace vlastní označení.

Definice 4.3 Nechť $S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = \alpha$ je derivace větné formy α. Jestliže byl v každém řetězci $\alpha_i,\ i = 1, \ldots, n-1$ přepsán nejlevější (nejpravější) nonterminál, pak tuto derivaci nazýváme *levou* (*pravou*) derivací větné formy α.

První derivace věty *aabbcd* je tedy derivací levou, druhá je derivací pravou.

Je-li $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$ levá derivace věty w, pak je každé $\alpha_i,\ i = 0, 1, \ldots, n-1$ tvaru $x_i A_i \beta_i$, kde $x_i \in \Sigma^*,\ A_i \in N$ a $\beta_i \in (N \cup \Sigma^*)$. K získání větné formy α_{i+1} bude přepsán nonterminál A_i. Obrácená situace platí pro pravou derivaci.

4.2 Fráze větné formy

Definice 4.4 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť řetězec $\lambda = \alpha\beta\gamma$ je větná forma. Podřetězec β se nazývá *frází větné formy* λ vzhledem k nonterminálu A z N, jestliže platí:

$$\begin{aligned} S &\Rightarrow^* \alpha A \gamma \\ A &\Rightarrow^+ \beta \end{aligned}$$

Podřetězec β je *jednoduchou frází větné formy* λ, jestliže platí:

$$\begin{aligned} S &\Rightarrow^* \alpha A \gamma \\ A &\Rightarrow \beta \end{aligned}$$

Příklad 4.4 Máme nalézt všechny fráze a všechny jednoduché větné formy ⟨výraz⟩ + ⟨term⟩ ∗ ⟨faktor⟩ v gramatice, která popisuje aritmetický výraz. Jak již bylo uvedeno, derivace této větné formy má tvar:

$$\langle\text{výraz}\rangle \Rightarrow \langle\text{výraz}\rangle + \langle\text{term}\rangle \Rightarrow \langle\text{výraz}\rangle + \langle\text{term} * \langle\text{faktor}\rangle$$

Protože platí

$$\begin{aligned} \langle\text{výraz}\rangle &\Rightarrow^* \langle\text{výraz}\rangle + \langle\text{term}\rangle \qquad \text{a} \\ \langle\text{term}\rangle &\Rightarrow \langle\text{term}\rangle * \langle\text{faktor}\rangle \end{aligned}$$

je řetězec ⟨term⟩ ∗ ⟨faktor⟩ jednoduchou frází větné formy ⟨výraz⟩ + ⟨term⟩ ∗ ⟨faktor⟩ vzhledem k nonterminálu ⟨term⟩.

Dále je:

$$\begin{aligned} \langle\text{výraz}\rangle &\Rightarrow^* \langle\text{výraz}\rangle \\ \langle\text{výraz}\rangle &\Rightarrow^+ \langle\text{výraz}\rangle + \langle\text{term}\rangle * \langle\text{faktor}\rangle \end{aligned}$$

a z toho vyplývá, že větná forma ⟨výraz⟩ + ⟨term⟩ ∗ ⟨faktor⟩ je frází sama k sobě vzhledem k výchozímu symbolu. Tato skutečnost je důsledkem triviálního případu v definici fráze, kdy jsou řetězce α a γ prázdné. Jiné fráze větné formy ⟨výraz⟩ + ⟨term⟩ ∗ ⟨faktor⟩ neexistují.

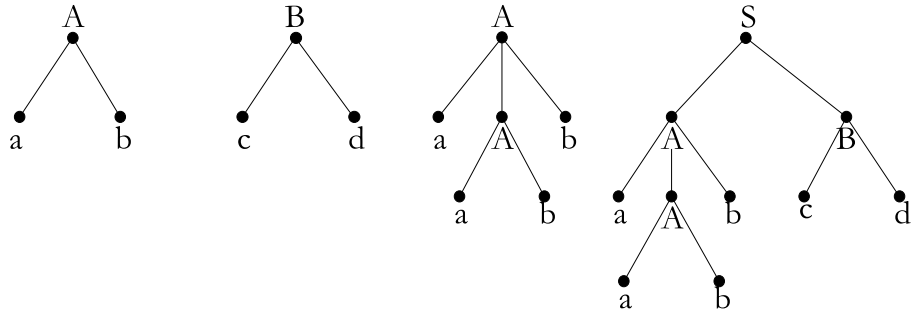
Pojem fráze je stěžejním pojmem pro syntaktickou analýzu. Celá třída syntaktických analyzátorů je postavena na metodách hledání nejlevější jednoduché fráze větné formy (věty). Protože dále budeme pojmu nejlevější jednoduchá fráze často používat, zavedeme pro něj speciální označení, *l-fráze.*

Ve větné formě ⟨výraz⟩ + ⟨term⟩ ∗ ⟨faktor⟩ je jediná jednoduchá fráze: ⟨term⟩ ∗ ⟨faktor⟩. Tato fráze je tedy zároveň l-frází.

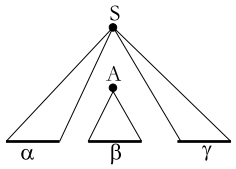
S pojmem fráze větné formy je velmi úzce svázán pojem podstrom příslušného derivačního stromu. Podstromem derivačního stromu budeme rozumět tu část tohoto stromu, která je vymezena jistým uzlem, tzv. kořenem podstromu, spolu se všemi uzly, které jsou z kořene podstromu dostupné prostřednictvím příslušných hran, včetně těchto hran.

Příklad 4.5 Podstromy derivačního stromu věty *aabbcd* z obr. 4.2 jsou stromy z obr. 4.4.

Předpokládejme nyní, že nonterminál A je kořenem podstromu derivačního stromu. Je-li β řetězec koncových uzlů tohoto podstromu, pak jistě platí $A \Rightarrow^+ \beta$. Nechť α je řetězec koncových



Obrázek 4.4: Podstromy derivačního stromu



Obrázek 4.5: Fráze větné formy

uzlů vlevo, γ řetězec koncových uzlů derivačního stromu vpravo od β. Pak platí $S \Rightarrow^* \alpha\beta\gamma$; S je kořenem derivačního stromu. To ovšem znamená, že β je frází větné formy $\alpha\beta\gamma$ vzhledem k nonterminálu A. Situaci ilustruje obr. 4.5.

Podstrom derivačního stromu tedy odpovídá frázi příslušné větné formy. Fráze je tvořena koncovými uzly podstromu. Jednoduchá fráze odpovídá podstromu, jenž je výsledkem přímé derivace $A \Rightarrow \beta$.

Příklad 4.6 V gramatice z příkladu 4.1 nalezněte fráze věty $a^2b^2c^2d^2$. Nejdříve sestavíme derivační strom. Pro jeho konstrukci vytvořme (např.) levou derivaci této věty:

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcBd \Rightarrow aabbccdd$$

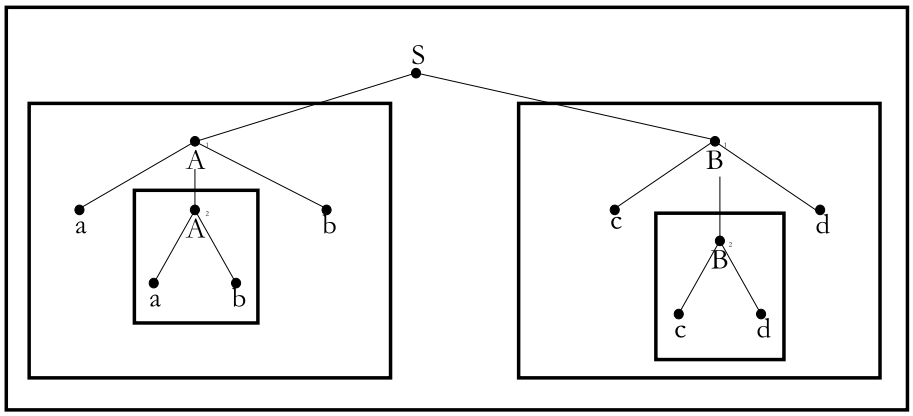
Vyznačené podstromy odpovídají dále uvedeným frázím definovaným k příslušným nonterminálům. Horní indexy slouží k rozlišení výskytu téhož nonterminálu (viz 4.6)

Fráze	Nonterminál
$aabbccdd$	S
$aabb$	A^1
ab	A^2
$ccdd$	B^1
cd	B^2

Fráze ab a cd jsou jednoduché, ab je l-fráze.

4.3 Víceznačnost gramatik

Definice 4.5 Nechť G je gramatika. Říkáme, že věta w generovaná gramatikou G je *víceznačná*, existují-li alespoň dva různé derivační stromy s koncovými uzly tvořícími větu w. *Gramatika* G



Obrázek 4.6: Podstromy derivačního stromu

je víceznačná, jestliže generuje alespoň jednu víceznačnou větu. V opačném případě mluvíme o jednoznačné gramatice.

Povšimněte si, že definujeme víceznačnou gramatiku, nikoli víceznačný jazyk. V mnoha případech lze vhodnými transformacemi víceznačné gramatiky odstranit víceznačnost vět, aniž se samozřejmě změní jazyk generovaný získanou jednoznačnou gramatikou. Existují však jazyky, které nelze generovat jednoznačnou gramatikou. Takové jazyky jsou pak nazývány jazyky s *inherentní víceznačností.*

Příklad 4.7 Uvažujme gramatiku $G = (\{E\}, \{+, -, *, /, (,), i\}, P, E)$, kde P je množina pravidel

$$E := E \ + \ E \ \mid E \ - \ E \ \mid E \ * \ E \ \mid E \ / \ E \ \mid (\ E \ ) \ \mid i$$

Jazyk $L(G)$ je totožný s jazykem generovaným gramatikou z příkladu 4.3 a je tvořen aritmetickými výrazy s binárními operacemi.

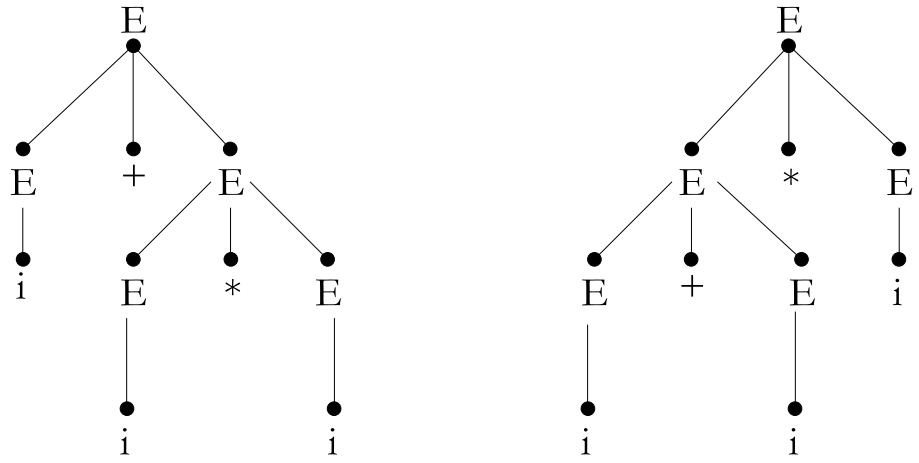
Tato gramatika je na rozdíl od gramatiky z příkladu 4.3 víceznačná. Vezměme například větu $i + i * i$ a uvažujme všechny možné derivační stromy příslušející k této větě (viz obr. 4.7).

Existence dvou různých derivačních stromů k větě $i+i*i$ dokazuje, že gramatika G je víceznačná. Označuje-li se + sečítání a $*$ násobení, pak není jasné, zda první operací bude násobení (první derivační strom), a nebo sečítání (druhý derivační strom). Jednoznačná gramatika z příkladu 4.3 na druhé straně respektuje obvyklou prioritou operací (násobení má přednost před sečítáním), jak je patrno z jediného derivačního stromu věty i+i*i na obr. 4.8

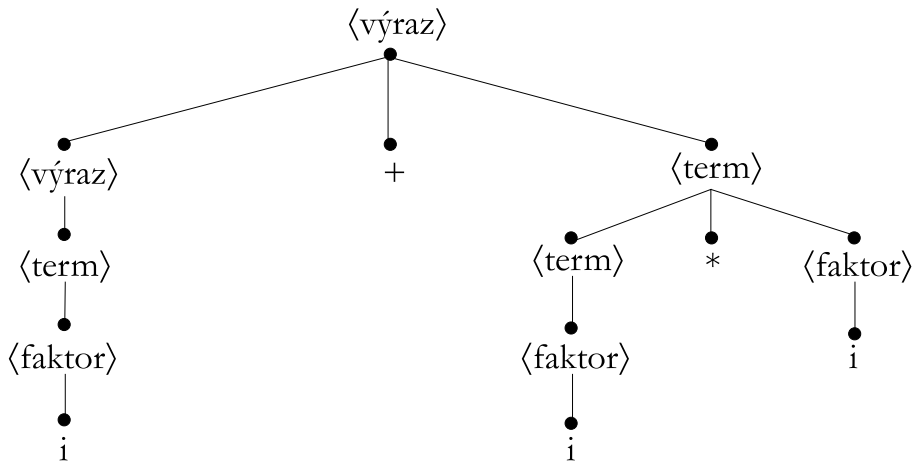
Pro bezkontextové gramatiky bylo dokázáno, že problém, zda daná gramatika je a nebo není víceznačná, je nerozhodnutelný, tj. neexistuje algoritmus, který by v konečném čase odhalil víceznačnost každé bezkontextové gramatiky.

Příklad 4.8 Gramatika obsahující pravidlo tvaru $A \to A$ je zřejmě víceznačná. Toto pravidlo můžeme vypustit, aniž změníme jazyk generovaný takto zredukovanou gramatikou.

Poznámka 4.2 Víceznačnost gramatiky, pokud negeneruje jazyk s inherentní víceznačností, je obecně pokládána za negativní rys, poněvadž vede k větám, jež mají několik interpretací. Na druhé straně však víceznačná gramatika může být jednodušší, než odpovídající jednoznačná



Obrázek 4.7: Derivační stromy věty $i + i * i$



Obrázek 4.8: Derivační strom věty $i + i * i$ v G_2

gramatika (viz příklad 4.7). Této skutečnosti se někdy využívá při konstrukci překladačů takovým způsobem, že se použije víceznačné gramatiky a nežádoucí interpretace víceznačné věty se vyloučí dodatečným sémantickým pravidlem.

Příklad 4.9 Jedním z nejznámějších příkladů víceznačnosti v programovacích jazycích jsou konstrukce s `then` a `else`. Skutečně, přepisovací pravidla

$$\begin{aligned} S &\to \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ S \\ S &\to \texttt{if}\ b\ \texttt{then}\ S \\ S &\to p \end{aligned}$$

kde b značí booleovský výraz a p jiný, než podmíněný příkaz, vedou k víceznačné interpretaci podmíněného příkazu. Např. k větě

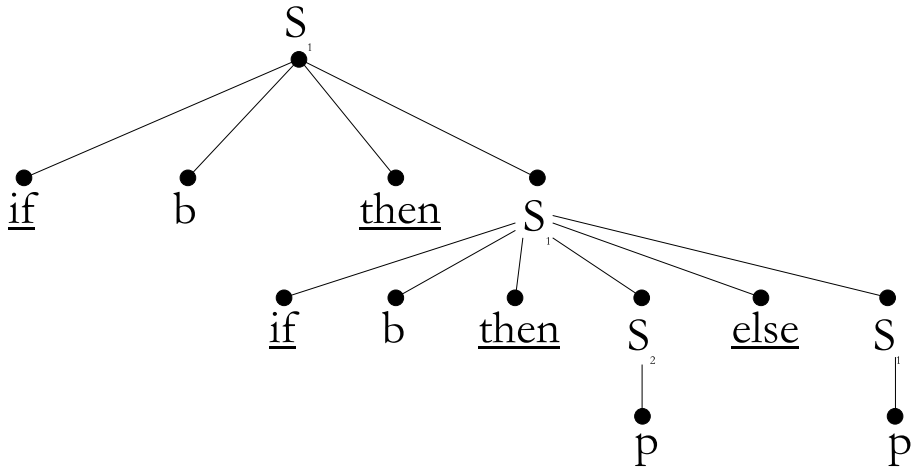
$$\texttt{if}\ b\ \texttt{then if}\ b\ \texttt{then}\ p\ \texttt{else}\ p$$

existují dva různé derivační stromy. Zatím, co Algol 60 tuto konstrukci nedovoluje (za `then` musí být složený příkaz), jazyk Pascal uvedenou víceznačnost řeší sémantickým pravidlem: „k danému

`else` se vztahuje nejbližší předchozí `then`". Poznamenejme, že i toto pravidlo lze postihnout také syntakticky:

$$\begin{aligned} S_1 &\rightarrow \texttt{if}\ b\ \texttt{then}\ S_1 \mid \texttt{if}\ b\ \texttt{then}\ S_2\ \texttt{else}\ S_1 \mid p \\ S_2 &\rightarrow \texttt{if}\ b\ \texttt{then}\ S_2\ \texttt{else}\ S_2 \mid p \end{aligned}$$

S použitím těchto přepisovacích pravidel obdržíme pro větu `if` b `then if` b `then` p `else` p jediný derivační strom na obr. 4.9.



Obrázek 4.9: Derivační strom podmíněného příkazu

4.4 Rozklad věty

Konstrukci derivace či derivačního stromu pro danou větu nebo větnou formu nazýváme *rozkladem* nebo *syntaktickou analýzou* této věty nebo větné formy. Program, který provádí rozklad vět určitého jazyka, se nazývá *syntaktický analyzátor* (anglicky také parser, to parse = rozložit).

Algoritmy syntaktické analýzy lze rozdělit podle způsobu, kterým je konstruována derivace věty, tj. vytvářen derivační strom, do těchto dvou základních skupin:

- syntaktická analýza shora dolů
- syntaktická analýza zdola nahoru

Při syntaktické analýze shora dolů začínáme derivační strom budovat od výchozího symbolu (kořene derivačního stromu) a postupnými přímými derivacemi dojdeme k terminálním symbolům, které tvoří analyzovanou větu (koncovým uzlům derivačního stromu). Problém spočívá ve správnosti volby přímých derivací, tj. pořadí používání přepisovacích pravidel.

Při syntaktické analýze zdola nahoru začínáme derivační strom budovat od koncových uzlů a postupnými přímými redukcemi dojdeme ke kořenu (výchozímu symbolu gramatiky).

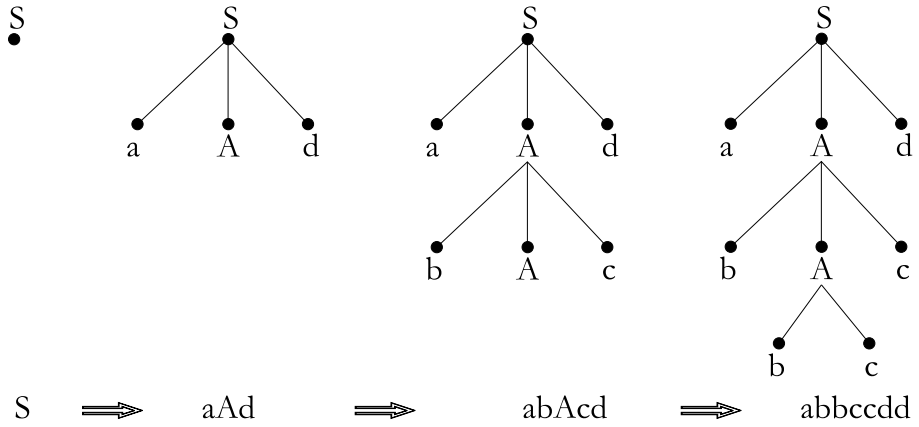
Základním problémem této třídy syntaktických analyzátorů je hledání prvního podřetězce věty (v dalších krocích větných forem), který může být redukován k jistému nonterminálu – kořenu podstromu derivačního stromu. Tento podřetězec, jak již víme, se nazývá l-fráze.

Příklad 4.10 Myšlenku obou typů syntaktické analýzy ilustrujeme na příkladě gramatiky, která generuje jazyk $L = \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$. Tato gramatika má pravidla

$$\begin{aligned} S &\rightarrow aSd \mid aAd \\ A &\rightarrow bAc \mid bc \end{aligned}$$

kde S je výchozí symbol.

Na obr. 4.10 je uvedena konstrukce derivačního stromu metodou shora dolů spolu s odpovídající levou derivací věty *abbccd*. Na obr. 4.11 je znázorněna konstrukce derivačního stromu metodou



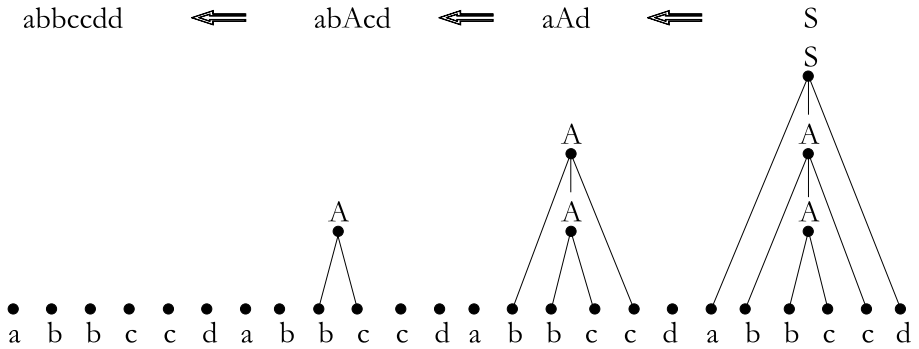
Obrázek 4.10: Syntaktická analýza shora dolů

zdola nahoru. Odpovídající pravá derivace je zapsána zprava doleva.

Vraťme se opět k základním problémům syntaktické analýzy. Při analýze shora dolů konstruujeme levou derivaci věty. Předpokládejme, že v jistém kroku analýzy je A nejlevějším nonterminálem, který má být přepsán. Dále předpokládejme, že gramatika obsahuje n pravidel s levou stranou A:

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

Jak poznáme, kterým řetězcem α_i je třeba nahradit nonterminál A? Podobně při analýze zdola



Obrázek 4.11: Syntaktická analýza zdola nahoru

nahoru, kdy je v každém kroku redukována l-fráze větné formy, spočívá hlavní problém v určení začátku a konce l-fráze.

Jedním z řešení těchto problémů je náhodný výběr některé z možných alternativ. Ukáže-li se později, že zvolená alternativa nebyla správná, je třeba proces rozkladu „vrátit“ a uvažovat jinou alternativu. (Při syntaktické analýze shora dolů se například pokoušíme přepisovat A řetězci $\alpha_1, \alpha_2 \ldots$ tak, aby se prefix získané levé derivace, který obsahuje pouze terminální symboly, shodoval s prefixem analyzované věty). Tento typ analýzy se nazývá *syntaktická analýza s návratem* (Syntax analysis with backup). I když počet návratů je omezený, je patrné, že analýza s návratem je časově náročná. Kromě toho jsou návraty zdrojem komplikací při sémantickém vyhodnocování překládaného programu.

Praktickým řešením problémů syntaktické analýzy programovacích jazyků jsou tzv. deterministické gramatiky (kapitola 6.), které umožňují na základě kontextu zpracovávaného podřetězce větné formy, určit správnou alternativu v každém kroku analýzy. Tento typ syntaktické analýzy se nazývá *deterministická syntaktická analýza* (deterministický rozklad) nebo *syntaktická analýza bez návratu*. Pro ilustraci role kontextu uvažujme větu $i + i * i$ v gramatice z příkladu 4.3. Po zpracování podřetězce $i + i$ nám kontext reprezentovaný terminálem $*$ pomůže určit frázi, kterou není $i + i$, ale podřetězec $i * i$.

4.5 Transformace bezkontextových gramatik

Obecně neexistuje algoritmická metoda, která by umožňovala sestrojit gramatiku generující jazyk s libovolnou strukturou. Existuje však celá řada užitečných transformací gramatik, které dovolují modifikovat gramatiku, aniž by byl porušen generovaný jazyk. V tomto odstavci popíšeme některé z těchto transformací formou matematických zápisů příslušných algoritmů.

Definice 4.6 Říkáme, že gramatiky G_1 a G_2 jsou *ekvivalentní*, jestliže platí $L(G_1) = L(G_2)$, tj. jestliže jimi generované jazyky jsou totožné.

V některých případech se může stát, že sestrojená gramatika obsahuje zbytečné symboly a nadbytečná přepisovací pravidla. Nehledě k tomu, že tato skutečnost může být důsledkem chyby v konstrukci gramatiky, je velmi pravděpodobné, že syntaktická analýza bude probíhat méně efektivně. Je proto důležité odstranit z gramatiky zbytečné symboly a nadbytečná pravidla.

Uvažujme například gramatiku $G = (\{S, A\}, \{a, b\}, P, S)$, kde $P = \{S \to a, A \to b\}$. Je zřejmé, že nonterminál A a terminál b se nemohou objevit v žádné větné formě. Proto je můžeme z gramatiky G odstranit spolu s nadbytečným pravidlem $A \to b$, aniž se změní $L(G)$.

Definice 4.7 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Říkáme, že symbol $X \in (N \cup \Sigma)$ je v gramatice G *zbytečný*, jestliže neexistuje derivace tvaru $S \Rightarrow^* wXy \Rightarrow^* wxy$. Řetězce w, x, y jsou v Σ^*.

Abychom zjistili, zda nonterminál A je zbytečný, popíšeme algoritmus určující, může-li se z nonterminálu A generovat řetězec terminálních symbolů, tj. zjišťujeme, zda $\{w \mid A \Rightarrow^* w,\ w \in \Sigma^*\} = \emptyset$

Tento algoritmus lze formulovat jako algoritmus, který zjišťuje, je-li jazyk, generovaný bezkontextovou gramatikou, neprázdný.

Algoritmus 4.1 Je $L(G)$ neprázdný?

Vstup: gramatika $G = (N, \Sigma, P, S)$.

Výstup: ANO je-li $L(G) \neq \emptyset$, NE v opačném případě.

Metoda: Sestrojíme množiny $N_0, N_1, \ldots$ rekurzivně takto

(1) $N_0 = \emptyset, i = 1$

(2) $N_i = \{A \mid A \to \alpha$ je v $P \wedge \alpha \in (N_{i-1} \cup \Sigma)^*\} \cup N_{i-1}$

(3) Je-li $N_i \neq N_{i-1}$, polož $i = i + 1$ a vrať se ke kroku 2. Je-li $N_i = N_{i-1}$, polož $N_t = N_i$

(4) Jestliže výchozí symbol S je v N_t, pak je výstup ANO, jinak NE.

Poznámka 4.3 Jestliže množina nonterminálů N má n prvků, pak, protože $N_t \subseteq N$, musí algoritmus 4.1 skončit maximálně po $n + 1$ iteracích kroku 2.

Věta 4.1 Algoritmus 4.1 má výstup ANO, právě když $S \Rightarrow^* w$ pro nějaké $w \in \Sigma^*$ Důkaz: Nejprve dokážeme indukcí pro i implikaci:

(1) Jestliže platí $A \in N_i$, pak $A \overset{*}{\Rightarrow} w$ pro nějaké $w \in \Sigma^*$

Pro $i = 0$ implikace platí, protože $N_0 = \emptyset$. Předpokládejme, že (1) platí pro i, a že A je prvkem množiny N_{i+1}. Je-li zároveň $A \in N_i$, pak je induktivní krok triviální. Jestliže A leží v $N_{i+1} - N_i$, pak existuje pravidlo $A \to X_1 \ldots X_k$, kde X_j je buď terminální symbol, nebo $X_j \in N_i$, $j = 1, \ldots, k$. Pro každé j tedy existuje řetězec $w_j \in \Sigma^*$ takový, že $X_j \Rightarrow^* w_j$ a platí tak

$$A \Rightarrow X_1 \ldots X_k \overset{*}{\Rightarrow} w_1 X_2 \ldots X_k \overset{*}{\Rightarrow} \ldots \overset{*}{\Rightarrow} w_1 \ldots w_k = w$$

Opačnou implikaci:

(2) Jestliže $A \overset{n}{\Rightarrow} w$, pak $A \in N_i$ pro nějaké i

dokážeme opět indukcí.

Je-li $n = 1$, pak zřejmě $i = 1$. Předpokládejme, že (2) platí pro n, a že $A \Rightarrow^{n+1} w$. Pak můžeme psát $A \Rightarrow X_1 \ldots X_k \Rightarrow^n w$, kde $w = w_1 \ldots w_k$ a $X_j \Rightarrow^{n_j} w_j$ pro $j = 1, \ldots, k$, $n_j \leq n$. (Podřetězce w_j jsou fráze řetězce w vzhledem k nonterminálům X_j v případě, že $X_j \in N$, pak $X_j \in N_{i_j}$ pro nějaké i_j. Položme $i_j = 0$, jestliže $X_j \in \Sigma$. Nechť $i = 1 + max(i_1, \ldots, i_k)$. Pak, podle definice, $A \in N_i$. Položíme-li nyní $A = S$ v implikacích (1) a (2), dostáváme důkaz věty 4.1. □

Definice 4.8 Říkáme, že symbol $X \in (N \cup \Sigma)$ je *nedostupný* v gramatice $G = (N, \Sigma, P, S)$, jestliže X se nemůže objevit v žádné větné formě.

Algoritmus 4.2 Odstranění nedostupných symbolů

Vstup: Gramatika $G = (N, \Sigma, P, S)$.

Výstup: Gramatika $G' = (N', \Sigma', P', S)$, pro kterou platí

(i) $L(G') = L(G)$

(ii) Pro všechna X z $(N' \cup \Sigma')$ existují řetězce α a β z $(N' \cup \Sigma')^*$ tak, že $S \Rightarrow \alpha X \beta$ v gramatice G'.

Metoda:

(1) Položíme $V_0 = \{S\}$ a $i = 1$

(2) Konstruujeme $V_i = \{X \mid A \to \alpha X \beta \in P \wedge A \in V_{i-1}\} \cup V_{i-1}$

(3) Je-li $V_i \neq V_{i-1}$, polož $i = i + 1$ a vrať se ke kroku 2.
Je-li $V_i = V_{i-1}$, pak
$N' = V_i \cap N$
$\Sigma' = V_i \cap \Sigma$
$P' \subseteq P$ obsahuje ta pravidla, která jsou tvořena pouze symboly z V_i.

Algoritmus 3.2 je podobný algoritmu 4.1. Protože je $V_i \subset N \cup \Sigma$, je počet opakování bodu (2) algoritmu 4.2 konečný. Důkaz algoritmu se provede indukcí pro i této ekvivalence:

$$S \overset{*}{\Rightarrow} \alpha X \beta, \text{ právě když } X \in V_i \text{ pro nějaké } i.$$

Nyní zformulujeme algoritmus pro odstranění zbytečných symbolů.

Algoritmus 4.3 Odstranění zbytečných symbolů.

Vstup: Gramatika $G = (N, \Sigma, P, S)$ generující neprázdný jazyk.

Výstup: Gramatika $G' = (N', \Sigma', P', S)$, pro kterou platí:

(i) $L(G) = L(G')$
(ii) Žádný symbol v $N' \cup \Sigma$' není zbytečný

Metoda:

(1) Na gramatiku G aplikuj algoritmus 4.1 s cílem získat množinu N_t. Polož $\overline{G} = (N \cap N_t, \Sigma, P_1, S)$, kde P_1 obsahuje pravidla tvořené pouze symboly z $N_t \cup \Sigma$
(2) Algoritmus 4.2 aplikuj na gramatiku $\overline{G}$. Výsledkem je gramatika $G' = (N', \Sigma', P', S)$, která neobsahuje zbytečné symboly.

V kroku (1) algoritmu 4.3 jsou z G odstraněny všechny nonterminály, které nemohou generovat terminální řetězce. V kroku (2) jsou pak odstraněny všechny symboly, které jsou nedostupné. Obrácené pořadí použití algoritmů 4.1 a 4.2 nemusí vždy vést ke gramatice bez zbytečných symbolů.

Věta 4.2 Gramatika G' z algoritmu 4.3 je ekvivalentní gramatice G a nemá zbytečné symboly.

Důkaz: Dokážeme sporem, že gramatika G' nemá zbytečné symboly. Předpokládejme, že $A \in N'$ je zbytečný symbol. Podle definice zbytečných symbolů musíme uvažovat dva případy:

1. Neplatí $S \Rightarrow^*_{G'} \alpha A \beta$ pro všechna α a β. V tomto případě se však dostáváme do sporu s krokem (2) algoritmu 4.3.

2. Platí $S \Rightarrow^*_{G'} \alpha A \beta$ pro nějaká α a β, avšak neplatí $A \Rightarrow^*_{G'} w, w \in \Sigma'^*$. Pak A nebude odstraněn v kroku (2) algoritmu 4.3 a navíc, platí-li $A \Rightarrow^*_G \gamma B \delta$, pak ani B nebude odstraněn v kroku (2). Platí-li však $A \Rightarrow^*_G w$, pak také platí $A \Rightarrow^*_{G'} w$ a tedy neplatí-li $A \Rightarrow^*_{G'} w$, pak neplatí ani $A \Rightarrow^*_G w$, což je spor s krokem (1) algoritmu 4.3.

Důkaz, že G' neobsahuje ani zbytečný terminál se provede obdobně. □

Příklad 4.11 Uvažujme gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$$\begin{aligned} S &\rightarrow a \mid A \\ A &\rightarrow AB \\ B &\rightarrow b \end{aligned}$$

Aplikujeme-li na G algoritmus 4.3, pak v kroku 1 získáme $N_t = \{S, B\}$, takže $\overline{G} = (\{S, B\}, \{a, b\}, \{S \rightarrow a, B \rightarrow b\}, S\})$. Po aplikování algoritmu 4.2 na $\overline{G}$ obdržíme $V_2 = V_1 = \{S, a\}$. Výsledkem je tedy ekvivalentní gramatika

$$G' = (\{S\}, \{a\}, \{S \rightarrow a\}, S).$$

Kdybychom jako první aplikovali algoritmus 4.2, zjistíme, že všechny symboly v G jsou dostupné a algoritmus 4.2 tedy gramatiku G nezmění. Po aplikování algoritmu 4.1 získáme $N_t = \{S, B\}$, takže výsledná gramatika bude $\overline{G}$ a ne G'.

Další důležitou transformací, kterou nyní popíšeme, je transformace odstraňující z gramatiky pravidla tvaru $A \rightarrow \epsilon$ (ϵ je prázdný řetězec), tzv. ϵ-pravidla. Jestliže ovšem $L(G)$ má obsahovat také prázdný řetězec, pak není možné aby G neobsahovala žádné ϵ-pravidlo. Následující definice gramatiky bez ϵ-pravidla respektuje tuto skutečnost.

Definice 4.9 Říkáme, že gramatika $G = (N, \Sigma, P, S)$ je *gramatikou bez ϵ-pravidel*, jestliže buď P neobsahuje žádné ϵ-pravidlo, nebo existuje jediné ϵ-pravidlo tvaru $S \rightarrow \epsilon$ a výchozí symbol S se nevyskytuje na pravé straně žádného pravidla z P.

Algoritmus 4.4 Transformace na gramatiku bez ϵ-pravidel.

Vstup: Gramatika $G = (N, \Sigma, P, S)$

Výstup: Ekvivalentní gramatika $G' = (N', \Sigma', P', S')$ bez ϵ-pravidel.

Metoda:

(1) Sestroj $N_\epsilon = \{A \mid A \in N \text{ a } A \Rightarrow^* \epsilon\}$. Konstrukce množiny N_ϵ je analogická konstrukci N_t z 4.1.

(2) Nechť P' je množina pravidel, kterou konstruujeme takto:

a) Jestliže $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \ldots B_k \alpha_k$ je v $P, k \geq 0$ a každé B_i je v N_ϵ, $1 \leq i \leq k$, avšak žádný ze symbolů řetězců α_j není v N_ϵ, $0 \leq j \leq k$, pak k P' přidej všechna pravidla tvaru

$$A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \ldots X_k \alpha_k$$

kde X_i je buď B_i nebo ϵ. Nepřidávej ϵ-pravidlo $A \rightarrow \epsilon$, které se objeví, jsou-li všechna $\alpha_i = \epsilon$.

b) Jestliže $S \in N_\epsilon$ pak k P' přidej pravidla

$$S' \rightarrow \epsilon \mid S$$

S' je nový výchozí symbol. Polož $N' = N \cup \{S'\}$ Jestliže $S \notin N_\epsilon$, pak $N' = N$ a $S' = S$

(3) Výsledná gramatika má tvar $G' = (N', \Sigma', P', S')$

Příklad 4.12 Uvažujme gramatiku $G = (\{S\}, \{a, b\}, P, S)$, kde P obsahuje pravidla $S \to aSbS \mid bSaS \mid \epsilon$. Po aplikování algoritmu 4.4 získáme gramatiku G' bez ϵ-pravidel. $G' = (\{S', S\}, \{a, b\}, P', S')$, kde P' obsahuje pravidla

$$\begin{aligned} S' &\to S \mid \epsilon \\ S &\to aSbS \mid bSaS \mid aSb \mid abS \mid ab \mid bSa \mid baS \mid ba \end{aligned}$$

Věta 4.3 Algoritmus 4.3 převádí vstupní gramatiku G na ekvivalentní gramatiku G' bez ϵ-prav. Důkaz: G' zřejmě neobsahuje ϵ-pravidla kromě případného pravidla $S \to \epsilon$, je-li $\epsilon \in L(G)$. Důkaz, že $L(G) = L(G')$ se provede indukcí pro délku řetězce w v ekvivalenci

$$A \underset{G'}{\overset{*}{\Rightarrow}} w \text{ právě když } w \neq \epsilon \wedge A \underset{G}{\overset{*}{\Rightarrow}} w$$

Jinou užitečnou transformací je odstranění pravidel tvaru $A \to B$. □

Definice 4.10 Přepisovací pravidlo tvaru $A \to B, A, B \in N$ se nazývá *jednoduché pravidlo.*

Algoritmus 4.5 Odstranění jednoduchých pravidel.

Vstup: Gramatika G bez ϵ-pravidel.

Výstup: Ekvivalentní gramatika G' bez jednoduchých pravidel.

Metoda:

(1) Pro každé $A \in N$ sestroj množinu $N_A = \{B \mid A \Rightarrow^* B\}$ takto:
   a) $N_0 = \{A\}, i = 1$
   b) $N_i = \{C \mid B \to C$ je v P a $B \in N_{i-1}\} \cup N_{i-1}$
   c) Jestliže $N_i \neq N_{i-1}$, polož $i = i + 1$ a opakuj krok b). V opačném případě je $N_A = N_i$.

