Slezská univerzita v Opavě
Filozoficko-přírodovědecká fakulta v Opavě

Šárka Vavrečková

Skripta do předmětů

Teorie jazyků a automatů

II

Základy teoretické informatiky

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě

Opava
9. prosince 2015

Anotace: Tato skripta jsou určena pro studenty předmětů *Teorie jazyků a automatů II* (obory Informatika a výpočetní technika, Informatika dvouoborové) a *Základy teoretické informatiky II* (obor Aplikovaná informatika). Navazujeme na látku probíranou v předchozím semestru a přecházíme k pokročilejším tématům.

Teorie jazyků a automatů II, Základy teoretické informatiky II

RNDr. Šárka Vavrečková, Ph.D.

Dostupné na: http://vavreckova.zam.slu.cz/formal.html

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě
Bezručovo nám. 13, Opava

Sázeno v systému LaTeX

Obsah

Kapitola 1

Konečné automaty a regulární jazyky

V této kapitole si zopakujeme učivo o konečných automatech a probereme věty a důkazy, které jsme v minulém semestru brali jen okrajově.

1.1 Definice konečného automatu

Nejdřív si zopakujeme všechny definice týkající se konečného automatu.

Definice 1.1 (Konečný automat)

Konečný automat *je uspořádaná pětice* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$*, kde je*

Q ... *neprázdná konečná množina stavů*
Σ ... *neprázdná konečná abeceda (množina signálů)*
δ ... *přechodová funkce, definovaná níže*
q_0 ... *počáteční stav,* $q_0 \in Q$
F ... *množina koncových stavů,* $F \subseteq Q$*,* $F \neq \emptyset$

Přechodová funkce δ *konečného automatu (dále KA)* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *je definována takto:*
$\delta\colon\ Q \times \Sigma \to Q$ *(zápis pomocí množin)*
$\delta(q, a) = r,\ \text{kde } q, r \in Q,\ a \in \Sigma$ *(symbolický zápis)*

Definice 1.2 (Konfigurace, počáteční a koncová konfigurace)

Označme Σ^* *množinu všech slov, která lze utvořit ze symbolů abecedy* Σ*. Konfigurace KA je uspořádaná dvojice* (q, w)*, kde* $q \in Q,\ w \in \Sigma^*$ *(nepřečtená část vstupní pásky).*

Dále definujeme tyto pojmy:

- Počáteční konfigurace *je konfigurace* (q_0, w_0)*, kde* q_0 *je počáteční stav automatu a* w_0 *je startovací (počáteční) slovo*
- Koncová konfigurace *je konfigurace* (q_f, ε)*, kde* $q_f \in F$

Definice 1.3 (Přechod mezi konfiguracemi)

Relaci přechodu mezi konfiguracemi značíme symbolem $\vdash$ *a definujeme ji takto:*

$$(q_i, aw) \vdash (q_j, w) \quad \overset{def}{\iff} \quad \delta(q_i, a) = q_j, \quad kde\ q_i, q_j \in Q,\ a \in \Sigma,\ w \in \Sigma^* \tag{1.1}$$

Definice 1.4 (Výpočet slova v konečném automatu)

Výpočet slova v konečném automatu je posloupnost konfigurací začínající počáteční konfigurací s daným slovem a končící některou koncovou konfigurací, vztah mezi sousedními konfiguracemi je dán relací přechodu $\vdash$.

Reflexivní a tranzitivní uzávěr relace přechodu mezi konfiguracemi $\vdash$ označíme $\vdash^*$, tranzitivní uzávěr $\vdash^+$.

Definice 1.5 (Rozpoznání (přijímání) slova konečným automatem)

Konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *rozpoznává (přijímá) slovo* w*, pokud existuje posloupnost výpočtu tohoto slova v automatu, tedy pokud se lze z počáteční konfigurace* (q_0, w_0) *postupným uplatňováním relací přechodu dostat do některé koncové konfigurace:*

$$(q_0, w) \vdash^* (q_f, \varepsilon), \quad kde\ q_f \in F \tag{1.2}$$

Definice 1.6 (Jazyk konečného automatu)

Jazyk konečného automatu $\mathcal{A}$ *je množina všech slov* w*, která automat přijímá:*

$$L(\mathcal{A}) = \{w \in \Sigma^* \ ;\ (q_0, w) \vdash^* (q_f, \varepsilon),\ kde\ q_f \in F\} \tag{1.3}$$

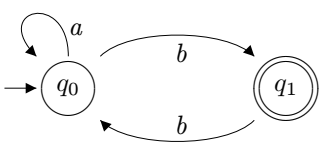
Automat $\mathcal{A}$ *rozpoznává jazyk* L_j*, pokud přijímá* právě *slova jazyka* L_j *(tj. přijímá všechna slova jazyka, ale nepřijímá žádné slovo do jazyka nepatřící).*

