Jazyky a překlady

Cvičení

Bořivoj Melichar, Jan Antoš, Jan Holub, Milan Šimánek

červen 2004

Předmluva

Předmět Jazyky a překlady je koncipován obvyklým způsobem. To znamená, že výuka je založena na přednáškách a cvičeních. Podkladem pro přednášky je učební text Melichar, B.: Jazyky a překlady, který vydalo Vydavatelství ČVUT v roce 2003 (2. vydání).

Tento učební text je koncipován jako podklad pro cvičení. Obsahuje především úlohy, které ilustrují jednotlivá probíraná témata. Tyto úlohy jsou vybrány z celé řady aplikačních oblastí a přitom jsou voleny tak, že nevyžadují žádné speciální znalosti. Tento učební text se opírá o učební text určený pro přednášky. Pokud jsou při řešení nějaké úlohy potřebné algoritmy nebo jiné informace, je uvedena reference na příslušnou část textu pro přednášky, která obsahuje kód [JPR03].

Je příjemnou povinností autorů poděkovat všem, kteří se podíleli na přípravě tohoto textu. Byl to především Ing. Petr Kaláb, který se ochotně ujal recenze a svými připomínkami přispěl k tomu, aby text byl srozumitelnější. Dále upozornil na celou řadu chyb a nepřesností. Grafickou podobu textu připravila Olga Vrtišková, která ochotně reagovala na všechny změny, doplňky a opravy.

Praha, červen 2004 kolektiv autorů

Obsah

1 Gramatiky pro jednoduché jazykové konstrukce

Při použití teorie jazyků v praktických aplikacích stojíme mnohdy před problémem, který můžeme formulovat takto:
Daný jazyk je popsán slovním popisem a je třeba sestrojit gramatiku, která tento jazyk generuje.
Takto formulovaný problém není obecně algoritmicky řešitelný, proto sestrojování gramatik může být někdy obtížné. V této kapitole uvedeme několik příkladů, které ukazují, jak pro některé jednoduché jazykové konstrukce sestrojit gramatiky.

1.1 Základní pojmy

Gramatika je čtveřice $G = (N, T, P, S)$, kde
N je množina neterminálních symbolů,
T je množina terminálních symbolů,
P je množina pravidel tvaru $\alpha \to \beta$, $\alpha \in (N \cup T)^* N (N \cup T)^*$, $\beta \in (N \cup T)^*$,
$S \in N$ je počáteční symbol.

Derivace v gramatice $G = (N, T, P, S)$ je relace mezi řetězci definovaná takto: $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$, jestliže v P je pravidlo $\alpha \to \beta$. Symboly $\Rightarrow^k$, $\Rightarrow^+$, $\Rightarrow^*$ budeme používat pro označení k-té mocniny, tranzitivního a tranzitivně-reflexivního uzávěru relace $\Rightarrow$.
Jazyk L generovaný gramatikou $G = (N, T, P, S)$ je definován takto:

$$L(G) = \{w : w \in T^*, S \Rightarrow^* w\}.$$

Bezkontextová gramatika je gramatika, ve které každé pravidlo má tvar:

$$A \to \alpha,\ A \in N,\ \alpha \in (N \cup T)^*.$$

Pravá regulární gramatika je gramatika, ve které každé pravidlo má jeden tvar z těchto tvarů:

$$A \to aB \text{ nebo } A \to a,\ A, B \in N,\ a \in T.$$

Výjimečně může být v regulární gramatice pravidlo $S \to \varepsilon$ v případě, že S se nevyskytuje na pravé straně žádného pravidla.
V dalším textu budeme dodržovat tyto konvence:

- malými písmeny ze začátku abecedy $(a, b, c, \ldots)$ budeme označovat terminální symboly,
- malými písmeny z konce abecedy $(\ldots, x, y, z)$ budeme označovat řetězce terminálních symbolů,
- velkými písmeny $(A, B, C, \ldots)$ budeme označovat neterminální symboly,
- řeckými písmeny $(\alpha, \beta, \gamma, \ldots)$ budeme označovat řetězce terminálních a neterminálních symbolů.

1.2 Gramatiky pro generování konečných jazyků

Příklad 1.1
Sestrojíme gramatiky, které generují jediný řetězec $kuku$. Uvedeme některé gramatiky:

$G_1 = (\{S\}, \{k, u\}, \{S \to kuku\}, S\}$,
$G_2 = (\{S, A\}, \{k, u\}, \{S \to AA, A \to ku\}, S\}$,
$G_3 = (\{S, A, B\}, \{k, u\}, \{S \to AA, A \to kB, B \to u\}, S\}$,
$G_4 = (\{S, A, B, C\}, \{k, u\}, \{S \to kA, A \to uB, B \to kC, C \to u\}, S\}$.

V každé z těchto gramatik existuje právě jedna derivace, která generuje řetězec $kuku$. V G_1 je tato derivace velmi jednoduchá:

$S \Rightarrow kuku.$

V G_2:

$S \Rightarrow AA \Rightarrow kuA \Rightarrow kuku.$

V G_3:

$S \Rightarrow AA \Rightarrow kBA \Rightarrow kuA \Rightarrow kukB \Rightarrow kuku.$

V G_4:

$S \Rightarrow kA \Rightarrow kuB \Rightarrow kukC \Rightarrow kuku.$

Gramatiky G_1, G_2 a G_3 jsou bezkontextové, gramatika G_4 je regulární.

Máme-li sestrojit gramatiku pro jazyk, který se skládá z konečného počtu slov, pak je problém sestrojení gramatiky velmi jednoduchý. V takovém případě může být problém algoritmizován a vytvořená gramatika bude regulární. Regulární gramatiku, která generuje konečný jazyk, můžeme sestrojit pomocí následujícího algoritmu.

Algoritmus 1.2
Sestrojení regulární gramatiky pro konečný jazyk.
Vstup: Množina slov tvořící konečný jazyk $L = \{x_1, x_2, \ldots, x_n\}$.
Výstup: Regulární gramatika $G = (N, T, P, S)$, která generuje daný jazyk L.
Metoda:

1. Sestrojíme množinu terminálních symbolů T tak, že do ní vložíme všechny symboly, které se vyskytují v řetězcích $x_1, x_2, \ldots, x_n$.

2. Pro každý řetězec $x_i = a_{i1}a_{i2}\ldots a_{ik}, x_i \in L$, vytvoříme pravidla:

$$\begin{array}{lcl} S & \to & a_{i1}A_{i1} \\ A_{i1} & \to & a_{i2}A_{i2} \\ & \vdots & \\ A_{i(k-1)} & \to & a_{ik}, \text{ kde} \end{array}$$

   $A_{i1}, A_{i2}, \ldots, A_{i(k-1)}$ jsou neterminální symboly. Vytvořená pravidla vložíme do P.

3. Množinu neterminálních symbolů vytvoříme tak, že do ní vložíme počáteční symbol S a všechny neterminální symboly vzniklé v kroku 2.

4. Počáteční symbol gramatiky bude symbol S.

Příklad 1.3
Je dána množina slov $L = \{les,\ lesák,\ prales,\ zálesák\}$. Pomocí algoritmu 1.2 sestrojíme gramatiku, která generuje jazyk L takto:

1. Množina terminálních symbolů bude $T = \{l, e, s, k, p, r, a, z, á\}$.

2. Množina pravidel bude obsahovat tato pravidla:

S	$\to$	$l\,A_1$	S	$\to$	$l\,A_3$	S	$\to$	$p\,A_7$	S	$\to$	$z\,A_{12}$
A_1	$\to$	$e\,A_2$	A_3	$\to$	$e\,A_4$	A_7	$\to$	$r\,A_8$	A_{12}	$\to$	$á\,A_{13}$
A_2	$\to$	s	A_4	$\to$	$s\,A_5$	A_8	$\to$	$a\,A_9$	A_{13}	$\to$	$l\,A_{14}$
			A_5	$\to$	$á\,A_6$	A_9	$\to$	$l\,A_{10}$	A_{14}	$\to$	$e\,A_{15}$
			A_6	$\to$	k	A_{10}	$\to$	$e\,A_{11}$	A_{15}	$\to$	$s\,A_{16}$
						A_{11}	$\to$	s	A_{16}	$\to$	$á\,A_{17}$
									A_{17}	$\to$	k
		(les)			$(lesák)$			$(prales)$			$(zálesák)$

3. Množina neterminálních symbolů bude definovaná takto:

$$N = \{S\} \cup \{A_i : 1 \leq i \ \leq 17\}.$$

4. Počáteční symbol je S.

Z uvedeného příkladu je vidět, že postup uvedený v algoritmu 1.2 vede na gramatiku se značným počtem neterminálních symbolů a pravidel. V některých případech je možné počet pravidel a počet neterminálů snížit.

Příklad 1.4
Množina slov $L = \{les,\ lesák,\ prales,\ zálesák\}$ generuje také gramatika s těmito pravidly:

S	$\to$	$l\,A_1$	S	$\to$	$l\,A_3$	S	$\to$	$p\,A_7$	S	$\to$	$z\,A_{12}$
A_1	$\to$	$e\,A_2$	A_3	$\to$	$e\,A_4$	A_7	$\to$	$r\,A_8$	A_{12}	$\to$	$á\,A_{13}$
A_2	$\to$	s	A_4	$\to$	$s\,A_5$	A_8	$\to$	$a\,A_9$	A_{13}	$\to$	$l\,A_3$
			A_5	$\to$	$á\,A_6$	A_9	$\to$	$l\,A_1$			
			A_6	$\to$	k						
(les)			$(lesák)$			$(prales)$			$(zálesák)$		

Snížení počtu pravidel jsme dosáhli tím, že jsme uvážili, že dvojice slov (*les, prales*) a (*lesák, zálesák*) mají stejné přípony.

Při snižování počtu pravidel a neterminálních symbolů je nutno postupovat obezřetně, protože můžeme dojít k následujícímu výsledku:

Příklad 1.5
Protože slova *les* a *lesák* mají stejnou předponu *les*, můžeme pravidla gramatiky z předchozího příkladu upravit takto:

S	$\to$	$l\,A_1$	A_2	$\to$	$s\,A_5$	S	$\to$	$p\,A_7$	S	$\to$	$z\,A_{12}$
A_1	$\to$	$e\,A_2$	A_5	$\to$	$á\,A_6$	A_7	$\to$	$r\,A_8$	A_{12}	$\to$	$á\,A_{13}$
A_2	$\to$	s	A_6	$\to$	k	A_8	$\to$	$a\,A_9$	A_{13}	$\to$	$l\,A_1$
						A_9	$\to$	$l\,A_1$			
(les)			$(lesák)$			$(prales)$			$(zálesák)$		

Tato gramatika generuje i jiné řetězce než gramatika předchozí. Jazyk L', který tato gramatika generuje, obsahuje tyto řetězy:

$$L' = L \cup \{\textit{pralesák, záles}\}$$

Sestrojováním gramatik pro jazyky, které jsou zadány výčtem řetězců (v případě konečných jazyků) se zabývá obor nazývaný *inference gramatik* (odvozování gramatik), který patří do umělé inteligence.

1.3 Gramatiky pro generování posloupností a seznamů

V programovacích jazycích se velice často vyskytují konstrukce, ve kterých objekty podobné povahy tvoří posloupnost nebo seznam. Například celé číslo je posloupnost číslic. Identifikátor je posloupnost písmen a číslic začínající písmenem. Zavedeme tuto terminologii:

Posloupnost je řetězec, ve kterém jsou jednotlivé prvky umístěny za sebou bez oddělovacích symbolů.

Seznam je řetězec, ve kterém jsou jednotlivé prvky odděleny oddělovacími symboly.

Pro generování posloupností a seznamů je možno sestrojit regulární gramatiky tehdy, když je možno prvky posloupností generovat regulárními gramatikami. V tabulce 1.1 jsou uvedeny čtyři gramatiky pro generování posloupností a seznamů v případech, kdy buď může nebo nemůže být generován prázdný řetězec. Jednotlivé symboly použité v tabulce 1.1 mají tento význam:

- L je seznam nebo posloupnost (ve všech případech je počátečním symbolem),
- R, R' je zbytek seznamu nebo posloupnosti,
- a je prvek seznamu nebo posloupnosti,
- , (čárka) je použita jako oddělovací symbol v seznamech.

	posloupnosti	seznamy
bez prázdného řetězce	R_1: $L \to a$ $L \to aL$	R_2: $L \to a$ $L \to aR$ $R \to, L$
s prázdným řetězcem	R_3: $L \to \varepsilon$ $L \to aR$ $L \to a$ $R \to aR$ $R \to a$	R_4: $L \to \varepsilon$ $L \to a$ $L \to aR'$ $R' \to, R$ $R \to a$ $R \to aR'$

Tabulka 1.1: Pravidla regulárních gramatik pro generování posloupností a seznamů

Příklad 1.6
Regulární gramatika pro generování zápisu celého čísla bez znaménka má tvar $G_1 = (\{C\}, \{1,2,3,4,5,6,7,8,9,0\}, P, C)$, kde P obsahuje pravidla:

$$\begin{array}{lcl} C & \to & 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0 \\ C & \to & 1C \mid 2C \mid 3C \mid 4C \mid 5C \mid 6C \mid 7C \mid 8C \mid 9C \mid 0C \end{array}$$

Pokud není třeba v zápisu čísla rozlišovat různé číslice, můžeme gramatiku zapsat jednodušším způsobem tak, že všechny číslice nahradíme jediným terminálním symbolem:

$$G_2 = (\{C\}, \{d\}, \{C \to d \mid dC\}, C).$$

Při konstrukci gramatik G_1 a G_2 byla jako vzor použita gramatika R_1 z tabulky 1.1.

Příklad 1.7
Regulární gramatika pro generování identifikátorů podle obvyklé definice (identifikátor je posloupnost písmen a číslic začínající písmenem) má tvar:

$$G = (\{ID, ZI\}, \{p, d\}, P, ID),$$

kde P obsahuje pravidla:

$$\begin{array}{lcl} ID & \to & pZI \mid p \\ ZI & \to & pZI \mid dZI \mid p \mid d \end{array}$$

Při tvorbě této gramatiky byla jako vzor použita gramatika R_1 z tabulky 1.1 s tím, že první prvek posloupnosti má jiný charakter než všechny ostatní prvky.

Neterminální symboly interpretujeme takto: *ID*-identifikátor, *ZI*-zbytek identifikátoru. Terminální symboly mají tento význam: *p*-písmeno, *d*-číslice.

Příklad 1.8
Regulární gramatika pro generování seznamu formálních parametrů procedury nebo funkce v jazyce PASCAL má tvar $G = (\{SFP, ZSFP\}, \{f, ;\}, P, SFP)$, kde P obsahuje pravidla:

$$\begin{array}{lcl} SFP & \rightarrow & f \mid f\ ZSFP \\ ZSFP & \rightarrow & ;\ SFP \end{array}$$

Význam jednotlivých symbolů je tento:
SFP – seznam formálních parametrů,
ZSFP – zbytek seznamu formálních parametrů,
f – formální parametr.
Jako vzor pro sestrojení této gramatiky byla použita gramatika R_2 z tabulky 1.1.

Regulární gramatiky pro generování posloupností a seznamů, jejichž pravidla jsou uvedena v tabulce 1.1, obsahují poměrně velké množství pravidel a to zejména v případě, kdy se připouštějí prázdné posloupnosti nebo seznamy. V řadě případů nelze pro prvky seznamů nebo posloupností sestrojit regulární gramatiky, ale je možno sestrojit bezkontextové gramatiky. V takových případech je možné i pro popis struktury posloupností nebo seznamu použít bezkontextových gramatik pro generování posloupností a seznamu. V tabulce 1.2 jsou uvedena pravidla gramatik s pravou rekurzí, tabulka 1.3 obsahuje pravidla s levou rekurzí a konečně tabulka 1.4 obsahuje pravidla LL(1) gramatik.

Poznámka:
Pravidla s pravou rekurzí mají tvar $A \rightarrow \alpha A$, pravidla s levou rekurzí mají tvar $A \rightarrow A\alpha$. $LL(1)$ gramatiky jsou takové gramatiky, pro které lze sestrojit jednoduchý deterministický syntaktický analyzátor. Podrobněji v kapitole 10.
Gramatiky v tabulkách 1.1 – 1.4 jsou označeny písmeny s indexy. Přitom gramatiky označené různými písmeny se stejnými indexy jsou ekvivalentní:

$$L(R_i) = L(A_i) = L(B_i) = L(C_i), \text{ pro } i = 1, 2, 3, 4.$$

Ve všech pravidlech je použito označení:
L – seznam nebo posloupnost (počáteční symbol),
R – zbytek seznamu nebo posloupnosti,
a – prvek seznamu nebo posloupnosti,
, – oddělovací symbol.

	posloupnosti	seznamy
bez prázdného řetězce	B_1 : $L \to a$ $L \to aL$	B_2 : $L \to a$ $L \to a, L$
s prázdným řetězcem	B_3 : $L \to \varepsilon$ $L \to aL$	B_4 : $L \to \varepsilon$ $L \to R$ $R \to a, R$ $R \to a$

Tabulka 1.2: Pravidla gramatik s pravou rekurzí

	posloupnosti	seznamy
bez prázdného řetězce	C_1 : $L \to a$ $L \to La$	C_2 : $L \to a$ $L \to L, a$
s prázdným řetězcem	C_3 : $L \to \varepsilon$ $L \to La$	C_4 : $L \to \varepsilon$ $L \to R$ $R \to R, a$ $R \to a$

Tabulka 1.3: Pravidla gramatik s levou rekurzí

	posloupnosti	seznamy
bez prázdného řetězce	A_1: $L \to aR$ $R \to aR$ $R \to \varepsilon$	A_2: $L \to aR$ $R \to , aR$ $R \to \varepsilon$
s prázdným řetězcem	A_3: $L \to \varepsilon$ $L \to aL$	A_4: $L \to \varepsilon$ $L \to aR$ $R \to , aR$ $R \to \varepsilon$

Tabulka 1.4: Pravidla $LL(1)$ gramatik

1.4 Gramatiky pro generování závorkových struktur

Kromě seznamů a posloupností se v programovacích jazycích často vyskytují konstrukce nazývané závorkové struktury. Příkladem mohou být:

- kulaté závorky ve výrazech,
- hranaté závorky u indexovaných proměnných,
- příkazové závorky *begin* a *end* používané k vyznačení blokové struktury programu nebo k vyznačení složených příkazů.

Společným znakem těchto struktur je:

1. jejich symetrie: každé „otevírací“ závorce odpovídá „zavírací“ závorka,
2. možnost vnoření: dovnitř jedné dvojice závorek je možno vnořit další dvojici závorek,
3. vnoření je možné do libovolné hloubky.

Na rozdíl od dosud probíraných konstrukcí (konečné konstrukce, posloupnosti a seznamy) není možno závorkové struktury generovat pomocí regulárních gramatik. Pro generování závorkových struktur je nutno použít bezkontextových gramatik.

Příklad 1.9
Gramatika, která generuje jazyk $L = \{(^i\ )^i : i \geq 0\}$ má tvar:

$$G = (\{S\}, \{(,)\}, P, S),$$

kde množina pravidel P obsahuje pravidla:
$S \to \varepsilon$
$S \to (S)$

Příklad 1.10
Jazyk $L = \{(^i\ )^i : i > 0\}$ můžeme generovat pomocí těchto ekvivalentních gramatik:

$$G_1 = (\{S\}, \{(,)\}, P, S),$$

kde množina pravidel P obsahuje pravidla:
$S \to (\ )$
$S \to (S)$

$$G_2 = (\{S, R\}, \{(,)\}, P, S),$$

kde množina pravidel P obsahuje pravidla:
$S \to (R$
$R \to)$
$R \to S$

Příklad 1.11
Jazyk $L = \{a^{i_1}b^{i_1}a^{i_2}b^{i_2}\ldots a^{i_k}b^{i_k} : i_1, i_2, \ldots, i_k \geq 0\}$ můžeme generovat pomocí gramatiky $G = (\{S, R\}, \{a, b\}, P, S\}$, kde P obsahuje pravidla:

$$\begin{array}{lcl} S & \rightarrow & \varepsilon \mid aRbS \\ R & \rightarrow & aRb \mid \varepsilon \end{array}$$

Příklad 1.12
Jazyk závorkové struktury výrazu, tj. takové struktury, kde každé otevírací závorce odpovídá zavírací závorka, závorky jsou do sebe libovolně vnořovány a umisťovány vedle sebe, můžeme generovat pomocí těchto ekvivalentních gramatik:

$$\begin{array}{lcl} G_1 & = & (\{S\}, \{(,)\}, \{S \rightarrow (S)S \mid \varepsilon\}, S), \\ G_2 & = & (\{S\}, \{(,)\}, \{S \rightarrow S(S) \mid \varepsilon\}, S). \end{array}$$

Poznámka:
Je důležité si všimnout, že v téměř všech uvedených příkladech gramatik pro generování závorkových struktur jsou odpovídající si závorky generovány vždy jediným pravidlem, neboli odpovídající si závorky jsou umístěny na pravé straně jediného pravidla. To znamená, že symetrie generovaných řetězů se projevuje i v pravidlech. Výjimku tvoří gramatika G_2 v příkladu 1.10, kde jsou použita pravidla:

$$\begin{array}{lcl} S & \rightarrow & (\, R \\ R & \rightarrow &)\mid S) \end{array}$$

Zde se při generování otevírací závorky do větné formy přidá symbol R, který nutně způsobí přidání zavírací závorky do větné formy. Dosadíme-li do pravidla $S \rightarrow (\, R$ za R, dostaneme pravidla

$$S \rightarrow (\,) \mid (S),$$

což jsou pravidla gramatiky G_1 z příkladu 1.10.

Kromě závorkových struktur typu otevírací závorka — zavírací závorka se v programovacích jazycích vyskytují struktury, které mají podobné vlastnosti, ale skládají se z více než dvou párových symbolů:
počátečního symbolu,
středního symbolu a
koncového symbolu.

Příklad je vybrán z programovacího jazyka, kde jsou možné podmíněné výrazy tvaru:

if výraz *then* výraz *close*

Příklad 1.13
„Třízávorková“ struktura se symboly *if*, *then* a *close* může být generována například touto gramatikou:

$G = (\{S\}, \{if, then, close\}, P, S)$, kde P obsahuje pravidla:

$$S \to \varepsilon \mid if\ S\ then\ S\ close\ S$$

Tato gramatika generuje například tyto řetězce:

if then close
if then close if then close
if if then close then if then close close

Poznámka:
Všimněte si, že všechny řetězce, které jsou generovány uvedenou gramatikou obsahují stejný počet symbolů *if*, *then* a *close*. Připomeňme, že jazyk $L = \{a^n b^n c^n : i \geq 0\}$, který obsahuje řetězce se stejným počtem symbolů a, b, c v uvedeném pořadí, není bezkontextový, ale je kontextový.
Řetězec

$$if^i\ then^i\ close^i$$

je generován gramatikou G pouze pro $i = 0, 1$. To znamená, že uvedená gramatika negeneruje všechny řetězce jazyka L.

Závorkové struktury se mohou skládat i ze čtyř symbolů. Příklad je z jazyka ALGOL 68:

if výraz *then* příkaz *else* příkaz *fi*.

Příklad 1.14
„Čtyřzávorková“ struktura se symboly *if*,*then*,*else*,*fi* může být generována například gramatikou $G = (\{S\}, \{if, then, else, fi\}, P, S)$, kde P obsahuje pravidla:

$$S \to \varepsilon \mid if\ S\ then\ S\ else\ S\ fi\ S.$$

Tato gramatika generuje například řetězce:

if then else fi
if then else fi if then else fi
if if then else fi then if then else fi else fi
if then if then else fi else if then else fi fi

Připomeňme opět, že jazyk $L = \{if^i\ then^i\ else^i\ fi^i : i \geq 0\}$ je kontextový a není bezkontextový. Gramatika G generuje pouze řetězce $if^i\ then^i\ else^i\ fi^i$ pro $i = 0, 1$.

Zobecníme-li dosavadní úvahy o závorkových strukturách, můžeme sestrojit gramatiku pro „n-závorkovou“ strukturu (nikoliv však $a^i b^i c^i \ldots$).

Příklad 1.15
„n-závorková“ struktura se symboly $a_1, a_2, \ldots, a_n$ může být generována gramatikou $G = (\{S\}, \{a_1, a_2, \ldots, a_n\}, P, S)$, kde P obsahuje pravidla

$$S \to \varepsilon \mid a_1 S a_2 S a_3 \ldots S a_n S.$$

Sestrojte několik řetězců pro $n = 5, 6$. Pokuste se najít v nějakém jazyce závorkovou strukturu pro $n > 4$.

1.5 Příklady pro cvičení

Cvičení 1.16
Sestrojte regulární gramatiku, která generuje jazyk $L = \{begin, end, else, if, then\}$.

Cvičení 1.17
Sestrojte regulární gramatiku, která generuje jazyk $L = \{he, she, his, her\}$. Proveďte možná zjednodušení tak, aby gramatika měla co nejmenší počet pravidel a neterminálních symbolů.

Cvičení 1.18
Sestrojte regulární gramatiku, která generuje klíčová slova jazyka PL0:
$L = \{begin, call, const, do, end, if, odd, procedure, read, then, var, while, write\}$.
Proveďte možná zjednodušení výsledné gramatiky.

Cvičení 1.19
Sestrojte regulární gramatiku pro generování identifikátorů v jazyce C takovou, aby měla minimální počet pravidel.

Cvičení 1.20

a) Sestrojte regulární gramatiku, která generuje zápisy desetinných čísel, tj. posloupnost číslic, uvnitř každé je nejvýše jedna tečka.

b) Získanou gramatiku upravte tak, aby tečka mohla být na začátku čísla a na konci čísla. Samostatná tečka není zápisem čísla.

Cvičení 1.21
Sestrojte regulární gramatiky pro generování čísel v PASCALU.

Cvičení 1.22
Sestrojte gramatiku, která generuje parametrovou část deklarace procedury v PASCALU.

Cvičení 1.23
Sestrojte gramatiku, která generuje složené příkazy v jazyce PASCAL. Příkazy jiné než složené považujte za terminální symboly.

Cvičení 1.24
Sestrojte gramatiku, která generuje hlavičku procedury v PASCALU.

Cvičení 1.25
Sestrojte gramatiku, která generuje příkaz `repeat` v PASCALU. Jiný příkaz než `repeat` považujte za terminální symbol.

Cvičení 1.26
Sestrojte gramatiku, která generuje výrazy s operátory + a − bez závorek. Operandy považujte za terminální symboly.

Cvičení 1.27
Sestrojte gramatiku, která generuje výrazy s operátory + a ∗ bez závorek. Operandy považujte za terminální symboly.

Cvičení 1.28
Sestrojte gramatiku, která generuje jazyk $L = \{(^i\ )^{2i} : i > 0\}$.

Cvičení 1.29
Sestrojte gramatiku, která generuje jazyk $L = \{ww^R : w \in \{a, b\}^*\}$, kde w^R je reverze řetězce w.

Cvičení 1.30
Sestrojte gramatiku, která generuje jazyk $L = \{wcw^R : w \in \{a, b\}^*\}$, kde w^R je reverze řetězce w.

Cvičení 1.31
Sestrojte gramatiku, která generuje výrazy s operátory + a − a se závorkami. Operandy považujte za terminální symboly.

Cvičení 1.32
Sestrojte gramatiku, která generuje výrazy s operátory + a ∗ a se závorkami. Operandy považujte za terminální symboly.

Cvičení 1.33
Sestrojte gramatiky, které generují tyto jazyky:

a) $\{a^i b^j : i \geq j \geq 0\}$,

b) $\{a^i b^j : j \geq i \geq 0\}$,

c) $\{a^i b^j c^k : i = j \text{ nebo } j = k\}$,

d) $\{a^i b^j : 3j \geq i \geq 0\}$,

e) $\{w : w \in \{a, b\}^*, w$ obsahuje stejný počet výskytů symbolu a jako symbolu $b\}$,

f) $\{a^i b^i a^j b^j : i, j \geq 0\}$,

g) $\{a^i b^j : 0 \leq i \leq j \leq 2i\}$,

h) $\{a^i b^j c^k : k \leq max(i, j)\}$.

Cvičení 1.34
Sestrojte gramatiky, které generují tyto jazyky:

a) $\{a^n b^n : n \geq 0\} \cup \{a^n c^n : n \geq 0\}$,

b) $\{a^n x^n : n \geq 0, x \in \{b, c\}\}$,

c) $\{a^n w : n \geq 0, w \in \{b, b^n d, c\}^n\}$,

d) $\{a^n 0 b^n : n \geq 0\} \cup \{a^n 1 b^{2n} : n \geq 1\}$,

e) $\{a^i b^i c^k 0 : i, k \geq 0\} \cup \{a^k b^i c^i 1 : i, k :\geq 0\}$.

Cvičení 1.35
Sestrojte gramatiku, která generuje aritmetické výrazy s unárními operátory $+$ a $-$ a s binárními operátory $+, -, *, /, \uparrow$. Výrazy mohou obsahovat závorky. Operandy považujte za terminální symboly.

Cvičení 1.36
Sestrojte gramatiku, která generuje logické výrazy s unárním operátorem *not* a binárními operátory *or* a *and*. Ve výrazu mohou být také kulaté závorky. Operandy považujte za terminální symboly.

2 Regulární gramatiky

Regulární gramatiky jsou nejjednodušším typem gramatik. Přesto se hodí na popis jazyků, se kterými se setkáváme v mnoha aplikacích.

2.1 Základní pojmy

Regulární gramatiky $G = (N, T, P, S)$ dělíme (pro $A, B \in N, a \ \in T, x \in T^*$) na:

1. *pravé regulární* gramatiky, jestliže pravidla v P mají tvar:
   $A \to aB$
   $A \to a$
   $S \to \varepsilon$ v případě, že S není na pravé straně žádného pravidla,

2. *pravé lineární* gramatiky jestliže pravidla v P mají tvar:
   $A \to xB$
   $A \to x$,

3. *levé regulární* gramatiky, jestliže pravidla v P mají tvar:
   $A \to Ba$
   $A \to a$,
   $S \to \varepsilon$ v případě, že S není na pravé straně žádného pravidla,

4. *levé lineární* gramatiky jestliže pravidla v P mají tvar:
   $A \to Bx$
   $A \to x$.

Všechny typy regulárních gramatik jsou navzájem ekvivalentní a můžeme je převádět z jedné formy do jiné.

2.2 Pravé a levé regulární gramatiky

Příklad 2.1
Je dána levá regulární gramatika
$G_L = (\{S, A\}, \{0, 1\}, P_L, S)$, která obsahuje pravidla:
$S \to A1$
$A \to S0$
$A \to 0$.
$L(G_L) = \{(01)^n : \ n > 0\}$.
Sestrojme ekvivalentní pravou regulární gramatiku G_P pomocí Algoritmu 2.14 [JPR03].

Postup bude:
Pro pravidla

$S \to A1$ a $A \to S0$
vzniknou pravidla
$A \to 1S$ a $S \to 0A$.
Pro pravidlo $S \to A1$ se vytvoří pravidlo $A \to 1$.
Pro pravidlo $A \to 0$ se vytvoří pravidlo $S' \to 0A$, kde S' je nový počáteční symbol. Výsledná gramatika je
$G_P = (\{S', S, A\}, \{0, 1\}, P_P, S')$, kde P_P obsahuje pravidla:
$S' \to 0A$
$S \to 0A$
$A \to 1S$
$A \to 1$.

Příklad 2.2
Sestrojme pravou regulární gramatiku pro jazyk, který obsahuje řetězce tvaru:
$list\ id; n; id; id; \ldots ; n; n; id\#$

To znamená, že řetěz začíná slovem $list$ a končí symbolem # a mezi těmito dvěma symboly je seznam identifikátorů (id) a čísel (n), v němž každé dva prvky jsou odděleny středníkem. Identifikátory a čísla považujme za terminální symboly.
Pravá regulární gramatika má tvar:
$G_P = (\{S, L, R\}, \{list, id, n, ;, \#\}, P, S)$, kde P obsahuje pravidla:
$S \to list\, L$
$L \to id\, R \mid n\, R$
$R \to ; L \mid \#$

Tuto pravou regulární gramatiku převedeme na levou regulární gramatiku $G_L = (N', \{list, id, n, ;, \#\}, P', S')$ pomocí Algoritmu 2.12 [JPR03]:

1. $N' = \{S, L, R\} \cup \{S'\}$,

2. $P' = \{\ L \to S\, list$
$R \to L\, id$
$R \to L\, n$
$L \to R\, ;$
$S' \to R\, \#$
$L \to list\ \}$.

Je patrné, že symbol S a pravidlo $L \to S\, list$ jsou zbytečné a můžeme je vynechat, protože pro S není v gramatice žádné pravidlo.

Porovnejme derivace řetězce $list\, id; n; id\#$ v obou gramatikách:

G_P: $S \Rightarrow list\, L \Rightarrow list\, id\, R \Rightarrow list\, id; L \Rightarrow list\, id; n\, R \Rightarrow list\, id; n; L \Rightarrow list\, id; n; id\, R \Rightarrow list\, id; n; id\, \#$,

G_L: $S' \Rightarrow R\, \# \Rightarrow L\, id\, \# \Rightarrow R; id\, \# \Rightarrow L\, n; id\, \# \Rightarrow R; n; id\, \# \Rightarrow L\, id; n; id\, \# \Rightarrow list\, id; n; id\, \#$.

Příklad 2.3
Sestrojme pravou regulární gramatiku pro jazyk nad abecedou $\{a, b\}$, která generuje všechny řetězce začínající řetězcem aba.

$G_P = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde pravidla v P mají tvar:
$S \to aA$
$A \to bB$
$B \to a \mid aC$
$C \to a \mid aC \mid b \mid bC.$

Tuto gramatiku převedeme na levou regulární gramatiku
$G_L = (N', \{a, b\}, P', S')$ pomocí Algoritmu 2.14 [JPR03]:

1. $N' = \{S, A, B, C\} \cup \{S'\}$
2. $P' = \{\ A \to Sa$
   $B \to Ab$
   $C \to Ba$
   $C \to Ca$
   $C \to Cb$
   $S' \to Ba$
   $S' \to Ca$
   $S' \to Cb$
   $A \to a\ \}.$

Symbol S a pravidlo $A \to Sa$ jsou zbytečné a můžeme je z výsledné gramatiky vyloučit, protože pro symbol S není v gramatice žádné pravidlo. Dále je možné nahradit symbol C symbolem S', protože pravidla pro S' a C mají stejné pravé strany, a poté vynechat duplicitní pravidla. Výsledná gramatika bude mít tvar:

$G_L = (\{S', A, B\}, \{a, b\}, P', S')$, kde P' obsahuje:
$S' \to Ba$
$S' \to S'a$
$S' \to S'b$
$B \to Ab$
$A \to a.$

Příklad 2.4
Sestrojme levou regulární gramatiku pro pro jazyk nad abecedou $\{a, b\}$, která generuje všechny řetězce končící řetězcem aba.

$G_L = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde pravidla v P mají tvar:
$S \to Aa$
$A \to Bb$
$B \to a \mid Ca$
$C \to a \mid Ca \mid b \mid Cb.$

Tuto gramatiku převedeme na pravou regulární gramatiku
$G_P = (N', \{a, b\}, P', S')$ pomocí Algoritmu 2.14 [JPR03].

1. $N' = \{S, A, B, C\} \cup \{S'\}$
2. $P' = \{\ A \rightarrow aS$
   $B \rightarrow bA$
   $C \rightarrow aB$
   $C \rightarrow aC$
   $C \rightarrow bC$
   $A \rightarrow a$
   $S' \rightarrow aB$
   $S' \rightarrow aC$
   $S' \rightarrow bC\ \}.$

Symbol S a pravidlo $A \rightarrow aS$ jsou zbytečné a můžeme je z výsledné gramatiky vyloučit, protože pro symbol S není v gramatice žádné pravidlo. Dále je možné nahradit symbol C symbolem S' a poté vynechat duplicitní pravidla. Výsledná gramatika bude mít tvar:

$G_P = (\{S', A, B\}, \{a, b\}, P', S')$, kde P' obsahuje:
$S' \rightarrow aB$
$S' \rightarrow aS'$
$S' \rightarrow bS'$
$B \rightarrow bA$
$A \rightarrow a.$

2.3 Pravé a levé lineární gramatiky

Příklad 2.5
Sestrojme pravou lineární gramatiku pro jazyk, který obsahuje řetězce tvaru:

$list\ id; n; id; id; \ldots; n; n; id\#$

Jedná se o stejný jazyk jako v příkladu 2.2.

Pravá lineární gramatika má tvar:
$G_{PL} = (\{S, R\}, \{list, id, n, ;, \#\}, P, S)$, kde
$S \rightarrow list\ nR$
$S \rightarrow list\ idR$
$R \rightarrow ; nR$
$R \rightarrow ; idR$
$R \rightarrow \#$
Tuto gramatiku převedeme na pravou regulární gramatiku $G_{PR} = (N', T, P', S')$, přičemž použijeme Algoritmus 2.9 [JPR03]:

1. $N' = N$, $P' = \emptyset$,

2. $P' = \{\, R \rightarrow \#$

3. $\begin{aligned} S &\rightarrow list\, S_1 \\ S_1 &\rightarrow n\, R \\ S_1 &\rightarrow id\, R \\ R &\rightarrow ;\, R_1 \\ R_1 &\rightarrow n\, R \\ R_1 &\rightarrow id\, R \,\}, \end{aligned}$

   $N' = \{S, R\} \cup \{S_1, R_1\}$.

Protože pravidla se symboly S_1 a R_1 na levé straně mají stejné pravé strany, můžeme pravidla s S_1 a R_1 na levé straně sloučit. Vznikne tato gramatika:

$G'_{PR} = (\{S, R_1, R\}, \{list, id, n, ;, \#\}, P'', S)$ s pravidly
$S \rightarrow list\, R_1$
$R_1 \rightarrow n\, R \mid id\, R$
$R \rightarrow ; R_1 \mid \#$.

Výsledná gramatika je stejná (až na pojmenování neterminálních symbolů) jako gramatika v příkladu 2.2.

Příklad 2.6
Sestrojme pravou lineární gramatiku pro jazyk nad abecedou $\{a, b\}$, která generuje všechny řetězce začínající řetězcem aba.

$G_{PL} = \{S, A\}, \{a, b\}, P, S\}$, s pravidly:
$S \rightarrow abaA$
$A \rightarrow aA \mid bA \mid \varepsilon$.
Tuto gramatiku dále převedeme na pravou regulární gramatiku
$G_{PR} = (N', T, P', S)$ pomocí Algoritmu 2.9 [JPR03]:

1. $N' = N$, $P' = \emptyset$,

2. $\begin{aligned} P' = \{\, A &\rightarrow aA \\ A &\rightarrow bA\}, \end{aligned}$

3. $\begin{aligned} P' = P' \cup \{\, S &\rightarrow aS_1 \\ S_1 &\rightarrow bS_2 \\ S_2 &\rightarrow aA \,\}, \end{aligned}$

   $N' = \{S, A\} \cup \{S_1, S_2\}$,

4. neprovedeme nic,

5. $P' = P' \cup \{A \to \varepsilon\}$,

6. po odstranění ε-pravidel pomocí Algoritmu 2.4 [JPR03] dostaneme novou množinu pravidel:

$$\begin{aligned} P'' = \{S &\to aS_1 \\ S_1 &\to bS_2 \\ S_2 &\to aA \mid a \\ A &\to a \mid b \mid aA \mid bA \}. \end{aligned}$$

Tato gramatika je stejná jako gramatika v příkladu 2.3.

Příklad 2.7
Sestrojme levou lineární gramatiku pro jazyk nad abecedou $\{a, b\}$, která generuje všechny řetězce končící řetězcem aba.
$G_{PL} = \{S, A\}, \{a, b\}, P, S\}$, s pravidly:
$S \to Aaba$
$A \to Aa \mid Ab \mid \varepsilon$.
Tuto gramatiku dále převedeme na levou regulární gramatiku
$G_{LR} = (N', T, P', S)$ pomocí Algoritmu 2.9 [JPR03] :

1. $N' = N$, $P' = \emptyset$,

2. $$\begin{aligned} P' = \{ A &\to Aa \\ A &\to Ab\}, \end{aligned}$$

3. $$\begin{aligned} P' = P' \cup \{ S &\to S_1a \\ S_1 &\to S_2b \\ S_2 &\to Aa \}, \end{aligned}$$

   $N' = \{S, A\} \cup \{S_1, S_2\}$,

4. neprovedeme nic,

5. $P' = P' \cup \{A \to \varepsilon\}$,

6. po odstranění ε-pravidel pomocí Algoritmu 2.4 [JPR03] dostaneme novou množinu pravidel:

$$\begin{aligned} P' = \{ S &\to S_1a \\ S_1 &\to S_2b \\ S_2 &\to Aa \mid a \\ A &\to a \mid b \mid Aa \mid Ab \}. \end{aligned}$$

Tato gramatika je stejná jako gramatika v příkladu 2.4.

Příklad 2.8
Sestrojme pravou lineární gramatiku, která generuje výrazy s operátory + a $*$ bez závorek.

$G_{PL} = (\{E, R\}, \{a, +, *\}, P, E)$, kde P obsahuje pravidla:
$E \to a \ + R \mid a \ * R \mid a$
$R \to E$

Tuto gramatiku převedeme na pravou regulární gramatiku pomocí Algoritmu 2.9 [JPR03].

V jednotlivých krocích dostaneme tyto výsledky:

1. $N' = \{E, R\}, P' = \emptyset$,
2. $P' = \{E \to a\}$,
3. $P' = P' \cup \{E \to a \ E_1, E_1 \to +R, E_1 \to *R\}$,
4. neprovedeme nic,
5. $P' = P' \cup \{R \to E\}, G_1 = (\{E, E_1, R\}, \{a, +, *\}, P', E)$,
6. neprovedeme nic,
7. odstraníme jednoduchá pravidla pomocí Algoritmu 2.6 [JPR03].
   $P' = \{E \to a \mid aE_1, E_1 \to +R \mid *R, R \to a \mid aE_1\}$.

Protože neterminální symboly E a R generují stejné řetězce, můžeme jeden z nich nahradit druhým. Potom:

$P = \{E \to a \mid aE_1, E_1 \to +E \mid *E\}$.

Tvůrčí úkol

Navrhněte algoritmus pro přímou transformaci pravé (levé) lineární gramatiky na levou (pravou) lineární gramatiku.

2.4 Řetězce a jejich části

Je–li dán řetězec $x = x_1x_2 \dots x_n$, pak řetězec $y = x_1x_2 \dots x_j, j \leq n$, je předponou řetězce x, řetězec $z = x_jx_{j+1} \dots x_n, j \geq 1$ je příponou řetězce x a řetězec $x_ix_{i+1} \dots x_j, 1 \leq i \leq j \leq n$ je neprázdný podřetězec (faktor) řetězce x. Množinu všech předpon řetězce x označme $Pref(x)$. Množinu všech přípon označme $Suf(x)$. Množinu všech neprázdných podřetězců daného řetězce x označme $Fac(x)$. Podposloupnost řetězce x nazveme řetězec y, který vznikne z řetězce x tak, že jeho některé symboly vynecháme. Množinu všech podposloupností řetězce x označme $Sub(x)$.

Příklad 2.9
Sestrojme pravou regulární gramatiku, která generuje řetězec $x = abba$ a všechny jeho předpony.

$G_{Pref}(\{S, A, B, C\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to aA$
$S \to a$
$A \to bB$
$A \to b$
$B \to bC$
$B \to b$
$C \to a$

Příklad 2.10
Sestrojme levou regulární gramatiku, která generuje řetězec $x = abba$ a všechny jeho přípony.

$G_{Suf} = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to Aa$
$S \to a$
$A \to Bb$
$A \to b$
$B \to Cb$
$B \to b$
$C \to a$

Příklad 2.11
Sestrojme pravou lineární gramatiku, která generuje množinu $Fac(abba)$.

$G_{Fac} = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to aA$
$S \to bB$
$S \to bC$
$A \to bB$
$A \to \varepsilon$
$B \to bC$
$B \to \varepsilon$
$C \to a$
$C \to \varepsilon$

Příklad 2.12
Sestrojme pravou regulární gramatiku, která generuje množinu $Sub(abba)$.

$G_{Sub} = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to aA$
$S \to bB$
$S \to b$
$S \to a$
$S \to \varepsilon$
$A \to bB$
$A \to b$
$A \to a$
$B \to bC$
$B \to a$
$B \to b$
$C \to a$

2.5 Gramatiky a operace nad jazyky

S jazyky je možné provádět množinové operace sjednocení, průnik a rozdíl. Dále je možné vytvořit komplement jazyka. Další operace související s tím, že jazyky jsou množiny řetězců, jsou součin jazyků, mocnina jazyka a iterace jazyka (odst. 1.1 [JPR03]). Algoritmy, které generují jazyky vzniklé sjednocením, součinem a iterací jazyka (odst. 1.4 [JPR03]) jsou orientovány na bezkontextové gramatiky. Upravme tyto algoritmy tak, aby je bylo možno ve zjednodušené podobě použít pro lineární a regulární gramatiky. Algoritmus pro konstrukci gramatiky pro sjednocení jazyků (Algoritmus 1.20 [JPR03]) je možno přímo použít pro levé i pravé lineární gramatiky. Pro regulární gramatiky použít nelze, protože vznikají dvě jednoduchá pravidla a může být porušena vlastnost regulárních gramatik, která se týká generování prázdného řetězce. Je proto nutné tento algoritmus upravit.

Algoritmus 2.13
Konstrukce regulární gramatiky (levé nebo pravé) pro sjednocení jazyků L_1 a L_2, které jsou popsány buď levými nebo pravými regulárními gramatikami.
Vstup: Dvě regulární gramatiky G_1 a G_2 (obě buď levé nebo pravé), které generují jazyky L_1 a L_2.
Výstup: Regulární gramatika G taková, že $L(G) = L_1 \cup L_2$.
Metoda: Označme $G_1 = (N_1, T, P_1, S_1)$ a $G_2 = (N_2, T, P_2, S_2)$.

1. Za předpokladu, že množiny neterminálních symbolů mají prázdný průnik, vytvoříme gramatiku:
   $G' = (N_1 \cup N_2 \cup \{S\}, T, P_1 \cup P_2 \cup \{S \to S_1 | S_2\}, S)$, kde S je nový neterminální symbol.
2. Jsou–li v gramatikách G_1 a G_2 pravidla $S_1 \to \varepsilon$ a $S_2 \to \varepsilon$ nebo je–li toto pravidlo alespoň v jedné z nich, pak tato pravidla vyloučíme a přidáme pravidlo $S \to \varepsilon$. Dostaneme gramatiku:
   $G'' = (N, T, (P_1 - \{S_1 \to \varepsilon\}) \cup (P_2 - \{S_2 \to \varepsilon\}) \cup \{S \to \varepsilon\}$
   $\cup \{S \to S_1 | S_2\}, S)$.
   V opačném případě $G'' = G'$.
3. Z gramatiky G'' vyloučíme jednoduchá pravidla $S \to S_1$ a $S \to S_2$ tak, že pro všechna pravidla tvaru $S_1 \to \alpha$ a $S_2 \to \beta$ přidáme do P pravidla $S \to \alpha$ a $S \to \beta$. Výsledná gramatika $G = (N, T, P, S)$ je levá nebo pravá regulární gramatika v závislosti na tom, jakého typu byly vstupní gramatiky.
4. Nakonec vyloučíme zbytečné symboly pomocí Algoritmu 3.12 [JPR03].

Poznámka:
Pokud vstupní gramatiky G_1 a G_2 v Algoritmu 2.13 mají nějaké společné neterminální symboly, pak stačí v jedné z těchto gramatik takové symboly přejmenovat. Pokud jsou množiny terminálních symbolů vstupních gramatik G_1 a G_2 různé, stačí vytvořit sjednocení $T = T_1 \cup T_2$ a množinu T použít v obou gramatikách.

Příklad 2.14
Jsou dány jazyky $L_1 = a^*$, $L_2 = b^*$. Sestrojme regulární gramatiku, která generuje jazyk $L = a^* \cup b^*$. Pravé regulární gramatiky G_1 a G_2 pro jazyky L_1 a L_2 mají tvar:

$G_1 = (\{S_1, A\}, \{a, b\}, P_1, S_1)$, kde P_1 obsahuje pravidla:

$S_1 \to \varepsilon$	$S_1 \to aA$	$A \to aA$
	$S_1 \to a$	$A \to a$

$G_2 = (\{S_2, B\}, \{a, b\}, P_2, S_2)$, kde P_2 obsahuje pravidla:

$S_2 \to \varepsilon$	$S_2 \to bB$	$B \to bB$
	$S_2 \to b$	$B \to b$

Při použití Algoritmu 2.12 vzniknou tyto gramatiky:

$G' = (\{S, S_1, S_2, A, B\}, \{a, b\}, P', S)$, kde P' obsahuje pravidla:

$S \to S_1$	$S_1 \to \varepsilon$	$S_2 \to \varepsilon$
$S \to S_2$	$S_1 \to aA$	$S_2 \to bB$
	$S_1 \to a$	$S_2 \to b$
	$A \to aA$	$B \to bB$
	$A \to a$	$B \to b$

Protože obě gramatiky obsahují ε–pravidla, vyloučíme je a vytvoříme novou gramatiku:

$G'' = (\{S, S_1, S_2, A, B\}, \{a, b\}, P'', S)$, kde P'' obsahuje pravidla:

$S \to \varepsilon$	$S_1 \to aA$	$S_2 \to bB$
$S \to S_1$	$S_1 \to a$	$S_2 \to b$
$S \to S_2$	$A \to aA$	$B \to bB$
	$A \to a$	$B \to b$

Po vyloučení jednoduchých pravidel dostaneme gramatiku:

$\overline{G} = (\{S, S_1, S_2, A, B\}, \{a, b\}, \overline{P}, S)$, kde $\overline{P}$ obsahuje pravidla:

$S \to \varepsilon$	$S_1 \to aA$	$S_2 \to bB$
$S \to aA$	$S_1 \to a$	$S_2 \to b$
$S \to a$	$A \to aA$	$B \to bB$
$S \to bB$	$A \to a$	$B \to b$
$S \to b$		

Je zřejmé, že v gramatice $\overline{G}$ jsou symboly S_1 a S_2 zbytečné a můžeme je proto vyloučit spolu s pravidly, ve kterých se vyskytují. Tak dostaneme výslednou gramatiku:

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

$S \to \varepsilon$	$A \to aA \mid a$
$S \to aA \mid a$	$B \to bB \mid b$
$S \to bB \mid b$	

Poznámka:
V uvedeném příkladu jsou počáteční symboly vstupních gramatik zbytečné. Zdůvodněte proč.

Algoritmus pro konstrukci gramatiky pro součin jazyků (Algoritmus 1.21 [JPR03]) je určen pro bezkontextové gramatiky a není možno jej použít ani pro lineární ani pro regulární gramatiky. Důvodem je vznik pravidla $S \to S_1S_2$, které nelze jednoduše nahradit pravidly lineárních nebo regulárních gramatik. Proto jej musíme pro tyto případy přeformulovat.

Následující algoritmus je zformulován pro pravé regulární gramatiky. Pro levé regulární gramatiky je možno jej jednoduše upravit.

Algoritmus 2.15
Konstrukce pravé regulární gramatiky pro součin jazyků L_1 a L_2, které jsou popsány pravými regulárními gramatikami.
Vstup: Dvě pravé regulární gramatiky G_1 a G_2, které generují jazyky L_1 a L_2.
Výstup: Pravá regulární gramatika G taková, že $L(G) = L_1.L_2$.
Metoda: Označíme $G_1 = (N_1, T, P_1, S_1)$ a $G_2 = (N_2, T, P_2, S_2), N_1 \cap N_2 = \emptyset$.

1. Vytvořme gramatiku
   $G' = (N_1 \cup N_2 \cup \{S\}, T, P', S)$, kde P' vytvoříme takto:
   (a) $P' = P_2 \cup \{S \to S_1\}$,
   (b) do P' přidáme všechna pravidla tvaru $A \to aB$, která jsou v množině P_1,
   (c) pro všechna pravidla tvaru $A \to a$ z množiny P_1 vytvoříme pravidla tvaru $A \to aS_2$ a přidáme je do P',
   (d) pro pravidlo $S_1 \to \varepsilon$ vytvoříme pravidlo $S \to S_2$ a přidáme do P'.
2. Vyloučíme jednoduchá pravidla vzniklá v bodech 1.a) a 1.d) pomocí Algoritmu 2.6 [JPR03].
3. Vyloučíme ε–pravidla pomocí Algoritmu 2.4 [JPR03].
4. Vyloučíme zbytečné symboly pomocí Algoritmu 3.12 [JPR03].

Algoritmus 2.15 je možno použít i pro lineární gramatiky. V tomto případě není nutné vyloučení ε–pravidel a jednoduchých pravidel.

Příklad 2.16
Sestrojme regulární gramatiku, která generuje jazyk $L = a^*b^*$. Pravé regulární gramatiky G_1 a G_2 jsou uvedeny v příkladu 2.14. Při použití Algoritmu 2.15 vznikne gramatika:

$G' = (\{S, S_1, S_2, A, B\}, \{a, b\}, P', S)$, kde množina pravidel P' je postupně konstruována takto:

a) $P'_a = \{S \to S_1, S_2 \to \varepsilon, S_2 \to bB, S_2 \to b, B \to bB, B \to b\}$,

b) $P'_b = P'_a \cup \{S_1 \to aA, A \to aA\}$,

c) $P'_c = P'_b \cup \{S_1 \to aS_2, A \to aS_2\}$,

d) $P'_d = P'_c \cup \{S \to S_2\}$.

Po těchto operacích obsahuje množina $P' = P'_d$ tato pravidla:

$S \to S_1$	$S_1 \to aA$	$S_2 \to \varepsilon$
$S \to S_2$	$S_1 \to aS_2$	$S_2 \to bB$
	$A \to aA$	$S_2 \to b$
	$A \to aS_2$	$B \to bB$
		$B \to b$

Po vyloučení jednoduchých pravidel získáme gramatiku:

$G'' = (\{S, S_1, S_2, A, B\}, \{a, b\}, P'', S)$, kde P'' obsahuje pravidla:

$S \to aA$	$S_1 \to aA$	$S_2 \to \varepsilon$
$S \to aS_2$	$S_1 \to aS_2$	$S_2 \to bB$
$S \to \varepsilon$	$A \to aA$	$S_2 \to b$
$S \to bB$	$A \to aS_2$	$B \to bB$
$S \to b$		$B \to b$

Je zřejmé, že symbol S_1 je zbytečný a proto jej vyloučíme i s pravidly, kde se vyskytuje. Po vyloučení ε–pravidel získáme gramatiku:

$\overline{G} = (\{S, S_2, A, B\}, \{a, b\}, \overline{P}, S)$, kde $\overline{P}$ obsahuje pravidla:

$S \to aA$	$A \to aA$	$S_2 \to bB$
$S \to aS_2$	$A \to aS_2$	$S_2 \to b$
$S \to a$	$A \to a$	$B \to bB$
$S \to \varepsilon$		$B \to b$
$S \to bB$		
$S \to b$		

Gramatika $\overline{G}$ neobsahuje žádné zbytečné symboly a proto výsledná gramatika $G = G'$.

Algoritmus pro konstrukci gramatiky pro iteraci jazyka (Algoritmus 1.22 [JPR03]) opět nelze použít ani pro lineární ani pro regulární gramatiky. Proto jej přeformulujeme pro pravé regulární gramatiky.

Algoritmus 2.17
Konstrukce pravé regulární gramatiky pro iteraci jazyka L, který je popsán pravou regulární gramatikou.
Vstup: Pravá regulární gramatika G, která generuje jazyk L.
Výstup: Pravá regulární gramatika G', která generuje jazyk $L(G') = L^*$.
Metoda: Označme $G = (N, T, P, S)$.

1. Vytvořme gramatiku $G_1 = (N \cup \{S'\}, T, P_1, S')$, kde P_1 vytvoříme takto:

(a) $P_1 = P - \{S \to \varepsilon\} \cup \{S' \to \varepsilon\}$, když $S \to \varepsilon \in P$,

(b) do P_1 přidáme pravidla tvaru $S' \to \alpha$ pro všechna pravidla tvaru $S \to \alpha$ z množiny pravidel P,

(c) do P_1 přidáme pravidla tvaru $S' \to a\ S'$ pro všechna pravidla tvaru $S \to a$ z množiny pravidel P,

(d) do P_1 přidáme pravidla tvaru $A \to aS'$ pro všechna pravidla tvaru $A \to a$ z množiny pravidel P, kde $A \neq S$.

2. Po vynechání zbytečných symbolů (Algoritmus 3.12 [JPR03]) dostaneme výslednou gramatiku G' takovou, že $L(G') = L^*$.

Příklad 2.18
Je dán jazyk $L = ab^* \cup \{\varepsilon\}$. Pravá regulární gramatika, která jej generuje má tvar $G = (\{S, B\}, \{a, b\}, P, S)$, kde množina P obsahuje pravidla:

$S \to \varepsilon$	$B \to bB$
$S \to aB$	$B \to b$
$S \to a$	

Gramatika, která generuje jazyk L^* má tvar:

$G_1 = (\{S', S, B\}, \{a, b\}, P_1, S')$, kde množina P_1 obsahuje pravidla:

$S' \to \varepsilon$	$S' \to aS'$
$S' \to aB$	$B \to bB$
$S' \to a$	$B \to bS'$
	$B \to b$

2.6 Příklady pro cvičení

Cvičení 2.19
Sestrojte regulární gramatiku, která generuje jazyk $L = \{(01)^n : n > 0\}$.

Cvičení 2.20
Sestrojte regulární gramatiku pro jazyk nad abecedou {0,1}, který se skládá z řetězců, ve kterých jsou buď dvě nuly nebo dvě jedničky těsně vedle sebe.

Cvičení 2.21
Sestrojte regulární gramatiku, která generuje jazyk nad abecedou {0,1} takový, že v každém řetězci je každá jednička následována alespoň dvěma nulami.

Cvičení 2.22
Sestrojte regulární gramatiku, která generuje jazyk nad abecedou {0,1} takový, že uvnitř každého řetězce je podřetězec 01010.

Cvičení 2.23
Sestrojte regulární gramatiku, která generuje jazyk nad abecedou {0,1}, a v každém řetězci je alespoň 5 nul a 2 jedničky.

Cvičení 2.24
Sestrojte pravou (G_P) a levou (G_L) regulární gramatiku, které generují jazyk nad abecedou {0,1}, a v každém řetězci je počet nul dělitelný třemi. Gramatiku G_P převeďte na levou regulární gramatiku G'_L a tuto porovnejte s G_L.

Cvičení 2.25
Převeďte následující pravé regulární gramatiky na levé regulární gramatiky.

a) $G = (\{ID, ZI\}, \{p, d\}, P, ID)$, kde P obsahuje pravidla:
$ID \rightarrow p\,ZI | \,p$
$ZI \rightarrow p\,ZI | \,d\,ZI | \,p | \,d$

b) $G = (\{SFP, ZSFP\}, \{id, ;\}, P, SFP)$, kde P obsahuje pravidla:
$SFP \rightarrow id | \,id\,ZSFP$
$ZSFP \rightarrow ;\, SFP$

Cvičení 2.26
Je dán konečný jazyk $L = \{závod, vodník, podvod, závodník\}$. Proveďte tyto kroky:

a) Sestrojte lineární gramatiku G_{lin} pro L.

b) Pro získanou gramatiku sestrojte ekvivalentní regulární gramatiku G_{reg}.

c) Proveďte možná zjednodušení gramatiky G_{reg}.