(2) Sestroj P' takto: Jestliže $B \to \alpha$ je v P a není jednoduchým pravidlem, pak pro všechna A, pro něž $B \in N_A$, přidej k P' pravidla $A \to \alpha$

(3) Výsledná gramatika je $G = (N, \Sigma, P', S)$

Příklad 4.13 Uvažujme gramatiku s přepisovacími pravidly:

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

Tato gramatika se liší od gramatiky z příkladu 4.3 pouze jiným značením nonterminálu a generuje tudíž stejný jazyk aritmetických výrazů

Po aplikování algoritmu 4.5 bude

$$\begin{aligned} N_E &= \{E, T, F\} \\ N_T &= \{T, F\} \\ N_F &= \{F\} \end{aligned}$$

a výsledná množina přepisovacích pravidel, neobsahující jednoduchá pravidla, bude:

$$\begin{aligned} E &\to E + T \mid T * F \mid (E) \mid i \\ T &\to T * F \mid (E) \mid i \\ F &\to (E) \mid i \end{aligned}$$

Věta 4.4 Algoritmus 4.4 převádí vstupní gramatiku G na ekvivalentní gramatiku G' bez jednoduchých pravidel. Důkaz: Gramatika G' zřejmě neobsahuje jednoduchá pravidla. Abychom ukázali, že $L(G) = L(G')$, dokážeme že platí $L(G') \subseteq L(G)$ a také $L(G) \subseteq L(G')$.

1. $L(G') \subseteq L(G)$

   Nechť $w \in L(G')$. Pak existuje v G' derivace $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$.

   Jestliže k derivaci $\alpha_i \Rightarrow_{G'} \alpha_{i+1}$ bylo použito pravidla $A \to \beta$, pak existuje nonterminál B ($A = B$ případně) takový, že $A \Rightarrow^*_G B$ a $B \Rightarrow_G \beta$ a tedy $A \Rightarrow^*_G \beta$ a $\alpha_i \Rightarrow^*_G \alpha_{i+1}$. Z toho plyne, že platí $S \Rightarrow^*_G w$ a w je tudíž v $L(G)$.

2. $L(G) \subseteq L(G')$

   Nechť $w \in L(G)$ a $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$ je levá derivace řetězce w v gramatice G. Nechť $i_1, i_2, \ldots i_k$ je posloupnost indexů tvořená pouze těmi j, pro které v levé derivaci $\alpha_{j-1} \Rightarrow \alpha_j$ nebylo použito jednoduchého pravidla. Speciálně $i_k = n$, protože poslední aplikované pravidlo v derivaci řetězce w nemůže být jednoduché. Derivace řetězce w v G je levou derivací, a proto aplikací jednoduchých pravidel nahrazujeme pouze nejlevější nonterminál jiným nonterminálem (reprezentuje levou stranu pravidla v G'). Platí tedy $S \Rightarrow_{G'} \alpha_{i_1} \Rightarrow_{G'} \ldots \Rightarrow_{G'} \alpha_{i_1} = w$ a tudíž $w \in L(G')$.

Dokázali jsme tak ekvivalenci gramatik G a G'. □

Definice 4.11 Říkáme, že G je *gramatika bez cyklu*, jestliže v ní neexistuje derivace tvaru $A \Rightarrow^+ A$ pro žádné A z N. Jestliže G je gramatika bez cyklu a bez ϵ-pravidel a nemá žádné zbytečné symboly, pak říkáme, že G je *vlastní gramatika*.

Věta 4.5 Je-li L bezkontextový jazyk, pak existuje vlastní gramatika G taková, že $L = L(G)$. Důkaz: Vyplývá z algoritmů 4.1–4.5. Gramatika, jež neobsahuje ϵ-pravidla a jednoduchá pravidla, je zřejmě gramatikou bez cyklu. Existence ϵ-pravidel a jednoduchých pravidel, na druhé straně, neimplikuje existenci cyklu (tj. derivace tvaru $A \Rightarrow^+ A$ pro nějaké $A \in N$). □

Poznámka 4.4 Gramatiky, které obsahují ϵ-pravidla a cykly, komplikují obvykle algoritmy rozkladu. Většina efektivních syntaktických analyzátorů je konstruována pro vlastní gramatiku.

Další transformací, kterou uvedeme, je odstranění levé rekurze. Tato transformace je důležitá pro použití analýzy typu shora-dolů.

Definice 4.12 Nechť $G = (N, \Sigma, P, S)$. Přepisovací pravidlo z P se nazývá *rekurzi zleva (rekurzivní zprava)*, jestliže je tvaru $A \to A\alpha (A \to \alpha A), A \in N, \alpha \in (N \cup \Sigma)^*$. Jestliže v G existuje derivace $A \Rightarrow^+ \alpha A \beta$, pro nějaké $A \in N$, říkáme, že gramatika G je *rekurzivní*. Je-li $\alpha = \epsilon$, pak mluvíme o *gramatice rekurzivní zleva*, je-li $\beta = \epsilon$, pak říkáme, že G je *rekurzivní zprava*.

Poznamenejme, že je-li jazyk $L(G)$ nekonečný, pak G musí být rekurzivní.

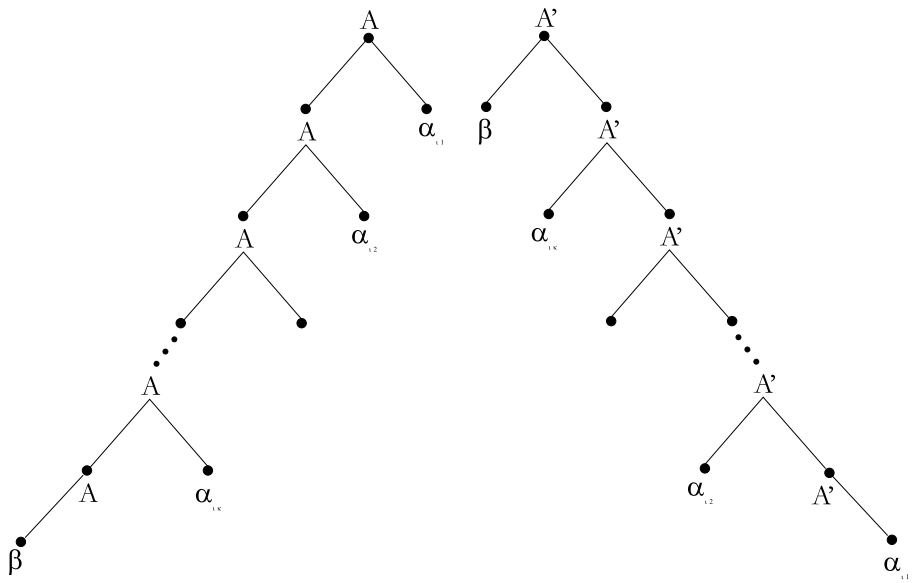
Dříve, než zformujeme algoritmus eliminující levou rekurzi, ukážeme, jakým způsobem lze odstranit přepisovací pravidla rekurzivní zleva.

Definice 4.13 Pravidlo $A \to \alpha$, s levou stranou tvořenou nonterminálem A, budeme nazývat A-pravidlo (Neplést s ϵ-pravidlem.)

Věta 4.6 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ jsou všechna A-pravidla. Žádný z řetězců β_i nezačíná neterminálem A. Gramatika $G' = (N \cup \{A'\}, \Sigma, P', S)$, kde P' obsahuje namísto uvedených pravidel pravidla

$$\begin{aligned} A &\to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \mid \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A' \\ A' &\to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m \mid \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \end{aligned}$$

je ekvivalentní s gramatikou G, tj. $L(G) = L(G')$. Důkaz: Uvedena transformace nahrazuje pravidla rekurzivní zleva pravidly, která jsou rekurzivní zprava. Označíme-li jazyky $L_1 = \{\beta_1, \beta_2, \ldots, \beta_n\}$ a $L_2 = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ vidíme, že v G lze z neterminálu A derivovat řetězce tvořící jazyk $L_1 L_2^*$. Právě tyto řetězce můžeme však derivovat z A také v gramatice G'. Efekt popisované transformace ilustruje obrázek 4.12. □



Obrázek 4.12: Odstranění levé rekurze: nalevo část stromu v G, napravo část stromu v G'

Příklad 4.14 Uvažujme gramatiku G z předcházejícího příkladu s pravidly

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

Po odstranění pravidel rekurzivních zleva získáme ekvivalentní gramatiku G' s pravidly

$$\begin{aligned} E &\to T \mid TE' \\ E' &\to +T \mid +TE' \\ T &\to F \mid FT' \\ T' &\to *F \mid *FT' \\ F &\to (E) \mid i \end{aligned}$$

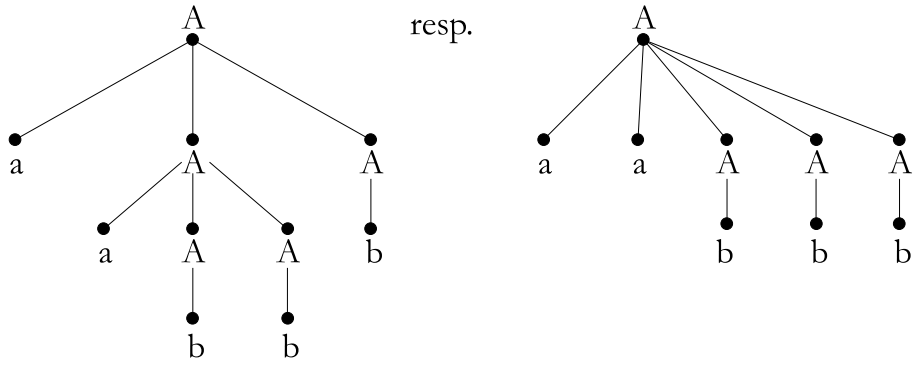
Věta 4.7 Nechť $G = (N, \Sigma, P, S)$ je gramatika. $A \to \alpha B \beta,\ B \in N; \alpha, \beta \in (N \cup \Sigma)^*$ je pravidlo z P a $B \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$ jsou všechna B-pravidla v P. Nechť $G' = (N, \Sigma, P', S)$, kde

$$P' = P - \{A \to \alpha B \beta\} \cup \{A \to \alpha\gamma_1\beta \mid \alpha\gamma_2\beta \mid \ldots \mid \alpha\gamma_n\beta\}.$$

Pak $L(G) = L(G')$. Důkaz: Výsledkem této transformace je odstranění pravidla $A \to \alpha B \beta$ z gramatiky G tím způsobem, že za nonterminál B „dosadíme“ všechny pravé strany B-pravidel. Formálně se důkaz provede tak, že se ukáže platnost konjunkce $L(G) \subseteq L(G) \wedge L(G') \subseteq L(G)$. Efekt této transformace ilustruje obr. 4.13, jenž znázorňuje derivační stromy řetězce *aabbb* v gramatice G resp. G'. Gramatika G obsahuje pravidlo $A \to aAA$ a dvě A-pravidla $A \to aAA \mid b$. Položíme-li $\alpha = a, B = A, \beta = A$, můžeme eliminovat pravidlo $A \to aAA$ zavedením těchto A-pravidel v gramatice G':

$$A \to aaAAA \mid abA \mid b$$

□



Obrázek 4.13: Derivační stromy věty *aabbb* v G resp. G'

Algoritmus 4.6 Odstranění levé rekurze.

Vstup: Vlastní gramatika $G = (N, \Sigma, P, S)$

Výstup: Gramatika G' bez levé rekurze.

Metoda:

(1) Nechť $N = \{A_1, A_2, \ldots, A_n\}$. Gramatiku budeme transformovat tak, že je-li $A_i \to \alpha$ pravidlo, pak α začíná buď terminálem, nebo nonterminálem $A_j,\ j > i$. K tomu účelu položme $i = 1$.

(2) Nechť $A_i \to A_i\alpha_1 \mid \ldots \mid A_i\alpha_m \mid \beta_1 \mid \ldots \mid \beta_p$ jsou všechna A_i-pravidla a nechť žádné β_i nezačíná nonterminálem A_k, je-li $k \leq i$.
Nahraď všechna A_i-pravidla těmito pravily:

$$\begin{aligned} A_i &\to \beta_1 \mid \ldots \mid \beta_p \mid \beta_1 A_i' \mid \ldots \mid \beta_p A_i' \\ A_i' &\to \alpha_1 \mid \ldots \mid \alpha_m \mid \alpha_1 A_i' \mid \ldots \mid \alpha_m A_i' \end{aligned}$$

kde A_i' je nový nonterminál. Takto všechna A_i-pravidla začínají buď terminálem, nebo nonterminálem $A_k, k > i$.

(3) Je-li $i = n$, pak jsme získali výslednou gramatiku G'. V opačném případě polož $i = i + 1$ a $j = 1$.

(4) Každé pravidlo tvaru $A_i \to A_j\alpha$ nahraď pravidly $A_i \to \beta_1\alpha \mid \ldots \mid \beta_p\alpha$ kde $A_j \to \beta_1 \mid \ldots \mid \beta_p$ jsou všechna A_j-pravidla. Po této transformaci budou všechna A_j-pravidla začínat buď terminálem, nebo nonterminálem $A_k, k > j$, takže také všechna A_i-pravidla budou mít tuto vlastnost.

(5) Je-li $j = i - 1$, pak přejdi ke kroku (2). Jinak $j = j + 1$ a opakuj krok (4).

Věta 4.8 Každý bezkontextový jazyk lze generovat gramatikou bez levé rekurze. Důkaz: Nechť G je vlastní gramatika. Algoritmus 4.6 používá pouze transformací z vět 4.6 a 4.7 a tudíž je $L(G) = L(G')$.

Dále musíme dokázat, že G' není gramatikou rekurzivní zleva. Formální důkaz nebudeme provádět (viz [**?**]). Uvědomíme si však, že krok (2) odstraňuje pravidla rekurzivní zleva. Stejně tak pravidla vzniklá aplikací kroku (4) nejsou rekurzivní zleva. Protože krok (2) je aplikován na všechny nonterminální symboly, nemůže výsledná gramatika obsahovat pravidla rekurzivní zleva. □

Příklad 4.15 Nechť G je gramatika s pravidly:

$$\begin{aligned} A &\to BC \mid a \\ B &\to CA \mid Ab \\ C &\to AB \mid CC \mid a \end{aligned}$$

Položme $A_1 = A, A_2 = B$ a $A_3 = C$. V každém kroku uvedeme pouze nová pravidla pro ty nonterminály, jejichž pravidla se mění.

krok (2),	i=1,	beze změny
krok (4),	i=2, j=1,	$B \to CA \mid BCb \mid ab$
krok (2),	i=2,	$B \to CA \mid ab \mid CAB' \mid abB'$
		$B' \to CbB' \mid Cb$
krok (4),	i=3, j=1,	$C \to BCB \mid aB \mid CC \mid a$
krok (4),	i=3, j=2,	$C \to CACB \mid abCB \mid CAB'CB \mid abB'CB \mid aB \mid CC \mid a$
krok (2),	i=3,	$C \to abCB \mid abB'CB \mid aB \mid a \mid abCBC' \mid abB'CBC' \mid aBC' \mid aC'$
		$C' \to ACBC' \mid AB'CBC' \mid CC' \mid ACB \mid AB'CB \mid C$

4.6 Chomského normální forma

Definice 4.14 Gramatika $G = (N, \Sigma, P, S)$ je v *Chomského normální formě* (CNF), jestliže každé pravidlo z P má jeden z těchto tvarů:

(1) $A \to BC, A, B, C \in N$ nebo

(2) $A \to a$, $a \in \Sigma$ nebo

(3) Je-li $\epsilon \in L(G)$, pak $S \to \epsilon$ je pravidlo z P a výchozí symbol S se neobjeví na pravé straně žádného pravidla.

Chomského normální forma gramatiky je prostředkem zjednodušujícím tvar reprezentace bezkontextového jazyka. Nyní popíšeme algoritmus převodu obecné bezkontextové gramatiky do Chomského normální formy.

Algoritmus 4.7 Převod do Chomského normální formy.

Vstup: Vlastní gramatika $G = (N, \Sigma, P, S)$ bez jednoduchých pravidel.

Výstup: Gramatika $G' = (N', \Sigma, P', S')$ v CNF taková, že $L(G) = L(G')$

Metoda: Z gramatiky G získáme ekvivalentní gramatiku G' v CNF takto:

(1) Množina pravidel P' obsahuje všechna pravidla tvaru $A \to a$ z P

(2) Množina pravidel P' obsahuje všechna pravidla tvaru $A \to BC$ z P

(3) Je-li pravidlo $S \to \epsilon$ v P, pak $S \to \epsilon$ je také v P'

(4) Pro každé pravidlo tvaru $A \to X_1 \dots X_k$, kde $k > 2$ z P přidej k P' tuto množinu pravidel. Symbolem X_i' značíme nonterminál X_i, je-li $X_i \in N$, nebo nový nonterminál, je-li $X_i \in \Sigma$:

$$\begin{aligned} A &\to X_1' \langle X_2 \dots X_k \rangle \\ \langle X_2 \dots X_k \rangle &\to X_2' \langle X_3 \dots X_k \rangle \\ &\vdots \\ \langle X_{k-1} X_k \rangle &\to X_{k-1}' X_k' \end{aligned}$$

kde každý symbol $\langle X_i \dots X_k \rangle$ značí nový nonterminální symbol.

(5) Pro každé pravidlo tvaru $A \to X_1 X_2$, kde některý ze symbolů X_1 nebo X_2 leží v Σ přidej k P' pravidlo $A \to X_1' X_2'$.

(6) Pro každý nový nonterminál tvaru a' přidej k P' pravidlo $a' \to a$. Výsledná gramatika je $G' = (N', \Sigma, P', S')$; množina N' obsahuje všechny nonterminály tvaru $\langle X_i \dots X_k \rangle$ a a'.

Věta 4.9 Nechť L je bezkontextový jazyk. Pak existuje bezkontextová gramatika G v CNF taková, že $L = L(G)$. Důkaz: Podle věty 4.5 má jazyk L vlastní gramatiku G. Stačí tedy dokázat, že výsledkem algoritmu 4.7 je ekvivalentní gramatika G' tj. $L(G) = L(G')$. Tato ekvivalence však plyne bezprostředně z aplikace věty 4.7 na každé pravidlo z P', které má nonterminály tvaru a' a $\langle X_i ... X_j \rangle$. Výsledkem bude gramatika G'. □

Příklad 4.16 Nechť G je gramatika s pravidly

$$\begin{aligned} S &\to aAB \mid BA \\ A &\to BBB \mid a \\ B &\to AS \mid b \end{aligned}$$

Chomského normální forma $G' = (N', \{a, b\}, P', S)$ má pravidla:

$$\begin{aligned} S &\to a' \langle AB \rangle \mid BA \\ A &\to B \langle BB \rangle \mid a \\ B &\to AS \mid b \\ \langle AB \rangle &\to AB \\ \langle BB \rangle &\to BB \\ a' &\to a \end{aligned}$$

Nová množina nonterminálů je tedy $N' = \{S, A, B, \langle AB \rangle, \langle BB \rangle, a'\}$

Příklad 4.17 Předpokládejme, že gramatika G je v CNF, a že řetězci $w \in L(G)$ přísluší derivace v G délky p. Jaká je délka věty w? Řešení: Nechť $|w| = n$. Pak pro odvození věty bylo třeba n-krát aplikovat pravidlo tvaru $A \to a$ a (n-1)-krát pravidlo tvaru $A \to BC$. Dostáváme tedy vztah

$$p = n + n - 1 = 2n - 1$$

a vidíme, že p je vždy liché. Řešení příkladu je tedy

$$|w| = \frac{p+1}{2}$$

□

4.7 Greibachova normální forma

Definice 4.15 Gramatika $G = (N, \Sigma, P, S)$ je v Greibachové normální formě (GNF), je-li G gramatikou bez ϵ-pravidel a jestliže každé pravidlo (s výjimkou případného pravidla $S \to \epsilon$) má tvar $A \to a\alpha$ kde $a \in \Sigma$ a $\alpha \in N^*$.

Lemma 4.1 Nechť $G = (N, \Sigma, P, S)$ je gramatika bez levé rekurze. Pak existuje lineární uspořádání $<$ definované na množině nonterminálních symbolů N takové, že je-li $A \to B\alpha$ v P, pak $A < B$. Důkaz: Nechť R je relace na množině N taková, že ARB platí právě když $A \Rightarrow^+ B\alpha$ pro nějaké $\alpha \in (N \cup \Sigma)^*$. Z definice levé rekurze plyne, že R je částečné uspořádání (R je tranzitivní). Každé částečné uspořádání pak lze rozšířit na lineární uspořádání. □

Nyní zformulujeme algoritmus převodu gramatiky G do Greibachové normální formy.

Algoritmus 4.8 Převod do Greibachové normální formy.

Vstup: Vlastní gramatika bez levé rekurze $G = (N, \Sigma, P, S)$

Výstup: Ekvivalentní gramatika G' v GNF

Metoda:

(1) Podle lemmy 4.1 vytvoř lineární uspořádání $<$ na N takové, že každé A-pravidlo začíná buď terminálem, nebo nějakým nonterminálem B takovým, že $A < B$. Nechť

$$N = \{A_1, A_2, \ldots, A_n\} \text{ a } A_1 < A_2 < \ldots < A_n.$$

(2) Polož $i = n - 1$

(3) Je-li $i = 0$ přejdi k bodu (5), je-li $i \neq 0$ nahraď každé pravidlo tvaru $A_i \to A_j\alpha$, kde $j > i$ pravidly $A_i \to \beta_1\alpha \mid \ldots \mid \beta_m\alpha$, kde $A_j \to \beta_1 \mid \ldots \mid \beta_m$ jsou všechna A_j-pravidla. (Každý z řetězců $\beta_1 \ldots \beta_m$ začíná terminálem.)

(4) Polož $i = i - 1$ a opakuj krok (3).

(5) V tomto okamžiku všechna pravidla (s vyjímkou pravidla $S \to \epsilon$) začínají terminálním symbolem. V každém pravidle $A \to aX_1 \ldots X_k$ nahraď ty symboly X_j, které jsou terminálnímy symboly, novým nonterminálem X'_j.

(6) Pro všechna X'_j z bodu (5) přidej pravidla $X'_j \to X_j$

Věta 4.10 Nechť L je bezkontextový jazyk. Pak existuje gramatika G v GNF taková, že $L = L(G)$. Důkaz: Z definice uspořádání $<$ vyplývá, že všechna A_n-pravidla začínají terminály, a že po opakovaném provádění kroku (3) algoritmu 4.8 budou všechna A_i-pravidla pro $i = 1, \ldots, n-1$ začínat také terminálními symboly. Krok (5) a (6) převádí nepočáteční terminální symboly pravých stran na nonterminály, aniž se podle věty 4.10 mění generovaný jazyk. □

Příklad 4.18 Převeďme gramatiku G s pravidly

$$\begin{aligned} E &\to T \mid TE' \\ E' &\to +T \mid +TE' \\ T &\to F \mid FT' \\ T' &\to *F \mid *FT' \\ F &\to (E) \mid i \qquad \text{do GNF.} \end{aligned}$$

Řešení: Podle lemmy 4.1 je $E < T < F$. Jako lineární uspořádání na množině nonterminálů vezměme uspořádání

$$E' < E < T' < T < F$$

Všechna F-pravidla začínají terminálem (důsledek skutečnosti, že F je největší prvek v uspořádání $<$). Další největší symbol T má pravidla $T \to F \mid FT'$ a po aplikaci kroku (3) dostáváme pravidla $T \to (E) \mid i \mid (E)T' \mid iT'$.

Výsledná gramatika v GNF má pravidla:

$$\begin{aligned} E &\to (E)' \mid i \mid (E)'T' \mid iT' \mid (E)'E' \mid iE' \mid (E)'T'E' \mid iT'E' \\ E' &\to +T \mid +TE' \\ T &\to (E)' \mid i \mid (E)'T' \mid iT' \\ T' &\to *F \mid *FT' \\ F &\to (E)' \mid i \\ )' &\to) \end{aligned}$$

□

Nevýhodou algoritmu 4.8 je velké množství nových pravidel. Existuje alternativní algoritmus převodu do GNF, [**?**], který nezvětšuje tak výrazně počet pravidel gramatiky, avšak zavádí zase více nonterminálů.

4.8 Cvičení

Cvičení 4.8.1 Pro výpočet tranzitivního uzávěru binární relace definované na konečné množině (např. slovníku gramatiky) se velmi často používá reprezentace relace prostřednictvím booleovské matice a operací nad touto maticí. Je-li A booleovská matice reprezentující binární relaci R na množině M (tj. $A[p,q] = \text{true}$, je-li $(p,q) \in R$ a $A[p,q] = \text{false}$, je-li $(p,q) \notin R$, $p, q \in M$ a $A[p,q]$ je prvek matice A v řádku označeném p a sloupci označeném q), pak tranzitivní uzávěr relace R je relace R^+, jež je reprezentován maticí A^+:

(3) $$A^+ = A + A^2 + \ldots + A^n,$$

kde $n = \min(|M| - 1, |R|)$ a operace sečítání, resp. násobení jsou interpretovány jako disjunkce, resp. konjunkce.

Pro výpočet tranzitivního uzávěru existuje efektivnější postup než podle 3, nazývaný, podle autora, *Warshallův algoritmus*.

Algoritmus 4.9 Warshallův algoritmus pro výpočet tranzitivního uzávěru binární relace.

Vstup: Booleovská matice A reprezentující binární relaci R.

Výstup: Booleovská matice B reprezentující binární relaci R^+.

Metoda:

(1) Polož $B = A$ a $i = 1$.

(2) Pro všechna j, jestliže $B[j,i] = \text{true}$, pak pro $k = 1, \ldots, n$ polož $B[j,k] = B[j,k] + B[i,k]$.

(3) Polož $i = i + 1$.

(4) Je-li $i \leq n$, vrať se ke kroku (2); v opačném případě je B výsledná matice.

Ukažte, že algoritmus 4.9 počítá skutečně tranzitivní uzávěr binární relace.

Cvičení 4.8.2 V gramatice z příkladu 4.3 vytvořte levou a pravou derivaci a derivační strom věty $i * (i + i - i)$. Nalezněte všechny fráze, jednoduché fráze a l-frázi této věty.

Cvičení 4.8.3 Nechť $G = (N, \Sigma, P, S)$ je gramatika a α_1, α_2 řetězce $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$. Navrhněte algoritmus, který zjišťuje, zda platí $\alpha_1 \Rightarrow_G^* \alpha_2$.

Cvičení 4.8.4 Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, která má pravidla

$$\begin{aligned} S &\rightarrow bA \mid aB \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

Ukažte, že tato gramatika je víceznačná (uvažujte např. větu *aabbab*). Pokuste se nalézt ekvivalentní jednoznačnou gramatiku.

Cvičení 4.8.5 Gramatiku s pravidly

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aB \mid bS \mid b \\ B &\rightarrow AB \mid Ba \\ C &\rightarrow AS \mid b \end{aligned}$$

transformujte na ekvivalentní gramatiku bez zbytečných symbolů.

Cvičení 4.8.6 Nalezněte gramatiku bez ϵ-pravidel, jež je ekvivalencí s gramatikou

$$\begin{aligned} S &\rightarrow ABC \\ A &\rightarrow BB \mid \epsilon \\ B &\rightarrow CC \mid a \\ C &\rightarrow AA \mid b \end{aligned}$$

Cvičení 4.8.7 Ke gramatice

$$\begin{aligned} S &\to A \mid B \\ A &\to C \mid D \\ B &\to D \mid E \\ C &\to S \mid a \mid \epsilon \\ D &\to S \mid b \\ E &\to S \mid c \mid \epsilon \end{aligned}$$

nalezněte ekvivalentní vlastní gramatiku.

Cvičení 4.8.8 Formulujte algoritmus, který testuje, zda gramatika G je a nebo není gramatikou bez cyklu.

Cvičení 4.8.9 V gramatice

$$\begin{aligned} S &\to AB \\ A &\to BS \mid b \\ B &\to SA \mid a \end{aligned}$$

odstraňte levou rekurzi.

Cvičení 4.8.10 Do Chomského normální formy převeďte gramatiku

$$G = (\{S, T, L\}, \{a, b, +, -, *, /, [,]\}, P, S)$$

kde P obsahuje pravidla

$$\begin{aligned} S &\to T + S \mid T - S \mid T \\ T &\to L * T \mid L/T \mid L \\ L &\to [S] \mid a \mid b \end{aligned}$$

Cvičení 4.8.11 Gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$ s pravidly

$$\begin{aligned} S &\to Ba \mid Ab \\ A &\to Sa \mid AAb \mid a \\ B &\to Sb \mid BBa \mid b \end{aligned}$$

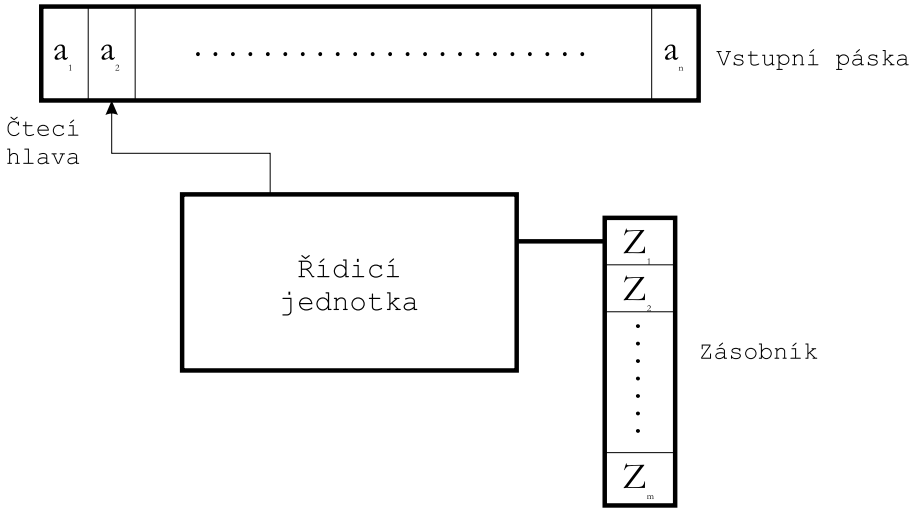
převeďte do Greibachové normální formy.

Cvičení 4.8.12 Sestrojte bezkontextovou gramatiku jazyka PL0, jehož grafy syntaxe jsou v příloze.

Kapitola 5

Zásobníkové automaty a jejich vztah k bezkontextovým jazykům

V této kapitole se budeme zabývat abstraktním zařízením – zásobníkovým automatem, který představuje přirozený model syntaktického analyzátoru bezkontextových jazyků. Zásobníkový automat je jednocestný nedeterministický automat, jenž je opatřen zásobníkem reprezentujícím nekonečnou paměť. Schéma zásobníkového automatu je uvedeno na obr. 5.1. Vstupní páska je



Obrázek 5.1: Schéma zásobníkového automatu

rozdělena na jednotkové záznamy, každý záznam obsahuje právě jeden vstupní symbol. Obsah jednotkového záznamu může být čten čtecí hlavou, nemůže však být přepsán (read only input tape). V určitém časovém okamžiku může čtecí hlava zůstat na daném záznamu, nebo se posune o jeden záznam doprava (jednocestný automat). Konečná řídící jednotka realizuje operace posuvu čtecí hlavy a ukládání informace do zásobníku prostřednictvím funkce přechodů definované nad vstupní abecedou, množinou stavů řídící jednotky a vrcholem zásobníku.

Obsahem této části je základní definice zásobníkového automatu, varianty zásobníkových automatů a základní výsledek teorie zásobníkových automatů – formální jazyk je bezkontextový, právě když je přijatelný zásobníkovým automatem. Dále bude definována důležitá podtřída

bezkontextových jazyků – deterministické bezkontextové jazyky, jež jsou přijímány tzv. deterministickými zásobníkovými automaty.

5.1 Základní definice zásobníkového automatu

Definice 5.1 *Zásobníkový automat* P je sedmice

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F), \text{ kde}$$

(1) Q je konečná množina stavových symbolů reprezentujících vnitřní stavy řídicí jednotky

(2) Σ je konečná vstupní abeceda; jejími prvky jsou vstupní symboly

(3) Γ je konečná abeceda zásobníkových symbolů

(4) δ je zobrazení z množiny $Q \times (\Sigma \in \{\epsilon\}) \times \Gamma$ do konečné množiny podmnožin množiny $Q \times \Gamma^*$ popisující funkci přechodů

(5) $q_0 \in Q$ je počáteční stav řídicí jednotky

(6) $Z_0 \in \Gamma$ je symbol, který je na počátku uložen do zásobníku – tzn. *startovací symbol*

(7) $Z \subseteq Q$ je množina koncových stavů

Definice 5.2 *Konfigurací* zásobníkového automatu P nazveme trojici

$$(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$$

kde

(1) q je přítomný stav řídící jednotky

(2) w je doposud nepřečtená část vstupního řetězce; první symbol řetězce w je pod čtecí hlavou. Je-li $w = \epsilon$, pak byly všechny symboly ze vstupní pásky přečteny.

(3) α je obsah zásobníku. Pokus nebude uvedeno jinak, budeme zásobník reprezentovat řetězcem, jehož nejlevější symbol koresponduje s vrcholem zásobníku. Je-li $\alpha = \epsilon$, pak je zásobník prázdný.

Definice 5.3 *Přechod* zásobníkového automatu P budeme reprezentovat binární relací $\vdash_P$ (nebo $\vdash$ bude-li zřejmé, že jde o automat P), která je definována na množině konfigurací zásobníkového automatu P. Relace

$$(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$$

platí, jestliže $\delta(q, a, Z)$ obsahuje prvek (q', γ) pro nějaké $q \in Q, a \in (\Sigma \cup \{\epsilon\}), w \in \Sigma^*, Z \in \Gamma$ a $\alpha, \gamma \in \Gamma^*$.

Relaci $\vdash_P$ interpretujeme tímto způsobem. Nachází-li se zásobníkový automat P ve stavu q a na vrcholu zásobníku je uložen symbol Z, pak po přečtení vstupního symbolu $a \neq \epsilon$ může automat přejít do stavu q', přičemž se čtecí hlava posune doprava a na vrchol zásobníku se uloží, po odstranění symbolu Z, řetězec γ. Je-li $\gamma = \epsilon$, pak je pouze odstraněn vrchol zásobníku.

Je-li $a = \epsilon$, neposouvá se čtecí hlava, což znamená, že přechod do nového stavu a nový obsah zásobníku není určován příštím vstupním symbolem. Tento typ přechodu budeme nazývat *ϵ-přechodem*. Poznamenejme, že ϵ-přechod může nastat i tehdy, když byly přečteny všechny vstupní symboly.

Relace $\vdash^i, \vdash^+, \vdash^*$ jsou definovány obvyklým způsobem. Relace $\vdash^+$ resp. $\vdash^*$ je tranzitivní, resp. tranzitivní a reflexivní uzávěr relace $\vdash$, $\vdash^i$ je i-tá mocnina relace $\vdash$, $i \geq 0$.

Počáteční konfigurace zásobníkového automatu má tvar (q_0, w, Z_0) pro $w \in \Sigma^*$, tj. automat je v počátečním stavu q_0, na vstupní pásce je řetězec w a v zásobníku je startovací symbol Z_0. *Koncová konfigurace* má tvar (q, ϵ, α), kde $q \in F$ je koncový stav a $\alpha \in \Gamma^*$.

Definice 5.4 Platí-li pro řetězec $w \in \Sigma^*$ relace $(q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha)$ pro nějaké $q \in F$ a $\alpha \in \Gamma^*$, pak říkáme, že řetězec w je přijímán zásobníkovým automatem P. Množinu $L(P)$ všech řetězců přijímaných zásobníkovým automatem P, který nazýváme *jazykem přijímaným zásobníkovým automatem*.

Příklad 5.1 Uvažujme jazyk $L = \{0^n 1^n \mid n \geq 0\}$. Zásobníkový automat P, který přijímá jazyk L, má tvar

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{Z, 0\}, \delta, q_0, Z, \{q_0\})$$

kde

$$\begin{aligned}
\delta(q_0, 0, Z) &= \{(q_1, 0Z)\} \\
\delta(q_1, 0, 0) &= \{(q_1, 00)\} \\
\delta(q_1, 1, 0) &= \{(q_2, \epsilon)\} \\
\delta(q_2, 1, 0) &= \{(q_2, \epsilon)\} \\
\delta(q_2, \epsilon, Z) &= \{(q_0, \epsilon)\}
\end{aligned}$$

a pracuje tak, že kopíruje všechny nuly ze vstupního řetězce do zásobníku. Objeví-li se na vstupu symbol 1, pak odstraňuje jednu nulu ze zásobníku. Kromě toho požaduje, aby všechny nuly předcházely všechny jedničky. Například pro vstupní řetězec 0011 realizuje automat P tyto přechody:

$$\begin{aligned}
(q_0, 0011, Z) &\vdash (q_1, 011, 0Z) \\
&\vdash (q_1, 11, 00Z) \\
&\vdash (q_2, 1, 0Z) \\
&\vdash (q_2, \epsilon, Z) \\
&\vdash (q_0, \epsilon, \epsilon)
\end{aligned}$$

Ukažme, že skutečně platí $L(P) = L$. Z definice funkce přechodů δ a relace $\vdash$ plyne $(q_0, \epsilon, Z) \vdash^0 (q_0, \epsilon, Z)$ a tedy $\epsilon \in L(P)$. Dále platí

$$\begin{aligned}
(q_0, 0, Z) &\vdash (q_1, \epsilon, OZ) \\
(q_1, 0^i, 0Z) &\vdash^i (q_1, \epsilon, 0^{i+1}Z) \\
(q_1, 1, 0^{i+1}Z) &\vdash (q_2, \epsilon, 0^i Z) \\
(q_2, 1^i, p^i Z) &\vdash^i (q_2, \epsilon, Z) \\
(q_2, \epsilon, Z) &\vdash (q_2, \epsilon, Z)
\end{aligned}$$

což dohromady implikuje

$$(q_0, 0^n 1^n, Z) \overset{2n+1}{\vdash} (q_0, \epsilon, \epsilon) \qquad \text{pro } n \geq 1$$

a tedy $L \subseteq L(P)$.

Dokázat opačnou inkluzi $L(P) \subseteq L$, tj. dokázat, že zásobníkový automat nepřijímá jiné řetězce, než $0^n 1^n, n \geq 1$, je obtížnější. Příjme-li zásobníkový automat P vstupní řetězec, pak musí projít posloupností stavů q_0, q_1, q_2, q_0.