Značíme $L_j = L(\mathcal{A})$ *(jazyk* L_j *je rozpoznáván automatem* $\mathcal{A}$*, je jeho jazykem).*

Příklad 1.1

Zopakujeme si všechny možné zápisy konečného automatu. Základní specifikace je následující:
$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \delta,\ q_0,\ \{q_1\})$

Diagram:



δ-funkce:

$\delta(q_0, a) = q_0$
$\delta(q_0, b) = q_1$
$\delta(q_1, b) = q_0$

Tabulka přechodů:

stav\vstup	a	b
$\rightarrow q_0$	q_0	q_1
$\leftarrow q_1$	-	q_0

Jeden z možných výpočtů v automatu:
$(q_0, aab) \vdash (q_0, ab) \vdash (q_0, b) \vdash (q_1, \varepsilon)$ proto $aab \in L(\mathcal{A})$

Předchozí výpočet byl úspěšný, jednalo se o slovo patřící do jazyka rozpoznávaného automatem. Nicméně automat může mít na vstupu i slova nepatřící do jeho jazyka, například:

- $(q_0, bba) \vdash (q_1, ba) \vdash (q_0, a) \vdash (q_0, \varepsilon)$
  tato konfigurace není koncová: $q_0 \notin F$, proto $bba \notin L(\mathcal{A})$
- $(q_0, ba) \vdash (q_1, a) \vdash$???
  dál nelze pokračovat, ale konfigurace není koncová, proto $ba \notin L(\mathcal{A})$

Jazyk automatu:
$L(\mathcal{A}) = \{a^n b \ ; \ n \geq 0\} \cdot \left\{(ba^*b)^i \ ; \ i \geq 0\right\} \quad = \quad a^*b(ba^*b)^*$

Dále jsme si definovali nedeterministický konečný automat a ukázali, že třídy jazyků rozpoznávaných deterministickými a nedeterministickými automaty jsou navzájem ekvivalentní. Následoval totální (úplný) automat a dále postup redukce stavů automatu.

1.2 K uzávěrovým vlastnostem

V minulém semestru jsme se důkladně zabývali zejména uzávěrovými vlastnostmi třídy jazyků generovaných konečnými automaty vzhledem k regulárním operacím – sjednocení, zřetězení a iteraci. Nyní se zaměříme i na další operace, včetně příslušných důkazů.

1.2.1 Pozitivní iterace

Operace pozitivní iterace je podobná operaci iterace (Kleeneho uzávěru, hvězdičce) s tím rozdílem, že pokud do původního jazyka nepatřilo prázdné slovo, nebude patřit ani do výsledného jazyka. Srovnejme definici operace iterace a pozitivní iterace (původní jazyk označme L):

$$\text{iterace:} \quad L^* = \bigcup_{i=0}^{\infty} L^i \qquad \text{pozitivní iterace:} \quad L^+ = \bigcup_{i=1}^{\infty} L^i$$

Věta 1.1

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci pozitivní iterace.

Postup

Vezměme jakýkoliv konečný automat $\mathcal{A}_1 = (Q, \Sigma, \delta_1, q_0, F)$ rozpoznávající jazyk L_1. Sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ takový, že $L(\mathcal{A}) = L_1^+ = \bigcup_{i=1}^{\infty} L_1^i$. Tentokrát si nemusíme „hrát“ s počátečním stavem, protože do jazyka explicitně nepřidáváme prázdné slovo.

Postupujeme následovně:

- použijeme původní počáteční stav, množinu stavů, množinu koncových stavů,
- zajistíme iteraci – v koncových stavech přidáme reakce stejné, jaké jsou v počátečním stavu.

Zápis přechodové funkce:

$$\delta(p, a) = \begin{cases} \delta_1(p, a) \ ; \ p \in Q - F, \ a \in \Sigma \\ \delta_1(p, a) \cup \delta_1(q_0, a) \ ; \ p \in F, \ a \in \Sigma \end{cases}$$

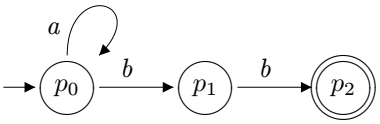
První řádek předpisu použijeme pro všechny stavy kromě koncových (jen přejmeme chování z původního automatu), druhý řádek platí pro koncové stavy. V nich necháme původní přechody a přidáme ty přechody, které vedou z počátečního stavu.