Cvičení 2.27
Sestrojte pravou (G_{PL}) a levou (G_{LL}) lineární gramatiku, která generují řetězce nad abecedou {p,n,:,;} (p ... příkaz, n ... návěští), které mají tyto vlastnosti:

a) seznam příkazů může být prázdný,

b) každý příkaz je od následujícího oddělen jedním nebo několika středníky,

c) každý příkaz může mít libovolný počet návěští, za každým návěštím je dvojtečka.

Gramatiku G_{PL} převeďte na pravou regulární gramatiku G_{PR} a gramatiku G_{LL} převeďte na levou regulární gramatiku G_{LR}. Gramatiku G_{LR} dále převeďte na pravou regulární gramatiku G'_{PR} a porovnejte s G_{PR}.

Cvičení 2.28
Navrhněte algoritmus na konstrukci regulární gramatiky, která generuje doplněk zadaného regulárního jazyka L.

3 Konečné automaty

3.1 Základní pojmy

Deterministický konečný automat M je pětice $M = (Q, T, \delta, q_0, F)$, kde
Q je konečná množina vnitřních stavů,
T je konečná množina vstupních symbolů,
δ je zobrazení z $Q \times T$ do Q,
$q_0 \in Q$ je počáteční stav,
$F \subset Q$ je množina koncových stavů.

Konečný automat je úplně určený, když $\delta(q, a)$ je definováno pro všechny dvojice $(q, a) \in Q \times T$.

Konečný automat je nedeterministický, když δ je zobrazení z $Q \times T$ do množiny podmnožin Q.

Nedeterministický konečný automat s ε–přechody je takový, že δ je zobrazení z $Q \times (T \cup \{\varepsilon\})$ do množiny podmnožin Q. U nedeterministického konečného automatu s více počátečními stavy je místo stavu q_0 množina počátečních stavů $I \subset Q$.

Konfigurace konečného automatu je dvojice (q, w), kde
q je vnitřní stav,
w je dosud nepřečtená část vstupního řetězce.

Počáteční konfigurace konečného automatu $M = (Q, T, \delta, q_0, F)$ je dvojice (q_0, w), kde $w \in T^*$ je vstupní slovo. Koncová konfigurace je dvojice (q, ε), kde $q \in F$.

Přechodem v konečném automatu M nazveme relaci mezi dvěma konfiguracemi a zapisujeme ji takto:
$(q, aw) \vdash (p, w)$, kde $q, p \in Q, a \ \in T, w \in T^*$.

Tento přechod je možný tehdy, když $\delta(q, a)$ obsahuje p. Pomocí symbolů $\vdash^k$, $\vdash^+$, $\vdash^*$ označujeme k-tou mocninu, tranzitivní a konečně reflexivní a tranzitivní uzávěr relace $\vdash$.

Jazyk přijímaný konečným automatem $M = (Q, T, \delta, q_0, F)$ je definován takto:
$L(M) = \{w : w \in T^*, (q_0, w) \vdash^* (q, \varepsilon), q \in F\}$.

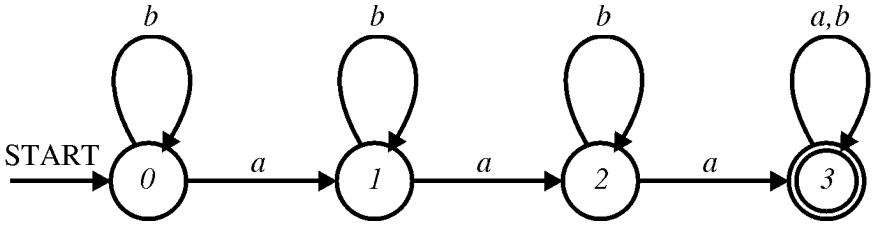
Konečné automaty M a M' nazýváme ekvivalentní, jestliže $L(M) = L(M')$.

3.2 Konstrukce deterministických konečných automatů

Příklad 3.1
Sestrojme konečný automat M nad abecedou $= \{a, b\}$ takový, který přijímá jazyk $L(M) = \{x : x \in \{a, b\}^*, x$ obsahuje alespoň tři symboly $a\}$.

$M = (\{0, 1, 2, 3\}, \{a, b\}, \delta, 0, \{3\})$, kde zobrazení δ je definováno tabulkou přechodů a přechodovým diagramem na obr. 3.1.

δ	a	b
0	1	0
1	2	1
2	3	2
3	3	3

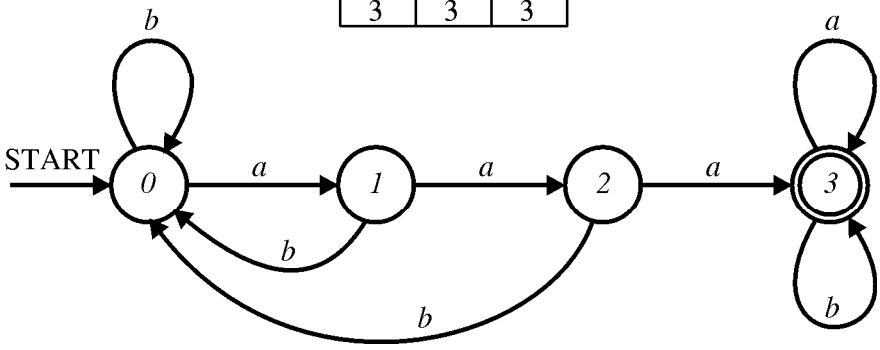


Obrázek 3.1: Tabulka přechodů a přechodový diagram konečného automatu z příkladu 3.1

Příklad 3.2
Sestrojme konečný automat M nad abecedou $T = \{a, b\}$ takový, který přijímá jazyk $L(M) = \{x : x \in \{a, b\}^*, x \text{ obsahuje řetězec } aaa\} = \{xaaay : x, y \in \{a, b\}^*\}$.

$M = (\{0, 1, 2, 3\}, \{a, b\}, 0, \{3\})$, kde zobrazení δ je definováno tabulkou přechodů a přechodovým diagramem na obr. 3.2.

δ	a	b
0	1	0
1	2	0
2	3	0
3	3	3

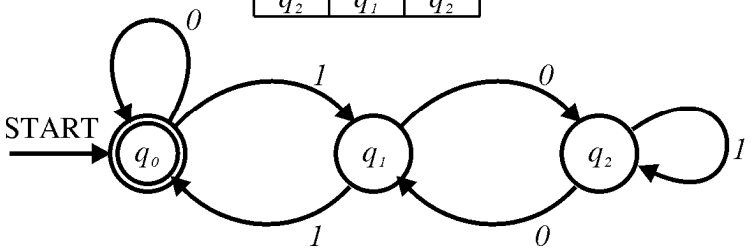


Obrázek 3.2: Tabulka přechodů a přechodový diagram konečného automatu z příkladu 3.2

Příklad 3.3
Sestrojme konečný automat M nad abecedou $T = \{0, 1\}$ takový, který přijímá jazyk $L(M) = \{x : x \in \{0, 1\}^*, x$ je v binární soustavě násobek tří$\}$.

$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_0\})$, kde zobrazení δ je definováno tabulkou přechodů a přechodovým diagramem na obr. 3.3.

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2



Obrázek 3.3: Tabulka přechodů a přechodový diagram konečného automatu z příkladu 3.3

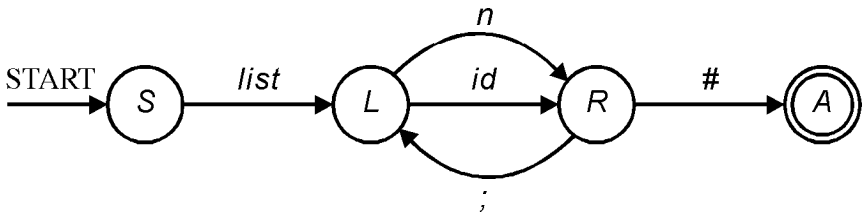
Ukažme několik posloupností přechodů tohoto automatu:
$(q_0, 0) \vdash (q_0, \varepsilon)$,
$(q_0, 11) \vdash (q_1, 1) \vdash (q_0, \varepsilon)$,
$(q_0, 100) \vdash (q_1, 00) \vdash (q_2, 0) \vdash (q_1, \varepsilon)$, (tento řetězec nebyl přijat, stav q_1 není koncový),
$(q_0, 110) \vdash (q_1, 10) \vdash (q_0, 0) \vdash (q_0, \varepsilon)$,
$(q_0, 1001) \vdash (q_1, 001) \vdash (q_2, 01) \vdash (q_1, 1) \vdash (q_0, \varepsilon)$.

Příklad 3.4
Sestrojme konečný automat přijímající jazyk, který obsahuje řetězce tvaru:
$list\ id; n; id; id; \ldots ; n; n; id\#$
Každý řetězec začíná slovem $list$ a končí symbolem #. Mezi těmito dvěma symboly je seznam identifikátorů (id) a čísel (n), v němž každé dva prvky jsou odděleny středníkem. Identifikátory a čísla považujeme za vstupní symboly.
Konečný automat $M = (\{S, L, R, A\}, \{list, id, n, \#, ;\}, \delta, S, \{A\})$ má zobrazení δ definováno tabulkou na obr. 3.4. Zde je i jeho přechodový diagram.

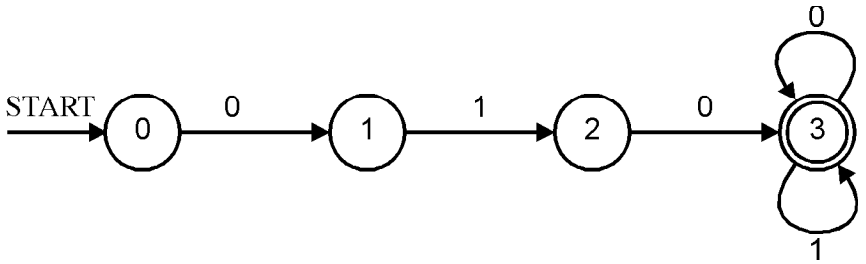
Příklad 3.5
Sestrojme konečný automat, který přijímá řetězce nad abecedou $\{0, 1\}$ začínající řetězcem 010. Výsledný automat má přechodový diagram na obr. 3.5.

Automat je zkonstruován tak, že je v něm cesta pro řetězec 010 na začátku a potom smyčka pro libovolný řetězec.

δ	*list*	*id*	*n*	;	#
S	*L*				
L		*R*	*R*		
R				*L*	*A*
A					



Obrázek 3.4: Tabulka přechodů a přechodový diagram konečného automatu z příkladu 3.4



Obrázek 3.5: Přechodový diagram deterministického konečného automatu z příkladu 3.5

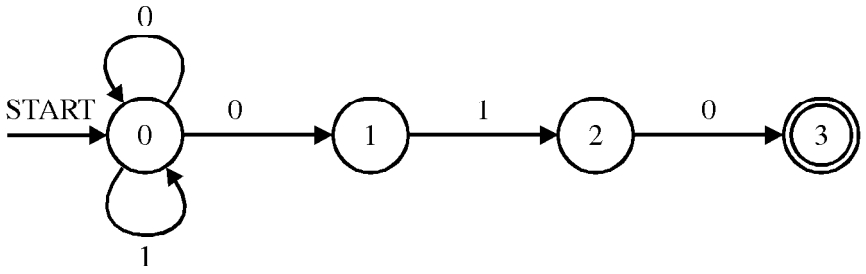
3.3 Konstrukce nedeterministických konečných automatů

Někdy je vhodné sestrojit nejdříve nedeterministický konečný automat a potom jej převést na deterministický.

Příklad 3.6
Sestrojme konečný automat, který přijímá řetězce nad abecedou $\{0, 1\}$ takové, které končí řetězcem 010. Tento automat má přechodový diagram na obr. 3.6. Získaný automat je zkonstruován tak, že je v něm na začátku smyčka pro libovolný řetězec a do koncového stavu vede cesta pro řetězec 010. V tomto případě je ovšem automat nedeterministický.

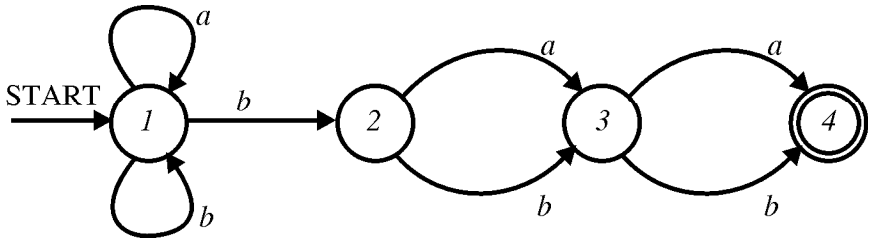
Příklad 3.7
Sestrojme nedeterministický konečný automat M, který přijímá jazyk $L(M) = \{x : x \in \{a, b\}^*$, třetí symbol od konce je symbol $b\}$.

$M = (\{1, 2, 3, 4\}, \{a, b\}, \delta, 1, \{4\})$. Tabulka přechodů a přechodový diagram automatu M je na obr. 3.7.



Obrázek 3.6: Přechodový diagram nedeterministického konečného automatu z příkladu 3.6

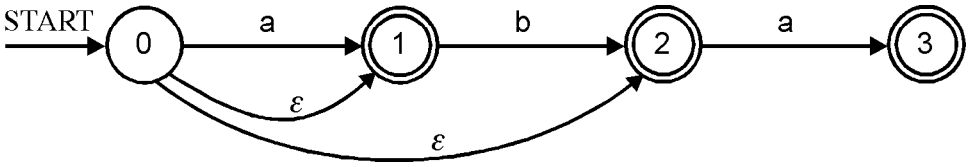
δ	a	b
1	1	1,2
2	3	3
3	4	4
4		



Obrázek 3.7: Přechodový diagram a tabulka přechodů z příkladu 3.8

3.4 Konečné automaty s ε–přechody a více počátečními stavy

Příklad 3.8
Je dán konečný automat s ε–přechody $M = (\{0,1,2,3\}, \{a,b\}, \delta, 0, \{1,2,3\})$, kde zobrazení δ je definováno přechodovým diagramem na obr. 3.8. Tento automat přijímá všechny faktory řetězce aba.



Obrázek 3.8: Konečný automat s ε–přechody z příkladu 3.8

Jeho tabulka přechodů má tvar:

δ	a	b	ε
0	1		1, 2
1		2	
2	3		
3			

Sestrojme ekvivalentní deterministický konečný automat M'. Přitom použijeme Algoritmus 2.41 a Algoritmus 2.51 [JPR03].

$\varepsilon - CLOSURE(0) = \{0, 1, 2\}$,
$\varepsilon - CLOSURE(1) = \{1\}$,
$\varepsilon - CLOSURE(2) = \{2\}$,
$\varepsilon - CLOSURE(3) = \{3\}$.

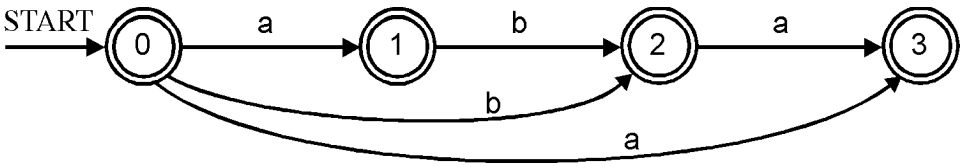
$$\begin{aligned}\delta'(0,a) &= \delta(0,a) \cup \delta(1,a) \cup \delta(2,a)\\ &= \{1\} \quad \cup \emptyset \quad \cup \{3\}\\ &= \{1,3\},\end{aligned}$$

$$\begin{aligned}\delta'(0,b) &= \delta(0,b) \cup \delta(1,b) \cup \delta(2,b)\\ &= \emptyset \quad \cup \{2\} \quad \cup \emptyset\\ &= \{2\}.\end{aligned}$$

Výsledkem této konstrukce je nedeterministický konečný automat M' (viz Algoritmus 2.41 [JPR03]), jehož tabulka přechodů má tvar:

δ'	a	b
0	1,3	2
1		2
2	3	
3		

Přechodový diagram tohoto automatu je na obr. 3.9.

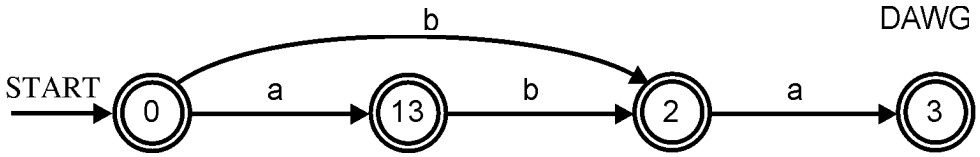


Obrázek 3.9: Nedeterministický konečný automat z příkladu 3.8

Výsledný deterministický konečný automat je
$M'' = (\{0, 13, 2, 3\}, \{a, b\}, \delta'', 0, \{0, 13, 2, 3\})$, jehož tabulka přechodů má tvar:

δ"	a	b
0	13	2
13		2
2	3	
3		

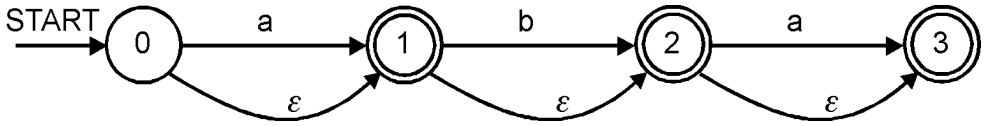
Přechodový diagram automatu M'' je na obr. 3.10.



Obrázek 3.10: Deterministický konečný automat z příkladu 3.8

Automat M'' se nazývá faktorový automat. Tento automat přijímá všechny podřetězce (faktory) řetězce *aba*. Jiný název pro tento automat je $DAWG$ (Directed Acyclic Word Graph).

Příklad 3.9
Je dán konečný automat s ε–přechody $M = (\{0,1,2,3\}, \{a,b\}, \delta, 0, \{0,1,2,3\})$, kde zobrazení δ je definováno přechodovým diagramem na obr. 3.11. Automat přijímá všechny podposloupnosti řetězce *aba*.



Obrázek 3.11: Konečný automat s ε–přechody z příkladu 3.9

Jeho tabulka má tvar:

δ	a	b	ε
0	1		1
1		2	2
2	3		3
3			

Sestrojme ekvivalentní deterministický konečný automat. Přitom použijeme Algoritmy 2.41 a 2.51 [JPR03].

$\varepsilon - CLOSURE(0) = \{0,1,2,3\}$,
$\varepsilon - CLOSURE(1) = \{1,2,3\}$,
$\varepsilon - CLOSURE(2) = \{2,3\}$,
$\varepsilon - CLOSURE(3) = \{3\}$.

$$
\begin{aligned}
\delta'(0,a) &= \delta(0,a) \cup \delta(1,a) \cup \delta(2,a) \cup \delta(3,a) \\
&= \{1\} \quad \cup \emptyset \quad \cup \{3\} \quad \cup \emptyset \\
&= \{1,3\},
\end{aligned}
$$

$$
\begin{aligned}
\delta'(0,b) &= \delta(0,b) \cup \delta(1,b) \cup \delta(2,b) \cup \delta(3,b) \\
&= \emptyset \quad \cup \{2\} \quad \cup \emptyset \quad \cup \emptyset \\
&= \{2\},
\end{aligned}
$$

$$
\begin{aligned}
\delta'(1,a) &= \delta(1,a) \cup \delta(2,a) \cup \delta(3,a) \\
&= \emptyset \quad \cup \{3\} \quad \cup \emptyset \\
&= \{3\},
\end{aligned}
$$

$$
\begin{aligned}
\delta'(1,b) &= \delta(1,b) \cup \delta(2,b) \cup \delta(3,b) \\
&= \{2\} \quad \cup \emptyset \quad \cup \emptyset \\
&= \{2\},
\end{aligned}
$$

$$
\begin{aligned}
\delta'(2,a) &= \delta(2,a) \cup \delta(3,a) \\
&= \{3\} \quad \cup \emptyset \\
&= \{3\},
\end{aligned}
$$

$$
\begin{aligned}
\delta'(2,b) &= \delta(2,b) \cup \delta(3,b) \\
&= \{\emptyset\},
\end{aligned}
$$

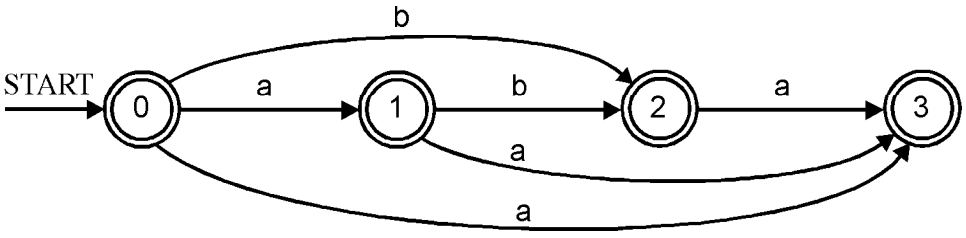
$$\delta'(3,a) = \emptyset,$$

$$\delta'(3,b) = \emptyset.$$

Výsledkem této konstrukce je nedeterministický konečný automat M', jehož tabulka přechodů má tvar:

δ'	a	b
0	1,3	2
1	3	2
2	3	
3		

Přechodový diagram tohoto automatu je na obr. 3.12.

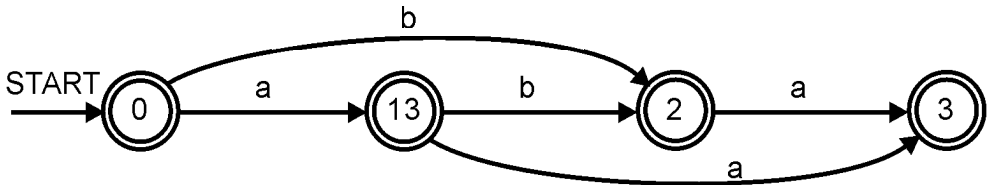


Obrázek 3.12: Nedeterministický konečný automat z příkladu 3.9

Výsledný deterministický konečný automat je
$M'' = (\{0,13,2,3\}, \{a,b\}, \delta'', 0, \{0,13,2,3\})$, jehož tabulka přechodů má tvar:

δ"	a	b
0	13	2
13	3	2
2	3	
3		

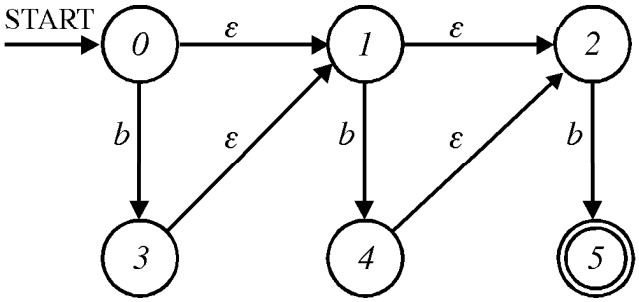
Přechodový diagram automatu M'' je na obr. 3.13.



Obrázek 3.13: Deterministický konečný automat z příkladu 3.9

Automat M'' přijímá všechny podposloupnosti řetězce *aba*. Nazývá se také *DASG* (Directed Acyclic Subsequence Graph).

Příklad 3.10
Je dán konečný automat, jehož přechodový diagram je na obr. 3.14. Sestrojme

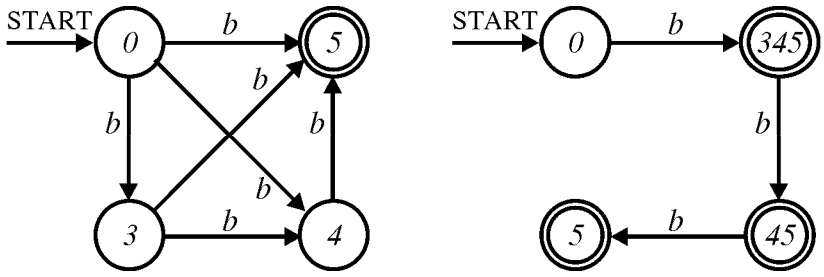


Obrázek 3.14: Konečný automat s ε–přechody z příkladu 3.10

ekvivalentní deterministický konečný automat bez ε–přechodů.
Nejdříve určíme (viz. Algoritmus 2.41 [JPR03]):

$\varepsilon - CLOSURE(0) = \{0, 1, 2\}$,
$\varepsilon - CLOSURE(1) = \{1, 2\}$,
$\varepsilon - CLOSURE(2) = \{2\}$,
$\varepsilon - CLOSURE(3) = \{3, 1, 2\}$.
$\varepsilon - CLOSURE(4) = \{4, 2\}$.
$\varepsilon - CLOSURE(5) = \{5\}$.

Výsledný konečný automat má přechodový diagram na obr. 3.15.



Obrázek 3.15: Nedeterministický a deterministický konečný automat z příkladu 3.10

Příklad 3.11
Sestrojme konečný automat, který přijímá jazyk

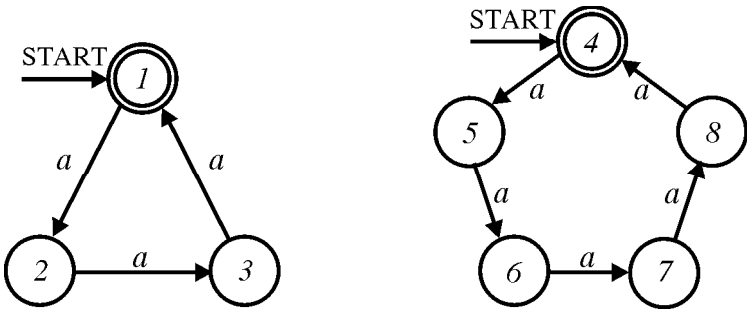
$L = \{x : x \in \{a\}^*,\ |x| \text{ je dělitelné třemi nebo pěti}\}$.

Nejdříve sestrojíme dva automaty, které přijímají jazyky:

$L_1 = \{x_1 : x_1 \in \{a\}^*,\ |x_1| \text{ je dělitelné třemi}\}$ a

$L_2 = \{x_2 : x_2 \in \{a\}^*,\ |x_2| \text{ je dělitelné pěti}\}$.

Přechodové diagramy těchto automatů jsou na obr. 3.16. Výsledný automat

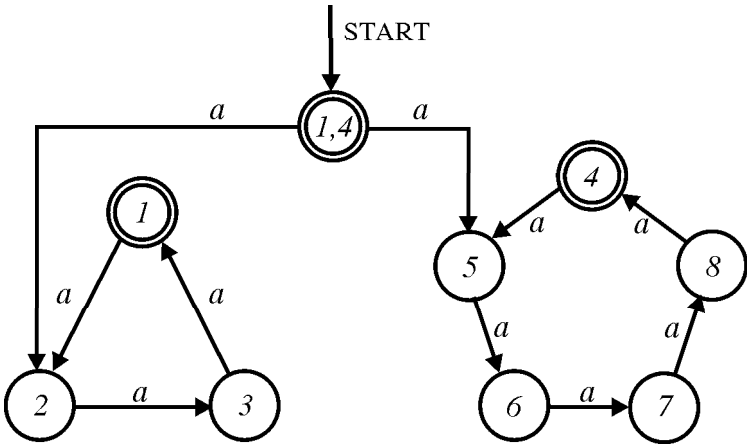


Obrázek 3.16: Konečné automaty z příkladu 3.11 přijímající jazyky L_1 a L_2

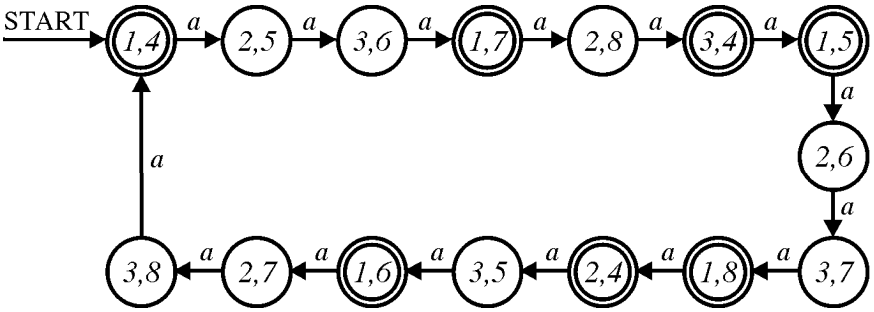
sestrojíme tak, že to bude automat, který má množinu počátečních stavů $I = \{1, 4\}$.

Sestrojíme-li automat s jedním počátečním stavem (Algoritmus 2.46 [JPR03]), pak dostaneme nedeterministický konečný automat, jehož přechodový diagram je na obr. 3.17. Po determinizaci tohoto konečného automatu dostaneme automat s přechodovým diagramem na obr. 3.18.

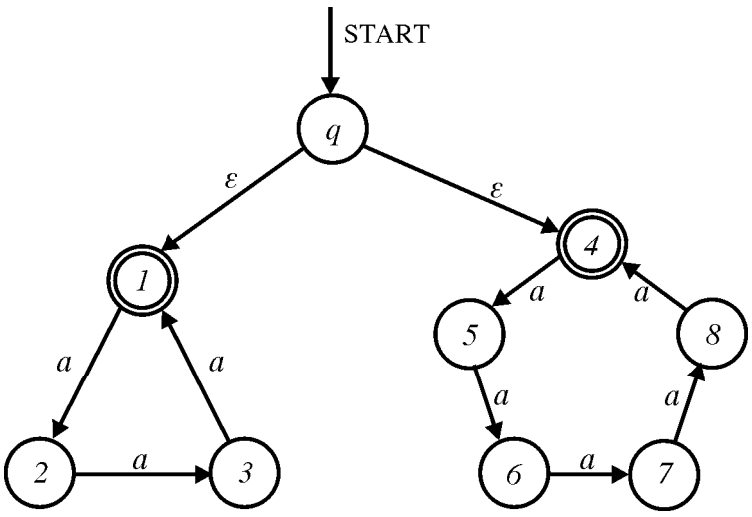
Podobného výsledku dosáhneme, když vytvoříme nový stav q a přidáme ε přechody $\delta(q, \varepsilon) = \{1, 4\}$. Přechodový diagram tohoto automatu je na obr 3.19. Po odstranění ε–přechodů získáme konečný automat (Algoritmus 2.41 [JPR03]), jehož přechodový diagram je na obr 3.20. Po jeho determinizaci dostaneme konečný automat s přechodovým diagramem na obr. 3.21. Tento automat je ekvivalentní s automatem na obr. 3.18.



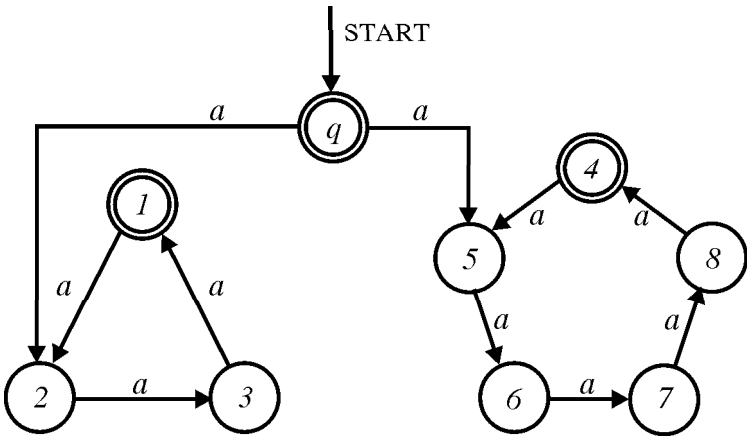
Obrázek 3.17: Přechodový diagram konečného automatu s jedním počátečním stavem z příkladu 3.11



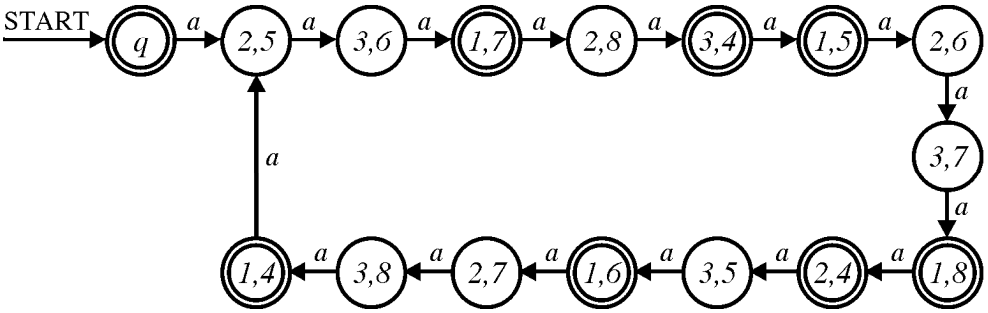
Obrázek 3.18: Přechodový diagram deterministického konečného automatu z příkladu 3.11



Obrázek 3.19: Přechodový diagram automatu s ε–přechody z příkladu 3.11



Obrázek 3.20: Přechodový diagram konečného automatu po odstranění ε–přechodů z příkladu 3.11



Obrázek 3.21: Přechodový diagram konečného automatu po odstranění ε–přechodů z příkladu 3.11

3.5 Konstrukce deterministických konečných automatů pro zadané nedeterministické konečné automaty

V celém tomto odstavci je pro konstrukci deterministických konečných automatů použit Algoritmus 2.51 [JPR03].

Příklad 3.12
Je dán nedeterministický konečný automat $M = (\{z, f\}, \{a, b\}, \delta, z, \{f\})$, kde zobrazení δ je definováno tabulkou:

δ	a	b
z	$\{z, f\}$	$\emptyset$
f	$\emptyset$	$\{f\}$

Tomuto nedeterministickému automatu odpovídá úplně určený deterministický konečný automat $M = (\{[z], [f], [z, f]\}, \{a, b\}, \delta', [z], \{[f], [z, f]\})$, kde δ' je definováno tabulkou:

δ'	a	b
$[z]$	$[z, f]$	$\emptyset$
$[z, f]$	$[z, f]$	$[f]$
$[f]$	$\emptyset$	$[f]$
$\emptyset$	$\emptyset$	$\emptyset$

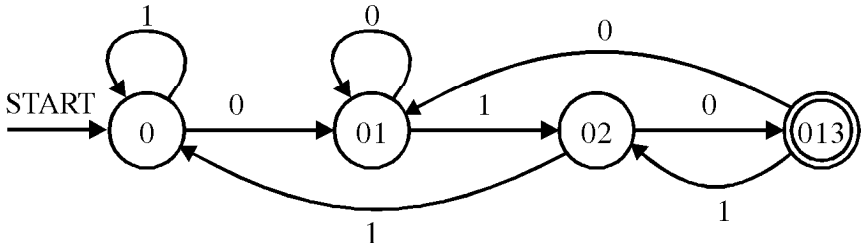
Příklad 3.13
Je dán nedeterministický konečný automat $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_0, \{q_2, q_4\})$, zobrazení δ je definováno tabulkou:

δ	0	1
q_0	q_0, q_3	q_0, q_1
q_1		q_2
q_2	q_2	q_2
q_3	q_4	
q_4	q_4	q_4

Sestrojme deterministický konečný automat M' tak, aby $L(M) = L(M')$. $M' = (\{[q_0], [q_0, q_3], [q_0, q_1], [q_0, q_3, q_4], [q_0, q_1, q_2], [q_0, q_1, q_4], [q_0, q_2, q_3], [q_0, q_1, q_2, q_4], [q_0, q_2, q_3, q_4]\}, \{0, 1\}, \delta', [q_0], \{[q_0, q_3, q_4], [q_0, q_1, q_2], [q_0, q_1, q_4], [q_0, q_2, q_3], [q_0, q_1, q_2, q_4], [q_0, q_2, q_3, q_4]\})$, kde δ' je definováno tabulkou:

δ'	0	1
$[q_0]$	$[q_0, q_3]$	$[q_0, q_1]$
$[q_0, q_3]$	$[q_0, q_3, q_4]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_3]$	$[q_0, q_1, q_2]$
$[q_0, q_3, q_4]$	$[q_0, q_3, q_4]$	$[q_0, q_1, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_2, q_3]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_4]$	$[q_0, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$
$[q_0, q_2, q_3]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2, q_4]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$
$[q_0, q_2, q_3, q_4]$	$[q_0, q_2, q_3, q_4]$	$[q_0, q_1, q_2, q_4]$

Příklad 3.14
Sestrojme deterministický konečný automat, který je ekvivalentní nedeterministickému konečnému automatu z příkladu 3.6. Jeho přechodový diagram je na obr. 3.22.



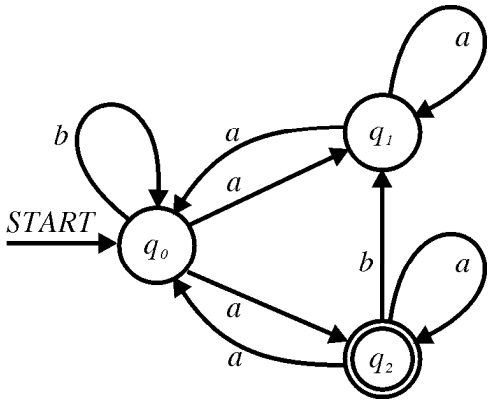
Obrázek 3.22: Přechodový diagram deterministického konečného automatu z příkladu 3.14

Příklad 3.15
Je dán nedeterministický konečný automat $M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\}\}$, kde

$\delta(q_0, a) = \{q_1, q_2\}$,
$\delta(q_1, a) = \{q_0, q_1\}$,
$\delta(q_2, a) = \{q_0, q_2\}$,
$\delta(q_0, b) = \{q_0\}$,
$\delta(q_2, b) = \{q_1\}$.

Jeho přechodový diagram je na obr. 3.23.
Sestrojme deterministický konečný automat M' takový, aby platilo $L(M) = L(M')$.
$M' = (\{[q_0], [q_1], [q_0, q_1], [q_1, q_2], [q_0, q_1, q_2], \}, \{a, b\}, \delta', [q_0], \{[q_1, q_2], [q_0, q_1, q_2]\})$, kde zobrazení δ je dáno tabulkou přechodů:

δ'	a	b
$[q_0]$	$[q_1, q_2]$	$[q_0]$
$[q_1]$	$[q_0, q_1]$	$\emptyset$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0]$
$[q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$

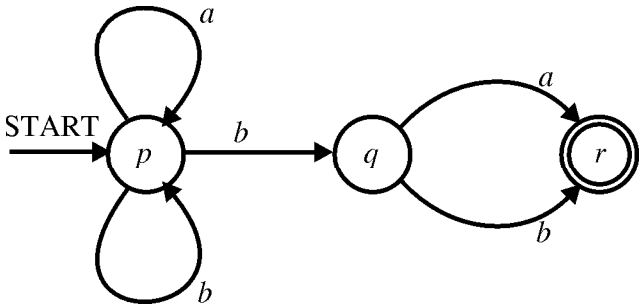


Obrázek 3.23: Přechodový diagram nedeterministického konečného automatu z příkladu 3.15

Příklad 3.16
Sestrojme deterministický konečný automat M, který přijímá jazyk $L(M) = \{x : x \in \{a, b\}^*$, druhý symbol zprava je symbol $b\}$.

Nejdříve sestrojíme nedeterministický konečný automat M', který přijímá jazyk $L(M)$. $M' = (\{p, q, r\}, \{a, b\}, \delta', p, \{r\})$. Jeho tabulka přechodů a přechodový diagram je na obr. 3.24.

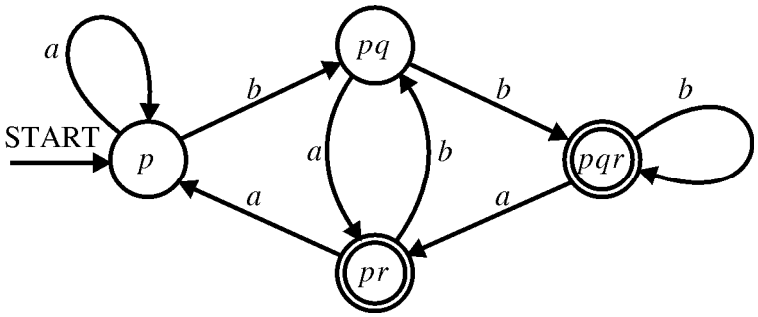
Deterministický konečný automat M sestrojený pomocí Algoritmu 2.51 [JPR03] má tabulku přechodů a přechodový diagram na obr. 3.25.



d′	a	b
p	p	p,q
q	r	r
r		

Obrázek 3.24: Přechodový diagram a tabulka přechodů nedeterministického konečného automatu z příkladu 3.16

p	a	b
pq	p	pq
pr	pr	pqr
pqr	p	pq
	pr	pqr



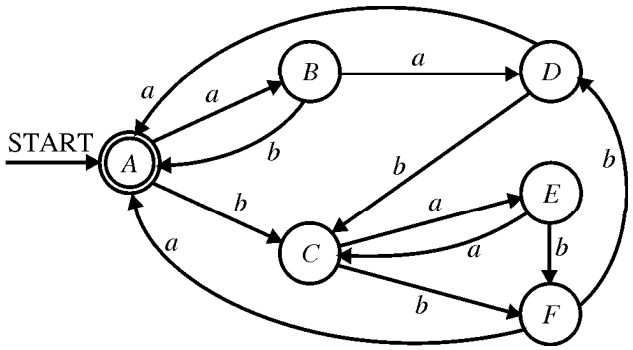
Obrázek 3.25: Tabulka přechodů a přechodový diagram deterministického konečného automatu z příkladu 3.16

3.6 Minimalizace množiny stavů konečného automatu

Při návrhu automatu se nám většinou podaří do jeho popisu zavést tzv. ekvivalentní stavy. V této kapitole si proto ukážeme způsob, jak v automatu nahradit ekvivalentní stavy tak, aby výsledný automat byl ekvivalentní s automatem původním.

Příklad 3.17
Je dán konečný automat $M = (\{A, B, C, D, E, F\}, \{a, b\}, \delta, A, \{A\})$, kde zobrazení δ je uvedeno na obr. 3.26 ve formě přechodového diagramu a tabulky přechodů. Při podrobnějším rozboru lze zjistit, že stavy C a E můžeme nahradit jedním stavem CE s tím, že příslušným způsobem upravíme zobrazení δ.

δ	a	b
A	B	C
B	D	A
C	E	F
D	A	C
E	C	F
F	A	D



Obrázek 3.26: Přechodový diagram a tabulka přechodů konečného automatu z příkladu 3.17

Definice 3.18
Nechť konečný automat $M = (Q, T, \delta, q_0, F)$. Stavy automatu $q_i, q_j \in Q$ jsou *ekvivalentní* tehdy, když všechny možné posloupnosti přechodů začínající ve stavech q_i a q_j a pro každý řetězec vstupních symbolů vedou v obou případech buď do koncového stavu nebo do stavu, který není koncový. Přesněji to znamená: Pro každý řetězec $x \in T^*$, který může být automatem M přečten, platí:

$(q_i, x) \vdash \ldots \vdash (q_1, \varepsilon)$,
$(q_j, x) \vdash \ldots \vdash (q_2, \varepsilon)$ a buď $q_1, q_2 \in F$ nebo $q_1, q_2 \notin F$.

Pokud stavy q_i a q_j nejsou ekvivalentní, pak jsou *rozlišitelné*.

Metoda minimalizace spočívá v nalezení tříd těchto ekvivalentních stavů a jejich nahrazení stavem jediným. Metoda, kterou zde uvedeme je použitelná pouze pro úplně určené automaty. To ovšem neomezuje použitelnost metody na automaty neúplně určené, neboť vždy lze jednoduše zkonstruovat k neúplně určenému automatu automat úplně určený tak, aby oba automaty byly ekvivalentní (viz. Algoritmus 2.22 [JPR03]).

Metoda postupuje „odzadu“. Nejdříve rozliší stavy do dvou tříd podle toho, zda jsou či nejsou koncové. V dalším kroku se rozliší stavy uvnitř každé třídy podle toho, zda pro konkrétní symbol existuje přechod do koncového stavu či nikoliv. To znamená, podle toho, do jaké třídy vede přechod pro konkrétní symbol. V tabulce přechodů se toto dělení provádí tak, že pro každý stav a symbol se určí, do které třídy patří cílový stav přechodu. Stavy se shodnými hodnotami v řádku tabulky přechodů zůstávají ve stejné třídě, stavy s různými hodnotami je nutno rozdělit do separátních tříd. Tento proces se opakuje tak dlouho, dokud dochází v každé iteraci k nárůstu počtu tříd.

Časová složitost popsaného algoritmu je v nejhorším případě $O(n^2)$, v nejlepším případě $O(n)$, kde n je počet stavů původního automatu.

Algoritmus 3.19
Minimalizace deterministického úplně určeného konečného automatu.
Vstup: Deterministický úplně určený konečný automat $M = (Q, T, \delta, q_0, F)$.
Výstup: Konečný automat M_{min} ekvivalentní s automatem M, který má nejmenší počet stavů.
Metoda:

1. Vytvoříme dvě třídy vnitřních stavů. $Q_{1,1} = F$ a $Q_{1,2} = Q \setminus F$. Třída $Q_{1,1}$ bude tedy obsahovat všechny koncové stavy automatu M, třída $Q_{1,2}$ všechny ostatní stavy automatu M. Přiřadíme do k dvojku ($k := 2$).
2. Vytvoříme tabulku přechodů pro všechny stavy $q \in Q$. Místo cílových stavů budeme ale v tabulce uvádět třídu cílového stavu.
3. Každou třídu $Q_{k,x}$ rozdělíme do tříd $Q_{k+1,y1}$ až $Q_{k+1,yn}$ tak, aby stavy v jedné třídě měly shodné řádky v přechodové tabulce.
4. $k := k + 1$.
5. Kroky 2, 3 a 4 opakujeme tak dlouho, dokud další iterace neobsahuje stejné třídy jako iterace předchozí, t.j. $Q_{k,x} = Q_{k-1,x}$ pro všechna x.
6. Výsledný automat bude obsahovat pro každou třídu jeden vnitřní stav.

Příklad 3.20
Úplně určený deterministický konečný automat M je zadán v příkladu 3.17. Sestrojme k němu ekvivalentní deterministický automat s minimálním počtem stavů. Postup bude tento:

Krok 1: Rozdělíme stavy automatu do dvou tříd podle toho, zda jsou či nejsou koncové. Výsledné dělení do tříd 1 a 2 je uvedeno v posledním sloupci tabulky na obr. 3.27. Třída koncových stavů 1 obsahuje jediný koncový stav automatu: $Q_{1,1} = \{A\}$. Zbylé stavy jsou všechny obsaženy v třídě 2: $Q_{1,2} = \{B, C, D, E, F\}$.

δ	a	b	$Q_{1,x}$
A	B	C	1
B	D	A	2
C	E	F	2
D	A	C	2
E	C	F	2
F	A	D	2

Obrázek 3.27: Tabulky přechodů po prvním kroku minimalizace konečného automatu z příkladu 3.20

Krok 2: Tabulku přechodů pro automat M překreslíme tak, že cílové stavy nahradíme číslem třídy, do které cílový stav patří. Například ze stavu A přechází automat pro symbol a do stavu B, který je obsažen v třídě $Q_{1,2}$. Pro kombinaci (A, a) proto bude v tabulce uvedena hodnota 2. Výsledek je vidět v tabulce na obr. 3.28 a) ve sloupcích pro δ, a, b.

$Q_{1,x}$	δ	a	b	$Q_{2,x}$
1	A	2	2	1
2	B	2	1	2
2	C	2	2	3
2	D	1	2	4
2	E	2	2	3
2	F	1	2	4

a)

$Q_{2,x}$	δ	a	b	$Q_{3,x}$
1	A	2	3	1
2	B	4	1	2
3	C	3	4	3
3	E	3	4	3
4	D	1	3	4
4	F	1	4	5

b)

Obrázek 3.28: Tabulky přechodů při minimalizaci konečného automatu z příkladu 3.20, kroky 2, 3 a 4

Krok 3: $k = 2$. Podle přechodů pro symboly a a b vytvoříme novou sadu tříd. Třídu 1 již nelze dál dělit a proto zůstane zachována: $Q_{2,1} = \{A\}$. Třída 2 se rozpadne na tři různé třídy podle toho, jak vypadají přechody pro konkrétní stavy. Je vidět, že například stavy B a C se liší v přechodu pro symbol b, proto je musíme separovat do dvou tříd, v tomto případě konkrétně do tříd $Q_{2,2}$ a $Q_{2,3}$. Tímto způsobem rozdělíme třídu $Q_{1,2}$ na třídy $Q_{2,2} = \{B\}, Q_{2,3} = \{C, E\}, Q_{2,4} = \{D, F\}$. Výsledné dělení do tříd je vyznačeno v posledním sloupci tabulky na obr. 3.28a).

Krok 4: Opakujeme kroky 2 a 3 pro třídy $Q_{2,x}$. Aktualizovaná tabulka přechodů je uvedena v tabulce na obr. 3.28b). Vytvoříme další iteraci tříd $Q_{3,x}$. Jak je z tabulky patrné, není nutné dělit třídy $Q_{2,1}, Q_{2,2}$ a $Q_{2,3}$. Proto $Q_{3,1} = Q_{2,1}, Q_{3,2} = Q_{2,2}$ a $Q_{3,3} = Q_{2,3}$. Třídu $Q_{2,4}$ však rozdělit musíme, protože stavy D a F mají v tabulce odlišné hodnoty v řádcích. Vytvoříme proto třídy $Q_{3,4} = \{D\}$ $Q_{3,5} = \{F\}$. Výsledné dělení je opět znázorněno v posledním sloupci tabulky na obr. 3.28b). Poté znovu opakujeme kroky 2 a 3, tentokrát ale pro třídy $Q_{3,x}$. Dostaneme tabulku na obr. 3.29a) a z ní zjistíme, že již není nutné žádnou třídu dále dělit. Získali jsme 5 tříd vzájemně ekvivalentních stavů.

$Q_{3,x}$	δ	a	b
1	A	2	3
2	B	4	1
3	C	3	5
3	E	3	5
4	D	1	3
5	F	1	4

$a)$

δ	a	b
1	2	3
2	4	1
3	3	5
4	1	3
5	1	4

$b)$

Obrázek 3.29: Tabulky přechodů při minimalizaci konečného automatu z příkladu 3.20, kroky 4, 5

Krok 5: Výsledný automat M_{min} sestrojíme takto: Pro každou třídu $Q_{3,x}$ vytvoříme jeden stav automatu. Máme pět tříd, automat M_{min} bude tedy obsahovat pět stavů. Třída $Q_{3,1}$ obsahovala stav A, který byl počátečním stavem automatu M a zároveň byl koncovým stavem v automatu M. V novém automatu M_{min} proto bude stav 1 odpovídající třídě $Q_{3,1}$ stavem počátečním a zároveň stavem koncovým. Tabulka přechodů nového automatu M_{min} odpovídá přechodům v tabulce na obr. 3.29a) a je uvedena v tabulce na obr. 3.29b).
Z tohoto výsledku je zřejmé, že stavy C a E jsou ekvivalentní (srovnej s příkladem 3.17)

Příklad 3.21
Úplně určený deterministický konečný automat $M = (\{A, B, C, D, E, F, G\}, \{a, b\}, \delta, A, \{D\})$, kde zobrazení δ je zadáno tabulkou přechodů zobrazenou v tabulce na obr. 3.30a). Sestrojme k němu ekvivalentní deterministický automat s minimálním počtem stavů.

δ	a	b	$Q_{1,x}$
A	C	B	1
B	G	D	1
C	D	E	1
D	D	D	2
E	D	F	1
F	D	C	1
G	B	D	1

$a)$

$Q_{1,x}$	δ	a	b	$Q_{2,x}$
1	A	1	1	1
1	B	1	2	3
1	C	2	1	4
1	E	2	1	4
1	F	2	1	4
1	G	1	2	3
2	D	2	2	2

$b)$

Obrázek 3.30: Tabulky přechodů při minimalizaci konečného automatu z příkladu 3.21

Postup dělení na třídy je zachycen v tabulkách na obr. 3.30a),b) a 3.31a). Jak je patrné z tabulky na obr. 3.31a), stavy B a G jsou ekvivalentní a proto je možné je nahradit stavem jediným a rovněž tak stavy C, E a F jsou ekvivalentní a taktéž je možné je nahradit jediným stavem. Výsledný automat M_{min}, který představuje minimální formu zadaného vstupního automatu M má tabulku přechodů uvedenou v tabulce na obr. 3.31b).

$Q_{2,x}$	δ	a	b
1	A	4	3
2	D	2	2
3	B	3	2
3	G	3	2
4	C	2	4
4	E	2	4
4	F	2	4

$a)$

δ	a	b
1	4	3
2	2	2
3	3	2
4	2	4

$b)$

Obrázek 3.31: Tabulky přechodů při minimalizaci konečného automatu z příkladu 3.21

3.7 Operace s konečnými automaty

Příklad 3.22
Jsou dány automaty M_1 a M_2 nad abecedou $\{0,1\}$. Automat M_1 přijímá řetězce začínající 010:

$L_1 = \{w : w \in \{0,1\}^*, w = 010v, v \in \{0,1\}^*\}$,

automat M_2 přijímá řetězce začínající 101:

$L_2 = \{w : w \in \{0,1\}^*, w = 101v, v \in \{0,1\}^*\}$.

Sestrojme konečný automat M, který přijímá jazyk $L_1 \cup L_2$. Nejprve sestrojíme automaty M_1 a M_2.

$M_1 = \{\{0,1,2,3,\emptyset\}, \{0,1\}, \delta_1, 0, \{3\}\}$,

$M_2 = \{\{0,1,2,3,\emptyset\}, \{0,1\}, \delta_2, 0, \{3\}\}$.

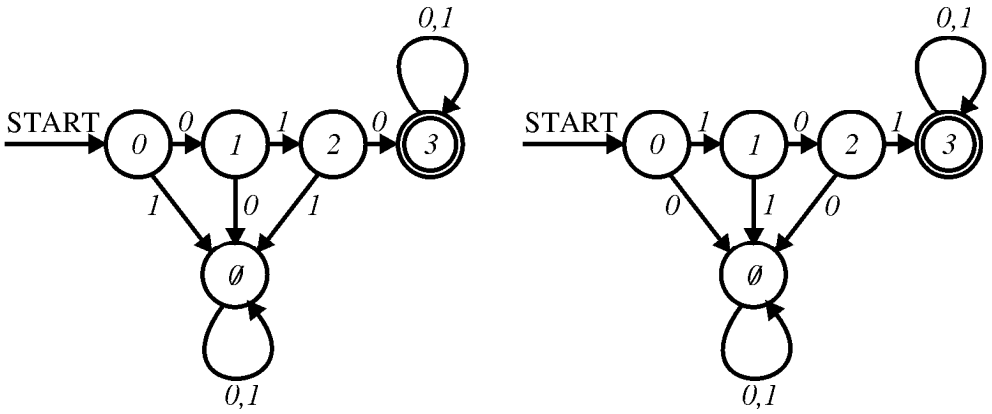
Jejich tabulky přechodů a přechodové diagramy jsou na obr. 3.32

M_1 :

δ_1	0	1
0	1	$\emptyset$
1	$\emptyset$	2
2	3	$\emptyset$
3	3	3
$\emptyset$	$\emptyset$	$\emptyset$

M_2 :

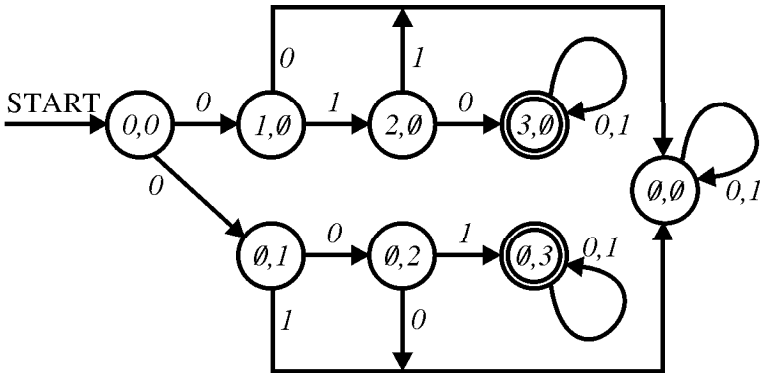
δ_2	0	1
0	$\emptyset$	1
1	2	$\emptyset$
2	$\emptyset$	3
3	3	3
$\emptyset$	$\emptyset$	$\emptyset$



Obrázek 3.32: Přechodové diagramy a tabulky přechodů automatů M_1 a M_2 z příkladu 3.22

Automat M získáme jako sjednocení automatů M_1 a M_2 podle Algoritmu 2.71 [JPR03].
$M = \{\{(0,0),(1,\emptyset),(\emptyset,1),(2,\emptyset),(\emptyset,2),(3,\emptyset),(\emptyset,3),(\emptyset,\emptyset)\}, \{0,1\}, \delta, (0,0), \{(3,\emptyset),(\emptyset,3)\}\}$ a jeho tabulka přechodů a přechodový diagram je na obr. 3.33. Stavy $(3,\emptyset)$ a $(\emptyset,3)$ jsou ekvivalentní.

δ	0	1
$(0,0)$	$(1,\emptyset)$	$(\emptyset,1)$
$(1,\emptyset)$	$(\emptyset,\emptyset)$	$(2,\emptyset)$
$(\emptyset,1)$	$(\emptyset,2)$	$(\emptyset,\emptyset)$
$(2,\emptyset)$	$(3,\emptyset)$	$(\emptyset,\emptyset)$
$(\emptyset,2)$	$(\emptyset,\emptyset)$	$(\emptyset,3)$
$(3,\emptyset)$	$(3,\emptyset)$	$(3,\emptyset)$
$(\emptyset,3)$	$(\emptyset,3)$	$(\emptyset,3)$
$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$

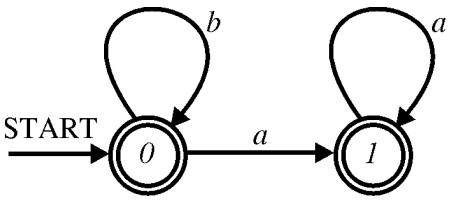


Obrázek 3.33: Tabulka přechodů a přechodový diagram konečného automatu z příkladu 3.22

Příklad 3.23
Jsou dány automaty M_1 a M_2 nad abecedou $\{a,b\}$. Automat M_1 přijímá řetězce tvaru b^*a^*, automat M_2 přijímá řetězce obsahující alespoň dva symboly a. Sestrojme automat M přijímající jazyk $L(M_1) \cap L(M_2)$.
Zde je třeba definovat automaty M_1 a M_2.
$M_1 = (\{0,1\},\{a,b\},\delta_1,0,\{1\})$,
kde zobrazení δ_1 je zadáno tabulkou přechodů a přechodovým diagramem na obr. 3.34.

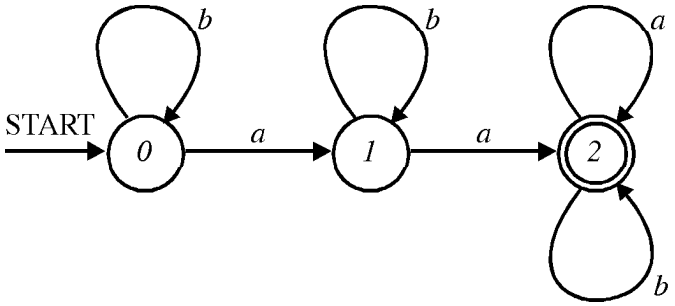
$M_2 = (\{0,1,2\},\{a,b\},\delta_2,0,\{2\})$,
kde zobrazení δ_2 je zadáno tabulkou přechodů a přechodovým diagramem na obr. 3.35.

$M = \{\{(0,0),(1,1),(1,2)\},\{a,b\},\delta,(0,0),\{(1,2)\}\}$.
Tabulka přechodů a přechodový diagram je na obr. 3.36.