Jestliže platí $(q_0, w, Z) \vdash^i (q_1, \epsilon, \alpha)$, pak $w = 0^i$ a $\alpha = 0^i Z$. Podobně, jestliže $(q_2, w, \alpha) \vdash^i (q_2, \epsilon, \beta)$ pak $w = 1^i$ a $\alpha = 0^i \beta$. To však znamená, že relace $(q_1, w, \alpha) \vdash (q_2, \epsilon, \beta)$ platí, právě když $w = 1$ a $\alpha = 0\beta$ a relace $(q_2, w, Z) \vdash^* (q_0, \epsilon, \epsilon)$ platí, právě když $w = \epsilon$. Tedy, jestliže platí $(q_0, w, Z) \vdash^i (q_0, \epsilon, \alpha)$ pro nějaké $i > 0$, pak $w = 0^n 1^n, i = 2n + 1$ a $\alpha = \epsilon$, tj. $L(P) \subseteq L$.

5.2 Varianty zásobníkových automatů

V tomto odstavci uvedeme dvě varianty základní definice zásobníkového automatu, které nám usnadní objasnit vztah mezi zásobníkovými automaty a bezkontextovými jazyky.

Definice 5.5 *Rozšířeným zásobníkovým automatem* rozumíme sedmici

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

kde δ je zobrazení z konečné podmnožiny $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^*$ do množiny podmnožin množiny $Q \times \Gamma^*$. Ostatní symboly mají stejný význam jako v základní definici 5.1.

Relace $(q, aw, \alpha, \gamma) \vdash (q', w, \beta, \gamma)$ platí, jestliže $\delta(q, a, \alpha)$ obsahuje (q', β) pro $q \in Q, a \in \Sigma \cup \{\epsilon\}$ a $\alpha \in \Gamma^*$. Tato relace odpovídá přechodu, v němž je vrcholový řetězec zásobníku odstraněn a nahrazen řetězcem β. (Připomeňme, že v základní definici je $\alpha \in \Gamma$ nikoliv $\alpha \in \Gamma^*$). Jazyk definovaný automatem P, je

$$L(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \alpha), w \in F, \alpha \in \Gamma^*\}$$

Rozšířený zásobníkový automat může, na rozdíl od základní definice provádět přechody i v případě, že je zásobník prázdný.

Příklad 5.2 Uvažujme rozšířený zásobníkový automat P, který přijímá jazyk

$$\begin{aligned} L &= \{ww^R \mid w \in \{a, b\}^*\}, \\ P &= (\{q, p\}, \{a, b\}, \{a, b, S, Z\}, \delta, q, Z, \{p\}) \end{aligned}$$

kde zobrazení δ je definováno takto:

$$\begin{aligned} \delta(q, a, \epsilon) &= \{(q, a)\} \\ \delta(q, b, \epsilon) &= \{(q, b)\} \\ \delta(q, \epsilon, \epsilon) &= \{(q, S)\} \\ \delta(q, \epsilon, aSa) &= \{(q, S)\} \\ \delta(q, \epsilon, bSb) &= \{(q, S)\} \\ \delta(q, \epsilon, SZ) &= \{(p, \epsilon)\} \end{aligned}$$

Pro vstupní řetězec $aabbaa$ realizuje automat P tyto přechody:

$$\begin{aligned}(q, aabbaa, Z) &\vdash (q, abbaa, aZ)\\ &\vdash (q, bbaa, aaZ)\\ &\vdash (q, baa, baaZ)\\ &\vdash (q, baa, SbaaZ)\\ &\vdash (q, aa, bSbaaZ)\\ &\vdash (q, aa, SaaZ)\\ &\vdash (q, a, aSaaZ)\\ &\vdash (q, a, SaZ)\\ &\vdash (q, \epsilon, aSaZ)\\ &\vdash (q, \epsilon, SZ)\\ &\vdash (q, \epsilon, \epsilon)\end{aligned}$$

Vidíme, že automat P pracuje tak, že nejdříve ukládá do zásobníku prefix vstupního řetězce, pak na vrchol zásobníku uloží znak S značící střed; pak vrcholový řetězec aSa nebo bSb nahrazuje symbolem S. Tento příklad ilustruje také nedeterministickou povahu zásobníkového automatu, protože zobrazením není určeno, zda má být přečten vstupní symbol a uložen na vrchol zásobníku ($\delta(q, x, \epsilon) = (q, x), x \in \{a, b\}$), nebo zda má být proveden ϵ-přechod, který uloží na vrchol zásobníku středový symbol S, ($\delta(q, \epsilon, \epsilon) = (q, S)$).

Z definice rozšířeného zásobníkového automatu vyplývá, že je-li jazyk L přijímán zásobníkovým automatem, pak je také přijímán rozšířeným zásobníkovým automatem. Ukažme nyní, že platí i opačná implikace.

Věta 5.1 Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je rozšířený zásobníkový automat. Pak existuje zásobníkový automat P_1 takový, že $L(P_1) = L(P)$.

Důkaz: Položme $m = \max\{|\alpha| \mid \delta(q, a, \alpha) \neq \emptyset$ pro nějaké $q \in Q$ a $\alpha \in \Sigma \cup \{\epsilon\}\}$

Zásobníkový automat P_1 budeme konstruovat tak, aby simuloval automat P. Protože automat P neurčuje přechody podle vrcholu zásobníku, ale podle vrcholového řetězce zásobníku, bude automat P_1 ukládat m vrcholových symbolů v jakési „vyrovnávací paměti“ řídící jednotky tak, aby na počátku každého přechodu věděl, jakých m vrcholových symbolů je v zásobníku automatu P. Nahrazuje-li automat P k vrcholových symbolů řetězcem délky l, pak se totéž provede ve vyrovnávací paměti automatu P_1. Jestliže $l < k$, pak P_1 realizuje $k - l$ ϵ-přechodů, které přesouvají $k - l$ symbolů z vrcholu zásobníku do vyrovnávací paměti. Automat P_1 pak může simulovat další přechod automatu P. Je-li $l \geq k$, pak se symboly přesouvají z vyrovnávací paměti do zásobníku.

Formálně můžeme konstrukci zásobníkového automatu P_1 popsat takto:

$$P_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, Z_1, F_1) \qquad \text{kde}$$

(1) $Q_1 = \{[q, \alpha] \mid q \in Q, \alpha \in \Gamma_1^*$ a $0 \leq |\alpha| \leq m\}$

(2) $\Gamma_1 = \Gamma \cup \{Z_1\}$

(3) Zobrazení δ_1 je definováno takto:

a) Předpokládejme, že $\delta(q, a, X_1 \ldots X_k)$ obsahuje $(r, Y_1 \ldots Y_l)$.

(i) Jestliže $l \geq k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$

$$\delta_1([q, X_1 \dots X_k\alpha], a, Z) \text{ obsahuje } ([r, \beta], \gamma Z)$$

kde $\beta\gamma = Y_1 \dots Y_l\alpha$ a $|\beta| = m$.

(ii) Je-li $l < k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$

$$\delta_1([q, X_1 \dots X_k\alpha], a, Z) \text{ obsahuje } ([r, Y_1 \dots Y_l\alpha Z], \epsilon)$$

b) Pro všechna $q \in Q, Z \in \Gamma_1$ a $\alpha \in \Gamma_1$ taková, že $|\alpha| < m$

$$\delta_1([q, \alpha], \epsilon, Z) = \{([q, \alpha, Z], \epsilon)\}$$

Tato pravidla vedou k naplnění vyrovnávací paměti (obsahuje m symbolů).

(4) $q_1 = [q_0, Z_0, Z_1^{m-1}]$. Vyrovnávací paměť obsahuje na počátku symbol Z_0 na vrcholu a $m-1$ symbolů Z_1 na dalších místech. Symboly Z_1 jsou speciální znaku pro označení dna zásobníku.

(5) $F_1 = \{[q, \alpha] \mid q \in F, \alpha \in \Gamma_1^*\}$

Lze ukázat, že

$$(q, aw, X_1 \dots X_k X_{k+1} \dots X_n) \underset{P}{\vdash} (r, w, Y_1 \dots Y_l X_{k+1} \dots X_n)$$

platí, právě když $([q, \alpha], aw, \beta) \vdash_{P_1}^+ ([r, \alpha'], w, \beta')$ kde

$$\begin{aligned} \alpha\beta &= X_1 \dots X_n Z_1^m \\ \alpha'\beta' &= Y_1 \dots Y_l X_{k+1} \dots X_n Z_1^m \\ |\alpha| &= |\alpha'| = m \end{aligned}$$

a mezi těmito dvěma konfiguracemi automatu P_1 není žádná konfigurace, ve které by druhý člen stavu (vyrovnávací paměť) měl délku m.

Tedy relace

$$(g_0, w, Z_0) \underset{P}{\vdash} (q, \epsilon, \alpha) \qquad \text{pro } q \in F, \alpha \in \Gamma^*$$

platí, právě když

$$([q_0, Z_0, Z_1^{m-1}], w, Z_1) \overset{*}{\underset{P_1}{\vdash}} ([q, \beta], \epsilon, \gamma)$$

kde $|\beta| = m$ a $\beta\gamma = \alpha Z_1^m$. Tedy $L(P) = L(P_1)$. □

Definice 5.6 Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový nebo rozšířený zásobníkový automat. Řetězec w je přijímán s *vyprázdněním zásobníku*, jestliže platí $(q_0, w, Z_0) \vdash^+ (q, \epsilon, \epsilon), a \in F$. Označme $L_\epsilon(P)$ množinu všech řetězců, které jsou přijímány zásobníkovým automatem P s vyprázdněním zásobníku.

Věta 5.2 Nechť L je jazyk přijímaný zásobníkovým automatem $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F), L = L(P)$. Lze zkonstruovat zásobníkový automat P' takový, že $L_\epsilon(P') = L$.

Důkaz: Opět budeme konstruovat automat P' tak, aby simuloval automat P. Kdykoli automat P dospěje do koncového stavu, bude automat P, nebo přejde do speciálního tvaru q_ϵ, který způsobí vyprázdnění zásobníku. Musíme však uvážit situaci, kdy automat P je v konfiguraci s prázdným zásobníkem, nikoliv však v koncovém stavu. Abychom zabránili případům, že automat P' přijímá

řetězec, který nemá být přijat, přidáme k zásobníkové abecedě automatu P' znak, jenž bude označovat dno zásobníku a může být vybrán pouze tehdy, je-li automat P' ve stavu q_ϵ. Formálně vyjádřeno, nechť

$$P = (Q \cup \{q_\epsilon, q'\}, \Sigma, \Gamma \cup \{Z'\}, \delta', q', Z', \emptyset), \quad \text{kde}$$

symbolem $\emptyset$ vyznačujeme, že P přijímá řetězec s vyprázdněním zásobníku. Zobrazení δ nyní definujeme takto:

(1) Jestliže $\delta(q, a, Z)$ obsahuje (r, γ), pak $\delta'(q, a, Z)$ obsahuje (r, γ) pro všechna $q \in Q, a \in \Sigma \cup \{\epsilon\}$ a $Z \in \Gamma$.

(2) $\delta(q', \epsilon, Z') = \{(q_0, Z_0 Z')\}$. První přechod zásobníkového automatu P' uloží do zásobníku řetězec $Z_0 Z'$, kde Z' je speciální znak označující dno zásobníku, a přejde do počátečního stavu automatu P.

(3) Pro všechna $q \in F$ a $Z \in \Gamma \cup \{Z'\}$ obsahuje $\delta'(q, \epsilon, Z)$ prvek (q_ϵ, ϵ).

(4) Pro všechna $Z \in \Gamma \cup \{Z'\}$ je $\delta(q_\epsilon, \epsilon, Z) = \{(q_\epsilon, \epsilon)\}$.

Nyní zřejmě platí

$$\begin{aligned} (q', w, Z') \quad &\vdash_{P'} \quad (q_0, w, Z_0 Z') \\ &\vdash^{n}_{P'} \quad (q, \epsilon, Y_1 \ldots Y_r) \\ &\vdash_{P'} \quad (q_\epsilon, \epsilon, Y_2 \ldots Y_r) \\ &\vdash^{r-1}_{P'} \quad (q_\epsilon, \epsilon, \epsilon) \qquad \text{kde } Y_r = Z', \end{aligned}$$

právě když

$$(q_0, w, Z_0) \overset{n}{\underset{P}{\vdash}} (q, \epsilon, Y_1 \ldots Y_{r-1})$$

pro $q \in F$ a $Y_1 \ldots Y_{r-1} \in \Gamma^*$. Tudíž $L_\epsilon(P') = L(P)$.

Předchozí věta platí také obráceně □

Věta 5.3 Nechť $P = (q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ je zásobníkový automat. Lze zkonstruovat zásobníkový automat P' takový, že $L(P) = L_\epsilon(P)$.

Důkaz: Zásobníkový automat P' konstruujeme tak, že má speciální symbol Z' na dně zásobníku. Jakmile je tento symbol ze zásobníku odstraněn, přechází automat P' do nového koncového stavu $q_f : F' = \{q_f\}$. Formální konstrukci automatu P' nebudeme provádět. □

5.3 Ekvivalence bezkontextových jazyků a jazyků přijímaných zásobníkovými automaty

Nejdříve ukážeme, jak lze zkonstruovat nedeterministický syntaktický analyzátor jazyka generovaného bezkontextovou gramatikou, který pracuje metodou shora–dolů.

Věta 5.4 Nechť $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Z gramatiky G můžeme zkonstruovat zásobníkový automat R takový, že $L_\epsilon(R) = L(G)$.

Důkaz: Zásobníkový automat R konstruujeme tak, aby vytvářel levou derivaci vstupního řetězce v gramatice G. Nechť $R = (\{q\}, \Sigma, N \cup \Sigma, \delta, S, \emptyset)$, kde δ je definováno takto:

(1) Je-li $A \to \alpha$ pravidlo z P, pak $\delta(q, \epsilon, A)$ obsahuje (q, α).

(2) $\delta(q, a, a) = \{(q, \epsilon)\}$ pro všechna $a \in \Sigma$.

Nyní ukážeme, že $A \Rightarrow^m w$, právě když a jen když $(q, w, \epsilon) \vdash^n (q, \epsilon, \epsilon)$ pro nějaké $m, n \geq 1$.

Část „jen když“ dokážeme indukcí pro m. Předpokládejme, že $A \Rightarrow^m w$. Je-li $m = 1$ a $w = a_1 \dots a_k, k \geq 0$ pak

$$\begin{aligned} (q, a_1 \dots a_k, A) &\vdash (q, a_1 \dots a_k) \\ &\vdash^k (q, \epsilon, \epsilon) \end{aligned}$$

Předpokládejme nyní, že platí $A \Rightarrow^m w$ pro $m > 1$. První krok této derivace musí mít tvar $A \Rightarrow X_1 X_2 \dots X_k$, přičemž $X_i \Rightarrow^{m_i} x_i$ pro $m_i < m, 1 \leq i \leq k$ a $x_1 x_2 \dots x_k = w$. Tedy

$$(q, w, A) \vdash (q, w, X_1 \dots X_k)$$

Je-li $X_i \in N$, pak podle induktivního předpokladu

$$(q, x_i, X_i) \stackrel{*}{\vdash} (q, \epsilon, \epsilon)$$

Je-li $X_i = x_i, x_i \in \Sigma$, pak

$$(q, x_i, X_i) \vdash (q, \epsilon, \epsilon)$$

Nyní již vidíme, že levé derivaci

$$A \Rightarrow X_1 \dots X_k \stackrel{m_1}{\Rightarrow} x_1 X_2 \dots X_k \stackrel{m_2}{\Rightarrow} \dots \stackrel{m_k}{\Rightarrow} x_1 x_2 \dots x_k = w$$

odpovídá tato posloupnost přechodů:

$$(q, x_1 \dots x_k, A) \vdash (q, x_1 \dots x_k, X_1 \dots X_k) \stackrel{*}{\vdash} (q, x_2 \dots x_k, X_2 \dots X_k) \dots \stackrel{*}{\vdash} (q, \epsilon, \epsilon)$$

Část „když“, tj. je-li $(q, w, A) \vdash^n (q, \epsilon, \epsilon)$, pak $\Rightarrow^+ w$ dokážeme indukcí pro n. Pro $n = 1, w = \epsilon$ a $A \to \epsilon$ je pravidlo v P. Předpokládejme, že dokazovaná relace platí pro všechna $n' < n$. Pak první přechod automatu R musí mít tvar

$$(q, w, A) \vdash (q, w, X_1 \dots X_k)$$

a $(q, x_i, X_i) \vdash^{n_i} (q, \epsilon, \epsilon)$ pro $1 \leq i \leq k$, kde $w = x_1 \dots x_k$. Pak $A \to X_1 \dots X_k$ je pravidlo z P a derivace $X_i \Rightarrow^+ x_i$ plyne z induktivního předpokladu. Je-li $X_i \in \Sigma$ pak $X_i \Rightarrow^0 x_i$.

Tedy $A \Rightarrow X_1 \dots X_k \Rightarrow^* x_1 X_2 \dots X_k \Rightarrow^* \dots \Rightarrow^* x_1 \dots x_{k-1} X_k \Rightarrow^* x_1 \dots x_{k-1} x_k = w$ je levá derivace řetězce w z A.

Vezmeme-li $S = A$ jako speciální případ, dostaneme $S \Rightarrow^+ w$, právě když $(q, w, s) \vdash^+ (q, \epsilon, \epsilon)$ a tedy $L_\epsilon(R) = L(G)$. □

Příklad 5.3 Ke gramatice G s pravidly

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

sestrojme zásobníkový automat P takový, že $L_\epsilon(P) = L(G)$

Řešení:

$$P = (\{q\}, \{+, *, (,), i\}, \{+, *, (,), i, E, T, F\}, \delta, q, E, \emptyset)$$

kde δ je zobrazení

$$\begin{aligned}
\delta(q, \epsilon, E) &= \{(q, E + T), (q, T)\} \\
\delta(q, \epsilon, T) &= \{(q, T * F), (q, F)\} \\
\delta(q, \epsilon, F) &= \{(q, (E)), (q, i)\} \\
\delta(q, \epsilon, a) &= \{(q, \epsilon)\} \quad \text{pro všechna} \quad a \in \{+, *, (,), i\}
\end{aligned}$$

Pro vstupní řetězec $i + i * i$ může zásobníkový automat P realizovat tuto posloupnost přechodů:

$$\begin{aligned}
(q, i + i * i, E) &\vdash (q, i + i * i, E + T) \\
&\vdash (q, i + i * i, T + T) \\
&\vdash (q, i + i * i, F + T) \\
&\vdash (q, i + i * i, i + T) \\
&\vdash (q, +i * i, +T) \\
&\vdash (q, i * i, T) \\
&\vdash (q, i * i, T * F) \\
&\vdash (q, i * i, F * F) \\
&\vdash (q, i * i, i * F) \\
&\vdash (q, *i, *F) \\
&\vdash (q, i, F) \\
&\vdash (q, i, i) \\
&\vdash (q, \epsilon, \epsilon)
\end{aligned}$$

□

Nyní ukážeme, jakým způsobem lze k bezkontextové gramatice G definovat rozšířený zásobníkový automat reprezentující syntaktický analyzátor zdola–nahoru.

Věta 5.5 Nechť $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Rozšířený zásobníkový automat $R = (\{q, r\}, \Sigma, N \cup \Sigma \cup \{\not\$\}, \delta, q, \not\$, \{r\})$, kde zobrazení δ je definováno takto:

(1) $\delta(q, a, \epsilon) = \{(1, a)\}$ pro všechna $a \in \Sigma$.

(2) Je-li $A \to \alpha$ pravidlo z P, pak $\delta(q, \epsilon, \alpha)$ obsahuje (q, A).

(3) $\delta(q, \epsilon, S\ \not\$) = \{(r, \epsilon)\}$.

Přijímá jazyk $L(G)$, tj. $L(R) = L(G)$.

Důkaz: Ukážeme, že automat R pracuje tak, že vytváří pravou derivaci postupnými redukcemi l-fráze počínaje terminálním řetězcem (umístěným na vstupní pásce) a konče výchozím symbolem S. Jinými slovy automat R realizuje syntaktickou analýzu zdola–nahoru.

Změna konvence: Vrchol zásobníku budeme nyní uvádět *vpravo*. Induktivní hypotéza, kterou dokážeme indukcí pro n má tvar:

$$S \underset{rm}{\overset{*}{\Rightarrow}} \alpha A y \underset{rm}{\overset{n}{\Rightarrow}} xy \quad \text{implikuje} \quad (q, xy, \not\$) \overset{*}{\vdash} (q, y, \not\$, \alpha A), \quad \underset{rm}{\Rightarrow} \quad \text{značí pravou derivaci.}$$

Pro $n = 0$ tato hypotéza platí, protože $\alpha A = x$ a tudíž automat R provádí přechody podle $\delta(q, a, \epsilon) = \{(q, a)\}, a \in \Sigma$, přesouvající řetězec x do zásobníku.

Nyní předpokládejme, že induktivní hypotéza platí pro všechny derivace, kratší než n. Můžeme psát

$$\alpha A \gamma \underset{rm}{\Rightarrow} \alpha\beta y \underset{rm}{\overset{n-1}{\Rightarrow}} xy$$

Je-li řetězec $\alpha\beta$ tvořen pouze terminálními symboly, pak $\alpha\beta = x$ a $(q, xy, \not\$) \vdash^* (q, y, \not\$\alpha\beta) \vdash (q, y, \not\$\alpha A)$. Není-li $\alpha\beta \in \Sigma^*$, pak můžeme psát $\alpha\beta = \gamma B z$, kde B je nejpravější nonterminál a podle induktivní hypotézy

$$S \underset{rm}{\overset{*}{\Rightarrow}} \gamma B z y \underset{rm}{\overset{n-1}{\Rightarrow}} xy$$

implikuje $(q, xy, \not\$) \vdash^* (q, zy, \not\$\gamma B)$. Platí tedy $(q, zy, \not\$\gamma B) \vdash^* (q, y, \not\$\gamma B z) \vdash (q, y, \not\$\alpha A)$ a tudíž jsme dokázali induktivní hypotézu. Protože $(q, \epsilon, \not\$S) \vdash (r, \epsilon, \epsilon)$ dostáváme $L(G) \subseteq L(R)$.

Abychom dokázali, že $L(R) \subseteq L(G)$, ukažme, že platí implikace:

$$\text{Jestliže } (q, xy, \not\$) \overset{n}{\vdash} (q, y, \not\$\alpha A), \text{ pak } \alpha A y \overset{*}{\Rightarrow} xy$$

Pro $n = 0$ tato implikace platí. Předpokládejme, že platí také pro všechny posloupnosti přechodů kratší než n. Protože vrchol zásobníku je nonterminální symbol, prováděl se poslední přechod podle předpisu (2) v zobrazení δ a můžeme psát:

$$(q, xy, \not\$) \overset{n-1}{\vdash} (q, y, \not\$\alpha\beta) \vdash (q, y, \not\$\alpha A)$$

kde $A \to \beta$ je pravidlo z P. Existuje tedy derivace $\alpha A y \Rightarrow \alpha\beta y \Rightarrow^* xy$.

Jako speciální případ uvažujme implikaci:

$$\text{je-li} \quad (q, w, \not\$) \overset{*}{\vdash} (q, \epsilon, \not\$S) \quad \text{pak} \quad S \overset{*}{\Rightarrow} w$$

Protože však platí $(q, w, \not\$) \vdash^* (q, \epsilon, \not\$S) \vdash (r, \epsilon, \epsilon)$ je $L(R) \subseteq L(G)$ a společně s první částí důkazu dostáváme $L(G) = L(R)$. □

Poznamenejme, že automat R skutečně představuje model syntaktického analyzátoru, který vytváří pravou derivaci vstupního řetězce postupnými redukcemi l-fráze větných forem (počáteční větná forma je vstupní řetězec, koncová větná forma je výchozí symbol gramatiky). Bezprostředně po přechodu automatu je pravá větná forma $\alpha A x$ reprezentována obsahem zásobníku (řetězec αA) a nezpracovanou částí vstupního řetězce (řetězec x). Následující činností automatu je přesunutí prefixu řetězce x na vrchol zásobníku a redukce l-fráze dané vrcholovým řetězcem zásobníku. Tento typ syntaktické analýzy se, jak již víme, nazývá syntaktická analýza zdola–nahoru.

Příklad 5.4 Sestrojme syntaktický analyzátor R zdola–nahoru pro gramatiku s pravidly

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

Nechť R je rozšířený zásobníkový automat

$$R = (\{q, r\}, \{+, *, (,), i\}, \{\not\$, E, T, F, +, *, (,), i\}, \delta, q, \not\$, \{r\})$$

kde δ je zobrazení

(1) $\delta(q, a, \epsilon) = \{(q, a)\}$ pro všechna $a \in \{i, +, *, (,)\}$.

(2)

$$\begin{aligned}
\delta(q, \epsilon, E + T) &= \{(q, E)\} \\
\delta(q, \epsilon, T) &= \{(q, E)\} \\
\delta(q, \epsilon, T * F) &= \{(q, T)\} \\
\delta(q, \epsilon, F) &= \{(q, T)\} \\
\delta(q, \epsilon, (E)) &= \{(q, F)\} \\
\delta(q, \epsilon, i) &= \{(q, F)\}
\end{aligned}$$

(3) $\delta(q, \epsilon, \not\$E) = \{(r, \epsilon)\}$

Pro vstupní řetězec $i + i * i$ může rozšířený zásobníkový automat R provést tuto posloupnost přechodů:

$$\begin{aligned}
(q, i + i * i, \not\$) &\vdash (q, +i * i, \not\$i) \\
&\vdash (q, +i * i, \not\$F) \\
&\vdash (q, +i * i, \not\$T) \\
&\vdash (q, +i * i, \not\$E) \\
&\vdash (q, i * i, \not\$E+) \\
&\vdash (q, *i, \not\$E + i) \\
&\vdash (q, *i, \not\$E + F) \\
&\vdash (q, *i, \not\$E + T) \\
&\vdash (q, i, \not\$E + T*) \\
&\vdash (q, \epsilon, \not\$E + T * i) \\
&\vdash (q, \epsilon, \not\$E + T * F) \\
&\vdash (q, \epsilon, \not\$E + T) \\
&\vdash (q, \epsilon, \not\$E) \\
&\vdash (q, \epsilon, \epsilon)
\end{aligned}$$

Na závěr tohoto odstavce ukážeme , že bezkontextové jazyky jsou ekvivalentní jazykům přijímaným zásobníkovými automaty, tj., že lze také ke každému zásobníkovému automatu R vytvořit bezkontextovou gramatiku G takovou, že $L(R) = L(G)$.

Věta 5.6 Nechť $R = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Pak existuje bezkontextová gramatika G, pro kterou platí $L(G) = L(R)$.

Důkaz: Gramatiku G budeme konstruovat tak, aby levá derivace terminálního řetězce w přímo korespondovala s posloupností přechodů automatu R. Budeme používat nonterminálních symbolů tvaru $[qZr]$ kde $q, r \in Q, Z \in \Gamma$. Vrchol zásobníku uvádíme opět *vlevo*.

Nechť gramatika $G = (N, \Sigma, P, S)$ je formálně definována takto:

(1) $N = \{[qZr] \mid q, r \in Q, Z \in \Gamma\} \cup \{S\}$

(2) Jestliže $\delta(q, a, Z)$ obsahuje $(r, X_1 \dots X_k)$ $k \geq 1$, pak k množině přepisovacích pravidel P přidej všechna pravidla tvaru

$$[qZs_k] \to a[rX_1s_1][s_1X_2s_2] \dots [s_{k-1}X_ks_k]$$

pro každou posloupnost $s_1, s_2, \dots, s_k$ stavů z množiny Q.

(3) Jestliže $\delta(q, a, Z)$ obsahuje (r, ϵ), pak k P přidej pravidlo $[qZr] \to a$.

(4) Pro každý stav $q \in Q$ přidej k P pravidlo $S \to [q_0Z_0q]$.

Indukcí lze opět dokázat, že pro všechna $q, r \in Q$ a $Z \in \Gamma$ platí $[qZr] \Rightarrow^m w$, právě když $(q, w, Z) \vdash^n (r, \epsilon, \epsilon)$. Speciální případ této ekvivalence je $S \Rightarrow [q_0Z_0q] \Rightarrow^+ w$, právě když $(q_0, w, Z_0) \vdash^+ (q, \epsilon, \epsilon), q \in F$. To však znamená, že $L_\epsilon(R) = L(G)$. □

Příklad 5.5 Nechť $P = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$ kde zobrazení δ je dáno takto:

$$\begin{aligned}
\delta(q_0, 0, Z_0) &= \{(q_0, XZ_0)\} \\
\delta(q_0, 0, X) &= \{(q_0, XX)\} \\
\delta(q_0, 1, X) &= \{(q_1, \epsilon)\} \\
\delta(q_1, 1, X) &= \{(q_1, \epsilon)\} \\
\delta(q_1, \epsilon, X) &= \{(q_1, \epsilon)\} \\
\delta(q_0, \epsilon, Z_0) &= \{(q_1, \epsilon)\}
\end{aligned}$$

Zkonstruujme gramatiku $G = (N, \Sigma, P, S)$ takovou, že $L(P) = L(G)$. Množiny nonterminálů a terminálů mají tvar:

$$\begin{aligned}
N &= \{S, [q_0Xq_0], [q_0Xq_1], [q_1Xq_0], [q_1Xq_1], [q_0Z_0q_0], [q_0Z_0q_1], [q_1Z_0q_0], [q_1Z_0q_1]\} \\
\Sigma &= \{0, 1\}
\end{aligned}$$

Množina N obsahuje některé nonterminály, které jsou v gramatice G nedostupné. Abychom je nemuseli dodatečně odstraňovat, začneme konstruovat pravidla gramatiky G počínaje S-pravidly a přidávejme pouze ty nonterminály, které se objevují na pravých stranách konstruovaných pravidel.

Podle bodu (4) sestrojíme S-pravidla:

$$S \to [q_0Z_0q_0] \qquad S \to [q_0Z_0q_1]$$

Nyní přidáme pravidla pro nonterminál $[q_0Z_0q_0]$

$$\begin{aligned}
[q_0Z_0q_0] &\to 0[q_0Xq_0][q_0Z_0q_0] \\
[q_0Z_0q_0] &\to 0[q_0Xq_1][q_1Z_0q_0]
\end{aligned}$$

vyplývající z $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$.

Pravidla pro nonterminál $[q_0Z_0q_1]$

$$\begin{aligned}
[q_0Z_0q_1] &\to 0[q_0Xq_0][q_0Z_0q_1] \\
[q_0Z_0q_1] &\to 0[q_0Xq_1][q_1Z_0q_1]
\end{aligned}$$

jsou požadována zobrazením $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$.

Pravidla pro zbývající nonterminály jsou:

$$\begin{aligned}
[q_0Xq_0] &\rightarrow 0[q_0Xq_0][q_0Xq_0] \\
[q_0Xq_0] &\rightarrow 0[q_0Xq_1][q_1Xq_0] \\
[q_0Xq_1] &\rightarrow 0[q_0Xq_0][q_0Xq_1] \\
[q_0Xq_1] &\rightarrow 0[q_0Xq_1][q_1Xq_1]
\end{aligned}$$

protože, $\delta(q_0, 0, X) = \{(q_0, XX)\}$;

$$\begin{aligned}
[q_0Xq_1] &\rightarrow 1, \text{ protože } \delta(q_0, 1, X) = \{(q_1, \epsilon)\} \\
[q_1Z_0q_1] &\rightarrow \epsilon, \text{ protože } \delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\} \\
[q_1Xq_1] &\rightarrow \epsilon, \text{ protože } \delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\} \\
[q_1Xq_1] &\rightarrow 1, \text{ protože } \delta(q_1, 1, X) = \{(q_1, \epsilon)\}.
\end{aligned}$$

Povšimněme si nyní, že pro nonterminály $[q_1Xq_0]$ a $[q_1Z_0q_0]$ nejsou definována žádná pravidla, a proto ani z nonterminálu $[q_0Z_0q_0]$ ani z $[q_0Xq_0]$ nemohou být derivovány terminální řetězce. Odstraníme-li tedy pravidla obsahující některý z uvedených čtyř nonterminálů, dostaneme ekvivalentní gramatiku generující jazyk $L(P)$:

$$\begin{aligned}
S &\rightarrow [q_0Z_0q_1] \\
[q_0Z_0q_1] &\rightarrow 0[q_0Xq_1][q_1Z_0q_1] \\
[q_0Xq_1] &\rightarrow 0[q_0Xq_1][q_1Xq_1] \\
[q_1Z_0q_1] &\rightarrow \epsilon \\
[q_0Xq_1] &\rightarrow 1 \\
[q_1Xq_1] &\rightarrow \epsilon \\
[q_1Xq_1] &\rightarrow 1
\end{aligned}$$

Výsledky dokázané v tomto odstavci nyní můžeme shrnout takto:

Tvrzení

- L je jazyk generovaný bezkontextovou gramatikou,
- L je jazyk přijímaný zásobníkovým automatem,
- L je jazyk přijímaný zásobníkovým automatem s vyprázdněním zásobníku a
- L je jazyk přijímaný rozšířeným zásobníkovým automatem

jsou vzájemně ekvivalentní.

5.4 Deterministický zásobníkový automat

V předchozím odstavci jsem ukázali, že ke každé bezkontextové gramatice lze sestrojit zásobníkový automat, který reprezentuje syntaktický analyzátor pro věty generované danou gramatikou. Tento analyzátor je obecně nedeterministický. Z hlediska aplikací teorie formálních jazyků

v překladačích jsou důležité tzv. deterministické bezkontextové jazyky, které lze analyzovat deterministickými syntaktickými analyzátory. V tomto odstavci definujeme třídu zásobníkových automatů, které v každém okamžiku mohou přijít pouze do jediné konfigurace, tzv. deterministické zásobníkové automaty, a jim odpovídající deterministické jazyky.

Definice 5.7 Zásobníkový automat $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ nazýváme deterministickým zásobníkovým automatem, jestliže pro každé $q \in Q$ a $Z \in \Gamma$ platí buď pro každé $a \in \Sigma$ obsahuje $\delta(q, a, Z)$ nanejvýš jeden prvek a $\delta(q, \epsilon, Z) = \emptyset$, nebo $\delta(q, a, Z) = \emptyset$ pro všechna $a \in \Sigma$ a $\delta(q, \epsilon, Z)$ obsahuje nejvýše jeden prvek.

Protože zobrazení $\delta(q, a, Z)$ obsahuje nejvýše jeden prvek, budeme místo $\delta(q, a, Z) = \{(r, \gamma)\}$ psát $\delta(q, a, Z) = (r, \gamma)$.

Příklad 5.6 Deterministický zásobníkový automat, který přijímá jazyk $L = \{wcw^R \mid w \in \{a, b\}^+\}$, má tvar:

$$P = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{Z, a, b\}, \delta, q_0, Z, \{q_2\})$$

kde zobrazení δ je definováno takto:

$$\begin{aligned}
\delta(q_0, X, Y) &= (q_0, XY) \quad \text{pro všechna } X \in \{a, b\} \text{ a } Y \in \{Z, a, b\} \\
\delta(q_0, c, Y) &= (q_1, Y) \quad \text{pro všechna } Y \in \{a, b\} \\
\delta(q_1, X, X) &= (q_1, \epsilon) \quad \text{pro všechna } X \in \{a, b\} \\
\delta(q_1, \epsilon, Z) &= (q_2, \epsilon)
\end{aligned}$$

Automat P pracuje tak, že dokud nepřečte středový symbol c, ukládá symboly vstupního řetězce do zásobníku. Potom přejde do stavu q_1 a srovnává další vstupní symboly se symboly zásobníku.

Definici deterministického zásobníkového automatu můžeme rozšířit tak, aby zahrnovala rozšířené zásobníkové automaty.

Definice 5.8 Rozšířený zásobníkový automat $R = (q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je (rozšířeným) deterministickým automatem, jestliže platí:

(1) Pro každé $q \in Q, a \in \Sigma \cup \{\epsilon\}$ a $\gamma \in \Gamma^*$ obsahuje $\delta(q, a, \gamma)$ nejvýše jeden prvek.

(2) Je-li $\delta(q, a, \alpha) \neq \emptyset, \delta(q, a, \beta) \neq \emptyset$ a $\alpha \neq \beta$, pak řetězec α není sufixem řetězce β a ani β není sufixem α.

(3) Je-li $\delta(q, a, \alpha) \neq \emptyset$ a $\delta(q, \epsilon, \beta) \neq \emptyset$, pak řetězec α není sufixem řetězce β a ani β není sufixem α.

Poznámka 5.1 Definice 5.8 předpokládá, že vrchol zásobníku je vpravo; pokud je vlevo, pak je třeba nahradit slovo „sufix“ slovem „prefix“.

Mezi zásobníkovým automatem a deterministickým zásobníkovým automatem bohužel neplatí vztah jako mezi nedeterministickým a deterministickým konečným automatem, tzn., že ne každý jazyk přijímaný zásobníkovým automatem může být přijímán deterministickým zásobníkovým automatem.

Definice 5.9 Jazyk L se nazývá *deterministický bezkontextový jazyk*, jestliže existuje deterministický zásobníkový automat P takový, že $L(P) = L$.

Třída deterministických bezkontextových jazyků představuje vlastní podtřídu jazyků bezkontextových. V následující kapitole se budeme zabývat některými důležitými typy deterministických bezkontextových jazyků a gramatik.

5.5 Cvičení

Cvičení 5.5.1 Sestrojte zásobníkový automat, který přijímá jazyk

$$L = \{a^n b^m \mid n \leq m \leq 2n\}$$

Cvičení 5.5.2 Pro gramatiky

(a)

$$S \to aSb \mid \epsilon$$

(b)

$$\begin{aligned} S &\to AS \mid b \\ A &\to SA \mid a \end{aligned}$$

(c)

$$\begin{aligned} S &\to SS \mid A \\ A &\to 0A1 \mid S \mid 01 \end{aligned}$$

sestrojte zásobníkové automaty modelující syntaktickou analýzu shora–dolů a zdola–nahoru.