Příklad 1.2

Postup si ukážeme na automatu rozpoznávajícím jazyk $L_1 = \{a^i bb \,;\, i \geq 0\} = a^*bb$ (stejném jako u iterace). Tento jazyk je rozpoznáván automatem $\mathcal{A}_1 = (\{p_0, p_1, p_2\},\ \{a, b\},\ \delta_1,\ p_0,\ \{p_2\})$ určeným takto:

$\mathcal{A}_1$	a	b
$\rightarrow p_0$	p_0	p_1
p_1		p_2
$\leftarrow p_2$		

$$\delta_1(p_0, a) = p_0$$
$$\delta_1(p_0, b) = p_1$$
$$\delta_1(p_1, b) = p_2$$

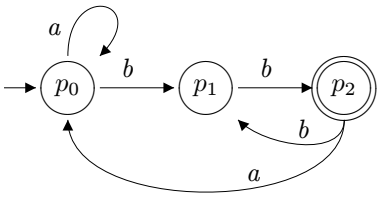


Sestrojíme automat $\mathcal{A}$ rozpoznávající jazyk $L(\mathcal{A}) = L_1^+$: $\mathcal{A} = (\{p_0, p_1, p_2\},\ \{a, b\},\ \delta,\ p_0,\ \{p_2\})$.

Přidáme nové přechody vedoucí z koncových stavů (v tomto případě ze stavu p_2) – „zkopírujeme“ přechody z počátečního stavu.

$\mathcal{A}$	a	b
$\rightarrow p_0$	p_0	p_1
p_1		p_2
$\leftarrow p_2$	p_0	p_1

$$\delta(p_0, a) = p_0 \qquad \delta(p_2, a) = p_0$$
$$\delta(p_0, b) = p_1 \qquad \delta(p_2, b) = p_1$$
$$\delta(p_1, b) = p_2$$



Důkaz (Věta 1.1): Tento důkaz bude velmi podobný důkazu pro iteraci. Dokážeme, že výše popsaný algoritmus vytvoří automat rozpoznávající jazyk, který je pozitivní iterací jazyka původního automatu, tedy že $L(\mathcal{A}) = L(\mathcal{A}_1)^+$. Použijeme důkaz matematickou indukcí. Protože platí $L^+ = \bigcup_{i=1}^{\infty} L^i$, matematická indukce se bude týkat různého počtu zřetězení jazyka, tedy postupně probereme $L^1,\ L^2$, atd.

Ke konečnému automatu $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ rozpoznávajícímu jazyk L_1 sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ tak, jak bylo popsáno v konstrukci před příkladem.

Báze: *Dokazujeme* $L_1^1 \subseteq L(\mathcal{A})$: rozlišíme dva případy – $|w| = 0$ a $|w| \geq 1$. První případ je triviální. $\varepsilon \in L_1$ právě tehdy a jen tehdy, když $\varepsilon \in L(\mathcal{A})$, protože s počátečním stavem q_0 se v algoritmu nijak nemanipuluje, včetně toho, zda patří do množiny koncových stavů.

Jestliže $|w| \geq 1$:

$\Rightarrow$ v automatu $\mathcal{A}_1$ existuje výpočet
$(q_0, w) \vdash \ldots \vdash (q_f, \varepsilon),\ q_x \in Q_1,\ q_f \in F_1$

$\Rightarrow$ protože veškeré přechody z δ_1 existují i v δ, pak v automatu $\mathcal{A}$ existuje výpočet
$(q_0, w) \vdash \ldots \vdash (q_f, \varepsilon),\ q_x \in Q,\ q_f \in F$

$\Rightarrow$ $w \in L(\mathcal{A}) \quad \Rightarrow \quad L_1^1 \subseteq L(\mathcal{A})$

Předpoklad: Dále budeme předpokládat, že $L_1^k \subseteq L(\mathcal{A})$ pro nějaké $k \geq 1$.

Krok indukce: *Dokazujeme*, že $L_1^{k+1} \subseteq L(\mathcal{A})$.
Vezměme slovo $w = w_1 \cdot w_2 \cdot \ldots \cdot w_{k+1}$, kde $|w_i| \geq 1,\ w_i = a_i \cdot w_i',\ w_i \in L_1$ pro $1 \leq i \leq k+1$.