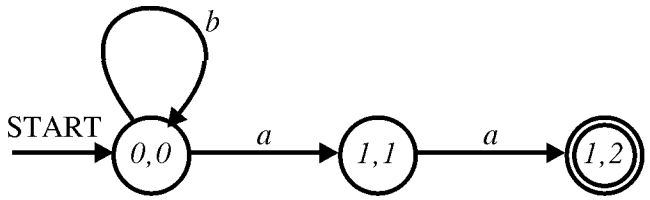
δ_1	a	b
0	1	0
1	1	

Obrázek 3.34: Tabulka přechodů a přechodový diagram automatu M_1 z příkladu 3.23



δ_2	a	b
0	1	0
1	2	1
2	2	2

Obrázek 3.35: Tabulka přechodů a přechodový diagram automatu M_2 z příkladu 3.23

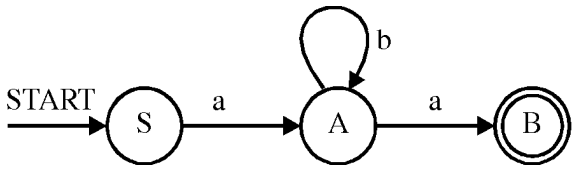


δ	a	b
(0,0)	(1,1)	(0,0)
(1,1)	(1,2)	
(1,2)		

Obrázek 3.36: Tabulka přechodů a přechodový diagram automatu M z příkladu 3.23

Příklad 3.24
Sestrojme konečný automat, který přijímá doplněk jazyka
$L = ab^*a$.
Tuto úlohu budeme řešit ve třech krocích:

1. Sestrojíme konečný automat, který přijímá jazyk L. Jeho přechodový diagram je na obr. 3.37.

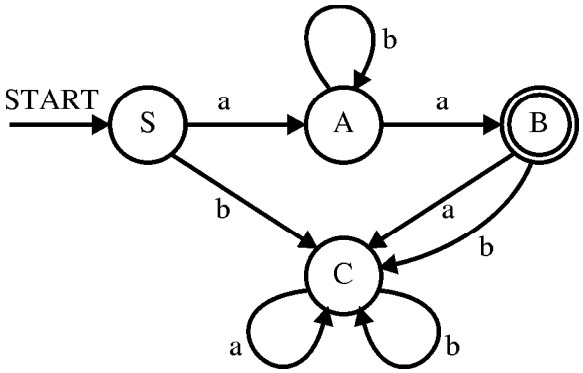


Obrázek 3.37: Konečný automat, který přijímá jazyk $L = ab^*a$ z příkladu 3.24

2. Protože sestrojený automat není úplný, doplníme jej na úplný pomocí Algoritmu 2.22 [JPR03]. Vytvoříme stav C a do něj nasměrujeme chybějící přechody. Přechodový diagram tohoto automatu je na obr. 3.38.

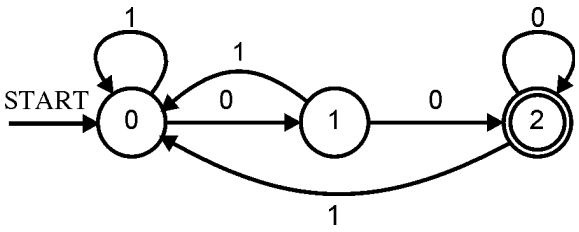
3. Výsledný automat vytvoříme tak, že zaměníme role koncových stavů. Ve výsledném automatu bude množina koncových stavů $F = \{S, A, C\}$. Stav B nebude koncový.

Příklad 3.25
Jsou dány automaty M_1 a M_2 nad abecedou $\{0, 1\}$. Automat M_1 přijímá jazyk řetězců končících dvěma nulami: $L_1 = \{w : v \in \{0, 1\}^*, w = v00\}$ a jeho přechodový diagram je na obr. 3.39. Automat M_2 přijímá jazyk řetězců končících dvěma jedničkami: $L_2 = \{w : v \in \{0, 1\}^*, w = v11\}$ a jeho přechodový diagram

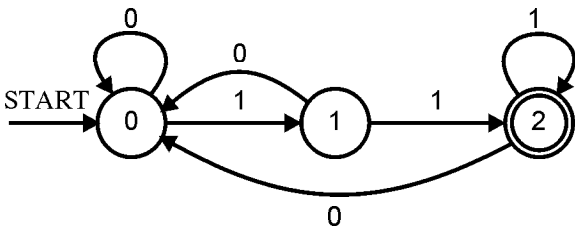


Obrázek 3.38: Úplný konečný automat, který přijímá jazyk $L = ab^*a$ z příkladu 3.24

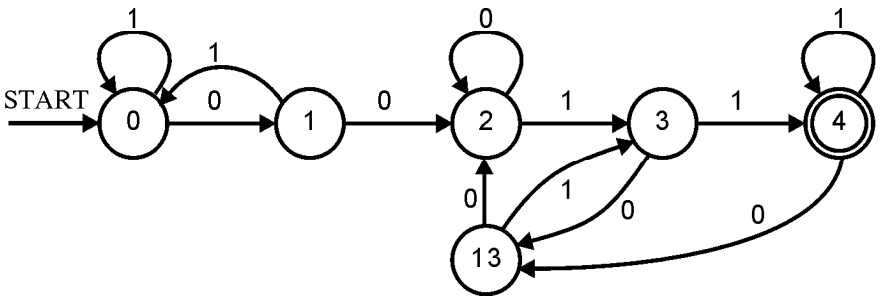
je na obr. 3.40. Sestrojme automat M přijímající jazyk $L_1.L_2$ (zřetězení jazyků L_1 a L_2) pomocí Algoritmu 2.80 [JPR03]. Automat M má přechodový diagram, který je zobrazen na obr. 3.41.



Obrázek 3.39: Automat přijímající jazyk L_1 z příkladu 3.25

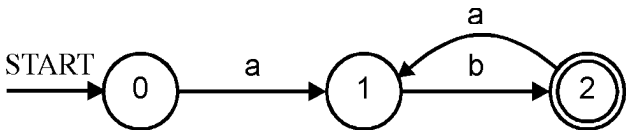


Obrázek 3.40: Automat přijímající jazyk L_2 z příkladu 3.25



Obrázek 3.41: Automat M přijímající zřetězení jazyků L_1 a L_2 z příkladu 3.25

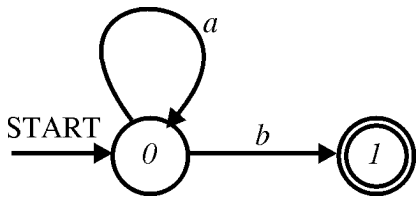
Příklad 3.26
Sestrojme automat přijímající jazyk $L = (ab)^*$. Nejprve sestrojíme automat M' přijímající jazyk $L' = \{ab\}$. Automat M získáme jako automat přijímající iteraci jazyka L' pomocí Algoritmu 2.82 [JPR03]. Deterministický automat M je znázorněn na obr. 3.42.



Obrázek 3.42: Automat přijímající iteraci jazyka L' z příkladu 3.26

Příklad 3.27
Sestrojme konečný automat přijímající iteraci jazyka $L = a^*b$. Nejdříve sestrojíme konečný automat přijímající jazyk L. Jeho tabulka přechodů a přechodový diagram je na obr. 3.43. Dalším krokem je sestrojení nedeterministického ko-

	a	b
0	0	1
1		



Obrázek 3.43: Tabulka přechodů a přechodový diagram konečného automatu z příkladu 3.27

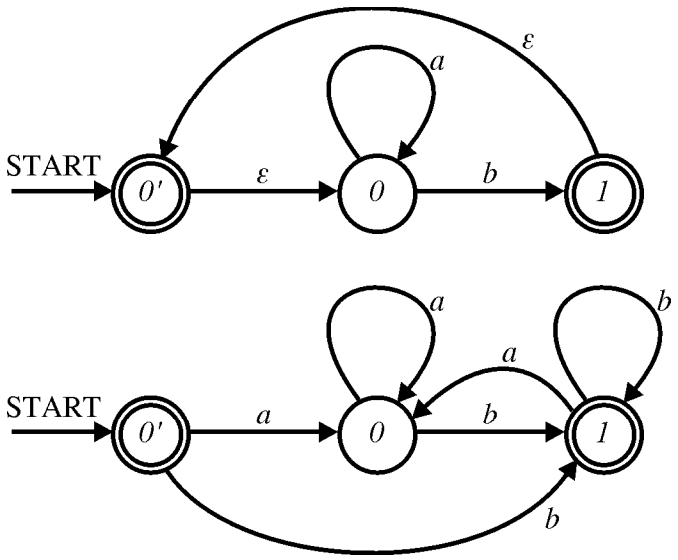
nečného automatu s ε-přechody, který přijímá jazyk L^*. Ten sestrojíme tak,

že vytvoříme nový počáteční stav $0'$, který bude současně koncový, a doplníme přechody:

$\delta(1, \varepsilon) = 0'$,

$\delta(0', \varepsilon) = 0$.

Přechodový diagram tohoto automatu a jeho ekvivalentu bez ε-přechodů je na obr. 3.44.



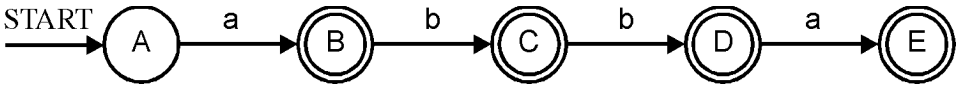
Obrázek 3.44: Přechodový diagram konečného automatu s ε-přechody, který přijímá jazyk L^* z příkladu 3.27 a jeho deterministický ekvivalent

3.8 Konečné automaty pro řetězce a jejich části

Příklad 3.28

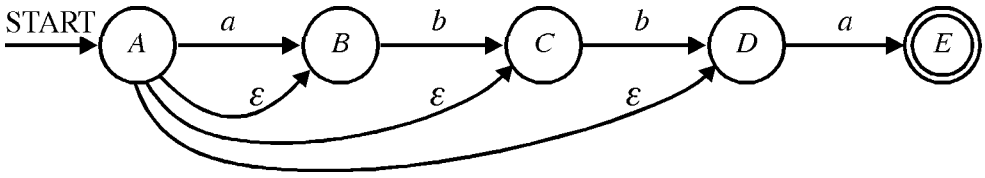
Sestrojme konečný automat, který přijímá řetězec $x = abba$ a všechny jeho neprázdné předpony:

$M_{Pref} = (\{A, B, C, D, E\}, \{a, b\}, \delta, A, \{B, C, D, E\})$, kde zobrazení δ je zadáno přechodovým diagramem na obr. 3.45.



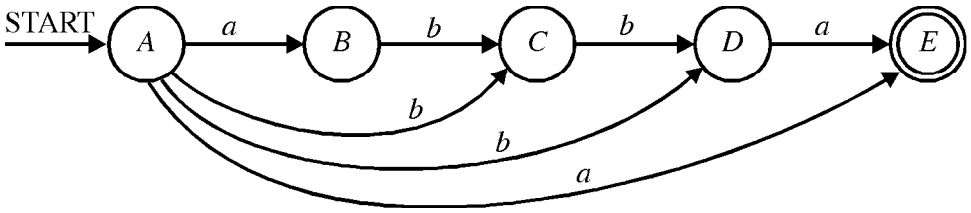
Obrázek 3.45: Přechodový diagram automatu M_{Pref} z příkladu 3.28

Příklad 3.29
Sestrojme konečný automat, který přijímá řetězec $x = abba$ a všechny jeho neprázdné přípony. $M_{Suf} = (\{A, B, C, D, E\}, \{a, b\}, \delta, A, \{E\})$, kde zobrazení δ je zadáno přechodovým diagramem na obr. 3.46.

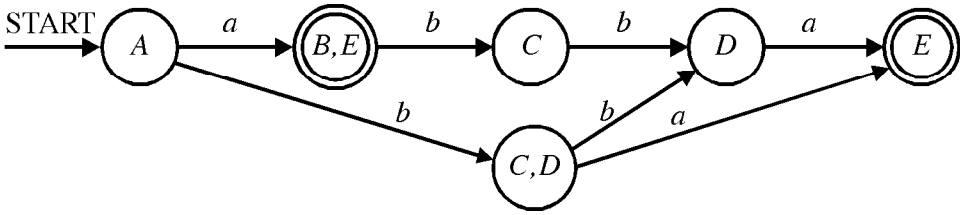


Obrázek 3.46: Přechodový diagram konečného automatu M_{Suf} z příkladu 3.29

Tento automat obsahuje ε–přechody. Po jejich odstranění dostaneme nedeterministický konečný automat s přechodovým diagramem na obr. 3.47. Deterministický konečný automat má přechodový diagram na obr. 3.48



Obrázek 3.47: Přechodový diagram nedeterministického konečného automatu z příkladu 3.29 po odstranění ε–přechodů



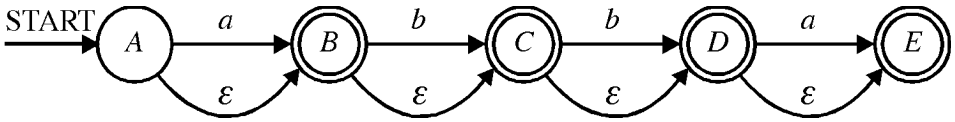
Obrázek 3.48: Deterministický konečný automat z příkladu 3.29

Příklad 3.30
Sestrojme konečný automat, který přijímá množinu řetězců $Fac(abba)$.

$M_{Fac} = (\{A, B, C, D, E\}, \{a, b\}, \delta, A, \{B, C, D, E\})$, kde zobrazení δ je stejné jako v přechodovém diagramu na obr. 3.46. Automat M_{Fac} se liší od automatu M_{Suf} z příkladu 3.29 jen tím, že stavy B, C a D jsou koncové. To znamená, že přijímá předpony všech přípon (viz příklad 3.28). Podobným postupem jako v příkladu 3.29 získáme deterministický konečný automat ekvivalentní s automatem M_{Fac}, jehož přechodový diagram se liší od přechodového diagramu na obr. 3.48 tím, že má všechny stavy kromě stavu A koncové.

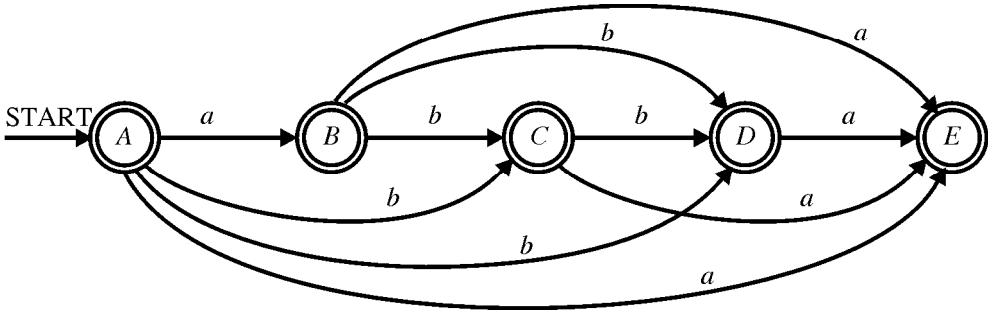
Příklad 3.31
Sestrojme konečný automat, který přijímá množinu řetězců $Sub(abba)$.

$M_{Sub} = (\{A, B, C, D, E\}, \{a, b\}, \delta, A, \{B, C, D, E\})$, kde zobrazení δ je zadáno přechodovým diagramem na obr. 3.49.

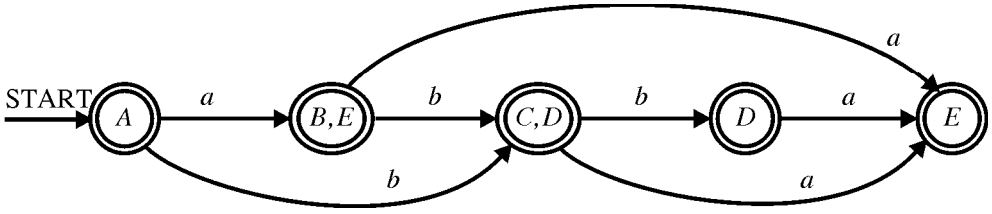


Obrázek 3.49: Přechodový diagram konečného automatu M_{Sub} z příkladu 3.31

Tento konečný automat obsahuje ε–přechody. Tyto ε-přechody zajistí, že je možné libovolný počet symbolů vynechat. Po jejich odstranění získáme nedeterministický konečný automat s přechodovým diagramem na obr. 3.50. Deterministický konečný automat má přechodový diagram na obr. 3.51.



Obrázek 3.50: Přechodový diagram nedeterministického konečného automatu z příkladu 3.31 po odstranění ε–přechodů



Obrázek 3.51: Přechodový diagram deterministického konečného automatu z příkladu 3.31

3.9 Příklady pro cvičení

Cvičení 3.32
Sestrojte konečný automat, který přijímá konečný jazyk $L = \{les, lesák, prales, zálesák\}$.

Cvičení 3.33
Sestrojte konečný automat, který přijímá jazyk řetězců tvaru:
$x, x, ..., x$;

V každém řetězci může být konečný počet symbolů x a přitom každé dva jsou odděleny čárkou.

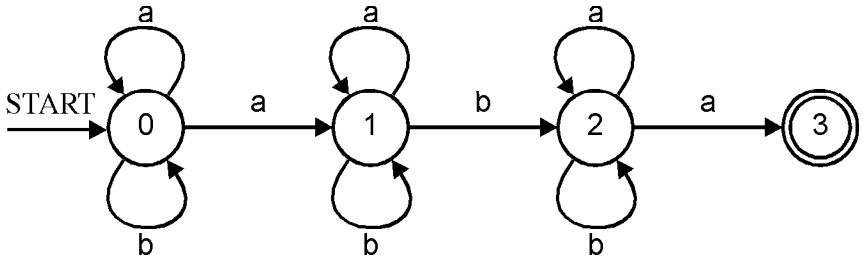
Cvičení 3.34
Sestrojte deterministické konečné automaty přijímající tyto jazyky:

(a) $L(M) = \{x : x \in \{4, 8, 1\}^*$, x obsahuje podřetězec $481\}$.

(b) $L(M) = \{x : x \in \{a\}^*$, délka x $(|x|)$ je dělitelná buď dvěma nebo třemi $\}$.

(c) $L(M) = \{x : x \in \{0, 1\}^*$, počet nul je sudý a počet jedniček je dělitelný třemi $\}$.

Cvičení 3.35
Sestrojte deterministický konečný automat ekvivalentní nedeterministickému automatu, který je zadán přechodovým diagramem na obr. 3.52.

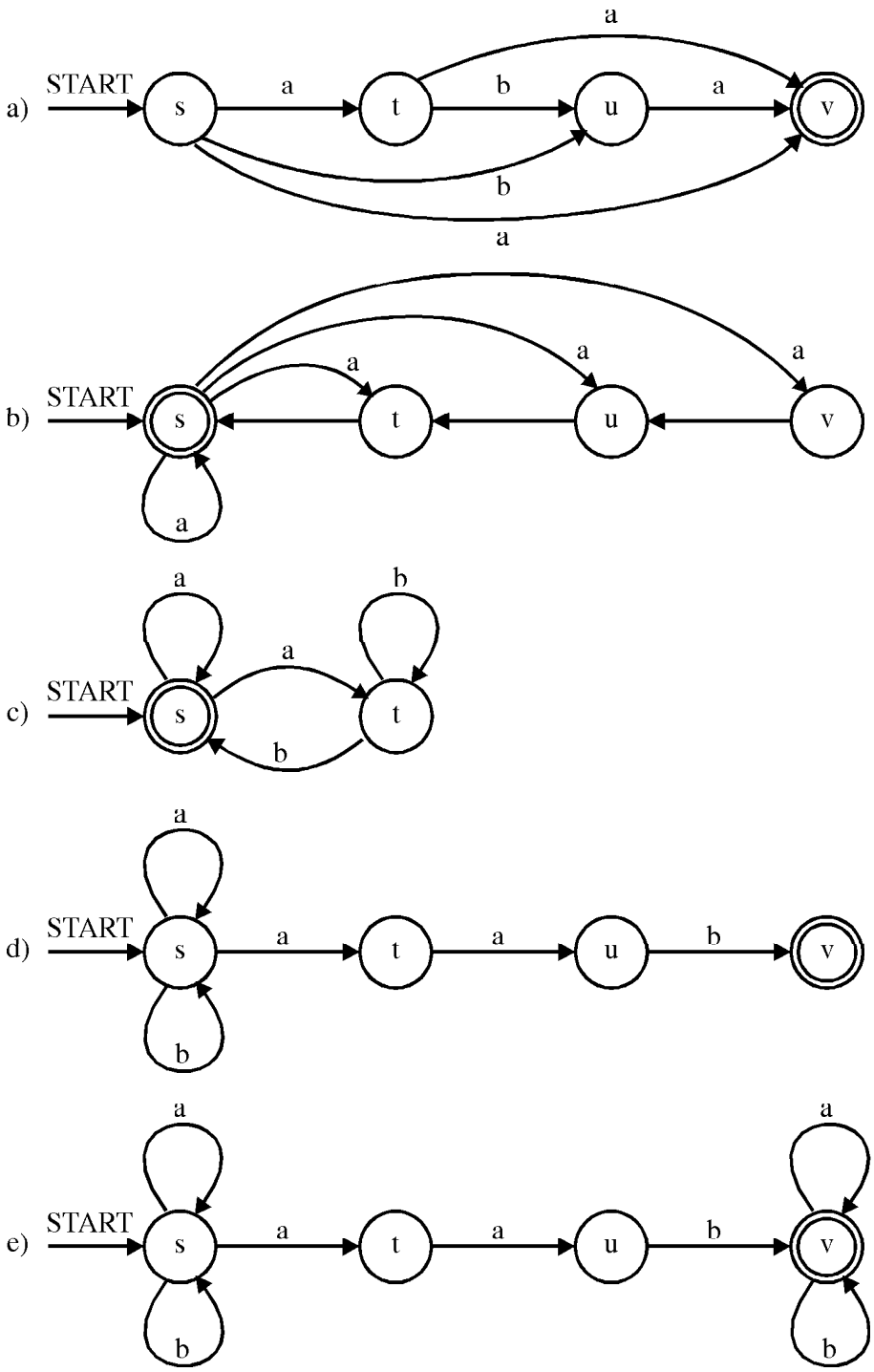


Obrázek 3.52: Nedeterministický konečný automat z cvičení 3.35

Cvičení 3.36
Sestrojte deterministické konečné automaty ekvivalentní automatům, které jsou zadány přechodovými diagramy na obr. 3.53.

Cvičení 3.37
Sestrojte konečný automat, který přijímá jazyk $L = \{(01)^n : n > 0\}$.

Cvičení 3.38
Sestrojte konečný automat přijímající jazyk nad abecedou {0,1}, který se skládá z řetězců, ve kterých jsou buď dvě nuly nebo dvě jedničky těsně vedle sebe.



Obrázek 3.53: Nedeterministické automaty z cvičení 3.36

Cvičení 3.39
Sestrojte konečný automat, který přijímá jazyk nad abecedou {0,1} takový, že v každém řetězci je každá jednička následována alespoň dvěma nulami.

Cvičení 3.40
Sestrojte konečný automat, který přijímá jazyk nad abecedou {0,1} takový, že uvnitř každého řetězce je podřetězec 01010.

Cvičení 3.41
Sestrojte konečný automat, který přijímá jazyk nad abecedou {0,1}, a v každém řetězci je alespoň 5 nul a 2 jedničky.

Cvičení 3.42
Sestrojte deterministický konečný automat, který přijímá řetězce nul a jedniček, pro které platí:

(a) počet nul je dělitelný třemi,

(b) počet jedniček je dělitelný pěti,

(c) počet nul je dělitelný třemi a počet jedniček je dělitelný pěti,

(d) binární číslo, které řetězec nul a jedniček reprezentuje, je dělitelné třemi.

Cvičení 3.43
Kolik stavů bude mít minimalizovaný automat, pokud původní automat byl úplně určený a měl všechny stavy koncové?

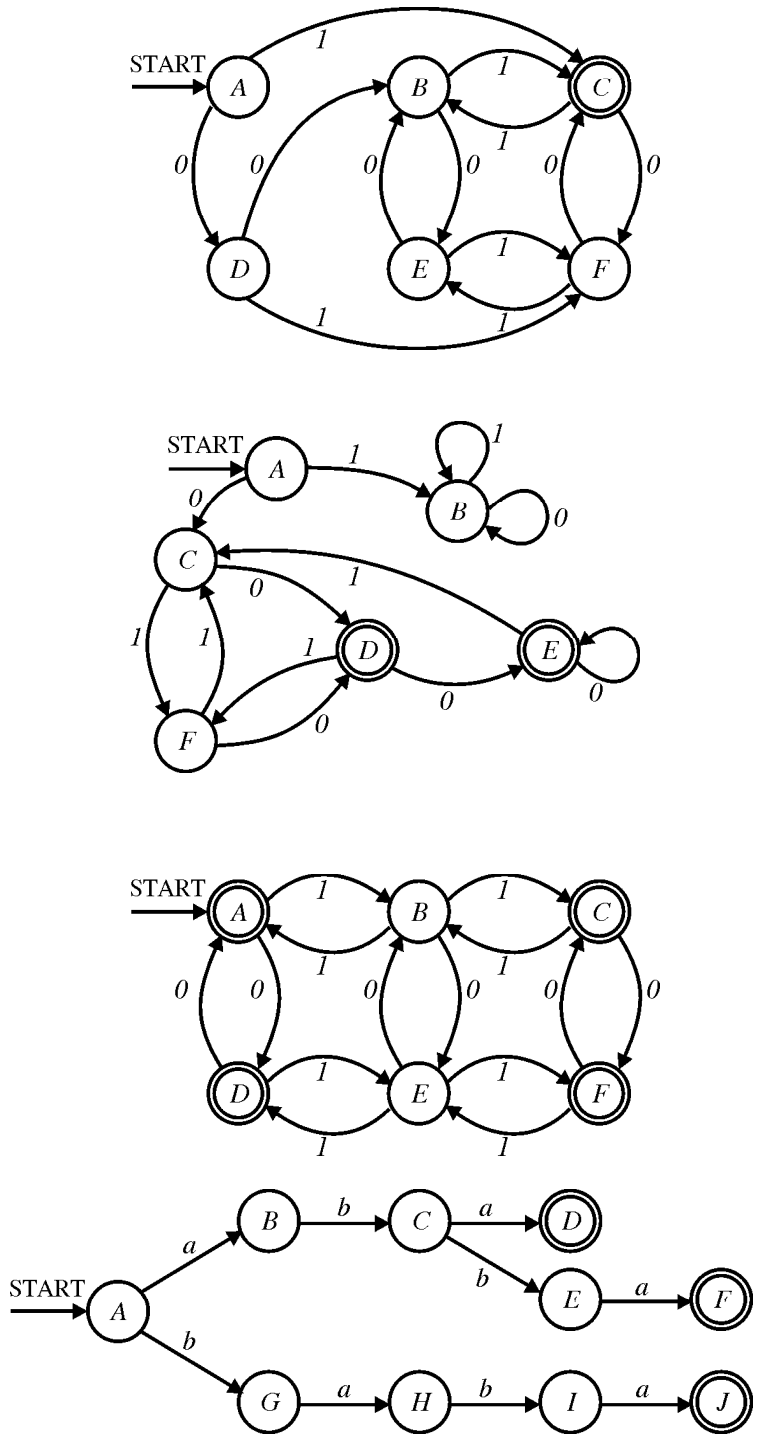
Cvičení 3.44
Minimalizujte automaty zadané přechodovými diagramy na obrázku 3.54.

Cvičení 3.45
Jsou dány deterministické konečné automaty $M_1 = (\{1,2\}, \{a,b\}, \delta_1, 1, \{2\})$, kde zobrazení δ_1 je zadáno tabulkou přechodů:

δ_1	a	b
1	1	2
2	2	1

$M_2 = (\{1,2,3\}, \{a,b\}, \delta_2, 1, \{3\})$, kde zobrazení δ_2 je zadáno tabulkou přechodů:

δ_2	a	b
1	2	3
2	3	1
3	1	2



Obrázek 3.54: Konečné automaty z cvičení 3.44

Sestrojte konečné automaty přijímající jazyky:
$L_1 = L(M_1) \cup L(M_2)$,
$L_2 = L(M_1) \cap L(M_2)$,
$L_3 = L(M_1) \cdot L(M_2)$,
$L_4 = L(M_2)^*$.

Cvičení 3.46
Pro následující dvojice deterministických konečných automatů sestrojte automaty přijímající sjednocení, průnik, a zřetězení jazyků přijímaných těmito automaty.

(a) $M_{1a} = (\{1,2\}, \{a,b\}, \delta_{1a}, 1, \{2\})$
$M_{2a} = (\{1,2\}, \{a,b\}, \delta_{2a}, 1, \{1\})$

δ_{1a}	a	b
1	2	2
2	1	1

δ_{2a}	a	b
1	1	2
2	2	1

(b) $M_{1b} = (\{1,2,3\}, \{a,b\}, \delta_{1b}, 1, \{2,3\})$
$M_{2b} = (\{1,2,3\}, \{a,b\}, \delta_{2b}, 1, \{1,3\})$

δ_{1b}	a	b
1	2	3
2	3	1
3	1	2

δ_{2b}	a	b
1	3	2
2	1	3
3	2	1

(c) $M_{1c} = (\{1,2\}, \{a,b\}, \delta_{1c}, 1, \{2\})$
$M_{2c} = (\{1,2\}, \{a,b\}, \delta_{2c}, 1, \{2\})$

δ_{1c}	a	b
1	2	2
2	1	1

δ_{2c}	a	b
1	2	1
2	1	2

4 Regulární výrazy

4.1 Základní pojmy

Regulární výraz nad abecedou A lze sestrojit podle následujících pravidel:

1. $\emptyset, \varepsilon, a$ jsou regulární výrazy pro všechna $a \in A$.
2. Jsou-li x, y regulární výrazy nad A, pak:

   (a) $(x + y)$ (sjednocení)
   (b) (xy) (zřetězení)
   (c) $(x)^*$ (iterace)

   jsou regulární výrazy nad A.

Hodnotou h(x) regulárního výrazu x je regulární jazyk a je definován takto:

1. $h(\emptyset) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$,
2. $h(x + y) = h(x) \cup h(y)$,
   $h(xy) = h(x).h(y)$,
   $h(x^*) = (h(x))^*$.

Příklad 4.1
Mějme regulární výraz ab. Hodnotou tohoto regulárního výrazu je regulární jazyk L, tedy množina řetězců. Podle definice hodnoty regulárního výrazu platí:

$$h(ab) = h(a).h(b) = \{a\}.\{b\} = \{ab\}.$$

Hodnotou regulárního výrazu ab je tedy jednoprvková množina řetězců nad abecedou $A = \{a, b\}$, jenž obsahuje jediný řetězec ab. Tato množina tvoří regulární jazyk L.

4.2 Konstrukce regulárních výrazů

Příklad 4.2
Sestrojme regulární výraz, jehož hodnotou je jazyk tvořený jediným řetězcem *oko*.

Začneme u elementárního regulárního výrazu. Regulární výraz o popisuje jediný řetězec o, neboť hodnota $h(o)$ výrazu o je jazyk $h(o) = \{o\}$. Podobně je $h(k) = \{k\}$. Oba jazyky dále můžeme zřetězit

$$h(ok) = h(o).h(k) = \{o\}.\{k\} = \{ok\}.$$

Regulární výraz ok opět zřetězíme s výrazem o

$$h((ok)o) = h(ok).h(o) = (h(o).h(k)).h(o) =$$

$$= (\{o\}.\{k\}).\{o\} = \{ok\}.\{o\} = \{oko\}.$$

Řešením je výraz $(ok)o$. Operace zřetězení je asociativní. To znamená, že řešením je tedy i výraz $o(ko)$, neboť

$$h(o(ko)) = h(o).h(ko) = h(o).(h(k).h(o)) =$$

$$= \{o\}.(\{k\}.\{o\}) = \{o\}.\{ko\} = \{oko\},$$

$$(ok)o = o(ko) = oko.$$

Příklad 4.3
Sestrojme regulární výraz jehož hodnotou je jazyk {host, most}.

Z předchozího příkladu je zřejmé, jak sestrojit regulární výraz pro jednoslovný jazyk. Sestrojme regulární výraz pro jazyk s konečnou množinou řetězců.

Nejprve sestrojíme regulární výrazy pro jednotlivé řetězce:

$$h(host) = \{host\},$$

$$h(most) = \{most\}.$$

Nyní použijeme funkci sjednocení:

$$h((host) + (most)) = h(host) \cup h(most) =$$

$$= \{host\} \cup \{most\} = \{host, most\}.$$

Při zápisu regulárních výrazů má operace sjednocení nižší prioritu než operace zřetězení, proto lze řešení příkladu zapsat jako $host + most$.

Všimneme-li si, že řetězce *host* a *most* končí na shodnou příponu *ost*. Můžeme tedy vytvořit nejprve regulární výraz pro jazyk {h, m} a ten dále zřetězit s regulárním výrazem pro jazyk {ost}. Tím získáme regulární výraz $(h+m)ost$, který je rovněž správným řešením příkladu, neboť platí:

$$h((h + m)ost) = h(h + m).h(ost) = (h(h) \cup h(m)).h(ost) =$$

$$= (\{h\} \cup \{m\}).\{ost\} = \{h, m\}.\{ost\} = \{host, most\}.$$

Příklad 4.4
Sestrojme regulární výraz popisující nekonečný jazyk řetězců {otec, praotec, prapraotec, praprapraotec, ...}.

Každý regulární výraz popisující nekonečný jazyk obsahuje alespoň jednu iteraci, protože každý nekonečný jazyk obsahuje řetězce, ve kterých se některá skupina symbolů může opakovat. V našem případě se opakuje předpona *pra*. Výsledný regulární výraz je složen z opakující se předpony *pra* a přípony *otec* takto:

$$(pra)^* otec.$$

Příklad 4.5
Jaký regulární výraz nad jednoznakovou abecedou $\{a\}$ popisuje všechny řetězce se sudým počtem znaků?

Sudý počet znaků řetězce můžeme chápat jako libovolný počet dvojic znaků aa. Tyto dvojice se v řetězci mohou libovolně opakovat. Řešením je regulární výraz

$$(aa)^*.$$

Příklad 4.6
Vytvořme regulární výraz pro jazyk řetězců nad abecedou {0,1} složených z trojic nul a jedniček.

Základním kamenem je trojice nul a trojice jedniček, tedy regulární výrazy 000 a 111. Regulární výraz $000 + 111$ označuje množinu řetězců {000, 111}. Pomocí operace iterace získáme požadovaný regulární výraz:

$$(000 + 111)^*.$$

Příklad 4.7
Nalezněme regulární výraz pro jazyk řetězců zabezpečený sudou paritou. Takový jazyk nad abecedou {0,1} tvoří všechny řetězce se sudým počtem jedniček.

Regulární výraz pro jazyk řetězců se sudým počtem jedniček bez nul je:

$$(11)^*.$$

Za každou jedničkou může následovat libovolné množství nul:

$$(10^*10^*)^*.$$

Před první jedničkou může být také libovolné množství nul. Řešením je výraz:

$$0^*(10^*10^*)^*.$$

Jiným správným řešením je i výraz:

$$(0^*10^*1)^*0^*.$$

který vznikl také z výrazu $(11)^*$, ale zcela symetricky uvažujeme, že nuly mohou být před každou jedničkou a nakonec přidáme nuly tvořící konec řetězce. Oba regulární výrazy popisují stejný jazyk.

4.3 Úpravy regulárních výrazů

Dva různé regulární výrazy mohou mít stejnou hodnotu jako například v příkladu 4.7. Pro popis určitého jazyka jsou rovnocenné, ale při jejich porovnání, shodu jejich hodnoty přímo nezjistíme, protože zápis obou výrazů se liší. Říkáme, že takové regulární výrazy jsou *ekvivalentní*.

Z daného regulárního výrazu můžeme odvozovat ekvivalentní regulární výrazy odvozováním pomocí sady axiomů (viz odst. 2.4.1 [JPR03]), podobně jako upravujeme algebraické výrazy.

Příklad 4.8
Upravme výraz $V = 0^*(0^* + 1^*)$. Příklad řešíme takto:

$$\begin{array}{lcll} V & = & 0^*0^* + 0^*1^* & \text{(distributivnost)}, \\ V & = & 0^* + 0^*1^* & (a^*a^* = a^*), \\ V & = & 0^*(\varepsilon + 1^*) & \text{(distributivnost)}, \\ V & = & 0^*1^* & (\varepsilon + a^* = a^*). \end{array}$$

4.4 Regulární rovnice a jejich řešení

Další možností popisu regulárního jazyka je soustava regulárních rovnic. Takovou soustavu je možno řešit a jejím řešením je regulární výraz.

Příklad 4.9
Je dán jazyk L nad abecedou $\{0, 1\}$, do kterého patří všechny řetězce složené ze samých jedniček a ukončená nulou. Všechny tyto řetězce můžeme jednoduše popsat indukcí:

1. řetězec 0 patří do jazyka L,
2. pokud patří do jazyka L řetězec x, patří tam i řetězec $1x$.

Jinak řečeno x patřící do jazyka L je buď 0 nebo $1x$. To lze vyjádřit regulární rovnicí

$$x = 0 + 1x \tag{R}$$

Řešením této rovnice je takový regulární výraz, po jehož dosazení za x dostaneme ekvivalentní regulární výrazy na levé i pravé straně rovnice. V našem případě je takovým řešením výraz $x = 1^*0$, protože po dosazení do rovnice (R) dostaneme:

$$\begin{array}{lcll} 1^*0 & = & 0 + 1(1^*0) & \text{dosazením}, \\ 1^*0 & = & 0 + (11^*)0 & \text{protože } a(bc) = (ab)c, \\ 1^*0 & = & (\varepsilon + 11^*)0 & \text{protože } a + ba = (\varepsilon + b)a, \\ 1^*0 & = & 1^*0 & \text{protože } \varepsilon + aa^* = a^*. \end{array}$$

Zobecníme-li tento výsledek, můžeme pro regulární rovnici $x = \alpha x + \beta$, kde α a β jsou regulární výrazy, najít řešení ve tvaru $x = \alpha^*\beta$. Podobně pro rovnici $x = x\alpha + \beta$ existuje řešení $x = \beta\alpha^*$

Soustavy regulárních rovnic můžeme řešit např. postupným dosazováním podobně jako soustavu lineárních algebraických rovnic.

Příklad 4.10
Je dána soustava regulárních rovnic:

$$A = 0(A + B),$$
$$B = 1 + A.$$

Soustavu řešíme vyjádřením jednoho z výrazů (A nebo B) a dosazením do druhé rovnice. Začneme dosazením za B ze druhé rovnice $B = 1 + A$ do první rovnice:

$$A = 0(A + 1 + A).$$

Po úpravách dostaneme:

$$A = 0(A + 1),$$
$$A = 0A + 01.$$

Řešením první rovnice je:

$$A = 0^*01.$$

Výraz A dosadíme do druhé rovnice a získáme B:

$$B = 1 + 0^*01.$$

Po úpravách dostaneme:

$$B = (\varepsilon + 0^*0)1,$$
$$B = 0^*1.$$

Řešením soustavy rovnic je:

$$A = 0^*01,$$
$$B = 0^*1.$$

Příklad 4.11
Řešíme soustavu regulárních rovnic:

$$A = 0A + 1B + 1,$$
$$B = 1A + 0B + 0.$$

Vyjádříme-li A z první rovnice, dostaneme $A = 0^*(1B + 1)$ a dosadíme za A do druhé rovnice:

$$B = 10^*(1B + 1) + 0B + 0.$$

Upravme:

$$B = 10^*1B + 10^*1 + 0B + 0,$$
$$B = (10^*1 + 0)B + 10^*1 + 0.$$

Řešením druhé rovnice je:

$$B = (10^*1 + 0)^*(10^*1 + 0).$$

Výraz B dosadíme do rovnice $A = 0^*(1B + 1)$:

$$A = 0^*(1(10^*1 + 0)^*(10^*1 + 0) + 1).$$

A opět upravme:

$$A = 0^*(1(10^*1 + 0)^*(10^*1 + 0) + 1),$$
$$A = 0^*1((10^*1 + 0)^*(10^*1 + 0) + \varepsilon),$$
$$A = 0^*1(10^*1 + 0)^*.$$

Řešením soustavy rovnic je:

$$A = 0^*1(10^*1 + 0)^*,$$
$$B = (10^*1 + 0)^*(10^*1 + 0).$$

4.5 Derivace regulárních výrazů

Derivací regulárního výrazu V podle řetězce x je takový výraz, jehož hodnotou jsou řetězce vzniklé odtržením předpony x od řetězců z množiny $h(V)$. Pro výpočet derivací regulárního výrazu platí následující pravidla:

1. $\frac{dV}{d\varepsilon} = V$,
2. $\frac{d\varepsilon}{da} = \emptyset$,
3. $\frac{d\emptyset}{da} = \emptyset$,
4. $\frac{db}{da} = \emptyset$, jestliže $a \neq b$,
5. $\frac{db}{da} = \varepsilon$, jestliže $a = b$,
6. $\frac{d(U+V)}{da} = \frac{dU}{da} + \frac{dV}{da}$,
7. $\frac{d(UV)}{da} = \frac{dU}{da}V \ + \{\frac{dV}{da} : \varepsilon \in h(U)\}$,
8. $\frac{d(V^*)}{da} = \frac{dV}{da}V^*$,
9. Pro $x = a_1 a_2 ... a_n,\ a_i \in T$:
   $\frac{dV}{dx} = \frac{d}{da_n}(\frac{d}{da_{n-1}}(...\frac{d}{da_2}(\frac{dV}{da_1})...))$.

Příklad 4.12
Určeme derivaci výrazu $V = 10^*1$ podle 1:

$$\begin{aligned} \frac{dV}{d1} &= \frac{d1}{d1}0^*1 && \text{(pravidlo 7)}, \\ &= \varepsilon 0^*1 && \text{(pravidlo 5)}, \\ &= 0^*1 && \text{(úpravou)}. \end{aligned}$$

Příklad 4.13
Určeme derivaci výrazu $V = 010 + 101 + 0^*1 + 1^*0$ podle 0:

$$\begin{aligned} \frac{dV}{d0} &= \frac{d010}{d0} + \frac{d101}{d0} + \frac{d0^*1}{d0} + \frac{d1^*0}{d0} && \text{(pravidlo 6)} \\ &= \frac{d0}{d0}10 + \frac{d1}{d0}01 + \frac{d0^*}{d0}1 + \frac{d1}{d0} + \frac{d1^*}{d0}0 + \frac{d0}{d0} && \text{(pravidlo 7)} \\ &= \varepsilon 10 + \frac{d1}{d0}01 + \frac{d0^*}{d0}1 + \frac{d1}{d0} + \frac{d1^*}{d0}0 + \varepsilon && \text{(pravidlo 5)} \\ &= \varepsilon 10 + \frac{d0^*}{d0}1 + \frac{d1^*}{d0}0 + \varepsilon && \text{(pravidlo 4)} \\ &= \varepsilon 10 + \frac{d0}{d0}0^*1 + \frac{d1}{d0}1^*0 + \varepsilon && \text{(pravidlo 8)} \\ &= \varepsilon 10 + \frac{d0}{d0}0^*1 + \varepsilon && \text{(pravidlo 4)} \\ &= \varepsilon 10 + \varepsilon 0^*1 + \varepsilon && \text{(pravidlo 5)} \\ &= 10 + 0^*1 + \varepsilon && \text{(úpravou)} \end{aligned}$$

Příklad 4.14
Vyjádřeme derivaci výrazu $V = 10^*1^*0$ podle výrazu 100:

$$\begin{aligned}
\tfrac{dV}{d100} &= \tfrac{d}{d0}(\tfrac{d}{d0}(\tfrac{d}{d1}(10^*1^*0))) && \text{(pravidlo 9)}\\
&= \tfrac{d}{d0}(\tfrac{d}{d0}(\tfrac{d1}{d1}0^*1^*0)) && \text{(pravidlo 7)}\\
&= \tfrac{d}{d0}(\tfrac{d}{d0}(\varepsilon 0^*1^*0)) && \text{(pravidlo 5)}\\
&= \tfrac{d}{d0}(\tfrac{d}{d0}(0^*1^*0)) && \text{(úpravou)}\\
&= \tfrac{d}{d0}(\tfrac{d0^*}{d0}1^*0 + \tfrac{d1^*0}{d0}) && \text{(pravidlo 7)}\\
&= \tfrac{d}{d0}(\tfrac{d0^*}{d0}1^*0 + \tfrac{d1^*}{d0}0 + \tfrac{d0}{d0}) && \text{(pravidlo 7)}\\
&= \tfrac{d}{d0}(\tfrac{d0}{d0}0^*1^*0 + \tfrac{d1}{d0}1^*0 + \tfrac{d0}{d0}) && \text{(pravidlo 8)}\\
&= \tfrac{d}{d0}(\varepsilon 0^*1^*0 + \tfrac{d1}{d0}1^*0 + \varepsilon) && \text{(pravidlo 5)}\\
&= \tfrac{d}{d0}(0^*1^*0 + \varepsilon) && \text{(pravidlo 4)}\\
&= \tfrac{d0^*1^*0}{d0} + \tfrac{d\varepsilon}{d0} && \text{(pravidlo 6)}\\
&= \tfrac{d0^*1^*0}{d0} && \text{(pravidlo 2)}\\
&= \tfrac{d0^*}{d0}1^*0 + \tfrac{d1^*0}{d0} && \text{(pravidlo 7)}\\
&= \tfrac{d0^*}{d0}1^*0 + \tfrac{d1^*}{d0}0 + \tfrac{d0}{d0} && \text{(pravidlo 7)}\\
&= \tfrac{d0}{d0}0^*1^*0 + \tfrac{d1}{d0}1^*0 + \tfrac{d0}{d0} && \text{(pravidlo 8)}\\
&= \varepsilon 0^*1^*0 + \tfrac{d1}{d0}1^*0 + \varepsilon && \text{(pravidlo 5)}\\
&= \varepsilon 0^*1^*0 + \varepsilon && \text{(pravidlo 4)}\\
&= 0^*1^*0 + \varepsilon && \text{(úpravou)}
\end{aligned}$$

Příklad 4.15
Derivace stejného výrazu jako v příkladu 4.14 $V = 10^*1^*0$ podle výrazu 011:

$$\begin{aligned}
\tfrac{dV}{d011} &= \tfrac{d}{d1}(\tfrac{d}{d1}(\tfrac{d}{d0}(10^*1^*0))) && \text{(pravidlo 9)}\\
&= \tfrac{d}{d1}(\tfrac{d}{d1}(\tfrac{d1}{d0}0^*1^*0)) && \text{(pravidlo 7)}\\
&= \tfrac{d}{d1}(\tfrac{d}{d1}(\emptyset 0^*1^*0)) && \text{(pravidlo 4)}\\
&= \tfrac{d}{d1}(\tfrac{d\emptyset}{d1}) && \text{(úpravou)}\\
&= \tfrac{d\emptyset}{d1} && \text{(pravidlo 3)}\\
&= \emptyset && \text{(pravidlo 3)}
\end{aligned}$$

4.6 Integrály regulárních výrazů

Integrál regulárního výrazu V podle řetězce x je takový výraz, jehož hodnotou jsou řetězce vzniklé přidáním předpony x k řetězcům z $h(V)$. Pro určování integrálu regulárního výrazu platí následující pravidla:

1. $\int V\ d\varepsilon = V$,
2. $\int \varepsilon da = a$,
3. $\int \emptyset da = \emptyset$,
4. $\int V\ da = aV$,
5. Pro $x = a_1 a_2 ... a_n,\ a_i \in T$:
   $\int V\ dx = \int ...[\int(\int V\ da_n)da_{n-1}]...da_1$.

Příklad 4.16
Vypočtěme integrál výrazu $101 + 010$ podle řetězce 1:

$$\begin{aligned} \int V\ d1 &= \int(101 + 010)d1, \\ &= 1(101 + 010) && \text{(pravidlo 4)} \\ &= 1101 + 1010 && \text{(úpravou)} \end{aligned}$$

Příklad 4.17
Vypočtěme integrál výrazu $101 + 010$ podle řetězce 00:

$$\begin{aligned} \int V\ d00 &= \int[\int(101 + 010)d0]d0 && \text{(pravidlo 5)} \\ &= \int[0(101 + 010)]d0 && \text{(pravidlo 4)} \\ &= 0[0(101 + 010)] && \text{(pravidlo 4)} \\ &= 00101 + 00010 && \text{(úpravou)} \end{aligned}$$

Příklad 4.18
Vypočtěme integrál výrazu $(1^*01 + 0)^*1$ podle řetězce 101:

$$\begin{aligned} \int V\ d101 &= \int\Big\{\int[\int(1^*01 + 0)^*1d1]d0\Big\}d1 && \text{(pravidlo 5)} \\ &= \int\{\int 1(1^*01 + 0)^*1d0\}d1 && \text{(pravidlo 4)} \\ &= \int 01(1^*01 + 0)^*1d1 && \text{(pravidlo 4)} \\ &= 101(1^*01 + 0)^*1 && \text{(pravidlo 4)} \end{aligned}$$

4.7 Regulární výrazy pro řetězce a jejich části

Příklad 4.19
Sestrojme regulární výraz, který popisuje řetězec $x = abba$ a všechny jeho neprázdné předpony:

$$\begin{aligned} & a + ab + abb + abba \\ = \; & a(\varepsilon + b(\varepsilon + b(\varepsilon + a))). \end{aligned}$$

Příklad 4.20
Sestrojme regulární výraz, který popisuje řetězec $x = abba$ a všechny jeho neprázdné přípony:

$$\begin{aligned} & abba + bba + ba + a \\ = \; & (((a + \varepsilon)b + \varepsilon)b + \varepsilon)a. \end{aligned}$$

Příklad 4.21
Sestrojme regulární výraz, který popisuje množinu $Fac(abba)$. První způsob vychází z toho, že popisujeme předpony všech přípon:

$$\begin{aligned} RV_1 \; = \; & a(\varepsilon + b(\varepsilon + b(\varepsilon + a))) \\ + \; & b(\varepsilon + b(\varepsilon + a)) \\ + \; & b(\varepsilon + a) \\ + \; & a \end{aligned}$$

Druhý způsob je založen na tom, že popisujeme přípony všech předpon:

$$\begin{aligned} RV_2 \; = \; & (((a + \varepsilon)b + \varepsilon)b + \varepsilon)a \\ & +((a + \varepsilon)b + \varepsilon)b \\ & +(a + \varepsilon)b \\ & +a \end{aligned}$$

Dokažte ekvivalenci těchto dvou regulárních výrazů.

Příklad 4.22
Sestrojme regulární výraz, který popisuje množinu $Sub(abba)$.

$$RV = (a + \varepsilon)(b + \varepsilon)(b + \varepsilon)(a + \varepsilon)$$

4.8 Příklady pro cvičení

Cvičení 4.23
Nalezněte regulární výraz pro jazyk trojice řetězců {first, second, third}.

Cvičení 4.24
Jaký regulární výraz popisuje řetězce nad abecedou $T = \{0, 1\}$, které začínají nulou?

Cvičení 4.25
Jaký jazyk je hodnotou regulárního výrazu 1^*0^* ?

Cvičení 4.26
Nalezněte regulární výraz popisující řetězce nad abecedou $T = \{0, 1\}$, které obsahují buď samé jedničky, nebo samé nuly.

Cvičení 4.27
Určete regulární výraz pro řetězce nad abecedou $T = \{0, 1\}$, které mají délku dělitelnou třemi.

Cvičení 4.28
Určete regulární výraz, který popisuje řetězce s alespoň třemi nulami.

Cvičení 4.29
Nalezněte regulární výraz pro řetězce nad abecedou $T = \{a, b\}$, které obsahují

(a) sudý počet symbolů a,

(b) lichý počet symbolů b,

(c) sudý počet symbolů a nebo lichý počet symbolů b,

(d) sudý počet symbolů a a lichý počet symbolů b.

Cvičení 4.30
Jaký jazyk popisuje výraz $(0 + 10)^*$?

Cvičení 4.31
Určete hodnoty výrazů:

1. $(0^* + 10)^*$,
2. $(0^* + 1)^* + (1^* + 0)^*$.

Cvičení 4.32
Jaký regulární výraz popisuje všechny řetězce, které mají na sudých místech symbol 1?

Cvičení 4.33
Nalezněte regulární výrazy pro množiny řetězců nad abecedou $T = \{a, b\}$, které

1. obsahují podřetězec ab,
2. obsahují podřetězec aba,
3. neobsahují podřetězec a,
4. neobsahují podřetězec ab,
5. neobsahují podřetězec aba.

Cvičení 4.34
Jaký regulární výraz nad abecedou ternální soustavy $T = \{0, 1, 2\}$ vyjadřuje řetězce s nulovým kontrolním součtem? (Kontrolní součet je algebraický součet všech číslic čísla modulo 3)

Cvičení 4.35
Upravte a zjednodušte regulární výrazy:

1. $11^* + 0^*0 + \varepsilon$,
2. $0^*(1 + \varepsilon)0^*(0 + 1)^*$,
3. $(bb + aa)^*(\varepsilon + b(b + (bb)^*baa)) + aa + bb$.

Cvičení 4.36
Jsou následující dvojice výrazů ekvivalentní? Pokud ano, dokažte tvrzení pomocí elementárních úprav, pokud ne, najděte řetězec, který patří do hodnoty jednoho výrazu a současně nepatří do hodnoty druhého výrazu:

$a)$	$(0 + 1)^*$,	$0^* + 1^*$,
$b)$	$(0(120)^*12$,	$01(201)^*2$,
$c)$	$\emptyset^*$,	ε^*,
$d)$	$(0^*1^*)^*$,	$(0^*1)^*$,
$e)$	$(01 + 0)^*0$,	$0(10 + 0)^*$,
$f)$	$(10^*)^*$,	$(1 + 0)^*$.

Cvičení 4.37
Najděte regulární výraz V', který je doplňkem výrazu $V = (a + b)^*ab(a + b)^*$. ($h(V) \cup h(V') = T^*$ a současně $h(V) \cap h(V') = \emptyset$):

1. pro abecedu $T = \{a, b\}$,
2. pro abecedu $T = \{a, b, c\}$.

Cvičení 4.38
Platí následující rovnosti? Dokažte.

$$\begin{array}{lll} a) & (a^*b)^*a^* & = (a + b)^*, \\ b) & a(ba)^* & = (ab)^*a, \\ c) & a^* & = (aa)^* + a(aa)^*. \end{array}$$

Cvičení 4.39
Řešte regulární rovnice:

1. $V = 1V + 01(V + 1^*)$,
2. $V = V + V(1 + 0^*1 + \varepsilon)$,
3. $V = (V + 1)(0^*1 + 1)^*$.

Cvičení 4.40
Řešte soustavy regulárních rovnic:

1. $$\begin{aligned} A &= A + B, \\ B &= 0A + 1. \end{aligned}$$

2. $$\begin{aligned} A &= 01A + B, \\ B &= 0A + 1B. \end{aligned}$$

3. $$\begin{aligned} A &= 01A + B, \\ B &= A0 + 1B. \end{aligned}$$

4. $$\begin{aligned} A &= 0(1^* + 0)A + 1B, \\ B &= 0 + 1^*B. \end{aligned}$$

5. $$\begin{aligned} A &= 0A + 1B + 2C + 0, \\ B &= 2A + 0B + 1C + 2, \\ C &= 1A + 2B + 0C + 1. \end{aligned}$$

Cvičení 4.41
Určete derivace

1. $$\frac{d(1+0)1^*}{d0},$$
2. $$\frac{d(01+10)^*}{d0},$$
3. $$\frac{d(01^*+010+0^*10^*)}{d010}.$$

Cvičení 4.42
Derivujte podle 01 výrazy:

1. 01^*,
2. $(01)^*$,
3. $(0+1)^*$,
4. $1^*0^*1^*$,
5. $0(1^*0^*1^*)$,
6. $(1^*0^*1^*)1$,
7. $0(1^*0^*1^*)1$.

Cvičení 4.43
Derivujte výraz $01+(0^*1+1^*0)10^*$ podle

1. 0,
2. 1,
3. 01,
4. 10,
5. ε.

Cvičení 4.44
Určete integrály:

1. $$\int 0(1^* + 0^*)\ d1,$$

2. $$\int 0^*(1 + 0^*10)\ d0,$$

3. $$\int (1 + 0^*10)\ d\,011,$$

4. $$\int [0^* + 1(0^*10)^*01]\ d\,110,$$

5. $$\int [11^* + (0^*10)^* + 01 + \varepsilon]1^*\ d\,110,$$

6. $$\int (\varepsilon + 0 + 10 + 110 + 1110)\ d\,110.$$

Cvičení 4.45
Vypočtěte a zjednodušte regulární výrazy. Všimněte si, jak se projeví provedení operací derivace a integrálu podle stejného řetězce, provádíme-li operace v různém pořadí:

1. $$\frac{d}{d1}\Big[\int (0 + 10 + 110)^*(0 + 1)\ d1\Big],$$

2. $$\int \Big\{\frac{d}{d\,10}[(1 + 10 + 100)^*(0 + 1)]\Big\}\ d\,10.$$

5 Vztahy mezi formálními systémy pro popis regulárních jazyků

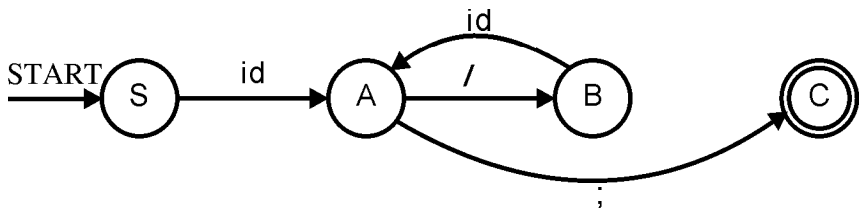
Pro popis regulárních jazyků máme k dispozici tři ekvivalentní formální systémy: regulární gramatiky, konečné automaty a regulární výrazy. Tato kapitola se zabývá příklady vzájemného převodu těchto formálních systémů.