Cvičení 5.5.3 Nalezněte gramatiku, která generuje jazyk $L(P)$, kde

$$P = (\{q_0, q_1, q_2\}, \{a, b\}, \{Z_0, A\}, \delta, q_0, Z_0, \{q_2\});$$

zobrazení δ má tvar:

$$\begin{aligned} \delta(q_0, a, Z_0) &= (q_1, AZ_0) \\ \delta(q_0, a, A) &= (q_1, AA) \\ \delta(q_1, a, A) &= (q_0, AA) \\ \delta(q_1, \epsilon, A) &= (q_2, A) \\ \delta(q_2, b, A) &= (q_2, \epsilon) \end{aligned}$$

Kapitola 6

Deterministické bezkontextové jazyky a jejich syntaktická analýza, precedenční jazyky

Velmi důležitou podmnožinou bezkontextových jazyků jsou z hlediska aplikací v programovacích jazycích *deterministické bezkontextové jazyky*. Jejich název je odvozen od zařízení, kterým lze tyto jazyky reprezentovat – deterministického zásobníkového automatu. Pro tyto jazyky lze sestrojit syntaktický analyzátor, který pracuje bez návratu – *deterministický syntaktický analyzátor*.

V této kapitole se seznámíme s některými deterministickými gramatikami, které generují deterministické jazyky:

1. precedenčními gramatikami
2. LL gramatikami
3. LR gramatikami

Existují bezkontextové jazyky, které nelze popsat žádnou deterministickou gramatikou. Menší obecnost algoritmů deterministického rozkladu však na druhé straně vyvažují přednosti, které deterministické jazyky přinášejí pro implementaci zvláště proto, že deterministickými gramatikami lze popsat většinu syntaktických rysů programovacích jazyků, které se normálně popisují bezkontextovou gramatikou.

Mezi nejvýznačnější vlastnosti deterministických jazyků z hlediska jejich implementace můžeme zařadit:

1. *Efektivnost rozkladu*

   Pro deterministický bezkontextový jazyk lze sestrojit deterministický syntaktický analyzátor, jenž pracuje v čase lineárně s lineárním paměťovým prostorem. Znamená to, že je-li n délka analyzovaného vstupního řetězce, pak existují konstanty c_1 a c_2 tak, že syntaktický analyzátor nespotřebuje více než $c_1 n$ operací (konečné délky) a více než $c_2 n$ paměti.

2. *Automatizovatelnost výstavby syntaktického analyzátoru*

   Jak dále uvidíme, syntaktická analýza je realizována jednoduchým algoritmem, který operuje nad určitou tabulkou (tabulkami), jež nese informace získané z deterministické gramatiky popisující analyzovaný jazyk. Existují algoritmy (konstruktory syntaktických analyzátorů), jejichž vstupem je deterministická gramatika a výstupem jsou tabulky pro vlastní syntaktický analyzátor.

3. *Snadnost lokalizace syntaktických chyb a připojení sémantického zpracování*

   Tyto vlastnosti jsou důsledkem skutečnosti, že deterministické syntaktické analyzátory pracují bez návratů.

6.1 Jednoduché precedenční gramatiky

Nejjednodušší třída deterministických syntaktických analyzátorů je založena na tzv. precedenčních relacích. V precedenčních gramatikách jsou hranice l-fráze stanovovány podle určitých (precedenčních) relací, které platí mezi symboly pravé větné formy. Syntaktické analyzátory postavené na precedenčních gramatikách patří k historicky nejstarším analyzátorům programovacích jazyků. Existuje řada různých tříd precedenčních gramatik; k nejznámějším patří:

- jednoduché precedenční gramatiky
- rozšířené precedenční gramatiky
- slabé precedenční gramatiky
- operátorové precedenční gramatiky

Z precedenčních relací patří k nejdůležitějším relace $\gtrdot$, která je definována mezi terminálními a nonterminálními symboly tak, že prohlížíme-li vstupní symboly pravé větné formy $\alpha\beta w$ zleva doprava a je-li řetězec β l-frází této větné formy, pak precedenční relace $\gtrdot$ platí mezi posledním symbolem l-fráze β a prvním symbolem terminálního podřetězce w. Relace $\gtrdot$ tedy slouží k určení konce l-fráze. Určení začátku (levého konce) l-fráze a pravidla, podle kterého je provedena redukce, se může provést několika způsoby v závislosti na typu precedence.

K ohraničení l-fráze β jednoduché precedenční gramatiky se používají tří precedenčních relací $\lessdot, \doteq, \gtrdot$. Mezi všemi dvojicemi symbolů řetězce α platí relace $\lessdot$ nebo $\doteq$; relace $\lessdot$ platí mezi posledním symbolem řetězce α a prvním symbolem řetězce β. Mezi všemi dvojicemi symbolů l-fráze β platí relace $\doteq$ a mezi posledním symbolem řetězce β a prvním symbolem řetězce w platí relace $\gtrdot$.

V jednoduché precedenční gramatice lze tedy l-frázi lokalizovat takto: Symboly dané větné formy prohlížíme tak dlouho, dokud nenajdeme první dvojici symbolů, pro které platí relace $\gtrdot$. Nalezli jsme tak konec l-fáze. Abychom našli její začátek, vracíme se (prohlížíme symboly větné formy zprava doleva) a hledáme první výskyt relace $\lessdot$. Řetězec mezi relacemi $\lessdot$ a $\gtrdot$ je hledanou l-frází. Předpokládáme-li, že pracujeme s gramatikou,v níž pravé straně pravidla (l-frázi) odpovídá jednoznačně levá strana pravidla, pak můžeme jednoznačně provést redukci. Tento proces se opakuje tak dlouho, dokud není celý vstupní řetězec zredukován až k výchozímu symbolu, nebo když už nemůže být provedena další redukce (v tom případě je signalizována chyba).

Definice 6.1 Nechť je $G = (N, \Sigma, P, S)$ bezkontextová gramatika. *Wirth-Weberovy precedenční relace* $\lessdot, \doteq, \gtrdot$ jsou definovány na množině $N \cup \Sigma$ takto:

(1) Existuje-li v P pravidlo $A \to \alpha X B \beta$ takové, že $B \Rightarrow^+ Y\gamma, A, B \in N, X, Y \in (N \cup \Sigma)$, pak $X \lessdot Y$.

(2) Existuje-li v P pravidlo $A \to \alpha X Y \beta$, pak $X \doteq Y$.

(3) Poněvadž symbol, jenž následuje za posledním symbolem l-fráze, musí být terminál, je relace $\gtrdot$ definována na $(N \cup \Sigma) \times \Sigma$

Existuje-li v P pravidlo $A \to \alpha B Y \beta$ takové, že $B \Rightarrow^+ \gamma X$ a $Y \Rightarrow^* a\delta$, pak $X \gtrdot a$.

Při konstrukci syntaktického analyzátoru je účelné přidat na začátek a na konec vstupního řetězce speciální koncový znak; použijeme znaku $\sharp$. Předpokládáme, že platí $\sharp \lessdot X$ pro všechna X, pro která $S \Rightarrow^+ X\alpha$ a $Y \gtrdot \sharp$ pro všechna Y, pro která $S \Rightarrow^+ \alpha Y$.

Poznámka 6.1 Precedenční relace čteme stejně jako relace $<, =, >$. Nemají však jejich vlastnosti. Relace $\doteq$ není obvyklou ekvivalencí, relace $\gtrdot$ a $\lessdot$ nejsou normálně tranzitivní a mohou být symetrické a reflexivní.

Definice 6.2 Gramatika G se nazývá *UI-gramatikou* (Uniquely Invertible), jestliže její žádná dvě pravidla nemají stejnou pravou stranu.

Není obtížné ukázat, že každá gramatika má alespoň jednu UI-gramatiku.

Definice 6.3 Nechť $G = (N, \Sigma, P, S)$ je vlastní gramatika, $\epsilon \notin L(G)$. Existuje-li pro každou dvojici symbolů z $N \cup \Sigma$ nanejvýš jedna Wirth-Weberova precedenční relace, pak se gramatika G nazývá *precedenční gramatikou*. Je-li precedenční gramatika UI-gramatikou, nazveme ji *jednoduchou precedenční gramatikou*.

Příklad 6.1 Uvažujme gramatiku G s pravidly:

$$S \to aSSb \mid c$$

precedenční relace pro symboly gramatiky G jsou spolu s koncovým znakem $\sharp$ obsaženy v tab. 6.1. Tato tabulka se nazývá precedenční maticí.

	S	a	b	c	$\sharp$
S	$\doteq$	$\lessdot$	$\doteq$	$\lessdot$	
a	$\doteq$	$\lessdot$		$\lessdot$	
b		$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
c		$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\sharp$		$\lessdot$		$\lessdot$	

Tabulka 6.1: Precedenční matice

Prázdná políčka v tab. 6.1 značí, že pro odpovídající symboly není definována precedenční relace. Vyskytnou-li se v průběhu analýzy takové symboly vedle sebe, pak jde zřejmě o syntaktickou chybu. Poněvadž jsou precedenční relace definovány jednoznačně a žádná dvě pravidla nemají stejnou pravou stranu, je G jednoduchá precedenční gramatika.

6.2 Výpočet precedenčních relací

Uvažujme gramatiku z příkladu 6.1 s pravidly

$$S \to aSSb \mid c.$$

Vyjdeme-li přímo z definice 6.1 precedenčních relací, pak nejsnáze určíme relaci $\doteq$, protože je definována pro každou dvojici sousedních symbolů pravé strany každého pravidla. Z prvního pravidla plyne $a \doteq S, S \doteq S$ a $S \doteq b$. Poněvadž druhé pravidlo má pravou stranu délky 1, v dané gramatice již jiné relace $\doteq$ neplatí.

Abychom nalezli relace $\lessdot$, uvažujme dvojice sousedních symbolů tvaru XC. Symbol X je v relaci $\lessdot$ s nejlevějším symbolem každého řetězce, jenž je netriviálně derivovatelný z C. V příkladě uvažujme dvojice aS a SS. Poněvadž z S mohou být derivovány pouze řetězce začínající symbolem a nebo c, je $a \lessdot a, a \lessdot c, S \lessdot a, S \lessdot c$.

Relaci $\gtrdot$ vypočítáme tak, že uvažujeme dvojice sousedních symbolů tvaru CX. Nejdříve najdeme všechny poslední symboly Y řetězců derivovatelných z C. Pak nalezneme všechny terminální symboly d, kterými začínají řetězce derivovatelné z X. (Je-li X terminál, pak $d = X$ je jediná možnost.) Pro každé Y a d platí $Y \gtrdot d$. V příkladě jsou uvažované dvojice SS a Sb, Y je b nebo c, d je a nebo c a tedy $b \gtrdot a, b \gtrdot c, c \gtrdot a, c \gtrdot c, b \gtrdot b, c \gtrdot b$.

Pro praktické účely je zřejmě uvedený způsob hledání precedenčních relací nevhodný. Ukážeme proto metodu výpočtu těchto relací, která využívá reprezentace binárních relací booleovskými maticemi.

Nejdříve přeformulujeme definici precedenčních relací $\lessdot$ a $\gtrdot$.

Věta 6.1 Nechť $G = (N, \Sigma, P, S)$ je vlastní gramatika, $\epsilon \notin L(G)$ a nechť F a L jsou relace:

$$\begin{aligned} F &= \{(A, X) \mid A \to X\alpha \text{ je v } P, X \in (N \cup \Sigma), \alpha \in (N \cup \Sigma)^*\}, \\ L &= \{(A, X) \mid A \to \alpha X \text{ je v } P, X \in (N \cup \Sigma), \alpha \in (N \cup \Sigma)^*\} \end{aligned}$$

pak

(1) $X \lessdot Y$, právě když existuje pravidlo $A \to \alpha XB\beta$ takové, že platí BF^+Y.

(2) $X \gtrdot a$, právě když $a \in \Sigma$ a existuje pravidlo $A \to \alpha BY\beta$ takové, že platí BL^+X a YF^*a.

Důkaz: Tvrzení plyne bezprostředně z definice 6.1

Na základě věty 6.1 můžeme již stanovit algoritmus výpočtu precedenčních relací prostřednictvím booleovských matic: Označme symbolem $[R]$ booleovskou matici jež reprezentuje binární relaci R (viz 4.8.1).

Algoritmus 6.1 Výpočet precedenčních relací prostřednictvím booleovských matic.

Vstup: Vlastní gramatika $G = (N, \Sigma, P, S), \epsilon \notin L(G)$.

Výstup: Matice $[\doteq], [\lessdot]$ a $[\gtrdot]$ reprezentující precedenční relace.

Metoda:

(1) Na základě pravidel gramatiky G definuj matice $[\doteq], [F]$ a $[L]$.

(2) Vypočítej matici $[\lessdot]$ jako součin matic $[\doteq]$ a $[F^+]$, kde matice $[F^+]$ reprezentuje tranzitivní uzávěr relace F a lze jej efektivně vypočítat aplikací Warshallova algoritmu (algoritmus 4.9).

$$[\lessdot] = [\doteq] \cdot [F^+]$$

(3) Vypočítej matici $[\overline{\gtrdot}]$ podle vztahu

$$[\overline{\gtrdot}] = [L^+]' \cdot [\doteq] \cdot [F^*]$$

kde

$[A]'$ značí transpozici matice A (popisuje inverzní relaci)

$[F^*]'$ značí tranzitivní a reflexivní uzávěr relace $F, [F^*] = [F^+] + [I], [I]$ je jednotková matice.

Z matice $[\overline{\gtrdot}]$ získáme již snadno matici $[\gtrdot]$ tak, že relaci $\overline{\gtrdot}$ zúžíme pouze na ty prvky $(X, Y) \in \overline{\gtrdot}$, pro něž platí $Y \in \Sigma$.

Ověření algoritmu 6.1 ponecháme jako cvičení.

□

Příklad 6.2 Ilustrujme nyní výpočet precedenčních relací algoritmem 6.1 na příkladě jednoduché precedenční gramatiky z příkladu 6.1:

(1)

$$[\doteq] = \begin{array}{c} \\ S \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{cccc} S & a & b & c \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad [F] = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad [L] = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(2) $[F^+] = [F]$,

$$[\lessdot] = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(3) $[L^+] = [L]$,

$$[L^+]' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$

$$[F^*] = [F^+] + [I] = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$[\overline{\gtrdot}] = [L^+]' \cdot [\doteq] \cdot [F^*] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

$$[\gtrdot] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Pro konstrukci úplné precedenční matice využijeme pak matic $[\doteq], [\lessdot]$ a $[\gtrdot]$ a hodnot SF^*X resp. SL^+X pro doplnění řádku resp. sloupce označeného symbolem $\sharp$:

	S	a	b	c	$\sharp$
S	$\doteq$	$\lessdot$	$\doteq$	$\lessdot$	
a	$\doteq$	$\lessdot$		$\lessdot$	
b		$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
c		$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\sharp$		$\lessdot$		$\lessdot$	

6.3 Algoritmus syntaktické analýzy jednoduchých precedenčních jazyků

Lemma 6.1 Nechť $G = (N, \Sigma, P, S)$ je vlastní gramatika, $\epsilon \notin L(G)$.

(1) Jestliže $X \lessdot A$ nebo $X \doteq A$ a $A \to Y\alpha$ je pravidlo z P, pak $X \lessdot Y$.

(2) Jestliže $A \lessdot a$, $A \doteq a$ nebo $A \gtrdot a$ a $A \to \alpha Y$ je pravidlo z P, pak $Y \gtrdot a$.

Důkaz: Tvrzení (1) ponecháme jako cvičení a dokážeme tvrzení (2):

(a) Je-li $A \lessdot a$, pak existuje pravá strana pravidla tvaru $\beta_1 AB\beta_2$ taková, že $B \Rightarrow^+ a\gamma$ pro nějaké $\gamma \in (N \cup \Sigma)^*$. Z předpokladu pravidla $A \to \alpha Y$ plyne okamžitě $Y \gtrdot a$.

(b) Je-li $A \doteq a$, pak existuje pravá strana $\beta_1 Aa\beta_2$. Protože platí $a \Rightarrow^* a$ a $A \Rightarrow^+ \alpha Y$, opět dostáváme $Y \gtrdot a$.

(c) Jestliže $A \gtrdot a$, pak existuje pravá strana $\beta_1 BX\beta_2$, kde $B \Rightarrow^+ \gamma A$ a $X \Rightarrow^* a\delta$ pro nějaké řetězce γ a δ. Pak však platí $B \Rightarrow^+ \gamma\alpha Y$ a také v tomto případě dostáváme $Y \gtrdot a$.

□

Následující věta spolu s uvedenou lemmou nám umožní formulovat algoritmus rozkladu jazyka generovaného jednoduchou precedenční gramatikou.

Věta 6.2 Nechť $G = (N, \Sigma, P, S)$ je vlastní gramatika, $\epsilon \notin L(G)$. Jestliže

$$\sharp S\sharp \underset{rm}{\overset{n}{\Rightarrow}} X_p X_{p-1} \dots X_{k+1} A a_1 \dots a_q \underset{rm}{\Rightarrow} X_p X_{p-1} \dots X_{k+1} X_k \dots X_1 a_1 \dots a_q$$

pak je:

(1) pro $p < i < k \quad X_{i+1} \lessdot X_i$ nebo $X_{i+1} \doteq X_i$

(2) $X_{k+1} \lessdot X_k$

(3) pro $1 \leq i < k$, $\quad X_{i+1} \doteq X_i$

(4) $X_1 \gtrdot a_1$

Důkaz: Důkaz provedeme indukcí. Pro $n = 0$ máme $\sharp S \sharp \Rightarrow_{rm} \sharp X_k \ldots X_1 \sharp$. Z definice precedenčních relací plyne $\sharp \lessdot X_k, X_{i+1} \doteq X_i$ pro $k > i \geq 1$ a $X_1 \gtrdot \sharp$. Řetězec $X_k \ldots X_1$ nemůže být prázdný, poněvadž $\epsilon \notin L(G)$.

Nyní předpokládejme, že tvrzení platí pro n a uvažujme derivaci

$$\begin{aligned} \sharp S \sharp &\Rightarrow_{rm}^{n} X_p \ldots X_{k+1} A a_1 \ldots a_q \\ &\Rightarrow_{rm} X_p \ldots X_{k+1} X_k \ldots X_1 a_1 \ldots a_q \\ &\Rightarrow_{rm} X_p \ldots X_{j+1} Y_r \ldots Y_1 X_{j-1} \ldots X_1 a_1 \ldots a_{q'} \end{aligned}$$

tj. nonterminál X_j je nahrazen $Y_r \ldots Y_1$ v poslední přímé derivaci, takže $X_{j-1} \ldots X_1$ jsou terminální symboly.

Podle induktivní hypotézy je $X_{j+1} \lessdot X_j$ nebo $X_{j+1} \doteq X_j$. Tedy, na základě lemmy 6.1 (1) je $X_{j+1} \lessdot Y_r$. Podobně mezi X_j a symbolem vpravo (kterým může být také a_1) platí jedna z precedenčních relací a tedy podle 6.1 (2) platí $Y_1 \gtrdot X_{j-1}$ nebo, je-li $j = 1, Y_1 \gtrdot a_1$. Poněvadž $Y_r \ldots Y_1$ je pravá strana pravidla, dostáváme $Y_r \doteq Y_{r-1}, \ldots, Y_2 \doteq Y_1$. Konečně, platnost relace $X_{j+1} \lessdot X_i$ nebo $X_{i+1} \doteq X_i$ plyne pro $p < i < j$ z induktivního předpokladu. □

Jestliže G je precedenční gramatika, pak tvrzení věty 6.2 můžeme na základě definice zesílit takto:

V bodě (1): „právě jedna z relací $\lessdot$ nebo $\doteq$“.

V bodech (1)–(4): „a žádné jiné relace neplatí“.

Podle uvedených tvrzení lze zkonstruovat algoritmus deterministické syntaktické analýzy pro jednoduchou precedenční gramatiku. V každém kroku analýzy je v každé pravé větné formě $X_p X_{p-1} \ldots X_{k+1} X_k \ldots X_1 a_1 \ldots a_q$ precedenčními relacemi jednoznačně definována l-fráze $X_k \ldots X_1$ a jednoznačně definována redukce podle pravidla $A \to X_k \ldots X_1$. Snadno dokazatelným důsledkem této skutečnosti je, že jednoduchá precedenční gramatika je jednoznačná.

Ještě dříve, než uvedeme vlastní algoritmus syntaktické analýzy, definujme pojem pravý rozklad, který nám umožní popsat výsledek syntaktické analýzy zdola–nahoru.

Definice 6.4 Nechť $G = (N, \Sigma, P, S)$ je gramatika, jejíž pravidla jsou očíslována indexy $1, 2, \ldots, p$ (tj. p je mohutnost množiny P) a nechť α je větná forma.

Pravým rozkladem řetězce α budeme rozumět v opačném pořadí zapsanou posloupnost indexů pravidel, jež byly použity v pravé derivaci větné formy α.

Algoritmus 6.2 Rozklad vět jednoduchého precedenčního jazyka.

Vstup: Jednoduchá precedenční gramatika $G = (N, \Sigma, P, S)$ precedenční matice a vstupní řetězec $w = a_1 \ldots a_n \sharp$.

Pravidla z P jsou očíslována $1, 2, \ldots, p$.

Výstup: Pravý rozklad řetězce w, je-li w větou jazyka $L(G)$, jinak *error*.

Metoda: Činnost algoritmu popíšeme opět přechody mezi konfiguracemi. Konfigurace tohoto algoritmu je dána trojicí $(\alpha X, x, \pi)$, kde

(a) αX je obsah zásobníku Z, X je vrchol zásobníku

(b) x je doposud nepoužitá část vstupního řetězce

(c) π je řetězec, jenž po skončení algoritmu reprezentuje pravý rozklad řetězce w. Konfigurace $(\sharp, w, \epsilon)$ je počáteční konfigurací algoritmu (v zásobníku Z je uložen koncový symbol $\sharp$)

(1) Hledání l-fráze. $(\alpha, ax, \pi) \vdash (\alpha a, x, \pi)$, je-li $X \lessdot a$ nebo $X \doteq a$ (X je nejpravější symbol řetězce α); vrať se k (1). Je-li $X \gtrdot a$, přejdi k (2). Není-li pro (X, a) definována precedenční relace pak *error*.

(2) Redukce.

$$(\alpha X_{k+1} X_k \dots X_1, x, \pi) \vdash (\alpha X_{k+1} A, x, \pi i),$$

jestliže platí:

a) $X_{j+1} \doteq X_j$ pro $1 \leq j < k$

b) $X_{k+1} \lessdot X_k$

c) a je-li $A \to X_k \dots X_1$ i-tým pravidlem z P.

Je-li $x = \sharp$, pak algoritmus končí; πi je pravý rozklad řetězce w. Jinak přejdi k (1). Nemůže-li být redukce provedena, pak *error*.

Snadno lze dokázat, že je počet operací algoritmu 6.2 lineárně závislý na délce vstupního řetězce. Činnost tohoto algoritmu ukážeme na příkladě.

Příklad 6.3 Uvažujme precedenční gramatiky z příkladu 6.1

(1) $S \to aSSb$

(2) $S \to c$

Její precedenční matice je v tab. 6.1. Pro vstupní řetězec *accb* algoritmus 6.2 prochází touto posloupností konfigurací:

$$\begin{aligned}
(\sharp, accb\sharp, \epsilon) &\vdash (\sharp a, ccb\sharp, \epsilon) \\
&\vdash (\sharp ac, cb\sharp, \epsilon) \\
&\vdash (\sharp aS, cb\sharp, 2) \\
&\vdash (\sharp aSc, b\sharp, 2) \\
&\vdash (\sharp aSS, b\sharp, 22) \\
&\vdash (\sharp aSSb, \sharp, 22) \\
&\vdash (\sharp S, \sharp, 221)
\end{aligned}$$

Pravým rozkladem je řetězec $\pi = 221$. Dále popíšeme chování algoritmu 6.2, není-li w v $L(G)$.

Uvažujme vstupní řetězec acb a odpovídající posloupnost konfigurací:

$$\begin{aligned}(\sharp, acb\sharp, \epsilon) &\vdash (\sharp a, cb\sharp, \epsilon)\\ &\vdash (\sharp ac, b\sharp, \epsilon)\\ &\vdash (\sharp aS, b\sharp, 2)\\ &\vdash (\sharp aSb, \sharp, 2)\\ &\quad error\end{aligned}$$

V poslední konfiguraci je jako l-fráze identifikován řetězec aSb, protože platí $\sharp \lessdot a \doteq S \doteq b \gtrdot \sharp$. Tuto frázi však nelze redukovat, poněvadž v G neexistuje pravidlo s pravou stranou aSb.

6.4 Linearizace precedenční matice

Uvážeme-li, že běžný programovací jazyk má kolem 150-ti symbolů, pak je k reprezentaci precedenční matice třeba obsadit minimálně $150 \times 150 \times 2$ bitů. (Reprezentace $\lessdot, \doteq$ a $\gtrdot$ jsou zobrazitelné nejméně na 2 bitech). Proto bylo nutno hledat cesty, jak snížit nároky na paměť pro ukládání informace o precedenčních relacích. Jednou z cest jsou *precedenční funkce*, které vzniknou tzv. *linearizací precedenční matice*.

Definice 6.5 Nechť je $G = (N, \Sigma, P, S)$ jednoduchá precedenční gramatika. M je její precedenční matice. Jsou-li f a g celočíselné nezáporné funkce definované na množině $N \cup \Sigma \cup \{\sharp\}$ tak, že

(1) je-li $M(X,Y) = \lessdot$, pak $f(X) < g(Y)$

(2) je-li $M(X,Y) = \doteq$, pak $f(X) = g(Y)$

(3) je-li $M(X,Y) = \gtrdot$, pak $f(X) > g(Y),\ X,Y \in N \cup \Sigma \cup \{\sharp\}$

Příklad 6.4 Uvažujme jednoduchou precedenční gramatiku G s pravidly

$$S \to aSc \mid bSc \mid c$$

Její precedenční matice je uvedena v tab. 6.2.

	S	a	b	c	$\sharp$
S				$\doteq$	
a	$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	
b	$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	
c				$\gtrdot$	$\gtrdot$
$\sharp$		$\lessdot$	$\lessdot$	$\lessdot$	

Tabulka 6.2: Precedenční matice

Hodnoty precedenčních funkcí f a g jsou uvedeny v tab. 6.3.

Nároky na paměť se tedy použitím precedenčních funkcí nezanedbatelně sníží. Má-li gramatika n symbolů, pak precedenční matice vyžaduje $(n+1) \times (n+1)$ elementárních paměťových

	f	g
S	1	0
a	0	1
b	0	1
c	2	1
$\sharp$	0	0

Tabulka 6.3: Hodnoty precedenčních funkcí

míst (uvažujme rovněž koncový symbol $\sharp$), kdežto precedenční funkce potřebuje pouze $2(n+1)$ paměťových míst. Precedenční funkce jsou uloženy jako vektory. Náhrada precedenční matice precedenčními funkcemi se nazývá linearizací precedenční matice.

Použití precedenčních funkcí místo precedenční matice však přináší jednu nevýhodu. „Prázdná“ políčka v matici jsou během analýzy interpretována jako syntaktické chyby. Uvažujme např. precedenční matici z předešlého příkladu (tab. 6.2). Vyskytnou-li se např. vedle sebe symboly cb, je zřejmé, že vstupní řetězec není syntakticky správně utvořen. Poněvadž jsou však precedenční funkce definovány pro všechny symboly gramatiky G, bude mezi symboly cb vypočtena precedenční relace $\gtrdot$. I když je jasné, že v některém z příštích kroků nebude možné provést redukci, nevýhodou je, že se indikace chyby oddálí. Tento problém je částečně řešitelný pr slabé precedenční gramatiky, kde algoritmus nemusí rozlišovat relace $\lessdot$ a $\doteq$. Pak lze navrhnout precedenční matici tak, že je pro příslušné precedenční funkce případ $f(X) = g(Y)$ interpretován oprávněně jako syntaktická chyba. Sníží se tím počet případů, kdy analyzátor neohlásí co nejdříve chybu.

Ne však ke každé precedenční matici existují precedenční funkce. V algoritmu 6.3 je uveden postup pro nalezení precedenčních funkcí, pokud existují, pro danou precedenční matici. Používá precedenční matice M jejíž prvky $0, -1, 1$ reprezentují precedenční relace takto: Platí-li mezi symbolem označujícím i-tý řádek a symbolem, který označuje j-tý sloupec relace

(1) $\lessdot$ pak $M_{i,j} = -1$

(2) $\doteq$ pak $M_{i,j} = 0$

(3) $\gtrdot$ pak $M_{i,j} = 1$

(4) relace není definována, pak $M_{i,j} =$ blank

Algoritmus 6.3

Vstup: Precedenční matice M řádu n, jejíž hodnoty jsou $-1, 0, 1$ a blank.

Výstup: Precedenční funkce reprezentovaná celočíselnými vektory $f = (f_1, \ldots, f_n)$ a $g = (g_1, \ldots, g_n)$, pro které platí

$f_i < g_j$ je-li $M_{i,j} = -1$
$f_i = g_j$ je-li $M_{i,j} = 0$
$f_i > g_j$ je-li $M_{i,j} = 1$
nebo „NE“, pokud takové funkce neexistují.

Metoda:

(1) Vytvoř orientovaný graf s $2 \times n$ uzly, jenž se nazývá *linearizačním grafem* pro M. Na počátku označ uzly grafu symboly $F_1, F_2, \ldots, F_n$ a $G_1, G_2, \ldots, G_n$. V průběhu algoritmu budou tyto uzly „reprezentovány“ novými uzly podle kroků (2) a (3). V každém okamžiku bude existovat uzel $\widehat{F_i}$ reprezentující uzel F_i a uzel $\widehat{G_j}$ reprezentující uzel G_j. Na počátku polož $\widehat{F_i} = F_i$ a $\widehat{G_j} = G_j$ pro všechna i a j. Pro každé i i j prováděj kroky (2) a (3).

(2) Je-li $M_{i,j} = 0$, vytvoř nový uzel N spojením uzlů $\widehat{F_i}$ a $\widehat{G_j}$. Uzel N bude nyní reprezentovat všechny uzly, které byly dříve reprezentovány uzly $\widehat{F_i}$ a $\widehat{G_j}$.

(3) Je-li $M_{i,j} = 1$, sestroj hranu, která vede z uzlu $\widehat{F_i}$ do uzlu $\widehat{G_j}$. Je-li $M_{i,j} = -1$, sestroj hranu, která vede z uzlu $\widehat{G_j}$ do uzlu $\widehat{F_i}$.

(4) Je-li výsledný graf cyklický, je výstup algoritmu „NE“. Precedenční funkce k matici M neexistují.

(5) Je-li linearizační graf acyklický, pak je f_i rovno délce nejdelší cesty z uzlu $\widehat{F_i}$ a g_j rovno délce nejdelší cesty z uzlu $\widehat{G_j}$.

Poznámka 6.2 V kroku (4) algoritmu 6.3 je nutno zjistit, zda je orientovaný graf cyklický.

Příklad postupu:

(i) Najdi uzel grafu, jenž nemá žádné vycházející hrany. Neexistuje-li takový uzel, je graf cyklický. V opačném případě tento uzel odstraň a přejdi k bodu (ii).

(ii) Je-li výsledný graf prázdný, pak je původní graf acyklický. Jinak opakuj bod (i).

V kroku (5) algoritmu 6.3 máme nalézt délku nejdelší cesty z daného uzlu orientovaného acyklického grafu. Lze použít této techniky: ke každému uzlu N_i grafu budeme postupně přiřazovat celá čísla k_i, která budou po skončení tohoto algoritmu reprezentovat délku nejdelší cesty z N_i.

(i) Na začátku polož pro všechny uzly N_i hodnotu $k_i = 0$.

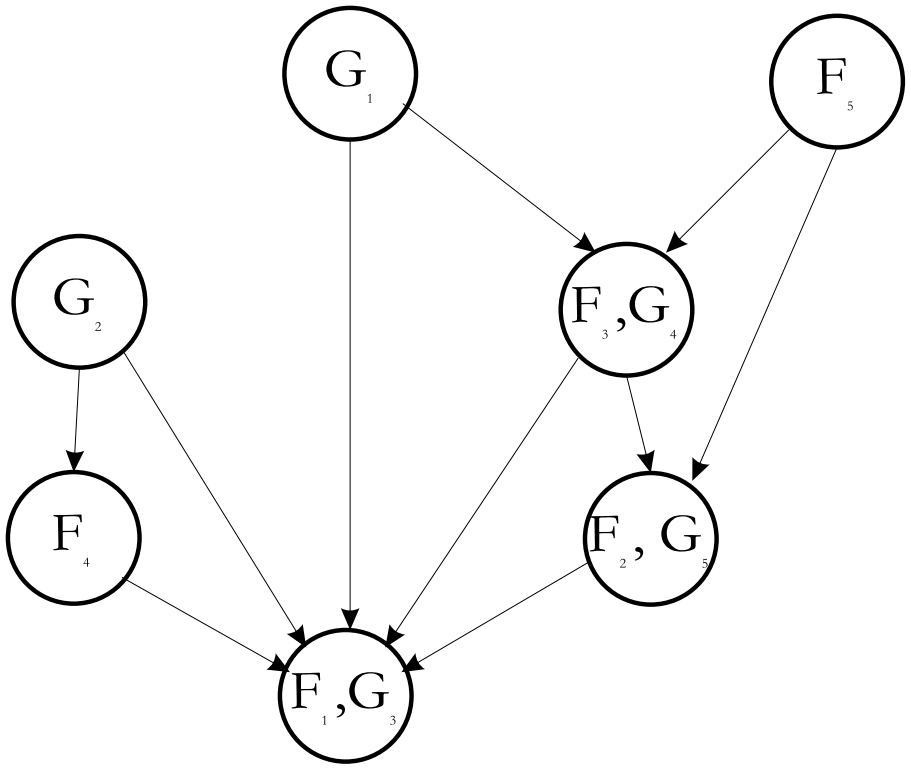
(ii) Krok (iii) opakuj tak dlouho, dokud se mění hodnota některého k_i. Nezmění-li aplikace kroku (iii) žádnou z hodnot k_i, pak příslušná k_i udávají nejdelší cestu z uzlu N_i.

(iii) Vezmi uzel N grafu. Vycházejí-li u uzlu N hrany vedoucí do uzlů $N_1, N_2, \ldots, N_m$ a příslušejí-li těmto uzlům hodnoty $k_1, k_2, \ldots, k_m$, pak polož $k = \max(k_1, k_2, \ldots, k_m) + 1$; k je hodnota, která přísluší k uzlu N. Tento krok proveď pro všechny uzly.

Je-li d délka nejdelší cesty v orientovaném acyklickém grafu, pak je pro jeden uzel třeba provést krok (iii) maximálně d-krát.

Příklad 6.5 Uvažujme precedenční matici M

	1	2	3	4	5
1	-1	-1	0		-1
2			1	-1	0
3	-1		1	0	
4		-1	1		
5				1	1



Obrázek 6.1: Linearizační graf

Po provedení algoritmu 6.3 získáme linearizační graf znázorněn na obr. 6.1. Podle kroku (2) algoritmu 6.3 jsou spojeny uzly $(F_3, G_4), (F_2, G_5)$ a (F_1, G_3). Linearizační graf je acyklický, výsledné precedenční funkce jsou:

$$\begin{aligned} f &= (0, 1, 2, 1, 3) \\ g &= (3, 2, 0, 2, 1) \end{aligned}$$

Např. $f_5 = 3$, poněvadž nejdelší cesta z uzlu F_5 má délku 3.

6.5 Stratifikace gramatiky

Pro danou gramatiku programovacího jazyka velmi často nejsou precedenční relace definovány jednoznačně. Znamená to, že pro dva symboly z $N \cup \Sigma$ platí více, než jedna precedenční relace. Zdrojem takových kolizí může být levá rekurze. Předpokládejme, že gramatika obsahuje pravidlo $A \to A\alpha$ a pravidlo $B \to \beta CA\gamma$. Pro symboly CA pak platí zároveň relace $C \doteq A$ a $C \lessdot A$. Tento typ kolize lze někdy odstranit přidáním nového nonterminálu D a zaměněním pravidla $B \to \beta CA\gamma$ dvojicí pravidel $B \to \beta CD\gamma$ a $D \to A$. Nyní platí $C \doteq D$ a $C \lessdot A$.

Podobně, obsahuje-li gramatika pravidla rekurzivní zprava, mohou nastat kolize mezi relacemi $\doteq$ a $\gtrdot$, které by bylo možno odstranit zavedením nového nonterminálu. Tato metoda se nazývá *stratifikací (rozvrstvením) gramatiky*. Uvedený postup však nemusí vždycky vést k jednoduché precedenční gramatice. Provedené změny mohou být důsledkem jiných kolizí v jednoznačnosti precedenčních relací. Platí-li mezi symboly A a B současně relace $\lessdot$ a relace $\gtrdot$, pak tento postup nevede k úspěšnému cíli a vhodnější je použít některého z jiných typů syntaktické analýzy.

Nehledě k tomu, že stratifikace zvětšuje precedenční matici, uvedené manipulace s gramatikou, která má kolem 100 pravidel u jazyků jako je Algol 60, vyžadují příliš mnoho času i pro zkušené osoby.

Stratifikaci budeme nyní ilustrovat na příkladě gramatiky pro popis aritmetického výrazu.

Příklad 6.6 Uvažujme gramatiku G s pravidly:

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

Z prvního pravidla plyne relace $+ \doteq T$. Poněvadž je však gramatika G rekurzivní zleva v T, platí též $+ \lessdot T$. Podobný problém nastává s dvojicí symbolů $(, E$. Stratifikací obdržíme gramatiku s pravidly

(1) $E \to E_1$
(2) $E_1 \to E_1 + T_1$
(3) $E_1 \to T_1$
(4) $T_1 \to T$
(5) $T \to T * F$
(6) $T \to F$
(7) $F \to (E)$
(8) $F \to i$

která, jak je vidět z její precedenční matice v tab. 6.4, je jednoduchou precedenční gramatikou. Tab. 6.4 obsahuje rovněž hodnoty precedenčních funkcí f a g.