$\Rightarrow$ pro každé $i \in \{1, \ldots, k+1\}$ existuje výpočet v $\mathcal{A}_1$:
$(q_0^1, w_i) \vdash (q_{x_i}, w_i') \vdash \ldots \vdash (q_{f_i}, \varepsilon),\ q_{x_i} \in Q_1,\ q_{f_i} \in F_1$

$\Rightarrow$ v $\mathcal{A}_1$ existují předpisy $\delta_1(q_0^1, a_i) \ni q_{x_i}$

$\Rightarrow$ v $\mathcal{A}$ existují předpisy $\delta(q_0, a_i) \ni q_{x_i},\ \delta(q_f, a_i) \ni q_{x_i}$ pro některé $q_f \in F$

$\Rightarrow$ pro každé $i \in \{1, \ldots, k+1\}$ v automatu $\mathcal{A}$ existují podvýpočty
$(q_0, w_i) \vdash (q_{x_i}, w_i') \vdash \ldots \vdash (q_{f_i}, \varepsilon),\ q_{x_i} \in Q,\ q_{f_i} \in F$
$(q_f, w_i) \vdash (q_{x_i}, w_i') \vdash \ldots \vdash (q_{f_i}, \varepsilon),\ q_{x_i} \in Q,\ q_{f_i} \in F$

$\Rightarrow$ první použijeme pro slovo w_1, druhý pro slova $w_i,\ i \in \{2, \ldots, k+1\}$:
$(s_0, w_1 \cdot w_2 \cdot \ldots \cdot w_{k+1}) \vdash (q_{x_1}, w_1' \cdot w_2 \cdot \ldots \cdot w_{k+1}) \vdash \ldots$
$\vdash (q_{f_1}, w_2 \cdot \ldots \cdot w_{k+1}) \vdash (q_{x_2}, w_2' \cdot \ldots \cdot w_{k+1}) \vdash \ldots$
$\vdash (q_{f_k}, w_{k+1}) \vdash (q_{x_{k+1}}, w_{k+1}') \vdash \ldots \vdash (q_{f_{k+1}}, \varepsilon)$

$\Rightarrow$ $w \in L(\mathcal{A}) \quad \Rightarrow \quad L_1^{k+1} \subseteq L(\mathcal{A})$

Zbývá dokázat opačný směr – pokud slovo $w \notin L_1^*$, pak také $w \notin L(\mathcal{A})$. Do funkce δ jsme přidávali pouze takové přechody, které zajistily „návrat“ na začátek po ukončení zpracování jednoho slova z jazyka L_1, proto můžeme říct, že pokud $w \notin L_1^*$, pak $w \notin L(\mathcal{A})$.

Z toho vyplývá, že jazyk automatu $\mathcal{A}$ je pozitivní iterací jazyka automatu $\mathcal{A}_1$. □

1.2.2 Zrcadlový obraz

Věta 1.2

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci zrcadlového obrazu (reverze).

Postup

Je dán regulární jazyk L_1 rozpoznávaný konečným automatem $\mathcal{A}_1$, kde $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$, $L_1 = L(\mathcal{A}_1)$. Vytváříme jazyk $L = L_1^R$ a konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F),\ L = L(\mathcal{A})$. Postupujeme takto:

- převrátíme všechny přechody,
- vytvoříme nový počáteční stav q_0, platí $q_0 \notin Q_1$,
- ze stavu q_0 budou vést přechody, které (po převrácení přechodů) vedou z bývalých koncových stavů (tj. z momentálních „virtuálních počátečních stavů“),
- množina koncových stavů bude obsahovat pouze původní počáteční stav, ale pokud do jazyka L_1 patřilo i slovo ε, pak tam zařadíme i stav q_0.

Takže platí:

- $Q = Q_1 \cup \{q_0\}$,

- $F = \begin{cases} \{q_0^1\} \ ; \ \varepsilon \notin L_1 \\ \{q_0^1, q_0\} \ ; \ \varepsilon \in L_1 \end{cases}$
- přechodová funkce: $\delta(p,a) = \begin{cases} \{r \ ; \ \delta_1(r,a) \ni p, \ a \in \Sigma\} \ ; \ p \in Q_1 \\ \{r \ ; \ \delta_1(r,a) \ni q_f, \ q_f \in F_1, \ a \in \Sigma\} \ ; \ p = q_0 \end{cases}$

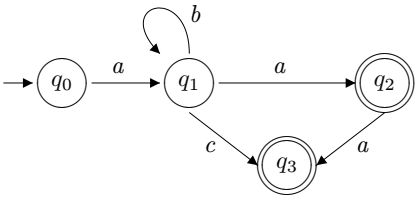
Příklad 1.3

Je dán tento jazyk a konečný automat, který ho rozpoznává:
$L_1 = \{ab^i a^{\{1,2\}} \ ; \ i \geq 0\} \cup \{ab^i c \ ; \ i \geq 0\}$
$= \{ab^i \ ; \ i \geq 0\} \cdot \{a, aa, c\}$

$\mathcal{A}_1 = (\{q_0, q_1, q_2, q_3\}, \ \{a,b,c\}, \ \delta_1, \ q_0, \ \{q_2, q_3\})$