5.1 Vztah mezi regulárními gramatikami a konečnými automaty

Pro každou regulární gramatiku G lze sestrojit konečný automat M takový, že $L(G) = L(M)$. Ke každému konečnému automatu M lze sestrojit regulární gramatiku G takovou, že $L(M) = L(G)$. (odst. 2.5.1 a 2.5.2 [JPR03])

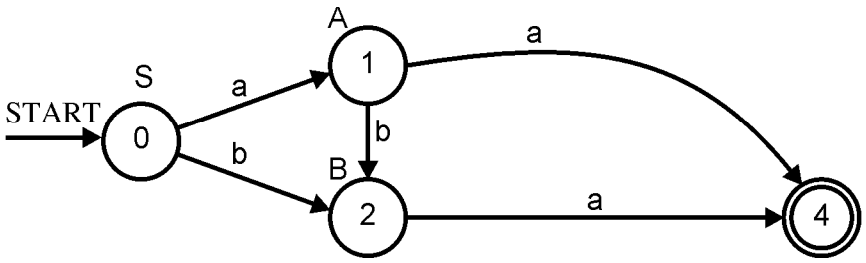
Příklad 5.1
Sestrojme konečný automat pro gramatiku $G = (\{S, A, B\}, \{id, /, ;\}, P, S)$, kde P obsahuje pravidla:
$S \rightarrow idA$
$A \rightarrow ; | /B$
$B \rightarrow idA$
Pro gramatiku G dostaneme pomocí Algoritmu 2.98 [JPR03] konečný automat $M = (\{S, A, B, C\}, \{id, /, ;\}, \delta, S, \{C\})$,
kde δ je definováno přechodovým diagramem na obr. 5.1. Přechodová tabulka má tvar:

δ	id	$/$	$;$
S	$\{A\}$	$\emptyset$	$\emptyset$
A	$\emptyset$	$\{B\}$	$\{C\}$
B	$\{A\}$	$\emptyset$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\emptyset$



Obrázek 5.1: Přechodový diagram konečného automatu z příkladu 5.1

Příklad 5.2
Sestrojme pravou regulární gramatiku pro konečný automat, který je zadán přechodovým diagramem na obr. 5.2. Pro zadaný automat dostaneme pomocí Algoritmu 2.104 [JPR03] gramatiku $G = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:
$S \rightarrow aA|bB$
$A \rightarrow a\ |bB$
$B \rightarrow a\ |$



Obrázek 5.2: Přechodový diagram konečného automatu z příkladu 5.2

5.2 Vztah mezi regulárními výrazy a konečnými automaty

Pro každý regulární výraz V lze sestrojit konečný automat M takový, že $h(V) = L(M)$. Pro každý konečný automat M lze najít regulární výraz V takový, že $L(M) = h(V)$. (odst. 2.5.3 a 2.5.4 [JPR03])

Příklad 5.3
Sestrojme konečný automat pro regulární výraz $V = (a + b)^*ab(a + b)^*$.

1. Algoritmus 2.107 [JPR03] (metoda sousedů):
   $V' = (a_1 + b_2)^*a_3b_4(a_5 + b_6)^*$,
   $Z = \{a_1, b_2, a_3\}$,
   $P = \{a_1a_1, a_1b_2, a_1a_3, b_2a_1, b_2b_2, b_2a_3, a_3b_4, b_4a_5, b_4b_6, a_5a_5, a_5b_6, b_6a_5, b_6b_6\}$,
   $F = \{b_4, a_5, b_6\}$.
   Počáteční stav automatu bude q_0, množina koncových stavů F a pro přechodovou funkci δ platí:
   $\delta(q_0, x)$ obsahuje x_i pro všechna $x_i \in Z$ taková, že x_i vzniklo očíslováním x,
   $\delta(x_i, y)$ obsahuje y_j pro všechny dvojice $x_iy_j \in P$ takové, že y_j vzniklo očíslováním y.
   Pro daný regulární výraz tedy sestrojíme konečný automat
   $M_1 = (\{q_0, a_1, b_2, a_3, b_4, a_5, b_6\}, \{a, b\}, \delta, q_0, \{b_4, a_5, b_6\})$, kde zobrazení δ je definováno následující tabulkou:

δ	a	b
q_0	$\{a_1, a_3\}$	$\{b_2\}$
a_1	$\{a_1, a_3\}$	$\{b_2\}$
b_2	$\{a_1, a_3\}$	$\{b_2\}$
a_3	$\emptyset$	$\{b_4\}$
b_4	$\{a_5\}$	$\{b_6\}$
a_5	$\{a_5\}$	$\{b_6\}$
b_6	$\{a_5\}$	$\{b_6\}$

2. Použijeme Algoritmus 2.110 [JPR03] (metoda derivací):

   $Q = \{V\}, Q_0 = \{V\}$

   $\frac{dV}{da} = (a+b)^*ab(a+b)^* + b(a+b)^* = V_1$

   $\frac{dV}{db} = (a+b)^*ab(a+b)^* = V$

   $Q = \{V, V_1\}, Q_1 = \{V_1\}$

   $\frac{dV_1}{da} = (a+b)^*ab(a+b)^* + b(a+b)^* = V_1$

   $\frac{dV_1}{db} = (a+b)^*ab(a+b)^* + (a+b)^* = V_2$

   $Q = \{V, V_1, V_2\}, Q_2 = \{V_2\}$

   $\frac{dV_2}{da} = (a+b)^*ab(a+b)^* + b(a+b)^* + (a+b)^* = V_3$

   $\frac{dV_2}{db} = (a+b)^*ab(a+b)^* + (a+b)^* = V_2$

   $Q = \{V, V_1, V_2, V_3\}, Q_3 = \{V_3\}$

   $\frac{dV_3}{da} = (a+b)^*ab(a+b)^* + b(a+b)^* + (a+b)^* = V_3$

   $\frac{dV_3}{db} = (a+b)^*ab(a+b)^* + (a+b)^* = V_2$

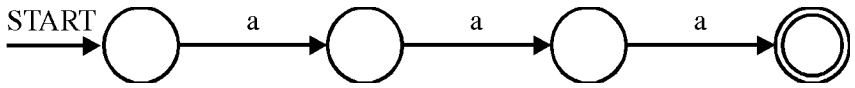
   $Q = \{V, V_1, V_2, V_3\}, Q_4 = \emptyset$

   Množinu stavů konečného automatu tvoří všechny výrazy vzniklé derivací. Množina koncových stavů je tvořena výrazy, jejichž hodnota obsahuje ε. Pokud platí $\frac{dV_1}{da} = V_2$, pak $\delta(V_1, a)$ obsahuje V_2.
   Pro daný regulární výraz sestrojíme konečný automat
   $M_2 = (\{V, V_1, V_2, V_3\}, \{a, b\}, \delta, \{V_2, V_3\})$, kde přechodová funkce δ je definována tabulkou:

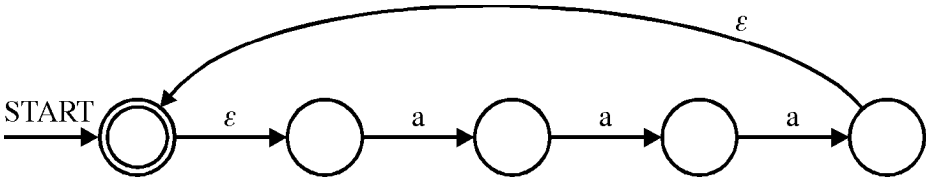
δ	a	b
V	V_1	V
V_1	V_1	V_2
V_2	V_3	V_2
V_3	V_3	V_2

Ukažte ekvivalenci automatů M_1 a M_2.

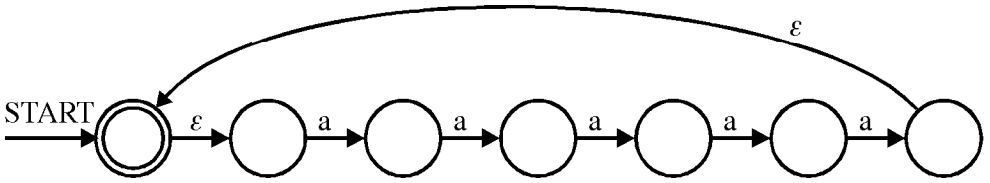
Příklad 5.4
Je dán regulární výraz $R = (aaa)^* + (aaaaa)^*$, který popisuje jazyk $h(R) = \{x : x \in \{a\}^*, |x| \text{ je dělitelná třemi nebo pěti}\}$. Konečný automat přijímající řetězec aaa má přechodový diagram:



Konečný automat, který přijímá jazyk $(aaa)^*$ sestrojíme tak, že přidáme nový počáteční stav a dva ε–přechody. Jeho přechodový diagram má tvar:



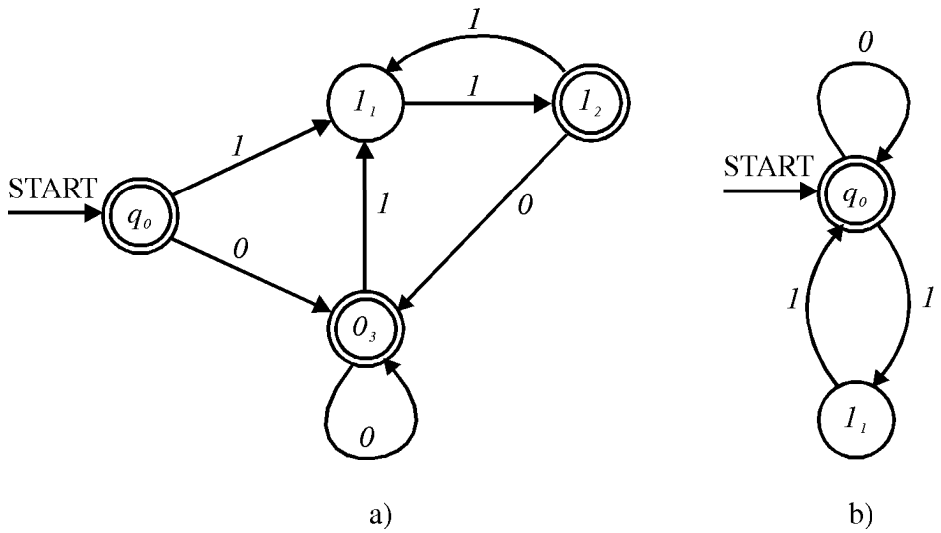
Podobně postupujeme pro $(aaaaa)^*$ a dostaneme:



Výsledek dostaneme tak, že pro dva posledně zmíněné automaty provedeme sjednocení.

Příklad 5.5
Sestrojme ekvivalentní konečný automat pro regulární výraz $(11+0)^*(00+1)^*$. Nejdříve sestrojíme konečný automat pro výraz $(11+0)^*$. Přitom použijeme metodu sousedů: $V' = (1_1 1_2 + 0_3)^*$,
$Z = \{1_1, 0_3\}$,
$P = \{1_1 1_2, 1_2 0_3, 0_3 1_1, 1_2 1_1, 0_3 0_3\}$,
$F = \{1_2, 0_3\}$.
Přechodový diagram tohoto automatu je na obr. 5.3. V automatu na obr. 5.3 a) jsou stavy q_0, 0_3 a 1_2 ekvivalentní a proto můžeme automat zjednodušit na tvar podle obr. 5.3 b).

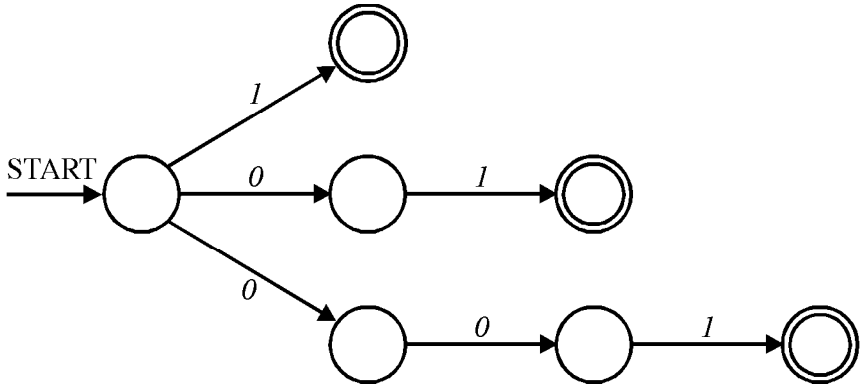
Pokud použijeme metodu derivací, získáme pro výraz $V = (11+0)^*$ přímo minimální automat.
$\frac{dV}{d0} = \frac{d(11+0)^*}{d0} = \frac{d(11+0)}{d0}(11+0)^* = (\emptyset + \varepsilon)(11+0)^* = (11+0)^* = V$,
$\frac{dV}{d1} = \frac{d(11+0)^*}{d1} = \frac{d(11+0)}{d1}(11+0)^* = (1+\emptyset)(11+0)^* = 1(11+0)^* = V_1$,
$\frac{dV_1}{d0} = \frac{d1(11+0)^*}{d0} = \emptyset(11+0)^* = \emptyset$,
$\frac{dV_1}{d1} = \frac{d1(11+0)^*}{d1} = \varepsilon(11+0)^* = (11+0)^* = V$.

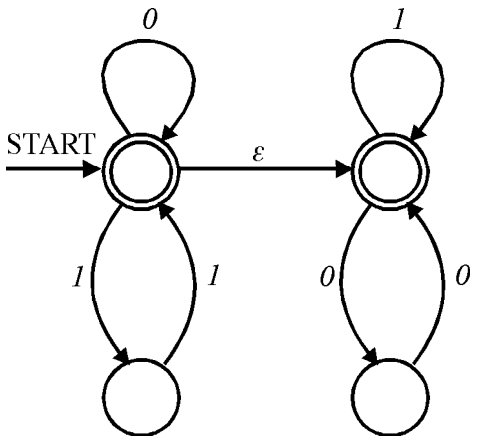


Obrázek 5.3: Přechodové diagramy konečných automatů z příkladu 5.5; a) neoptimalizovaný automat, b) minimalizovaný automat

Výsledný automat je na obr. 5.3 b) s tím, že $q_0 = V$ a $1_1 = V_1$. Konečný automat pro výraz $V' = (00 + 1)^*$ je podobný automatu na 5.3 b) s tím, že ohodnocení přechodů je provedeno opačně ($0 \to 1, 1 \to 0$). Výsledný automat získáme tak, že oba dílčí automaty spojíme dohromady ε–přechodem z koncového stavu automatu pro V do počátečního stavu automatu V'.Výsledný automat je na obr. 5.4.

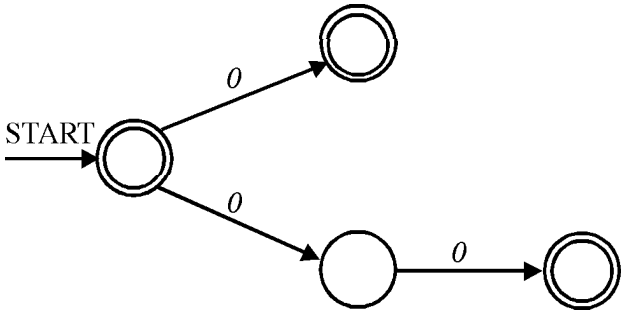
Příklad 5.6
Je dán regulární výraz $R = (1 + 01 + 001)^*(\varepsilon + 0 + 00)$, který popisuje jazyk $h(R) = \{x : x \in \{0, 1\}^*$, žádný podřetězec neobsahuje více než dvě po sobě následující nuly$\}$.
Konečný automat sestrojíme pomocí postupné konstrukce. Pro výraz $(1+01+001)$ sestrojíme konečný automat M_1:





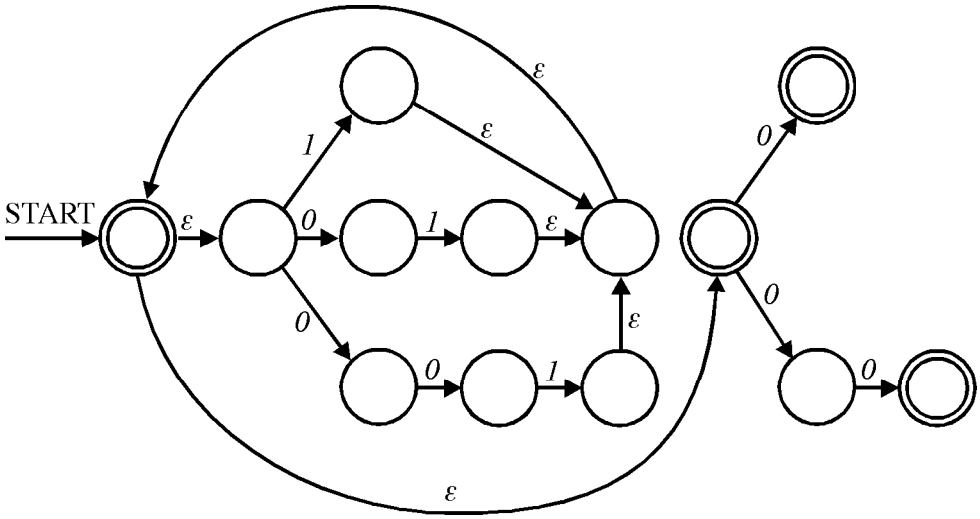
Obrázek 5.4: Výsledný konečný automat z příkladu 5.5

Pro výraz $(\varepsilon + 0 + 00)$ sestrojíme automat M_2:

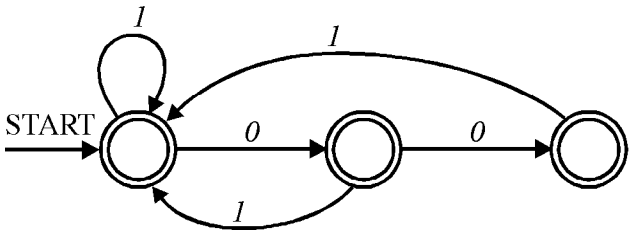


Nakonec sestrojíme automat, který přijímá jazyk $L(M) = L(M_1)^* L(M_2)$. Jeho přechodový diagram je na obr. 5.5. Po odstranění ε–přechodů a konstrukci deterministického automatu dostaneme automat M, jehož přechodový diagram je na obr. 5.6.

Příklad 5.7
Najděme regulární výraz popisující jazyk přijímaný konečným automatem na obr. 5.7. Tento automat přijímá řetězce nad abecedou $\{0, 1\}$, která jsou binárními zápisy čísel dělitelných třemi. Pro řešení této úlohy použijeme Algoritmus 2.120 [JPR03]. Sestavíme soustavu regulárních rovnic:
$X_0 = \varepsilon + X_0 0 + X_1 1$
$X_1 = X_0 1 + X_2 0$
$X_2 = X_1 0 + X_2 1$
a vyřešíme ji vzhledem k X_0. Ze třetí rovnice dostaneme:
$X_2 = X_1 01^*$ a dosadíme do druhé rovnice:
$X_1 = X_0 1 + X_1 01^* 0$ a dostaneme:
$X_1 = X_0 1(01^*0)^*$.



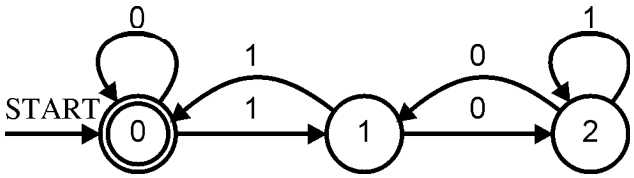
Obrázek 5.5: Nedeterministický konečný automat s ε–přechody z příkladu 5.6



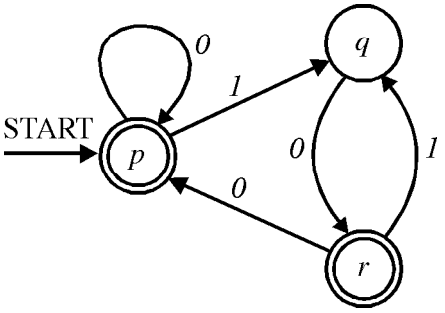
Obrázek 5.6: Deterministický konečný automat z příkladu 5.6

Poté dosadíme do první rovnice:
$X_0 = \varepsilon + X_0 0 + X_0 1(01^*0)^*1 = \varepsilon + X_0(0 + 1(01^*0)^*1)$
$X_0 = (0 + 1(01^*0)^*1)^*$
Regulární výraz popisující binární čísla dělitelná třemi má tedy tvar:
$V = (0 + 1(01^*0)^*1)^*$.

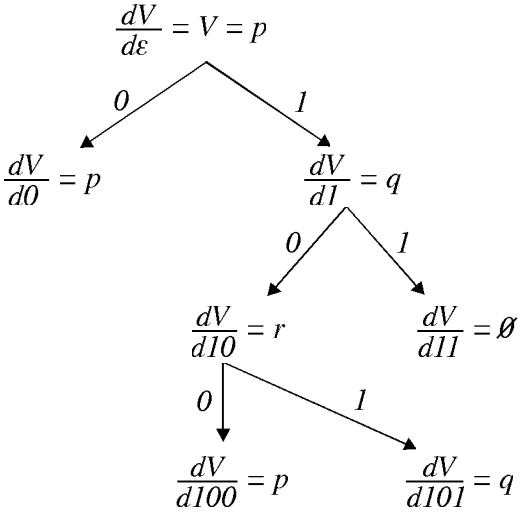
Příklad 5.8
Najděme regulární výraz V popisující jazyk přijímaný konečným automatem M na obr 5.8. Pro konstrukci regulárního výrazu použijeme metodu integrálu (Algoritmus 2.124 [JPR03]). Konečný automat $M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{p\})$, kde zobrazení δ je znázorněno přechodovým diagramem na 5.8. Nejdříve sestrojíme strom pro všechny potřebné derivace.
Nejkratší řetězce jsou:
$x_p = \varepsilon$,
$x_q = 1$,
$x_r = 0$.



Obrázek 5.7: Přechodový diagram konečného automatu z příkladu 5.7



Obrázek 5.8: Přechodový diagram konečného automatu M z příkladu 5.8



$\int \frac{dV}{d100} d0 = 0p$
$\int \frac{dV}{d101} d1 = 1q$
$r = 0p + 1q$
$\int \frac{dV}{d10} d0 = 0r$
$\int \frac{dV}{d11} d1 = \emptyset$
$q = 0r$
$\int \frac{dV}{d0} \cdot d0 = 0p$

$\int \frac{dV}{d1} \cdot d1 = 1p$
$p = 0p + 1q + \varepsilon$
Máme tedy soustavu rovnic:

(1) $r = 0p + 1q$,

(2) $q = 0r$,

(3) $p = 0p + 1q + \varepsilon$.

Z rovnice (2) dosadíme do rovnic (1) a (3) za q a dostaneme

(1') $r = 0p + 10r$,

(3') $p = 0p + 10r + \varepsilon$.

Vyřešíme rovnici (1') a dostaneme:
$r = (10)^*0p$.
Dosadíme za r do rovnice (3'):
$p = 0p + 10(10)^*0p + \varepsilon$,
$p = (0 + 10(10)^*0)p + \varepsilon$,
$p = (0 + 10(10)^*0)^*$.
Po úpravě
$(0 + 10(10)^*0)^* = ((\varepsilon + 10(10)^*)0)^* = ((10)^*0)^*$
Řešení tedy zní:
$V = p = ((10)^*0)^*$,
$r = ((10)^*0)((10)^*0)^*$,
$q = 0((10)^*0)((10)^*0)^*$.

5.3 Vztah mezi regulárními gramatikami a regulárními výrazy

Pro každou regulární gramatiku G lze najít regulární výraz V takový, že $L(G) = h(V)$. Pro každý regulární výraz V lze sestrojit regulární gramatiku G takovou, že $h(V) = L(G)$. (odst. 2.5.5 a 2.5.6 [JPR03])

Příklad 5.9
Najděme regulární výraz popisující jazyk generovaný gramatikou
$G = (\{S, A, B, C\}, \{0, 1\}, P, S)$, kde P obsahuje pravidla:
$S \to 0A|1B$
$A \to 0|0C$
$B \to 1|1C$
$C \to 0A|1B$
Pro řešení použijeme Algoritmus 2.132 [JPR03]. Sestavíme soustavu regulárních rovnic:
$S = 0A + 1B$
$A = 0 + 0C$

$B = 1 + 1C$
$C = 0A + 1B$
a vyřešíme ji vzhledem k počátečnímu symbolu, to jest vzhledem k S.
Druhou a třetí rovnici dosadíme do čtvrté:
$C = 0(0 + 0C) + 1(1 + 1C) = 00 + 00C + 11 + 11C = (00 + 11)C + 00 + 11$
Řešením získáme:
$C = (00 + 11)^*(00 + 11)$
Pro A a B získáme:
$A = 0(00 + 11)^*(00 + 11) + 0 = 0(00 + 11)^*$
$B = 1(00 + 11)^*(00 + 11) + 1 = 1(00 + 11)^*$
Dále dosadíme za A a B do první rovnice:
$S = 00(00 + 11)^* + 11(00 + 11)^* = (00 + 11)(00 + 11)^*$
Výsledný regulární výraz má tvar: $V = (00 + 11)(00 + 11)^*$

Příklad 5.10
Najděme regulární gramatiku generující jazyk popsaný regulárním výrazem $V = (0 + 1)^*0$.

1. Metoda postupného vytváření gramatiky (Algoritmus 2.135 [JPR03]): nejprve vytvoříme gramatiky pro nejjednodušší regulární výrazy a z nich pak konstruujeme gramatiky pro součin, součet a iteraci regulárních výrazů.

   $G_0 = (\{A\}, \{0\}, \{A \to 0\}, A)$

   $G_1 = (\{B\}, \{1\}, \{B \to 1\}, B)$

   $G_{0+1} = (\{C\}, \{0, 1\}, \{C \to 0|1\}, C)$

   $G_{(0+1)^*} = (\{D\}, \{0, 1\}, \{D \to \varepsilon|0D|1D\}, D)$

   $G_{(0+1)^*0} = (\{E\}, \{0, 1\}, \{E \to 0|0E|1E\}, E)$

   Pro daný regulární výraz jsme dostali gramatiku
   $G = (\{E\}, \{0, 1\}, \{E \to 0|0E|1E\}, E)$.

2. Metoda derivací (Algoritmus 2.137 [JPR03]): regulární výraz opakovaně derivujeme podle symbolů abecedy.

   $\frac{dV}{d0} = (0 + 1)^*0 + \varepsilon = V_1$

   $\frac{dV}{d1} = (0 + 1)^*0 = V$

   $\frac{dV_1}{d0} = (0 + 1)^*0 + \varepsilon = V_1$

   $\frac{dV_1}{d1} = (0 + 1)^*0 = V$

   Množinu neterminálních symbolů gramatiky tvoří výraz V a všechny výrazy vzniklé derivací. Množina pravidel obsahuje pravidla:

   $V_1 \to aV_2$ pokud $\frac{dV_1}{da} = V_2$,

   $V_1 \to a$ pokud $\frac{dV_1}{da} = V_2$ a $\varepsilon \in h(V_2)$,

$V \to \varepsilon$ pokud $\varepsilon \in h(V)$.

Pro daný regulární výraz dostáváme gramatiku
$G = (\{V, V_1\}, \{0, 1\}, \{V \to 0|0V_1|1V, V_1 \to 0|0V_1|1V\}, V)$,
kterou lze zredukovat na gramatiku
$G' = (\{V\}, \{0, 1\}, \{V \to 0|0V|1V\}, V)$, protože neterminální symboly V a V_1 generují stejné řetězce.

5.4 Příklady pro cvičení

Cvičení 5.11
K dané regulární gramatice sestrojte konečný automat a najděte ekvivalentní regulární výraz:

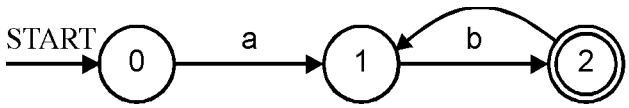
1. $G = (\{L, R\}, \{a\}, P, L)$, kde P obsahuje pravidla:
   $L \to aR$
   $L \to a$
   $R \to, L$

2. $G = (\{L, R\}, \{a\}, P, L)$, kde P obsahuje pravidla:
   $L \to \varepsilon$
   $L \to aR'$
   $L \to a$
   $R' \to, R$
   $R \to aR'$
   $R \to a$

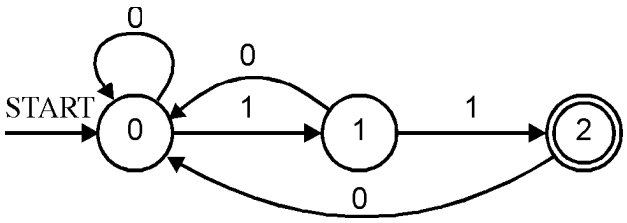
Cvičení 5.12
K danému konečnému automatu najděte ekvivalentní regulární gramatiku a regulární výraz:

1. $M = (\{0, 1, 2\}, \{a, b\}, \delta, \{2\})$, kde zobrazení δ je zadáno na obr. 5.9.

2. $M = (\{0, 1, 2\}, \{0, 1\}, \delta, \{2\})$, kde zobrazení δ je zadáno na obr. 5.10.



Obrázek 5.9: Přechodový diagram konečného automatu ze cvičení 5.12

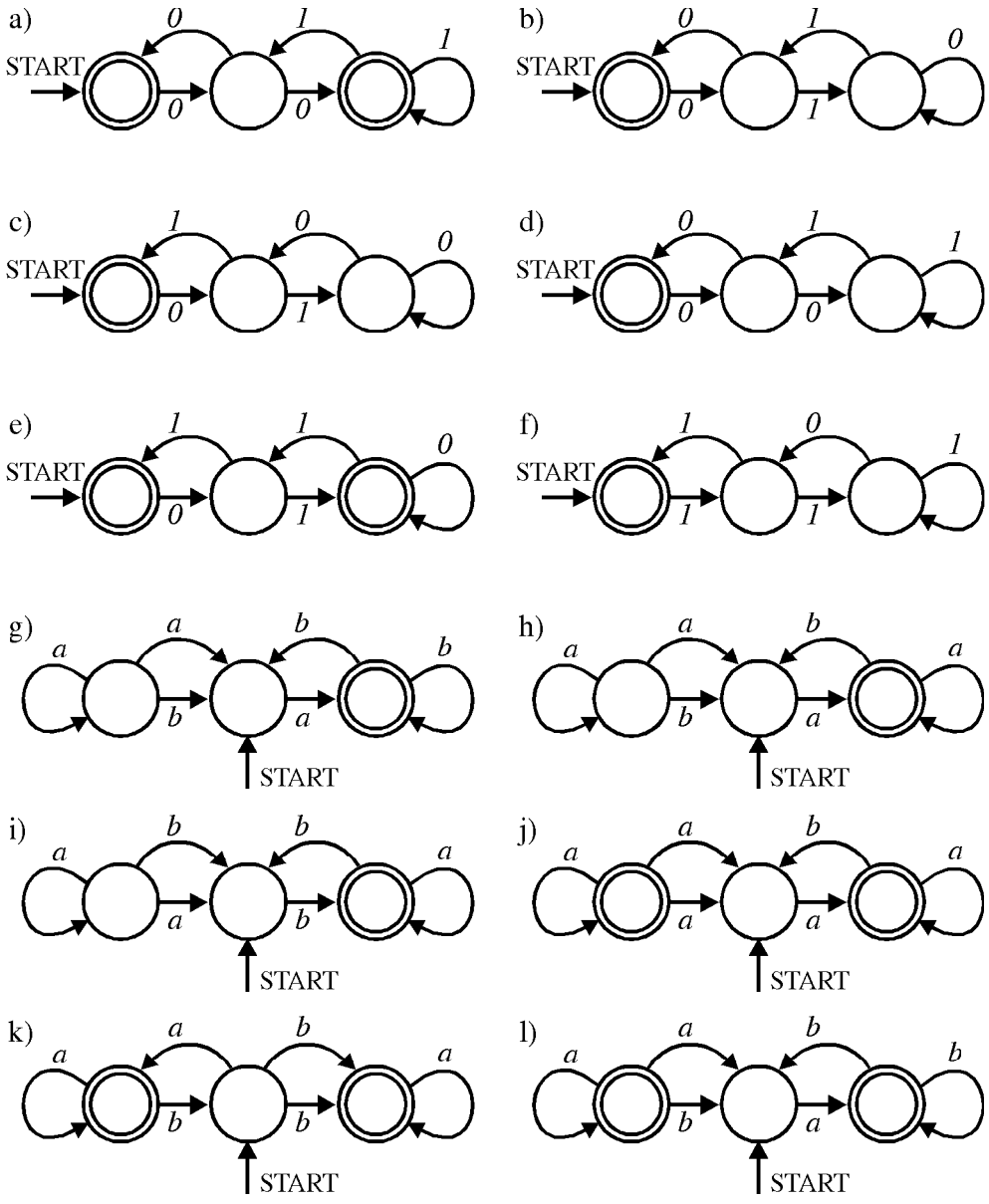


Obrázek 5.10: Přechodový diagram konečného automatu ze cvičení 5.12

Cvičení 5.13
K danému regulárnímu výrazu sestrojte konečný automat a ekvivalentní regulární gramatiku:

1. $1(0+1)^*101$
2. $1(1010^* + 1(010)^*1)0$
3. $(11 + (00)^*01)^*1$
4. $((0^* + 1)(1^*0))$
5. $(b + a(aa^*b)^*b)^*$
6. $(a + b^*)abb(a^* + b)$
7. $(000^* + 111^*)^*$
8. $(01 + 10)(01 + 10)(01 + 10)$
9. $(0 + 1(01^*0)^*1)^*$
10. $(00 + 11)^*(01 + 10)(00 + 11)^*$
11. $(000)^*1 + (00)^*1$
12. $(0(01)^*(1 + 00)) + 1(10)^*(0 + 11)^*$

Cvičení 5.14
Pro konečné automaty zadané přechodovými diagramy na obr. 5.11 sestrojte ekvivalentní regulární gramatiky a regulární výrazy.



Obrázek 5.11: Přechodové diagramy konečných automatů ze cvičení 5.14

6 Implementace konečných automatů

6.1 Implementace deterministických konečných automatů

V této kapitole se budeme zabývat čtyřmi způsoby reprezentace konečných automatů ve formě algoritmů (tj. implementace konečných automatů). První dva způsoby implementace jsou založeny na tom, že tabulka přechodů je reprezentována jako celočíselná obdélníková matice nebo řídká matice, kdy jsou v paměti uloženy jen některé prvky. Další dva způsoby implementace spočívají ve vytvoření programu, který realizuje činnost automatu bez explicitního použití tabulky přechodů.

6.1.1 Tabulka přechodů jako celočíselná matice

Metoda implementace deterministického konečného automatu popsaná v tomto odstavci spočívá v tom, že tabulka přechodů zadaného automatu je reprezentována jako obdélníková celočíselná matice, se kterou pracuje univerzální algoritmus.

Reprezentace konečného automatu $M = (K, T, \delta, q_0, F)$ je provedena takto:

Tabulka přechodů (zobrazení δ) je reprezentována jako matice, jejíž řádky jsou označeny stavy, sloupce vstupními symboly a hodnoty matice jsou hodnoty přechodové funkce δ. Jednotlivé stavy a vstupní symboly jsou zakódovány jako celá čísla. Množina koncových stavů je reprezentována jako množina $F = \{f_1, f_2, \ldots, f_{|F|}\}$.

Univerzální algoritmus v jazyce PASCAL má tvar:

```
type TYPSTAV = (NEDEFINOVÁN,Q0,Q1,...,Q|K|);
     TYPSYMBOL = 1 .. |T|;
var  STAV : TYPSTAV;
     SYMBOL : TYPSYMBOL;
     TABULKAPŘECHODŮ : array[Q0 .. Q|K|,TYPSYMBOL] of TYPSTAV;

procedure DALŠÍSYMBOL(var X:TYPSYMBOL); external;
procedure ČTENÍTABULKYPŘECHODŮ; external;
begin
   ČTENÍTABULKYPŘECHODŮ;
   F := [QF1,QF2, ... ,Q|F|];
   STAV := Q0;
   while not EOF(INPUT) and (STAV <> NEDEFINOVÁN) do
      begin
         DALŠÍSYMBOL(SYMBOL);
         STAV := TABULKAPŘECHODŮ[STAV,SYMBOL];
      end;
   if STAV in F then WRITE(OUTPUT,'VSTUP PŘIJAT')
               else WRITE(OUTPUT,'VSTUP NEPŘIJAT')
end.
```

Poznámky k programu:

Zápisy $|K|$, $|T|$ a $|F|$ jsou implementační konstanty a představují počet stavů, počet vstupních symbolů a počet koncových stavů implementovaného automatu.

Procedura `ČTENÍTABULKYPŘECHODŮ` zajistí definici hodnot jednotlivých prvků této tabulky.

Procedura `DALŠÍSYMBOL` přečte při každém vyvolání jeden vstupní symbol a převede jej do vnitřní reprezentace, tj. na celé číslo v intervalu $\langle 1, |T| \rangle$.

Metoda implementace deterministického konečného automatu popsaná v tomto odstavci je univerzální, tj. je použitelná pro každý deterministický konečný automat. Její použití není efektivní z hlediska využití paměti v případě, že značný počet prvků v tabulce přechodů má hodnotu stejnou. V takovém případě je možno tabulku přechodů považovat za řídkou matici (tj. matici s mnoha stejnými prvky) a jako takovou ji v počítači reprezentovat.

6.1.2 Tabulka přechodů jako řídká matice

Řídká matice je taková matice, ve které většina prvků má stejnou hodnotu. V tomto odstavci ukážeme způsob implementace deterministického konečného automatu založený na tom, že tabulka přechodů je v paměti uložena jako řídká matice, tj. v paměti jsou uloženy jen ty prvky, které mají jiné hodnoty než `NEDEFINOVÁN`.

Je dán konečný automat, který přijímá desetinná čísla se znaménkem $+$ nebo $-$. Pokud je v čísle uvedena desetinná tečka, musí být před ní i za ní alespoň jedna číslice (d).

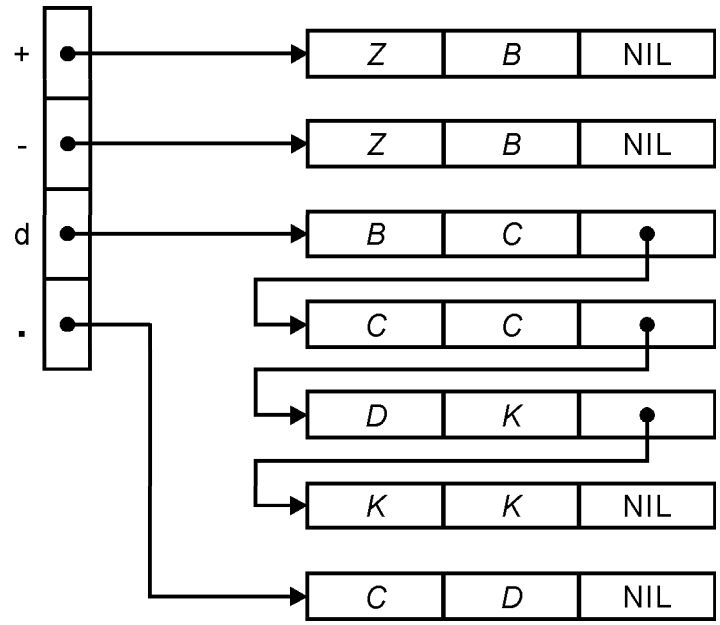
Tabulka přechodů je uvedena dále.

	$+$	$-$	d	.
Z	B	B		
B			C	
C			C	D
D			K	
K			K	

Tuto tabulku můžeme reprezentovat jako řídkou matici například podle obr. 6.1.

Vlastnosti tohoto zobrazení tabulky přechodů můžeme shrnout takto: Každému vstupnímu symbolu odpovídá ukazatel na informace o příslušném sloupci tabulky přechodů. Informace o sloupci tabulky přechodů obsahují zřetězený seznam dvojic (starý stav, nový stav).

Program, který pracuje s tabulkou přechodů uloženou jako řídká matice po sloupcích, má v jazyce PASCAL tvar:



Obrázek 6.1: Datová struktura pro reprezentaci tabulky přechodů

```
program KONEČNÝAUTOMAT(INPUT, OUTPUT);
type TYPSTAV = (NEDEFINOVÁN,Q0,...,Q|K|);
     TYPSYMBOL = 1 .. |T|;
     PŘECHOD = record
                  STARÝSTAV : TYPSTAV;
                  NOVÝSTAV  : TYPSTAV;
                  NÁSLEDNÍK : ^PŘECHOD;
               end;

var STAV : TYPSTAV;
    SYMBOL : TYPSYMBOL;
    UKAZATELSLOUPCŮ : array [TYPSYMBOL] of ^PŘECHOD;
    F : set of TYPSTAV;
    K : ^PŘECHOD;

procedure ERROR; external;
procedure ČTENÍTABULKYPŘECHODŮ; external;
procedure DALŠÍSYMBOL(var X:TYPSYMBOL); external;
begin
   ČTENÍTABULKYPŘECHODŮ;
   F := [QF1,QF2,..., Q|F|];
   STAV := Q0;
   while not EOF(INPUT) and STAV <> NEDEFINOVÁN do
```

```
      begin
         DALŠÍSYMBOL(SYMBOL);
         K := UKAZATELSLOUPCŮ[SYMBOL];
         while STAV <> K^.STARÝSTAV and K^.NÁSLEDNÍK <> NIL do
            K := K^.NÁSLEDNÍK;
         if K^.STARÝSTAV = STAV then
            STAV := K^.NOVÝSTAV
         else
            STAV := NEDEFINOVÁN
      end;
   if STAV in F then WRITE (OUTPUT, 'VSTUP PŘIJAT')
               else WRITE (OUTPUT, 'VSTUP NEPŘIJAT')
end.
```

Poznámky k programu:
$|K|$, $|T|$ a $|F|$ představují počet stavů, počet vstupních symbolů a počet koncových stavů. Procedura `ČTENÍTABULKYSYMBOLŮ` zajistí vytvoření datové struktury pro informace o jednotlivých sloupcích tabulky přechodů a uloží do ní potřebné hodnoty. Současně zajistí nastavení ukazatelů v poli `UKAZATELSLOUPCŮ` na začátky údajů o jednotlivých sloupcích ve vytvořené datové struktuře.

Procedura `DALŠÍSYMBOL` přečte při každém vyvolání jeden vstupní symbol a převede jej do vnitřní reprezentace, tj. na celé číslo v intervalu $\langle 1, |T| \rangle$.

Způsob implementace konečného automatu popsaný v tomto odstavci je použitelný pro každý deterministický konečný automat. Výhodné je jeho použití zejména v případě, že v tabulce přechodů konečného automatu je velké množství stejných položek. V případě, že v tabulce přechodů je málo stejných položek, je tato metoda nevhodná.

6.1.3 Konečný automat implementovaný jako program, stav reprezentován jako proměnná

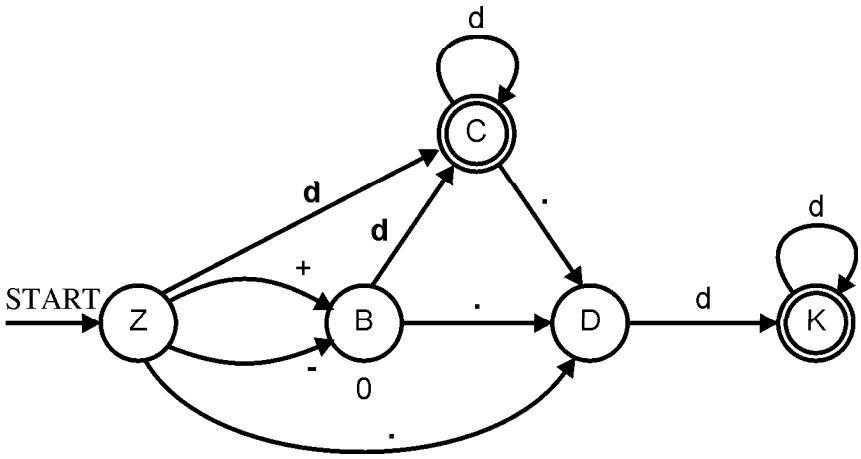
V tomto odstavci ukážeme metodu implementace deterministického konečného automatu, která spočívá v tom, že řídící část automatu je reprezentována příkazem cyklu while, uvnitř kterého je řídící struktura (příkaz case), která odpovídá struktuře přechodového diagramu daného automatu. Přitom stav automatu je reprezentován proměnnou. Princip této metody si ukážeme na příkladu.

Je dán konečný automat, jehož přechodový diagram je na obr. 6.2.

Konečný automat podle obr. 6.2 čte desetinná čísla, která mohou být bez znaménka nebo se znaménkem + nebo −, tečka může být před první číslicí, za tečkou musí být alespoň jedna číslice.

Program pro reprezentaci tohoto automatu má tvar:

```
program ANALÝZAČÍSLA(INPUT,OUTPUT);
var   STAV : (NEDEFINOVÁN,Z,B,C,D,K);
      SYMBOL : CHAR;
```



Obrázek 6.2: Konečný automat pro analýzu desetinných čísel

```
begin
   STAV := Z;
   while not EOF(INPUT) and STAV <> NEDEFINOVÁN do
      begin
         READ(INPUT,SYMBOL);
         case STAV of
            Z : if '0' <=  SYMBOL <= '9' then
                   STAV := C
                else
                   if SYMBOL = '+' or SYMBOL = '-' then
                      STAV := B
                   else
                      if SYMBOL = '.' then
                         STAV := D
                      else
                         STAV := NEDEFINOVÁN;
            B : if '0' <= SYMBOL <= '9' then
                   STAV := C
                else
                   if SYMBOL = '.' then
                      STAV := D
                   else
                      STAV := NEDEFINOVÁN;
            C : if '0' <= SYMBOL <= '9' then
                   STAV := C
                else
                   if SYMBOL = '.' then
```

```
                  STAV := D
               else
                  STAV := NEDEFINOVÁN;
         D : if '0' <= SYMBOL <= '9' then
                  STAV := K
               else
                  STAV := NEDEFINOVÁN;
         K : if '0' <= SYMBOL <= '9' then
                  STAV := K
               else
                  STAV := NEDEFINOVÁN;
      end; (* of case *)
   end; (* of while *)
if STAV = C or STAV = K then
   WRITE (OUTPUT, 'ŘETĚZ PŘIJAT')
else
   WRITE (OUTPUT, 'ŘETĚZ NEPŘIJAT')
end.
```

Tato metoda je opět univerzální, ale vyžaduje, aby pro každý automat byl vytvořen speciální program. Při jeho konstrukci je možno použít optimalizaci. V našem příkladu je činnost automatu ve stavech D a K stejná, proto můžeme úseky programu pro D a K ztotožnit, např. takto:

```
D,K : if '0' <= SYMBOL <= '9' then
         STAV := K
      else
         STAV := NEDEFINOVÁN;
```

Přitom stavy D a K nelze ztotožnit.

6.1.4 Konečný automat implementovaný jako objektová struktura

V tomto odstavci ukážeme metodu implementace konečného deterministického automatu jako objektové struktury. Objektová struktura zde ukázaná je jen jedna z mnoha možných. Konkrétní strukturu je nutno volit s ohledem na konkrétní aplikaci.

Konečný deterministický automat se skládá z instancí objektu Stav reprezentující jednotlivé stavy automatu. Každý stav obsahuje pole instancí objektu Přechod. Celá struktura je zapouzdřena v objektu KonečnýDeterministickýAutomat.

Tento způsob implementace je univerzální, ale vyžaduje sestavit v paměti konkrétní objektovou strukturu (tělo konstruktoru objektu KonečnýDeterministickýAutomat).

Následující příklad implementuje automat přijímající desetinná čísla z obr. 6.2.

```
public class Stav {
  public boolean jeKoncový;
  public Přechod[ ] přechody;

  public Stav přejdi(char symbol) {
    for (int i=0; i < přechody.length; i++) {
            if (přechody[i].symbol == symbol)
                  return přechody[i].cílovýStav;
    }
    return null;
  }
}

public class Přechod {
  public char symbol;
  public Stav cílovýStav;

  public Přechod(char symbol, Stav cílovýStav){
    this.symbol = symbol;
    this.cílovýStav = cílovýStav;
  }
}

public class KonečnýDeterministickýAutomat {
  private Stav počátečníStav;

  public KonečnýDeterministickýAutomat(){
    Stav Z = new Stav(); Z.jeKoncový = false;
    Stav C = new Stav(); C.jeKoncový = true;
    Stav B = new Stav(); B.jeKoncový = false;
    Stav D = new Stav(); D.jeKoncový = false;
    Stav K = new Stav(); K.jeKoncový = true;

    Z.přechody = new Přechod[ ] {
                   new Přechod('+',B),
                   new Přechod('-',B),
                   new Přechod('.',D),
                   new Přechod('d',C),
                 };
    C.přechody = new Přechod[ ] {
                   new Přechod('.',D),
                   new Přechod('d',C),
                 };
    B.přechody = new Přechod[ ] {
```

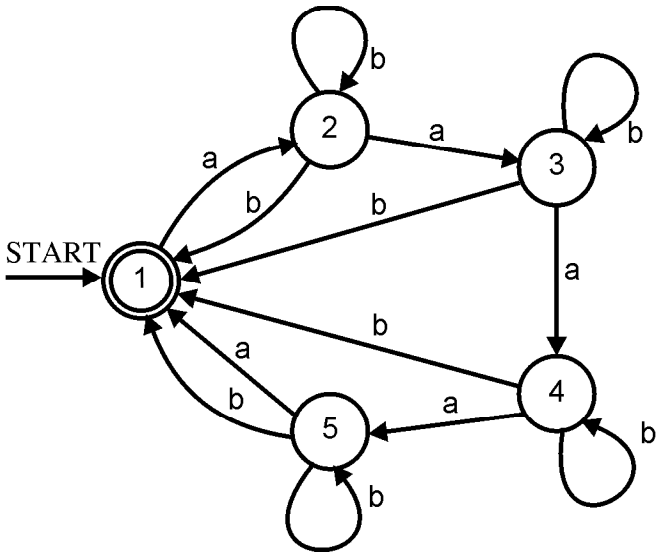
```
                  new Přechod('.',D),
                  new Přechod('d',C),
                };
    D.přechody = new Přechod[ ] {
                  new Přechod('d',K),
                };
    K.přechody = new Přechod[ ] {
                  new Přechod('d',K),
                };
    počátečníStav = Z;
  }

  public boolean přijmi(String slovo){
    Stav aktuálníStav = počátečníStav;

    for(int i=0; i<slovo.length(); i++){
        char symbol =
          ((slovo.charAt(i) >= '0' && slovo.charAt(i) <= '9') ?
          'd' : slovo.charAt(i));
        aktuálníStav = aktuálníStav.přejdi(symbol);
        if (aktuálníStav == null) return false;
                                          //neexistuje přechod,
                                          //slovo nelze přijmout
    }
    return aktuálníStav.jeKoncový;
  }
}
```

6.2 Implementace nedeterministických konečných automatů

V předchozích odstavcích jsou popsány metody implementace deterministických konečných automatů. Jestliže je dán nedeterministický konečný automat, pak můžeme jeho implementaci provést tak, že nejdříve sestrojíme ekvivalentní deterministický konečný automat a ten potom implementujeme. Tento postup však nemusí být efektivní, protože se může stát, že pro nedeterministický konečný automat s n stavy má ekvivalentní deterministický konečný automat 2^n stavů.

Příklad 6.1
Je dán nedeterministický konečný automat $NKA = (\{1,2,3,4,5\}, \{a,b\}, \delta, 1, \{1\})$, kde zobrazení δ je definováno tabulkou. Tabulka přechodů a přechodový diagram je na obr. 6.3.
Tento automat je nedeterministický a má 5 stavů. Ekvivalentní deterministický automat má $2^5 = 32$ stavů. Přechodová tabulka tohoto automatu je uvedena

	a	b
1	{2}	$\emptyset$
2	{3}	{1,2}
3	{4}	{1,3}
4	{5}	{1,4}
5	{1}	{1,5}



Obrázek 6.3: Tabulka přechodů a přechodový diagram nedeterministického konečného automatu z příkladu 6.1

dále.

Budeme-li implementovat nedeterministický konečný automat s n stavy výše uvedeným způsobem, může pro větší n nastat situace, že deterministický konečný automat není možno prakticky implementovat. Proto se nyní budeme zabývat metodami implementace nedeterministických konečných automatů. Uvedeme dvě metody. První metoda je založena na použití tabulky přechodů. Druhá metoda se opírá o implementaci automatu ve formě programu.

V obou případech jsou stavy automatu reprezentovány stavovým vektorem. Každému stavu nedeterministického konečného automatu odpovídá jeden prvek tohoto vektoru. Jestliže se automat nachází v určitém stavu, bude mít odpovídající prvek stavového vektoru hodnotu *true*, jinak bude mít hodnotu *false*. Nedeterministický konečný automat budeme implementovat tak, že současně budeme procházet všemi možnými cestami. To znamená, že ve stavovém vektoru může mít více prvků hodnotu *true*, což odpovídá tomu, že automat se může nacházet ve více stavech. Pokud má nedeterministický konečný automat velké množství stavů, může být výhodné implementovat automat pomocí objektové struktury uvedené v odstavci 6.1.4 a seznam stavů, ve kterých se automat může

nacházet ve stejném okamžiku, lze implementovat pomocí spojového seznamu nebo jiné dynamické struktury.

	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$[1]$	$[2]$	$\emptyset$
$[2]$	$[3]$	$[1,2]$
$[3]$	$[4]$	$[1,3]$
$[4]$	$[5]$	$[1,4]$
$[5]$	$[1]$	$[1,5]$
$[1,2]$	$[2,3]$	$[1,2]$
$[1,3]$	$[2,4]$	$[1,3]$
$[1,4]$	$[2,5]$	$[1,4]$
$[1,5]$	$[1,2]$	$[1,5]$
$[2,3]$	$[3,4]$	$[1,2,3]$
$[2,4]$	$[3,5]$	$[1,2,4]$
$[2,5]$	$[1,3]$	$[1,2,5]$
$[3,4]$	$[4,5]$	$[1,3,4]$
$[3,5]$	$[1,4]$	$[1,3,5]$
$[4,5]$	$[1,5]$	$[1,4,5]$
$[1,2,3]$	$[2,3,4]$	$[1,2,3]$
$[1,2,4]$	$[2,3,5]$	$[1,2,4]$
$[1,2,5]$	$[1,2,3]$	$[1,2,5]$
$[1,3,4]$	$[2,4,5]$	$[1,3,4]$
$[1,3,5]$	$[1,2,4]$	$[1,3,5]$
$[1,4,5]$	$[1,2,5]$	$[1,4,5]$
$[2,3,4]$	$[3,4,5]$	$[1,2,3,4]$
$[2,3,5]$	$[1,3,4]$	$[1,2,3,5]$
$[2,4,5]$	$[1,3,5]$	$[1,2,4,5]$
$[3,4,5]$	$[1,4,5]$	$[1,3,4,5]$
$[1,2,3,4]$	$[2,3,4,5]$	$[1,2,3,4]$
$[1,2,3,5]$	$[1,2,3,4]$	$[1,2,3,5]$
$[1,2,4,5]$	$[1,2,3,5]$	$[1,2,4,5]$
$[1,3,4,5]$	$[1,2,4,5]$	$[1,3,4,5]$
$[2,3,4,5]$	$[1,3,4,5]$	$[1,2,3,4,5]$
$[1,2,3,4,5]$	$[1,2,3,4,5]$	$[1,2,3,4,5]$

6.2.1 Univerzální algoritmus používající tabulku přechodů

```
program  NKA(INPUT, OUTPUT);
type TYPSTAV = (Q0, Q1, ..., QN);
     TYPSYMBOL = 1 .. |T|;

var SYMBOL: TYPSYMBOL;
    STAV, NSTAV: ARRAY [Q0..QN] of BOOLEAN;
    TABULKAPŘECHODŮ: ARRAY [Q0..QN, TYPSYMBOL] of set of TYPSTAV;
    PŘIJAT, DEFINOVÁN: BOOLEAN;
    PŘECHODY, F: SET OF TYPSTAV;
    I, J: TYPSTAV;

procedure DALŠÍSYMBOL(var X: TYPSYMBOL); external;
procedure ČTENÍTABULKYPŘECHODŮ; external;

begin
   ČTENÍTABULKYPŘECHODŮ;
   F := [QF1, QF2, ..., QFK];
   STAV[Q0] := TRUE;
   for I := Q1 to QN do STAV[I] := FALSE;
   DEFINOVÁN := TRUE;
   while not EOF(INPUT) and DEFINOVÁN do
      begin
         DALŠÍSYMBOL(SYMBOL);
         for I := Q0 to QN do NSTAV[I] := FALSE;
         for I:= Q0 to QN do
            if STAV[I] then
               begin
                  PŘECHODY := TABULKAPŘECHODŮ[I, SYMBOL];
                  for J := Q0 to QN do
                     if J in PŘECHODY then NSTAV[J] := TRUE;
               end;
         DEFINOVÁN := FALSE;
         for I := Q0 to QN do DEFINOVÁN := DEFINOVÁN or NSTAV[I];
         STAV := NSTAV;
      end;

   PŘIJAT := FALSE;
   for I := Q0 to QN do PŘIJAT := PŘIJAT or (I in F and STAV[I]);
      if PŘIJAT then WRITE(OUTPUT, 'VSTUP PŘIJAT')
                else WRITE(OUTPUT, 'VSTUP NEPŘIJAT');
end.
```

V uvedeném programu jsou použity dva stavové vektory STAV a NSTAV.

Důvodem k tomu je skutečnost, že při výpočtu nové hodnoty stavového vektoru je po celou dobu nutno uchovat hodnotu starého stavového vektoru. Činnost automatu může skončit ze dvou důvodů. Ukončení vstupního řetězu je důvod k normálnímu ukončení práce automatu. Ukončení při nedefinovaném stavu znamená, že automat neumožňuje přechod do žádného stavu, tj. stavový vektor má všechny hodnoty *false*.

Uvedený program představuje univerzální implementaci pro libovolný konečný automat.

6.2.2 Specifický algoritmus bez použití tabulky přechodů

Druhá metoda implementace nedeterministického konečného automatu, při které je automat implementován jako program bez použití tabulky přechodů vede samozřejmě na různé programy pro různé automaty. Proto uvedeme implementaci automatu z příkladu 6.1.

```
program NKA(INPUT,OUTPUT);

type TYPSTAV = (JEDNA,DVĚ,TŘI,ČTYŘI,PĚT);
     TYPSYMBOL = (A,B);

var SYMBOL: TYPSYMBOL;
    STAV, NSTAV:array TYPSTAV of BOOLEAN;
    I: TYPSTAV;
    DEFINOVÁN: BOOLEAN;

procedure DALŠÍSYMBOL(var X: TYPSYMBOL); external;

begin
   STAV[JEDNA] := TRUE;
   for I := DVĚ to PĚT do STAV[I] := FALSE;
   DEFINOVÁN := TRUE;
   while not EOF(INPUT) and DEFINOVÁN do
      begin
         DALŠÍSYMBOL(SYMBOL);
         for I := JEDNA to PĚT do NSTAV := FALSE;
         if STAV[JEDNA] then
            case SYMBOL of
               A : NSTAV[DVĚ] := TRUE;
               B :
            end;
         if STAV[DVĚ] then
            case SYMBOL of
               A : NSTAV[TŘI] := TRUE;
```

```
            B : begin
                   NSTAV[JEDNA] := TRUE;
                   NSTAV[DVĚ] := TRUE;
                end;
         if STAV[TŘI] then
            case SYMBOL of
               A : NSTAV[ČTYŘI] := TRUE;
               B : begin
                      NSTAV[JEDNA] := TRUE;
                      NSTAV[TŘI] := TRUE
                   end;
            end;
         if STAV[ČTYŘI] then
            case SYMBOL of
               A : NSTAV[PĚT] := TRUE;
               B : begin
                     NSTAV[JEDNA] := TRUE;
                     NSTAV[ČTYŘI] := TRUE;
                   end;
            end;
         if STAV[PĚT] then
            case SYMBOL of
               A : NSTAV[JEDNA] := TRUE;
               B : begin
                      NSTAV[JEDNA] := TRUE;
                      NSTAV[PĚT] := TRUE
                   end;
            end;
         DEFINOVÁN := NSTAV[JEDNA] or NSTAV[DVĚ] or NSTAV[TŘI]
                      or NSTAV[ČTYŘI] or NSTAV[PĚT];
         STAV := NSTAV
      end;
   if STAV[JEDNA] then WRITE (OUTPUT, 'ŘETĚZ PŘIJAT')
                  else WRITE (OUTPUT, 'ŘETĚZ NEPŘIJAT')
end.
```

7 Jednoznačné a nejednoznačné gramatiky

7.1 Základní pojmy

Bezkontextová gramatika je taková gramatika, která má všechna pravidla ve tvaru $A \to \alpha$, kde $A \in N, \alpha \in (N \cup T)^*$.