	E	E_1	T_1	T	F	i	$($	$+$	$*$	$)$	$\sharp$	f	g
E										$\doteq$		2	2
E_1								$\doteq$		$\gtrdot$	$\gtrdot$	4	3
T_1								$\gtrdot$		$\gtrdot$	$\gtrdot$	5	4
T								$\gtrdot$	$\doteq$	$\gtrdot$	$\gtrdot$	6	5
F								$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	7	6
i								$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	7	7
$($	$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$					2	7
$+$			$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$					4	4
$*$					$\doteq$	$\lessdot$	$\lessdot$					6	6
$)$								$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	7	2
$\sharp$		$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$					1	8

Tabulka 6.4: Precedenční matice a funkce

6.6 Jiné třídy precedenčních gramatik

Rozšíření precedenční gramatiky

Definice Wirth-Weberových precedenčních relací lze zobecnit tak, že místo precedenčních relací mezi dvěma symboly definujeme precedenční relace mezi dvojicemi řetězců. Tak zvané (m,n)-precedenční relace $\lessdot, \doteq, \gtrdot$ jsou definovány jako podmnožina množiny

$$(N \cup \Sigma \cup \{\sharp\})^m \times (N \cup \Sigma \cup \{\sharp\})^n$$

Jsou-li tyto relace disjunktní, pak mluvíme o (m,n)-precedenční gramatice. Věty jazyka, jenž je generován tzv. *rozšířenou precedenční gramatikou*, mohou být analyzovány algoritmem, jenž se jen nepatrně liší od algoritmu 6.2. Pro vyhodnocení precedenční relace se musí v algoritmu rozkladu uvažovat m symbolů na vrcholu zásobníku a n příštích vstupních symbolů. I když z praktických důvodů (rozsah precedenční matice) nebývá $n+m > 5$, je třída (m,n) precedenčních jazyků významně větší něž třída jednoduchých precedenčních jazyků. Zmíněný algoritmus rozkladu však opět vyžaduje, aby byla (m,n)-precedenční gramatika zároveň UI-gramatikou.

Slabé precedenční gramatiky

Definice 6.6 Nechť $G = (N, \Sigma, P, S)$ bezkontextová gramatika, která nemá žádné ϵ-pravidla. Říkáme, že G je slabá precedenční gramatika, platí-li tyto dvě podmínky:

(1) Relace $\gtrdot$ je disjunktní se sjednocením relací $\lessdot$ a $\doteq$.

(2) Jsou-li $A \to \alpha X \beta$ a $B \to \beta$ pravidla z P a $X \in N \cup \Sigma$, pak neplatí žádná z relací $X \lessdot B$ nebo $X \doteq B$.

K určení konce l-fráze opět slouží relace $\gtrdot$. K určení začátku l-fráze však nelze použít relací $\lessdot$ a $\doteq$, poněvadž nejsou disjunktní. Algoritmus rozkladu hledá začátek l-fráze tak, že porovnává řetězec slov od konce fráze s pravými stranami pravidel. V případě, že by byla některá pravá strana pravidla sufixem jiné pravé strany, mohlo by dojít k nejednoznačnosti. Jsou-li např. $A \to \gamma$ a $B \to \beta\gamma$ pravidla z P a $\alpha\beta\gamma w$ pravá větná forma, ve které byl nalezen konec l-fráze mezi řetězci γ a w, pak by nemuselo být jasné, zda se má redukovat řetězec γ, nebo řetězec $\beta\gamma$. Podmínka (2) v definici slabé precedenční gramatiky však zaručuje, že je l-fráze jednoznačně dána nejdelší pravou stranou pravidla, kterého lze k redukci použít. V našem příkladě je tedy $\alpha B w$ příští pravou větnou formou v procesu rozkladu.

Třída slabých precedenčních jazyků není větší než třída jednoduchých precedenčních jazyků. Existuje algoritmus, jenž každou slabou precedenční gramatiku převede na jednoduchou precedenční gramatiku. Výhodou slabých precedenčních gramatik je snad přirozenější a čitelnější popis syntaxe jazyka.

Operátorové gramatiky

Definice 6.7 Nechť $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Jestliže platí:

(1) Žádná pravá strana pravidla z P neobsahuje dva sousední nonterminály.

(2) Mezi každými dvěma symboly z $(\Sigma \cup \{\sharp\})$ je definována nanejvýš jedna precedenční relace $\lessdot$ nebo $\doteq$ nebo $\gtrdot$ takto:

(a) $a \doteq b$ je-li $A \to \alpha a \gamma b \beta$ v P a $\gamma \in (N \cup \{\epsilon\})$

(b) $a \lessdot b$ je-li $A \to \alpha a B \beta$ v P a $B \Rightarrow^+ \gamma b \delta$, kde $\gamma \in (N \cup \{\epsilon\})$

(c) $a \gtrdot b$ je-li $A \to \alpha B b \gamma$ v P a $B \Rightarrow^+ \delta a \gamma$, kde $\gamma \in (N \cup \{\epsilon\})$

(d) $\sharp \lessdot a$ je-li $S \Rightarrow^+ \gamma a \alpha,\ \gamma \in (N \cup \{\epsilon\})$

(e) $a \gtrdot \sharp$ je-li $S \Rightarrow^+ \alpha a \gamma,\ \gamma \in (N \cup \{\epsilon\})$,

pak G se nazývá *operátorovou precedenční gramatikou.*

Příklad 6.7 Uvažujme gramatiku G s pravidly:

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to i \mid (E) \end{aligned}$$

Tato gramatika splňuje první podmínku z definice operátorové precedenční gramatiky. V tab. 6.5 jsou uvedeny precedenční relace definované mezi terminálními symboly gramatiky G podle definice operátorové precedenční gramatiky.

	$+$	$*$	$($	$)$	i	$\sharp$
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$*$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$($	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
$)$	$\gtrdot$	$\gtrdot$	$\doteq$	$\gtrdot$		$\gtrdot$
i	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
$\sharp$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

Tabulka 6.5: Precedenční matice

Protože precedenční relace jsou definovány jednoznačně, je G operátorovou precedenční gramatikou.

Operátorové precedenční jazyky jsou vlastní podmnožinou jednoduchých precedenčních jazyků. Nebylo by obtížně sestrojit deterministicky syntaktický analyzátor, jenž na základě precedenční matice pro operátorovou precedenční gramatiku identifikuje l-frázi a provádí příslušnou redukci. Výhodou této metody jsou menší nároky na paměť, protože precedenční relace jsou definovány pouze pro terminální symboly gramatiky.

6.7 Cvičení

Cvičení 6.7.1 Která z těchto gramatik je jednoduchá precedenční gramatika?

(a)

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{else}\ S \mid a \\ E &\to E\ \texttt{or}\ b \mid b \end{aligned}$$

(b)

$$\begin{aligned} S &\to AS \mid A \\ S &\to (S) \mid (). \end{aligned}$$

(c)

$$\begin{aligned} S &\to SA \mid A \\ A &\to (S) \mid (). \end{aligned}$$

Cvičení 6.7.2 Uvažujme gramatiku G, která popisuje seznamové struktury, např. $((a,(a,\wedge)),(a,a),(a))$:

$$\begin{aligned} S &\to a \mid \wedge \mid (T) \\ T &\to T,S \mid S \end{aligned}$$

Ukažte, že G není jednoduchá precedenční gramatika.

Cvičení 6.7.3 Gramatika G s pravidly

$$\begin{aligned} S &\to a \mid \wedge \mid (R) \\ T &\to S,T \mid S \\ R &\to T \end{aligned}$$

je ekvivalentní s gramatikou z cvičení 6.7.2. Pro gramatiku G vytvořte precedenční matici a precedenční funkce.

Kapitola 7

LL jazyky a jejich syntaktická analýza

LL jazyky představují vlastní podtřídu deterministických bezkontextových jazyků, pro které lze zkonstruovat deterministický syntaktický analyzátor. Analyzátory těchto jazyků realizují syntaktickou analýzu shora–dolů. Název LL je odvozen z charakteristiky analýzy: *Left* to right parse – rozklad zleva–doprava a *Left* parse – levý rozklad.

Jak uvidíme později, analyzátory LL jazyků nemají, z hlediska co nejobecnějšího použití, takové vlastnosti jako analyzátory LR jazyků. Přesto jsou však velmi rozšířené a oblíbené. Hlavním důvodem je mnohem menší pracnost a složitost jejich realizace.

7.1 Základní princip syntaktické analýzy LL jazyků

Předpokládejme, že $G = (N, \Sigma, P, S)$ je jednoznačná gramatika, a že řetězec $w = a_1 a_2 \dots a_n$ je větou z $L(G)$. Pak existuje jednoznačná posloupnost větných forem $\delta_0, \delta_1, \dots, \delta_m$:

$$\begin{array}{lcl} \delta_0 & = & S \\ \delta_r & \Rightarrow^{P_r} & \delta_{r+1},\ r = 0, 1, \dots, m-1 \\ \delta_m & = & w \end{array}$$

P_r značí index přepisovací pravidla, jež bylo použito pro expanzi nejlevějšího nonterminálu ve větné formě δ_r.

Vzhledem k tomu, že vytváříme levou derivaci věty w, je každá větná forma δ_r (s výjimkou δ_m) tvaru:

$$\delta_r = a_1 a_2 \dots a_j A \beta, \quad A \in N, \beta \in (N \cup \Sigma)^*,$$

přičemž terminální prefix $a_1 \dots a_j$ větné formy δ_r je shodný s prefixem věty w.

Jsou-li $A \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_l$ A-pravidla z P, pak základní problém deterministické syntaktické analýzy spočívá v jednoznačném výběru toho pravidla $A \to \alpha_i$, jehož aplikací dostaneme z větné formy δ_r příští větnou formu δ_{r+1}.

V modelu syntaktického analyzátoru shora–dolů, jenž je reprezentován zásobníkovým automatem zkonstruovaným podle věty 4.1, tento problém řešen není; výsledný automat je nedetermi-

nistický.

Uvažme však například gramatiku tvaru

$$G = (\{X, Y\}, \{a, b, c\}, P, X)$$

s pravidly

$$\begin{aligned} X &\rightarrow aXa \mid cYc \mid b \\ Y &\rightarrow aYbX \mid c \end{aligned}$$

a nějakou levou derivaci, např. věty *acacbbca*:

$$X \Rightarrow aXa \Rightarrow acYca \Rightarrow acaYbXca \Rightarrow acacbXca \Rightarrow acacbbca$$

Vidíme, že v této gramatice můžeme vytvořit deterministicky levou derivaci každé věty, protože výběr pravé strany pravidla pro nonterminál X i Y je zcela jednoznačně určen následujícím symbolem vstupní věty. Konkrétně, je-li

$$\delta_r = a_1 a_2 \ldots a_j A \beta$$

r-tá větná forma, pak výběr pravidla pro přechod k příští větné formě δ_{r+1} v závislosti na nejlevějším nonterminálu A a příštím vstupním symbolu a_{j+1} popisuje tabulka 7.1.

	$\mathbf{a_{j+1}}$		
A	a	b	c
S	$X \rightarrow aXa$	$X \rightarrow b$	$X \rightarrow cYc$
Y	$Y \rightarrow aYbX$		$Y \rightarrow c$

Tabulka 7.1:

Jednoznačné přiřazení přepisovacího pravidla k dvojici (nonterminál, příští vstupní symbol), tj. k řádku a sloupci v tab. 7.1, bylo velmi jednoduché, poněvadž každé pravidlo uvažované gramatiky začíná terminálním symbolem a tento jediný terminální symbol je dostačující k deterministickému výběru správné alternativy. Uvažujme nyní obecnější případ.

Předpokládejme, že pro expanzi nonterminálu A ve větné formě

$$\delta_r = a_1 \ldots a_j A \beta$$

máme k dispozici příštích k-vstupních symbolů $a_{j+1} \ldots a_{j+k}$ vstupní věty w a že žádné z A-pravidel

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_l$$

negeneruje prázdný řetězec, tj. neplatí $\alpha_i \Rightarrow^* \epsilon$ pro žádné α_i. Dále předpokládejme, že ke každé pravé straně α_i umíme nalézt množinu terminálních řetězců x, pro které platí

buď $\alpha_i \Rightarrow^* x$, kde $|x| < k$

nebo $\alpha_i \Rightarrow^* x\gamma$, kde $|x| = k$ a $\gamma \in (N \cup \Sigma)^*$,

t.j množinu terminálních řetězců délky nejvýše k, které tvoří prefixy derivací z α_i. Jestliže dovedeme na základě těchto množin zkonstruovaných ke každé pravé straně α_i jednoznačně určit, podle příštích k symbolů $a_{j+1} \ldots a_{j+k}$, kterého pravidla se má použít k expanzi nonterminálu A, pak je problém deterministické syntaktické analýzy opět vyřešen.

Příklad 7.1 Uvažujme gramatiku $G = (\{S, B, C\}, \{a, b, c, d\}, P, S)$ s pravidly

$$\begin{aligned} S &\rightarrow aB \mid aCC \\ B &\rightarrow Sb \mid b \\ C &\rightarrow cC \mid d \end{aligned}$$

Z tvaru S-pravidel vidíme, že větu generovanou v této gramatice nelze deterministicky syntakticky analyzovat na základě pouze jediného příštího vstupního symbolu (je-li tímto symbolem symbol a, pak nemůžeme rozhodnout, zda se má aplikovat pravidlo $S \rightarrow aB$, nebo $S \rightarrow aCC$). Uvažujeme tedy případ $k = 2$ a podle pravidel gramatiky sestrojme ke každé pravé straně pravidla množinu terminálních řetězců délky rovné nebo menší k, kterým začínají derivace z α:

A	aB	$\{aa, ab\}$
	aCC	$\{ac, ad\}$
B	Sb	$\{aa, ab, ac, ad\}$
	b	$\{b\}$
C	cC	$\{cc, cd\}$
	d	$\{d\}$

Tabulka 7.2:

Protože množiny příslušející A-pravidlům jsou disjunktní, je možné k danému nonterminálu jednoznačně přiřadit, na základě příštích dvou symbolů, přepisovací pravidlo, jehož se má použít. Toto přiřazení zobrazuje tabulka 7.3.

	aa	**ab**	**ac**	**ad**	**b**	**cc**	**cd**	**d**
S	$S \rightarrow aB$	$S \rightarrow aB$	$S \rightarrow aCC$	$S \rightarrow aCC$				
B	$B \rightarrow Sb$	$B \rightarrow Sb$	$B \rightarrow Sb$	$B \rightarrow Sb$	$B \rightarrow b$			
C						$C \rightarrow cC$	$C \rightarrow cC$	$C \rightarrow d$

Tabulka 7.3:

Doposud jsme předpokládali, že gramatika, pro níž hledáme deterministický syntaktický analyzátor shora–dolů, nemá ϵ-pravidla. Ukážeme nyní, že diskutovaná metoda, využívající k predikci pravidla pro derivaci příštích vstupních symbolů věty, může být aplikována také na gramatiku s ϵ-pravidly.

Uvažujme opět větnou formu

$$\delta_r = a_1 \dots a_j A \beta$$

a A-pravidla $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_l$. Předpokládejme, že jedno z A-pravidel může vést k náhradě nonterminálu A prázdným řetězcem, t.j pro některý řetězec α_i platí $\alpha_i \Rightarrow^* \epsilon$. Aplikací tohoto pravidla pak lze získat v následujících krocích větnou formu

$$\delta_{r+s} = a_1 \dots a_j \beta, \;\; s \geq 1.$$

Jak poznáme, kdy takový případ nastane a kdy tedy použít pravidla $A \rightarrow \alpha_i$. Povšimněme si, že příští vstupní symboly $a_{j+1}, aj + 2, \dots, a_{j+k}$ v takovém případě neodpovídají terminálnímu prefixu derivace z řetězce α_i, ale terminálnímu prefixu derivace z řetězce β. Tento prefix však

musí ležet v množině terminálních řetězců (délky rovné k nebo menší než k), které mohou následovat za nonterminálem A v některé větné formě a jak uvidíme později, tuto množinu umíme algoritmicky nalézt. Za předpokladu, že taková množina terminálních řetězců konečné délky určuje jednoznačně, kdy se má aplikovat pravidlo $A \to \alpha_i,\ \alpha_i \Rightarrow^* \epsilon$, pak je problém deterministické analýzy řešitelný i pro gramatiky, jež obsahují ϵ-pravidla.

Příklad 7.2 Uvažujme gramatiku $G = (\{S, B, C\}, \{a, b, c\}, P, S)$ s pravidly

$$\begin{aligned} S &\to aBb \mid bCa \mid c \\ B &\to cS \mid \epsilon \\ C &\to bB \mid a \end{aligned}$$

Nejprve spočítáme množiny řetězců délky 1 (nebo 0 v případě $\alpha_i \Rightarrow^* \epsilon$), kterými začínají derivace z pravých stran pravidel.

S	aBc	$\{a\}$
	bCa	$\{b\}$
	c	$\{c\}$
B	cS	$\{c\}$
	ϵ	$\{\epsilon\}$
C	bB	$\{b\}$
	a	$\{a\}$

Tabulka 7.4:

V důsledku pravidla $B \to \epsilon$ nalezneme množinu terminálních řetězců délku l, které se mohou objevit bezprostředně za nonterminálem B. Z pravidla $S \to aBb$ resp. z derivace $S \Rightarrow bCa \Rightarrow bbBa$ plyne, že tato množina obsahuje prvek b resp. a. Protože množina $\{a, b\}$ příslušející pravidlu $B \to \epsilon$ je disjunktní s množinou $\{c\}$, jež přísluší druhému B-pravidlu, a protože i množiny pro S-pravidla resp. C-pravidla jsou disjunktní, lze na základě jediného příštího vstupního symbolu jednoznačně vybrat pravidlo, jež se má v levé derivaci aplikovat. Přiřazení pravidel v závislosti na tomto příštím symbolu uvádí tab. 7.5.

	a	**b**	**c**
S	$S \to aBb$	$S \to bCa$	$S \to c$
B	$B \to \epsilon$	$B \to \epsilon$	$B \to cS$
C	$C \to a$	$C \to bB$	

Tabulka 7.5:

7.2 Výpočet množin FIRST a FOLLOW

V tomto odstavci ukážeme, jakým způsobem lze algoritmicky hledat v dané gramatice množiny terminálních řetězců, které umožňují predikci v deterministické syntaktické analýze shora–dolů. Nejdříve zavedeme jejich pojmenování a přesnou definici.

Konvence 7.1 Nechť Σ je abeceda a k nezáporné celé číslo. Symbolem Σ^{*k} značíme množinu řetězců nad abecedou Σ, jejíchž délka je menší nebo rovna k, tj. $\Sigma^{*k} = \{x \mid x \in \Sigma^* \wedge |x| \leq k\}$.

Definice 7.1 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Množinu $\mathrm{FIRST}_k(\alpha),\ \alpha \in (N \cup \Sigma)^*$ definujeme takto:

$$\mathrm{FIRST}_k(\alpha) = \{x \mid x \in \Sigma^{*k} \text{ a buď } |x| < k \wedge \alpha \stackrel{*}{\Rightarrow} x \text{ nebo } |x| = k \wedge \alpha \stackrel{*}{\Rightarrow} x\gamma \text{ pro nějaké } \gamma \in (N \cup \Sigma)^*\}$$

Příklad 7.3 V gramatice z příkladu 7.2 (viz tab. 7.4) je

$$\mathrm{FIRST}_1(aBc) = \mathrm{FIRST}_1(a) = \{a\}$$
$$\mathrm{FIRST}_1(B) = \{c, \epsilon\}$$

Definice 7.2 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Množinu $\mathrm{FOLLOW}_k(A),\ A \in N$ definujeme takto:

$$\mathrm{FOLLOW}_k(A) = \{x \mid x \in \Sigma^{*k} \wedge S \stackrel{*}{\Rightarrow} \alpha A \beta \wedge x \in \mathrm{FIRST}_k(\beta), \beta \in (N \cup \Sigma)^*\}$$

Příklad 7.4 V gramatice z příkladu 7.2, jež má pravidla

$$\begin{aligned} S &\rightarrow aBb \mid bCa \mid c \\ B &\rightarrow cS \mid \epsilon \\ C &\rightarrow bB \mid a \end{aligned}$$

je $\mathrm{FOLLOW}_1(S) = \{a, b\}$, protože existují derivace

$$\begin{aligned} S &\Rightarrow aBb \Rightarrow acSb \\ S &\Rightarrow bCa \Rightarrow bbBa \Rightarrow bbcSa; \\ &\quad \mathrm{FOLLOW}_1(B) = \{a, b\}, \end{aligned}$$

protože existují derivace

$$\begin{aligned} S &\Rightarrow aBb \\ S &\Rightarrow bCa \Rightarrow bbBa; \\ &\quad \mathrm{FOLLOW}_1(C) = \{a\} \end{aligned}$$

protože existuje derivace

$$S \Rightarrow bCa.$$

Jiné derivace, jež by měly vliv na další prvky uvedených množin FOLLOW neexistují.

Konvence 7.2 V případě, že $k = 1$ budeme psát namísto $\mathrm{FIRST}_1(\alpha)$ nebo $\mathrm{FOLLOW}_1(A)$ pouze $\mathrm{FIRST}(\alpha)$ nebo $\mathrm{FOLLOW}(A)$.

Poznámka 7.1 Z definice 7.2 plyne, že množina $\mathrm{FOLLOW}_k(A)$ obsahuje prázdný řetězec ϵ, právě když existuje v gramatice větná forma, jejímž posledním symbolem je nonterminál A, t.j $S \Rightarrow^* \alpha A$ implikuje $\epsilon \in \mathrm{FOLLOW}_k(A)$.

Je jasné, že pro praktické aplikace nepřichází v úvahu způsob výpočtu množin FIRST_k a FOLLOW_k, který jsme používali doposud v příkladech 7.1–7.4. Dříve, než formulujeme algoritmy pro výpočet těchto množin, předešleme, že dominantní praktický význam ve třídě diskutovaných gramatik má případ $k = 1$, a že gramatika, ke které hledáme deterministický syntaktický analyzátor shora–dolů, nesmí obsahovat levou rekurzi.

Abychom nalezli množinu $\mathrm{FIRST}(\alpha),\ \alpha = X_1 X_2 \ldots X_m$, je třeba vypočítat množiny $\mathrm{FIRST}(X)$ pro všechny symboly X řetězce α. Množiny $\mathrm{FIRST}(X)$ jsou definovány podle pravidel, jež plynou z definice 7.1:

1. Je-li X terminál, pak FIRST$(X) = \{X\}$
2. Je-li X nonterminál a $X \to a\alpha$ pravidlo, pak k množině FIRST(X) přidej terminál a.
   Je-li $X \to \epsilon$ pravidlo, pak k FIRST(X) přidej ϵ.
3. Je-li $X \to Y_1 Y_2 \ldots Y_m$ pravidlo, pak pro všechna i, pro která platí $Y_j \in N$ a FIRST(Y_j) obsahuje ϵ pro $j = 1, \ldots, i-1$, přidej k FIRST(X) prvky množiny FIRST(Y_i) různé od ϵ. Platí-li $\epsilon \in$ FIRST(Y_j) pro všechna $j = 1, \ldots, m$, pak k FIRST(X) přidej také ϵ.

Známe-li množiny FIRST(X_i), pak FIRST$(X_1 X_2 \ldots X_m)$ vypočítáme takto:

Množina FIRST$(X_1 \ldots X_m)$ bude obsahovat všechny symboly z FIRST(X_1) různé od ϵ. Jestliže FIRST(X_1) obsahuje ϵ, pak k FIRST$(X_1 \ldots X_m)$ přidej všechny symboly z FIRST(X_2) různé od ϵ. Jestliže FIRST(X_1) i FIRST(X_2) obsahuje ϵ, pak k FIRST$(X_1 \ldots X_m)$ přidej symboly z FIRST(X_3) různé od ϵ, atd. Nakonec k FIRST$(X_1 \ldots X_m)$ přidáme ϵ v případě, že všechny množiny FIRST(X_i), $i = 1, \ldots, m$, obsahují ϵ.

Na základě tohoto postupu můžeme formulovat algoritmus výpočtu množiny FIRST(α), jenž je vhodný zejména pro jeho programovou realizaci.

Algoritmus 7.1 Výpočet množiny FIRST(α).

Vstup: Gramatika $G = (N, \Sigma, P, S)$ bez levé rekurze a řetězec $\alpha = X_1 X_2 \ldots X_n$, $\alpha \in (N \cup \Sigma)^*$.

Výstup: Množina FIRST(α).

Metoda:

(1) Vypočítej množinu $N_\epsilon = \{A \mid A \Rightarrow^* \epsilon\}$.

(2) Vytvoř binární relaci F na množině $N \cup \Sigma$ takto:

$$F = \{(X, Y) \mid X \to \alpha Y \beta \text{ je v } P, Y \in (N \cup \Sigma),\ \alpha \in N_\epsilon^*,\ \beta \in (N \cup \Sigma)^*\}$$

(3) Vytvoř binární relaci H na množině $N \cup \Sigma \cup \{\epsilon\}$ takto:

$$H = F^+ \cup \{(X, \epsilon) \mid X \in N_\epsilon\} \cup \{(a, a) \mid a \in (\Sigma \cup \{\epsilon\})\},$$

kde F^+ je tranzitivní uzávěr relace F.

(4) Polož FIRST$(X_i) = \{a \mid (X_i, a) \in H,\ X_i \in (N \cup \Sigma \cup \{\epsilon\}),\ a \in (\Sigma \cup \{\epsilon\})\}$ pro $i = 1, \ldots, n$.

(5) Polož $i = 1$ a FIRST$(\alpha) = \emptyset$.

(6) FIRST(α) := FIRST$(\alpha) \cup$ FIRST(X_i).

(7) Je-li $i = n$ nebo $i < n \wedge \epsilon \notin$ FIRST(X_i), pak FIRST(α) je výsledná množina. V opačném případě FIRST(α) := FIRST$(\alpha) - \{\epsilon\}$ a vrať se ke kroku (6).

Poznámka 7.2 Formulace algoritmu 7.1 převádí těžiště výpočtu množiny FIRST(α) do konstrukce tranzitivního uzávěru relace F, pro níž můžeme využít operací s booleovskými maticemi. Poznamenejme, že relace H definuje pro symbol $X \in (N \cup \Sigma \cup \{\epsilon\})$ ty symboly Y z množiny $(N \cup \Sigma \cup \{\epsilon\})$, pro něž platí $X \Rightarrow^* Y\alpha$, $\alpha \in (N \cup \Sigma)^*$.

Příklad 7.5 Pro pravé strany α pravidel gramatiky $G = (\{A, B, C\}, \{a, b, c\}, P, A)$,

$$\begin{aligned} A &\to BCb \mid ab \\ E &\to bB \mid \epsilon \\ T &\to cA \mid \epsilon \end{aligned}$$

spočítejte podle algoritmu 7.1 množiny FIRST(α).

(1) Zřejmě je $N_\epsilon = \{B, C\}$.

(2) Relaci F reprezentujeme booleovskou maticí

$$[F] = \begin{array}{c} \\ A \\ B \\ C \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & a & b & c \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

(3) Vypočítáme matici H

$$[H] = \begin{array}{c} \\ A \\ B \\ C \\ a \\ b \\ c \\ \epsilon \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & a & b & c & \epsilon \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

(4) Dostáváme:

$$\begin{aligned} \text{FIRST}(A) &= \{a, b, c\} \\ \text{FIRST}(B) &= \{b, \epsilon\} \\ \text{FIRST}(C) &= \{c, \epsilon\} \\ \text{FIRST}(\epsilon) &= \{\epsilon\} \\ \text{FIRST}(X) &= \{X\} \text{ pro } X = a, b, c \end{aligned}$$

(5)–(7) Dostáváme:

$$\begin{aligned} \text{FIRST}(BCb) &= (\text{FIRST}(B) - \{\epsilon\}) \cup (\text{FIRST}(C) - \{\epsilon\}) \cup \text{FIRST}(b) = \{b, c\} \\ \text{FIRST}(aB) &= \{a\} \\ \text{FIRST}(bB) &= \{b\} \\ \text{FIRST}(\epsilon) &= \{\epsilon\} \\ \text{FIRST}(cA) &= \{c\} \end{aligned}$$

Výpočet množiny FOLLOW(A)

Při výpočtu množiny FOLLOW(A), pro všechny nonterminály $A \in N$ můžeme postupovat podle těchto pravidel:

1. FOLLOW(S), kde S je výchozí symbol gramatiky, obsahuje ϵ.
2. Je-li $A \to \alpha B\beta, \beta \neq \epsilon$, pravidlo gramatiky pak všechny prvky z FIRST(β), vyjma ϵ, jsou obsaženy v FOLLOW(B).
3. Jsou-li $A \to \alpha B$ nebo $A \to \alpha B\beta, \beta \Rightarrow^* \epsilon$ pravidla gramatiky, pak všechny prvky z FOLLOW(A) jsou ve FOLLOW(B).

Příklad 7.6 Uvažujme gramatiku z předchozího příkladu 7.5, jež má pravidla

$$\begin{array}{rcl} A & \to & BCd \mid aB \\ B & \to & bB \mid \epsilon \\ C & \to & cA \mid \epsilon \end{array}$$

Položme FOLLOW(A) = FOLLOW(B) = FOLLOW(C) = $\emptyset$

1. FOLLOW(A) := $\{\epsilon\}$
2. $$\begin{array}{rcll} \text{FOLLOW}(B) & := & \text{FOLLOW}(B) \cup \text{FIRST}(Cb) & [\text{pravidlo } A \to BCb] \\ \text{FOLLOW}(C) & := & \text{FOLLOW}(C) \cup \text{FIRST}(b) & [\text{pravidlo } A \to BCb] \end{array}$$
3. $$\begin{array}{rcll} \text{FOLLOW}(B) & := & \text{FOLLOW}(B) \cup \text{FOLLOW}(A) & [\text{pravidlo } A \to aB] \\ \text{FOLLOW}(A) & := & \text{FOLLOW}(A) \cup \text{FOLLOW}(C) & [\text{pravidlo } C \to cA] \end{array}$$

tedy

$$\begin{array}{rcl} \text{FOLLOW}(A) & = & \{\epsilon, b\} \\ \text{FOLLOW}(B) & = & \{b, c, \epsilon\} \\ \text{FOLLOW}(C) & = & \{b\} \end{array}$$

Algoritmus 7.2 Výpočet množiny FOLLOW(A).

Vstup: Gramatika $G = (N, \Sigma, P, S)$, $A \in N$.

Výstup: Množina FOLLOW(A).

Metoda:

(1) Vypočítej množinu $N_\epsilon = \{A \mid A \Rightarrow^* \epsilon\}$

(2) Vytvoř relaci L na množině $(N \cup \Sigma)$ takto:

$$L = \{(X, Y) \mid X \to \alpha Y \beta, \text{ je pravidlo z } P, \beta \in N_\epsilon^*, \alpha \in (N \cup \Sigma)\}$$

(3) Vypočítej relaci $(L^{-1})^*$.

(4) Polož $R_A = \{X \mid (A, X) \in (L^{-1})^*\}$, FOLLOW$(A) = \emptyset$.
Množina R_A obsahuje zřejmě právě ty nonterminály, ze kterých lze derivovat řetězce končící nonterminálem A (ve smyslu relace $\Rightarrow^*$).

(5) Pro každý výskyt nonterminálu $X \in R_A$ na pravé straně $\alpha \times \beta$ pravidla proveď:

$$\text{FOLLOW}(A) := \text{FOLLOW}(A) \cup (\text{FIRST}(\beta) - \{\epsilon\})$$

(6) Je-li $S \in R_A$, pak FOLLOW$(A) :=$ FOLLOW$(A) \cup \{\epsilon\}$

Příklad 7.7 Pro všechny nonterminály gramatiky $G = (\{A, B, C\}, \{a, b, c\}, P, S)$,

$$\begin{aligned} A &\to aBC \mid bB \\ B &\to CbA \mid c \mid \epsilon \\ C &\to aBa \mid cBaC \mid \epsilon \end{aligned}$$

spočítejte podle algoritmu 7.2 množinu FOLLOW.

(1) $N_\epsilon = \{B, C\}$.

(2)

$$[L] = \begin{array}{c} \\ A \\ B \\ C \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & a & b & c \end{array} \\ \left[\begin{array}{cccccc} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

(3)

$$[L^{-1*}] = \begin{array}{c} \\ A \\ B \\ C \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & a & b & c \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

(4)

$$\begin{aligned} R_A &= \{A, B\} \\ R_B &= \{A, B\} \\ R_C &= \{A, B, C\} \end{aligned}$$

(5)

$$\begin{aligned}
\text{FOLLOW}(A) &= (\text{FIRST}(C) - \{\epsilon\}) \cup \text{FIRST}(a) \cup \text{FIRST}(aC) \\
\text{FOLLOW}(B) &= (\text{FIRST}(C) - \{\epsilon\}) \cup \text{FIRST}(a) \cup \text{FIRST}(aC) \\
\text{FOLLOW}(C) &= (\text{FIRST}(C) - \{\epsilon\}) \cup \text{FIRST}(a) \cup \text{FIRST}(aC) \cup \text{FIRST}(bA)
\end{aligned}$$

Dosadíme-li nyní hodnoty množin FIRST, $(\text{FIRST}(C) = \{a, c, \epsilon\})$, pak s přihlédnutím k bodu (6) dostáváme výsledné množiny FOLLOW:

$$\begin{aligned}
\text{FOLLOW}(A) &= \{\epsilon, a, c\} \\
\text{FOLLOW}(B) &= \{\epsilon, a, c\} \\
\text{FOLLOW}(C) &= \{\epsilon, a, b, c\}
\end{aligned}$$

7.3 Definice $LL(k)$-gramatiky

Definice 7.3 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Říkáme, že G je $LL(k)$ *gramatika* pro nějaké pevné $k \geq 0$, jestliže z libovolných dvou levých derivací tvaru

$$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_1\beta \overset{*}{\Rightarrow} wx \quad \text{a}$$

$$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_2\beta \overset{*}{\Rightarrow} wy$$

takových, že $\text{FIRST}_k(x) = \text{FIRST}_k(y)$, plyne $\alpha_1 = \alpha_2$.

Definice 7.3 říká, že G je $LL(k)$ gramatika, jestliže pro danou větnou formu tvaru $wA\beta$ a pro prvních k terminálních symbolů, které mají být vygenerovány z $A\beta$, existuje v gramatice G právě jedno přepisovací pravidlo $A \to \alpha$.

Definice 7.4 Je-li G $LL(k)$ gramatika pro nějaké k, pak říkáme, že G je LL gramatika. Jazyk generovaný $LL(k)$ gramatikou nazýváme $LL(k)$ jazykem nebo, nespecifikujeme-li hodnotu k, také LL jazykem.

Příklad 7.8 Uvažujme gramatiku

$$\begin{aligned}
S &\to aAS \mid b \\
A &\to a \mid bSA
\end{aligned}$$

Intuitivně G je $LL(1)$ gramatika, poněvadž pro libovolnou větnou formu s nejlevějším nonterminálem C a vstupní řetězec s příštím terminálem c existuje pouze jedno pravidlo, které derivuje terminální řetězec začínající symbolem c. Skutečně, začínají-li x a y terminálem a, použilo se pravidla $S \to aAS (\alpha_1 = \alpha_2 = aAS)$; začínají-li x a y terminálem b, pak se použilo pravidla $S \to b (\alpha_1 = \alpha_2 = b)$. Poněvadž z S nemůže být derivován prázdný řetězec ϵ, není možné, aby platilo, že $x = y = \epsilon$. Analogický rozbor lze provést pro tuto dvojici derivací: $S \Rightarrow^* wA\beta \Rightarrow w\alpha_1\beta \Rightarrow^* wx$ a $S \Rightarrow^* wA\beta \Rightarrow w\alpha_2\beta \Rightarrow^* wy$. Tato gramatika je příkladem tzv. jednoduché $LL(1)$ gramatiky.

Definice 7.5 Gramatika G, jež nemá žádná ϵ-pravidla, se nazývá *jednoduchou* $LL(1)$ *gramatikou*, začíná-li pro každý nonterminál A každá pravá strana terminálním symbolem a jsou-li tyto terminály v rámci A-pravidel vzájemně různé.

Gramatika z předchozího příkladu, stejně jako gramatika, jejíž pravidla jsou uvedena v tabulce 7.1, je jednoduchou $LL(1)$ gramatikou. V jednoduché $LL(1)$ gramatice tedy k dané dvojici $(A, a), A \in N, a \in \Sigma$ existuje nanejvýš jedno pravidlo tvaru $A \to a\alpha$.

Věta 7.1 Nechť $G = (N, \Sigma, P, S)$ je gramatika. G je $LL(k)$ gramatika právě, když platí: Jestliže $A \to \alpha_1$ a $A \to \alpha_2$ jsou dvě různá pravidla z P, pak $\text{FIRST}_k(\alpha_1\beta) \cap \text{FIRST}_k(\alpha_2\beta) = \emptyset$ pro všechna β, pro která platí $S \Rightarrow^* wA\beta$.

Důkaz:

Část „jen když“:

Předpokládejme, že $\text{FIRST}_k(\alpha_1\beta) \cap \text{FIRST}_k(\alpha_2\beta)$ obsahuje řetězec x. Pak podle definice množiny FIRST existují derivace

$$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_1\beta \Rightarrow wxy$$

$$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_2\beta \Rightarrow wxz$$

pro nějaké y, z.

Je-li $|x| < k$, pak $y = z = \epsilon$. Protože $\alpha_1 \neq \alpha_2$, G není $LL(k)$ gramatikou.

Část „když“:

Předpokládejme, že G není $LL(k)$. Pak existují dvě derivace

$$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_1\beta \overset{*}{\Rightarrow} wx,$$

$$S \overset{*}{\Rightarrow} wA\beta \Rightarrow w\alpha_2\beta \overset{*}{\Rightarrow} wy$$

takové, že řetězce x a y jsou shodné v prvních k symbolech, avšak $\alpha_1 \neq \alpha_2$. Pak $A \to \alpha_1$ a $A \to \alpha_2$ jsou různá pravidla v P a $\text{FIRST}_k(\alpha_1\beta)$ i $\text{FIRST}_k(\alpha_2\beta)$ obsahuje $\text{FIRST}_k(x) = \text{FIRST}_k(y)$.

Věta 7.1 má důležitou aplikaci na určitou třídu $LL(1)$ gramatik: □

Věta 7.2 Nechť $G = (N, \Sigma, P, S)$ je gramatika, která nemá žádná ϵ-pravidla. G je $LL(1)$ gramatika, právě když pro všechny nonterminály A z N má každá množina A-pravidel $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ tu vlastnost, že množiny $\text{FIRST}(\alpha_1)$, $\text{FIRST}(\alpha_2)$, $\ldots$, $\text{FIRST}(\alpha_n)$ jsou všechny po dvojicích disjunktní.