$\mathcal{A}_1$	a	b	c
$\rightarrow q_0$	q_1		
q_1	q_2	q_1	q_3
$\leftarrow q_2$	q_3		
$\leftarrow q_3$			

$\delta_1(q_0, a) = q_1$
$\delta_1(q_1, a) = q_2$
$\delta_1(q_1, b) = q_1$
$\delta_1(q_1, c) = q_3$
$\delta_1(q_2, a) = q_3$



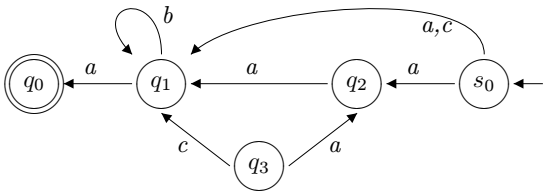
Sestrojíme automat $\mathcal{A}$ rozpoznávající jazyk $L(\mathcal{A}) = L_1^R$. V diagramu změníme orientaci hran, v předpisu přechodové funkce zaměníme stav v závorce se stavem za rovnítkem, v tabulce zaměníme umístění stavu v označení řádku a stavu v buňce na tomto řádku.

Protože automat $\mathcal{A}_1$ má dva koncové stavy, je třeba vytvořit nový stav s_0, který „přejme roli“ těchto stavů na začátku výpočtu každého slova v automatu $\mathcal{A}$.

$\mathcal{A} = (\{q_0, q_1, q_2, q_3, s_0\}, \ \{a,b,c\}, \ \delta, \ s_0, \ \{q_0\})$

$\mathcal{A}$	a	b	c
$\rightarrow s_0$	q_1, q_2		q_1
$\leftarrow q_0$			
q_1	q_0	q_1	
q_2	q_1		
q_3	q_2		q_1

$\delta(s_0, a) = \{q_1, q_2\}$
$\delta(s_0, c) = \{q_1\}$
$\delta(q_1, a) = \{q_0\}$
$\delta(q_1, b) = \{q_1\}$
$\delta(q_2, a) = \{q_1\}$
$\delta(q_3, a) = \{q_2\}$
$\delta(q_3, c) = \{q_1\}$



Je zřejmé, že stav q_3 je nedosažitelný, tedy dalším krokem by mohlo být jeho odstranění.

Důkaz (Věta 1.2): Ke konečnému automatu $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ rozpoznávajícímu jazyk L_1 sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ tak, jak bylo popsáno v konstrukci před příkladem. Dokážeme, že tento algoritmus vytvoří automat rozpoznávající jazyk, který je zrcadlovým obrazem jazyka původního automatu, tedy že $L(\mathcal{A}) = L(\mathcal{A}_1)^R$.

V důkazu budeme počítat s tím, že operace zrcadlového obrazu řetězců je sama k sobě inverzní, tj. $(w^R)^R = w$.

Dokazujeme $L(\mathcal{A}_1)^R \subseteq L(\mathcal{A})$:
Nechť $w \in L(\mathcal{A}_1), \ w^R \in L(\mathcal{A}_1)^R, \ w = a_1 \cdots a_n, \ n \geq 1$.

$\Rightarrow$ v $\mathcal{A}_1$ existuje posloupnost konfigurací
$(q_0^1, a_1 \cdots a_{n-1}a_n) \vdash \ldots \vdash (r, a_n) \vdash (q_f, \varepsilon)$, kde $r \in Q_1,\ q_f \in F_1$

$\wedge$ podle algoritmu

- pro všechny přechody podle $\delta_1(r, a) \ni p$ v automatu $\mathcal{A}_1$ je vytvořeno $\delta(p, a) \ni r$ v automatu $\mathcal{A}$ (otočení všech přechodů),
- pro $\delta_1(r, a) \ni q_f,\ r \in Q_1,\ a \in \Sigma$ jsme v $\mathcal{A}$ vytvořili $\delta(q_0, a) \ni r$

$\Rightarrow$ v $\mathcal{A}$ existuje posloupnost konfigurací
$(q_0, a_n a_{n-1} \cdots a_1) \vdash (r, a_{n-1} \cdots a_1) \vdash \ldots \vdash (q_0^1, \varepsilon)$, kde $r \in Q_1,\ q_f \in F_1$

$\Rightarrow$ $w^R \in L(\mathcal{A})$

Dokazujeme $L(\mathcal{A}_1)^R \supseteq L(\mathcal{A})$:
Nechť $w \in L(\mathcal{A}),\ w = a_1 \cdots a_n,\ n \geq 1$.