Gramatika G je *nejednoznačná (víceznačná)*, jestliže existuje věta $w \in L(G)$ taková, že pro ni existují nejméně dva různé derivační stromy. V opačném případě je gramatika *jednoznačná*. *„Hustá“* nejednoznačnost je taková, kdy počet derivačních stromů roste exponenciálně nebo rychleji vzhledem k délce vstupního řetězce.

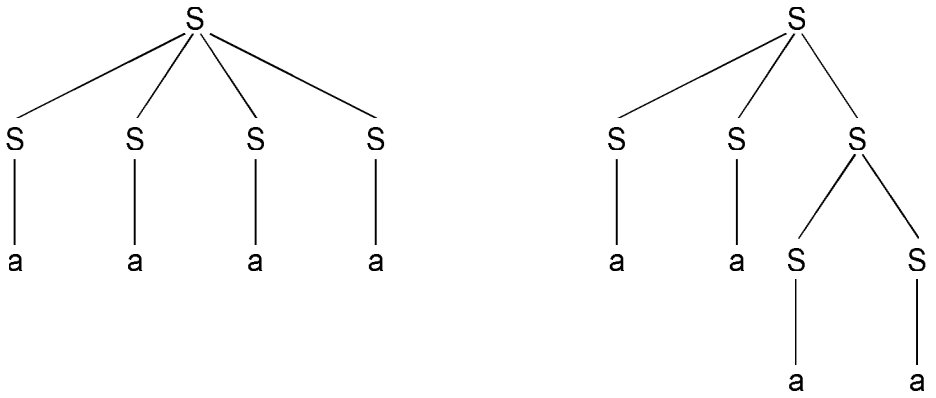
Gramatika $G = (N, T, P, S)$ obsahuje *cyklus*, je-li v ní možná derivace $A \Rightarrow^+ A$ pro nějaké $A \in N$.

7.2 Hustá a nekonečná nejednoznačnost

Příklad 7.1
Je dána gramatika $G = (\{S\}, \{a\}, P, S)$, kde P obsahuje pravidla:

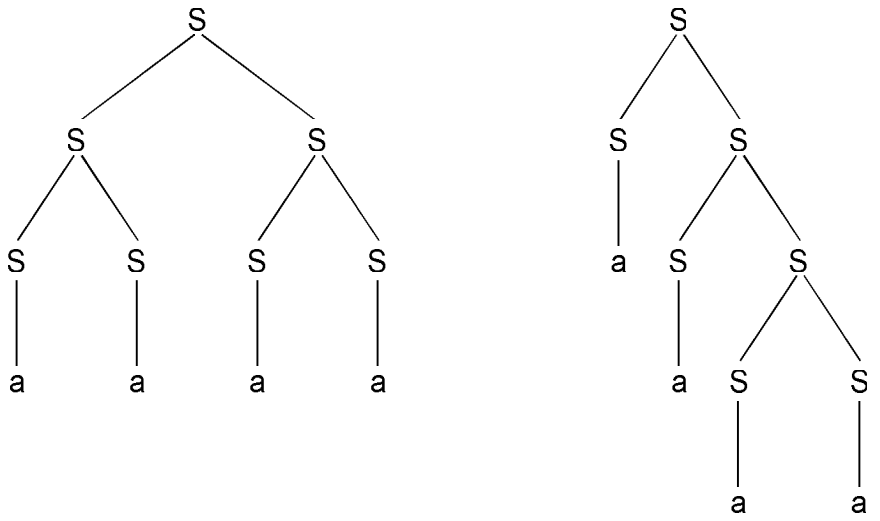
$S \to SSSS$
$S \to SSS$
$S \to SS$
$S \to a$.

Vytvoříme několik derivačních stromů pro řetězec *aaaa* (obr. 7.1 a 7.2):



Obrázek 7.1: Derivační stromy pro řetězec *aaaa*

Pokuste se sestrojit všechny derivační stromy pro řetězec *aaaa*. Je jich konečný počet. Tento typ nejednoznačnosti je příkladem husté nejednoznačnosti.



Obrázek 7.2: Derivační stromy pro řetězec *aaaa*

Příklad 7.2
Je dána gramatika $G = (\{S\}, \{a\}, \{S \rightarrow S, S \rightarrow a\}, S)$. Tato gramatika generuje jediný řetězec a, ale nekonečně mnoha způsoby:

$S \Rightarrow a$
$S \Rightarrow S \Rightarrow a$
$S \Rightarrow S \Rightarrow S \Rightarrow a$
$\vdots$

Důvodem tohoto jevu je cyklus v gramatice způsobený pravidlem $S \rightarrow S$. Cyklus může být způsoben i jinými konstrukcemi.

Příklad 7.3
Je dána gramatika $G = (\{S\}, \{a\}, P, S)$, kde P obsahuje pravidla:

$S \rightarrow SS$
$S \rightarrow a$
$S \rightarrow \varepsilon$.
Řetězec a může být generován opět nekonečně mnoha způsoby:

$S \Rightarrow a$
$S \Rightarrow SS \Rightarrow S \Rightarrow a$
$S \Rightarrow SS \Rightarrow S \Rightarrow SS \Rightarrow S \Rightarrow a$
$\vdots$

Je zřejmé, že v této gramatice je opět cyklus, ale není přímo vidět z pravidel. Tomuto typu nejednoznačnosti se říká nekonečná nejednoznačnost a je způsobena vždy cyklem v gramatice.

7.3 Asymetrické závorkové struktury

Asymetrické závorkové struktury jsou takové, kde počet otevíracích a zavíracích závorek není stejný. Budeme se zabývat těmito dvěma strukturami:

$L_o = \{a^n b^m : n \geq m \geq 0\}$,

$L_z = \{a^n b^m : m \geq n \geq 0\}$.

Jazyk L_o obsahuje řetězce, kde počet otevíracích závorek může být větší než počet zavíracích závorek. V jazyce L_z je to naopak.

Příklad 7.4
Pro generování jazyka L_z můžeme navrhnout gramatiku
$G_{1z} = (\{S, A\}, \{a, b\}, P_{1z}, S)$, kde P_{1z} obsahuje pravidla:

$S \to ASb$
$S \to \varepsilon$
$A \to a$
$A \to \varepsilon$.

Řetězec abb můžeme generovat těmito dvěma způsoby:

$S \Rightarrow ASb \Rightarrow AASbb \Rightarrow aASbb \Rightarrow aSbb \Rightarrow abb$
$S \Rightarrow ASb \Rightarrow AASbb \Rightarrow ASbb \Rightarrow aSbb \Rightarrow abb$

V první derivaci odpovídá a druhému b, ve druhé derivaci odpovídá a prvnímu b. Je zřejmé, že gramatika G_{1z} je nejednoznačná.

Pro generování jazyka L_z můžeme sestrojit gramatiky, které jsou jednoznačné. Použijeme tyto postupy:

1. Řetězce v jazyce L_z zapíšeme takto:

   $L_z = \{a^n b^n b^{m-n} : m \geq n \geq 0\}$.

   Potom sestrojíme gramatiku $G_{2z} = (\{S, A, B\}, \{a, b\}, P_{2z}, S)$, kde P_{2z} obsahuje pravidla:

   $S \to AB$
   $A \to aAb$
   $A \to \varepsilon$
   $B \to bB$
   $B \to \varepsilon$.

   Tato gramatika je jednoznačná a symbolům a odpovídá prvních n symbolů b.

2. Řetězce v jazyce L_z zapíšeme takto:

   $L_z = \{a^n b^{m-n} b^n : m \geq n \geq 0\}$.

Potom sestrojíme gramatiku $G_{3z} = (\{S, A, B\}, \{a, b\}, P_{z3}, S)$, kde P_{3z} obsahuje pravidla:

$S \to A$
$S \to \varepsilon$
$A \to aAb$
$A \to B$
$B \to bB$
$B \to \varepsilon$.

Tato gramatika je opět jednoznačná a symbolům a odpovídá posledních n symbolů b.

Příklad 7.5
Pro generování jazyka L_o můžeme navrhnout gramatiku $G_{1o} = (\{S, B\}, \{a, b\}, P_{1o}, S)$, kde P_{1o} obsahuje pravidla:

$S \to aSB$
$S \to \varepsilon$
$B \to b$
$B \to \varepsilon$.

Řetězec aab můžeme generovat těmito dvěma způsoby:

$S \Rightarrow aSB \Rightarrow aaSBB \Rightarrow aaSbB \Rightarrow aaSb \Rightarrow aab$
$S \Rightarrow aSB \Rightarrow aaSBB \Rightarrow aaSBb \Rightarrow aaSb \Rightarrow aab$

V první derivaci odpovídá b druhému a, ve druhé derivaci odpovídá b prvnímu a. Je opět zřejmé, že gramatika G_{1o} je nejednoznačná.

Pro generování jazyka L_o můžeme sestrojit jednoznačné gramatiky těmito postupy:

1. Řetězce v jazyce L_o zapíšeme takto:

   $L_o = \{a^{n-m}a^m b^m : n \geq m \geq 0\}$.

   Potom sestrojíme gramatiku $G_{2o} = (\{S, A, B\}, \{a, b\}, P_{2o}, S)$, kde P_{2o} obsahuje pravidla:

   $S \to AB$
   $A \to aA$
   $A \to \varepsilon$
   $B \to aBb$
   $B \to \varepsilon$.

   Tato gramatika je jednoznačná a symbolům b odpovídá posledních m symbolů a.

   Poznámka: Pro tuto gramatiku neexistuje deterministický syntaktický analyzátor. Je to způsobeno tím, že se nedá poznat, kde končí předpona a^{n-m}, protože počet symbolů b (m) je znám až na konci celého řetězce.

Ekvivalentní gramatika, pro kterou existuje deterministický syntaktický analyzátor je $G_{3o} = (\{S, A, B\}, \{a, b\}, P_{3o}, S)$, kde P_{3o} obsahuje pravidla:

$S \to A$
$A \to aA$
$A \to B$
$B \to aBb$
$B \to \varepsilon$.

2. Řetězce v jazyce L_o zapíšeme takto:

   $L_o = \{a^m a^{n-m} b^m : n \geq m \geq 0\}$.

   Potom sestrojíme gramatiku $G_{4o} = (\{S, A\}, \{a, b\}, P_{4o}, S)$, kde P_{4o} obsahuje pravidla:

   $S \to \varepsilon$
   $S \to aSb$
   $S \to A$
   $A \to aA$
   $A \to \varepsilon$

   Tato gramatika je jednoznačná a symbolům b odpovídá prvních m symbolů a. Ze stejných důvodů jako pro gramatiku G_{2o} pro ni neexistuje deterministický syntaktický analyzátor.

Jazyk L_o je speciální případ konstrukce známé pod názvem *„dangling“* **else**. Jedná se o případ, kdy lze zapsat příkazy:

if BV **then** P
if BV **then** P_1 **else** P_2

Vzhledem k tomu, že pouze pro gramatiku G_{3o} existuje deterministický syntaktický analyzátor, je ve všech jazycích, kde je tato konstrukce možná, použito pravidlo: Symbol **else** patří k nejbližšímu předcházejícímu **then**.

7.4 Podstatně nejednoznačné jazyky

Pro některé jazyky není možné sestrojit žádnou jednoznačnou gramatiku. Takové jazyky se nazývají podstatně nejednoznačné.

Příklad 7.6
Je dán jazyk $L = \{a^m b^n c^p : m, n, p \geq 0, m = n \text{ nebo } n = p\}$. Gramatika, která generuje tento jazyk je $G = (\{S, A, C, X, Y\}, \{a, b, c\}, P, S)$ s pravidly:

S	$\to$	XC	Y	$\to$	bYc	C	$\to$	cC
S	$\to$	AY	Y	$\to$	ε	C	$\to$	ε
X	$\to$	aXb	A	$\to$	aA			
X	$\to$	ε	A	$\to$	ε			

Pravidlo $S \rightarrow XC$ představuje předpis pro generování řetězců, kde $m = n$. Neterminální symbol X generuje symetrickou závorkovou strukturu $\{a^n b^n : n \geq 0\}$. Symbol C generuje libovolný řetězec složený ze symbolů c. Podobně v pravidle $S \rightarrow AY$ symbol Y generuje závorkovou strukturu $\{b^n c^n : n \geq 0\}$ a symbol A generuje řetězec složený ze symbolů a. Pro řetězec, ve kterém $m = n = p$, existují dva způsoby derivace. Ukažme si je pro $n = 2$. Řetězec $a^2b^2c^2$ můžeme generovat těmito dvěma způsoby:

1. $S \Rightarrow XC \Rightarrow aXbC \Rightarrow aaXbbC \Rightarrow aabbC \Rightarrow aabbcC \Rightarrow aabbccC \Rightarrow aabbcc$

2. $S \Rightarrow AY \Rightarrow aAY \Rightarrow aaAY \Rightarrow aaY \Rightarrow aabYC \Rightarrow aabbYcc \Rightarrow aabbcc$

Pro tento jazyk jednoznačná gramatika neexistuje.

7.5 Odstranění nejednoznačností v gramatice

Obecně neexistuje algoritmus, který by nejednoznačnou gramatiku transformoval na jednoznačnou. Z příkladu 7.6 je vidět, že to někdy ani není možné. Pro některé případy je naopak odstranění nejednoznačnosti možné a je znám způsob, jak to udělat. Uvedeme některé příklady.

Algoritmus 7.7
Nalezení a odstranění cyklu z bezkontextové gramatiky
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Bezkontextová gramatika $G' = (N', T, P', S')$ bez cyklů taková, že $L(G) = L(G')$.
Metoda:

1. Z gramatiky $G = (N, T, P, S)$ vyloučíme ε-pravidla a dostaneme gramatiku $G_1 = (N', T, P_1, S')$.

2. Pro všechny neterminální symboly $A \in N'$ sestrojíme v G_1 množiny $N_A^+ = \{B : A \Rightarrow^+ B, A \in N', B \in N'\}$. V derivacích tvaru $A \Rightarrow^+ B$ se mohou vyskytnout pouze jednoduchá pravidla.

3. Jestliže $A \in N_A^+$, pak pro symbol A je v gramatice cyklus. Tento cyklus odstraníme tak, že provedeme vyloučení jednoduchých pravidel pro všechny neterminální symboly z N_A^+ pomocí Algoritmu 2.6 [JPR03].

4. Krok 3 provedeme pro všechny neterminální symboly.

5. Výsledná gramatika je $G' = (N', T, P', S')$, kde P' vznikne z množiny P_1 náhradou pravidel podle bodu 3.

Příklad 7.8
Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to A \mid b$
$A \to Aa \mid B$
$B \to Bb \mid S$

Pomocí algoritmu 7.7 zjistíme, že $N_S^+ = \{A, B, S\}$. Protože $S \in N_S^+$ je v gramatice G cyklus. Cyklus odstraníme vyloučením jednoduchých pravidel pro neterminální symboly A, B, S. Dostaneme gramatiku $G' = (\{A, B, S\}, \{a, b\}, P_1, S)$, kde P_1 obsahuje pravidla:

$S \to b \mid Aa \mid Bb$
$A \to Aa \mid Bb \mid b$
$B \to Bb \mid b \mid Aa$

Tato gramatika neobsahuje cyklus.

Příklad 7.9
Je dána gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

$S \to ABS \mid a$
$A \to bA \mid \varepsilon$
$B \to cB \mid \varepsilon$

Pomocí algoritmu 7.7 zjistíme, zda je v této gramatice cyklus a v případě, že ano, odstraníme jej.

1. Vyloučíme ε-pravidla z G a dostaneme $G = (\{S, A, B\}, \{a, b, c\}, P_1, S)$, s pravidly v P_1:

   $S \to ABS \mid BS \mid AS \mid S \mid a$
   $A \to bA \mid b$
   $B \to cB \mid c$

2. Určíme $N_S^+ = \{S\}$, $N_A^+ = \emptyset$ a $N_B^+ = \emptyset$.

3. Protože $S \in N_S^+$, je v gramatice cyklus pro S, který odstraníme vypuštěním pravidla $S \to S$.

5. Výsledná gramatika je $G' = (\{S, A, B\}, \{a, b, c\}, P', S)$, kde množina P' obsahuje pravidla:

   $S \to ABS \mid BS \mid AS \mid a$
   $A \to bA \mid b$
   $B \to cB \mid c$

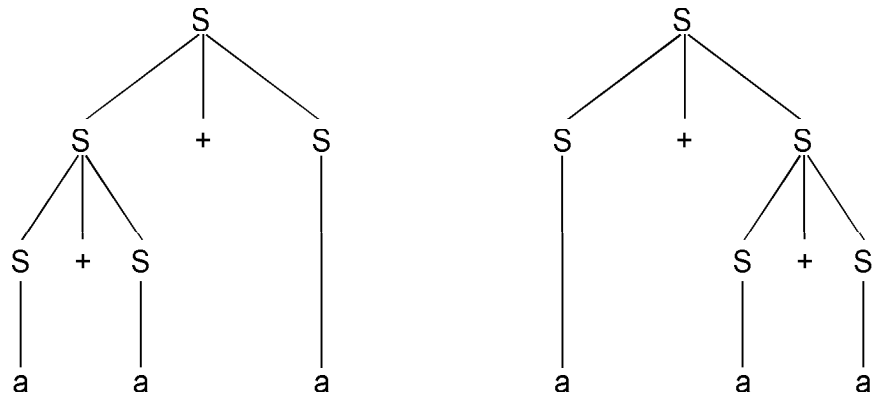
Jiný typ nejednoznačnosti je způsoben tím, že určitý neterminální symbol je současně levě i pravě rekurzivní. To může být způsobeno pravidlem tvaru $S \to S\,\alpha\,S$.

Příklad 7.10
Je dána gramatika $G = (\{S\}, \{a, +\}, P, S)$ s pravidly v P:

$S \to S + S$

$S \to a$

Pro řetězec $a + a + a$ existují dva derivační stromy uvedené na obr. 7.3.



Obrázek 7.3: Derivační stromy pro řetězce $a + a + a$

Levý derivační strom chápe výraz jako „levě asociativní“, tj. že by se výraz měl vyhodnocovat zleva doprava a pravý derivační strom chápe výraz jako „pravě asociativní“, tj. že by se měl vyhodnocovat zprava doleva.

Odstranění nejednoznačnosti je možné dosáhnout těmito postupy:

1. Dosadíme za první S do pravidla $S \to S\ + S$ ze druhého pravidla. Dostaneme gramatiku G_1 s pravidly:

   $S \to a\ + S$
   $S \to a$

   Tato gramatika generuje jen „pravě asociativní“ výrazy.

2. Obdobně provedeme dosazení za druhé S v pravidle $S \to S\ + S$. Dostaneme gramatiku G_2 s pravidly:

   $S \to S\ + a$
   $S \to a$

   Tato gramatika generuje jen „levě asociativní“ výrazy.

7.6 Příklady pro cvičení

Cvičení 7.11
Ukažte, že gramatika s pravidly tvaru $A \to \alpha A \mid \alpha A \beta A$ je nejednoznačná.

Cvičení 7.12
Odstraňte nejednoznačnost z gramatik v příkladech 7.2 a 7.3.

Cvičení 7.13
Navrhněte postup, jak gramatiku $G = (\{S\}, \{a, b\}, \{S \to SS \mid aa \mid b\}, S)$ převést na gramatiku jednoznačnou.

Cvičení 7.14
Sestrojte gramatiky pro generování obecné závorkové struktury, kde počet otevíracích (zavíracích) závorek je větší než počet zavíracích (otevíracích) závorek. Zjistěte, které z těchto gramatik jsou nejednoznačné a uveďte proč.

Cvičení 7.15
Je dána gramatika $G = (\{S\}, \{a, b\}, \{S \to aSb \mid SS \mid \varepsilon\}, S)$. Ukažte, že tato gramatika je nejednoznačná. Najděte ekvivalentní jednoznačnou gramatiku.

8 Transformace bezkontextových gramatik

8.1 Základní pojmy

V této kapitole uvedeme příklady transformace gramatik. Připomeneme si transformační algoritmy a ukážeme si některé jejich vlastnosti. Algoritmus, který určuje, zda jazyk generovaný gramatikou je prázdný, přeformulujeme tak, že nalezne množinu neterminálních symbolů N_t, ze kterých lze generovat terminální řetězce.

Jednoduché pravidlo je pravidlo tvaru $A \to B$, kde A, B jsou neterminální symboly, ε–pravidlo je pravidlo tvaru $A \to \varepsilon$.

Bezkontextová gramatika $G = (N, T, P, S)$ je *bez* $\varepsilon - pravidel$, když

1. P neobsahuje žádná ε-pravidla, nebo
2. P obsahuje jediné ε-pravidlo tvaru $S \to \varepsilon$ a S se nevyskytuje na pravé straně žádného pravidla v P.

Neterminální symbol A v bezkontextové gramatice $G = (N, T, P, S)$ nazveme *rekurzivní*, jestliže existuje derivace $A \Rightarrow^+ \alpha A\ \beta$ pro nějaké α a $\beta \in (N \bigcup T)^*$. Jestliže $\alpha = \varepsilon$, pak A nazveme *symbol rekurzivní zleva*, podobně jestliže $\beta = \varepsilon$, pak A nazveme *symbol rekurzivní zprava.*
Gramatiku s alespoň jedním neterminálním symbolem rekurzivním zleva (zprava) nazveme *rekurzivní zleva (zprava).*
Gramatiku, ve které všechny neterminální symboly (s vyjímkou startovacího symbolu) jsou rekurzivní, nazveme *rekurzivní.*

Vzhledem k tomu, že rekurzivita zleva znemožňuje použití gramatiky pro některé metody syntaktické analýzy, ukážeme, že pro každý bezkontextový jazyk existuje gramatika, která není rekurzivní zleva.

Bezkontextová gramatika $G = (N, T, P, S)$ je v normálním tvaru podle Chomského, jestliže každé pravidlo v P má jeden z následujících tvarů:

1. $A \to BC$ pro $A, B, C \in N$.
2. $A \to a$ pro $a \in T, A\ \in N$.
3. Jestliže $\varepsilon \in L(G)$, pak $S \to \varepsilon$ je pravidlo v P a S se nevyskytuje na pravé straně žádného pravidla.

Ukážeme, že každý bezkontextový jazyk může být generován gramatikou v Chomského normálním tvaru.

Bezkontextová gramatika G je v normálním tvaru podle Greibachové, jestliže G je bez ε-pravidel a každé pravidlo (kromě pravidla $S \to \varepsilon$) má tvar $A \to a\ \alpha$, kde $a \in T, \alpha \in N^*$.

Algoritmus 8.1
Určení neterminálních symbolů, ze kterých lze generovat terminální řetězce.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Množina $N_t \subset N$ taková, že pro každé $A \in N_t$ platí $A \Rightarrow^+ w$, $w \in T^*$.
Metoda:

1. $N_0 := \emptyset$, $i := 1$,
2. $N_i := N_{i-1} \cup \{A : A \to \alpha \in P,\ \alpha \in (N_{i-1} \cup T)^*\}$,
3. Jestliže $N_i \neq N_{i-1}$, pak $i := i + 1$ a přejdi na krok 2,
4. $N_t := N_i$.

Poznámka:
Gramatiku, která vznikne z gramatiky $G = (N, T, P, S)$ tak, že počáteční symbol S nahradíme neterminálním symbolem $A \in N$ označíme $G_A = (N, T, P, A)$. Analyzujeme–li algoritmus 8.1, zjistíme, že pro množiny N_i platí:

$$\text{Jestliže } A \in N_i, \text{ pak existuje } A \Rightarrow^j w,\ w \in T^* \text{ a } j \leq i.$$

Jestliže vytvoříme gramatiku $G_A = (N, T, P, A)$, pro $A \in N_t$, pak $L(G_A)$ je neprázdný jazyk. Množina N_t obsahuje všechny neterminální symboly, ze kterých lze generovat terminální řetězce. Jestliže $S \in N_t$, pak $G_S = (N, T, P, S) = G$ generuje neprázdný jazyk.

8.2 Odstranění ε–pravidel a jednoduchých pravidel

Algoritmus, který provádí převod na gramatiku bez ε-pravidel (Algoritmus 2.4 [JPR03]) budeme ilustrovat na příkladu.

Příklad 8.2
Z gramatiky pro generování jazyka PL0 vybereme část pravidel pro generování deklarací, některá pravidla zjednodušíme a potom odstraníme ε-pravidla.

Po zjednodušení budou vybraná pravidla vypadat takto:

$$\begin{array}{ll}
\mathit{PROGRAM} & \to \mathit{BLOK}\ . \\
\mathit{BLOK} & \to \mathit{DEKKONST}\ \mathit{DEKPROM}\ \mathit{DEKPROC}\ \mathit{PŘÍKAZ} \\
\mathit{DEKKONST} & \to \varepsilon \mid \mathit{const} \\
\mathit{DEKPROM} & \to \varepsilon \mid \mathit{var} \\
\mathit{DEKPROC} & \to \varepsilon \mid \mathit{procedure} \\
\mathit{PŘÍKAZ} & \to \varepsilon \mid \mathit{call}
\end{array}$$

Množina $N_\varepsilon = \{A : A \in N, A \Rightarrow^* \varepsilon\}$ obsahuje tyto prvky:

$$N_\varepsilon = \{\mathit{BLOK}, \mathit{DEKKONST}, \mathit{DEKPROM}, \mathit{DEKPROC}, \mathit{PŘÍKAZ}\}.$$

Po odstranění ε-pravidel dostaneme pravidla:

PROGRAM	$\rightarrow$	*BLOK* . | .			
BLOK	$\rightarrow$	*DEKKONST*	*DEKPROM*	*DEKPROC*	*PŘÍKAZ*
	|	*DEKKONST*	*DEKPROM*	*DEKPROC*	
	|	*DEKKONST*	*DEKPROM*		*PŘÍKAZ*
	|	*DEKKONST*		*DEKPROC*	*PŘÍKAZ*
	|		*DEKPROM*	*DEKPROC*	*PŘÍKAZ*
	|	*DEKKONST*	*DEKPROM*		
	|	*DEKKONST*		*DEKPROC*	
	|	*DEKKONST*			*PŘÍKAZ*
	|		*DEKPROM*	*DEKPROC*	
	|		*DEKPROM*		*PŘÍKAZ*
	|			*DEKPROC*	*PŘÍKAZ*
	|	*DEKKONST*			
	|		*DEKPROM*		
	|			*DEKPROC*	
	|				*PŘÍKAZ*
DEKKONST	$\rightarrow$	*const*			
DEKPROM	$\rightarrow$	*var*			
DEKPROC	$\rightarrow$	*procedure*			
PŘÍKAZ	$\rightarrow$	*call*			

Z tohoto příkladu je vidět, že výsledná gramatika má o hodně více pravidel než gramatika původní. Ukažme si nyní, jaký vliv má odstranění ε-pravidel na derivace a derivační stromy. V původní gramatice můžeme generovat větu, která obsahuje pouze tečku takto:

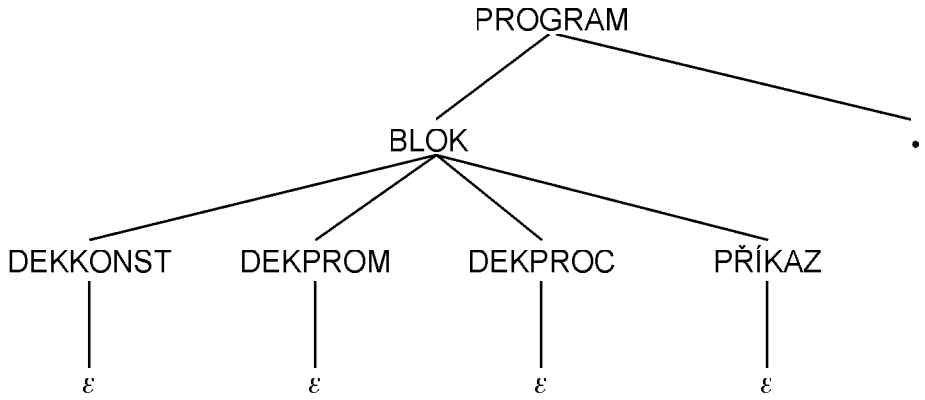
$$
\begin{aligned}
\mathit{PROGRAM} &\Rightarrow \mathit{BLOK}\ . \\
&\Rightarrow \mathit{DEKONST\ DEKPROM\ DEKPROC\ PŘÍKAZ}\ . \\
&\Rightarrow \mathit{DEKONST\ DEKPROM\ DEKPROC}\ . \\
&\Rightarrow \mathit{DEKONST\ DEKPROM}\ . \\
&\Rightarrow \mathit{DEKONST}\ . \\
&\Rightarrow\ .
\end{aligned}
$$

Této derivaci odpovídá derivační strom na obr. 8.1. V transformované gramatice bude podobná derivace vypadat takto:

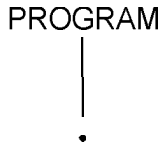
$$\mathit{PROGRAM} \Rightarrow\ .$$

a příslušný derivační strom je na obr. 8.2. Z tohoto příkladu je vidět, že převod na gramatiku bez ε–pravidel má tyto vlastnosti:

1. Zvyšuje počet pravidel gramatiky.
2. Zachovává počet neterminálních symbolů.
3. Zkracuje derivace a tím i zjednodušuje derivační stromy.



Obrázek 8.1: Derivační strom pro derivaci z příkladu 8.2



Obrázek 8.2: Derivační strom pro derivaci z příkladu 8.2 po odstranění ε–pravidel

Algoritmus 2.6 [JPR03], který provádí vyloučení jednoduchých pravidel budeme opět ilustrovat na příkladu.

Příklad 8.3
Je dána gramatika $G = (\{S, A, B, C, D\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla:

$S \to A \mid B$
$A \to C \mid aA \mid bS$
$C \to bC \mid a$
$B \to D \mid cB \mid dS$
$D \to dD \mid c$

Odstraňme z této gramatiky jednoduchá pravidla.
Nejdříve určíme $N_S = \{S, A, B, C, D\}$. To znamená, že existují derivace:

$S \Rightarrow A \Rightarrow C$
$S \Rightarrow B \Rightarrow D$

Dále určíme:

$N_A = \{A, C\}$
$N_B = \{B, D\}$
$N_C = \{C\}$
$N_D = \{D\}$

Výsledná gramatika je:

$$G' = (\{S, A, B, C, D\}, \{a, b, c, d\}, P', S),$$

kde P' bude obsahovat pravidla:
$S \to aA \mid bS \mid bC \mid a \mid cB \mid dS \mid dD \mid c$
$A \to aA \mid bS \mid bC \mid a$
$B \to cB \mid dS \mid dD \mid c$
$C \to bC \mid a$
$D \to dD \mid c$
V gramatice G existují například tyto derivace:
$S \Rightarrow A \Rightarrow C \Rightarrow a$
$S \Rightarrow B \Rightarrow D \Rightarrow c$,
které jsou v gramatice G' zkráceny na:
$S \Rightarrow a$
$S \Rightarrow c$.
Stejně jako v případě převodu na gramatiku bez ε-pravidel má operace vyloučení jednoduchých pravidel z gramatiky tyto vlastnosti:

1. Zvyšuje počet pravidel gramatiky.
2. Zachovává počet neterminálních symbolů.
3. Zkracuje derivace a tím i zjednodušuje derivační stromy.

Algoritmy pro vyloučení ε-pravidel a jednoduchých pravidel z gramatiky lze použít pro odstranění cyklu z gramatiky. Tento postup je uveden v algoritmu 7.7.

8.3 Odstranění levé rekurze

Velmi důležitou transformací je odstranění levé rekurze z bezkontextové gramatiky. Důležitost této transformace spočívá v tom, že pro gramatiky s levou rekurzí nelze sestrojit syntaktický analyzátor pracující metodou shora dolů.

Připomeňme si nejdříve základní pojmy:
V bezkontextové gramatice G je levá rekurze tehdy, když v ní je možná derivace tvaru

$$A \Rightarrow^+ A\alpha.$$

Nejjednodušší případ levé rekurze je její existence přímo v pravidlech gramatiky tvaru

$$A \to A\alpha.$$

Připomeňme, že kromě pravidel typu $A \to A\alpha$ musí být v gramatice pravidla typu $A \to \beta$, kde β je řetězec, ve kterém se nevyskytuje A. Kdyby pravidlo $A \to \beta$ v gramatice nebylo, nebylo by možné generovat ze symbolu A terminální

řetězce. Takovouto levou rekurzivitu můžeme z gramatiky odstranit takto:
Místo pravidel tvaru:

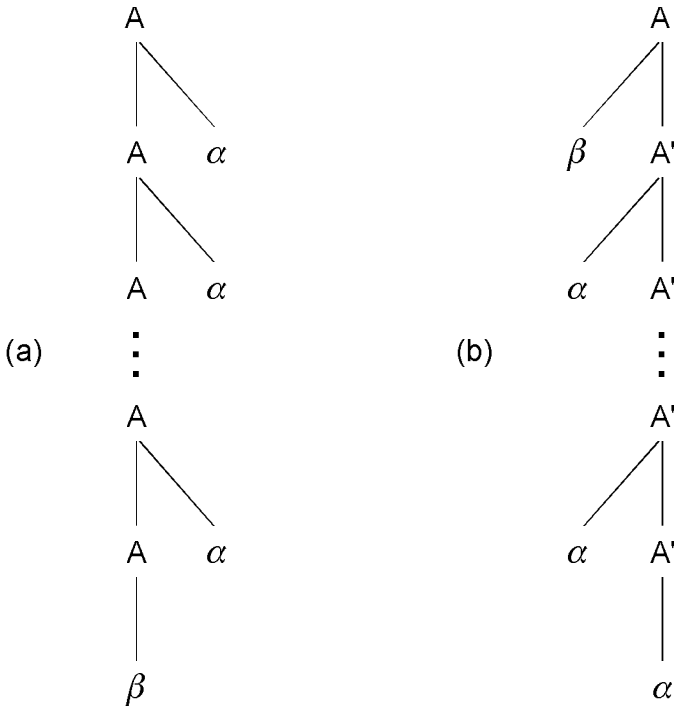
$A \to A\alpha \mid \beta$

vložíme do gramatiky pravidla tvaru:

$A \to \beta A' \mid \beta$

$A \to \alpha A' \mid \alpha$

Na obr. 8.3 jsou uvedeny derivační stromy pro řetězec $\beta\alpha\alpha\ldots\alpha$ v původní gramatice (a) a v gramatice po odstranění levé rekurze (b).



Obrázek 8.3: Derivační stromy pro řetězec $\beta\alpha\alpha\ldots\alpha$

Je zřejmé, že odstranění levé rekurze se vlastně provádí tak, že levá rekurze se transformuje na pravou rekurzi. V transformované gramatice je totiž možná derivace, ve které se projeví pravá rekurze:

$$A' \Rightarrow^k \alpha^k A', \quad k \geq 1.$$

Důvod tohoto jevu spočívá v tom, že rekurze v libovolné podobě je nutnou podmínkou pro to, aby gramatika generovala nekonečný jazyk. K tomu, aby gramatika G generovala nekonečný jazyk je nutné splnění dvou podmínek:

1. Existence rekurze alespoň u jednoho neterminálního symbolu (označme jej A),
2. symbol A nesmí být v gramatice G zbytečný.

Uvedený způsob odstranění levé rekurze má tu nevýhodu, že nově vzniklá pravidla:

$A \to \beta A' \mid \beta$

$A \to \alpha A' \mid \alpha$

způsobují, že gramatika po takové transformaci není $LL(1)$. Důvodem je kolize FIRST – FIRST (viz kap.10) v pravidlech pro A a A'. Z tohoto důvodu je navržen jiný způsob odstranění levé rekurze, který je založen na této transformaci:

Místo pravidel tvaru

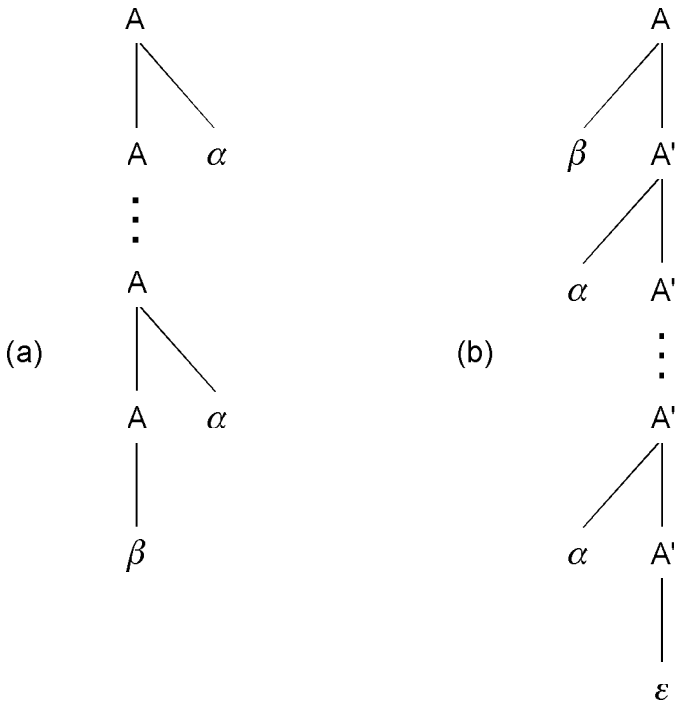
$A \to A\alpha \mid \beta$

vložíme do gramatiky pravidla:

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

Opět vidíme, že levá rekurze se transformuje na pravou. Derivační stromy pro řetězec $\beta\alpha \ldots \alpha$ v původní (a) a transformované (b) gramatice jsou na obr. 8.4.



Obrázek 8.4: Derivační stromy pro řetězec $\beta\alpha\alpha \ldots \alpha$

Levá rekurze může být v gramatice způsobena i jiným způsobem než pravidly tvaru $A \to A\alpha$. V dalším příkladu si ukážeme jiný případ levé rekurze a způsob jejího odstranění.

Příklad 8.4
Je dána gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:
$S \to ASa \mid b$
$A \to bA \mid \varepsilon$
V této gramatice je levá rekurze, protože je možná derivace:

$$S \Rightarrow ASa \Rightarrow Sa$$

Tato levá rekurze je způsobena pravidlem $S \to ASa$, protože ze symbolu A může generovat prázdný řetězec ($A \Rightarrow \varepsilon$). Tento typ levé rekurze se nazývá skrytá levá rekurze. Postup, jak odstranit tuto levou rekurzi, spočívá ve dvou krocích:

1. Odstranit ε-pravidla, která se vyskytnou v derivaci $A \Rightarrow^+ \varepsilon$.
2. Z výsledné gramatiky odstranit levou rekurzi.

Provedeme tuto transformaci pro zadanou gramatiku.

1. Po odstranění pravidla $A \to \varepsilon$ dostaneme gramatiku s pravidly:
   $S \to Sa \mid ASa \mid b$
   $A \to bA \mid b$
   V této gramatice již pravidlo $S \to ASa$ není levé rekurzivní. Gramatika je připravena pro odstranění levé rekurze podle Věty 3.26 [JPR03]. Pravidla pro neterminální symbol S můžeme schematicky zapsat:
   $S \to S\alpha \mid \beta_1 \mid \beta_2$.
   Všimněme si, že $\beta_1 = ASa$ obsahuje symbol S, pro který budeme odstraňovat levou rekurzi.
2. Po odstranění levé rekurze dostaneme novou gramatiku
   $G' = (\{S, A, S'\}, \{a, b\}, P', S)$,
   kde P' obsahuje pravidla:
   $S \to ASaS' \mid bS'$
   $S' \to aS' \mid \varepsilon$
   $A \to bA \mid b$

Příklad 8.5
Je dána gramatika $G = (\{S, A\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:
$S \to Aa \mid a$
$A \to Sb \mid b$
Tato gramatika je levě rekurzivní, protože je v ní možná derivace:

$$S \Rightarrow Aa \Rightarrow Sba.$$

Tato levá rekurze je způsobena existencí dvojice pravidel
$S \to Aa$, $A \to Sb$.
Postup odstranění takové rekurze, která je způsobena více než jedním pravidlem, spočívá ve dvou krocích:

1. Dosazováním do pravidel tvaru $A \to B\alpha$ za B tak, aby se levá rekurze projevila v jednom pravidle.

2. Odstraněním levé rekurze z levě rekurzivního pravidla.

Pro zadanou gramatiku bude odstranění levé rekurze probíhat takto:

1. Dosadíme do pravidla:
   $S \to Aa$
   za neterminální symbol A z pravidel:
   $A \to Sb \mid b$
   a dostaneme pravidla:
   $S \to Sba \mid ba$
   Po tomto dosazení získáme gramatiku
   $G_1 = (\{S\}, \{a, b\}, P_1, S)$,
   kde P_1 obsahuje pravidla
   $S \to Sba \mid ba \mid a$
   protože symbol A se stal nedostupným.

2. Po odstranění levé rekurze dostaneme:
   $G_2 = (\{S, S'\}, \{a, b\}, P_2, S)$,
   kde P_2 obsahuje pravidla:
   $S \to aS' \mid baS'$
   $S' \to baS' \mid \varepsilon$

V učebním textu [JPR03] je uveden algoritmus (Algoritmus 3.28), který provádí vyloučení rekurzivity zleva z gramatiky ve všech případech. Tento algoritmus předpokládá, že vstupní gramatika je vlastní, to znamená, že je bez cyklů, bez zbytečných symbolů a bez ε-pravidel. Poslední předpoklad, tj. že vstupní gramatika je bez ε-pravidel je zbytečně silný. Ukážeme si na příkladu, že někdy přítomnost ε-pravidel nevadí při odstranění rekurzivity zleva.

Příklad 8.6
Je dána gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:
$S \to Aa \mid B$
$A \to Sb \mid \varepsilon$
$B \to Bc \mid \varepsilon$
Při použití algoritmu na vyloučení rekurzivity zleva, (Algoritmus 3.28) učebního textu (Jazyky a překlady) bude postup vypadat takto:

1. Zavedeme uspořádání:
   $N = \{S, A, B\}$, $i := 1$.

2. Beze změn.

3. $i := 2$, $j = 1$.

4. Dosadíme do pravidla $A \to Sb$ z pravidel $S \to Aa \mid B$ a dostaneme pro A pravidla:
   $A \to Aab \mid Bb \mid \varepsilon$.

2. Vyloučíme rekurzivitu zleva v pravidlech $A \to Aab \mid Bb \mid \varepsilon$ a dostaneme:
   $A \to BbA' \mid A'$
   $A' \to abA' \mid \varepsilon$

3. $i := 3$, $j := 1$.

4. Beze změn.

5. $j := 2$.

4. Beze změn.

2. Vyloučíme rekurzivitu zleva v pravidlech $B \to Bc \mid \varepsilon$ a dostaneme:
   $B \to B'$
   $B' \to cB' \mid \varepsilon$

Výsledná gramatika má potom tvar:
$G' = (\{S, A, A', B, B'\}, \{a, b, c\}, P', S)$,
kde P' obsahuje pravidla:
$S \to Aa \mid B$
$A \to BbA' \mid A'$
$A' \to abA' \mid \varepsilon$
$B \to B'$
$B' \to cB' \mid \varepsilon$
Je vidět, že v tomto případě přítomnost ε-pravidel nebyla na závadu. Příklad 8.4 naopak ukazuje situaci, ve které musí být před odstraněním levé rekurze ε-pravidla vyloučena. Celou situaci můžeme obecně charakterizovat takto:
Jestliže je v gramatice možná derivace:

$$A \Rightarrow^+ B_1 B_2 \ldots B_n A\alpha \Rightarrow^+ A\ \alpha,$$

taková, že $B_i \Rightarrow^+ \varepsilon$, $i \in \langle 1, n \rangle$,
pak musíme před vyloučením levé rekurze z gramatiky vyloučit všechna ε-pravidla, která se vyskytnou právě v derivacích tvaru $B_i \Rightarrow^+ \varepsilon$. V opačném případě můžeme ε-pravidla v gramatice ponechat.

Příklad 8.7
Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:
$*$ $S \to Ba \mid Ab$
$A \to Sa \mid AAb \mid a$
$B \to Sb \mid BBa \mid b$.

Převeďte tuto gramatiku na ekvivalentní gramatiku bez rekurzivy zleva.
Řešení:

1. Zvolíme uspořádání: $\{S, A, B\}$ $(A_1 = S, A_2 = A, A_3 = B)$, $i := 1$,
2. $\emptyset$
3. $i := 2,\ j := 1$
4. $A \to Baa \mid Aba \mid AAb \mid a$

* 2. $A \to Baa \mid a \mid BaaA' \mid aA'$

* $A' \to ba \mid Ab \mid baA' \mid AbA'$

3. $i := 3,\ j := 1$
4. $B \to Bab \mid Abb \mid BBa \mid b$
5. $j := 2$
4. $B \to Bab \mid BBa \mid b \mid Baabb \mid abb \mid BaaA'bb \mid aA'bb$

* 2. $B \to a \mid abb \mid aA'bb \mid aB' \mid abbB' \mid aA'bbB'$

* $B' \to ab \mid Ba \mid aabb \mid aaA'bb \mid abB' \mid BaB' \mid aabbB' \mid aaA'bbB'$

Výsledná gramatika $G' = (\{S, A, B, A', B'\}, \{a, b\}, P', S\}$, kde P' obsahuje pravidla uvedená výše v řádcích označených hvězdičkou.

8.4 Transformace gramatiky do normálních tvarů Chomského a Greibachové

Příklad 8.8
Sestrojme gramatiku v Chomského normálním tvaru, která generuje jazyk

$$L = \{a^n b^n : n \geq 0\} - \{\varepsilon\} = \{a^n b^n : n \geq 1\}.$$

Začneme gramatikou pro jazyk $L_1 = \{a^n b^n : n \geq 0\}$.
Gramatika $G_1 = (\{S\}, \{a, b\}, \{S \to aSb \mid \varepsilon\}, S)$ generuje jazyk $L_1 = L_1(G_1)$.

Po odstranění ε–pravidla dostaneme gramatiku:
$G_2 = (\{S', S\}, \{a, b\}, \{S' \to \varepsilon \mid S, S \to aSb \mid ab\}, S')$.

Jestliže vynecháme ε–pravidlo a počáteční symbol S' dostaneme gramatiku $G_3 = (\{S\}, \{a, b\}, \{S \to aSb \mid ab\}, S)$, která generuje jazyk
$L(G_3) = \{a^n b^n : n \geq 1\}$.
Tuto gramatiku převedeme do normálního tvaru podle Chomského. Za symboly a,b dosadíme do pravidla $S \to aSb$ neterminální symboly A, B a a dostaneme pravidla $S \to ASB$, $A \to a$, $B \to b$.
Stejně dosadíme do pravidla $S \to ab$ a dostaneme pravidlo $S \to AB$. Pravidlo $S \to ASB$ není ve tvaru podle Chomského, proto jako poslední krok zavedeme neterminální symbol C a sestrojíme výslednou gramatiku $G = (\{S, A, B, C\}, \{a, b\}, P, S)$, kde množina P obsahuje pravidla:
$S \to AC \mid AB$
$C \to SB$
$A \to a$
$B \to b$.

Příklad 8.9
Gramatiku $G = (\{S\}, \{a, b\}, P, S)$, kde P obsahuje pravidla $S = aSb \,|\, SS \,|ab$ převeďme do Chomského normálního tvaru.
Podobně jako v předchozím příkladu zavedeme neterminální symboly A,B a jimi nahradíme symboly a,b v pravidlech gramatiky. Dostaneme pravidla:

$S \rightarrow ASB \,|\, SS \,|\, AB$
$A \rightarrow a$
$B \rightarrow b.$

Dále zavedeme neterminální symbol C a dostaneme gramatiku
$G' = (\{S, A, B, C\}, \{a, b\}, P', S)$, kde P' obsahuje pravidla:

$S \rightarrow AC \,|\, SS \,|\, AB$
$C \rightarrow SB$
$A \rightarrow a$
$B \rightarrow b.$

Příklad 8.10
Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \rightarrow aB \,|\, bA$
$A \rightarrow aS \,|\, bAA \,|\, a$
$B \rightarrow bS \,|\, aBB \,|\, b.$

Převeďme tuto gramatiku do normálního tvaru podle Chomského. Přitom použijeme Algoritmus 3.22 [JPR03].
Řešení:

2. $A \rightarrow a$
   $B \rightarrow b$
3. Neprovedeme nic
4. Neprovedeme nic
5. $A \rightarrow b'\langle AA\rangle$
   $\langle AA\rangle \rightarrow AA$
   $B \rightarrow a'\langle BB\rangle$
   $\langle BB\rangle \rightarrow BB$
6. $S \rightarrow a'B \,|\, b'A$
   $A \rightarrow a'S$
   $B \rightarrow b'S$
7. $a' \rightarrow a$
   $b' \rightarrow b.$

Výsledná gramatika $G' = (\{S, A, B, \langle AA\rangle, \langle BB\rangle, a', b'\}, \{a, b\}, P', S)$, kde P' obsahuje pravidla uvedená výše.

Příklad 8.11
Gramatiku $G_3 = (\{S\}, \{a, b\}, \{S \rightarrow aSb \,|\, ab\}, S)$, převeďme do normálního tvaru podle Greibachové.
V tomto případě stačí zavést symbol B a jím nahradit symbol b v obou pravidlech. Dostaneme tak gramatiku
$G_4 = (\{S, B\}, \{a, b\}, \{S \rightarrow aSB \,|\, aB, B \rightarrow b\}, S)$, která je v normálním tvaru podle Greibachové.

Příklad 8.12
Je dána gramatika $G = (\{A, C, Q, R\}, \{x, y\}, P, A)$, kde P obsahuje pravidla:
$A \rightarrow RQC \,|\, Ry$
$C \rightarrow xyA$
$Q \rightarrow xAy$
$R \rightarrow xx$.
Nalezněme G' tak, že $L(G) = L(G')$ a G' je v normálním tvaru podle Greibachové.

Řešení:
Použijeme Algoritmus 3.34 [JPR03]. Zadaná gramatika je vlastní (bez zbytečných symbolů, bez ε–pravidel a bez cyklů) a bez rekurzivity zleva.

1. Lineární uspořádání zvolíme takto: $N = \{A, C, Q, R\}$.
2. $i := 3$
3. Neprovedeme nic
4. $i := 2$
3. Neprovedeme nic
4. $i := 1$
3. V P nahradíme pravidla: $A \rightarrow RQC \,|\, Ry$
   pravidly: $A \rightarrow xxQC \,|\, xxy$
4. $i := 0$
5. Vytvoříme P' takto:
   $A \rightarrow xx'QC \,|\, xx'y'$
   $C \rightarrow xy'A$
   $Q \rightarrow xAy'$
   $R \rightarrow xx'$
6. Do P' přidáme pravidla:
   $x' \rightarrow x$
   $y' \rightarrow y$
   Výsledná gramatika je $G' = (\{A, C, Q, R, x', y'\}, \{x, y\}, P', A)$.

Příklad 8.13
Je dána gramatika $G = (\{S, X, Y, Z, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:
$S \to XYZ \mid a$
$Z \to XaZ \mid B$
$X \to Sb$
$B \to abS$
$Y \to aabb.$

Nalezněte G' takovou, že $L(G) = L(G')$ a G' je v normálním tvaru podle Greibachové.

Řešení:
Zadaná gramatika je vlastní, ale není bez levé rekurzivity. Proto musíme nejdříve odstranit levou rekurzivitu. Přitom použijeme Algoritmus 3.28 [JPR03].
Zvolíme uspořádání: $N = \{S, Z, X, B, Y\}$.
Při tomto uspořádání je třeba dosadit do pravidla $X \to Sb$ symbol S.
Dostaneme pravidla:
$X \to XYZb \mid ab.$
Po odstranění levé rekurze dostaneme pravidla: $X \to ab \mid abX'$
$X' \to YZb \mid YZbX'.$
Dostaneme gramatiku $G_1 = (\{S, X, Y, Z, B, X'\}, \{a, b\}, P_1, S)$, kde P_1 obsahuje pravidla:
$S \to XYZ \mid a$
$Z \to XaZ \mid B$
$B \to abS$
$Y \to aabb$
$X \to ab \mid abX'$
$X' \to YZb \mid YZbX'$
Tato gramatika již neobsahuje rekurzivitu zleva. Pro transformaci do normálního tvaru podle Greibachové použijeme Algoritmus 3.34 [JPR03]:

1. Lineární uspořádání bude: $N_1 = \{S, Z, X, X', B, Y\}$.
2. $i := 5$
3. Neprovedeme nic
4. $i := 4$
3. Pravidla $X' \to YZb \mid YZbX'$ nahradíme pravidly:
   $X' \to aabbZb \mid aabbZbX'.$
4. $i := 3$
3. Neprovedeme nic
4. $i := 2$
3. Nahradíme pravidla: $Z \to XaZ \mid B$
   pravidly: $Z \to abaZ \mid abX'aZ \mid abS.$
4. $i := 1$

3. Nahradíme pravidlo: $S \to XYZ$
   pravidly: $S \to abYZ \mid abX'YZ.$

Dále v 5. a 6. kroku získáme hledanou gramatiku
$G' = (\{S, X, Y, Z, B, X', a', b'\}, \{a, b\}, P', S)$, kde P' obsahuje pravidla:
$S \to ab'YZ \mid ab'X'YZ \mid a$
$Z \to ab'a'Z \mid ab'X'a'Z \mid ab'S$
$B \to ab'S$
$Y \to aa'b'b'$
$X \to ab' \mid ab'X'$
$X' \to aa'b'b'Zb' \mid aa'b'b'Zb'X'$
$a' \to a$
$b' \to b.$

8.5 Příklady pro cvičení

Cvičení 8.14
Z následující gramatiky odstraňte ε–pravidla a jednoduchá pravidla:
$S \to aSbb \mid T$
$T \to bTaa \mid S \mid \varepsilon.$

Cvičení 8.15
Následující gramatiky převeďte do normálního tvaru podle Chomského:

a) $S \to ABS \mid AB$
   $A \to aA \mid a$
   $B \to bA$

b) $S \to aAB \mid aBA \mid bAA \mid \varepsilon$
   $A \to aS \mid bAAA$
   $B \to aABB \mid aBAB \mid aBBA \mid bS$

c) $S \to aSb \mid bSa \mid SS \mid \varepsilon.$

Cvičení 8.16
Pro následující jazyky sestrojte bezkontextové gramatiky v normálním tvaru podle Chomského a Greibachové.
$L_1 = \{ww^R : w \in \{a, b, c\}^*\},$
$L_2 = \{w : w \in \{a, b, c\}^*, w = w^R\},$
$L_3 = \{a^{3n}b^{2n} : n \geq 0\},$
$L_4 = \{a^n b^n b^m c^m : m, n \geq 0\},$
$L_5 = \{a^n b^m c^m d^n : m, n \geq 0\},$
$L_6 = \{w : w \in \{a, b\}^*,$ počet symbolů a je větší než počet symbolů $b\}.$

9 Zásobníkové automaty

Modelem syntaktických analyzátorů bezkontextových jazyků jsou zásobníkové automaty. Tuto skutečnost můžeme intuitivně zdůvodnit tak, že během syntaktické analýzy je nutno zpracovávanou větnou formu uchovávat v nějaké datové struktuře. Adekvátní datovou strukturou je v tomto případě zásobník. Zásobníkový automat je nejjednodušší formální systém, který pracuje se zásobníkem jako s vnitřní pamětí. Z teoretického hlediska můžeme zdůvodnit použití zásobníkových automatů jako modelů syntaktických analyzátorů tím, že bylo dokázáno, že bezkontextové gramatiky a zásobníkové automaty definují stejnou třídu jazyků a to právě bezkontextové jazyky.

9.1 Základní pojmy

Zásobníkový automat je sedmice $R = (K, T, G, \delta, q_0, Z_0, F)$, kde
K je konečná množina vnitřních stavů,
T je konečná vstupní abeceda,
G je konečná abeceda zásobníku,
δ je zobrazení z $K \times (T \cup \{\varepsilon\}) \times G^*$ do množiny konečných podmnožin $K \times G^*$,
q_0 je počáteční stav,
Z_0 je počáteční symbol zásobníku,
$F \subset K$ je množina koncových stavů.
Konfigurace zásobníkového automatu R je trojice $(q, w, \alpha) \in K \times T^* \times G^*$, kde
q je okamžitý vnitřní stav,
w je dosud nezpracovaná část vstupního řetězce,
α je obsah zásobníku.

Počáteční konfigurace zásobníkového automatu je trojice $(q_0, w, Z_0), w \in T^*$. Zápis $\delta(q, b, \alpha) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_n, \gamma_n)\}$ interpretujeme tak, že zásobníkový automat ve stavu q přečte vstupní symbol b a přejde do stavu p_i $(i = 1, 2, \ldots, m)$ a řetězec α z vrcholu zásobníku se nahradí řetězcem γ_i. V případě, že $\delta(q, \varepsilon, \alpha) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$, pak se jedná o přechod do nového stavu a změnu obsahu zásobníku bez čtení vstupního symbolu.
Přechod zásobníkového automatu R je relace na množině konfigurací:

$(q, aw, \alpha\beta) \vdash (p, w, \gamma\beta)$,

když $\delta(q, a, \alpha)$ obsahuje $(p, \gamma), a \in T \cup \{\varepsilon\}, \alpha, \beta, \gamma \in G^*$.
k-tou mocninu, tranzitivní uzávěr a trazitivně reflexivní uzávěr relace $\vdash$ označujeme $\vdash^k, \vdash^+, \vdash^*$.
Jazyk přijímaný zásobníkovým automatem $R = (K, T, G, \delta, q_0, Z_0, F)$ je definován dvojím způsobem:

a) přechodem do koncového stavu:
$L(R) = \{w : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma), \gamma \in G^*, q \in F, w \in T^*\}$.

b) s prázdným zásobníkem:
$L(R) = \{w : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon), q \in K\}$.

Zásobníkový automat $R = (K, T, G, \delta, q_0, Z, F)$ je deterministický, jestliže platí:

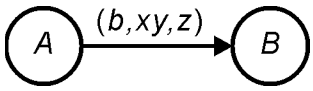
1. $|\delta(q, a, \gamma)| \leq 1$ pro všechna $q \in K$, $a \in T \cup \{\varepsilon\}$, $\gamma \in G^*$.
2. Je-li $\delta(q, a, \alpha) \neq \emptyset$, $\delta(q, a, \beta) \neq \emptyset$, $\alpha \neq \beta$, pak α není příponou β a β není příponou α ($\gamma\alpha \neq \beta$, $\alpha \neq \gamma\beta$).
3. Je-li $\delta(q, a, \alpha) \neq \emptyset$, $\delta(q, \varepsilon, \beta) \neq \emptyset$, pak α není příponou β a β není příponou α ($\gamma\alpha \neq \beta$, $\alpha \neq \gamma\beta$).

Tato definice předpokládá, že zásobník má vrchol vpravo. Jestliže vrchol zásobníku bude vlevo (viz analýzu shora dolů), pak je třeba slovo přípona všude nahradit slovem předpona a místo $\gamma\alpha \neq \beta$, $\alpha \neq \gamma\beta$ psát $\alpha\gamma \neq \beta$, $\alpha \neq \beta\gamma$.

9.2 Deterministické zásobníkové automaty pro typické bezkontextové konstrukce

V tomto odstavci ukážeme několik příkladů zásobníkových automatů, které přijímají jednoduché bezkontextové jazyky. Přitom budeme pro zápis zobrazení navržených automatů používat přechodový diagram podobně jako u konečných automatů. Musíme ovšem rozšířit způsob ohodnocování hran. Hranu v přechodovém diagramu ohodnotíme trojicí (a, α, β), kde $a \in T \cup \{\varepsilon\}$ představuje buď čtený symbol nebo prázdný řetězec, α je řetězec vyloučený z vrcholu zásobníku a β je řetězec vložený na vrchol zásobníku.