Důkaz: Tvrzení je bezprostředním důsledkem věty 7.1 a skutečností, že G neobsahuje ϵ-pravidla. V tom případě je $\text{FIRST}_1(\alpha\beta) = \text{FIRST}_1(\alpha)$ pro všechna $S \Rightarrow^* \gamma A\beta$ a $A \to \alpha$. □

Ve větě 7.2 byl předpoklad, že G neobsahuje žádné ϵ-pravidlo, podstatný. Dříve, než uvedeme větu, která stanovuje $LL(1)$ podmínku pro gramatiku, která připouští i ϵ-pravidla, rozšíříme definici množiny FIRST (definice 7.1) takovým způsobem, aby ji bylo možno konstruovat i pro množinu n řetězců.

Definice 7.6 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť $L \subseteq (N \cup \Sigma)^*$. Pak

$$\text{FIRST}_k(L) = \bigcup_{\alpha \in L} \text{FIRST}_k(\alpha)$$

Věta 7.3 Gramatika $G = (N, \Sigma, P, S)$ je $LL(1)$ gramatikou právě když platí implikace: Jsou-li $A \to \alpha_1$ a $A \to \alpha_2$ dvě různá pravidla pro libovolné $A \in N$, pak

$$\text{FIRST}(\alpha_1\text{FOLLOW}(A)) \cap \text{FIRST}(\alpha_2\text{FOLLOW}(A)) = \emptyset.$$

Důkaz: Důkaz se provede analogicky důkazu věty 7.1. Ponecháme jej jako cvičení.

Na základě věty 7.3 lze formulovat dvě nutné a postačující podmínky pro $LL(1)$ gramatiky. □

Věta 7.4 Gramatika G je $LL(1)$ gramatikou, právě když pro každou množinu A-pravidel $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ platí tyto podmínky:

(1) $\text{FIRST}(\alpha_1), \text{FIRST}(\alpha_2), \ldots, \text{FIRST}(\alpha_n)$ jsou po dvojicích disjunktní (podmínka FF)

(2) Jestliže $\alpha_i \Rightarrow^* \epsilon$, pak $\text{FIRST}(\alpha_j) \cap \text{FOLLOW}(A) = \emptyset$ pro $1 \leq j \leq n, i \neq j$ (podmínka FFL)

Připomeňme, že relace $\alpha_i \Rightarrow^* \epsilon$ může platit nanejvýš pro jediné pravidlo $A \to \alpha_i$. V opačném případě by nebyla splněna podmínka FF, poněvadž by průnik dvojice množin FIRST obsahoval ϵ.

Příklad 7.9 Uvažujme gramatiku $G = (\{E, T, F\}, \{i, +, *, (,)\}, P, S)$ s pravidly

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to i \mid (E) \end{aligned}$$

popisující strukturu aritmetických výrazů. Tato gramatika nemůže být LL gramatikou, protože obsahuje levou rekurzi. Po odstranění levé rekurze (viz příklad 4.14) dostaneme gramatiku s pravidly

$$\begin{aligned} E &\to TE' \mid T \\ E' &\to +TE' \mid +T \\ T &\to FT' \mid F \\ T' &\to *FT' \mid *F \\ F &\to (E) \mid i \end{aligned}$$

Vidíme, že ani tato gramatika není LL gramatikou, protože např. průnik $\text{FIRST}_k(TE') \cap \text{FIRST}_k(T)$ je neprázdný pro libovolné k.

Připustíme-li, že daná gramatika může obsahovat ϵ-pravidla, pak můžeme uvedená pravidla v jistém smyslu zjednodušit:

$$\begin{aligned} E &\to TE' \\ E' &\to +TE' \mid \epsilon \\ T &\to FT' \\ T' &\to *FT' \mid \epsilon \\ F &\to (E) \mid i \end{aligned}$$

Podle věty 7.4 nyní vyšetříme, zda tato gramatika je $LL(1)$ gramatikou. Nejdříve musíme spočítat množiny FIRST a FOLLOW:

$$\begin{aligned}
\mathrm{FIRST}(E) &= \mathrm{FIRST}(T) = \mathrm{FIRST}(F) = \{(, i\} \\
\mathrm{FIRST}(E') &= \{+, \epsilon\} \\
\mathrm{FIRST}(T') &= \{*, \epsilon\} \\
\mathrm{FOLLOW}(E) &= \mathrm{FOLLOW}(E') = \{), \epsilon\} \\
\mathrm{FOLLOW}(T) &= \mathrm{FOLLOW}(T') = \{+,), \epsilon\} \\
\mathrm{FOLLOW}(F) &= \{+, *,), \epsilon\}
\end{aligned}$$

Vidíme, že F-pravidla neobsahují ϵ-pravidlo a splňují podmínku FF. E'-pravidla i T'-pravidla splňují jak podmínku FF, tak i FFl a tudíž je tato gramatika $LL(1)$ gramatikou.

Definice 7.7 Nechť G je gramatika. Jestliže pro každá dvě různá A-pravidla $A \to \alpha_1$ a $A \to \alpha_2$ z P platí

$$\mathrm{FIRST}_k(\alpha_1 \mathrm{FOLLOW}_k(A)) \cap \mathrm{FIRST}_k(\alpha_2 \mathrm{FOLLOW}_k(A)) = \emptyset,$$

pak G se nazývá *silná $LL(k)$ gramatika.*

Z věty 7.3 plyne, že každá $LL(1)$ gramatika je silná $LL(1)$ gramatika. Tento poznatek, jak ilustruje následující příklad, nelze zobecnit pro $k > 1$.

Příklad 7.10 Uvažujme gramatiku G s pravidly

$$\begin{aligned}
S &\to aAaa \mid bAba \\
A &\to b \mid \epsilon
\end{aligned}$$

Podle věty 7.1 se můžeme snadno přesvědčit, že G je $LL(2)$ gramatikou. Spočítáme-li však $\mathrm{FOLLOW}_2(A) = \{aa, ba\}$ a vyhodnotíme-li podmínku pro silné $LL(k)$ gramatiky, dostaneme pro A-pravidla.

$$\mathrm{FIRST}_2(b\ \mathrm{FOLLOW}_2(A)) \cap \mathrm{FIRST}_2(\mathrm{FOLLOW}_2(A)) = \{ba\}$$

a vidíme tedy, že G není silná $LL(2)$ gramatika.

Věta 7.5 Nechť G je gramatika, jež obsahuje levou rekurzi. Pak G není $LL(k)$ gramatika pro žádné k.

Důkaz: Podle definice levé rekurze existuje v G pro některý nonterminál A derivace $A \Rightarrow^+ A\alpha$. Předpokládejme, že G nemá zbytečné symboly, takže platí $A \Rightarrow^+ u$ a $\alpha \Rightarrow^* v$ pro jisté řetězce $u, v \in \Sigma^*$. Pak můžeme zkonstruovat tyto derivace:

$$S \overset{*}{\Rightarrow} wA\delta \overset{+}{\Rightarrow} w\alpha^k\delta \overset{+}{\Rightarrow} wuv^kx \quad \text{a}$$

$$S \overset{*}{\Rightarrow} wA\delta \overset{+}{\Rightarrow} wA\alpha^k\delta \overset{+}{\Rightarrow} wA\alpha^{k+1}\delta \overset{+}{\Rightarrow} wuv^{k+1}x$$

Protože však $\mathrm{FIRST}_k(uv^kx) = \mathrm{FIRST}_k(uv^{k+1}x)$ pro libovolné k, nemůže být $LL(k)$ gramatikou. □

Následující věta shrnuje některé důležité poznatky o $LL(k)$ gramatikách a jazycích.

Věta 7.6 (1) Každá $LL(k)$ gramatika je jednoznačná.

(2) Je-li G bezkontextová gramatika, která není $LL(k)$ gramatikou pro dané k_1, pak je nerozhodnutelné, zda G má ekvivalentní $LL(k)$ gramatiku pro některé $k_2 > k_1$.

(3) Pro $k \geq 0$ jsou $LL(k)$ jazyky vlastní podmnožinou $LL(k+1)$ jazyků.

Tvrzení (2) a (3) vylučují existenci algoritmu, který vždy transformuje danou bezkontextovou gramatiku na $LL(k)$ gramatiku, nebo danou $LL(k)$ gramatiku na $LL(1)$ gramatiku. V odstavci 7.5 ukážeme, jakým způsobem lze však v některých případech dílčími transformacemi získat $LL(1)$ gramatiku. S jednou z nich jsme se již setkali v příkladě 7.8; je to odstranění levé rekurze.

7.4 Algoritmus syntaktické analýzy pro $LL(k)$ gramatiky

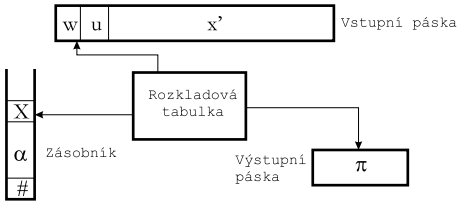
Algoritmus syntaktické analýzy pro $LL(k)$ gramatiky patří do skupiny tzv. prediktivních algoritmů rozkladu, protože provádí predikci přepisovacích pravidel podle příštích terminálních symbolů analyzované věty. Výstupem tohoto algoritmu je levý rozklad, jehož definice je analogická definici pravého rozkladu.

Definice 7.8 Nechť G je gramatika a nechť všechna pravidla z P jsou očíslována indexy $1, 2, \ldots, p$. Nechť $\alpha \in (N \cup \Sigma)^*$ je větná forma v G. *Levým rozkladem* řetězce α nazýváme posloupnost indexů přepisovacích pravidel, jež byla postupně aplikována v levé derivaci větné formy α, tj. je-li

$$S \overset{p_{i_1}}{\Rightarrow} \delta_1 \overset{p_{i_2}}{\Rightarrow} \delta_2 \overset{p_{i_3}}{\Rightarrow} \ldots \overset{p_{i_n}}{\Rightarrow} \delta_n = \alpha$$

levá derivace větné formy α, pak $\pi = p_{i_1} p_{i_2} \ldots p_{i_n}$ je levý rozklad této větné formy.

Algoritmus rozkladu pro $LL(k)$ gramatiky používá vstupní pásky, zásobníku, tzv. rozkladové tabulky a výstupní pásky (viz obr. 7.1). Podobně, jako v algoritmu 6.2, použijeme k jeho popisu konfigurací a přechodů mezi konfiguracemi.



Obrázek 7.1: Prediktivní algoritmus rozkladu

Na vstupní pásce je uložen řetězec terminálních symbolů (vstupních symbolů) wx, jehož levý rozklad π, existuje-li, bude zobrazen na výstupní pásce. Položme $wx = wux'$.

Podřetězec w reprezentuje již zpracovanou část vstupního řetězce, x je zbývající část vstupního řetězce; podřetězec $u = \mathrm{FIRST}_k(x)$ reprezentuje prefix (délky k) řetězce x.

Zásobník obsahuje řetězec $X\alpha\sharp$, kde $X\alpha$ je řetězec nad abecedou zásobníku, kterou označíme symbolem Γ. Speciální koncový znak $\sharp$ indikuje konec zásobníku. Symbol X je na vrcholu zásobníku.

Výstupní páska obsahuje řetězec π, jenž je tvořen čísly (indexy) přepisovacích pravidel. Konfigurace algoritmu je dána trojicí $(x, X\alpha, \pi)$, kde:

1) x je dosud nezpracovaná část vstupního řetězce

2) $X\alpha$ je obsah zásobníku (X je vrchol)

3) π je řetězec na výstupní pásce

Přechod z jedné konfigurace do druhé je předepsán rozkladovou tabulkou M, která reprezentuje zobrazení z $(\Gamma \cup \{\sharp\}) \times \Sigma^{*k}$ do množiny obsahující tyto prvky:

1) (β, i), kde β je řetězec z Γ^*, i je číslo přepisovacího pravidla

2) *pop* (indikuje odstranění symbolu ze zásobníku)

3) *accecpt* (indikuje přijetí vstupního řetězce)

4) *error* (indikuje syntaktickou chybu)

Nyní, je-li na vrcholu zásobníku symbol X a je-li $u = \mathrm{FIRST}_k(x)$ podřetězec reprezentující k příštích symbolů vstupního řetězce, pak $M(X, u)$ předepisuje tyto typy přechodů (přechod budeme označovat symbolem $\vdash$):

(1) $(x, X\alpha, \pi) \vdash (x, \beta\alpha, \pi i)$, je-li $M(X, u) = (\beta, i)$

Symbol X na vrcholu zásobníku je přepsán řetězcem β; k řetězci π je připojen index i. Není čten příští vstupní symbol.

(2) $(x, X\alpha, \pi) \vdash (x', \alpha, \pi)$, je-li $M(X, u) = pop,\ x = ax', X = a$

Shoduje-li se první symbol řetězce x se symbolem X na vrcholu zásobníku, je symbol z vrcholu odstraněn a čtecí hlava je posunuta o jeden symbol doprava.

(3) Přejde-li algoritmus do konfigurace $(\epsilon, \sharp, \pi)$, pak rozklad končí. Předpokládáme, že $M(\sharp, \epsilon)$ je vždy *accept*. Řetězec π je levým rozkladem původního vstupního řetězce.

(4) Jestliže algoritmus rozkladu přejde do konfigurace $(x, X\alpha, \pi)$ a $M(X, u) = error$, pak vstupní řetězec není syntakticky správně utvořen.

Příklad 7.11 Ilustrujme činnost uvedeného algoritmu na příkladě $LL(1)$ gramatiky z příkladu 7.8, jež má pravidla:

(1) $S \rightarrow aAS$
(2) $S \rightarrow b$
(3) $A \rightarrow a$
(4) $A \rightarrow bSA$

Prediktivní algoritmus používá rozkladové tabulky, tab. 7.6.

$u = \text{FIRST}(x)$

Symbol X na vrcholu zásobníku

	a	b	ϵ
S	$aAS, 1$	$b, 2$	*error*
A	$a, 3$	$bSA, 4$	*error*
a	*pop*	*error*	*error*
b	*error*	*pop*	*error*
$\sharp$	*error*	*error*	*accept*

Tabulka 7.6:

Bude-li na vstupní pásce uložen řetězec *abbab*, pak algoritmus projde těmito konfiguracemi:

$$\begin{aligned}
(abbab, S\sharp, \epsilon) &\vdash (abbab, aAS\sharp, 1) \\
&\vdash (bbab, AS\sharp, 1) \\
&\vdash (bbab, bSAS\sharp, 14) \\
&\vdash (bab, SAS\sharp, 14) \\
&\vdash (bab, bAS\sharp, 142) \\
&\vdash (ab, AS\sharp, 142) \\
&\vdash (ab, aS\sharp, 1423) \\
&\vdash (b, S\sharp, 1423) \\
&\vdash (b, b\sharp, 14232) \\
&\vdash (\epsilon, \sharp, 14232)
\end{aligned}$$

Pro první přechod je $M(S, a) = (aAS, 1)$, takže vrchol zásobníku S je přepsán řetězcem aAS; na výstupní pásku je zapsána 1. Pro další přechod je $M(a, a) = pop$, což odpovídá odstranění symbolu a z vrcholu zásobníku a posuvu čtecí hlavy o jeden symbol doprava. Pokračujeme-li takto, dosáhneme koncové konfigurace $(\epsilon, \sharp, 14232)$. Řetězec *abbab* je tedy větou jazyka $L(G)$. Jeho levý rozklad je $\pi = 14232$.

Z tab. 7.6 vidíme, že rozkladová tabulka pro $LL(1)$ gramatiku má jednoduchý tvar. Popsaný prediktivní algoritmus provádí pro záznamy typu (β, i) přímou derivaci $X \Rightarrow \beta$, (X je vrchol zásobníku), podle i-tého pravidla $X \Rightarrow \beta$. Z toho pak vyplývá, že v případě $LL(1)$ gramatiky je abeceda zásobníku $\Gamma = (N \cup \Sigma \cup \{\sharp\})$. Nad touto abecedou je také definována tabulka M.

Ke každé $LL(k)$ gramatice existuje rozkladová tabulka M, která definuje popsaný prediktivní algoritmus realizující rozklad vět $LL(k)$ jazyka. Nyní ukážeme, jakým způsobem lze sestrojit rozkladovou tabulku pro $LL(1)$ gramatiky.

Algoritmus 7.3 Konstrukce rozkladové tabulky pro $LL(1)$ gramatiku

Vstup: $LL(1)$ gramatika $G = (N, \Sigma, P, S)$, jejíž každé pravidlo je opatřeno indexem.

Výstup: Rozkladová tabulka M pro gramatiku G

Metoda: Tabulka M je definována na $(N \cup \Sigma \cup \{\sharp\}) \times (\Sigma \cup \{\epsilon\})$ takto:

(1) Je-li $A \to \alpha$ i-té pravidlo z P, pak $M(A, a) = (\alpha, i)$ pro všechna a z množiny $\text{FIRST}(\alpha), a \neq \epsilon$. Je-li také ϵ prvkem množiny $FOLLOW(A)$.

(2) $M(a,a) = pop$ pro všechna a z množiny Σ

(3) $M(\sharp, \epsilon) = accept$

(4) V ostatních případech $M(X,a) = error$, pro $X \in N \cup \Sigma \cup \{\sharp\}, a \in \Sigma \cup \{\epsilon\}$

Příklad 7.12 Uvažujme $LL(1)$ gramatiku z příkladu 7.10. Nejdříve očíslujeme její pravidla:

(1)	E	$\rightarrow$	TE'
(2)	E'	$\rightarrow$	$+TE'$
(3)	E'	$\rightarrow$	ϵ
(4)	T	$\rightarrow$	FT'
(5)	T'	$\rightarrow$	$*FT'$
(6)	T'	$\rightarrow$	ϵ
(7)	F	$\rightarrow$	(E)
(8)	F	$\rightarrow$	i

Pro ilustraci ukažme, jak podle algoritmu 7.3 vypočítáme první a druhý řádek příslušné rozkladové tabulky M. Celá tabulka je uvedena v tabulce 7.7. Prázdná políčka reprezentují prvky *error*.

	i	(	)	+	*	ϵ
E	$TE', 1$	$TE', 1$				
E'			$\epsilon, 3$	$+TE', 2$		$\epsilon, 3$
T	$FT', 4$	$FT', 4$				
T'			$\epsilon, 6$	$\epsilon, 6$	$*FT', 5$	$\epsilon, 6$
F	$i, 8$	$(E), 7$				
i	pop					
(		pop				
)			pop			
+				pop		
*					pop	
$\sharp$						$accept$

Tabulka 7.7:

Poněvadž FIRST$(TE') = \{(, i\}$, bude $M[E, (] = (TE', 1)$ a $M[E, i] = (TE', 1)$. Všechny ostatní položky prvního řádku jsou *error*. Ke druhému řádku se vztahují přepisovací pravidla (2) a (3). Nejdříve vypočteme FIRST$(+TE') = \{+\}$, takže $M[E', +] = (+TE', 2)$. Poněvadž FIRST$(\epsilon) = \{\epsilon\}$, musíme vypočítat $FOLLOW(E') = \{\epsilon,)\}$. Opět podle bodu (1) algoritmu 7.3 je $M[E', \epsilon] = M[E',)] = (\epsilon, 3)$.

Prediktivní algoritmus používající rozkladové tabulky z tab. 7.7 prochází, např pro vstupní řetězec (i*i), touto posloupností konfigurací:

$$\begin{aligned} [(i*i), E\sharp, \epsilon] &\vdash [(i*i), TE'\sharp, 1] \\ &\vdash [(i*i), FT'E'\sharp, 14] \\ &\vdash [(i*i), (E)T'E'\sharp, 147] \end{aligned}$$

$$
\begin{aligned}
\vdash\ & [i*i), E)T'E'\sharp, 147] \\
\vdash\ & [i*i), TE')T'E'\sharp, 1471] \\
\vdash\ & [i*i), FT'E')T'E'\sharp, 14714] \\
\vdash\ & [i*i), iT'E')T'E'\sharp, 147148] \\
\vdash\ & [*i), T'E')T'E'\sharp, 147148] \\
\vdash\ & [*i), *FT'E')T'E'\sharp, 1471485] \\
\vdash\ & [i), FT'E')T'E'\sharp, 1471485] \\
\vdash\ & [i), iT'E')T'E'\sharp, 14714858] \\
\vdash\ & [), T'E')T'E'\sharp, 14714858] \\
\vdash\ & [), E')T'E'\sharp, 147148586] \\
\vdash\ & [),)T'E'\sharp, 1471485863] \\
\vdash\ & [\epsilon, T'E'\sharp, 1471485863] \\
\vdash\ & [\epsilon, E'\sharp, 14714858636] \\
\vdash\ & [\epsilon, \sharp, 147148586363]
\end{aligned}
$$

Je-li $k > 1$, je konstrukce rozkladové tabulky pro $LL(k)$ gramatiky poněkud složitější. Rozkladová tabulka je v tomto případě definována jako zobrazení z množiny $(\Gamma \cup \{\sharp\}) \times \Sigma^k$, kde Γ obsahuje kromě terminálních a nonterminálních symbolů jména tzv. $LL(k)$ tabulek, které se vytvářejí pro jednotlivá pravidla $LL(k)$ gramatiky. Důležitá je však skutečnost, že složitější je pouze konstrukce rozkladové tabulky a nikoliv vlastní algoritmus rozkladu.

7.5 Problém transformace na $L(1)$ gramatiku

Ve třídě $LL(k)$ gramatik mají pro praktické aplikace zvláštní význam právě $LL(1)$. Tento význam plyne ze dvou skutečností:

(1) Algoritmus pro konstrukci rozkladové tabulky, včetně algoritmů pro výpočet množin FIRST a FOLLOW, je jednodušší pro $k = 1$ než pro $k > 1$.

(2) Rozkladová tabulka (pro reálné programovací jazyky) není pro vyšší hodnoty k prakticky uložitelná v paměti počítače. (Sloupce tabulky jsou označeny nejen řetězci délky k, ale i řetězci délky menší než k, viz tab. 7.5). Z věty 7.6 však víme, že $LL(1)$ jazyky jsou vlastní podtřídou $LL(k)$ jazyků pro $k \leq 2$, tj. neplatí, že pro každou $LL(k)$ gramatiku existuje ekvivalentní $LL(1)$ gramatika.

Přes tyto skutečnosti nacházejí $LL(1)$ gramatiky při implementaci syntaktického analyzátoru programovacích jazyků značné využití. Teoretické překážky, které vyplývají z věty 7.6 se obcházejí už při definici jazyka (jeho syntax je volena tak, aby byla bezprostředně popsatelná $LL(1)$ gramatikou; viz jazyk PASCAL). Dále je pak možné, v případě, když daná gramatika jazyka není $LL(1)$ gramatikou, aplikovat určité transformace, jež mohou vést k $LL(1)$ gramatice. Takové transformace budeme nyní diskutovat.

1 Odstranění levé rekurze

Tato transformace je důležitá pro všechny syntaktické analyzátory shora–dolů včetně analyzátorů, jež pracují s návraty. Jádrem transformace pro odstranění levé rekurze je podle algoritmu 4.6 náhrada A-pravidel

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

pravidly

$$\begin{aligned} A &\to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \mid \beta_1 A' \mid \ldots \mid \beta_m A' \\ A' &\to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \mid \alpha_1 A' \mid \ldots \mid \alpha_n A' \end{aligned}$$

Pro transformaci na $LL(1)$ gramatiku však tato náhrada není vhodná, protože zavedená A-pravidla a A'-pravidla nesplňují podmínku FF z věty 7.4 ($\mathrm{FIRST}(\beta_i) = \mathrm{FIRST}(\beta_i A')$ stejně jako $\mathrm{FIRST}(\alpha_i) = \mathrm{FIRST}(\alpha_i A')$).

Připusťme, na rozdíl od algoritmu 4.6, že výsledná gramatika bez levé rekurze může obsahovat ϵ-pravidla. Pak lze levou rekurzi nonterminálu A odstranit nahrazením uvedených A-pravidel pravidly

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \epsilon, \end{aligned}$$

jež generují právě takové řetězce, jako A a A'-pravidla z algoritmu 4.6.

Poznamenejme, že kdybychom na gramatiku pro aritmetický výraz z příkladu 7.8 aplikovali tento způsob odstranění levé rekurze, získali bychom okamžitě výslednou $LL(1)$ gramatiku.

2 Faktorizace

Tato transformace se uplatňuje u pravidel tvaru

$$A \to \beta\alpha_1 \mid \beta\alpha_2 \mid \ldots \mid \beta\alpha_n,$$

t.j A-pravidel, jejichž pravé strany mají společný prefix $\beta, \beta \in (N \cup \Sigma)^+$. Je zřejmé, že tato vlastnost vede k nesplnění podmínky FF. Faktorizace je postup analogický „vytýkání před závorku“ v aritmetice. Spočívá v nahrazení uvedených A-pravidel pravidly

$$\begin{aligned} A &\to \beta A' \\ A' &\to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \end{aligned}$$

kde A' je nový nonterminál.

Příklad 7.13 Uvažujme gramatiku s pravidly

$$S \to aS \mid a$$

Tato gramatika je $LL(2)$ gramatikou, avšak ne $LL(1)$ gramatikou. Po faktorizaci dostaneme ekvivalentní gramatiku

$$\begin{aligned} S &\to aS' \\ S' &\to S \mid \epsilon \end{aligned}$$

která je $LL(1)$ gramatikou, poněvadž FIRST$(S) = \{a\}$ a FOLLOW$(S') = \{\epsilon\}$.

Faktorizace se často uplatňuje ve spojení s eliminací pravidla gramatiky podle věty 4.7. Jestliže se v gramatice vyskytují pravidla

$$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

která nesplňují podmínku FF věty 7.4, přičemž tato pravidla nelze faktorizovat, pak je možné opakovanou eliminací některých pravidel získat faktorizovatelná A-pravidla.

Příklad 7.14 Uvažujme pravidla

$$\begin{aligned} A &\to aB \mid CD \\ C &\to aE \mid bF \end{aligned}$$

Vidíme, že A-pravidla nesplňují podmínku FF. Eliminujeme-li pravidlo $A \to CD$ dostáváme nová A-pravidla

$$A \to aB \mid aED \mid bFD$$

která po faktorizaci

$$\begin{aligned} A &\to aA' \mid bFD \\ A' &\to B \mid ED \end{aligned}$$

již podmínku FF splňují.

3 Redukce množiny FOLLOW(A)

Tato transformace se může úspěšně uplatnit v případě nesplnění podmínky FFL věty 7.4, tj. v případě že existují A-pravidla $A \to \alpha_1 \mid \alpha_2$ taková, že $\alpha_1 \Rightarrow^* \epsilon$ a FIRST$(\alpha_2) \cap$FOLLOW$(A) \neq \emptyset$. Transformace spočívá v přidání nového nonterminálu gramatiky, který vede ke zmenšení počtu prvků množiny FOLLOW(A) a případně k disjunktnosti množiny FIRST(α_2) a FOLLOW(A). Na druhé straně však tato úprava může zapříčinit změnu ve splnění podmínky FF věty 7.4.

Příklad 7.15 Uvažujme gramatiku s pravidly

$$\begin{aligned} A &\to BaC \\ B &\to abC \mid \epsilon \\ C &\to cBb \mid a \end{aligned}$$

Poněvadž FOLLOW$(B) = \{a, b\}$ a FIRST$(abC) \cap$ FOLLOW$(B) = \{a\}$, tato gramatika není $LL(1)$ gramatikou. Redukujme proto množinu FOLLOW(B) zavedením nového nonterminálu $D = \langle BA \rangle$ a příslušných D-pravidel

$$D \to abCa \mid a$$

V ekvivalentní gramatice

$$\begin{aligned} A &\to DC \\ B &\to abC \mid \epsilon \\ C &\to cBb \mid a \\ D &\to abCa \mid a \end{aligned}$$

již platí $\mathrm{FOLLOW}(B) = \{b\}$ a tudíž $\mathrm{FIRST}(abC) \cap \mathrm{FOLLOW}(B) = \emptyset$. Po faktorizaci D-pravidel dostáváme gramatiku

$$\begin{aligned} A &\to DC \\ B &\to abC \mid \epsilon \\ C &\to cBb \mid a \\ D &\to aD' \\ D' &\to bCa \mid \epsilon \end{aligned}$$

která již je $LL(1)$ gramatikou ($\mathrm{FOLLOW}(D') = \{c, a\}$).

Na závěr znovu zdůrazněme, že uvedené transformace nemusí vést nutně k cíli, a to ani v případě, že pro daný jazyk $LL(1)$ gramatika existuje.

Příklad 7.16 Uvažujme gramatiku G s pravidly

$$\begin{aligned} S &\to A \mid B \\ A &\to a \mid cA \\ B &\to b \mid cB \end{aligned}$$

S-pravidla nesplňují podmínku FF a je tedy třeba nejdříve provést jejich eliminaci:

$$S \to a \mid cA \mid b \mid cB$$

Po faktorizaci těchto nových S-pravidel dostáváme

$$\begin{aligned} S &\to a \mid b \mid cS' \\ S' &\to A \mid B \end{aligned}$$

Vidíme, že jsme dospěli k výchozímu problému (tentokrát u S'-pravidel) a opakování tohoto postupu dále již nemá smysl. Přitom jazyk $L(G) = \{c^n a \mid n \geq 0\} \cup \{c^n b \mid n \geq 0\}$ je $LL(1)$ jazykem a může být generován $LL(1)$ gramatikou s pravidly

$$\begin{aligned} S &\to a \mid b \mid BA \\ A &\to a \mid b \\ B &\to cC \\ C &\to \epsilon \mid cC \end{aligned}$$

7.6 Cvičení

Cvičení 7.6.1 Gramatika G s pravidly

(1) $E \to TE'$

(2) $E' \to +E$

(3) $E' \to \epsilon$

(4) $T \rightarrow FT'$

(5) $T' \rightarrow T$

(6) $T' \rightarrow \epsilon$

(7) $F \rightarrow PF'$

(8) $F' \rightarrow *F'$

(9) $F' \rightarrow \epsilon$

(10) $P \rightarrow (E)$

(11) $P \rightarrow a$

(12) $P \rightarrow b$

(13) $P \rightarrow \varepsilon$

popisuje regulární výrazy nad abecedou $\{a, b\}$; ε značí prázdný symbol výrazu.

a) Pro každý nonterminál této gramatiky vypočítejte množiny FIRST a FOLLOW

b) Ukažte, že G je $LL(1)$ gramatika

c) Vytvořte rozkladovou tabulku pro G

d) Realizujte rozklad věty $\varepsilon + ab^* + a^*b$

Cvičení 7.6.2 Ukažte, že každý regulární jazyk je $LL(1)$ jazykem. Dále ukažte, že jazyk generovaný $LL(0)$ gramatikou má nanejvýš jednu větu.

Cvičení 7.6.3 Ukažte, že gramatika s pravidly $S \rightarrow aaSbb \mid a \mid \epsilon$ je $LL(2)$ gramatikou. Nalezněte ekvivalentní $LL(1)$ gramatiku.

Cvičení 7.6.4 Sestrojte $LL(1)$ gramatiku jazyka PL0, jehož grafy syntaxe jsou uvedeny v příloze. Na základě této gramatiky sestrojte rozkladovou tabulku.

Kapitola 8

LR jazyky a jejich syntaktická analýza

$LR(k)$[1] gramatiky reprezentují největší třídu gramatik, ke kterým lze vždy sestrojit deterministický syntaktický analyzátor *zdola–nahoru*. Důležitost LR jazyků plyne z těchto dvou skutečností:

(1) Pro všechny programovací jazyky, které lze popsat bezkontextovou gramatikou, můžeme prakticky sestrojit LR syntaktické analyzátory.

(2) Metody syntaktické analýzy LR jazyků jsou obecnější, než metody analýzy jednoduchých nebo operátorových precedenčních jazyků nebo metody deterministické syntaktické analýzy shora–dolů a přitom vedou ke stejně efektivním analyzátorům.

Zásadní nevýhodou syntaktické analýzy $LR(k)$ jazyků je přílišná pracnost implementace. Při aplikacích na typické programovací jazyky je prakticky nemožné konstruovat LR analyzátor ručně, bez použití konstruktoru LR analyzátoru. Tímto problémem se budeme zabývat v odst. 8.2 a 8.3.

8.1 Definice $LR(k)$ gramatiky

Předpokládejme, že $G = (N, \Sigma, P, S)$ je jednoznačná gramatika, a že $w = a_1 a_2 \dots a_n$ je větou z $L(G)$. Pak existuje jednoznačná posloupnost větných forem $\delta_0, \delta_1, \dots, \delta_m$, které tvoří pravou derivaci věty w:

$$\begin{array}{rcll} \delta_0 & = & S & \\ \delta_i & \Rightarrow^{p_i} & \delta_{i+1}, & i = 0, 1, \dots, m-1 \\ \delta_m & = & w & \end{array}$$

p_i značí index přepisovacího pravidla, kterého jsme použili pro expanzi *nejpravějšího* neterminálu větné formy δ_i. Řetězec indexů $p_{m-1} p_{m-2} \dots p_1 p_0$ reprezentuje pravý rozklad věty w.

[1] Název LR je odvozen z „*Left* to right“ a „*Right* parse“.

Neformálně řečeno, jestliže můžeme jednoznačně určit l-frázi každé větné formy δ_i pro $i = m, m-1, \ldots, 1$ a nonterminál, k němuž bude l-fráze redukována tak, že prohlížíme symboly větné formy δ_i zleva doprava, přičemž nám „stačí“ nanejvýš k symbolů za koncem l-fráze, pak gramatiku G nazýváme $LR(k)$ gramatikou. Je-li tedy

$$\begin{aligned} \delta_{i-1} &= \beta A z, \\ \delta_i &= \beta \alpha z \end{aligned}$$

(tj. α je l-frází větné formy δ_i) pak na základě nejvýše k symbolů řetězce z (tj. k příštích vstupních symbolů) dovedeme jednoznačně určit l-frázi α a nonterminál A, ke kterému se redukuje (podle pravidla $A \to \alpha$). V případě $LR(k)$ gramatiky je tak jednoznačně určena relace $\delta_{i-1} \Rightarrow_{rm} \delta_i$; $\Rightarrow_{rm}$ značí pravou derivaci.

Aby v průběhu syntaktické analýzy $LR(k)$ jazyků nevznikaly komplikace a výskytem výchozího symbolu na pravé straně pravidla, definujeme tzv. rozšířenou gramatiku (augmented grammar).

Definice 8.1 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Gramatika $G' = (N \cup \{S'\}, P \cup \{S' \to S\}, S')$ se nazývá *rozšířenou gramatikou* gramatiky G.

Rozšířená gramatika G' je zřejmě ekvivalentní s gramatikou G; přitom zaručuje, že pro $\delta_i = S'$ proces syntaktické analýzy zdola–nahoru končí přijetím vstupního řetězce. Pravidlu $S' \to S$ obvykle přiřadíme index 0.

Definice 8.2 Nechť $G = (N, \Sigma, P, S)$ je gramatika a G' její rozšířená gramatika. Jestliže pro každé dvě pravé derivace v gramatice G'

$$\begin{aligned} S' &\Rightarrow^* \beta A z \Rightarrow \beta \alpha z \\ S' &\Rightarrow^* \gamma B v \Rightarrow \beta \alpha u \end{aligned}$$

pro které $\mathrm{FIRST}_k(z) = \mathrm{FIRST}_k(u)$ platí $\beta A z = \gamma B v$ (tj. $\beta = \gamma, A = B$ a $z = v$), pak G se nazývá $LR(k)$ gramatikou, $k \geq 0$.

Příklad 8.1 Uvažujme pravou lineární gramatiku G s pravidly

$$\begin{aligned} S &\to C \mid D \\ C &\to aC \mid b \\ D &\to aD \mid c \end{aligned}$$

Ukážeme, že G je $LR(1)$ gramatika. Každá (pravá) derivace v rozšířené gramatice gramatiky G má tvar

$$S' \Rightarrow S \Rightarrow C \overset{i}{\Rightarrow} a^i C \Rightarrow a^i b$$

nebo

$$S' \Rightarrow S \Rightarrow D \overset{i}{\Rightarrow} a^i D \Rightarrow a^i c$$

Ve shodě s definicí 8.2 uvažujme derivace tvaru

$$\begin{aligned} S' &\Rightarrow^* \beta A z \Rightarrow \beta \alpha z \\ S' &\Rightarrow^* \gamma B v \Rightarrow \beta \alpha u \end{aligned}$$

Poněvadž G je pravá lineární gramatika, musí platit $z = v = \epsilon$. Je-li (podle definice) $\mathrm{FIRST}(z) = \mathrm{FIRST}(u)$, pak je ale také $u = \epsilon$. Musíme proto ukázat, že $\beta A = \gamma B$, tj. $\beta = \gamma$ a $A = B$.

Předpokládejme, že v derivaci $\gamma B v \Rightarrow \beta \alpha u$ jsme použili pravidla $B \to \delta$. Existují tři situace, které musíme uvažovat:

(1) $A = S'$, tj. derivace $S' \Rightarrow^* \beta Az$ je triviální.

Pak je ovšem $\beta = \epsilon$ a $\alpha = S$. Protože existuje pouze jedna cesta, jak získat větnou formu S, je $\gamma = \epsilon$ a $B = S'$ a tedy $\beta = \gamma$ a $A = B$.

(2) $A = C$

Pak α je buď aC, nebo b. V prvním případě $B = C$, protože pouze nonterminály C a S mají pravidla, jež končí na C. Kdyby platilo $B = S$, pak by z tvaru derivace v G muselo být $\gamma = \epsilon$. Protože $\gamma B \neq \beta\alpha$, platí tedy $B = C, \delta = aC$ a $\gamma = \beta$. V druhém případě ($\alpha = b$) platí $B = C$ také, poněvadž pouze nonterminál C má pravidlo, jež končí symbolem b. Opět dostáváme $B = A$ a $\gamma = \beta$.

(3) $A = D$

Tento případ je symetrický s případem (2).

Poznamenejme, že G je ve skutečnosti $LR(0)$ gramatika a že G není $LL(1)$ gramatika.

Příklad 8.2 Gramatika s pravidly

$$\begin{aligned} S &\rightarrow Ab \mid Bc \\ A &\rightarrow Aa \mid \epsilon \\ B &\rightarrow Ba \mid \epsilon \end{aligned}$$

je ekvivalentní s gramatikou z předchozího příkladu. Uvažujme dvě pravé derivace v její rozšířené gramatice:

$$\begin{aligned} S' \Rightarrow S &\Rightarrow^* Aa^k b \Rightarrow a^k b \\ S' \Rightarrow S &\Rightarrow^* Ba^k c \Rightarrow a^k c \end{aligned}$$

Tyto dvě derivace splňují předpoklad definice 8.2 pro $\beta = \epsilon, \alpha = \epsilon, z = a^k b, \gamma = \epsilon, v = a^k c$. Poněvadž $A \neq B$, pro libovolné k, není uvažovaná gramatika LR gramatikou.