$\Rightarrow$ v $\mathcal{A}$ existuje posloupnost konfigurací
$(q_0, a_1 a_2 \cdots a_n) \vdash (s, a_1 a_2 \cdots a_n) \vdash \ldots \vdash (s_f, \varepsilon)$, kde $s \in Q,\ s_f \in F$

$\wedge$ podle algoritmu

- pro všechny přechody podle $\delta_1(r, a) \ni p$ v automatu $\mathcal{A}_1$ je vytvořeno $\delta(p, a) \ni r$ v automatu $\mathcal{A}$,
- pro $\delta_1(s, a) \ni s_f,\ s \in Q_1,\ a \in \Sigma$ jsme v $\mathcal{A}$ vytvořili $\delta(q_0, a) \ni r$

$\Rightarrow$ v $\mathcal{A}_1$ existuje posloupnost konfigurací
$(q_0^1, a_n \cdots a_2 a_1) \vdash \ldots \vdash (s, a_1) \vdash (s_f, \varepsilon)$, kde $s \in Q_1,\ s_f \in F_1$

$\Rightarrow$ $w^R \in L(\mathcal{A}_1)$

Pro prázdné slovo $w = \varepsilon$ tvrzení o obou inkluzích platí také: právě tehdy, když $\varepsilon \in L(\mathcal{A}_1)$, platí podle algoritmu $q_0 \in F$, tedy $\varepsilon \in L(\mathcal{A}) \iff \varepsilon \in L(\mathcal{A}_1)^R$. □

1.2.3 Průnik

Věta 1.3

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci průniku.

Následující postup se nám bude hodit i později, podobný algoritmus totiž budeme potřebovat i u operace rozdílu jazyků.

Postup

Budeme předpokládat, že některé dva regulární jazyky L_1 a L_2 jsou rozpoznávány *deterministickými* konečnými automaty
$\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1),\ L_1 = L(\mathcal{A}_1)$ a
$\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2),\ L_2 = L(\mathcal{A}_2),\ Q_1 \cap Q_2 = \emptyset$.

Vytváříme jazyk $L = L_1 \cap L_2$ a automat
$\mathcal{A} = (Q, \Sigma_1 \cap \Sigma_2, \delta, q_0, F),\ L = L(\mathcal{A})$.

Potřebujeme, aby automat $\mathcal{A}$ rozpoznával právě ta slova, která rozpoznávají automaty $\mathcal{A}_1$ a $\mathcal{A}_2$. Toho docílíme jednoduše tak, že totéž slovo necháme paralelně zpracovávat oba původní automaty, resp. v automatu $\mathcal{A}$ spustíme paralelní simulaci výpočtu původních automatů. Vyhledáme dvojice přechodů, jeden z automatu $\mathcal{A}_1$, druhý z automatu $\mathcal{A}_2$, takové, že oba tyto přechody lze použít ve stejné situaci (v našem případě při stejném vstupním signálu). Takže pokud v prvním automatu máme přechod $\delta_1(r,a) = s$ a v druhém $\delta_2(u,a) = v$, pak v automatu $\mathcal{A}$ bude přechod $\delta([r,u],a) = [s,v]$.

Množinu stavů výsledného automatu bude tvořit množina uspořádaných dvojic takových, že první prvek dvojice je stav z Q_1, druhý prvek je stav z Q_2. Tedy se vlastně jedná o vytvoření kartézského součinu množin Q_1 a Q_2, ale ve skutečnosti některé jeho prvky (nové stavy) budou v automatu $\mathcal{A}$ nedosažitelné či nadbytečné, tedy v reálu půjde pravděpodobně o podmnožinu kartézského součinu. Shrňme, jak bude výsledný automat vypadat:

- množina stavů je množinou uspořádaných dvojic:
  $Q = Q_1 \times Q_2 = \{[x,y] \ ; \ x \in Q_1, \ y \in Q_2\}$
- počáteční stav je uspořádaná dvojice $q_0 = [q_0^1, q_0^2]$
- množina koncových stavů bude podmnožinou množiny Q obsahující jen ty dvojice původních stavů, které jsou v automatech $\mathcal{A}_1$ a $\mathcal{A}_2$ koncové:
  $F = F_1 \times F_2 = \{[x,y] \ ; \ x \in F_1, \ y \in F_2\}$
- přechodová funkce δ vychází z přechodových funkcí δ_1 a δ_2:
  $\delta([x,y],a) = [u,v]$, kde $\delta_1(x,a) = u$, $\delta_2(y,a) = v$, $a \in \Sigma_1 \cap \Sigma_2$

Příklad 1.4

Sestrojíme automat rozpoznávající průnik dvou jazyků, potřebujeme k nim ekvivalentní *deterministické* automaty:

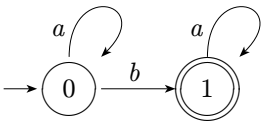
$L_1 = \{a^i b a^j \ ; \ i,j \geq 0\} = a^*ba^* \quad \mathcal{A}_1 = (\{0,1\}, \ \{a,b\}, \ \delta_1, \ 0, \ \{1\})$

$L_2 = \{(ab)^i a a^j \ ; \ i,j \geq 0\} = (ab)^*aa^* \quad \mathcal{A}_2 = (\{2,3,4\}, \ \{a,b\}, \ \delta_2, \ 2, \ \{3,4\})$

Automaty pro jazyky L_1 a L_2:

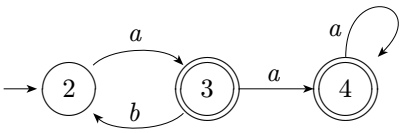
$\mathcal{A}_1$	a	b
$\to 0$	0	1
$\leftarrow 1$	1	

$\delta_1(0,a) = 0$
$\delta_1(0,b) = 1$
$\delta_1(1,a) = 1$



$\mathcal{A}_2$	a	b
$\to 2$	3	
$\leftarrow 3$	4	2
$\leftarrow 4$	4	

$\delta_2(2,a) = 3$
$\delta_2(3,a) = 4$
$\delta_2(3,b) = 2$
$\delta_2(4,a) = 4$



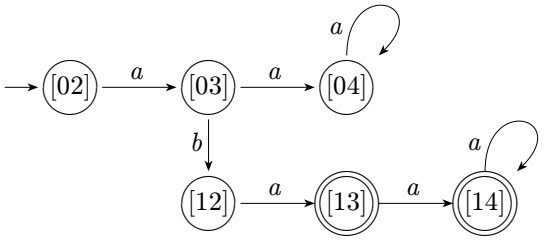
Množina stavů bude množinou uspořádaných dvojic původních stavů, ale z důvodu zkrácení a zjednodušení zápisu nebudeme psát čárku oddělující prvky uspořádané dvojice. Sestrojíme automat $\mathcal{A} = (Q, \Sigma, \delta, [02], F)$, kde

- $Q = Q_1 \times Q_2 = \{[02],[03],[04],[12],[13],[14]\}$
- $F = F_1 \times F_2 = \{[13],[14]\}$

- přechodová funkce δ:

$\mathcal{A}$	a	b
$\rightarrow$ [02]	[03]	
[03]	[04]	[12]
[04]	[04]	
[12]	[13]	
$\leftarrow$ [13]	[14]	
$\leftarrow$ [14]	[14]	

$\delta([02], a) = [03]$
$\delta([03], a) = [04]$
$\delta([03], b) = [12]$
$\delta([04], a) = [04]$
$\delta([12], a) = [13]$
$\delta([13], a) = [14]$
$\delta([14], a) = [14]$



Stav [04] je, jak vidíme, nadbytečný, proto by nebyl problém ho odstranit.

Důkaz (Věta 1.3): Je třeba dokázat, že pro jakékoliv slovo $w \in \Sigma_1^* \cap \Sigma_2^*$ platí: $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \iff w \in L(\mathcal{A})$. Budeme používat stejné značení jako ve výše uvedeném popisu konstrukce. Předpokládáme, že automaty $\mathcal{A}_1$ a $\mathcal{A}_2$ jsou deterministické.

Dokazujeme $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \subseteq L(\mathcal{A})$:
Nechť $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2),\ |w| = n,\ n \geq 1$.

- $\Rightarrow$ v $\mathcal{A}_1$ existuje výpočet $(q_0^1, w) \vdash^* (q_f^1, \varepsilon),\ q_f^1 \in F_1$ o délce n přes posloupnost stavů $q_0^1 = r_0, \ldots, r_n = q_f^1$
- $\wedge$ v $\mathcal{A}_2$ existuje výpočet $(q_0^2, w) \vdash^* (q_f^2, \varepsilon),\ q_f^2 \in F_2$ o délce n přes posloupnost stavů $q_0^2 = s_0, \ldots, s_n = q_f^2$
- $\wedge$ v množině stavů Q budou dle algoritmu obsaženy obsaženy stavy $[r_i, s_i],\ 0 \leq i \leq n$
- $\wedge$ pro přechodovou funkci δ platí $\delta([x, y], a) = [u, v],$ kde $\delta_1(x, a) = u,\ \delta_2(y, a) = v,\ a \in \Sigma_1 \cap \Sigma_2$
- $\Rightarrow$ v $\mathcal{A}$ existuje výpočet $([q_0^1, q_0^2], w) \vdash^* ([q_f^1, q_f^2], \varepsilon)$ o délce n přes posloupnost stavů $[q_0^1, q_0^2] = [r_0, s_0], \ldots, [r_n, s_n] = [q_f^1, q_f^2]$
- $\Rightarrow$ $w \in L(\mathcal{A})$

Dokazujeme $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \supseteq L(\mathcal{A})$:
Nechť $w \in L(\mathcal{A}),\ |w| = n,\ n \geq 1$.