Například graf na obr. 9.1 představuje přechod ze stavu A do stavu B při čtení symbolu b, přičemž se z vrcholu zásobníku vyloučí řetězec xy a na vrchol zásobníku se vloží symbol z.



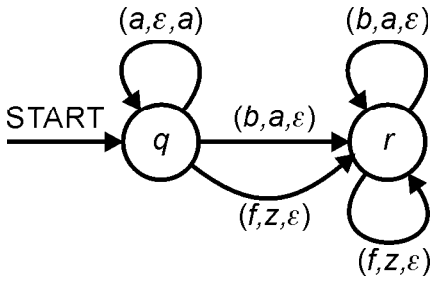
Obrázek 9.1: Přechod ze stavu A do stavu B

V dalších příkladech budeme předpokládat, že každý vstupní řetězec bude ukončen zvláštním symbolem, který budeme důsledně označovat f. Toto není vážné omezení, protože při analýze je vstupní řetězec obvykle tvořen textovým souborem, který končí značkou „konec souboru“.

Příklad 9.1
Je dán jazyk $L = \{a^i b^i : i \geq 0\}$. Gramatika, která tento jazyk generuje, má například tvar $G = (\{S\}, \{a, b\}, \{S \to aSb|\varepsilon\}, S)$. Sestrojíme zásobníkový automat R, který přijímá jazyk L.

Zásobníkový automat bude pracovat takto: Čtené symboly a uloží do zásobníku. Při čtení prvního symbolu b se přejde do stavu r a při čtení každého symbolu b se jeden symbol a ze zásobníku vyloučí. V případě, že při čtení posledního symbolu b byl ze zásobníku vyloučen poslední symbol a, byl vstupní řetězec správný. Zásobníkový automat, který přijímá jazyk L, je definován takto:

$R = (\{q, r\}, \{a, b, f\}, \{a, z\}, \delta, q, z, \emptyset)$, kde δ je definováno takto:
$\delta(q, a, \varepsilon) = (q, a)$
$\delta(q, b, a) = (r, \varepsilon)$
$\delta(r, b, a) = (r, \varepsilon)$
$\delta(r, f, z) = (r, \varepsilon)$
$\delta(q, f, z) = (r, \varepsilon)$
Přechodový diagram tohoto automatu je na obr. 9.2.



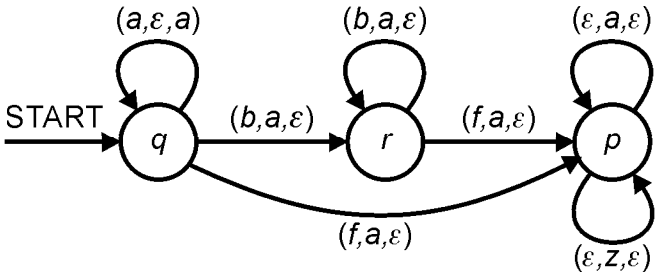
Obrázek 9.2: Zásobníkový automat pro jazyk $L = \{a^i b^i : i \geq 0\}$ z příkladu 9.1

Pro vstupní řetězec a^3b^3f provede zásobníkový automat R tuto posloupnost přechodů:

$(q, a^3b^3f, z) \vdash (q, a^2b^3f, za) \vdash (q, ab^3f, za^2)$
$\vdash (q, b^3f, za^3) \vdash (r, b^2f, za^2) \vdash (r, bf, za)$
$\vdash (r, f, z) \vdash (r, \varepsilon, \varepsilon)$.

Příklad 9.2
Je dán jazyk $L = \{a^i b^j : i > j \geq 0\}$. Gramatika, která tento jazyk generuje, má tvar:
$G = (\{S, A, B\}, \{a, b\}, \{S \to aB, B \to aB|A, A \to aA|\varepsilon\}, S)$.
Zásobníkový automat, který přijímá jazyk L je definován takto:
$R = (\{q, r, p\}, \{a, b, f\}, \{a, z\}, \delta, q, z, \emptyset)$, kde zobrazení δ je definováno takto:
$\delta(q, a, \varepsilon) = (q, a)$
$\delta(q, b, a) = (r, \varepsilon)$
$\delta(q, f, a) = (p, \varepsilon)$
$\delta(r, b, a) = (r, \varepsilon)$
$\delta(r, f, a) = (p, \varepsilon)$
$\delta(p, \varepsilon, a) = (p, \varepsilon)$
$\delta(p, \varepsilon, z) = (p, \varepsilon)$
Přechodový diagram tohoto automatu je na obr. 9.3.

Automat pracuje tak, že symboly a kopíruje do zásobníku. Při čtení symbolů b vylučuje ze zásobníku symboly a. Při nalezení konce vstupního řetězce musí v zásobníku být alespoň jeden symbol a. Je-li jich více, automat je postupně vyloučí. Analýza končí vyprázdněním zásobníku.



Obrázek 9.3: Zásobníkový automat pro jazyk $L = \{a^i b^j : i > j \geq 0\}$ z příkladu 9.2

Příklad 9.3

Je dán jazyk $L = \{a^i b^j : 0 \leq i < j\}$. Gramatika, která tento jazyk generuje má tvar:

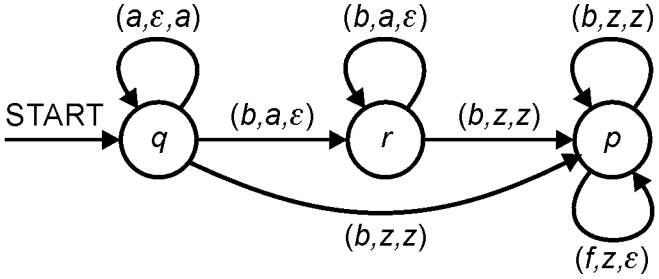
$G = (\{S, A, B\}, \{a, b\}, \{S \to AB, A \to aAb|\varepsilon, B \to bB|b, S)$.

Zásobníkový automat, který přijímá jazyk L je definován takto:

$R = (\{q, r, p\}, \{a, b, f\}, \{a, z\}, \delta, q, z, \emptyset)$, kde zobrazení δ je definováno takto:

$\delta(q, a, \varepsilon) = (q, a)$
$\delta(q, b, a) = (r, \varepsilon)$
$\delta(q, b, z) = (p, z)$
$\delta(r, b, a) = (r, \varepsilon)$
$\delta(r, b, z) = (p, z)$
$\delta(p, b, z) = (p, z)$
$\delta(p, f, z) = (p, \varepsilon)$

Přechodový diagram tohoto automatu je na obr. 9.4.



Obrázek 9.4: Zásobníkový automat pro jazyk $L = \{a^i b^j : 0 \leq i < j\}$ z příkladu 9.3

Příklad 9.4
Je dán jazyk $L = \{a^i b^i : i \geq 0\} \cup \{a^i c^i : i \geq 0\}$. Gramatika, která tento jazyk generuje má tvar : $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

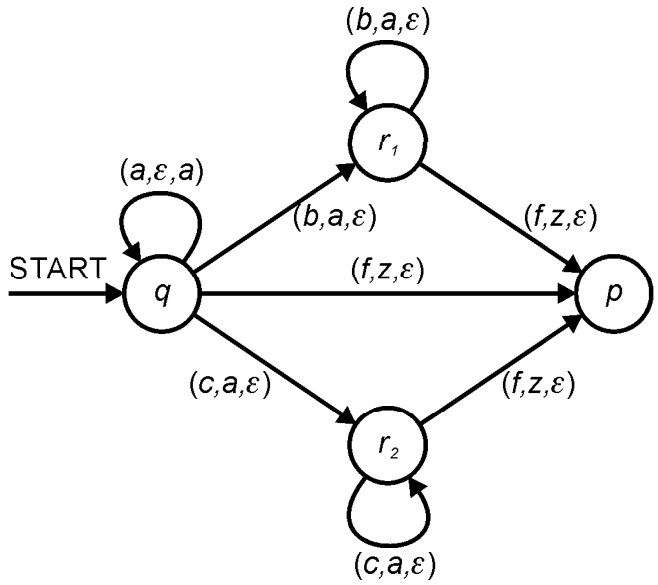
$S \rightarrow A \mid B$
$A \rightarrow aAb \mid \varepsilon$
$B \rightarrow aBc \mid \varepsilon$

Zásobníkový automat, který přijímá jazyk L je definován takto:

$R = (\{q, r_1, r_2, p\}, \{a, b, c, f\}, \{a, z\}, \delta, q, z, \emptyset)$, kde zobrazeni δ je definováno takto:

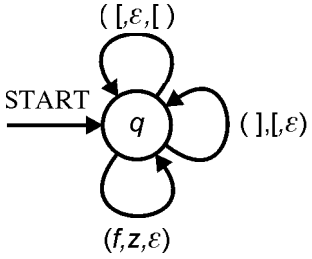
$\delta(q, a, \varepsilon) = (q, a)$
$\delta(q, b, a) = (r_1, \varepsilon)$
$\delta(q, c, a) = (r_2, \varepsilon)$
$\delta(q, f, z) = (p, \varepsilon)$
$\delta(r_1, b, a) = (r_1, \varepsilon)$
$\delta(r_1, f, z) = (p, \varepsilon)$
$\delta(r_2, c, a) = (r_2, \varepsilon)$
$\delta(r_2, f, z) = (p, \varepsilon)$

Přechodový diagram tohoto automatu je na obr. 9.5.



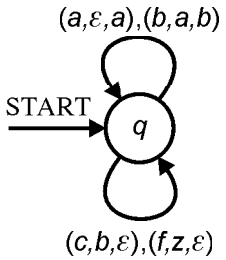
Obrázek 9.5: Zásobníkový automat pro jazyk $L = \{a^i b^i : i \geq 0\} \cup \{a^i c^i : i \geq 0\}$ z příkladu 9.4

Příklad 9.5
Sestrojíme zásobníkový automat pro analýzu závorkové struktury, vytvořenou z dvojice závorek [a]. Bezkontextová gramatika, která tuto strukturu generuje má tvar $G = (\{S\}, \{[,]\}, \{S \to [S]S \mid \varepsilon\}, S)$.
Zásobníkový automat, který přijímá $L(G)$ je definován takto:
$R = (\{q\}, \{[,], f\}, \{[, z\}, \delta, q, z, \emptyset)$, kde zobrazení δ je definováno takto:
$\delta(q, [, \varepsilon) = (q, [\ )$
$\delta(q,], [\ ) = (q, \varepsilon)$
$\delta(q, f, z) = (q, \varepsilon)$
Přechodový diagram je na obr. 9.6.



Obrázek 9.6: Zásobníkový automat pro analýzu dvouzávorkové struktury z příkladu 9.5

Příklad 9.6
Sestrojíme zásobníkový automat pro analýzu závorkové struktury vytvořené z trojice „závorek“ a, b, c. Bezkontextová gramatika, která tuto strukturu generuje má tvar $G = (\{S\}, \{a, b, c\}, \{S \to aSbScS|\varepsilon\}, S)$.
Zásobníkový automat, který přijímá $L(G)$ je definován takto:
$R = (\{q\}, \{a, b, c, f\}, \{a, b, z\}, \delta, q, z, \emptyset)$, kde zobrazení δ je definováno takto:
$\delta(q, a, \varepsilon) = (q, a)$
$\delta(q, b, a) = (q, b)$
$\delta(q, c, b) = (q, \varepsilon)$
$\delta(q, f, z) = (q, \varepsilon)$
Přechodový diagram tohoto automatu je na obr. 9.7.



Obrázek 9.7: Zásobníkový automat pro analýzu třízávorkové struktury z příkladu 9.6

9.3 Zásobníkový automat jako model syntaktického analyzátoru

Pro každou bezkontextovou gramatiku $G = (N, T, P, S)$ existuje zásobníkový automat $R = (K, T, G, \delta, q_0, Z_0, F)$ takový, že $L(G) = L(R)$.

Z této základní vlastnosti bezkontextových gramatik a zásobníkových automatů plyne, že zásobníkové automaty chápeme jako modely syntaktických analyzátorů bezkontextových jazyků. Uvedeme jednu z možných konstrukcí zásobníkových automatů pro zadané bezkontextové gramatiky.

Je dána bezkontextová gramatika $G = (N, T, P, S)$. Sestrojíme zásobníkový automat R takový, že $L(G) = L(R)$.

Konstrukci (označíme ji A) provedeme takto:

$R = (\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, kde zobrazení δ je definováno takto:

(1) $\delta(q, \varepsilon, A) = \{(q, \alpha) : A \to \alpha \in P\}$ pro všechna $A \in N$,

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

Operace (1) je expanze, tj. náhrada neterminálního symbolu ve větné formě pravou stranou některého pravidla, operace (2) je srovnání, tj. přečtení symbolu ze vstupního řetězce a jeho srovnání se symbolem v zásobníku. Vrchol zásobníku u tohoto typu automatu budeme zapisovat vždy vlevo.

Příklad 9.7

S použitím uvedené konstrukce sestrojíme zásobníkový automat pro gramatiku $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla:

(1) $S \to AB$ (3) $A \to ab$ (5) $B \to cd$

(2) $A \to aAb$ (4) $B \to cBd$

Zásobníkový automat vytvoříme takto:

$R = (\{q\}, \{a, b, c, d\}, \{S, A, B, a, b, c, d\}, \delta, q, S, \emptyset)$, kde δ je definováno takto:

$$\begin{aligned}
\delta(q, \varepsilon, S) &= \{(q, AB)\} \\
\delta(q, \varepsilon, A) &= \{(q, aAb), (q, ab)\} \\
\delta(q, \varepsilon, B) &= \{(q, cBd), (q, cd)\} \\
\delta(q, a, a) &= \{(q, \varepsilon)\} \\
\delta(q, b, b) &= \{(q, \varepsilon)\} \\
\delta(q, c, c) &= \{(q, \varepsilon)\} \\
\delta(q, d, d) &= \{(q, \varepsilon)\}.
\end{aligned}$$

Vidíme, že vzniklý zásobníkový automat je nedeterministický. Pro vstupní řetězec $aabbcd$ může automat provést tuto posloupnost přechodů:

$$\begin{aligned}
& (q, aabbcd, S) \\
\vdash\ & (q, aabbcd, AB) && (1) \\
\vdash\ & (q, aabbcd, aAbB) && (2) \\
\vdash\ & (q, abbcd, AbB) \\
\vdash\ & (q, abbcd, abbB) && (3) \\
\vdash\ & (q, bbcd, bbB) \\
\vdash\ & (q, bcd, bB) \\
\vdash\ & (q, cd, B)
\end{aligned}$$

$\vdash$ $(q, \quad cd, \quad cd) \qquad (5)$
$\vdash$ $(q, \quad d, \quad d)$
$\vdash$ $(q, \quad \varepsilon, \quad \varepsilon)$

Uvedené posloupnosti přechodů automatu R odpovídá tato levá derivace v gramatice $G : S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcd$
Levý rozklad řetězce $aabbcd$ je: $1, 2, 3, 5$.
Zásobníkový automat podle uvedené konstrukce představuje model syntaktického analyzátoru pracujícího metodou shora dolů.

Příklad 9.8
S použitím konstrukce B (odst. 3.6.3 [JPR03]) sestrojíme zásobníkový automat pro gramatiku z příkladu 9.7.
Zásobníkový automat bude vypadat takto:

$R = (\{q, r\}, \{a, b, c, d\}, \{S, A, B, a, b, c, d, \#\}, \delta, q, \#, \{r\})$, kde δ je definováno takto:

$\delta(q,a,\varepsilon) = \{(q, a)\}$
$\delta(q,b,\varepsilon) = \{(q, b)\}$
$\delta(q,c,\varepsilon) = \{(q, c)\}$
$\delta(q,d,\varepsilon) = \{(q, d)\}$
$\delta(q,\varepsilon,AB) = \{(q, S)\}$
$\delta(q,\varepsilon,aAb) = \{(q, A)\}$
$\delta(q,\varepsilon,ab) = \{(q, A)\}$
$\delta(q,\varepsilon,cBd) = \{(q, B)\}$
$\delta(q,\varepsilon,cd) = \{(q, B)\}$
$\delta(q,\varepsilon,\#S) = \{(r, \varepsilon)\}$.

Vzniklý zásobníkový automat je opět nedeterministický. Pro vstupní řetěz $aabbcd$ může tento automat provést tuto posloupnost přechodů.

$(q, aabbcd, \#)$
$\vdash (q, abbcd, \#a)$
$\vdash (q, bbcd, \#aa)$
$\vdash (q, bcd, \#aab)$
$\vdash (q, bcd, \#aA)$
$\vdash (q, cd, \#aAb)$
$\vdash (q, cd, \#A)$
$\vdash (q, d, \#Ac)$
$\vdash (q, \varepsilon, \#Acd)$
$\vdash (q, \varepsilon, \#AB)$
$\vdash (q, \varepsilon, \#S)$
$\vdash (q, \varepsilon, \varepsilon)$

Tato posloupnost přechodů automatu odpovídá této obrácené pravé derivaci řetězu $aabbcd$:

$$aabbcd \Leftarrow aAbcd \Leftarrow Acd \Leftarrow AB \Leftarrow S$$

Zásobníkový automat tohoto typu představuje model syntaktického analyzátoru pracujícího metodou zdola nahoru. Oba typy zásobníkových automatů,

které jsme zkonstruovali, jsou automaty *nedeterministické*. To znamená, že jich není možno přímo použít jako algoritmů syntaktické analýzy. Algoritmy syntaktické analýzy můžeme sestrojit na základě těchto nedeterministických automatů dvojím způsobem:

1. Deterministickou simulaci nedeterministických automatů – dostaneme tak algoritmy syntaktické analýzy s návraty.
2. Doplnění konstrukce zásobníkových automatů tak, aby alespoň pro jistou třídu gramatik byly deterministické.

9.4 Syntaktická analýza metodou shora dolů s návratem

Zásobníkový automat sestrojený na základě konstrukce A v minulém odstavci je nedeterministický proto, že při expanzi má obvykle možnost výběru několika alternativ pro určitý neterminální symbol. Deterministická simulace tohoto automatu spočívá v tom, že se postupně systematicky probírají všechny možnosti expanzí neterminálních symbolů. V případě, že při následujících operacích srovnání došlo k chybě, vrací se algoritmus zpět a vybírá další možnosti pro předchozí expanze.

Algoritmus používá dvou zásobníků Z_1 a Z_2 a čítače, jehož hodnota definuje pozici vstupního ukazatele, tj. ukazuje, který symbol vstupního řetězce bude jako další přečten.

Algoritmus 9.9
Syntaktická analýza metodou shora dolů s návratem.
Vstup: Bezkontextová gramatika $G = (N, T, P, S), n \geq 0$, vstupní řetězec $w = a_1 a_2 \ldots a_n$. Předpokládejme, že pravidla z P jsou očíslována čísly $1, 2, \ldots, p$.
Výstup: Levý rozklad řetězce w, pokud existuje, jinak „chyba“. Je-li gramatika G víceznačná, pak algoritmus vytvoří některý z levých rozkladů.
Metoda:

1. Pro každý neterminální symbol $A \in N$ uspořádáme pravé strany všech pravidel. Nechť A_i je index i-té pravé strany pro neterminální symbol A. Například jsou-li $A \to \alpha_1 \mid \alpha_2 \mid \ldots \alpha_n$ všechna pravidla pro A a jsou-li uspořádaná, pak je A_1 indexem pro α_1, A_2 pro α_2 atd.
2. Konfigurace algoritmu bude čtveřice (s, i, α, β), kde:
   (a) s je stav výpočtu,
   (b) i je hodnota vstupního ukazatele; předpokládejme, že $(n+1)$-ní vstupní symbol je koncový symbol #, tedy $i \in \{1, 2, \ldots, n+1\}$.
   (c) α reprezentuje obsah zásobníku Z_1, $\alpha \in (T \cup I)^*$, kde I je množina indexů pravých stran pravidel,
   (d) β reprezentuje obsah zásobníku Z_2, $\beta \in (N \cup T \cup \{\#\})^*$.

Vrchol zásobníku Z_1 bude napravo (poslední symbol řetězce α), vrchol zásobníku Z_2 bude nalevo (první symbol řetězce β). Řetězec β reprezentuje právě získanou větnou formu, řetězec α reprezentuje historii výběru alternativ a vstupní symboly, které byly zpracovány.
Algoritmus může být v jednom z těchto tří stavů:
q ... normální stav,
b ... návrat,
f ... koncový stav.

3. Počáteční konfigurace algoritmu je $(q, 1, \varepsilon, S\#)$, kde ε je prázdný řetězec.
4. Činnost algoritmu je dána posloupností konfigurací počínaje počáteční konfigurací. Zápisem $(s, i, \alpha, \beta) \vdash (s', i', \alpha', \beta')$ budeme označovat přechod z konfigurace (s, i, α, β) do konfigurace $(s', i', \alpha', \beta')$.
   Existuje těchto pět typů přechodů:
   (a) Expanze:
   $(q, i, \alpha, A\beta) \vdash (q, i, \alpha A_1, \gamma_1\beta)$,
   kde $A \to \gamma_1$ je pravidlo z P a γ_1 je první pravá strana (alternativa) pro neterminální symbol A. Tento krok odpovídá expanzi pomocí pravidla $A \to \gamma_1$.
   (b) Srovnání:
   $(q, i, \alpha, a\beta) \vdash (q, i + 1, \alpha a, \beta)$
   za předpokladu, že $a_i = a$, $i \leq n$. Shoduje-li se i-tý vstupní symbol s prvním symbolem derivace, přesuneme vrchol zásobníku Z_2 na vrchol Z_1 a zvýšíme vstupní ukazatel.
   (c) Přijetí:
   $(q, n + 1, \alpha, \#) \vdash (f, n + 1, \alpha, \varepsilon)$
   V tomto kroku jsme dospěli ke koncové konfiguraci. Derivovaná věta se shoduje se vstupním řetězcem. Levý rozklad obdržíme z řetězce α, aplikujeme-li na každý symbol řetězce α tuto funkci h:
   $h(a) = \varepsilon$ pro všechna $a \in T$,
   $h(A_i) = p$, kde p je číslo pravidla $A \to \gamma$, γ je i-tá pravá strana pro neterminální symbol A.
   (d) Neshoda vstupního a derivovaného symbolu:
   $(q, i, \alpha, a\beta) \vdash (b, i, \alpha, a\beta)$, je-li $a_i \neq a$.
   Tento krok signalizuje, že je třeba provést návrat, protože derivovaná větná forma neodpovídá vstupnímu řetězci.
   (e) Návrat na vstupu:
   $(b, i, \alpha a, \beta) \vdash (b, i - 1, \alpha, a\beta)$ pro všechna $a \in T$.
   Vstupní symbol a přesuneme zpět ze Z_1 do Z_2 a snížíme vstupní ukazatel.
   (f) Zkus jinou alternativu: $(b, i, \alpha A_j, \gamma_j\beta) \vdash$
      i. $(q, i, \alpha A_{j+1}, \gamma_{j+1}\beta)$, jestliže γ_{j+1} je $(j + 1)$-ní alternativa pro A. (Na vrchol Z_2 je namísto γ_j uložen řetězec γ_{j+1}).

ii. $(b, i, \alpha, A\beta)$, jestliže neexistuje další alternativa pro A. Index A_j je odstraněn z vrcholu Z_1, neterminál A je uložen namísto γ na vrchol Z_2.

iii. („žádná konfigurace“), je-li $i = 1$ a $A = S$. Tyto podmínky indikují, že byly zkoušeny všechny možné levé derivace, aniž byl nalezen rozklad w.

Algoritmus nyní probíhá takto:

Krok 1: Počínaje počáteční konfigurací hledej další konfigurace $C_0 \rightarrow C_1 \rightarrow \ldots$, dokud je to možné.

Krok 2: Je-li poslední nalezená konfigurace $(f, n+1, \gamma, \varepsilon)$, aplikuj na řetězec γ funkci h; $h(\gamma)$ je levý rozklad věty w. Není-li poslední nalezená konfigurace konfigurací koncovou, je výstupem algoritmu „chyba“.

Příklad 9.10
Je dána gramatika $G=(\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde pravidla v P jsou označena takto:

(S_1) $S \rightarrow AB$

(A_1) $A \rightarrow ab$

(A_2) $A \rightarrow aAb$

(B_1) $B \rightarrow cBd$

(B_2) $B \rightarrow cd$

Algoritmus 9.9 pro vstupní řetězec $aabbcd$ provede tuto posloupnost přechodů:

$$
\begin{aligned}
(q, 1, \varepsilon, S\#) &\vdash (q,\ 1,\ S_1 & ,\ & AB\#)\\
&\vdash (q,\ 1,\ S_1A_1 & ,\ & abB\#)\\
&\vdash (q,\ 2,\ S_1A_1a & ,\ & bB\#)\\
&\vdash (b,\ 2,\ S_1A_1a & ,\ & bB\#)\\
&\vdash (b,\ 1,\ S_1A_1 & ,\ & abB\#)\\
&\vdash (q,\ 1,\ S_1A_2 & ,\ & aAbB\#)\\
&\vdash (q,\ 2,\ S_1A_2a & ,\ & AbB\#)\\
&\vdash (q,\ 2,\ S_1A_2a\ A_1 & ,\ & abbB\#)\\
&\vdash (q,\ 3,\ S_1A_2a\ A_1a & ,\ & bbB\#)\\
&\vdash (q,\ 4,\ S_1A_2a\ A_1a\ b & ,\ & bB\#)\\
&\vdash (q,\ 5,\ S_1A_2a\ A_1a\ bb & ,\ & B\#)\\
&\vdash (q,\ 5,\ S_1A_2a\ A_1a\ bbB_1 & ,\ & cBd\#)\\
&\vdash (q,\ 6,\ S_1A_2a\ A_1a\ bbB_1c & ,\ & Bd\#)\\
&\vdash (q,\ 6,\ S_1A_2a\ A_1a\ bbB_1cB_1, & & cBdd\#)\\
&\vdash (b,\ 6,\ S_1A_2a\ A_1a\ bbB_1cB_1, & & cBdd\#)\\
&\vdash (q,\ 6,\ S_1A_2a\ A_1a\ bbB_1cB_2, & & cdd\#)
\end{aligned}
$$

$$\begin{aligned}
&\vdash (b\,,\,6,\,S_1A_2a\;A_1a\;bbB_1cB_2,\;\;cdd\#)\\
&\vdash (b\,,\,6,\,S_1A_2a\;A_1a\;bbB_1c\;\;\;,\;\;Bd\#)\\
&\vdash (b\,,\,5,\,S_1A_2a\;A_1a\;bbB_1\;\;\;\;,\;cBd\#)\\
&\vdash (q\,,\,5,\,S_1A_2a\;A_1a\;bbB_2\;\;\;\;,\;\;cd\#)\\
&\vdash (q\,,\,6,\,S_1A_2a\;A_1a\;bbB_2c\;\;,\;\;\;d\#)\\
&\vdash (q\,,\,7,\,S_1A_2a\;A_1a\;bbB_2cd\;\;,\;\;\;\;\#)\\
&\vdash (f,\,7,\,S_1A_2a\;A_1a\;bbB_2cd\;\;,\;\;\;\;\varepsilon)
\end{aligned}$$

Algoritmus zjistil, že v levé derivaci byla použita tato pravidla:

$(S_1)\ S \;\to\; AB$
$(A_2)\ A \;\to\; aAb$
$(A_1)\ A \;\to\; ab$
$(B_2)\ B \;\to\; cd$

Příklad 9.11

Je dána gramatika $G = (\{S\}, \{a, b\}, P, S)$. Pravidla v P jsou označena takto:

$(S_1)\ S \;\to\; Sa$
$(S_2)\ S \;\to\; b$

Algoritmus 9.9 začne provádět pro vstupní řetězec ba tuto posloupnost přechodů:

$$\begin{aligned}
(q,\ 1,\ \varepsilon,\ S\#)\ &\vdash (q,\ 1,\ \ S_1\ \ \ \ \ \ \ \ \ \ ,\ \ \ \ Sa\#)\\
&\vdash (q,\ 1,\ \ S_1\ S_1\ \ \ \ \ \ \ \ ,\ \ \ \ Saa\#)\\
&\ldots\\
&\vdash (q,\ 1,\ \ S_1\ S_1\ \ldots\ S_1\ \ \ ,\ \ Sa\ldots a\#)\\
&\ldots
\end{aligned}$$

a dostaneme se do nekonečného cyklu, ve kterém do obou zásobníků přidává levě rekurzivní pravidlo. Nebezpečí takového cyklu je vždy, když daná gramatika je levě rekurzivní. Proto algoritmus 9.9 pracuje správně jen pro gramatiky bez levé rekurze.

9.5 Příklady pro cvičení

Cvičení 9.12

Sestrojte deterministické zásobníkové automaty přijímající tyto jazyky:

a) $\{a^ib^j : i > j \geq 0\} \cup \{a^ic^k : k > i \geq 0\}$,
b) $\{a^ib^j : i > j \geq 0\} \cup \{a^ic^k : 0 \leq k < i\}$,
c) $\{a^ib^j : 0 \leq i < j\} \cup \{a^ic^k : 0 \leq i < j\}$,
d) $\{\{a, b\}^* :$ počet a je stejný jako počet $b\}$,
e) $\{\{a, b\}^* :$ počet a je větší než počet $b\}$.

Cvičení 9.13

Sestrojte deterministický zásobníkový automat, který přijímá čtyřzávorkové struktury generované gramatikou

$G = (\{S\}, \{a, b, c, d\}, \{S \to aSbScSdS \mid \varepsilon\}, S)$.

Cvičení 9.14
Sestrojte deterministický zásobníkový automat, který přijímá n–závorkovou strukturu.

Cvičení 9.15
Sestrojte deterministické zásobníkové automaty pro tyto jazyky:

a) $\{a^n b^n a^m b^m : n, m \geq 1\}$,
b) $\{a^n b^m : n \leq m \leq 2n\}$,
c) $\{a^n b^n c^m : n, m \geq 1\}$,
d) $\{a^m b^n c^n : n, m \geq 1\}$,
e) $\{a^i b^i a^j b^k : i, j \geq 1,\ k = 2j\}$,
f) $\{a^i b^j : 0 \leq i \leq 3j\}$.

Cvičení 9.16
Sestrojte ekvivalentní zásobníkové automaty podle konstrukce A. a B. z odstavce 3.6.3 [JPR03] pro tyto gramatiky:

$G = (\{S\}, \{a, b\}, \{S \to aSb \mid \varepsilon\}, S)$,
$G = (\{S, A\}, \{a, b\}, \{S \to aSb \mid A, A \to aA \mid a, S)$,
$G = (\{S, A, \{a, b\}, \{S \to aSbA \mid \varepsilon, A \to bA \mid b, S)$,
$G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

$$S \to A \mid B$$
$$A \to aAb \mid \varepsilon$$
$$B \to aBc \mid \varepsilon,$$

$G = (\{S\}, \{(,)\}, \{S \to (S)S \mid \varepsilon\}, S)$,
$G = (\{S\}, \{a, b, c\}, \{S \to aSbScS \mid \varepsilon\}, S)$.

Sestrojené zásobníkové automaty porovnejte s deterministickými zásobníkovými automaty z příkladů 9.1, 9.2, 9.3, 9.4, 9.5, a 9.6.

10 $LL(1)$ gramatiky

10.1 Základní pojmy

Bezkontextová gramatika $G = (N, T, P, S)$ je $LL(k)$ pro nějaké pevné celé číslo k, $(k \geq 0)$, jestliže v případě existence dvou levých derivací

(1) $S \overset{*}{\Rightarrow} vA\alpha \Rightarrow v\beta\alpha \overset{*}{\Rightarrow} vx$

(2) $S \overset{*}{\Rightarrow} vA\alpha \Rightarrow v\gamma\alpha \overset{*}{\Rightarrow} vy$

takových, že $FIRST_k(x) = FIRST_k(y)$, platí $\beta = \gamma$.

Bezkontextová gramatika $G = (N, T, P, S)$ je *silná* $LL(k)$ gramatika, když pro všechna $A \in N$ platí:

V případě, že $A \to \alpha$ a $A \to \beta$ jsou různá pravidla v P, platí:

$$FIRST_k(\alpha FOLLOW_k(A)) \cap FIRST_k(\beta FOLLOW_k(A)) = \emptyset.$$

Bezkontextová gramatika $G = (N, T, P, S)$ je $LL(1)$, když pro každé $A \in N$ platí:

Jestliže $A \to \alpha$ a $A \to \beta$ jsou různá pravidla v P, pak platí:

$$FIRST_1(\alpha FOLLOW_1(A)) \cap FIRST_1(\beta FOLLOW_1(A)) = \emptyset.$$

Bezkontextová gramatika $G = (N, T, P, S)$ se nazývá *q–gramatika*, když platí:

1. Pravá strana každého pravidla je buď prázdný řetěz nebo začíná terminálním symbolem.
2. Jestliže dvě pravidla mají stejnou levou stranu, pak jejich pravé strany začínají různými terminálními symboly, pokud některý z nich není prázdný řetěz.
3. Jestliže se v gramatice vyskytne pravidlo $A \to \varepsilon$, pak terminální symboly, kterými začínají pravé strany ostatních pravidel se symbolem A na levé straně nesmí být obsaženy ve $FOLLOW_1(A)$.

q–gramatika bez ε pravidel se nazývá *jednoduchá* $LL(1)$ gramatika (někdy také S–gramatika). Pro konstrukci rozkladové tabulky budeme používat Algoritmus 4.3, 4.13 a 4.19 [JPR03]. Pro syntaktickou analýzu použijeme Algoritmus 4.5 [JPR03]. Pro výpočet funkcí FIRST a FOLLOW použijeme Algoritmy 4.21 a 4.23 [JPR03].

10.2 Chybová hlášení při LL analýze

V dalších několika příkladech ukážeme rozkladové tabulky pro gramatiky, které generují posloupnosti, seznamy a závorkové struktury. Současně ukážeme průběh syntaktické analýzy pro správné řetězce a pro špatné řetězce. Přitom se pokusíme navrhnout formulaci chybového hlášení pro chybové položky v rozkladových tabulkách. Gramatiky použité v následujících příkladech jsou z tabulky 1.4 v odstavci 1.4.

Příklad 10.1
$A_1 = (\{L, R\}, \{a\}, P, L)$, kde P obsahuje pravidla:

(1) $L \to aR$
(2) $R \to aR$
(3) $R \to \varepsilon$

M	a	ε
L	$aR, 1$	$chyba1$
R	$aR, 2$	$\varepsilon, 3$

Analýza správné posloupnosti:

$$\begin{aligned}(aaa, L, \varepsilon) &\vdash (aaa,\ aR,\ 1\ \ \ \ ) \\ &\vdash (aa\ \ ,\ \ R,\ 1\ \ \ \ ) \\ &\vdash (aa\ \ ,\ aR,\ 12\ \ ) \\ &\vdash (a\ \ \ ,\ \ R,\ 12\ \ ) \\ &\vdash (a\ \ \ ,\ aR,\ 122\ ) \\ &\vdash (\varepsilon\ \ ,\ \ R,\ 122\ ) \\ &\vdash (\varepsilon\ \ ,\ \ \varepsilon,\ 1223)\end{aligned}$$

Analýza prázdného řetězce (jazyk $L(A_1)$ prázdný řetězec neobsahuje):

$(\varepsilon, L, \varepsilon) \vdash$ chyba 1

Chybové hlášení: Chyba 1 - vstupní řetězec je prázdný.

Příklad 10.2
$A_2 = (\{L, R\}, \{a, ;\}, P, L)$, kde P obsahuje pravidla:

(1) $L \to aR$
(2) $R \to ; aR$
(3) $R \to \varepsilon$

M	a	;	ε
L	$aR, 1$	$chyba1$	$chyba2$
R	$chyba3$	$; aR, 2$	$\varepsilon, 3$

Analýza správného seznamu:

$$\begin{aligned}
(a;a;a,L,\varepsilon) &\vdash (a;a;a,\ aR,1\ \ \ ) \\
&\vdash (\ ;a;a,\ \ R,1\ \ \ ) \\
&\vdash (\ ;a;a,;aR,12\ \ ) \\
&\vdash (\ \ a;a,\ aR,12\ \ ) \\
&\vdash (\ \ \ ;a,\ \ R,12\ \ ) \\
&\vdash (\ \ \ ;a,;aR,122\ ) \\
&\vdash (\ \ \ a,\ aR,122\ ) \\
&\vdash (\ \ \ \varepsilon,\ \ R,122\ ) \\
&\vdash (\ \ \ \varepsilon,\ \ \varepsilon,1223)
\end{aligned}$$

Analýza nesprávných seznamů:

$(;a;a,L,\varepsilon)\ \vdash\ chyba\ 1$

Chyba 1 - seznam začíná omezovačem.

$(\varepsilon,L,\varepsilon)\ \vdash\ chyba\ 2$

Chyba 2 - seznam je prázdný.

$$\begin{aligned}
(aa,L,\varepsilon) &\vdash (\,aa,aR,1\,) \\
&\vdash (\ \ a,\ \ R,1\,) \\
&\vdash\ chyba\ 3
\end{aligned}$$

Chyba 3 - v seznamu chybí omezovač.

$$\begin{aligned}
(a;;a,L,\varepsilon) &\vdash (\,a;;a,\ \ aR,\ 1\ \ ) \\
&\vdash (\ \ ;;a,\ \ \ R,\ 1\ \ ) \\
&\vdash (\ \ ;;a,\ ;aR,\ 12) \\
&\vdash (\ \ \ ;a,\ \ aR,\ 12) \\
&\vdash\ chyba\ 4
\end{aligned}$$

Chyba 4 - dva omezovače jsou těsně za sebou (chyba při srovnání).

$$\begin{aligned}
(a;,L,\varepsilon) &\vdash (\,a;,\ \ aR,1)\ \ ) \\
&\vdash (\ \ ;,\ \ \ R,1)\ \ ) \\
&\vdash (\ \ ;,;aR,12)) \\
&\vdash (\ \varepsilon\,,\ \ aR,12)) \\
&\vdash\ chyba\ 5
\end{aligned}$$

Chyba 5 - chybí poslední prvek seznamu (vstupní řetězec je přečten a zásobník není prázdný, chyba při srovnání).

Příklad 10.3
$A_3 = (\{L\}, \{a\}, P, L)$, kde P obsahuje pravidla:

(1) $L \to \varepsilon$
(2) $L \to aL$

M	a	ε
L	$aL, 2$	$\varepsilon, 1$

Analýza správné posloupnosti:

$$\begin{aligned}(aaa, L, \varepsilon) &\vdash (aaa, aL, 2\ \ \ \ ) \\ &\vdash (\ aa,\ \ L, 2\ \ \ \ ) \\ &\vdash (\ aa, aL, 22\ \ ) \\ &\vdash (\ \ a,\ \ L, 22\ \ ) \\ &\vdash (\ \ a, aL, 222\ ) \\ &\vdash (\ \ \varepsilon,\ \ L, 222\ ) \\ &\vdash (\ \ \varepsilon,\ \ \varepsilon, 2221)\end{aligned}$$

Žádná chyba nemůže nastat, protože neexistuje špatný řetězec. Platí totiž, že $L(G) = a^*$.

Příklad 10.4
$A_4 = (\{L, R\}, \{a, ;\}, P, L)$, kde P obsahuje pravidla:

(1) $L \to \varepsilon$
(2) $L \to aR$
(3) $R \to ; aR$
(4) $R \to \varepsilon$

M	a	$;$	ε
L	$aR, 2$	$chyba1$	$\varepsilon, 1$
R	$chyba3$	$;aR, 3$	$\varepsilon, 4$

Analýza správného seznamu:

$$\begin{aligned}(a;a;a, L, \varepsilon) &\vdash (a;a;a,\ \ aR, 2\ \ \ \ ) \\ &\vdash (\ ;a;a,\ \ \ R, 2\ \ \ \ ) \\ &\vdash (\ ;a;a,;aR, 23\ \ ) \\ &\vdash (\ \ a;a,\ \ aR, 23\ \ ) \\ &\vdash (\ \ \ ;a,\ \ \ R, 23\ \ ) \\ &\vdash (\ \ \ ;a,;aR, 233\ ) \\ &\vdash (\ \ \ \ a,\ \ aR, 233\ ) \\ &\vdash (\ \ \ \ \varepsilon,\ \ \ R, 233\ ) \\ &\vdash (\ \ \ \ \varepsilon,\ \ \ \varepsilon, 2334)\end{aligned}$$

Význam chybových položek v tabulce M je stejný jako v příkladu 10.2. Rozdíl je pouze v tom, že chyba 2 (seznam je prázdný) není v tomto případě chybou, protože prázdný seznam patří do jazyka $L(G)$ a v rozkladové tabulce $M(L, \varepsilon) = (\varepsilon, 1)$.

Příklad 10.5
Gramatika pro generování jednoduché závorkové struktury $G = (\{S\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

(1) $S \rightarrow aSbS$
(2) $S \rightarrow \varepsilon$

M	a	b	ε
S	$aSbS, 1$	$\varepsilon, 2$	$\varepsilon, 2$

Analýza správné struktury:

$$
\begin{aligned}
(abab, S, \varepsilon) &\vdash (abab, \ aSbS, \ 1\ ) \\
&\vdash (\ bab, \ SbS, \ 1\ ) \\
&\vdash (\ bab, \ bS, \ 12\ ) \\
&\vdash (\ ab, \ S, \ 12\ ) \\
&\vdash (\ ab, \ aSbS, \ 121\ ) \\
&\vdash (\ b, \ SbS, \ 121\ ) \\
&\vdash (\ b, \ bS, \ 1212\ ) \\
&\vdash (\ \varepsilon, \ S, \ 1212\ ) \\
&\vdash (\ \varepsilon, \ \varepsilon, \ 12122)
\end{aligned}
$$

Analýza nesprávných závorkových struktur:

$(b, S, \varepsilon) \vdash (b, \varepsilon, 2) \vdash$ *chyba*
- v řetězci je navíc zavírací závorka.

$$
\begin{aligned}
(a, S, \varepsilon) &\vdash (a, aSbS, 1\ ) \\
&\vdash (\varepsilon, \ SbS, 1\ ) \\
&\vdash (\varepsilon, \ \ bS, 12) \vdash \textit{chyba}
\end{aligned}
$$

- v řetězci chybí zavírací závorka.

10.3 Příklady pro cvičení

Cvičení 10.6
Sestrojte rozkladovou tabulku pro gramatiku G=($\{S, R\}, \{a, (,), .\}, P, S$), kde P obsahuje pravidla:

$S \rightarrow a$
$S \rightarrow (SR$
$R \rightarrow .SR$
$R \rightarrow) .$

$L(G)$ je jazyk seznamů podobný jako v LISPu.

Cvičení 10.7
Sestrojte rozkladovou tabulku pro gramatiku
$G = (\{S\}, \{a, b, c\}, \{S \rightarrow aSbScS \mid \varepsilon\}, S)$. $L(G)$ je jazyk třízávorkových struktur.

Cvičení 10.8
Sestrojte rozkladovou tabulku pro gramatiku $G = (\{S, A\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \rightarrow AS \mid \varepsilon$
$A \rightarrow aA \mid b.$

Cvičení 10.9
Je dána gramatika $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla:

$S \rightarrow BA$
$A \rightarrow BS$
$A \rightarrow d$
$B \rightarrow aA$
$B \rightarrow bS$
$B \rightarrow c.$

Sestrojte syntaktický analyzátor pro tuto gramatiku.

Cvičení 10.10
Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \rightarrow AB$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bA \mid \varepsilon$.

Sestrojte syntaktický analyzátor pro tuto gramatiku.

Cvičení 10.11
Syntaktický analyzátor pro gramatiku $G = (\{S\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla:

(1) $S \rightarrow aSS$
(2) $S \rightarrow bc$

(3) $S \to cSSS$
(4) $A \to a$

vytvoří tuto výstupní posloupnost: 1 4 3 2 4 2 1 4 2 4 4.
Zjistěte, zda je to levý rozklad nějaké věty z L(G).

Cvičení 10.12
Pro následující jazyky najděte $LL(1)$ gramatiky a sestrojte syntaktické analyzátory:

a) $\{1^n : n \geq 0\}$,
b) $\{1^n 0^n : n \geq 0\}$,
c) $\{1^n 0^m : 0 < n \leq m\}$.

Cvičení 10.13
Sestrojte syntaktický analyzátor pro gramatiku
$G = (\{S, A, B\}, \{a, b, c, d, x\}, P, S)$, kde P obsahuje pravidla:

(1) $S \to aAB$
(2) $S \to bBS$
(3) $S \to \varepsilon$
(4) $A \to cBS$
(5) $A \to \varepsilon$
(6) $B \to dB$
(7) $B \to x$

Cvičení 10.14
Sestrojte syntaktický analyzátor pro gramatiku $G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

(1) $S \to aAbBbS$
(2) $S \to \varepsilon$
(3) $A \to aBC$
(4) $A \to bA$
(5) $B \to aB$
(6) $B \to \varepsilon$
(7) $C \to cC$
(8) $C \to \varepsilon$

Vytvořte levý rozklad pro řetězce:
aaabab,
aabbaabb.

Cvičení 10.15
Určete, zda gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

(1) $S \to aAb$
(2) $S \to bB$
(3) $S \to \varepsilon$
(4) $A \to aAb$
(5) $A \to \varepsilon$
(6) $B \to bB$
(7) $B \to c$

je $LL(1)$ gramatika. V případě, že odpověď bude kladná, sestrojte rozkladovou tabulku.

Cvičení 10.16
Je dána gramatika
$G = (\{$*blok, deklarace, zbytek-seznamu-deklarací, příkaz, zbytek-seznamu-příkazů*$\}, \{\langle begin\rangle, \langle end\rangle, ;, p, d\}, P,$ *blok*$)$, kde P obsahuje pravidla:
(1) *blok* $\to$ $\langle begin\rangle$ *deklarace* ; *zbytek-seznamu-deklarací příkaz zbytek-seznamu-příkazů* $\langle end\rangle$
(2) *zbytek-seznamu-deklarací* $\to$ *deklarace* ; *zbytek-seznamu-deklarací*
(3) *zbytek-seznamu-deklarací* $\to$ ε
(4) *zbytek-seznamu-příkazů* $\to$; *příkaz zbytek-seznamu-příkazů*
(5) *zbytek-seznamu-příkazů* $\to$ ε
(6) *deklarace* $\to$ *d*
(7) *příkaz* $\to$ *p*
Sestrojte pro tuto $LL(1)$ gramatiku rozkladovou tabulku a naznačte, jak se bude provádět rozklad věty $\langle begin\rangle$ *d* ; *p* $\langle end\rangle$.

Cvičení 10.17
Je dána gramatika
$G = (\{\langle$ *deklarace* $\rangle, \langle$ *seznam-id* $\rangle, \langle$ *ident* $\rangle, \langle$ *zbytek-seznamu-id* $\rangle\}$,
$\{real, integer, id, ;\}, P, \langle$ *deklarace* $\rangle)$, kde P obsahuje pravidla:
(1) $\langle$ *deklarace* $\rangle \to real \langle$ *seznam-id* $\rangle$
(2) $\langle$ *deklarace* $\rangle \to integer \langle$ *seznam-id* $\rangle$
(3) $\langle$ *seznam-id* $\rangle \to \langle$ *ident* $\rangle$ $\langle$ *zbytek-seznamu-id* $\rangle$
(4) $\langle$ *zbytek-seznamu-id* $\rangle \to$; $\langle$ *ident* $\rangle$ $\langle$ *zbytek-seznamu-id* $\rangle$
(5) $\langle$ *zbytek-seznamu-id* $\rangle \to \varepsilon$
(6) $\langle$ *ident* $\rangle \to id$
Sestrojte pro tuto $LL(1)$ gramatiku rozkladovou tabulku a naznačte, jak se bude provádět rozklad věty: *real id* ; *id* .

Cvičení 10.18
Je dána gramatika G=($\{S, A, B, C, D\}, \{a, b, c, d, x, y, z\}, P, S$), kde P obsahuje pravidla:

$S \rightarrow aABbCD$
$S \rightarrow \varepsilon$
$A \rightarrow ASd$
$A \rightarrow \varepsilon$
$B \rightarrow SAc$
$B \rightarrow xC$
$B \rightarrow \varepsilon$
$C \rightarrow Sy$
$C \rightarrow Cz$
$C \rightarrow \varepsilon$
$D \rightarrow aBD$
$D \rightarrow \varepsilon$.

Vypočtěte hodnotu funkce $FOLLOW$ pro všechny neterminální symboly a zjistěte, zda daná gramatika je $LL(1)$. V případě kladné odpovědi sestrojte rozkladovou tabulku.

Cvičení 10.19
Sestrojte rozkladovou tabulku pro gramatiku
G=($\{S, A, B, C\}, \{a, b, c, d, x\}, P, S$), kde P obsahuje pravidla:

$S \rightarrow AbB$
$S \rightarrow d$
$A \rightarrow CAb$
$A \rightarrow B$
$B \rightarrow cSd$
$B \rightarrow \varepsilon$
$C \rightarrow a$
$C \rightarrow xd$.

Cvičení 10.20
Sestrojte rozkladovou tabulku pro gramatiku
$G = (\{E, E', T, T', F, F', P\}, \{+, *, \lambda, (,), a, b\}, P, E)$, kde P obsahuje pravidla:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow FT' \mid \varepsilon$
$F \rightarrow (E)F' \mid aF' \mid bF' \mid \lambda F'$
$F' \rightarrow *F' \mid \varepsilon$.

$L(G)$ je jazyk regulárních výrazů nad abecedou $\{a, b\}$.

Cvičení 10.21
Sestrojte rozkladovou tabulku pro gramatiku $G = (\{E, T, R\}, \{a, +, *\}, P, E)$, kde P obsahuje pravidla:

$E \to aTR$
$R \to +aTR$
$R \to \varepsilon$
$T \to *aT$
$T \to \varepsilon$.

Cvičení 10.22
Pro následující jazyky najděte $LL(1)$ gramatiky a sestrojte syntaktické analyzátory:

a) $\{1^n a 0^n : n \geq 0\}$,
b) $\{waw^R : w \in \{0,1\}^*\}$,
c) $\{1^n m 0^n : n > 0\} \cup \{m 1^n m 0^{2n} : n > 0\}$,
d) $\{1^n a 0^n 1^p a 0^p : n > 0, p \geq 0\}$,
e) $\{1^n 0^n : n > 0\}$.

Z každého jazyka vyberte řetězec delší než 5 znaků a proveďte jeho syntaktickou analýzu.

Cvičení 10.23
Sestrojte rozkladovou tabulku pro tuto gramatiku:
$G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla:

$S \to aSA$
$S \to \varepsilon$
$A \to bB$
$A \to cc$
$B \to bd$
$B \to \varepsilon$.

Cvičení 10.24
Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to aAaB \mid bAbB$
$A \to a \mid ab$
$B \to aB \mid a$.

Zjistěte, zda tato gramatika je $LL(k)$ gramatika pro $k \leq 3$.
Řešení: Gramatika je $LL(3)$, ale není silná $LL(3)$ gramatika.

Cvičení 10.25
Ukažte, že gramatika $G = (\{S, A, B\}, \{0, 1, a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to A \mid B$
$A \to aBb \mid 0$
$B \to aBbb \mid 1,$

není $LL(k)$ pro žádné k.

Cvičení 10.26
Je dána gramatika
$G = (\{Z, S, X\}, \{a, b\}, \{Z \to S, S \to X \mid bXa\,, X \to a\ \,, X \to \varepsilon\}, Z)$.
Zjistěte, zda gramatika je $LL(2)$ a zda je silná $LL(2)$.
Řešení: Gramatika G je $LL(2)$, ale není silná $LL(2)$.

Cvičení 10.27
Je dána gramatika
$G = (\{S, A\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$S \to abSa$
$S \to aaAb$
$S \to b$
$A \to baAb$
$A \to b.$

Sestrojte pro tuto gramatiku rozkladovou tabulku.
Platí
$FIRST_2(abSa) \cap FIRST_2(aaAb) = \{ab\} \cap \{aa\} = \emptyset$
$FIRST_2(abSA) \cap FIRST_2(bFOLLOW_2(S)) = \{ab\} \cap \{ba\} = \emptyset$
$FIRST_2(aaAb) \cap FIRST_2(bFOLLOW_2(S)) = \{aa\} \cap \{ba\} = \emptyset$
$FIRST_2(baAb) \cap FIRST_2(bFOLLOW_2(A)) = \{ba\} \cap \{bb\} = \emptyset,$
a proto gramatika G je $LL(2)$.

11 Transformace bezkontextové gramatiky na $LL(1)$ gramatiku

11.1 Základní pojmy

Je známo, že nelze najít algoritmus, který by libovolnou bezkontextovou gramatiku transformoval na $LL(k)$ gramatiku. Důvody k tomuto tvrzení jsou v zásadě dva.

Prvním důvodem je skutečnost, že pro některé bezkontextové jazyky neexistují $LL(k)$ gramatiky. Příkladem může být jazyk $L = \{a^n b^n : n \geq 0\} \cup \{a^n c^n : n \geq 0\}$.

Druhým důvodem pro výše uvedené tvrzení je skutečnost, že použitá transformační metoda vede do nekonečného cyklu i v případě, že transformujeme bezkontextovou gramatiku, ke které existuje ekvivalentní $LL(k)$ gramatika.

Přesto můžeme v řadě případů jednoduchými transformacemi získat z bezkontextové gramatiky $LL(k)$ gramatiku.

Základní transformace pro získání ekvivalentní $LL(1)$ gramatiky k zadané bezkontextové gramatice můžeme shrnout takto:

1. Odstranění levé rekurze:
   $A \to A\alpha \mid \beta$ transformujeme na $A \to \beta A' \mid \beta$, $A' \to \alpha A' \mid \alpha$

2. Levá faktorizace:
   $A \to \alpha\beta \mid \alpha\gamma$ transformujeme na $A \to \alpha A'$, $A' \to \beta \mid \gamma$

3. Rohová substituce:
   $A \to B\gamma$, $B \to \alpha \mid \beta$ transformujeme na $A \to \alpha\gamma \mid \beta\gamma$

4. Pohlcení terminálu:
   $A \to Ba$, $B \to \beta$ transformujeme na $A \to [Ba]$, $[Ba] \to \beta a$

5. Extrakce pravého kontextu:
   $A \to BC$, $C \to a\ \gamma$ transformujeme na $A \to Ba\ \gamma$

6. Substituce za neterminální symbol:
   $A \to \alpha B\beta$, $B \to \gamma \mid \delta$ transformujeme na $A \to \alpha\gamma\beta \mid \alpha\delta\beta$

V následujících příkladech si ukážeme, jak je možno tyto jednoduché transformace použít při získávání $LL(1)$ gramatiky ekvivalentní k zadané bezkontextové gramatice.

11.2 Transformace na $LL(1)$ gramatiky

Příklad 11.1
Je dána gramatika $G = (\{S\}, \{(,)\}, \{S \to S(S) \mid \varepsilon\}, S)$. Tato gramatika není $LL(k)$ pro žádné k, protože obsahuje levou rekurzi. Levou rekurzi odstraníme a dostaneme gramatiku s pravidly:

$$\begin{array}{lcl} S & \to & S' \mid \varepsilon \\ S' & \to & (S)S' \mid (S) \end{array}$$

Tato gramatika také není $LL(1)$, protože pravidla pro S' obsahují kolizi

$FIRST - FIRST$:
$FIRST_1((S)S') \cap FIRST_1((S)) = \{(\}$.

Provedeme levou faktorizaci a dostaneme gramatiku s pravidly:

$$\begin{array}{lcl} S & \to & S' \mid \varepsilon \\ S' & \to & (S)S'' \\ S'' & \to & S' \mid \varepsilon \end{array}$$

Protože pravidla pro S a S'' jsou stejná, budeme S a S'' považovat za stejné neterminály. Po této úpravě dostaneme:

$$\begin{array}{lcl} S & \to & S' \mid \varepsilon \\ S' & \to & (S)S \end{array}$$

Po dosazení za S' do pravidla $S \to S'$ dostaneme výslednou gramatiku

$$G' = (\{S\}, \{(,)\}, \{S \to (S)S \mid \varepsilon\}, S),$$

která je $LL(1)$ (viz příklad 10.7).

Při použití uvedené transformace pro odstranění levé rekurze je nutno provádět vždy levou faktorizaci. Spojíme proto transformaci odstranění levé rekurze a levou faktorizaci:

$$A \to A\alpha \mid \beta$$

transformujeme na

$$\begin{array}{lcl} A & \to & \beta \mid \beta A' \\ A' & \to & \alpha \mid \alpha A'. \end{array}$$

Po levé faktorizaci dostaneme:

$$\begin{array}{lcl} A & \to & \beta A'' \\ A' & \to & \alpha A'' \\ A'' & \to & A' \mid \varepsilon. \end{array}$$

Když dosadíme za A' do pravidla $A'' \to A'$ dostaneme:

$$\begin{array}{lcl} A & \to & \beta A'' \\ A'' & \to & \alpha A'' \mid \varepsilon. \end{array}$$

Můžeme proto nahradit pravidla:

$$A \to A\alpha \mid \beta$$

pravidly:

$$\begin{array}{lcl} A & \to & \beta A' \\ A' & \to & \alpha A' \mid \varepsilon \end{array}$$

Příklad 11.2
Gramatiku $G = (\{S\}, \{(,)\}, \{S \to S(S) \mid \varepsilon\}, S)$ budeme pomocí nové transformace odstranění levé rekurze kombinované s levou faktorizací transformovat na $LL(1)$ gramatiku takto:
Nejdříve získáme pravidla:

$$\begin{aligned} S &\to S' \\ S' &\to (S)S' \mid \varepsilon \end{aligned}$$

Protože díky pravidlu $S \to S'$ platí, že řetězy generované z S i S' jsou stejné, můžeme napsat ekvivalentní pravidla:

$$S \to (S)S \mid \varepsilon$$

Z uvedeného příkladu je vidět, že nový postup vede rychleji k cíli. Je známo, že levou rekurzi lze z gramatiky odstranit vždy. Nyní se budeme zabývat otázkou použití levé faktorizace.

Příklad 11.3
Je dána gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

$$\begin{aligned} S &\to BaS \mid BaA \\ A &\to cA \mid cB \\ B &\to bB \mid \varepsilon \end{aligned}$$

Gramatika není $LL(1)$, protože pravidla pro S a A nesplňují podmínku $FIRST-FIRST$.

Po levé faktorizaci dostaneme gramatiku

$$G' = (\{S, S', A, A', B\}, \{a, b, c\}, P', S),$$

která má v P pravidla:

$$\begin{aligned} S &\to BaS' \\ S' &\to S \mid A \\ A &\to cA' \\ A' &\to A \mid B \\ B &\to bB \mid a \end{aligned}$$

Tato gramatika je $LL(1)$ gramatika.

Je zajímavé, že byla nalezena třída gramatik, pro které je možno levou faktorizací získat $LL(k)$ gramatiky. Jsou to $LP(k)$ gramatiky (**l**eft **p**art), které jsou definovány takto:
Bezkontextová gramatika $G = (N, T, P, S)$ je $LP(k)$ gramatika, jestliže pro každá dvě různá pravidla v P tvaru

$A \to \alpha\beta_1$ a $A \to \alpha\beta_2$,

ve kterých α je nejdelší společná přepona z existence derivací:

$$S \Rightarrow^* wA\gamma \Rightarrow w\alpha\beta_1\gamma$$

$$S \Rightarrow^* wA\gamma' \Rightarrow w\alpha\beta_2\gamma'$$

plyne, že $FIRST_k(\beta_1\gamma) \cap FIRST_k(\beta_2\gamma') = \emptyset$.