Příklad 8.3 Uvažujme gramatiku G s pravidly

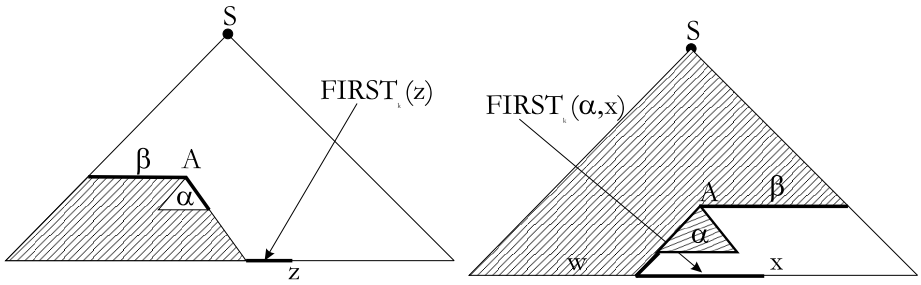
$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow CD \mid aE \\ C &\rightarrow ab \\ D &\rightarrow bb \\ E &\rightarrow bba \end{aligned}$$

a dvě pravé derivace v její rozšířené gramatice:

$$\begin{aligned} S' \Rightarrow S \Rightarrow AB &\Rightarrow ACD \Rightarrow ACbb \Rightarrow Aabbb \\ S' \Rightarrow S \Rightarrow AB &\Rightarrow AaE \Rightarrow Aabba \end{aligned}$$

Ve větné formě $Aabz$ nemůžeme určit, zda se pravý konec l-fráze nachází mezi b a z (když $z = bb$), nebo hned napravo od A (když $z = ba$), pouze na základě prvního symbolu řetězce z. G tudíž není $LR(1)$ gramatikou. Dá se však ukázat, že G je $LR(2)$ gramatikou.

Na obrázku 8.1 je názorně ukázána situace při konstrukci derivačního stromu v případě $LR(k)$ a $LL(k)$ gramatiky. Vyšrafovaná oblast znázorňuje již sestrojenou část derivačního stromu.



Obrázek 8.1: Syntaktická analýza věty vlevo: $LR(k)$, vpravo: $LL(k)$ jazyka

Následující věta shrnuje důležité poznatky o LR gramatikách a jazycích. Uvádíme ji bez důkazu.

Věta 8.1

(1) Každá $LR(k)$ gramatika je jednoznačná.

(2) Každý deterministický jazyk je generovatelný nějakou $LR(k)$ gramatikou.

(3) Každý $LR(k)$ jazyk má $LR(1)$ gramatiku.

Vidíme, že tvrzení (2) a (3) se zcela liší od obdobných poznatků ve třídě $LL(k)$ gramatik a jazyků. Na obr. 8.2 je znázorněna celá hierarchie tříd deterministických bezkontextových jazyků, s nimiž jsme se seznámili.

Ze skutečnosti, že každý deterministický jazyk lze popsat $LR(1)$ gramatikou, vyplývá, že $LR(1)$ gramatiky mají pro praktické aplikace mimořádný význam. Proto se v další části této kapitoly omezíme pouze na $LR(1)$ gramatiky a jejich podtřídy – jednoduché $LR(1)$ gramatiky a $LALR(1)$ gramatiky, jež vedou k jednodušší konstrukci syntaktického analyzátoru.

8.2 Syntaktická analýza LR-jazyků

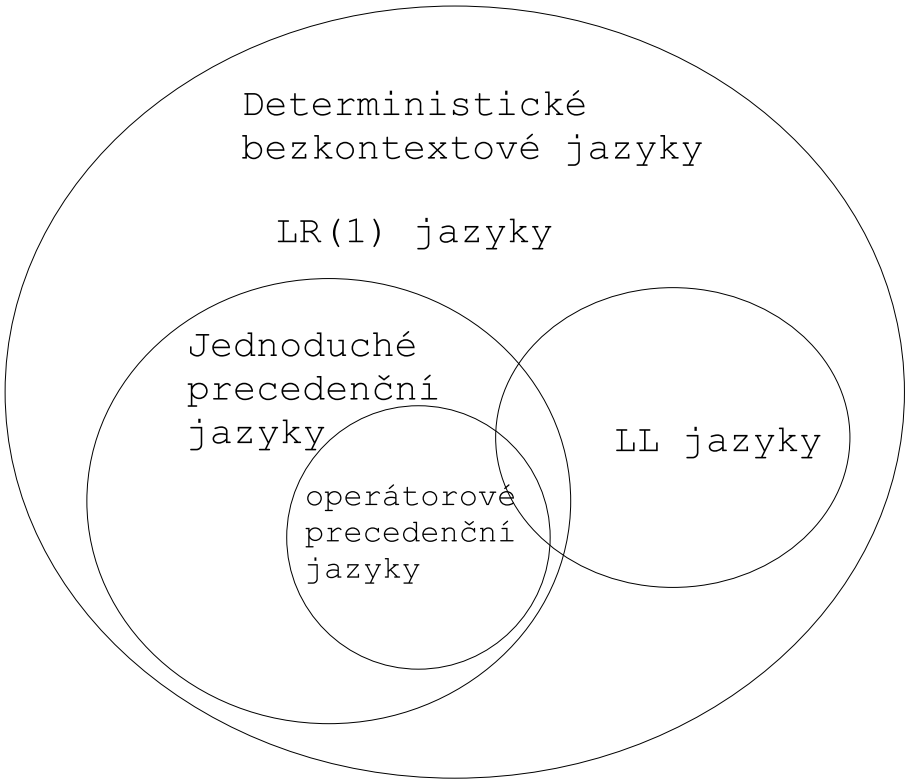
LR analyzátor se skládá, podobně jako LL analyzátor, ze dvou částí: *řídící části* a *rozkladové tabulky* (8.3). Řídící část analyzátoru zůstává stejná pro různé LR analyzátory; mění se pouze rozkladová tabulka. Z tohoto důvodu se problém automatické konstrukce LR analyzátoru prakticky redukuje na implementaci algoritmu, jež pro danou LR gramatiku vytváří rozkladovou tabulku. Funkci konstruktoru LR analyzátoru ilustruje obr. 8.4.

8.2.1 Algoritmus syntaktické analýzy

Syntaktický analyzátor LR jazyků představuje, podobně jako analyzátor jazyků precedenčních, realizaci rozšířeného zásobníkového automatu. Jeho struktura je uvedena na obr. 8.5.

Syntaktický analyzátor operuje nad vstupní páskou, jež obsahuje vstupní řetězec a je čtena zleva doprava, zásobníkem a rozkladovou tabulkou, která zobrazuje funkce, jež jsou realizovány řídící částí. Vstupní řetězec je ukončen vždy symbolem ♯. Výstupní páska obsahuje, v případě úspěšně ukončené analýzy, pravý rozklad vstupní věty. Zásobník obsahuje řetězce tvaru

$$s_0 X_1 s_1 X_2 s_2 \ldots X_m s_m,$$



Obrázek 8.2: Hierarchie deterministických bezkontextových jazyků

Obrázek 8.3: Schéma LR analyzátoru

s_m je vrchol zásobníku. Každý symbol X_i reprezentuje terminální nebo nonterminální symbol gramatiky. Zásobníkové symboly s_i budeme nazývat *stavy*. Stav s_m na vrcholu zásobníku sumarizuje informaci o celém obsahu zásobníku a slouží k rozhodnutí, zda se má provést redukce, nebo přesun vstupního symbolu do zásobníku.

Jádrem analyzátoru je rozkladová tabulka, která reprezentuje dvě dvouargumentové funkce, funkci akcí a funkci přechodů. Označme S_z množinu stavů (zásobníková abeceda $\Gamma = S_z \cup N \cup \Sigma$).

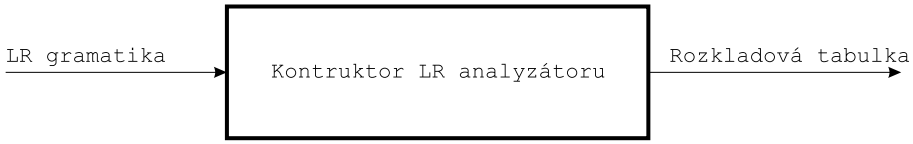
Funkce akcí je definována jako zobrazení

$$\text{AKCE: } S_z \times (\Sigma \cup \{\sharp\}) \to \{shift\ s, reduce\ i, error, accept\}.$$

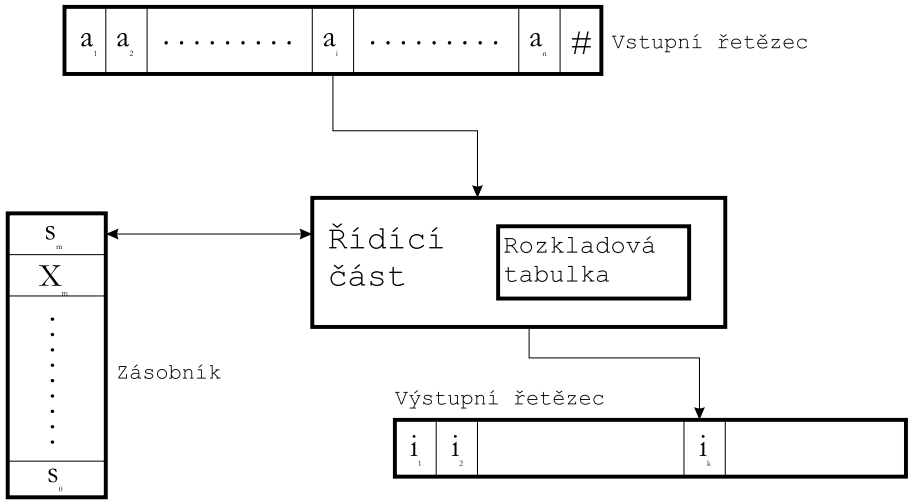
Funkce přechodů je definována jako zobrazení

$$\text{PŘECHOD: } S_z \times (\Sigma \cup N) \to S_z$$

a je ve skutečnosti, jak uvidíme později, přechodovou funkcí deterministického konečného automatu.



Obrázek 8.4: Konstruktor LR analyzátoru



Obrázek 8.5: Struktura LR syntaktického analyzátoru

Konfigurací LR analyzátoru budeme rozumět trojici řetězců

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \sharp, i_1 i_2 \dots i_k)$$

z nichž první představuje obsah zásobníku, druhý dosud nezpracovanou část vstupního řetězce a třetí, tvořící se pravý rozklad vstupního řetězce.

Příští činnost analyzátoru je určena na základě čteného vstupního symbolu a_i, vrchol zásobníku s_m a hodnoty funkce akcí AKCE$[s_m, a_i]$:

1. Je-li AKCE$[s_m, a_i]$ = *shift* s, pak analyzátor provede přesun do zásobníku a přejde do konfigurace

   $$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i s, a_{i+1} \dots a_n \sharp, i_1 i_2 \dots i_k)$$

   To tedy znamená, že do zásobníku se přesunul přečtený vstupní symbol a_i a stav $a =$ PŘECHOD$[s_m, a_i]$. Příštím vstupním symbolem se stává a_{i+1}.

2. Je-li AKCE$[s_m, a_i]$ = *reduce* i, pak analyzátor provede redukci podle i-tého pravidla gramatiky $A \to \alpha$.

   Výsledná konfigurace má tvar

   $$(s_0 X_1 s_1 X_2 s_2 \dots X_{m-r} s_{m-r} A s, a_i a_{i+1} \dots a_n \sharp, i_1 i_2 \dots i_k i)$$

   kde $s =$ PŘECHOD$[s_{m-r}, A]$, r je délka pravé strany α i-tého pravidla gramatiky. Ze zásobníku bylo nejdříve odstraněno $2r$ symbolů (r symbolů gramatiky a r stavových symbolů),

takže na vrcholu zásobníku se objevil symbol s_{m-r}. Pak analyzátor uložil na vrchol zásobníku nonterminál A a stav $s = \text{PŘECHOD}[s_{m-r}, A]$ a do výstupního řetězce přidal index i přepisovacího pravidla, podle něhož proběhla redukce. Vstupní symbol zůstává nezměněn. Poznamenejme, že hodnota $\text{AKCE}[s_m, a_i] = \textit{reduce } i$ vlastně říká, že $X_{m-r+1} \dots X_m = \alpha$ je l-fráze, a že má být redukována k nonterminálu A.

3. Je-li $\text{AKCE}[s_m, \sharp] = \textit{accept}$, pak rozklad věty byl úspěšně dokončen. Řetězec $i_1 i_2 \dots i_k$ tvoří pravý rozklad věty.

4. Je-li $\text{AKCE}[s_m, a_i] = \textit{error}$, pak byla objevena syntaktická chyba. Vstupní symbol a_i lze chápat jako její velmi blízkou lokalizaci.

Celou činnost analyzátoru lze nyní popsat velmi jednoduše. Na počátku je analyzátor v počáteční konfiguraci

$$(s_0, a_1 a_2 \dots a_n \sharp, \epsilon)$$

kde s_0 je vyznačený (počáteční) stavový symbol a $a_1 \dots a_n$ je vstupní řetězec.

Podle popsaného postupu se pak provádí přechody mezi konfiguracemi tak dlouho, pokud nenastane případ $\text{AKCE}[s_m, a_i] = \textit{accept}$, nebo $\text{AKCE} = \textit{error}$.

Příklad 8.4 Tab. 8.1 je rozkladová tabulka pro gramatiku

$$\begin{array}{lrcl}
(1) & E & \to & E + T \\
(2) & E & \to & T \\
(3) & T & \to & T * F \\
(4) & T & \to & T \\
(5) & F & \to & (E) \\
(6) & F & \to & i
\end{array}$$

Hodnoty funkce akcí jsou kódovány tímto způsobem:

$$\begin{array}{rcl}
s_i & - & \textit{shift } i \\
r_j & - & \textit{reduce } j \\
\text{acc} & - & \textit{accept} \\
 & - & \textit{error}
\end{array}$$

Hodnota funkce přechodů $\text{PŘECHOD}[s_m, a]$ pro $a \in \Sigma \cup \{\sharp\}$ je v tab. 8.1 zobrazena v části AKCE společně s kódem pro přesun do zásobníku (společně s akcí *shift*). V části PŘECHOD jsou pak zobrazeny hodnoty funkce přechodů pro nonterminály; uplatňují se v případě redukce.

S uvedenými konvencemi kódování funkcí akcí a přechodů ilustrujme činnost analyzátoru pro vstupní řetězec $i * i + 1$. Odpovídající posloupnost konfigurací je uvedena v tab. 8.2.

V řádku 1 je analyzátor v počáteční konfiguraci. Poněvadž podle rozkladové tabulky (tab. 8.1) je $\text{AKCE}[0, i] = s_5$, přesune se vstupní symbol a stavový symbol 5 do zásobníku (viz řádek 2). Nyní je pod čtecí hlavou symbol $*$ a protože $\text{AKCE}[5, *] = r_6$, provede se redukce podle 6-tého pravidla $F \to i$ takto:

(1) odstraní se 2 symboly ze zásobníku, takže na vrcholu se objeví stav 0,

Stav	Akce						Přechod		
	$\mathbf{a_i}$								
$\mathbf{s_m}$	**i**	+	*	(	)	♯	**E**	**T**	**F**
0	s_5			s_4			1	2	3
1		s_6				acc			
2		r_2	s_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	s_5			s_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	s_5			s_4				9	3
7	s_5			s_4					10
8		s_6			s_{11}				
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Tabulka 8.1: Rozkladová tabulka

(2) poněvadž PŘECHOD$[0, F] = 3$, do zásobníku se uloží symboly F a 3 a na výstup index pravidla $F \to i$, tj. 6 (viz řádek 3).

Podobným způsobem se pokračuje až do okamžiku, kdy AKCE$[1, \sharp]$ = acc (řádek 14), kdy činnost analyzátoru končí. Na výstupu je řetězec 64632641 reprezentující pravý rozklad věty $i * i + 1$.

8.3 Konstrukce rozkladové tabulky

Jádrem implementace LR analyzátoru je zřejmě konstrukce rozkladové tabulky. Existuje několik metod vytváření rozkladové tabulky, které se liší jak v pracnosti jejich konstrukce, požadavcích na paměť, tak také v mocnosti metod jednoznačně definovat funkci akcí pro danou gramatiku.

V této podkapitole se seznámíme se třemi metodami vytváření rozkladové tabulky:

- pro jednoduché $LR(1)$ gramatiky
- pro $LR(1)$ gramatiky
- pro $LALR(1)$ gramatiky

První případ představuje nejméně pracnou, avšak také nejomezenější cestu k získání rozkladové tabulky. $LR(1)$ gramatiky jsou velmi univerzální; rozkladová tabulka pro $LR(1)$ gramatiku je však náročná na paměť a její konstrukce je velmi pracná. Rozkladová tabulka pro $LALR(1)$ gramatiku představuje kompromis mezi jednoduchými $LR(1)$ a $LR(1)$ gramatikami. Obě podtřídy $LR(1)$ gramatik vymezíme prostřednictvím algoritmu vytváření jejich rozkladové tabulky; nebudeme uvádět její formální definice. Lze je nalézt např. v [**?**].

	Obsah zásobníku	Vstup	Výstup
1	O	$i*i+i\sharp$	
2	$Oi5$	$*i+i\sharp$	
3	$OF3$	$*i+i\sharp$	6
4	$OT2$	$*i+i\sharp$	64
5	$OT2*7$	$i+i\sharp$	64
6	$OT2*7i5$	$+i\sharp$	64
7	$OT2*7F10$	$+i\sharp$	646
8	$OT2$	$+i\sharp$	6463
9	$OE1$	$+i\sharp$	64632
10	$OE1+6$	$i\sharp$	64632
11	$OE1+6i5$	$\sharp$	64632
12	$OE1+6F3$	$\sharp$	646326
13	$OE1+6T9$	$\sharp$	6463264
14	$OE1$	$\sharp$	64632641

Tabulka 8.2: Činnost LR analyzátoru

8.3.1 Kanonický soubor $LR(0)$ položek, konstrukce SLR rozkladové tabulky

Definice 8.3 Nechť $G=(N,\Sigma,P,S)$ je gramatika. Řetězec $\gamma\in(N\cup\Sigma)^*$ se nazývá *perspektivním prefixem* (viable prefix) v gramatice G, jestliže existuje pravá derivace $S\Rightarrow^*\beta Az\Rightarrow\beta\alpha z$ taková, že γ je prefixem řetězce $\beta\alpha$.

Perspektivní prefix je tedy takovým prefixem pravé větné formy, který neobsahuje žádný symbol za pravým koncem l-fráze této větné formy. Označení „perspektivní“ plyne ze skutečnosti, že je možné vždy za perspektivní prefix přidat řetězec terminálních symbolů a dostat tak pravou větnou formu v G. Je-li

$$(s_0X_1s_1\ldots X_ms_m,a_ia_{i+1}\ldots a_n,i_1\ldots i_k)$$

konfigurace analyzátoru taková, že $\mathrm{AKCE}[s_m,a_i]\neq error$, pak právě $X_1X_2\ldots X_m$ představuje perspektivní prefix v gramatice G. V této podkapitole ukážeme konstrukci deterministického konečného automatu, který je schopen rozpoznávat perspektivní prefixy. To, že lze takový *konečný* automat sestrojit, patří k pozoruhodným výsledkům teorie LR-jazyků. Zřejmě je tedy množina perspektivních prefixů gramatiky G *kvalitativně* odlišná od jazyka generovaného gramatikou G.

Definice 8.4 Nechť $G=(N,\Sigma,P,S)$ je gramatika. Říkáme, že $[A\to\alpha_1\cdot\alpha_2,u],\alpha_1,\alpha_2\in(N\cup\Sigma)^*,u\in\Sigma^{*k}$ je $LR(k)$ *položka* pro gramatiku G a dané k, jestliže $A\to\alpha_1\alpha_2$ je pravidlo z P. Říkáme, že $LR(k)$ položka $[A\to\alpha_1\cdot\alpha_2,u]$ je *platná* pro řetězec $\beta\alpha_1$, jestliže existuje v G pravá derivace

$$S\stackrel{*}{\Rightarrow}\beta Az\Rightarrow\beta\alpha_1\alpha_2z$$

taková, že $u=\mathrm{FIRST}_k(z)$. (Řetězec $\beta\alpha_1$ je tedy perspektivním prefixem).

Konvence 8.1 V případě, že to nebude na újmu srozumitelnosti, budeme, namísto $LR(k)$ položka pro G a dané k, říkat krátce položka a v zápisu položky vynechávat hranaté závorky. Je-li $k=0$, pak je v položce druhá složka u vždy rovna ϵ a budeme ji také vynechávat.

Příklad 8.5 Například pravidlo $A \to XYZ$ generuje čtyři $LR(0)$ položky:

$$\begin{aligned} A &\to \cdot XYZ \\ A &\to X \cdot YZ \\ A &\to XY \cdot Z \\ A &\to XYZ\cdot \end{aligned}$$

Pravidlo $A \to \epsilon$ generuje jedinou položku

$$A \to \cdot$$

Intuitivně, každá položka indikuje pozicí tečky, jakou část l-fráze „vidíme“ v procesu rozkladu před provedením redukce podle daného pravidla. V příkladě 8.5 první položka říká, že očekáváme řetězec derivovatelný z XYZ, druhá říká, že na vstupu vidíme řetězec derivovatelný z X a očekáváme řetězec derivovatelný z YZ.

Jednotlivé položky můžeme interpretovat jako stavy nedeterministického konečného automatu, který rozpoznává perspektivní prefixy v gramatice G

Jednotlivé položky můžeme interpretovat jako stavy nedeterministického konečného automatu, který rozpoznává perspektivní prefixy v gramatice G. Sdružením položek do určitých množin můžeme dospět ke stavům LR analyzátoru, které, jak již jsme uvedli, přísluší deterministickému automatu. Tento automat, jak uvidíme, definuje funkci přechodu rozkladové tabulky.

Soubor množin $LR(0)$ položek, nazývaný *kanonický* $LR(0)$ *soubor*, tvoří základ konstrukce rozkladové tabulky pro třídu LR gramatik – *jednoduchých* LR *gramatik*, krátce SLR gramatik (Simple LR). Ukážeme nyní způsob vytváření tohoto kanonického $LR(0)$ souboru.

Nejdříve je třeba definovat dvě funkce, které nazveme UZÁVĚR a GOTO.

Definice 8.5 Nechť I je množina $LR(0)$ položek gramatiky G. Množina položek UZÁVĚR(I) je definována těmito pravidly:

(1) Každá položka z I je v množině UZÁVĚR(I).

(2) Je-li $[A \to \alpha_1 \cdot B\alpha_2]$ v množině UZÁVĚR(I) a $\beta \to \gamma$ je pravidlo gramatiky, pak $[B \to \cdot\gamma]$ je také v množině UZÁVĚR(I).

Intuitivně, je-li $[A \to \alpha_1 \cdot B\alpha_2] \in$ UZÁVĚR(I), pak to znamená, že v jistém okamžiku rozkladu očekáváme na vstupu řetězec derivovatelný z $B\alpha_2$. Existence pravidla $B \to \gamma$ pak implikuje, že v tomto okamžiku můžeme případně očekávat řetězec derivovatelný z γ, a proto je v množině UZÁVĚR(I) také položka $[B \to \cdot\gamma]$.

Příklad 8.6 Uvažujme rozšířenou gramatiku

$$\begin{aligned} E' &\to E \\ E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

Jestliže I je množina s jedinou položkou $\{[E' \to \cdot E]\}$, pak UZÁVĚR(I) obsahuje položky

$$\begin{aligned} E' &\to \cdot E \\ E &\to \cdot E + T \\ E &\to \cdot T \\ T &\to \cdot T * F \\ T &\to \cdot F \\ F &\to \cdot (E) \\ F &\to \cdot i \end{aligned}$$

Položka $[E' \to \cdot E]$ je v množině UZÁVĚR(I) na základě pravidla (1). Protože tečka leží bezprostředně před nonterminálem E, pak na základě pravidla (2) definice 8.5 a E-pravidel gramatiky, jsou v množině UZÁVĚR(I) obsaženy položky $[E \to \cdot E + T]$ a $[E \to \cdot T]$. Nyní, protože $[E \to \cdot T] \in$ UZÁVĚR(I) platí (podle pravidla (2)) $[T \to \cdot T * F] \in$ UZÁVĚR(I) a $[T \to \cdot F] \in$ UZÁVĚR(I) plyne $[F \to \cdot (E)] \in$ UZÁVĚR(I). Jiné položky množina UZÁVĚR$\{(E' \to \cdot E)\}$ neobsahuje.

Definice 8.6 Nechť I je množina $LR(0)$ položek a X je symbol gramatiky, $X \in (N \cup \Sigma)$. Funkce GOTO(I, X) je definována jako množina UZÁVĚR($\{[A \to \alpha_1 X \cdot \alpha_2] \mid [A \to \alpha_1 \cdot X\alpha_2] \in I\}$), tj. uzávěr všech položek $[A \to \alpha_1 X \cdot \alpha_2]$ takových, že $[A \to \alpha \cdot X\alpha_2]$ leží v I.

Intuitivně, je-li I množina položek, které jsou platné pro perspektivní prefix γ, pak GOTO(I, X) je množina položek, jež jsou platné pro perspektivní prefix γX (viz definice 8.4).

Příklad 8.7 Je-li I množina položek $\{[E' \to E\cdot], [E \to E \cdot +T]\}$, pak GOTO($I, +$) obsahuje právě položky:

$$\begin{aligned} E &\to E + \cdot T \\ T &\to \cdot T * F \\ T &\to \cdot F \\ F &\to \cdot (E) \\ F &\to \cdot i \end{aligned}$$

Protože první položka množiny I neobsahuje symbol $+$, nemá vliv na hodnotu funkce GOTO($I, +$). Druhá položka neobsahuje symbol $+$ právě za tečkou. Proto posuneme tečku za $+$, dostaneme položku $[E \to E + \cdot T]$ a spočítáme UZÁVĚR($\{E \to E + \cdot T\}$).

Nyní již můžeme formulovat algoritmus pro výpočet kanonického souboru množin $LR(0)$ položek.

Algoritmus 8.1 Výpočet kanonického souboru množin $LR(0)$ položek.

Vstup: Rozšířená gramatika ke gramatice $G = (N, \Sigma, P, S)$

Výstup: Kanonický soubor C množin $LR(0)$ položek

Metoda:

(1)

$$\begin{aligned} I_0 &:= \text{UZÁVĚR}(\{S \to \cdot S'\}) \\ C &:= \{I_0\} \\ r &:= 0 \end{aligned}$$

(2) pro všechna $I_k \in C, k = 0, 1, \ldots, r$ a pro všechna $X \in (N \cup \Sigma)$ proveď: je-li $\mathrm{GOTO}(I_k, X) \neq \emptyset \wedge \mathrm{GOTO}(I_k, X) \notin C$ pak
   (a) $r := r + 1$
   (b) $I_r := \mathrm{GOTO}(I_k, X)$
   (c) $C := C \cup \{I_r\}$

(3) opakuj krok (2) tak dlouho, pokud lze k souboru C přidávat novou množinu I_r.

Algoritmus 8.1 zřejmě končí, poněvadž při výpočtu funkce GOTO se posouvá tečka vždy o jeden symbol doprava.

Příklad 8.8 Uvažujme gramatiku z příkladu 8.6. Její kanonický soubor C $LR(0)$ položek má tvar:

$$C = \{I_0, I_1, \ldots, I_{11}\}$$

kde množiny položek I_k obsahují položky:

$I_0 = \mathrm{UZÁVĚR}(\{E' \to \cdot E\})$:
$E' \to \cdot E$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

$I_1 = \mathrm{GOTO}(I_0, E)$:
$E' \to E\cdot$
$E \to E \cdot + T$

$I_2 = \mathrm{GOTO}(I_0, T)$:
$E \to T\cdot$
$E \to T \cdot * F$

$I_3 = \mathrm{GOTO}(I_0, F)$:
$T \to F\cdot$

$I_4 = \mathrm{GOTO}(I_0, ()$:
$F \to (\cdot E)$
$E \to \cdot E + T$
$E \to \cdot T$
$F \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

$I_5 = \mathrm{GOTO}(I_0, i)$:
$F \to i\cdot$

$I_6 = \mathrm{GOTO}(I_1, +)$:
$E \to E + \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

$I_7 = \mathrm{GOTO}(I_2, *)$:
$T \to T * \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

$I_8 = \mathrm{GOTO}(I_4, E)$:
$F \to (E\cdot)$
$E \to E \cdot + T$

$I_9 = \mathrm{GOTO}(I_6, T)$:
$E \to E + T\cdot$
$T \to T \cdot * F$

$I_{10} = \mathrm{GOTO}(I_7, F)$:
$T \to T * F\cdot$

$I_{11} = \mathrm{GOTO}(I_8,))$:
$F \to (E)\cdot$

Funkce GOTO představuje funkci deterministického konečného automatu

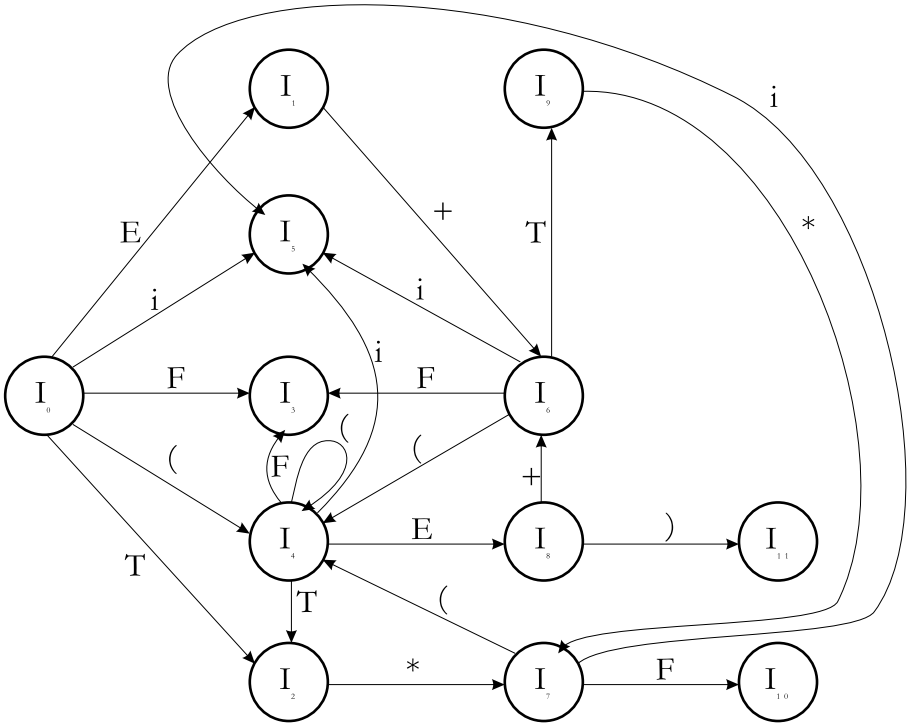
$$D = (N \cup \Sigma, C, I_0, \delta, C), \text{ kde } \delta(I, X) = \mathrm{GOTO}(I, X),$$

jehož vstupní abecedou je množina terminálních a nonterminálních symbolů, stavy jsou reprezentovány množinami I_k, I_0 je počáteční stav a každý stav může být koncovým stavem. Automat D rozpoznává množinu perspektivních prefixů v gramatice G. Tento překvapující fakt (automat D je deterministický pro každou gramatiku) plyne z konstrukce množin I_k podle algoritmu 8.1. Ekvivalentní nedeterministický automat je definován tak, že jeho stavy jsou jednotlivé $LR(0)$ položky a přechodová funkce δ je konstruována takto:

$\delta([A \to \alpha \cdot X\beta], x)$ obsahuje $[A \to \alpha X \cdot \beta]$

$\delta([A \to \alpha \cdot B\beta], \epsilon)$ obsahuje $[B \to \cdot\gamma], B \to \gamma$ je pravidlo v G.

Na obr. 8.6 je diagram přechodů konečného deterministického automatu D pro kanonický soubor množin $LR(0)$ položek z příkladu 8.8. V tomto diagrama jsou zobrazeny samozřejmě i ty hodnoty funkce GOTO, které jsme při konstrukci kanonického souboru explicitně nevypisovali, např. $\mathrm{GOTO}(I_4, () = I_4$.



Obrázek 8.6: Diagram přechodů automatu D

Jeden z hlavních teorému teorie rozkladu LR jazyků říká, že množinou platných položek pro perspektivní prefix γ je právě množina položek dosažitelná z počátečního stavu automatu D po cestě, jež odpovídá řetězci γ. Ukažme platnost tohoto tvrzení na předchozím příkladě.

Příklad 8.9 Uvažujme gramatiku G z příkladu 8.6 a její kanonický soubor množin $LR(0)$ položek z příkladu 8.8. Řetězec $E + T*$ je perspektivní prefix v této gramatice. Automat na obr. 8.6 po zpracování řetězce $E + T*$ bude ve stavu I_7; stav I_7 zahrnuje položky

$$\begin{aligned} T &\to T * \cdot F \\ F &\to \cdot(E) \\ F &\to \cdot i \end{aligned}$$

které představují právě platné položky pro perspektivní prefix $E+T*$. Podle definice 8.4 je $LR(0)$ položka $[A \to \alpha_1 \cdot \alpha_2]$ platnou pro prefix $\beta\alpha_1$, jestliže existuje v G pravá derivace $S \Rightarrow^* \beta A z \Rightarrow \beta\alpha_1\alpha_2 z$. Uvažujme proto následující tři pravé derivace v G.

$$\left.\begin{array}{ll} (1) & E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F \\ (2) & E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * (E) \\ (3) & E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + Ti \end{array}\right\} \Rightarrow \ldots \Rightarrow E + T * z,\ z \in \Sigma^*$$

Z první, resp. druhé, resp. třetí derivace plyne, že $[T \to T * \cdot F]$ resp. $[F \to \cdot(E)]$, resp. $[F \to \cdot i]$ je platnou položkou pro řetězec $E + T*$. Dá se ukázat, že pro tento řetězec jiná platná položka neexistuje.

Nyní popíšeme postup, jak lze získat z kanonického souboru $LR(0)$ položek, definujícího automat D, funkci akcí a funkci přechodů rozkladové tabulky. Obecně uvažovaný soubor nemusí obsahovat dostačující informaci pro konstrukci jednoznačné rozkladové tabulky. Popsaná metoda bude úspěšná pouze pro jednoduché $LR(1)$ jazyky ($SLR(1)$ jazyky); v mnoha případech je však úspěšná i v aplikacích z oblasti programovacích jazyků.

Algoritmus 8.2 Konstrukce $SLR(1)$ rozkladové tabulky.

Vstup: Kanonický soubor C množin $LR(0)$ položek pro rozšířenou gramatiku G'.

Výstup: Rozkladová tabulka zahrnující funkci akcí a přechodů, je-li G $SLR(1)$ gramatika.

Metoda: Nechť $C = \{I_0, I_1, \ldots, I_n\}$. Stavové symboly analyzátoru jsou $0, 1, \ldots, n$; stav i odpovídá množině položek I_i. Funkce akcí AKCE$[i, a]$ je definována takto:

(1) Je-li $[A \to \alpha \cdot a\beta] \in I_k$ a GOTO$(I_k, a) = I_j$, pak AKCE$[k, a] =$ *shift* j. Symbol a je terminál.

(2) Je-li $[A \to \alpha\cdot] \in I_k, A \to \alpha$ je i-té pravidlo, pak AKCE$[k, a] =$ *reduce* i pro všechna a, pro která $a \in$ FOLLOW(A). Pokud $S' \Rightarrow^* \gamma A$, pak $\sharp \in$ FOLLOW(A).

(3) Je-li $[S' \to S\cdot] \in I_k$, pak AKCE$[k, \sharp] =$ *accept*.

(4) Je-li GOTO$(I_k, A) = I_j, A \in N$, pak PŘECHOD$[k, A] = j$.

(5) Místa rozkladové tabulky, jež nebyla definována body (1)–(4), mají hodnotu *error*.

(6) Počáteční stav analyzátoru odpovídá množině I_0, která obsahuje položku $[S' \to \cdot S]$.

Jestliže v algoritmu 8.2 vznikne z bodů (1) a (2) nejednoznačnost v definici funkce AKCE, pak pro danou gramatiku nelze zkonstruovat $SLR(1)$ rozkladovou tabulku. Říkáme, že gramatika G není jednoduchá $LR(1)$ gramatika. Množina FOLLOW nám dává k dispozici *jeden* terminál za l-frází, podle něhož určujeme, ke kterému pravidlu se má l-fráze redukovat.

Příklad 8.10 Aplikujeme algoritmus 8.2 na gramatiku z příkladu 8.6. Opět využijeme kanonického souboru $LR(0)$ položek z příkladu 8.8 a příslušejícího automatu D z obr. 8.6

Uvažujme položku, která konstituuje počáteční stav analyzátoru

$$\begin{array}{lrcl} I_0: & E' & \to & \cdot E \\ & E & \to & \cdot E + T \\ & E & \to & \cdot T \\ & T & \to & \cdot T * F \\ & T & \to & \cdot F \\ & F & \to & \cdot(E) \\ & F & \to & \cdot i \end{array}$$

Položka $[F \to \cdot(E)]$ implikuje AKCE$[0, ($ = *shift* 4, položka $[F \to \cdot i]$ implikuje AKCE$[0, i] =$ *shift* 5.

Uvažujme

$$\begin{array}{lrcl} I_1: & E' & \to & E\cdot \\ & E & \to & E\cdot +T \end{array}$$

První položka dává AKCE$[1,\sharp]$ = *accept*, druhá, AKCE$[1,+]$ = *shift* 6.

Uvažujme

$$\begin{array}{lrcl} I_2: & E & \to & T\cdot \\ & T & \to & T\cdot *F \end{array}$$

Protože FOLLOW$(E) = \{\sharp, +,)\}$ a $E \to T$ je 2. pravidlo, z první položky plyne AKCE$[2,\sharp]$ = AKCE$[2,+]$ = AKCE$[2,)]$ = *reduce* 2. Z druhé položky plyne AKCE$[2,*]$ = *shift* 7. Pokračujeme-li stejným způsobem, získáme celou rozkladovou tabulku (viz tab. 8.1).

Zřejmě každá $SLR(1)$ gramatika je jednoznačná. Následující příklad ilustruje existenci jednoznačné gramatiky, která není $SLR(1)$ gramatikou.

Příklad 8.11 Uvažujme gramatiku G s pravidly

$$\begin{array}{ll} (1) & S \to L = R \\ (2) & S \to R \\ (3) & L \to *R \\ (4) & L \to i \\ (5) & R \to L \end{array}$$

Kanonický soubor množin $LR(0)$ položek získaný algoritmem 8.1 má tvar:

$$\begin{array}{ll} I_0: & S' \to \cdot S \\ & S \to \cdot L = R \\ & S \to \cdot R \\ & L \to \cdot * R \\ & L \to \cdot i \\ & R \to \cdot L \\ I_1: & S' \to S\cdot \\ I_2: & S \to L\cdot = R \\ & R \to L\cdot \\ I_3: & S \to R\cdot \\ I_4: & L \to *\cdot R \\ & R \to \cdot L \\ & L \to \cdot * R \\ & L \to \cdot i \end{array} \qquad \begin{array}{ll} I_5: & L \to i\cdot \\ I_6: & S \to L = \cdot R \\ & R \to \cdot L \\ & L \to \cdot * R \\ & L \to \cdot i \\ I_7: & L \to *R\cdot \\ I_7: & L \to *R\cdot \\ I_8: & R \to L\cdot \\ I_9: & S \to L = R\cdot \end{array}$$

Uvažujme množinu I_2. První položka implikuje AKCE$[2,=]$ = *shift* 6. Protože FOLLOW(R) obahuje $=$, (viz derivaci $S \Rightarrow L = R \Rightarrow *R = R$), z druhé položky plyne AKCE$[2,=]$ = *reduce* 5. Protože tedy hodnota AKCE$[2,=]$ není jednoznačně definována (ve stavu 2 při vstupním symbolu $=$ vzniká konflikt typu *shift* – *reduce*), není G $SLR(1)$ gramatikou.

8.3.2 Konstrukce kanonické LR rozkladové tabulky

Zůstaňme ještě u situace z předchozího příkladu, která je zdrojem nejednoznačné definice funkce AKCE. Tato situace, jež vede k selhání metody konstrukce SLR rozkladové tabulky, vzniká proto, že analyzátor je ve stavu k, I_k obsahuje položku $[A \to \alpha\cdot]$, zásobník obsahuje perspektivní prefix $\beta\alpha$ a ač je $a \in \mathrm{FOLLOW}(A)$, neexistuje v gramatice větná forma s prefixem βAa. To tedy znamená, že hodnota $\mathrm{AKCE}[k, a] = \mathit{reduce}\ i$ je irelevantní. Skutečně, v gramatice z předchozího příkladu neexistuje větná forma tvaru $R = \cdots$, takže ve stavu 2, který odpovídá perspektivnímu prefixu L je nesprávné provést redukci podle pravidla $R \to L$.

Konflikty v definice rozkladové tabulky mohou být ve většině případů (v případě $LR(1)$ gramatik vždy) odstraněny použitím $LR(1)$ položek. Podle definice 8.3 má $LR(1)$ položka tvar

$$[A \to \alpha_1 \cdot \alpha_2, a], a \in (\Sigma \cup \{\sharp\}).$$

Tato položka je platná pro perspektivní prefix $\beta\alpha_1$, jestliže existuje pravá derivace $S \Rightarrow^* \beta Az\sharp \Rightarrow \beta\alpha_1\alpha_2 z\sharp$ taková, že $a = \mathrm{FIRST}(z\sharp)$. Druhá složka $LR(1)$ položky se nevyužívá v případě $\alpha_2 \neq \epsilon$, avšak položka tvaru $[A \to \alpha_1\cdot, a]$ vede k redukci pouze v případě, že příštím vstupním symbolem je symbol a.

Příklad 8.12 Uvažujme gramatiku s pravidly

$$\begin{aligned} S &\to BB \\ B &\to aB \mid b \end{aligned}$$

a pravou derivaci

$$S \overset{*}{\Rightarrow} aaBab \Rightarrow aaaBab$$

v této gramatice.

Vidíme, že položka $[B \to a \cdot B, a]$ je platná pro perspektivní prefix $\beta\alpha_1 = aaa$ ($\beta = aa, \alpha_1 = a, A = B, \alpha_2 = B, z = ab$). Pro pravou derivaci $S \Rightarrow^* BaB \Rightarrow BaaB$ je pro perspektivní prefix Baa platnou položkou položka $[B \to a \cdot B, \sharp]$.

Metoda výpočtu souboru množin platných $LR(1)$ položek je v podstatě stejná jako metoda výpočtu kanonického souboru množin $LR(0)$ položek. Opět potřebujeme funkce UZÁVĚR a GOTO, podobné stejnojmenným funkcím z odst. 8.3.1.

Uvažujme položku tvaru $[A \to \alpha_1 \cdot B\alpha_2, a]$, jež je v množině platných položek pro perspektivní prefix $\beta\alpha_1$. Pak existuje pravá derivace tvaru $S \Rightarrow^* \beta Aax \Rightarrow \alpha_1 B\alpha_2 ax$. Předpokládejme, že řetězec $\alpha_2 ax$ derivuje terminální řetězec by. Pak pro každé pravidlo tvaru $\beta \to \gamma$ dostaneme derivaci $S \Rightarrow^* \beta\alpha_1 Bby \Rightarrow \beta\alpha_1\gamma by$, tedy $[B \to \cdot\gamma, b]$ je platnou položkou pro perspektivní prefix $\beta\alpha_1$. Zřejmě je b prvkem množiny $\mathrm{FIRST}(\alpha_2 a)$. V případě, že $\alpha_2 \Rightarrow^* \epsilon$ je $b = a$. Poněvadž $a \in (\Sigma \cup \{\sharp\})$ je $\mathrm{FIRST}(\alpha_2 ax) = \mathrm{FIRST}(\alpha_2 a)$.

Nyní již přistoupíme ke konstruktivním definicím funkcí UZÁVĚR a GOTO pro $LR(1)$ položky.

Definice 8.7 Nechť I je množina $LR(1)$ položek gramatiky G. Množina položek UZÁVĚR(I) je definována těmito pravidly:

(1) Každá položka z I je prvkem UZÁVĚR(I).

(2) Je-li $[A \to \alpha_1 \cdot B\alpha_2, a]$ v množině UZÁVĚR(I) a je-li $B \to \gamma$ pravidlo gramatiky, pak pro každý terminál b z množiny FIRST($\alpha_2 a$) je položka $[B \to \cdot\gamma, b]$ také prvkem množiny UZÁVĚR(I).

Definice 8.8 Nechť I je množina $LR(1)$ položek a X symbol gramatiky G. Funkce GOTO(I, X) je definována jako množina

$$\text{UZÁVĚR}(\{[A \to \alpha_1 X \cdot \alpha_2, a] \mid [A \to \alpha_1 \cdot X\alpha_2, a] \in I\})$$

Algoritmus 8.3 Výpočet množin $LR(1)$ položek.

Vstup: Rozšířená gramatika G'.

Výstup: Soubor C množin $LR(1)$ položek platných pro perspektivní prefixy gramatiky G'.

Metoda:

(1)

$$\begin{aligned} I_0 &:= \text{UZÁVĚR}(\{[S' \to \cdot S, \sharp]\}) \\ C &:= \{I_0\} \\ r &:= 0 \end{aligned}$$

(2) pro všechna $I_k \in C, k = 0, 1, \ldots, r$ a pro všechna $X \in (N \cup \Sigma)$ proveď:
je-li $\text{GOTO}(I_k, X) \neq \emptyset \wedge \text{GOTO}(I_k, X) \notin C$ pak
   (a) $r := r + 1$
   (b) $I_r := \text{GOTO}(I_k, X)$
   (c) $C := C \cup \{I_r\}$

(3) Opakuj krok (2), pokud lze k souboru C přidat novou množinu I_r.

Aplikaci algoritmu 8.3 spolu s výpočtem funkcí UZÁVĚR a GOTO nyní ukážeme na příkladě. Pro zkrácení, zápisem $[A \to \alpha_1 \cdot \alpha_2, a_1 \mid a_2]$ budeme rozumět dvě $LR(1)$ položky $[A \to \alpha_1 \cdot \alpha_2, a_1]$ a $[A \to \alpha_1 \cdot \alpha_2, a_2]$.

Příklad 8.13 Uvažujme rozšířenou gramatiku s pravidly

(1) $S' \to S$
(2) $S \to CC$
(3) $C \to cC$
(4) $C \to d$

Začneme výpočtem množiny $I_0 = \text{UZÁVĚR}(\{[S' \to \cdot S, \sharp]\})$. Podle definice 8.6 musíme k položce $[A \to \alpha_1 \cdot B\alpha_2, a]$ přidat položky $[B \to \cdot\gamma, b]$ pro každé B-pravidlo a každé $b \in \text{FIRST}(\alpha_2 a)$; zde je $A = S', \alpha_1 = \epsilon, B = S, \alpha_2 = \epsilon, a = \sharp$. Poněvadž gramatika obsahuje jediné S-pravidlo a $\text{FIRST}(\sharp) = \{\sharp\}$, přidáme k I_0 položku $[S \to \cdot CC, \sharp]$. V této položce je $\alpha_1 = \epsilon, B = C, \alpha_2 = C$ a poněvadž gramatika obsahuje dvě C-pravidla a $\text{FIRST}(C\sharp) = \{c, d\}$, dostáváme v I_0 tyto položky:

$$\begin{aligned} I_0: \quad S' &\to \cdot S, \sharp \\ S &\to \cdot CC, \sharp \\ C &\to \cdot cC, c \mid d \\ C &\to \cdot d, c \mid d \end{aligned}$$

Protože žádná z nových položek nemá nonterminál bezprostředně za tečkou, nelze již žádné další položky přidat a I_0 má tedy 6 položek.

Nyní spočítáme množiny $I_k = \text{GOTO}(I_0, X)$ pro různé symboly X gramatiky.

$$I_1 = \text{GOTO}(I_0, S) = \text{UZÁVĚR}(\{[S' \to S\cdot, \sharp]\})$$

Tento uzávěr je triviální, a proto

$$I_1: \quad S' \to S\cdot, \sharp$$

Pokračujeme

$$I_2 = \text{GOTO}(I_0, C) = \text{UZÁVĚR}(\{[S \to C \cdot C, \sharp]\}),$$

tedy

$$\begin{array}{lrll} I_2: & S & \to & C \cdot C, \sharp \\ & C & \to & \cdot cC, \sharp \\ & C & \to & \cdot d, \sharp \end{array}$$

Pro $X = c$ je

$$I_3 = \text{GOTO}(I_0, c) = \text{UZÁVĚR}(\{[C \to c \cdot C, c \mid d]\}),$$

tj.

$$\begin{array}{lrll} I_3: & C & \to & c \cdot C, c \mid d \\ & C & \to & \cdot cC, c \mid d \\ & C & \to & \cdot d, c \mid d \end{array}$$

Pro $X = d$ je

$$I_4 = \text{GOTO}(I_0, d) = \text{UZÁVĚR}(\{[C \to d\cdot, c \mid d]\}),$$

tj.

$$I_4: \quad C \to d\cdot, c \mid d$$

Z funkcí $\text{GOTO}(I_1, X)$ nedostaneme žádnou novou množinu položek. Z množiny I_2 však plyne

$$I_5 = \text{GOTO}(I_2, C) = \text{UZÁVĚR}(\{[S \to CC\cdot, \sharp]\}),$$

tedy

$$I_5: \quad S \to CC\cdot, \sharp$$

Podobně

$$\begin{array}{lcl} I_6 & = & \text{GOTO}(I_2, c) = \text{UZÁVĚR}(\{[C \to c \cdot C, \sharp]\}), \text{ tj.} \\ I_6 & = & C \to c \cdot C, \sharp \\ & & C \to \cdot cC, \sharp \\ & & C \to \cdot d, \sharp \end{array}$$

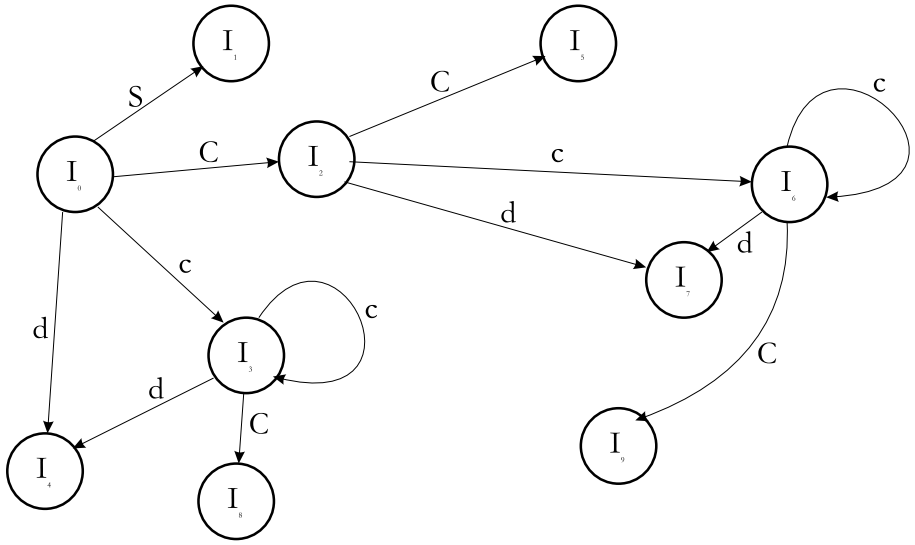
Povšimněme si rozdílu mezi množinami I_3 a I_6. Jednotlivé položky se liší pouze ve druhé složce. První složky, totožné s jedinými složkami $LR(0)$ položek se u I_3 a I_6 shodují. Obecně, vytvoříme-li pro tutéž gramatiku množiny $LR(0)$ položek, pak každá taková množina odpovídá jedné nebo

více množinám $LR(1)$ položek. Nyní dokončíme konstrukci souboru množin $LR(1)$ položek:

$$\begin{aligned} I_7 &= \text{GOTO}(I_2, d) \\ I_7 &: C \to d\cdot, \sharp \\ I_8 &= \text{GOTO}(I_3, C) \\ I_8 &: C \to cC\cdot, c \mid d \\ I_9 &= \text{GOTO}(I_6, C) = \{[C \to cC\cdot, \sharp]\} \end{aligned}$$

žádné další nové množiny I_k nelze již vytvořit.

Deterministický konečný automat D, který přísluší souboru $LR(1)$ položek, se vytváří stejným způsobem jako v případě kanonického souboru množin $LR(0)$ položek. Jeho diagram přechodů je na obr. 8.7.



Obrázek 8.7: Diagram přechodů pro množiny $LR(1)$ položek

Na závěr tohoto odstavce uvedeme algoritmus konstrukce kanonické LR rozkladové tabulky.

Algoritmus 8.4 Konstrukce kanonické $LR(1)$ rozkladové tabulky.

Vstup: Rozšířená gramatika G'.

Výstup: Kanonická LR rozkladová tabulka v případě, že G je $LR(1)$.

Metoda:

(1) Vytvoř soubor C množin $LR(1)$ položek pro gramatiku G'. Nechť $C = \{I_0, I_1, \ldots, I_n\}$.

(2) Množině $I_k, k = 0, 1, \ldots, n$ odpovídá stav k analyzátoru. Funkce akcí pro stav k je definována takto:

   (a) Je-li $[A \to \alpha_1 \cdot a\alpha_2, b] \in I_k$ a $\text{GOTO}(I_k, a) = I_j$, pak polož $\text{AKCE}[k, a] = \textit{shift } j$.

   (b) Je-li $[A \to \alpha\cdot, a] \in I_k, A \to \alpha$ je i-té pravidlo gramatiky G', polož $\text{AKCE}[k, a] = \textit{reduce } i$.

(c) Je-li $[S' \to S\cdot, \sharp] \in I_k$, pak polož AKCE$[k, \sharp] = accept$

Jestliže v tomto bodě vznikne nejednoznačnost v definici funkce AKCE, pak vstupní gramatika není $LR(1)$ gramatikou.

(3) Funkce přechodů pro stav k je definována takto: Je-li GOTO$(I_k, A) = I_j$, pak PŘECHOD$[k, A] = j$.

(4) Všechny položky rozkladové tabulky, které nebyly doposud definovány, mají hodnotu *error*.

(5) Počáteční stav analyzátoru odpovídá množině konstruované z položky $[S \to \cdot S', \sharp]$.

Příklad 8.14 Tab. 8.3 reprezentuje rozkladovou tabulku pro gramatiku z příkladu 8.13, získanou aplikací algoritmu 8.4. Používáme stejného kódování jak v tab. 8.1.

Stav	Akce			Přechod	
	c	**d**	$\sharp$	**S**	**C**
0	s_3	s_4		1	2
1			acc		
2	s_6	s_7			5
3	s_3	s_4			8
4	r_3	r_3			
5			r_1		
6	s_6	s_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

Tabulka 8.3: Rozkladová tabulka

Poznamenejme, že pro tuto gramatiku lze sestrojit SLR rozkladovou tabulku, jež má pouze 7 stavů. Obecně každá $SLR(1)$ gramatika je $LR(1)$ gramatikou, nikoli však naopak.

8.3.3 Konstrukce $LALR$ rozkladové tabulky

V tomto odstavci popíšeme metodu konstrukce $LALR$[2] rozkladové tabulky. Tato metoda vede k výrazně menší rozkladové tabulce, než je kanonická LR tabulka a přitom většina obecných syntaktických konstrukcí programovacích jazyků může být přirozeně $LALR$ gramatikou popsána. I když téměř totéž platí pro SLR gramatiky, existují konstrukce (viz př. 8.11), které nelze pohodlně SLR technikou zpracovat.

Pro srovnání velikostí syntaktických analyzátorů má SLR a $LALR$ rozkladová tabulka řádově stejný počet stavů, který dosahuje u jazyků typu Algol 60, několika stovek. Kanonická rozkladová tabulka má pro stejně rozsáhlý jazyk několik tisíců stavů. Proto je mnohem snazší a ekonomičtější konstruovat SLR nebo $LALR$ rozkladovou tabulku.

Uvažujme opět gramatiku z příkladu 8.13, jež má pravidla

(1) $S \to S'$

(2) $S \to CC$

[2] LookAhead LR

(3) $C \rightarrow cC$

(4) $C \rightarrow d$

spolu s jejími množinami $LR(1)$ položkami. Vezměme dvojici „podobných“ množin např. I_4 a I_7. Obě mají stejnou první složku položky, $C \rightarrow d\cdot$. Druhá složka, (lookahead), je v I_4 tvořena terminály c nebo d a v I_7 pouze symbolem $\sharp$.

Abychom pochopili rozdíl mezi rolí stavu I_4 a I_7 analyzátoru, uvažme, že G generuje jazyk $L(G) = \{w \mid w = c^*dc^*d\}$. Při zpracování vstupního řetězce $cc \cdots cdcc \cdots cd$, (viz tab. 8.3), analyzátor přesouvá první skupinu symbolů c a následující symbol d do zásobníku. Po přečtení tohoto d je analyzátor ve stavu 4. Pak se požaduje redukce podle $C \rightarrow d$, za předpokladu, že vstupním symbolem je c nebo d, tj. symbol začínající zbývající část c^*d vstupního řetězce. Je-li však vstupním symbolem $\sharp$, pak se správně hlásí chyba (např. ccd není v $L(G)$).

Po přečtení druhého d přejde analyzátor do stavy 7. Aby vstupní řetězec patřil do $L(G)$, musí již být na vstupní pásce pouze symbol $\sharp$. Proto je správné, že ve stavu 7, při vstupu $\sharp$, dochází k redukci a při vstupu c nebo d k indikaci chyby.

Nahraďme nyní množiny I_4 a I_7 jejich sjednocením I_{47}. Bude obsahovat tři položky: $[C \rightarrow d\cdot, c \mid d \mid \sharp]$. Funkce GOTO (viz obr. 8.6) z I_0, I_2, I_3 a I_6 pak při vstupu d nabude hodnoty I_{47}. Takto modifikovaný analyzátor se bude chovat v podstatě stejně jako původní, s tím rozdílem, že např. při vstupu ccd nebo $cdcdc$ provede redukci, kdežto původní analyzátor ohlásí chybu. Hlášení chyby se v modifikovaném analyzátoru oddálí do dalšího kroku.

Obecněji, v souboru množin $LR(1)$ položek můžeme vyhledat ty množiny, které mají stejnou množinu prvních složek položek – tzv. *jádro*. V příkladě 8.13 to jsou dvojice množin $(I_4, I_7), (I_3, I_6)$ a (I_8, I_9) s odpovídajícími jádry $\{C \rightarrow d\cdot\}, \{C \rightarrow c \cdot C, C \rightarrow \cdot cC, C \rightarrow \cdot d\}$ a $\{C \rightarrow cC\cdot\}$. Množiny se stejným jádrem sjednotíme do jediné množiny položek. Poznamenejme, že jádrem je vlastně množina $LR(0)$ položek. Protože jádro množiny GOTO(I, X) závisí pouze na jádru množiny I, mohou být funkční hodnoty funkce GOTO pro sjednocené množiny také sjednoceny a může být tak provedena modifikace funkce GOTO. Funkce akcí je modifikována tak, že zachovává všechny akce různé od *error* všech množin položek daného sjednocení.

Podívejme se nyní na problém, zda slučováním stavů analyzátoru nemohou vznikat konflikty v definici funkce akcí. Předpokládejme, že vyjdeme z množin $LR(1)$ položek $LR(1)$ gramatiky, tj. z množin, které nevedou k žádným konfliktům. Předpokládejme, že ve sjednocení množin se stejným jádrem je konflikt typu *shift–reduce*, tj. že toto sjednocení obsahuje položku $[A \rightarrow \alpha\cdot, a]$, která při vstupním symbolu a implikuje redukci a položku $[B \rightarrow \alpha_1 \cdot a\alpha_2, b]$, která při témže vstupním symbolu implikuje přesun do zásobníku. Pak některá množina $LR(1)$ položek, jež je částí sjednocení, obsahuje položku $[A \rightarrow \alpha\cdot, a]$, a protože jádro sjednocovaných množin je stejné, také položku $[B \rightarrow \alpha_1 \cdot a\alpha_2, c]$ pro nějaké c. To ovšem znamená, že už v původní gramatice je konflikt typu *shift–reduce* a soubor množin nepřísluší, proti předpokladu, $LR(1)$ gramatice. Ze vzniklého sporu vyvozujeme, že uvedené slučování množin $LR(1)$ položek nezavléká žádné konflikty typu *shift–reduce* v definici funkce akcí. Příklad 8.15 však ukazuje, že sloučení množin $LR(1)$ položek se stejným jádrem mohou vzniknout konflikty typu *reduce–reduce*.

Příklad 8.15 uvažujme gramatiku G s pravidly

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow aAd \mid bBd \mid aBf \mid bAf \\ A &\rightarrow c \\ B &\rightarrow c \end{aligned}$$

která generuje čtyři řetězce acd, acf, bcd a bcf. Vytvořením souboru množin $LR(1)$ platných položek se lze přesvědčit, že G je $LR(1)$ gramatika. Uvažujme množinu $\{[A \to c\cdot, d], [B \to c\cdot, f]\}$, která má položky platné pro perspektivní prefix ac a množinu $\{[A \to c\cdot, f], [B \to c\cdot, d]\}$ položek platných pro bc. Tyto množiny mají stejné jádro a nejsou zdrojem žádných konfliktů v definici funkce akcí. Provedeme-li však jejich sjednocení, dostaneme položky

$$\begin{aligned} A &\to c\cdot, d \mid f \\ B &\to c\cdot, d \mid f \end{aligned}$$

které jsou jasně zdrojem konfliktu, protože předepisují při vstupním symbolu d nebo f redukci podle pravidla $A \to c$ a zároveň podle pravidla $B \to c$.

Nyní již přistoupíme k formulaci algoritmu konstrukce $LALR$ rozkladové tabulky.

Algoritmus 8.5 Konstrukce $LALR$ rozkladové tabulky.

Vstup: Rozšířená gramatika G'.

Výstup: $LALR$ rozkladová tabulka, je-li G $LALR(1)$ gramatika.

Metoda:

(1) Vytvoř kanonický soubor $C = \{I_0, I_1, \ldots, I_n\} LR(1)$ položek gramatiky G'.

(2) Pro každé jádro $LR(1)$ položek nalezni všechny množiny $LR(1)$ položek a proveď jejich sjednocení. Nový soubor množin $LR(1)$ položek označme $C' = \{J_0, J_1, \ldots, J_m\}$.

(3) Soubor C' odpovídá množině stavů $LALR$ analyzátoru. Pro každý stav $k, k = 0, 1, \ldots, m$ je z množiny položek J_k konstruována funkce akcí AKCE$[k, a], a \in (\Sigma \cup \{\sharp\})$ stejným způsobem jako podle algoritmu 8.4. Jestliže výsledná definice funkce akcí není jednoznačná, pak G není $LALR(1)$.

(4) Nechť množina J_k vznikla sjednocením množin $I_{i_1}, I_{i_2}, \ldots, I_{i_r}$. Pak jádra množin $\mathrm{GOTO}(I_{i_1}, X), \mathrm{GOTO}(I_{i_2}, X), \ldots, \mathrm{GOTO}(I_{i_r}, X)$ jsou táž, poněvadž množiny $I_{i_1}, \ldots, I_{i_r}$ mají stejná jádra. Nechť $J_j = \mathrm{GOTO}(I_{i_1}, X) \cup \ldots \cup \mathrm{GOTO}(I_{i_r}, X)$. Pak upravená funkce GOTO má hodnotu $\mathrm{GOTO}(J_k, X) = J_j$ a příslušná hodnota funkce přechodů je PŘECHOD$[k, X] = j$.

Algoritmus 8.5 je jednoduchý. Pro praktické použití však není vhodný, poněvadž je časově a prostorově stejně náročný, jako algoritmus vytváření kanonické $LR(1)$ tabulky. Tato skutečnost je důsledkem bodu (1) algoritmu 8.5. V [**?**], [**?**] lze nalézt efektivnější algoritmus konstrukce $LALR$ rozkladové tabulky, který počítá $LALR(1)$ položky (sjednocením množin $LR(1)$ položek) přímo.

Příklad 8.16 Uvažujme opět gramatiku

(1) $S' \to S$

(2) $S \to CC$

(3) $C \to cC$

(4) $C \to d$

Její soubor $LR(1)$ položek je uveden v příkladě 8.13, příslušný graf funkce GOTO zobrazuje obr. 8.7.

Podle bodu (2) algoritmu 8.5 obdržíme tato, již zmíněná, sjednocení množin $LR(1)$ položek.

$$\begin{aligned} I_{36} &: C \to c \cdot C, c \mid d \mid \sharp \\ &\quad C \to \cdot cC, c \mid d \mid \sharp \\ &\quad C \to \cdot d, c \mid d \mid \sharp \\ I_{47} &: C \to d\cdot, c \mid d \mid \sharp \\ I_{89} &: C \to cC\cdot, c \mid d \mid \sharp \end{aligned}$$

Nový soubor má tvar $C' = \{I_0, I_1, I_2, I_{36}, I_{47}, I_5, I_{89}\}$.

Výsledná $LALR$ rozkladová tabulka je uvedena jako tab. 8.4. Podívejme se, jak se vytváří upravená funkce GOTO a příslušná funkce přechodů.

Stav	Akce			Přechod	
	c	**d**	♯	**S**	**C**
0	s_{36}	s_{47}		1	2
1			acc		
2	s_{36}	s_{47}			5
36	s_{36}	s_{47}			89
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

Tabulka 8.4: $LALR$ rozkladová tabulka

Uvažujme např. hodnotu GOTO(I_{36}, C). Protože v původním souboru $LR(1)$ položek platilo GOTO$(I_3, C) = I_8$ a GOTO$(I_6, C) = I_9$ je nyní GOTO$(I_{36}, C) = I_{89}$ a tedy PŘECHOD$[36, C] =$ 89. Podobně, z GOTO$(I_2, c) = I_6$ a $I_6 \subset I_{36}$ plyne GOTO$(I_2, c) = I_{36}$, a proto AKCE$[2, c] =$ *shift* 36.

Je-li na vstupu věta c^*dc^*d, pak $LR(1)$ analyzátor (tab. 8.3) i $LALR(1)$ analyzátor (tab. 8.4) provádí přesně stejnou posloupnost přechodů a redukcí, lišící se pouze ve jménech stavů, kterých analyzátor nabývá (uloží-li LR analyzátor na vrchol zásobníku např. stav 3 nebo 6, pak $LALR$ analyzátor uloží v obou případech stav 36). Obecně, v případě syntakticky správného vstupu se LR a $LALR$ analyzátory chovají ekvivalentně.

Jiná situace však nastává, je-li na vstupu řetězec, který nepatří do $L(G)$. V tom případě může $LALR$ analyzátor provést redukci v místě, kde kanonický LR analyzátor indikuje chybu. Je však důležité, že i když takový případ nastane, $LALR$ analyzátor také ohlásí chybu, a to dříve, než by došlo k přesunu dalšího vstupního symbolu do zásobníku. Například při vstupu $ccd\sharp$ kanonický LR analyzátor (viz tab 8.3) uloží do zásobníku řetězec

$$0\ c\ 3\ c\ 3\ d\ 4$$

a ve stavu 4, při vstupu ♯ ohlásí chybu. Na druhé straně $LALR$ analyzátor provede odpovídající posloupnost přechodů a uloží do zásobníku řetězec

$$0\ c\ 36\ c\ 36\ c\ d\ 47.$$

Protože však AKCE$[47, \sharp] =$ *reduce* 3. změní se obsah zásobníku na

$$0\ c\ 36\ c\ 36\ C\ 89.$$

Hodnota AKCE$[89, \sharp] = \mathit{reduce}\ 2$, proto nový obsah zásobníku bude

$$0\ c\ 36\ C\ 89.$$

Ještě jednou provede analyzátor tutéž redukci

$$0\ C\ 2$$

a teprve nyní, protože AKCE$[2, \sharp] = \mathit{error}$, indikuje chybu.

8.4 Cvičení

Cvičení 8.4.1 Pro gramatiku G s pravidly

$$\begin{aligned} S &\rightarrow AS \mid b \\ A &\rightarrow SA \mid a \end{aligned}$$

(a) vypište všechny její $LR(0)$ položky

(b) zkonstruujte nedeterministický konečný automat, jehož stavy jsou $LR(0)$ položky, který přijímá perspektivní prefixy gramatiky G

(c) vytvořte kanonický soubor $LR(0)$ položek gramatiky G a odpovídající konečný deterministický automat D. Ukažte, že tento automat je ekvivalentní automatu z (b)

(d) je G SLR gramatika? Pokud ano, vytvořte SLR rozkladovou tabulku

(e) je G $LALR$ gramatika? Je G $LR(1)$ gramatika?

Cvičení 8.4.2 Vytvořte SLR analyzátor pro gramatiku

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow TF \mid F \\ F &\rightarrow F^* \mid (E) \mid a \mid b \mid \varepsilon \end{aligned}$$

jež popisuje regulární výrazy nad $\{a, b\}$.

Cvičení 8.4.3 Určete, které z těchto gramatik jsou $LR(1)$

(a) $S \rightarrow AB, \quad A \rightarrow 0A1 \mid \epsilon, \quad B \rightarrow 1B \mid 1$

(b) $S \rightarrow 0S1 \mid A, \quad A \rightarrow 1A \mid 1$

(c) $S \rightarrow S + A \mid A, \quad A \rightarrow (S) \mid a(S) \mid a$

Cvičení 8.4.4 Vytvořte kanonickou $LR(1)$ rozkladovou tabulku pro tyto gramatiky:

(a)

$$\begin{aligned} S &\rightarrow ABAC \\ A &\rightarrow aD \\ B &\rightarrow b \mid c \\ C &\rightarrow c \mid d \\ D &\rightarrow D0 \mid 0 \end{aligned}$$

(b) $S \to aSS \mid b$

(c) $S \to SSa \mid b$

Která z těchto gramatik je $LALR$?

Cvičení 8.4.5 Gramatika z příkladu 8.11 není $SLR(1)$. Je $LALR(1)$ nebo $LR(1)$?

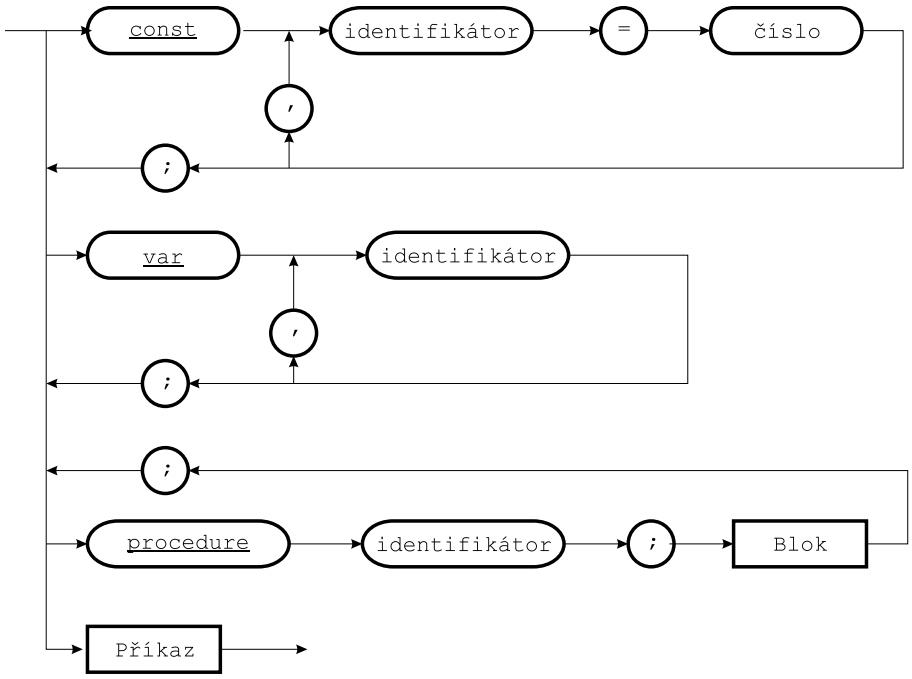
Cvičení 8.4.6 Ukažte, že gramatika z příkladu 8.15 je skutečně $LR(1)$.

Příloha

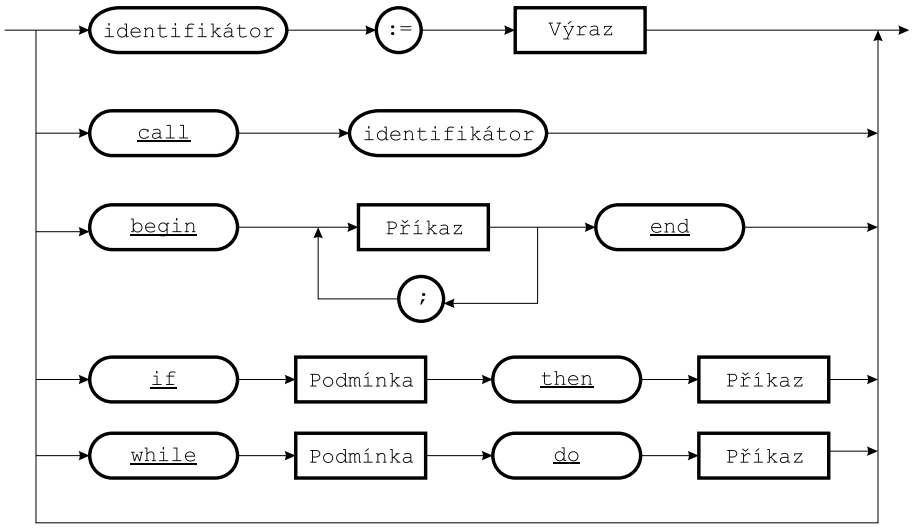
Grafy syntaka jazyka PL0

Program

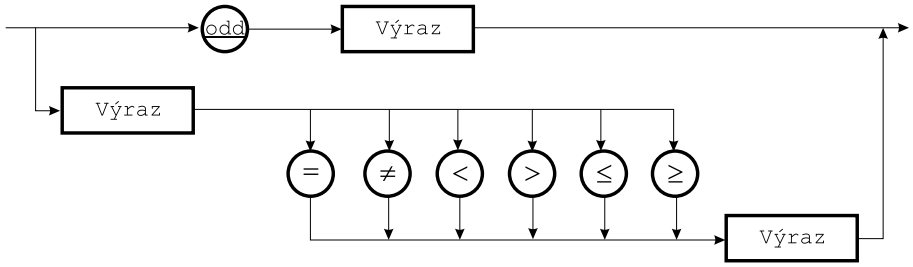
Blok



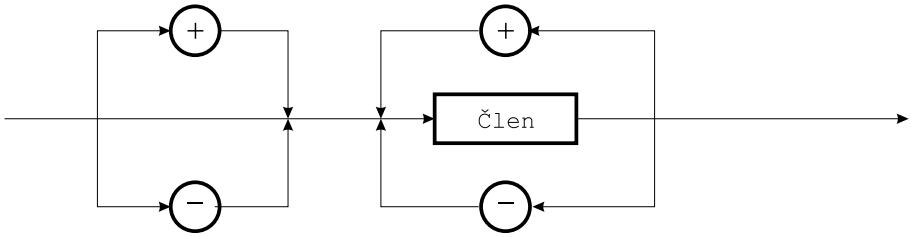
Příkaz



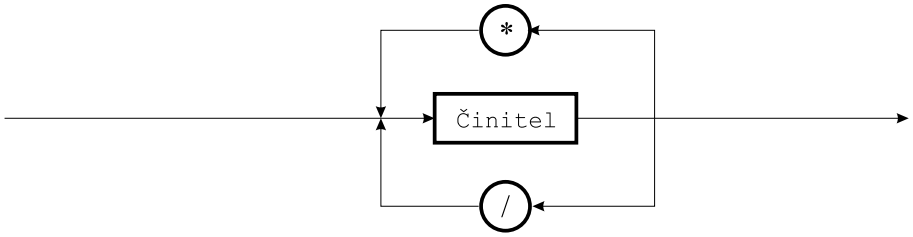
Podmínka



Výraz



Člen



Činitel