Silné $LP(k)$ gramatiky jsou definovány takto:
Jestliže pro každá dvě různá pravidla v P tvaru $A \to \alpha\beta_1$, $A \to \alpha\beta_2$, ve kterých je α nejdelší společná předpona platí:

$FIRST_k(\beta_1 FOLLOW_k(A)) \cap FIRST_k(\beta_2 FOLLOW_k(A)) = \emptyset$,
pak bezkontextová gramatika $G = (N, T, P, S)$ se nazývá silná $LP(k)$ gramatika.

$LP(k)$ gramatika se dá levou faktorizací transformovat na $LL(k)$ gramatiku. Podobně silná $LP(k)$ gramatika se dá levou faktorizací transformovat na silnou $LL(k)$ gramatiku.

Příklad 11.4
Je dána gramatika $G = (\{P, D, S\}, \{begin, end, ;, d, s\}, R, P)$, kde R obsahuje pravidla:

$$\begin{array}{lcl} P & \to & begin\ D; S\ end \\ D & \to & d; D \mid d \\ S & \to & s; S \mid s \end{array}$$

Jednotlivé symboly této gramatiky můžeme interpretovat takto:
P - program,
D - seznam deklarací,
S - seznam příkazů,
d - deklarace.
s - příkaz.

Tato gramatika není $LL(1)$, protože v pravidlech pro D a pro S jsou kolize $FIRST - FIRST$. Tato gramatika dokonce není ani $LP(1)$, protože faktorizací nezískáme $LL(1)$ gramatiku. Po faktorizaci dostaneme totiž gramatiku:

$$G_1 = (\{P, D, D', S, S'\}, \{begin, end, ;, d, s\}, R_1, P),$$

kde R_1 obsahuje pravidla:

$$\begin{array}{lcl} P & \to & begin\ D; S\ end \\ D & \to & dD' \\ D' & \to & ; D \mid \varepsilon \\ S & \to & sS' \\ S' & \to & ; S \mid \varepsilon \end{array}$$

Tato gramatika není $LL(1)$, protože platí $FIRST_1(; D) \cap FOLLOW_1(D') = \{;\}$. V pravidlech pro D' je tedy kolize $FIRST - FOLLOW$. Tuto kolizi je možno odstranit redukcí množiny $FOLLOW_1(D')$ transformací nazývanou pohlcení terminálu.

Redukci množiny $FOLLOW_1(D')$ musíme provést tak, že redukujeme i množinu $FOLLOW_1(D)$, protože platí

$$FOLLOW_1(D') = FOLLOW_1(D)$$

vzhledem k existenci pravidel $D \to dD'$ a $D' \to ; D$.

Provedeme tedy nejdříve pohlcení terminálu ; za symbolem D. Dostaneme gramatiku:

$$G_2 = (\{P, [D;], D, D', S, S'\}, \{begin,\ end,\ ;, d, s\}, R_2, P),$$

kde R_2 obsahuje pravidla:

$$\begin{array}{lcl} P & \to & begin\ [D;]S\ end \\ [D;] & \to & dD'; \\ D & \to & dD' \\ D' & \to & ;D \mid \varepsilon \\ S & \to & sS' \\ S' & \to & ;S \mid \varepsilon \end{array}$$

Jako další krok provedeme pohlcení středníku za symbolem D' a dostaneme gramatiku:

$$G_3 = (\{P, [D;], D, [D';], D', S, S'\}, \{begin,\ end,\ ;, d, s\}, R_3, P),$$

kde R_3 obsahuje pravidla:

$$\begin{array}{lcl} P & \to & begin\ [D;]S\ end \\ [D;] & \to & d[D';] \\ D & \to & dD' \\ [D';] & \to & ;[D;]|; \text{ (zde jsme nahradili } D; \text{ neterminálním symbolem } [D;]) \\ D' & \to & ;D \mid \varepsilon \\ S & \to & sS' \\ S' & \to & ;S \mid \varepsilon \end{array}$$

V této gramatice jsou D a D' zbytečné symboly a můžeme je vynechat. Gramatika G_3 je $LP(1)$, protože levou faktorizací pravidel pro $[D';]$ dostaneme gramatiku:

$$G_4 = (\{P, [D;], [D';], \bar{D}, S, S'\}, \{begin,\ end,\ ;, d, s\}, R_4, P),$$

kde R_4 obsahuje pravidla:

$$\begin{array}{lcl} P & \to & begin\ [D;]S\ end \\ [D;] & \to & d[D';] \\ [D';] & \to & ;\bar{D} \\ \bar{D} & \to & [D;] \mid \varepsilon \\ S & \to & sS' \\ S' & \to & ;S \mid \varepsilon \end{array}$$

Tato gramatika je $LL(1)$. Dosazením za neterminální symbol $[D';]$ získáme gramatiku pro popis struktury bloku, která má tvar:

$$G_5 = (\{P, [D;], \bar{D}, S, S'\}, \{begin,\ end,\ ;, d, s\}, R_5, P),$$

kde R_5 obsahuje pravidla:

(1) $P \rightarrow begin\ [D;]S\ end$
(2) $[D;] \rightarrow d; \bar{D}$
(3) $\bar{D} \rightarrow [D;]$
(4) $\bar{D} \rightarrow \varepsilon$
(5) $S \rightarrow sS'$
(6) $S' \rightarrow ; S$
(7) $S' \rightarrow \varepsilon$

Rozkladová tabulka pro tuto $LL(1)$ gramatiku má tvar:

	begin	*end*	;	*d*	*s*	ε
P	1					
$[D;]$				2		
$\bar{D}$				3	4	
S					5	
S'		7	6			

Příklad 11.5
Metoda pohlcení terminálu nevede vždy k cíli. Příkladem může být gramatika $G = (\{A, B, C\}, \{a, b, c\}, P, A)$, kde A obsahuje pravidla:

$A \rightarrow BaC$
$B \rightarrow \varepsilon \mid abC$
$C \rightarrow c \mid bC$

V této gramatice je kolize $FIRST - FOLLOW$, protože platí:

$$FIRST_1(abC) \cap FOLLOW_1(B) = \{a\}.$$

Po pohlcení terminálního symbolu a v pravidle $A \rightarrow BaC$ a následné levé faktorizaci získáme gramatiku:

$$G_1 = (\{A, C, [Ba], [Ba]'\}, \{a, b, c\}, P_1, A),$$

kde P_1 obsahuje pravidla:

$A \rightarrow [Ba]C$
$C \rightarrow c \mid bC$
$[Ba] \rightarrow a[Ba]'$
$[Ba]' \rightarrow \varepsilon \mid bCa.$

Tato gramatika opět obsahuje kolizi $FIRST - FOLLOW$, protože platí:

$$FIRST_1(bCa) \cap FOLLOW_1([Ba]') = \{b\}.$$

Pokud bychom tímto způsobem pokračovali dále, zjistíme, že odstranění kolize $FIRST - FOLLOW$ zavede vždy další kolizi $FIRST - FOLLOW$.

V tomto případě vede k cíli metoda substituce za neterminální symbol, který způsobuje kolizi $FIRST - FOLLOW$ a následná faktorizace. V našem případě provedeme substituci za symbol B v pravidle $A \rightarrow BaC$ gramatiky G. Dostaneme gramatiku:

$$G_2 = (\{A, C\}, \{a, b, c\}, P_2, A),$$

kde P_2 obsahuje pravidla:

$A \rightarrow aC \mid abCaC$
$C \rightarrow c \mid bC$

Provedeme-li faktorizaci pravidel pro symbol A, dostaneme gramatiku:

$$G_3 = (\{A, A', C\}, \{a, b, c\}, P_3, A),$$

kde P_3 obsahuje pravidla:

$A \rightarrow aA'$
$A' \rightarrow C \mid bCaC$
$C \rightarrow c \mid bC$

Tato gramatika není $LL(1)$ ani $LP(1)$, protože je zde konflikt $FIRST$ - $FIRST$ pro symbol A'. Dosazením za C do pravidla $A' \rightarrow C$, dostaneme gramatiku:

$$G_4 = (\{A, A', C\}, \{a, b, c\}, P_4, A),$$

kde P_4 obsahuje tato pravidla:

$A \rightarrow aA'$
$A' \rightarrow bC \mid bCaC \mid c$
$C \rightarrow c \mid bC$

Tato gramatika je $LP(1)$ a faktorizací pravidel $A' \rightarrow bC \mid bCaC$ dostaneme gramatiku:

$$G_5 = (\{A, A', A'', C\}, \{a, b, c\}, P_5, A),$$

kde P_5 obsahuje tato pravidla:

$A \rightarrow aA'$
$A' \rightarrow bCA'' \mid c$
$A'' \rightarrow \varepsilon \mid aC$
$C \rightarrow c \mid bC$

Tento postup, tj. odstranění kolize $FIRST - FOLLOW$ substitucí za neterminální symbol vede k cíli, tehdy, když neterminální symbol, za který dosazujeme není pravě rekurzivní.

V případě, že neterminální symbol, který způsobuje kolizi $FIRST - FOLLOW$ je pravě rekurzivní, není možno kolizi odstranit.

Příklad 11.6

Převod gramatiky G v příkladu 11.4 na $LL(1)$ gramatiku můžeme začít substitucí za neterminální symbol D v pravidle

$$P \rightarrow \ begin\ D; S\ end$$

Po této substituci dostaneme gramatiku:

$$G_6 = (\{P, D, S\}, \{begin,\ end,\ ;, d, s\}, R_6, P),$$

kde R_6 obsahuje pravidla:

$$\begin{array}{lcl} P & \rightarrow & begin,\ d; D; S\ end\ \mid begin\ d; S\ end \\ D & \rightarrow & d; D \mid d \\ S & \rightarrow & s; S\ \mid s \end{array}$$

Levou faktorizací gramatiky G_6 získáme gramatiku:

$$G_7 = (\{P, P', D, D', S, S'\}, \{begin,\ end,\ ;, d, s\}, R_7, P),$$

kde R_7 obsahuje pravidla:

$$\begin{array}{lcl} P & \rightarrow & begin,\ d; P' \\ P' & \rightarrow & D; S\ end\ \mid S\ end \\ D & \rightarrow & dD' \\ D' & \rightarrow & ; D \mid \varepsilon \\ S & \rightarrow & sS' \\ S' & \rightarrow & ; S\ \mid \varepsilon \end{array}$$

Je vidět, že gramatika G_7 není $LL(1)$ gramatika, protože neterminální symbol D' způsobuje kolizi $FIRST - FOLLOW$ stejným způsobem jako v gramatice G_1 z příkladu 11.4.

Odstranění kolize $FIRST - FOLLOW$ pomocí operací pohlcení terminálu a substituce za neterminální symbol naráží v některých případech na určité problémy.

V případě, že není možno použít substituci za neterminální symbol, protože tento symbol je pravě rekurzivní nebo není možno provést pohlcení terminálu, protože za neterminálem, který způsobuje kolizi se příslušný terminál nevyskytuje, je možno použít extrakci pravého kontextu.

Příklad 11.7

Je dána gramatika $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla:

$$\begin{array}{lcl} S & \rightarrow & AB \mid cB \\ A & \rightarrow & aAcB \mid \varepsilon \\ B & \rightarrow & ab \end{array}$$

Tato gramatika není $LL(1)$, protože platí $FIRST(aAcB) \cap FOLLOW(A) = \{a\}$. Abychom mohli tuto gramatiku transformovat na $LL(1)$ gramatiku, musíme redukovat množinu $FOLLOW(A)$ o symbol a. Skutečnost, že symbol a je prvkem množiny $FOLLOW(A)$, je způsobena pravidlem $A \rightarrow AB$. Protože za symbolem A není přímo terminální symbol a, ale neterminální symbol B, pro který platí $a \in FIRST(B)$, musíme nejprve provést extrakci pravého kontextu symbolu A v pravidle $S \rightarrow AB$ a teprve potom provést pohlcení terminálu. Po extrakci pravého kontextu dostaneme gramatiku s pravidly:

$$\begin{array}{lcl} S & \rightarrow & Aab \mid cB \\ A & \rightarrow & aAcB \mid \varepsilon \\ B & \rightarrow & ab. \end{array}$$

Nyní provedeme pohlcení symbolu a v pravidle $S \rightarrow Aab$. Výsledkem bude gramatika:

$$G_1 = (\{S, A, B, \lfloor Aa \rfloor\}, \{a, b, c\}, P_1, S)$$

s pravidly:

$$\begin{array}{lcl} S & \to & [Aa]b \mid cB \\ B & \to & ab \\ A & \to & aAcB \mid \varepsilon \\ [Aa] & \to & aAcBa \mid a. \end{array}$$

Po levé faktorizaci pravidel pro $[Aa]$ dostaneme $LL(1)$ gramatiku:

$$G' = (\{S, A, B, [Aa], [Aa]'\}, \{a, b, c\}, P', S)$$

s pravidly:

$$\begin{array}{lcl} S & \to & [Aa]b \mid cB \\ [Aa] & \to & a[Aa]' \\ A & \to & aAcB \mid \varepsilon \\ [Aa]' & \to & AcBa \mid \varepsilon \\ B & \to & ab. \end{array}$$

11.3 $LL(k)$ gramatiky

V tomto odstavci si ukážeme některé jazykové konstrukce, které nelze popsat pomocí $LL(k)$ gramatik. Dále si ukážeme, že pro každé $k \geq 1$ existuje jazyk, který je možno generovat $LL(k+1)$ gramatikou a není možno jej generovat $LL(k)$ gramatikou.

Bezkontextový jazyk L se nazývá $LL(k)$ jazyk, jestliže existuje $LL(k)$ gramatika G taková, že $L = L(G)$.

Bezkontextový jazyk L se nazývá LL jazyk, jestliže existuje $LL(k)$ gramatika G pro nějaké $k \geq 0$ taková, že $L = L(G)$.

Příklad 11.8
Nejjednodušším příkladem jazyka, který není $LL(k)$ pro žádné $k \geq 0$ je jazyk:

$$L_1 = \{a^n b^n : n \geq 0\} \cup \{a^n c^n : n \geq 0\}.$$

jazyk L_1 představuje symetrickou závorkovou strukturu, kde ke stejné otevírací závorce přísluší různé zavírací závorky, přičemž zavírací závorky musí být v určitém řetězci vždy stejné. Informaci o tom, jaké mají být použity zavírací závorky můžeme zjisti až po přečtení n otvíracích závorek. Protože n je libovolné, je možno najít takové n, pro které $n > k$, kde k je pevně dané číslo.

Gramatika, která generuje jazyk L_1 má tvar:

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

kde P obsahuje pravidla:

$$\begin{array}{lcl} S & \to & A \mid B \\ A & \to & aAb \mid \varepsilon \\ B & \to & aBc \mid \varepsilon. \end{array}$$

Tato gramatika obsahuje kolizi $FIRST - FIRST$, protože

$FIRST_k(A) \cap FIRST_k(B) = \{a^k\}$

pro libovolné $k \geq 1$.

Příklad 11.9
Je dán jazyk:

$$L_2 = \{a^n w : n \geq 0\}, w \in \{b, c\}^n\}.$$

Tento jazyk se velmi podobá jazyku L_1 z předchozího příkladu. Rozdíl je pouze v tom, že v jazyce L_2 se mohou vyskytovat zavírací závorky libovolného typu v každém řetězci. Přitom platí, že $L_1 \subset L_2$.

Jazyk L_2 je $LL(1)$ jazyk a $LL(1)$ gramatika, která jej generuje má tvar:

$$G = (\{S, A\}, \{a, b, c\}, \{S \to aSA \mid \varepsilon, A \ \to b \mid c\}, S).$$

Příklad 11.10
Jiný příklad jazyka, který není $LL(k)$ pro žádné k uvádí Backhouse (1979). Jedná se o aritmetické a booleovské výrazy generované gramatikou s těmito pravidly:

<výraz> → <booleovský výraz>| <aritmetický výraz>
<booleovský výraz> → <booleovský člen><zbytek booleovského výrazu>
<zbytek booleovského výrazu> → ε | *or* <booleovský člen>
<booleovský člen> → TRUE | FALSE | (<booleovský výraz>)
<aritmetický výraz> → <aritmetický člen><zbytek aritmetického výrazu>
<zbytek aritmetického výrazu> → ε | + <aritmetický člen>
<aritmetický člen> → 0 | 1 | (<aritmetický výraz>)

Tato gramatika je jednoznačná, není levě rekurzivní a přitom není $LL(k)$ pro žádné k a ani se na $LL(k)$ gramatiku nedá transformovat. Důvodem je skutečnost, že v této gramatice jsou možné například tyto dvě derivace:

<výraz> ⇒ <booleovský výraz>
⇒* $(((\cdots$ TRUE $\cdots)))$

<výraz> ⇒ <aritmetický výraz>
⇒* $(((\cdots (0) \cdots)))$

Z těchto derivací je vidět, že druh výrazu (tj. aritmetický nebo booleovský) se dá zjistit až po přečtení úvodní série otevíracích závorek, která může být libovolně dlouhá. V gramatice je tedy neodstranitelná kolize $FIRST - FIRST$, protože

$FIRST_k(\text{<booleovský výraz>}) \cap FIRST_k(\text{<aritmetický výraz>}) = \{(^k\}$

pro libovolné $k \geq 1$.

Příklad 11.11
Další typ jazykové konstrukce, která není $LL(k)$ je jazyk

$$L(G) = \{a^n 0 b^n : n \geq 0\} \cup \{a^n 1 b^{2n} : n \geq 0\},$$

který popisuje gramatika:

$$G = (\{S, A, B\}, \{0, 1, a, b\}, P, S),$$

kde P obsahuje pravidla:

$$\begin{array}{lcl} S & \rightarrow & A \mid B \\ A & \rightarrow & aAb \mid 0 \\ B & \rightarrow & aBbb \mid 1 \end{array}$$

Příklad 11.12
Nyní si ukážeme jazykovou konstrukci, kterou je možno popsat pomocí $LL(k)$ gramatiky.

$L_K = \{a^n w : n \geq 1, w \in \{b, c, b^k d\}^n\}$.

Jazyk L_K je $LL(k)$ jazyk a přitom není $LL(k-1)$ jazyk pro $k \geq 1$. Jazyk L_K je možno generovat gramatikou:

$$G_1 = (\{S, T, A, B\}, \{a, b, c, d\}, P, S),$$

kde P obsahuje pravidla:

$$\begin{array}{lcl} S & \rightarrow & aT \\ T & \rightarrow & SA \mid A \\ A & \rightarrow & bB \mid c \\ B & \rightarrow & b^{k-1}d \mid \varepsilon \end{array}$$

Tato gramatika je $LL(k)$ gramatika, protože platí:

$$FIRST_k(b^{k-1}d) \cap FOLLOW_k(B) = \emptyset \text{ pro } k \geq 1.$$

Gramatika není $LL(k-1)$, protože

$$FIRST_{k-1}(b^{k-1}d) \cap FOLLOW_k(B) = \{b^{k-1}\}.$$

Gramatika G_1 se nedá transformovat na $LL(k-1)$ gramatiku.

Jazyk L_K se dá generovat také následující $LL(k+1)$ gramatikou:

$$G_2 = (\{S, A\}, \{a, b, c, d\}, P, S),$$

kde P obsahuje pravidla:

$$\begin{array}{lcl} S & \rightarrow & aSA \mid aA \\ A & \rightarrow & b^k d \mid b \mid c. \end{array}$$

Tato gramatika je ekvivalentní s gramatikou G_1 a je $LL(k+1)$. Důkaz ekvivalence je jednoduchý:

Provedeme levou faktorizaci gramatiky G_2 a dostaneme gramatiku G_1. Vztah mezi gramatikami G_1 a G_2 má obecnější platnost:

Každou $LL(k+1)$ gramatiku bez ε–pravidel lze transformovat na $LL(k)$ gramatiku.

Z příkladu 11.12 je vidět, že množina $LL(k)$ jazyků je vlastní podmnožinou $LL(k+1)$ jazyků pro libovolné $k \geq 1$.

Příklad 11.13
Je zajímavé si uvědomit, jak vypadá množina $LL(0)$ jazyků. Nejdříve odpovězme na otázku, jaký tvar má $LL(0)$ gramatika. K tomu, aby bylo možno rozhodnout se při syntaktické analýze jednoznačně pro určitou expanzi bez znalosti dalších vstupních symbolů, je nutno, aby pro každý neterminální symbol bylo v gramatice právě jedno pravidlo. Taková gramatika generuje jediný řetězec, protože je v ní možná jediná derivace. $LL(0)$ jazyk je tedy takový jazyk, který obsahuje právě jeden řetězec. Můžeme tedy říci, že množina $LL(0)$ jazyků je vlastní podmnožinou $LL(1)$ jazyků.

Můžeme tedy učinit na základě úvahy v příkladu 11.13 a příkladu 11.12 závěr, že množina $LL(k)$ jazyků je vlastní podmnožinou $LL(k+1)$ jazyků pro libovolné $k \geq 0$. Totéž tvrzení platí o vztahu množin $LL(k)$ a $LL(k+1)$ gramatik.

11.4 Příklady pro cvičení

Cvičení 11.14
Pro gramatiky, které jsou uvedeny v celém tomto textu nebo vzniknou jako výsledky cvičení, proveďte toto:

1. Zjistěte, zda gramatika je $LL(1)$. Jestliže ano, sestrojte pro ni rozkladovou tabulku.

2. Pokud gramatika není $LL(1)$, transformujte ji na $LL(1)$ gramatiku a sestrojte pro ni rozkladovou tabulku.

3. Pokud gramatika není $LL(1)$, ani se nedá na $LL(1)$ gramatiku transformovat, zjistěte, proč je tomu tak.
   Důvody mohou být:

   (a) gramatika je nejednoznačná,

   (b) gramatika generuje jazyk, který obsahuje konstrukci, kterou nelze $LL(1)$ gramatikou generovat,

   (c) známými transformacemi nelze požadovanou transformaci provést.

Cvičení 11.15
Je dána gramatika $G = (\{E, E', T, T', F\}, \{+, *, a, (,)\}, P, E)$, kde P obsahuje pravidla:

$E \to TF'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to a \mid (E).$

Převeďte tuto gramatiku na q-gramatiku.

Cvičení 11.16
Je dána gramatika
$G = (\{S\}, \{a\}, \{S \to aS \mid a\}, S)$.
Ukažte, že tato gramatika je $LL(2)$. Pokuste se tuto gramatiku transformovat na $LL(1)$ gramatiku.

Cvičení 11.17
Je dána gramatika $G = (\{S, T\}, \{a, (,)\}, P, S)$, kde P obsahuje pravidla:

$S \to a \mid (T) \mid \varepsilon$
$T \to T, S \mid S.$

Převeďte tuto gramatiku na $LL(1)$ gramatiku a sestrojte pro ni syntaktický analyzátor.

Cvičení 11.18
Je dána gramatika booleovského výrazu takto:

$G = (\{S, F, T, E\}, \{\vee, \wedge, \neg, (,), a\}, P, E)$,

kde P obsahuje pravidla:

$E \to E \vee T \mid T$
$T \to T \wedge F \mid F$
$F \to \neg S \mid S$
$S \to (E) \mid a$

Transformujte tuto gramatiku na $LL(1)$ gramatiku a sestrojte pro ni syntaktický analyzátor.

Cvičení 11.19
Je dána gramatika

$G_i = (\{F, T\}, \{\uparrow, (,), a\}, P_i, F)$, $i = 1, 2$,

kde P_1 obsahuje pravidla:

$F \to F \uparrow T \mid T$
$T \to (F) \mid a$

a P_2 obsahuje pravidla:

$F \to T \uparrow F \mid T$
$T \to (F) \mid a.$

Převeďte gramatiky G_1 a G_2 na $LL(1)$ gramatiky.

Cvičení 11.20
Je dána gramatika
$G = (\{P, L, S\}, \{begin, end, ;, s\}, R, P)$,
kde R obsahuje pravidla:
$P \rightarrow begin\, L\, end$
$L \rightarrow L\,;\, S \mid S$
$S \rightarrow s.$
Transformujte tuto gramatiku na $LL(1)$ gramatiku a sestrojte pro ni syntaktický analyzátor.

Cvičení 11.21
Je dána gramatika
$G = (\{A, B, C\}, \{a, b, c, d\}, P, A)$, kde P obsahuje pravidla:
$A \rightarrow AcB \mid cC \mid C$
$B \rightarrow bB \mid d$
$C \rightarrow CaB \mid BbB \mid B$
Transformujte tuto gramatiku na $LL(1)$ gramatiku a sestrojte pro ni syntaktický analyzátor.

12 Formální překlady

12.1 Základní pojmy

Formální překlad je binární relace $Z \subset L \times V$. *Definiční obor* této relace je množina L a obor hodnot je množina V. Relace Z přiřazuje každému prvku w množiny L množinu jeho překladů $Z(w)$ z množiny V. Pokud $Z(w)$ obsahuje pro každé $w \in L$ nejvýše jeden prvek, pak Z je funkce (případně parciální) a takový překlad se nazývá jednoznačný.

Překladová gramatika je pětice $PG = (N, T, D, R, S)$, kde
N je konečná množina neterminálních symbolů,
T je konečná množina symbolů, které budeme nazývat vstupní symboly,
D je konečná množina symbolů, které budeme nazývat výstupní symboly,
R je konečná množina pravidel tvaru $A \to \alpha$, kde $A \in N$, $\alpha \in (N \cup T \cup D)^*$,
S je počáteční symbol.
Přitom platí, že $T \cap D = \emptyset$ a $(T \cup D) \cap N = \emptyset$.

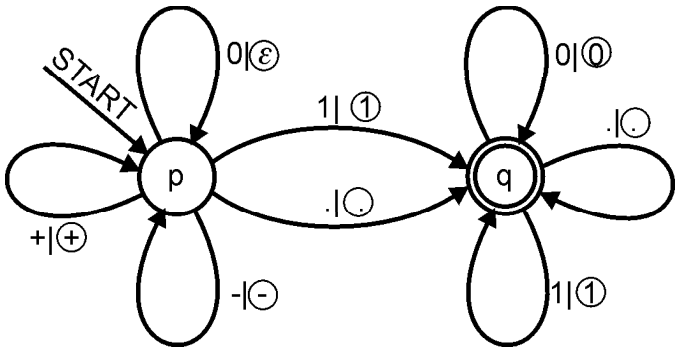
Konečný překladový automat je šestice $KPA = (Q, T, D, \delta, q_0, F)$, kde
Q je konečná množina vnitřních stavů,
T je konečná množina symbolů, které budeme nazývat vstupní symboly,
D je konečná množina symbolů, které budeme nazývat výstupní symboly,
δ je zobrazení z $Q \times (T \cup \{\varepsilon\})$ do množiny konečných podmnožin z $Q \times D^*$,
$q_0 \in Q$ je počáteční stav,
$F \subset Q$ je množina koncových stavů.

Zásobníkový překladový automat je osmice $ZPA = (Q, T, G, D, \delta, q_0, Z_0, F)$, kde
Q je konečná množina vnitřních stavů,
T je konečná množina symbolů, které budeme nazývat vstupní symboly,
D je konečná množina symbolů, které budeme nazývat výstupní symboly,
G je konečná množina symbolů, které budeme nazývat zásobníkové symboly,
δ je konečné zobrazení z $Q \times (T \cup \{\varepsilon\}) \times G^*$ do množiny konečných podmnožin $Q \times G^* \times D^*$,
$q_0 \in Q$ je počáteční stav,
$Z_0 \in G$ je počáteční symbol zásobníku,
$F \subset Q$ je množina koncových stavů.

12.2 Regulární překlady

Příklad 12.1
Sestrojme konečný překladový automat, který bude v binárním zlomku zapsaném ve tvaru s „binární“ tečkou vynechávat nevýznamné nuly před první jedničkou v celé části čísla nebo před tečkou. Konečný překladový automat má tvar:

$KPA = (\{p, q\}, \{+, -, ., 0, 1\}, \{\oplus, \ominus, \odot, ⓪, ①\}, p, \delta, \{q\})$,
kde zobrazení δ je zadáno přechodovým diagramem na obr. 12.1. Jako vstup



Obrázek 12.1: Přechodový diagram konečného překladového automatu z příkladu 12.1

tohoto automatu se předpokládá správně zapsané číslo s „binární“ tečkou. V tomto a dalších přechodových diagramech jsou hrany ohodnoceny dvojicemi $i|o$ (vstup|výstup).

12.2.1 Formální překlady výrazů

Výrazy jsou jazykové konstrukce, které se vyskytují téměř ve všech programovacích jazycích. Výraz je příkladem složené operace. Můžeme si jej představit jako jednu operaci s operandy, kterými mohou být hodnoty operandů nebo výrazy. Dá se říci, že výraz můžeme interpretovat jako strom, jehož vnitřní uzly jsou operátory a koncové uzly (listy) jsou operandy. Výraz zapsaný v obvyklém lineárním infixovém zápisu můžeme reprezentovat různými stromy, když jej budeme interpretovat podle různých jazykových definic.

Na obr. 12.2a) je uveden strom pro levě asociativní chápání operátorů bez priority (jazyk PL/360), na obr. 12.2b) je uveden strom pro použití obvyklých priorit operátorů (většina programovacích jazyků). Na obr. 12.2c) je uveden strom pro pravé asociativní chápání operátorů bez priorit (jazyk APL).

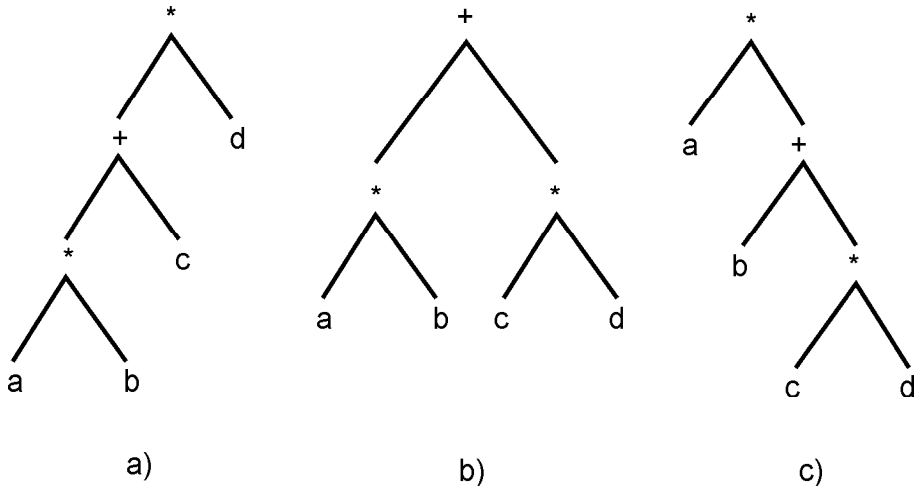
Kromě infixového zápisu můžeme výrazy zapisovat také v prefixovém a postfixovém zápisu. V tomto případě se jedná o prefixový nebo postfixový lineární zápis stromu výrazu. Výraz $a * b + c * d$, jehož význam je vyjádřen stromem na obr. 12.2b) má :

a) prefixový zápis $+ * ab * cd$,

b) postixový zápis $ab * cd * +$.

Uvedené zápisy výrazů můžeme charakterizovat takto :

- v infixovém zápisu jsou operátory uvedeny mezi operandy,
- v prefixovém zápisu jsou operátory uvedeny před operandy,
- v postfixovém zápisu jsou operátory uvedeny za operandy.



Obrázek 12.2: Stromy pro výraz $a * b + c * d$

Příklad 12.2
Sestrojme regulární překladovou gramatiku a konečný překladový automat, který popisuje překlad výrazu bez závorek a s operátory $+, -, *, /$ z infixového zápisu do postfixového. Všechny operátory ve výrazu mají stejnou prioritu a levou asociativu. Vstupní výrazy jsou zakončeny symbolem $=$.

a) Regulární překladová gramatika
$RPG = (\{S, A, A_P, A_M, A_K, A_L\}, \{a, +, -, *, /, =\}, \{@, \oplus, \ominus, \circledast, \oslash\}, R, S)$, kde R obsahuje pravidla:

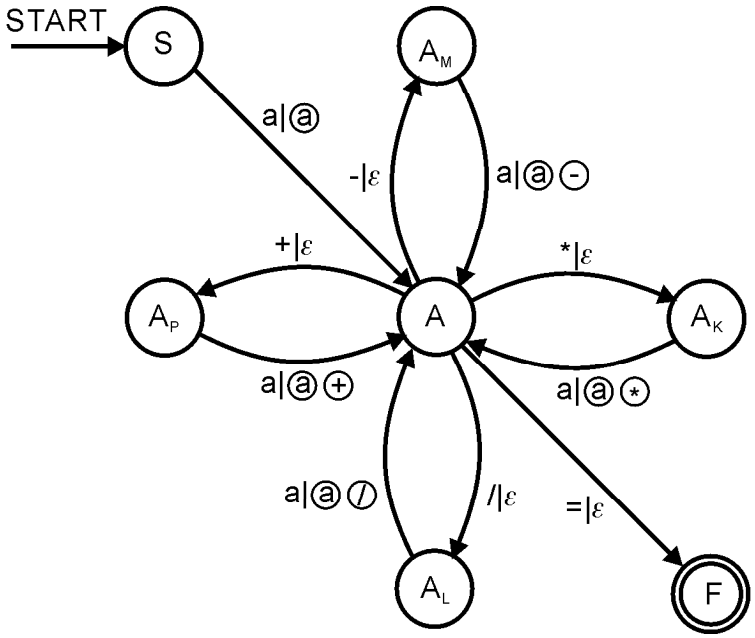
$S \to a\,@A$	$A \to =$
$A \to +A_P$	$A_P \to a\,@\oplus A$
$A \to -A_M$	$A_M \to a\,@\ominus A$
$A \to *\,A_K$	$A_K \to a\,@\circledast A$
$A \to /\,A_L$	$A_L \to a\,@\oslash A$

b) Konečný překladový automat
$KPA = (\{S, F, A, A_P, A_M, A_K, A_L\}, \{a, +, -, *, /, =\}, \{@, \oplus, \ominus, \circledast, \oslash\}, \delta, S, F)$. Zobrazení δ je zadáno přechodovým diagramem na obr. 12.3.

Sestrojená gramatika a automat popisuje překlad, do kterého patří například tyto dvojice řetězců:

$(a =,$	$@)$
$(a + a =,$	$@@\oplus)$
$(a - a * a =,$	$@@\ominus@\circledast)$
$(a/a + a =,$	$@@\oslash@\oplus).$

Konečný překladový automat můžeme interpretovat jako model kalkulačky. Vstupní symboly budeme chápat v tomto případě jako příkazy pro provedení



Obrázek 12.3: Přechodový diagram konečného překladového automatu z příkladu 12.2

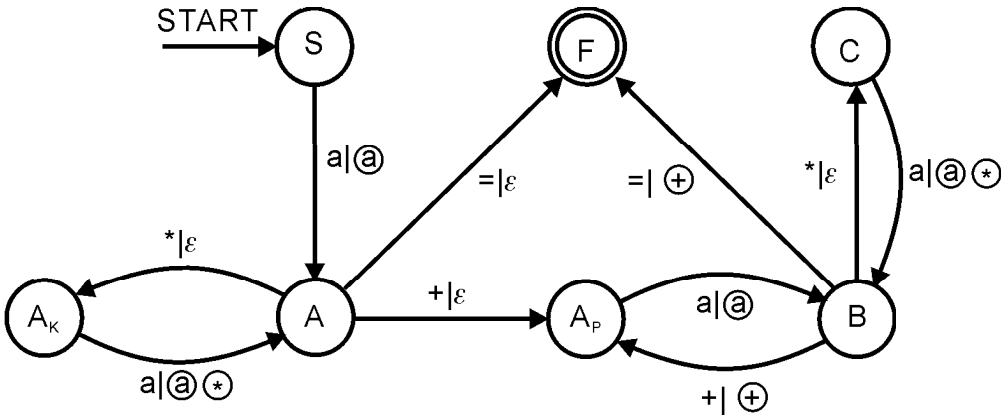
jednotlivých operací. Kalkulačka bude mít dva registry, které označíme jako X a Y. Vstupní symbol a představuje jedno číslo. Kalkulačka bude při jednotlivých přechodech provádět tyto operace:

- při přechodu ze stavu S do stavu A se provede uložení čteného čísla do registru X,
- při přechodech ze stavů A_P, A_M, A_K, A_L do stavu A se provede:
  a) uložení čteného čísla do registru Y,
  b) příslušná aritmetická operace $(+, -, *, /)$ s obsahem registrů X a Y, výsledek se uloží do registru X.

Kalkulačku je možno doplnit displejem, na kterém se zobrazuje vždy zadávané číslo nebo výsledek prováděné operace.

Příklad 12.3
Sestrojíme konečný překladový automat, který překládá výrazy zapsané v infixovém zápisu do postfixového zápisu. Vstupní výrazy obsahují operátory + a $*$ s běžnou prioritou, to znamená, že operátor $*$ má vyšší prioritu než operátor +. Vstupní výrazy neobsahují závorky a jsou ukončeny symbolem =.

Konečný překladový automat $KPA = (\{S, F, A, A_P, A_K, B, C\}, \{a, +, *, =\}, \{@, \oplus, \circledast\}, \delta, S, \{F\})$. Zobrazení δ je definováno přechodovým diagramem na obrázku 12.4.



Obrázek 12.4: Přechodový diagram konečného překladového automatu z příkladu 12.3

Tento automat můžeme opět interpretovat jako model kalkulačky. V tomto případě má kalkulačka tři registry X, Y, Z, symbol a představuje vstupující číslo. Při jednotlivých přechodech bude kalkulačka provádět operace podle tabulky 12.2.1. Zápis $a \rightarrow X$ představuje operaci uložení čteného čísla do registru

výchozí stav	cílový stav	prováděné operace
S	A	$a \rightarrow X$
A_K	A	$a \rightarrow Y; \langle X \rangle * \langle Y \rangle \rightarrow X$
A_P	B	$a \rightarrow Y$
C	B	$a \rightarrow Z\ ; \langle Y \rangle * \langle Z\ \rangle \rightarrow Y$
B	F	$\langle X \rangle + \langle Y \rangle \rightarrow X$
B	A_P	$\langle X \rangle + \langle Y \rangle \rightarrow X$

Tabulka 12.1:

X, zápis $\langle X \rangle op \langle Y \rangle \rightarrow Z$ znamená, že s obsahy registrů X a Y je provedena operace op a výsledek je uložen opět do registru Z. V případě, že by tato kalkulačka byla vybavena displejem, bylo by vhodné ve stavech A, AP a F zobrazovat hodnotu uloženou v registru X, ve stavu B zobrazovat hodnotu uloženou v registru Y.

Příklad 12.4
Sestrojíme konečný překladový automat pro překlad výrazu z infixového do postfixového zápisu. Vstupní výrazy obsahují operátory + a $*$ se stejnou prioritou a závorky, ale jen na jedné úrovni. To znamená, že závorky nemohou být vnořovány do sebe. Přechodový diagram konečného automatu je na obr. 12.5. Také tento automat je možné interpretovat jako model kalkulačky.

12.3 Bezkontextové formální překlady

V několika dalších příkladech ukážeme základní principy překladu výrazů z jednoho zápisu do jiného. Nejdříve uvedeme bezkontextové gramatiky, které generují aritmetické výrazy s operátory +, $*$ a závorkami v infixovém, prefixovém a postfixovém zápisu. Pravidla gramatik pro jednotlivé zápisy jsou uvedena v následující tabulce.

gramatika	*GPRE*	*GIN*	*GPOST*
zápis výrazu	prefixový	infixový	postfixový
pravidla			
(1)	$E \to +EE$	$E \to E+T$	$E \to EE+$
(2)		$E \to T$	
(3)	$E \to *EE$	$T \to T*F$	$E \to EE*$
(4)		$T \to F$	
(5)	$E \to a$	$F \to a$	$E \to a$
(6)		$F \to (E)$	
druh	$LL(1)$	$LR(1)$	$LR(1)$

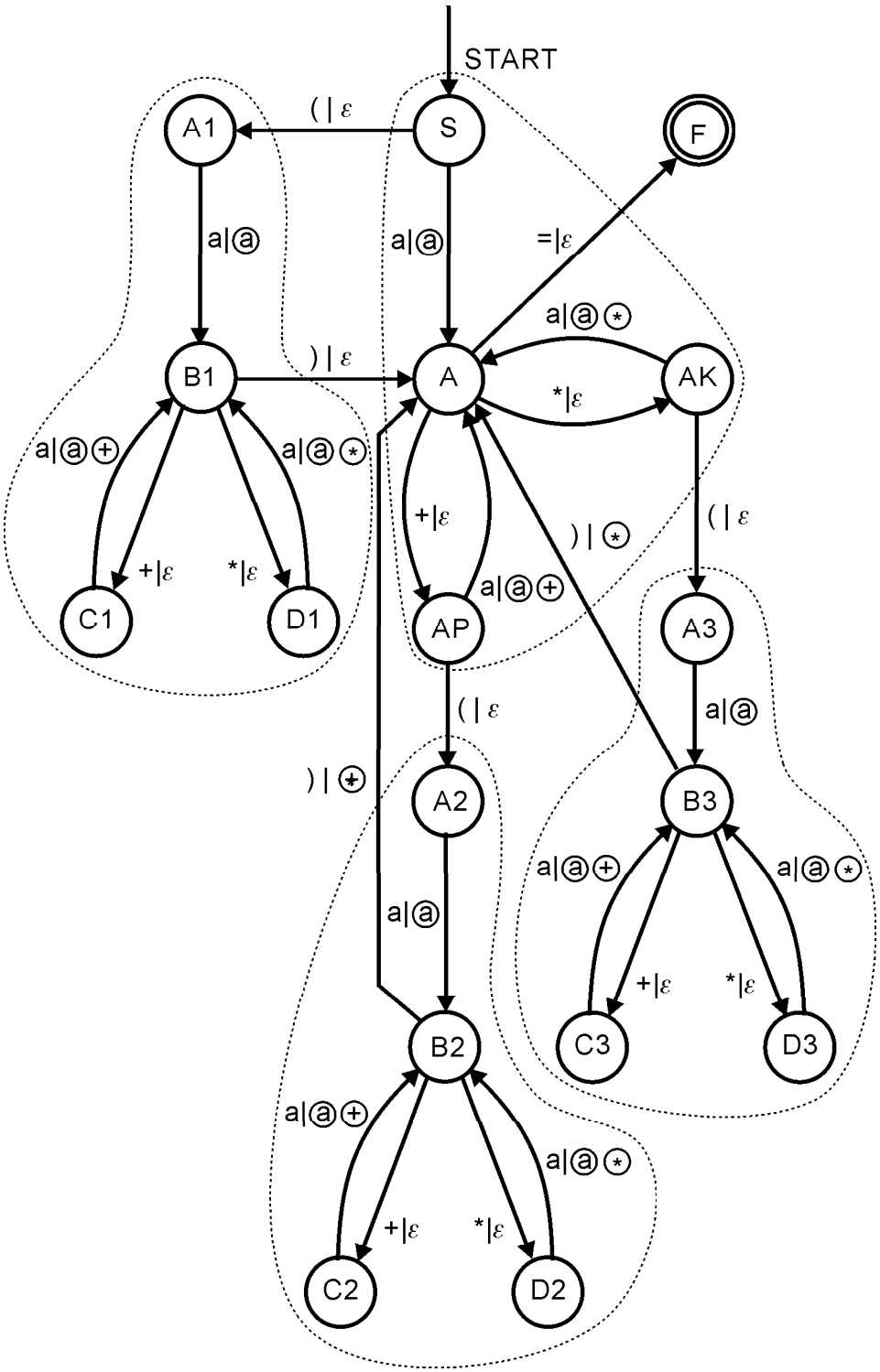
Neterminální symboly jsou E, T, F, terminální symboly jsou a, +, $*$, (,). Závorky se vyskytují jen u výrazů v infixovém zápisu. V následujících šesti příkladech jsou studovány překladové gramatiky pro jednotlivé překlady.

Příklad 12.5
Pro překlad výrazu z infixového do postfixového zápisu můžeme použít tyto dvě překladové gramatiky:
$PGINPOST1 = (\{E, T, F\}, \{a, +, *, (,)\}, \{@, \oplus, \circledast\}, R_1, E)$, kde R_1 obsahuje pravidla:

$$\begin{array}{lll} E \to E+T\oplus & T \to T*F\circledast & F \to a@ \\ E \to T & T \to F & F \to (E) \end{array}$$

Tato gramatika není překladová $LL(1)$ gramatika. Dá se však transformovat na překladovou $LL(1)$ gramatiku, která má tvar:
$PGINPOST2 = (\{E, E', T, T', F\}, \{a, +, *, (,)\}, \{@, \oplus, \circledast\}, R_2, E)$, kde R_2 obsahuje pravidla:

$$\begin{array}{lll} E \to TE' & T \to FT' & F \to a@ \\ E' \to +T\oplus E' & T' \to *F\circledast T' & F \to (E) \\ E' \to \varepsilon & T' \to \varepsilon & \end{array}$$



Obrázek 12.5: Přechodový diagram konečného překladového automatu z příkladu 12.4

Z tvaru této překladové gramatiky je vidět, že překlad z infixového do postfixového zápisu je možno provést deterministicky při syntaktické analýze metodou shora dolů.

Příklad 12.6
Při překladu výrazu z prefixového do infixového zápisu je třeba zajistit doplnění závorek. Závorky budeme doplňovat jen pro operaci sčítání.

$PGPREIN = (\{E\}, \{a, +, *\}, \{@, \oplus, \circledast, (,)\}, R, E)$ kde R obsahuje pravidla:

$$E \to +(E \oplus E) \qquad E \to *E \circledast E \qquad E \to a@$$

Tato překladová gramatika má vstupní gramatiku $LL(1)$ a je proto možný deterministický překlad z prefixového do infixového zápisu výrazu. Při použití uvedené překladové gramatiky mohou vzniknout výrazy, ve kterých jsou redundantní závorky. Odstranění redundatních závorek je možno provést pomocí překladové gramatiky z příkladu 12.11.

Příklad 12.7
Překlad výrazu z postfixového do infixového zápisu vyžaduje doplnění závorek podobně jako při překladu výrazu z prefixového do infixového zápisu. Překladová gramatika má tvar:
$PGPOSTIN = (\{E\}, \{a, +, *\}, \{@, \oplus, \circledast, (,)\}, R, E)$, kde R obsahuje pravidla:

$$E \to (E \oplus E +) \qquad E \to E \circledast E * \qquad E \to a@$$

Z tvaru pravidel je vidět, že tato gramatika obsahuje výstupní symbol před levě rekurzivním neterminálním symbolem a proto není možné sestrojit deterministický zásobníkový překladový automat pro překlad z postfixového do infixového zápisu. Tento překlad je možné popsat pomocí atributové gramatiky viz příklad 3.16.

Příklad 12.8
Překladová gramatika pro překlad výrazu z prefixového do postfixového zápisu má tvar:
$PGPREPOST = (\{E\}, \{a, +, *\}, \{@, \oplus, \circledast\}, R, E)$, kde R obsahuje pravidla:

$$E \to +EE\oplus \qquad E \to *EE\circledast \qquad E \to a@$$

Tato překladová gramatika má vstupní gramatiku $LL(1)$, a proto je možný deterministický překlad z prefixového do postfixového zápisu výrazu.

Příklad 12.9
Pro překlad výrazu z infixového zápisu do prefixového sestrojíme překladovou gramatiku:

$PGINPRE = (\{E, T, F\}, \{a, +, *, (,)\}, \{@, \oplus, \circledast\}, R, E)$, kde R obsahuje pravidla:

$$E \to \oplus E + T \qquad T \to \circledast T * F \qquad F \to a@$$
$$E \to T \qquad\qquad T \to F \qquad\qquad F \to (E)$$

Tato gramatika není překladová $LL(k)$ gramatika pro žádné $k \geq 0$. Důvodem je přítomnost výstupních symbolů před levě rekurzivními neterminálními symboly v pravidlech $E \to \oplus E+T$ a $T \to \circledast T*F$. Z tohoto důvodu se uvedená gramatika nedá transformovat na žádný tvar, kterému by odpovídal deterministický zásobníkový automat. Překlad výrazu z infixového zápisu do prefixového nelze provést deterministickým zásobníkovým překladovým automatem. Tento překlad je možné popsat pomocí atributové gramatiky viz příklad 3.16.

Příklad 12.10
Překlad výrazu z postfixového do prefixového zápisu popisuje překladová gramatika $PGPOSTPRE = (\{E\}, \{a, +, *\}, \{@, \oplus, \circledast\}, R, E)$, kde R obsahuje pravidla:

$$E \to \oplus EE+ \qquad E \to \circledast EE* \qquad E \to a@$$

Tato překladová gramatika má podobné vlastnosti jako gramatika z předchozího příkladu, a proto není možné provést překlad výrazu z postfixového do prefixového zápisu deterministickým zásobníkovým překladovým automatem. Tento překlad je možné popsat pomocí atributové gramatiky viz příklad 3.16.

Následující tabulka shrnuje základní vlastnosti překladu výrazu z jedné formy do jiné. Některé překladové gramatiky (*PGINPOST*, *PGPOSTPRE*, *PGINPRE*, *PGPOSTIN*) mají $LL(1)$ vstupní gramatiky, což dovoluje použití algoritmu formálního překladu řízeného LL analyzátorem.

Z překladové tabulky je zřejmé, že překladové gramatiky *PGPOSTPRE*, *PGINPRE* a *PGPOSTIN* vedou na nedeterministické překlady. To znamená, že tyto překlady nelze provést deterministickým zásobníkovým automatem. Překážkou je skutečnost, že výstupní datová struktura je řetězec, do kterého lze připojovat symboly pouze na konec. Tyto překlady lze však popsat pomocí atributových gramatik, které uvedené překlady popisují za předpokladu, že výstupní datová struktura je seznam nebo strom. V tom případě je možno i tyto překlady provádět deterministicky.

překladová gramatika	důležitá pravidla	typ vstupní gramatiky	typ překladu
$PGINPOST1$	$E \to E + T\oplus$ $T \to T * F\circledast$	$LR(1)$	$LR(1)$
$PGINPOST2$	$E' \to +T\oplus E'$ $T' \to *F\circledast T'$	$LL(1)$	$LL(1)$
$PGPREPOST$	$E \to +EE\oplus$ $E \to *EE\circledast$	$LL(1)$	$LL(1)$
$PGPREIN$	$E \to +(\!)E\oplus E(\!)$ $E \to *E\circledast E$	$LL(1)$	$LL(1)$
$PGPOSTPRE$	$E \to \oplus EE+$ $E \to \circledast EE*$	$LR(1)$	nedeterministický
$PGINPRE$	$E \to \oplus E + T$ $T \to \circledast T * F$	$LR(1)$	nedeterministický
$PGPOSTIN$	$E \to (\!)E\oplus E + (\!)$ $E \to E\circledast E*$	$LR(1)$	nedeterministický

Příklad 12.11
Překladová gramatika, která popisuje překlad výrazů s vynecháním redundantních závorek má tvar:
$PG = (\{E, T, F\}, \{a, +, *, (,)\}, \{@, \oplus, \circledast, (\!), (\!), \}, R, E)$, kde R obsahuje pravidla:

$E \to (E)$	$T \to (T)$	$F \to (\!)E + \oplus E)(\!)$
$E \to E + \oplus E$	$T \to F * \circledast F$	$F \to T$
$E \to T$	$T \to a\ @$	

Tato překladová gramatika popisuje překlad, při kterém je například výraz $((a + (a * a)) * a)$ přeložen na výraz $(a + a * a) * a$. Připomeňme, že vstupní gramatika není jednoznačná, ale každému vstupnímu řetězci odpovídá jen jeden výstupní řetězec.

12.3.1 Typické bezkontextové překlady

Dále uvedeme několik příkladů zásobníkových překladových automatů, které provádějí jednoduché syntaxí řízené překlady. Přitom budeme pro zápis zobrazení δ používat přechodový diagram podobně jako u konečných překladových automatů.
Hranu v přechodovém diagramu ohodnotíme trojicí $(a, \underline{\alpha}\beta, z)$, kde
$a \in T \cup \{\varepsilon\}$ je čtený symbol nebo prázdný řetězec,
$\underline{\alpha} \in G^*$ je řetězec, který je ze zásobníku vyloučen,
$\beta \in G^*$ je řetězec, který je do zásobníku vložen,
$z \in D^*$ je řetězec, který je přidán k výstupnímu řetězci.

V dalším příkladu uvedeme zásobníkové překladové automaty pro typické „zásobníkové“ překlady.

Příklad 12.12
Uvedeme zásobníkové překladové automaty, které provádějí tyto překlady:

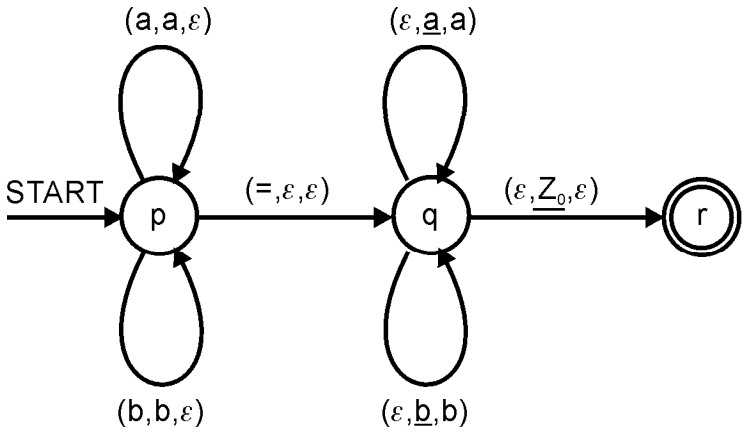
a) (w, w^R)

b) (w, ww^R),

kde w je libovolný řetězec nad abecedou $\{a, b\}$.
Zásobníkové překladové automaty jsou tyto:

a) $ZPA = (\{p, q, r\}, \{a, b, =\}, \{Z_0, a, b\}, \delta, p, Z_0, \{r\})$, kde δ je znázorněno přechodovým diagramem na obr. 12.6.

b) $ZPA = (\{p, q, r\}, \{a, b, =\}, \{Z_0, a, b\}, \{a, b\}, \delta, p, Z_0, \{r\})$, kde δ je znázorněno přechodovým diagramem na obr. 12.7.

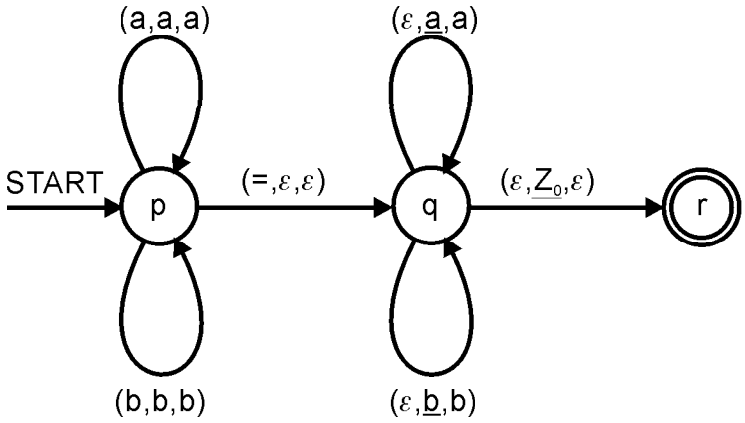


Obrázek 12.6: Přechodový diagram zásobníkového překladového automatu pro překlad (w, w^R)

Oba tyto automaty pracují různě jen ve stavu p při čtení vstupního řetězce. Automat pro překlad a) čte vstupní symboly a ukládá je do zásobníku. Automat pro překlad b) při čtení a ukládání do zásobníku ještě navíc vstupní symboly ukládá do výstupního řetězce. Ve stavu q oba automaty kopírují obsah zásobníku do výstupního řetězce.

Příklad 12.13
Sestrojíme překladovou gramatiku, která generuje překlad $\{(a^i b^j, x^j y^i) : i, j > 0)\}$.

Překladová gramatika má tvar $PG = (\{A, B\}, \{a, b\}, \{ⓧ, ⓨ\}, R, A)$, kde R obsahuje pravidla:



Obrázek 12.7: Přechodový diagram zásobníkového překladového automatu pro překlad (w, ww^R)

(1) $A \to aA\textcircled{y}$
(2) $A \to a\textcircled{y}B$
(3) $B \to b\textcircled{x}B$
(4) $B \to b\textcircled{x}$

Vstupní gramatika této překladové gramatiky je regulární, ale překlad nelze provést konečným překladovým automatem. Důvodem je pravidlo (1), jehož pravá strana obsahuje výstupní symboly až za neterminálním symbolem. Takový překlad je nutno provést zásobníkovým překladovým automatem.

12.4 Implementace formálních překladů

V tomto odstavci uvedeme příklad implementace formálního překladu, který vznikne rozšířením algoritmu syntaktické analýzy. Takové překlady se nazývají syntaxí řízené překlady.

Pro formální překlad při analýze shora dolů lze použít algoritmus, pomocí kterého je možno provést překlad pro každou překladovou gramatiku, jejíž vstupní gramatika je $LL(1)$. To znamená, že výstupní symboly mohou být v pravidlech gramatiky na libovolných místech.

Příklad 12.14
Sestrojme rozkladovou tabulku pro překladovou gramatiku $PG = (\{E, E', T, T', F\}, \{+, *, a, (,)\}, \{\oplus, \circledast, @\}, P, E)$ s pravidly:
$E \to TE'$ $\quad E' \to +T\oplus E'$ $\quad E' \to \varepsilon$
$T \to FT'$ $\quad T' \to *F\circledast T'$ $\quad T' \to \varepsilon$
$F \to (E)$ $\quad F \to a@$

Je zřejmé, že vstupní gramatika této překladové gramatiky je $LL(1)$, a proto pro G můžeme sestrojit rozkladovou tabulku.

	a	$+$	$*$	$($	$)$	ε
E	TE'			TE'		
E'		$+T\oplus E'$			ε	ε
T	FT'			FT'		
T'		ε	$*F\circledast T'$		ε	ε
F	$a@$			(E)		

Překlad vstupní věty $a + a * a$ proběhne tímto způsobem:

$$
\begin{array}{lrrll}
(a+a*a, E, \varepsilon) & \vdash (& a+a*a, & TE', & \varepsilon) \\
& \vdash (& a+a*a, & FT'E', & \varepsilon) \\
& \vdash (& a+a*a, & a@T'E', & \varepsilon) \\
& \vdash (& +a*a, & @T'E', & \varepsilon) \\
& \vdash (& +a*a, & T'E', & @) \\
& \vdash (& +a*a, & E', & @) \\
& \vdash (& +a*a, & +T\oplus E', & @) \\
& \vdash (& a*a, & T\oplus E', & @) \\
& \vdash (& a*a, & FT'\oplus E', & @) \\
& \vdash (& a*a, & a@T'\oplus E', & @) \\
& \vdash (& *a, & @T'\oplus E', & @) \\
& \vdash (& *a, & T'\oplus E', & @@) \\
& \vdash (& a, & *F\circledast T'\oplus E', & @@) \\
& \vdash (& a, & F\circledast T'\oplus E', & @@) \\
& \vdash (& a, & a@\circledast T'\oplus E', & @@) \\
& \vdash (& \varepsilon, & @\circledast T'\oplus E', & @@) \\
& \vdash (& \varepsilon, & \circledast T'\oplus E', & @@@) \\
& \vdash (& \varepsilon, & T'\oplus E', & @@@\circledast) \\
& \vdash (& \varepsilon, & \oplus E', & @@@\circledast) \\
& \vdash (& \varepsilon, & E', & @@@\circledast\oplus) \\
& \vdash (& \varepsilon, & \varepsilon, & @@@\circledast\oplus)
\end{array}
$$

Odpovídajícím výstupem pro vstup $a + a * a$ je řetězec $@@@\circledast\oplus$.

Implementaci formálního překladu lze provést pomocí postupu známého jako „rekurzivní sestup“ (odst. 5.4 [JPR03]).

Soubor rekurzivních procedur pro překladovou gramatiku bude používat proceduru `SROVNEJ`, jejíž deklaraci si můžeme představit takto:

```
var SYMBOL : VSTUPNÍSYMBOL;
    CHYBA  : BOOLEAN;
procedure SROVNEJ (X: VSTUPNÍSYMBOL);
begin
   if X = SYMBOL then LEXIKÁLNÍANALÝZA(SYMBOL)
                 else CHYBA := TRUE;
end;
```

Procedura LEXIKÁLNÍANALÝZA přečte ze vstupního souboru jeden symbol a uloží jej do výstupního parametru SYMBOL.
Vlastní soubor rekurzivních procedur bude mít tvar:

```
procedure E;
begin
   case SYMBOL of
      A,OTVZAV : begin T; ZE; end;
      otherwise : CHYBA := TRUE;
   end;
end;
procedure ZE;
begin
   case SYMBOL of
      PLUS : begin SROVNEJ(PLUS); T; VYSTUP(VPLUS); ZE end;
      ZAVZAV, KONEC : ;
      otherwise : CHYBA := TRUE;
   end;
end;
procedure T;
begin
   case SYMBOL of
      A, OTVZAV : begin F; ZT end;
      otherwise : CHYBA := TRUE;
   end;
end;
procedure ZT;
begin
   case SYMBOL of
      KRAT : begin SROVNEJ(KRAT);F;VYSTUP(VKRAT);ZT end;
      PLUS,ZAVZAV,KONEC : ;
      otherwise : CHYBA := TRUE;
   end;
end;
procedure F;
begin
   case SYMBOL of
      A : begin  SROVNEJ(A); VYSTUP(VA) end;
      OTVZAV : begin  SROVNEJ(OTVZAV);E; SROVNEJ(ZAVZAV) end;
      otherwise : CHYBA := TRUE;
   end;
end;
```

Vlastní formální překlad se provede takto:

```
begin
   CHYBA:=FALSE;
   LEXIKÁLNÍANALÝZA(SYMBOL);
   E;
end;
```

12.5 Příklady pro cvičení

Cvičení 12.15
Navrhněte konečný překladový automat, který na výstupu produkuje řetězec, který odpovídá stavům, kterými automat prošel.

Cvičení 12.16
Sestrojte konečné překladové automaty, které překládají výrazy z infixového do postfixového zápisu. Přitom vstupní výrazy mají tento charakter:

a) neobsahují závorky a operátory jsou $+$, $*$, $\uparrow$ (umocňování) s obvyklou prioritou,

b) obsahují závorky do hloubky 2 a operátory $+$, $*$ bez priorit,

c) obsahují závorky do hloubky 1 a operátory $+$, $*$ s obvyklou prioritou,

d) obsahují operátory $+$, $*$ a unární operátor f, který má prioritu vyšší, než operátor $*$.

Výrazy jsou vždy ukončeny symbolem $=$. Výsledné konečné překladové automaty interpretujte jako modely kalkulaček. Přitom stanovte, kolik je třeba registrů a jaké operace se budou provádět při jednotlivých přechodech.

Cvičení 12.17
Sestrojte konečné překladové automaty pro tyto překlady:

a) vstupní číslo je v osmičkové soustavě a jednotlivé číslice jsou kódovány jako tříciferná binární čísla, výstupní číslo je zapsáno obvyklým způsobem v osmičkové soustavě,

b) vstupní číslo je v šestnáctkové soustavě a číslice jsou kódovány jako čtyřciferná binární čísla, výstupní číslo je zapsáno obvyklým způsobem v šestnáctkové soustavě.

Každé vstupní číslo je ukončeno dvojtečkou.

Cvičení 12.18
Sestrojte překladové gramatiky pro definici překladů booleovských výrazů s operátory AND, OR a NOT:

a) z prefixového do infixového a postfixového zápisu,

b) z infixového do prefixového a postfixového zápisu,

c) z postfixového do prefixového a infixového zápisu.

Cvičení 12.19
Pro překladové gramatiky vzniklé řešením cvičení 12.18, které mají $LL(1)$ vstupní gramatiky, určete jakým způsobem je možno provést implementaci překladače.

Cvičení 12.20
Sestrojte překladovou gramatiku a konečný překladový automat pro tento překlad:
Vstupní abeceda je $\{a, b\}$, výstupní abeceda je $\{c, d\}$. Na výstupu se objeví symbol c v okamžiku, kdy počet symbolů a ve vstupním řetězci je dělitelný třemi. Symbol d se objeví na výstupu v okamžiku, kdy počet symbolů b je dělitelný pěti.

Cvičení 12.21
Navrhněte konečný překladový automat, který vstupní binární číslo vydělí pěti.

Cvičení 12.22
Navrhněte překladové gramatiky a zásobníkové překladové automaty, které popisují tyto překlady:

a) $\{(a^i b^i, x^{2i}) : i > 0\}$,

b) $\{(a^i b^j, x^{i+j}) : i, j > 0\}$,

c) $\{(a^i b^j, x^i y^{i+j}) : i, j > 0\}$,

d) $\{(a^i b^j, x^{j-i}) : j \geq i > 0\}$,

e) $\{(a^i b^j, x^{i-j}) : i \geq j > 0\}$.

13 Atributované překlady

13.1 Základní pojmy

Jsou dány dva jazyky L_1 a L_2 nad abecedami T a D. Dále jsou jednotlivým symbolům z T a D přiřazeny atributy. Atributovým překladem z jazyka L_1 do jazyka L_2 je relace $ZA \subset L_1 \times L_2$ a přitom symboly v řetězcích ve dvojici $(x, y) \in ZA$ mají definovány hodnoty atributů.

Jedním z formálních systémů pro popis atributovaného překladu jsou atributové překladové gramatiky.

Atributová překladová gramatika je trojice $APG = (G, A, F)$, kde G je překladová gramatika $G = (N, T \cup D, R, S_0)$, kde každé pravidlo v R je zapsáno ve tvaru:

(r) $X_0 \rightarrow X_1 X_2 \ldots X_{nr}$, kde $nr \geq 0$, $X_0 \in N, X_k \in (N \cup T \cup D)$ pro $1 \leq k \leq nr$,

A je konečná množina atributů:

Každému neterminálnímu symbolu $X \in N$ jsou přiřazeny dvě disjunktní množiny $I(X)$ a $S(X)$, kde $I(X)$ je množina dědičných a $S(X)$ množina syntetizovaných atributů symbolu X.

Každému vstupnímu symbolu X je přiřazena množina syntetizovaných atributů $S(X)$.

Každému výstupnímu symbolu $X \in D$ je přiřazena množina dědičných atributů $I(X)$.

F je konečná množina sémantických pravidel definovaná takto:

Pro každý symbol X_k $(1 \leq k \leq nr)$ v pravidle (r) a jeho dědičný atribut d je dáno sémantické pravidlo

$d = f_{rdk}(a_1, a_2, \ldots, a_n)$, kde $a_1, a_2, \ldots, a_n$ jsou atributy symbolů v témže pravidle r.

Pro každý syntetizovaný atribut s symbolu X_0 v pravidle r je definováno sémantické pravidlo

$s = f_{rso}(a_1, a_2, \ldots, a_n)$, kde $a_1, a_2, \ldots, a_n$ jsou atributy symbolů v témže pravidle r.

Aby bylo možno určit hodnoty všech atributů, musíme předpokládat, že:

a) hodnoty dědičných atributů počátečního symbolu jsou zadány,

b) hodnoty syntetizovaných atributů vstupních symbolů jsou zadány.

13.2 Regulární atributové překlady

Regulární atributový překlad je takový překlad, který se dá popsat regulární atributovou překladovou gramatikou, ve které neterminální symboly mají jen dědičné atributy.

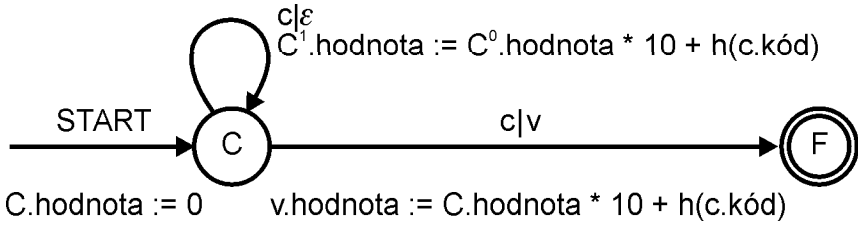
Příklad 13.1
Sestrojíme regulární atributovou překladovou gramatiku, která popisuje překlad desítkových celých čísel zapsaných obvyklým způsobem na jeden výstupní symbol, který jako atribut má hodnotu čteného čísla.
Regulární překladová gramatika má tvar:
$RPG = (\{C\}, \{c\} \cup \{v\}, R, C)$, kde R obsahuje pravidla:
$C \rightarrow c\,C \mid c\,v$
Přiřazení atributů jednotlivým symbolům uvádí následující tabulka:

symbol	dědičný atribut	syntetizovaný atribut
c		*kód*
C	*hodnota*	
v	*hodnota*	

K pravidlům překladové gramatiky jsou přidělena sémantická pravidla podle další tabulky:

syntaxe	sémantika
$C^0 \rightarrow c\,C^1$	$C^1.hodnota := C^0.hodnota * 10 + h(c.kód)$
$C \rightarrow c\,v$	$v.hodnota := C.hodnota * 10 + h(c.kód)$

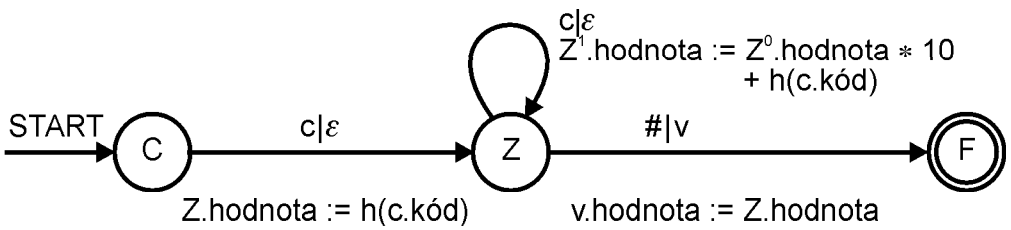
Funkce $h(x)$ má jako argument kód číslice a jako funkční hodnotu číselnou hodnotu číslice. Počáteční hodnota atributu počátečního symbolu $C.hodnota$ se rovná nule. Regulární atributové překlady je možno také popsat pomocí konečných překladových automatů, které jsou rozšířeny o možnost vyhodnocování sémantických pravidel při jednotlivých přechodech. Takový „atributovaný“ konečný překladový automat pro převod čísel bude mít atributovaný přechodový diagram podle obr. 13.1. Z tohoto diagramu je vidět, že sémantická pravidla



Obrázek 13.1: Přechodový diagram konečného překladového automatu z příkladu 13.1 s doplněnými sémantickými pravidly

pro výpočet atributů neterminálních a výstupních symbolů se připojí k hranám, které odpovídají příslušným pravidlům gramatiky. Uvedený konečný překladový automat je nedeterministický a není jej možno převést na deterministický, protože deterministický automat by musel podle čtené číslice poznat, že jde o poslední číslici. Deterministický automat můžeme sestrojit tehdy, když čtené číslo ukončíme speciálním znakem, při jehož čtení určíme hodnotu čísla.

Atributovaný přechodový diagram tohoto automatu je na obr. 13.2.
Tento automat odpovídá regulární atributové překladové gramatice s pravidly



Obrázek 13.2: Přechodový diagram deterministického konečného překladového automatu z příkladu 13.1

podle následující tabulky:

syntaxe	sémantika
$C \rightarrow cZ$	$Z.hodnota := h(c.kód)$
$Z^0 \rightarrow c\,Z^1$	$Z^1.hodnota := Z^0.hodnota * 10 + h(c.kód)$
$Z \rightarrow \#v$	$v.hodnota := Z.hodnota$

Příklad 13.2
Navrhneme regulární atributovou překladovou gramatiku, která popisuje zpracování seznamu konstant, při kterém se určí součet hodnot všech konstant v seznamu tvaru: $k_1; k_2; \ldots; k_n$, kde $n \geq 1$, k_i jsou konstanty pro $1 \leq i \leq n$. Regulární překladová gramatika má tvar:

$RPG = (\{S, Z\}, \{k, ;\} \cup \{v\}, R, S)$, kde R obsahuje pravidla uvedená v následující tabulce:

syntaxe	sémantika
$S \rightarrow kZ$	$Z.hodnota := k.shodnota + S.dhodnota$
$Z \rightarrow ;S$	$S.dhodnota := Z.dhodnota$
$S \rightarrow kv$	$v.dhodnota := S.dhodnota + k.shodnota$

Symboly S, Z a v mají dědičný atribut *dhodnota*, symbol k má syntetizovaný atribut *shodnota*. Přiřazení sémantických a syntaktických pravidel je zřejmé z tabulky. Platí, že na začátku $S.hodnota = 0$. Sestrojte pro tuto regulární atributovou překladovou gramatiku ekvivalentní deterministický konečný překladový automat, doplněný o sémantická pravidla. Uvažte, že je nutno ukončovat vstupující seznamy speciálním znakem.

Příklad 13.3
Navrhneme regulární atributovou překladovou gramatiku, která představuje model kalkulačky s klávesami:

1,2,3,4,5,6,7,8,9,0,+,*,=

To znamená, že vstupem do kalkulačky budou výrazy s operátory + a *. Symbol

= bude představovat konec výrazu. Budeme předpokládat, že operátory + a * mají stejnou prioritu.

Z příkladu 12.2 již víme, že kalkulačka tohoto typu bude mít dva registry. Těm budou v našem případě odpovídat atributy, ve kterých budou uchovávány mezivýsledky. Označme je x, y.

Regulární překladová gramatika bude definována takto:

$RPG = (\{S, P, A, M, N\}, \{c, +, *, =\}, \{D\}, R, S)$

Syntaktická a sémantická pravidla atributové gramatiky jsou uvedena v následující tabulce. V tabulce nejsou podrobně rozepisována pravidla pro jednotlivé číslice, ale vždy je uvedeno jen jedno pravidlo pro všechny číslice. Zápis $c.h$ představuje hodnotu vstupní číslice.

syntaxe	sémantika	
$S \to c\,P$	$P.x = c.h$	$P.y = 0$
$P^0 \to c\,P^1$	$P^1.x = P^0.x * 10 + c.h$	$P^1.y = P^0.y$
$P \to +A$	$A.x = P.x + P.y$	
$P \to *M$	$M.x = P.x + P.y$	
$P \to = D$	$D.x = P.x + P.y$	
$A \to c\,P$	$P.x = c.h$	$P.y = A.x$
$M \to c\,N$	$N.x = c.h$	$N.y = M.x$
$N^0 \to c\,N^1$	$N^1.x = N^0.x * 10 + c.h$	$N^1.y = N^0.y$
$N \to +A$	$A.x = N.x * N.y$	
$N \to *M$	$M.x = N.x * N.y$	
$N \to = D$	$D.x = N.x * N.y$	

Na obr. 13.3 je uveden přechodový diagram konečného překladového automatu s doplněnými sémantickými pravidly, který odpovídá atributové překladové gramatice a představuje model kalkulačky. Z tohoto automatu je vidět, jaký význam mají jednotlivé stavy automatu a tedy i neterminální symboly gramatiky.

S - počáteční stav, začátek zadávání vstupního řetězce,

P - stav po přečtení alespoň jedné číslice prvního čísla nebo čísla za operátorem +,

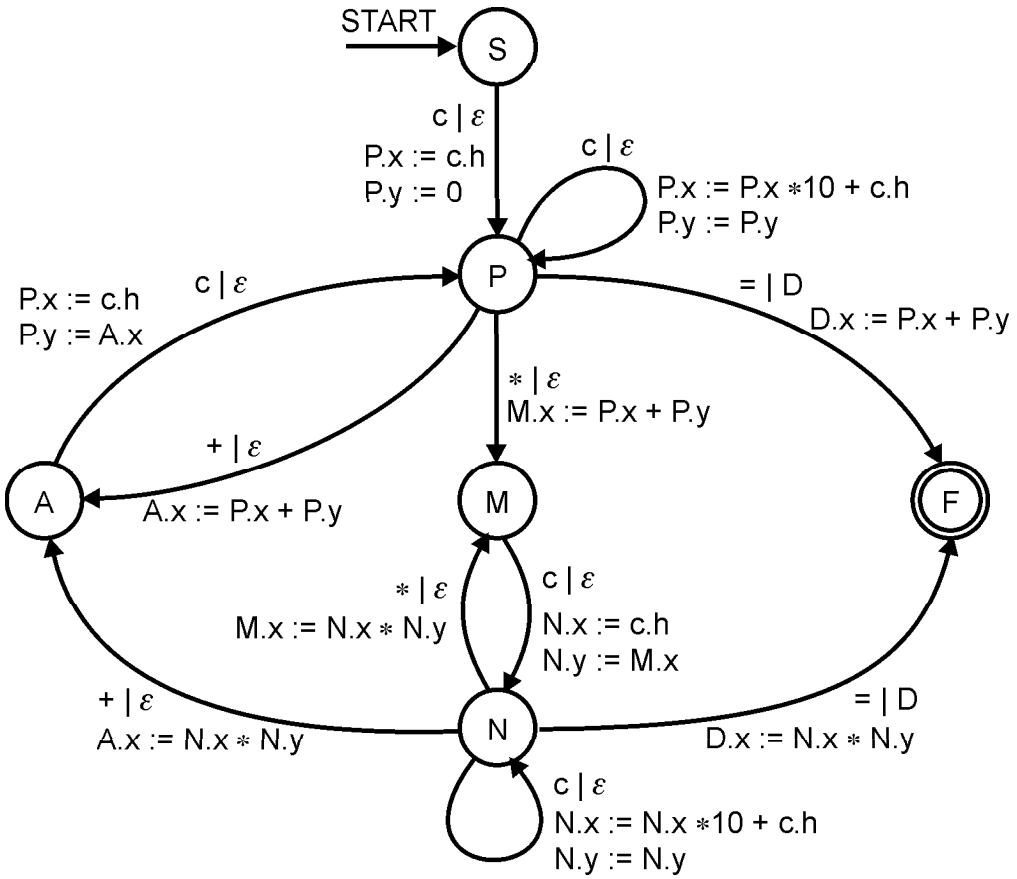
A - stav po přečtení operátoru +,

M - stav po přečtení operátoru *,

N - stav po přečtení alespoň jedné číslice čísla za operátorem *,

F - koncový stav.

Atribut x výstupního symbolu D je hodnota, která se zobrazí na displeji kalkulačky. Pokud bychom chtěli na displeji zobrazovat i mezivýsledky, bylo by nutné zobrazovat hodnotu atributu x u všech stavů.



Obrázek 13.3: Přechodový diagram atributovaného konečného překladového automatu z příkladu 13.3

13.3 Bezkontextové atributované překlady

Bezkontextový atributovaný překlad je takový překlad, který se dá popsat bezkontextovou atributovou překladovou gramatikou. Jako první překlad ale uvedeme atributovou gramatiku, která má základní gramatiku regulární. Protože tato gramatika obsahuje syntetizovaný atribut, není možná implementace pomocí atributovaného konečného automatu.

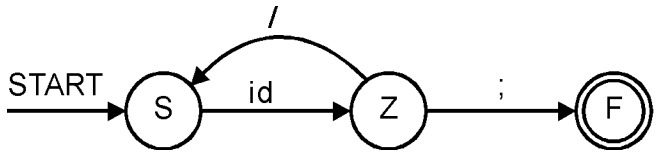
Příklad 13.4
Je dán jazyk seznamů, ve kterých se vyskytují identifikátory oddělené lomítky. Seznam je ukončen středníkem. Uvedeme regulární atributovou překladovou gramatiku, která popisuje způsob výpočtu počtu prvků seznamu. Výsledek, t.j. počet prvků seznamu, bude syntetizovaným atributem počátečního symbolu. Gramatika pro popis syntaxe vstupních řetězců je regulární a má tvar:

$G = (\{S, Z\}, \{id, ;, /\}, P, S)$,

neterminální symboly S a Z mají syntetizovaný atribut *počet*. Syntaktická a sémantická pravidla jsou uvedena v následující tabulce.

syntaxe	sémantika
$S \rightarrow id\ Z$	$S.počet := Z.počet+1$
$Z \rightarrow ;$	$Z.počet := 0$
$Z \rightarrow /\ S$	$Z.počet := S.počet$

Výpočet počtu prvků seznamu tímto způsobem nelze zajistit tak, že ke konečnému automatu přidáme možnost vyhodnocení sémantických pravidel. V tomto případě je nutno použít při vyhodnocování sémantický zásobník. Konečný automat, který odpovídá dané regulární gramatice má přechodový diagram podle obr. 13.4.



Obrázek 13.4: Přechodový diagram konečného automatu z příkladu 13.3

Skutečností je, že všechna sémantická pravidla můžeme vyhodnotit až v koncovém stavu F. Je to dáno tím, že první pravidlo, které můžeme vyhodnotit je pravidlo *Z.počet := 0*, a které se může vyhodnotit teprve při přechodu do stavu F. Teprve potom je možno vyhodnotit ostatní sémantická pravidla. Abychom věděli, která to jsou, musíme je uschovat a protože je budeme vyhodnocovat v obráceném pořadí než v jakém byla použita syntaktická pravidla při syntaktické analýze, je vhodnou datovou strukturou pro uchování těchto sémantických pravidel zásobník.
Uvažme způsob zpracování seznamu $id/id;$

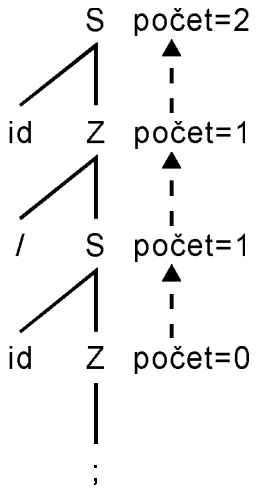
Tento řetězec má derivaci:

$S \Rightarrow id\ Z\ \Rightarrow id/S\ \Rightarrow id/id\ Z\ \Rightarrow id/id;$

Neterminální symboly v této derivaci představují stavy, kterými prochází konečný automat předtím, než se dostane do stavu F. V tomto stavu se vyhodnotí sémantické pravidlo $Z.počet := 0$. Abychom vyhodnotili sémantiku celého řetězce, musíme vyhodnotit tato sémantická pravidla v tomto pořadí:

$S.počet := Z.počet+1$
$Z.počet := S.počet$
$S.počet := Z.počet+1.$

Derivační strom pro řetězec $id/id;$ s vypočítanými hodnotami atributů a naznačenými závislostmi mezi atributy je na obr. 13.5.



Obrázek 13.5: Atributovaný derivační strom

Příklad 13.5
Je dán jazyk závorkové struktury výrazu. Tento jazyk můžeme generovat například gramatikami:

$G_1 = (\{S\}, \{(,)\}, \{S \to (S)S|\varepsilon\}, S)$
$G_2 = (\{S\}, \{(,)\}, \{S \to S(S)|\varepsilon\}, S)$

Tyto gramatiky rozšíříme na překladové doplněním počátečního pravidla

$S' \to S\ v$

s výstupním symbolem v. K takto získaným překladovým gramatikám doplníme sémantiku tak, abychom dokázali určit:

a) celkový počet závorek v řetězci,
b) maximální hloubku závorkové struktury,
c) kolikrát bylo použito pravidlo $S \to \varepsilon$.

a) Zavedeme atribut počet, který bude přidělen symbolu S jako syntetizovaný a symbolu v jako dědičný. Syntaktická a sémantická pravidla jsou v následující tabulce.

syntaxe	sémantika
$S' \to S\ v$	$v.počet := S.počet$
$S^0 \to (S^1)S^2$	$S^0.počet := S^1.počet + S^2.počet + 2$
$S \to \varepsilon$	$S.počet := 0$

b) Zavedeme atribut hloubka a opět jej přidělíme symbolu S jako syntetizovaný a symbolu v jako dědičný.

syntaxe	sémantika
$S' \to S\ v$	$v.hloubka := S.hloubka$
$S^0 \to (S^1)S^2$	$S^0.hloubka := MAX(S^1.hloubka + 1, S^2.hloubka)$
$S \to \varepsilon$	$S.hloubka := 0$

c) Zavedeme atribut kolik, který přidělíme symbolu S jako syntetizovaný a symbolu v jako dědičný.

syntaxe	sémantika
$S' \to S\ v$	$v.kolik := S.kolik$
$S^0 \to (S^1)S^2$	$S^0.kolik := S^1.kolik + S^2.kolik$
$S \to \varepsilon$	$S.kolik := 1$

Ve všech případech jsme vyšli z gramatiky G_1.
Sestrojte atributové gramatiky tak, že za základ použijete gramatiku G_2.

Příklad 13.6
Nyní uvedeme tři atributové gramatiky, které popisují překlady výrazů, pro něž nelze sestrojit deterministické zásobníkové překladové automaty. Překladové gramatiky pro tyto překlady jsou uvedeny v příkladech 12.7, 12.9 a 12.10.
Jedná se o překlady výrazů:

a) z infixového do prefixového zápisu,
b) z postfixového do prefixového zápisu,
c) z postfixového do infixového zápisu.

V sémantických pravidlech jsou použity funkce, které pracují se seznamy:

$LIST(x)$ vytvoří jednoprvkový seznam, který obsahuje x,
$CONS3(x, y, z)$ vytvoří seznam tvořený spojením seznamů x, y a z.
$CONS5(u, v, x, y, z)$ vytvoří seznam tvořený spojením seznamů u, v, x, y, z.

Ve všech případech je výstupní řetězec ve formě seznamu hodnotou atributu p u počátečního symbolu.

a) Překlad z infixového do prefixového zápisu výrazu (formální překlad viz příklad 12.8).

syntaxe	sémantika
$E^0 \to E^1 + T$	$E^0.p := CONS3(LIST(\oplus), E^1.p, T.p)$
$E \to T$	$E.p := T.p$
$T^0 \to T^1 * F$	$T^0.p := CONS3(LIST(\circledast), T^1.p, F.p)$
$T \to F$	$T.p := F.p$
$F \to a$	$F.p := LIST(@)$
$F \to (E)$	$F.p := E.p$

b) Překlad z postfixového do prefixového zápisu výrazu (formální překlad viz příklad 12.9).

syntaxe	sémantika
$E^0 \to E^1E^2+$	$E^0.p := CONS3(LIST(\oplus), E^1.p, E^2.p)$
$E^0 \to E^1E^2*$	$E^0.p := CONS3(LIST(\circledast), E^1.p, E^2.p)$
$E \to a$	$E.p := LIST(@)$

c) Překlad z postfixového do infixového zápisu výrazu (formální překlad viz příklad 12.6).

syntaxe	sémantika
$E^0 \to E^1E^2+$	$E^0.p := CONS5(LIST(\textcircled{(}), E^1.p, LIST(\oplus), E^2.p, LIST(\textcircled{)}))$
$E^0 \to E^1E^2*$	$E^0.p := CONS3(E^1.p, LIST(\circledast), E^2.p)$
$E \to a$	$E.p := LIST(@)$

Příklad 13.7
Je dán jazyk seznamů konstant. Sestrojíme atributovou gramatiku, ve které hodnotou syntetizovaného atributu počátečního symbolu bude hodnota nejmenší konstanty. Přitom použijeme dva syntetizované atributy:

min – hodnota minimální konstanty,
hod – hodnota vstupní konstanty.

Atribut *min* přidělíme symbolu L, atribut *hod* přidělíme symbolu k. Uvedeme dvě varianty založené na gramatikách s levou a pravou rekurzí.

a) Gramatika s levou rekurzí.

syntaxe	sémantika
$L^0 \to L^1/k$	$L^0.min := MIN(L^1.min, k.hod)$
$L \to k$	$L.min := k.hod$

b) Gramatika s pravou rekurzí.

syntaxe	sémantika
$L^0 \to k/L^1$	$L^0.min := MIN(k.hod, L^1.min)$
$L \to k$	$L.min := k.hod$

Příklad 13.8
Pro stejný vstupní jazyk jako v předchozím příkladu sestrojíme atributovou gramatiku, která provede výpočet součtu hodnot všech konstant. Uvedeme jen příklad gramatiky s pravou rekurzí. Použijeme syntetizovaný atribut *suma*, který přidělíme symbolu L a který bude obsahovat hodnotu součtu.

syntaxe	sémantika
$L^0 \rightarrow k/L^1$	$L^0.suma := k.hod + L^1.suma$
$L \rightarrow k$	$L.suma := k.hod$

Příklad 13.9
Pro stejný vstupní jazyk jako v příkladu 13.7 navrhneme atributovou gramatiku, která bude řešit tuto úlohu: Je zadána konstanta a je třeba určit, zda se daná konstanta v seznamu vyskytuje a na kterém je místě. Pořadí konstant ve vstupním seznamu je označeno přirozenými čísly. Zadaná konstanta je počáteční hodnotou dědičného atributu počátečního symbolu.
Použijeme tyto atributy:

pořadí – pořadí zpracovávané konstanty,
hledaná – hodnota zadané konstanty,
nalezená – pořadí nalezené konstanty,
hod – hodnota vstupní konstanty.

Jednotlivým symbolům budou přiděleny atributy podle následující tabulky.

symbol	dědičné atributy	syntetizované atributy
L	*hledaná*	*pořadí, nalezená*
k		*hod*

Nejdříve uvedeme pravidla gramatiky s levou rekurzí.

syntaxe	sémantika
$L^0 \rightarrow L^1/k$	$L^1.hledaná := L^0.hledaná$ $L^0.pořadí := L^1.pořadí + 1$ $L^0.nalezená :=$ **if** $L^1.nalezená \neq 0$ **then** $L^1.$ *nalezená* **else if** $L^0.hledaná = k.hod$ **then** $L^1.pořadí + 1$ **else** 0
$L \rightarrow k$	$L.pořadí := 1$ $L.nalezená :=$ **if** $L.hledaná = k.hod$ **then** 1 **else** 0

V sémantických pravidlech jsme pro zjednodušení zápisu použili podmíněný výraz tvaru **if** *podmínka* **then** *výraz1* **else** *výraz2*. Uvedený zápis má tento význam:
Jestliže *podmínka* je splněna, pak je hodnotou výrazu hodnota *výrazu1*, v opačném případě je hodnotou hodnota *výrazu2*.

Dále uvedeme pravidla gramatiky s pravou rekurzí. V tomto případě bude atribut *pořadí* dědičný a jeho počáteční hodnota bude 1.

syntaxe	sémantika
$L^0 \to k\ /L^1$	$L^1.pořadí := L^0.pořadí + 1$ $L^1.hledaná := L^0.hledaná$ $L^0.nalezená :=$ **if** $L^1.nalezená \neq 0$ **then** $L^1.nalezená$ **else if** $L^0.hledaná = k.hod$ **then** $L^0.pořadí$ **else** 0
$L \to k$	$L.nalezená :=$ **if** $L.hledaná = k.hod$ **then** $L.pořadí$ **else** 0

Příklad 13.10
Pro stejný vstupní jazyk jako v příkladu 13.7 sestrojíme atributovou gramatiku, která bude řešit tuto úlohu:
Je zadána dvojice konstant $a,\ b$, kde $a < b$. Je třeba určit, na kterých místech v seznamu jsou konstanty, jejichž hodnoty leží v intervalu $\langle a, b\rangle$. Přitom použijeme tyto atributy:

nalezené – vektor, ve kterém jsou uložena pořadí nalezených konstant,
počet – počet nalezených konstant,
pořadí – pořadí zpracovávané konstanty,
hod – hodnota vstupní konstanty,
a,b – hodnoty zadaných konstant.

Uvedeme gramatiku s levou rekurzí.
Atributy přidělíme jednotlivým symbolům podle následující tabulky.

symbol	dědičné atributy	syntetizované atributy
L	a, b	*pořadí,počet,nalezené*
k		*hod*

Syntaktická a sémantická pravidla jsou uvedena v další tabulce.

syntaxe	sémantika
$L^0 \to L^1/k$	$L^1.a := L^0.a$ $L^1.b := L^0.b$ $L^0.pořadí := L^1.pořadí + 1$ **if** $L^0.a \leq k.hod$ **and** $k.hod \leq L^0.b$ **then** **begin** $L^0.počet := L^1.počet + 1;$ $L^0.nalezené[L^1.počet + 1] := L^1.pořadí + 1$ **end** **else** **begin** $L^0.počet := L^1.počet;$ $L^0.nalezené := L^1.nalezené$ **end**
$L \to k$	**if** $L.a \leq k.hod$ **and** $k.hod \leq L.b$ **then** **begin** $L.nalezené[1] := 1; L.počet := 1$ **end** **else** $L.počet := 0$

V další skupině příkladů se budeme zabývat problémem zjednodušování aritmetických a logických výrazů.

Příklad 13.11
Sestrojíme atributovou gramatiku, která popisuje překlad aritmetického výrazu tak, že se během překladu provedou všechny operace s konstantami.
Přitom budeme používat tyto atributy:

překlad – přeložený tvar podvýrazu generovaného neterminálním symbolem,
konstanta – údaj o tom, zda přeložený tvar je nebo není konstanta,
hodnota – v případě, že přeložený tvar je konstanta, pak je to hodnota této konstanty.

Operandy ve výrazu budou mít jako atribut svůj textový tvar a konstanta bude mít jako atribut hodnotu.
Atributy budou přiděleny podle této tabulky.

symboly	syntentizované atributy
E, T	*překlad, konstanta, hodnota*
id	*jméno*
k	*jméno, hod*

Pravidla atributové gramatiky jsou uvedena v další tabulce (operátor | znamená zřetězení).

syntaxe	sémantika
$E^0 \to E^1 + T$	$E^0.konstanta := E^1.konstanta$ **and** $T.konstanta$ $E^0.hodnota :=$**if** $E^1.konstanta$ **and** $T.konstanta$ **then** $E^1.hodnota + T.hodnota$ **else** nedefinována $E^0.překlad :=$**if** $E^1.konstanta$ **and** $T.konstanta$ **then** TEXT$(E^1.hodnota + T.hodnota)$ **else** $E^1.překlad \mid' +' \mid T.překlad$
$E \to T$	$E.konstanta := T.konstanta$ $E.hodnota := T.hodnota$ $E.překlad := T.překlad$
$T \to id$	$T.konstanta :=$false $T.hodnota :=$nedefinována $T.překlad := id.jméno$
$T \to k$	$T.konstanta :=$true $T.hodnota := k.hod$ $T.překlad := k.jméno$

Funkce TEXT použitá v sémantickém pravidle převede hodnotu parametru do textového tvaru.

Příklad 13.12
Sestrojíme atributovou gramatiku, která bude provádět zjednodušení výrazu s použitím rovností:

$a + 0 = a$
$0 + a = a$

Atributy, které použijeme, jsou stejné jako v příkladu 13.11.
Syntaktická a sémantická pravidla jsou uvedena v následující tabulce.

syntaxe	sémantika
$E^0 \to E^1 + T$	$E^0.$*překlad* := **if** $E^1.$*konstanta* **and** $E^1.$*hodnota* $= 0$ **then** $T.$*překlad* **else if** $T.$*konstanta* **and** $T.$*hodnota* $= 0$ **then** $E^1.$*překlad* **else** $E^1.$*překlad* | '+' | $T.$*překlad* $E^0.$*konstanta* := **if** $E^1.$*konstanta* **and** $E^1.$*hodnota* $= 0$ **then** $T.$*konstanta* **else if** $T.$*konstanta* **and** $T.$*hodnota* $= 0$ **then** $E^1.$*konstanta* **else** false $E^0.$*hodnota* := **if** $E^1.$*konstanta* **and** $E^1.$*hodnota* $= 0$ **then** $T.$*hodnota* **else if** $T.$*konstanta* **and** $T.$*hodnota* $= 0$ **then** $E^1.$*hodnota*. **else** nedefinována
$E \to T$	$E.$*konstanta* := $T.$*konstanta* $E.$*hodnota* := $T.$*hodnota* $E.$*překlad* := $T.$*překlad*
$T \to id$	$T.$*konstanta* := false $T.$*hodnota* := nedefinována $T.$*překlad* := $id.$*jméno*
$T \to k$	$T.$*konstanta* := true $T.$*hodnota* := $k.$*hod* $T.$*překlad* := $k.$*jméno*

Příklad 13.13
Sestrojíme atributovou gramatiku, která bude provádět náhradu operace umocňování, operací násobení podle vztahů:

$x \uparrow 2 = x * x$
$x \uparrow 3 = x * x * x$
$\vdots$

pro libovolný exponent menší než zadané číslo mez. Použijeme atributy stejné jako v předchozím příkladu a přidáme k nim dědičný atribut mez, který bude přidělen symbolu E a jehož počáteční hodnota bude zadána. Syntaktická a sémantická pravidla jsou uvedena v následující tabulce. Funkce OPAK(n,*řetězec*) vytvoří posloupnost n řetězců.

syntaxe	sémantika
$E^0 \to E^1 \uparrow T$	$E^1.mez := E^0.mez$ $E^0.$*překlad*:=**if** $T.konstanta$ **and** $0 < T.hodnota$ **and** $T.hodnota < E^0.mez$ **then** $E^1.$*překlad*$\|OPAK(T.hodnota - 1\|' * ' \|E^1.$*překlad)* **else** $E^1.$*překlad*$\|' \uparrow ' \|T.$ *překlad* $E^0.konstanta$:=**if** $T.konstanta$ **and** $T.hodnota = 1$ **then** $E^1.konstanta$ **else** $t.konstanta$ **and** $T.hodnota = 0$ $E^0.hodnota$:=**if** $T.konstanta$ **and** $T.hodnota = 1$ **then** $E^1.hodnota$ **else if** $T.konstanta$ **and** $T.hodnota = 0$ **then** 1 **else** nedefinována
$E \to T$	$E.konstanta := T.konstanta$ $E.hodnota := T.hodnota$ $E.$*překlad*:= $T.$*překlad*
$T \to id$	$T.konstanta$:=false $T.hodnota$:=nedefinována *T.překlad:=id.jméno*
$T. \to k$	$T.konstanta$:=true $T.hodnota := k.hod$ *T.překlad:=k.jméno*

Příklad 13.14
Je dán jazyk logických výrazů s operátory **or, and** a **not**. Sestrojíme atributovou gramatiku, která popisuje zjednodušování logických výrazů. Při zjednodušování se budou provádět operace s konstantami a úpravy podle těchto rovností:

$true$ **or** $x = true \quad = x$ **or** $true$
$true$ **and** $x = x \quad = x$ **and** $true$
$false$ **or** $x = x \quad = x$ **or** $false$
$false$ **and** $x = false = x$ **and** $false$
not $true \quad = false$
not $false \quad = true$

V atributové gramatice budeme používat tyto atributy:

výraz – textový tvar upraveného výrazu,
druh – informace o výrazu s hodnotami (false,true,logvýraz),
jméno – textový tvar identifikátoru nebo konstanty.

Atributy budou jednotlivým symbolům přiděleny podle následující tabulky.

symboly	syntetizované atributy
E,T,F	*výraz, druh*
id	*jméno*

Následující tabulka obsahuje syntaktická a k nim přiřazená sémantická pravidla.

syntaxe	sémantika
$E^0 \rightarrow E^1$ **or** T	$E^0.druh :=$ **if** $E^1.druh =$ **true or** $T.druh =$ **true** **then true else** **if** $E^1.druh =$ **false then** $T.druh$ **else** **if** $T.druh =$ **false then** $E^1.druh$ **else** logvýraz $E^0.výraz :=$ **if** $E^1.druh =$ **true or** $T.druh =$ **true** **then** TEXT(true) **else** **if** $E^1.druh =$ **false then** $T.výraz$ **else** **if** $T.druh =$ **false then** $E^1.výraz$ **else** $E^1.výraz$ |'or'| $T.výraz$
$E \rightarrow T$	$E.druh := T.druh$ $E.výraz := T.výraz$
$T^0 \rightarrow T^1$ **and** F	$T^0.druh :=$ **if** $T^1.druh =$ **false** **or** $F.druh =$ **false then false else** **if** $T^1.druh =$ **true then** $F.druh$ **else** **if** $F.druh =$ **true** **then** $T^1.druh$ **else** logvýraz $T^0.výraz :=$ **if** $T^1.druh =$ **false** **or** $F.druh =$ **false then** TEXT(false) **else if** $T^1.druh =$ **true** **then** $F.výraz$ **else** **if** $F.druh =$ **true then** $T^1.výraz$ **else** $T^1.výraz$ |'and'| $F.výraz$
$T \rightarrow F$	$T.druh := F.druh$ $F.výraz := F.výraz$
$F^0 \rightarrow$ **not** F^1	$F^0.druh :=$ **if** $F^1.druh =$ logvýraz **then** logvýraz **else not** $F^1.druh$ $F^1.výraz :=$ **if** $F^1.druh =$ logvýraz **then** 'not'| $F^1.výraz$ **else** **if** $F^1.druh =$ **true** **then** TEXT(false) **else** TEXT(true)

$F \to (E)$	$F.druh := E.druh$ $F.výraz := E.výraz$
$F \to id$	$F.druh :=$ logvýraz $F.výraz := id.jméno$
$F \to$ **false**	$F.druh :=$ **false** $F.výraz :=$ TEXT(false)
$F \to$ **true**	$F.druh :=$ **true** $F.výraz :=$ TEXT(true)

Příklad 13.15
Je dán jazyk polynomů s reálnými koeficienty. Sestrojíme atributovou překladovou gramatiku, která popisuje překlad polynomů na jejich derivaci podle proměnné x. Polynom $a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$ je zapsán jako vstupní řetězec $a * x \uparrow n + a * x \uparrow n + \ldots + a * x \uparrow n + a * x + a$, kde hodnoty atributů symbolů a zleva doprava jsou:

$a_n, a_{n-1}, \ldots, a_1, a_0,$

a hodnoty atributů symbolů x zleva doprava jsou:

$n, n-1, \ldots, 2.$

Při derivaci jednoho prvku polynomu použijeme tyto vztahy:

$$\frac{d\, a*x^n}{dx} = a * n * x^{n-1} \qquad \text{pro } n > 1$$

$$\frac{d\, a*x}{dx} = a$$

$$\frac{da}{dx} = 0$$

Překladová gramatika $G = (\{P\}, \{a, x, n, +, *, \uparrow\}, \{ⓐ, ⓧ, ⓝ, ⊕, ⊛, ⇑\}, R, P)$. Jediný atribut, který použijeme, bude *hod*, který bude syntetizovaným atributem vstupních symbolů a, n a dědičným atributem výstupních symbolů ⓐ, ⓝ. Pravidla překladové gramatiky a k nim přidělená sémantická pravidla jsou uvedena v následující tabulce.

syntaxe	sémantika
$P \to a ⓐ * ⊛ x ⓧ \uparrow ⇑ n ⓝ + ⊕ P$	$ⓐ.hod := a.hod * n.hod$ $ⓝ.hod := n.hod - 1$
$P \to a^1 ⓐ * x + a^2$	$ⓐ.hod := a^1.hod$

Příklad 13.16
Je dán jazyk aritmetických výrazů s operátory + a $*$. Navrhneme atributovou gramatiku, která vytvoří derivaci vstupního výrazu podle zadané proměnné x. Při derivaci použijeme tyto vztahy:

$$\frac{d\,(a+b)}{dx} = \frac{da}{dx} + \frac{db}{dx}$$

$$\frac{d\,(a*b)}{dx} = a * \frac{db}{dx} + b * \frac{da}{dx}$$

$$\frac{dx}{dx} = 1$$

$$\frac{dk}{dx} = 0, \qquad \text{když } k \text{ je konstanta.}$$

V atributové gramatice použijeme atributy:

výraz – tvar vstupního výrazu,
derivace – derivace vstupního výrazu,
x – jméno proměnné, podle které se derivuje,
jméno – textový tvar identifikátoru nebo konstanty.

Jednotlivé atributy přidělíme symbolům podle následující tabulky.

symboly	dědičné atributy	syntetizované atributy
E, T, F	x	*výraz, derivace*
id, k		*jméno*

Syntaktická a sémantická pravidla jsou uvedena v další tabulce.

syntaxe	sémantika
$E^0 \to E^1 + T$	$E^1.x := E^0.x$ $T.x := E^0.x$ $E^0.$*výraz*$:= E^1.$*výraz* $\mid' +' \mid T.$*výraz* $E^0.derivace := E^1.derivace \mid' +' \mid T.derivace$
$E \to T$	$T.x := E.x$ $E.$*výraz*$:= T.vraz$ $E.derivace := T.derivace$
$T^0 \to T^1 * F$	$T^1.x := T^0.x$ $F.x := T^0.x$ $T^0.$*výraz*$:= T^1.$*výraz*$\mid' *' \mid F.$*výraz* $T^0.derivace := T^1.$*výraz* $\mid' *' \mid F.derivace \mid' +' \mid$ $F.$*výraz*$\mid' *' \mid T^1.derivace$
$T \to F$	$F.x := T.x$ $T.$*výraz*$:= F.$*výraz* $T.derivace := F.derivace$
$F \to (E)$	$E.x := F.x$ $F.$*výraz*$:= E.$*výraz* $F.derivace := E.derivace$
$F \to id$	$F.$*výraz*$:= id.$*jméno* $F.derivace :=$**if** $F.x = id.$*jméno* **then** $TEXT(1)$ **else** $TEXT(0)$
$F \to k$	$F.$*výraz*$:= k.$*jméno* $F.derivace := TEXT(0)$

Příklad 13.17
Je dán jazyk aritmetických výrazů s operátorem * a trigonometrickými funkcemi sin, cos, tg a cotg. Navrhneme atributovou překladovou gramatiku, která vytvoří derivaci vstupního výrazu podle zadané proměnné x. Při derivaci použijeme vztah pro derivaci součinu (viz příklad 13.16) a tyto další vztahy:

$\frac{d\ f(g)}{dx} = \frac{df}{dx} * \frac{dg}{dx}$ (derivace složené funkce)

$\frac{d\ sin(x)}{dx} = cos(x)$

$\frac{d\ cos(x)}{dx} = -sin(x)$

$\frac{d\ tg(x)}{dx} = \frac{1}{cos(x)}$

$\frac{d\ cotg(x)}{dx} = -\frac{1}{sin(x)}$

Syntaktická a sémantická pravidla atributové gramatiky jsou uvedena v následující tabulce.

syntaxe	sémantika
$T^0 \rightarrow T^1 * F$	$T^1.x := T^0.x$ $F.x := T^0.x$ $T^0.výraz := T^1.výraz \mid' *' \mid F.výraz$ $T^0.derivace := T^1.výraz \mid' *' \mid F.derivace \mid' +' \mid$ $F.výraz \mid' *' \mid T^1.derivace$
$T \rightarrow F$	$F.x := T.x$ $T.výraz := F.výraz$ $T.derivace := F.derivace$
$F \rightarrow (T)$	$T.x := F.x$ $F.výraz := T.výraz$ $F.derivace := T.derivace$
$F \rightarrow id$	$F.výraz := id.jméno$ $F.derivace :=$**if** $F.x = id.jméno$ **then** $TEXT(1)$ **else** $TEXT(0)$
$F \rightarrow k$	$F.výraz := k.jméno$ $F.derivace := TEXT(0)$
$F \rightarrow sin(T)$	$T.x := F.x$ $F.výraz :=' sin(' \mid T.výraz \mid')'$ $F.derivace :=' cos(' \mid T.výraz \mid') *' \mid T.derivace$
$F \rightarrow cos(T)$	$T.x := F.x$ $F.výraz :=' cos(' \mid T.výraz \mid')'$ $F.derivace :=' -sin(' \mid T.výraz \mid') *' \mid T.derivace$

syntaxe	sémantika
$F \to tg(T)$	$T.x := F.x$ $F.výraz{:=}'tg('\|T.výraz\|')'$ $F.derivace := TEXT(1)\|'/(cos('\|T.výraz\|')*'$ $\|T.derivace$
$F \to cotg(T)$	$T.x := F.x$ $F.výraz{:=}'cotg('\|T.výraz\|')'$ $F.derivace := TEXT(-1)\|'/(sin('\|T.výraz\|') *' \|T.derivace$

Příklad 13.18
Je dán jazyk aritmetických výrazů s operátory $+, *, /, \uparrow$ a s možností zápisu indexů. Navrhneme atributovou gramatiku, která bude konstruovat v matematice obvyklý grafický zápis vstupního výrazu.
Pro jednoduchost popisu zavedeme jediný atribut g, který bude mít jako svoji hodnotu tyto informace:
- grafické znázornění výrazu,
- šířku a výšku obdélníka, který opisuje grafický zápis výrazu.

Syntaktická a schematicky znázorněná sémantická pravidla obsahuje následující tabulka.

syntaxe	sémantika
$E^0 \to E^1 + T$	$E^0.g :=$ $\boxed{\boxed{E^1.g} + \boxed{T.g}}$
$E \to T$	$E.g := T.g$
$T^0 \to T^1 * F$	$T^0.g :=$ $\boxed{\boxed{T^1.g} * \boxed{F.g}}$
$T^0 \to T^1 / F$	$T^0.g :=$ $\boxed{\dfrac{\boxed{T^1.g}}{\boxed{F.g}}}$
$T \to F$	$T.g := F.g$
$F^0 \to F^1 \uparrow G$	$F^0.g :=$ $\boxed{\boxed{F^1.g}^{\boxed{G.g}}}$
$F^0 \to F^1[E]$	$F^0.g :=$ $\boxed{\boxed{F^1.g}_{\boxed{E.g}}}$
$F \to G$	$F.g := G.g$

syntaxe	sémantika
$G \to (E)$	$G.g := \boxed{(\ \boxed{E.g}\ )}$
$G \to a$	$G.g := \boxed{a.text}$

13.4 Jednoprůchodový algoritmus výpočtu atributů při LL analýze

V tomto odstavci uvedeme příklad na metodu výpočtu atributů, která je založena na LL analýze. Současně uvedeme implementaci této metody.

Příklad 13.19
Ukažme implementaci výpočtu atributů při LL analýze pro řešení úlohy z příkladu 13.9. V tomto příkladu jsou uvedeny dvě atributové gramatiky. Žádná z nich však nemá základní gramatiku $LL(1)$. První z uvedených atributových gramatik má základní gramatiku, která obsahuje levou rekurzi a proto nemůže být $LL(k)$ pro žádné k. Druhá atributová gramatika má základní gramatiku $LL(2)$. Vezměme tedy za základ tuto $LL(2)$ gramatiku a transformujme ji na $LL(1)$ gramatiku s tím, že je třeba také transformovat sémantická pravidla.

Na základní gramatiku použijme levou faktorizaci a dosazení a dostaneme $LL(1)$ gramatiku $G = (\{L, R\}, \{k, /\}, P, L)$, kde P obsahuje pravidla:

$L \to kR$
$R \to /kR \mid \varepsilon$

Sémantická pravidla a jejich přidělení syntaktickým obsahuje následující tabulka:

syntaxe	sémantika
(1) $L \to kR$	$R.pořadí := L.pořadí + 1$ $R.hledaná := L.hledaná$ $L.nalezená :=$ if $R.nalezená \neq 0$ then $R.nalezená$ else if $L.hledaná = k.hod$ then $L.pořadí$ else 0
(2) $R \to /L$	$L.pořadí := R.pořadí$ $L.hledaná := R.hledaná$ $R.nalezená := L.nalezená$
(3) $R \to \varepsilon$	$R.nalezená := 0$

Proveďme výpočet atributů za předpokladu, že počáteční hodnoty dědičných atributů počátečního symbolu L jsou dány takto: $L.pořadí = 1$, $L.hledaná = 20$. Rozkladová tabulka pro základní $LL(1)$ gramatiku má tvar:

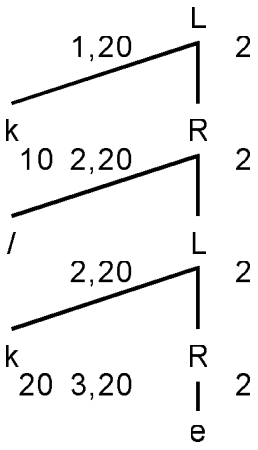
	k	$/$	ε
L	kR,1		
R		$/L$,2	ε,3

Algoritmus 6.27 [JPR03] provede pro vstupní řetězec $k(10)/k(20)$ tuto posloupnost operací:

konfigurace algoritmu	popis prováděných operací
$(\ k(10)/k(20), \varepsilon, L)$	expanze (1) $L \rightarrow kR$ určení atributů *L.pořadí* a *L.hledaná*
$(\ k(10)/k(20), L.pořadí = 1$ $L.hledaná = 20,\ kR\#(1))$	srovnání symbolu k uchování atributu *k.hod*
$(\quad /k(20), L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10,\quad \mathrm{R}\#(1))$	expanze (2) $R \rightarrow / L$ výpočet atributů *R.pořadí* a *R.hledaná*
$(\quad /k(20), L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20, L\#(2)\#(1))$	srovnání symbolu /
$(\quad k(20), L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20, L\#(2)\#(1))$	expanze $(1) L \rightarrow k\ R$ výpočet atributů *L.pořadí* a *L.hledaná*
$(\quad k(20), L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20$ $L.pořadí = 2$ $L.hledaná = 20, kR\#(1)\#(2)\#(1))$	srovnání symbolu k uchování atributu *k.hod*
$(\quad \varepsilon, L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20$ $L.pořadí = 2$ $L.hledaná = 20,$ $k.hod = 20,\quad \#(1)\#(2)\#(1))$	expanze (3) $R \rightarrow \varepsilon$ výpočet atributu *R.pořadí* a *R.hledaná*

(ε, $L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20$ $L.pořadí = 2$ $L.hledaná = 20$, $k.hod = 20$ $R.pořadí = 3$ $R.hledaná = 20$, #(3)#(1)#(2)#(1))	ukončení pravidla (3) výpočet atributu $R.nalezená$
(ε, $L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20$ $L.pořadí = 2$ $L.hledaná = 20$, $k.hod = 20$ $R.pořadí = 3$ $R.hledaná = 20$ $R.nalezená = 0$, #(1)#(2)#(1))	ukončení pravidla (1) výpočet atributu $L.nalezená$
(ε, $L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20$ $L.pořadí = 2$ $L.hledaná = 20$, $L.nalezená = 2$, #(2)#(1))	ukončení pravidla (2) výpočet atributu $R.nalezená$
(ε, $L.pořadí = 1$ $L.hledaná = 20$ $k.hod = 10$ $R.pořadí = 2$ $R.hledaná = 20$ $R.nalezená = 2$, #(1))	ukončení pravidla (1) výpočet atributu $L.nalezená$
(ε, $L.pořadí = 1$ $L.hledaná = 20$ $L.nalezená = 2$, ε)	ukončení výpočtu

Atributovaný derivační strom je uveden na obr. 13.6



Obrázek 13.6: Atributovaný derivační strom

13.5 Implementace výpočtu atributů metodou rekurzivního sestupu

Příklad 13.20
Jako příklad pro implementaci metodou rekurzivního sestupu vezmeme atributovou překladovou gramatiku z příkladu 13.19. Soubor rekurzivních procedur bude obsahovat procedury L a R.

```
procedure L(Lpořadí,Lhledaná:integer; var Lnalezená:integer);
  var Rpořadí,Rhledaná,Rnalezená,khod:integer;
    begin
      case SYMBOL of
        k:begin SROVNEJ(k,khod);
                Rpořadí := Lpořadí + 1;
                Rhledaná := L.hledaná;
                R(Rpořadí,Rhledaná,Rnalezená);
                if Rnalezená <> 0 then Lnalezená := Rnalezená
                                  else
                    if Lhledaná = khod then Lnalezená := Lpořadí
                                       else Lnalezená := 0;
        end;
        otherwise: CHYBA := true
      end;
    end;
```

```
procedure R(Rpořadí,Rhledaná:integer; var Rnalezená:integer);
  var Lpořadí,Lhledaná,Lnalezená:integer;
    begin
      case SYMBOL of
        lom:begin SROVNEJ(lom,nic);
                Lpořadí := Rpořadí;
                Lhledaná := Rhledaná;
                L(Lpořadí,Lhledaná,Lnalezená);
                Rnalezená := Lnalezená
            end;
        konec: Rnalezená := 0;
        otherwise: CHYBA := true
      end;
    end;

Hlavní program bude mít tvar:

program VYPOCET(input,output);
  var SYMBOL:VSTUPNISYMBOL;
       CHYBA:boolean;
  T,khod,nic:integer;
    Lhledaná,Lpořadí,Lnalezená:integer;

procedure SROVNEJ(X:VSTUPNÍSYMBOL; var Y:integer);
  begin
    if X = SYMBOL then
      begin
        Y := T;
        LEXIKÁLNÍANALÝZA(SYMBOL,T);
      end
                  else
      CHYBA := TRUE;
  end;
(* konec deklarace procedur *)

begin
    CHYBA := false;
    LEXIKÁLNÍANALÝZA(SYMBOL,khod);
    Lpořadí := 1;
    read(Lhledaná);
    L(Lpořadí,Lhledaná,Lnalezená);
    if not CHYBA then write(Lnalezená)
end.
```

13.6 Příklady pro cvičení

Cvičení 13.21
Navrhněte regulární překladovou gramatiku, která představuje model kalkulačky s klávesami $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, =, sin, +, *, \uparrow$. Určete, kolik registrů bude třeba na uchování mezivýsledků.

Cvičení 13.22
Je dána bezkontextová gramatika
$G = (E, E', T, T', F, a, +, *, (,), P, E)$, kde P obsahuje pravidla:

$$E' \rightarrow +TE'|\varepsilon \quad (13.1)$$
$$T \rightarrow FT' \quad (13.2)$$
$$T' \rightarrow *FT'|\varepsilon \quad (13.3)$$
$$F \rightarrow id|k|(E). \quad (13.4)$$

Doplňte tuto gramatiku na atributovou tak, aby počáteční symbol měl syntetizovaný atribut, jehož hodnota pro zadaný vstupní řetěz udává:

1. počet operátorů $+$,
2. počet operátorů $*$,
3. počet všech operátorů,
4. maximální hloubku závorkové struktury,
5. počet konstant,
6. počet identifikátorů,
7. kolikrát bylo použito ε-pravidlo,
8. kolik pravidel bylo použito v celé derivaci,
9. součet hodnot všech konstant,
10. aritmetický průměr hodnot všech konstant,
11. součin hodnot všech konstant,
12. hodnotu maximální konstanty.

Cvičení 13.23
Zvolte si vnitřní jazyk překladače jazyka PASCAL (ILP - Internal Language Pascal). Navrhněte atrubutové překladové gramatiky pro příklad následujících konstrukcí do jazyka ILP:

a) příkaz IF,

b) příkaz WHILE,

c) příkaz REPEAT,

d) příkaz FOR:

   i) beze změny velikosti kroku:
   $for\, i := V1\, to\, V2\, do\, P$,
   $for\, i := V1\, downto\, V2\, do\, P$,

   ii) se změnou velikosti kroku:
   $for\, i := V1\, step\, V3\, to\, V2\, do$,
   $for\, i := V1\, step\, V3\, downto\, V2\, do$,

e) příkaz CASE,

f) překlad deklarace funkce a procedury,

g) volání funkce a procedury,

h) překlad deklarace pole,

i) překlad deklarace recordu.