- $\Rightarrow$ v $\mathcal{A}$ existuje výpočet $([q_0^1, q_0^2], w) \vdash^* ([q_f^1, q_f^2], \varepsilon),\ q_f^1 \in F_1,\ q_f^2 \in F_2$ o délce n přes posloupnost stavů $[q_0^1, q_0^2] = [r_0, s_0], \ldots, [r_n, s_n] = [q_f^1, q_f^2]$
- $\Rightarrow$ vzhledem k algoritmu existuje v automatu $\mathcal{A}_1$ výpočet $(q_0^1, w) \vdash^* (q_f^1, \varepsilon),\ q_f^1 \in F_1$ o délce n přes posloupnost stavů $q_0^1 = r_0, \ldots, r_n = q_f^1$
  a v automatu $\mathcal{A}_2$ výpočet $(q_0^2, w) \vdash^* (q_f^2, \varepsilon),\ q_f^2 \in F_2$ o délce n přes posloupnost stavů $q_0^2 = s_0, \ldots, s_n = q_f^2$
- $\Rightarrow$ $w \in L(\mathcal{A}_1) \quad \wedge \quad w \in L(\mathcal{A}_2)$
- $\Rightarrow$ $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$

Zbývá prověřit algoritmus pro $|w| = 0$. Zde si stačí uvědomit, že $[q_0^1, q_0^2] \in F$ právě tehdy a jen tehdy, pokud $q_0^1 \in F_1$ a zároveň $q_0^2 \in F_2$. Proto $\varepsilon \in L(\mathcal{A}) \iff \varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. □

1.2.4 Doplněk

Věta 1.4

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci doplňku jazyka v dané abecedě.

Postup

Sestrojení konečného automatu rozpoznávajícího doplněk jazyka jiného konečného automatu nebudeme podrobně probírat, jen si tento algoritmus stručně nastíníme:

- vezmeme původní automat, budeme chtít, aby byl deterministický a totální,
- pokud F_1 je původní množina koncových stavů, nová množina koncových stavů bude $F = Q - F_1$.

Poznámka:

Důkaz této věty bude poněkud jiného druhu než jiné, které v těchto skriptech najdeme. Bylo by možné sice vytvořit podobný důkaz jako u předchozích vět, ale zde si ukážeme, že v některých případech to jde i mnohem jednodušeji. Jak víme z předchozího semestru, jazyk je ve skutečnosti množina, se kterou můžeme provádět různé množinové operace včetně sjednocení, průniku a doplňku, čehož zde využijeme.

Důkaz: Pro jakékoliv dva jazyky L_1, L_2 platí De Morganovy zákony:

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

A zároveň budeme předpokládat, že jazyky L_1, L_2 jsou regulární, tedy patří do třídy jazyků rozpoznávaných konečnými automaty. *Důkaz povedeme sporem.*

Předpokládejme tedy, že třída jazyků rozpoznávaných konečnými automaty není uzavřena vzhledem k operaci doplňku. Zároveň víme z předchozích vět a důkazů, že tato třída je uzavřena vzhledem k ostatním operacím, které se v uvedeném vztahu vyskytují – sjednocení a průniku.

$\Rightarrow$ Na levé straně uvedeného vztahu je rozhodně regulární jazyk, tedy $L_1 \cup L_2$ patří do třídy jazyků rozpoznávaných konečnými automaty.

$\wedge$ Pokud by tato třída nebyla uzavřena vzhledem k doplňku, pak by nebylo možno tvrdit, že $\overline{L_1}$ a $\overline{L_2}$ jsou regulární, tedy by některý z těchto jazyků nemusel být regulární. Totéž platí i o celé pravé straně $\overline{\overline{L_1} \cap \overline{L_2}}$.

$\Rightarrow$ Na levé straně máme vždy takový jazyk, který patří do třídy jazyků rozpoznávaných konečnými automaty, ale na pravé straně nikoliv $\Rightarrow$ *spor*.

$\Rightarrow$ Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci doplňku.

□

1.2.5 Rozdíl

Věta 1.5

Třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k operaci rozdílu.

Výsledkem rozdílu dvou jazyků je jazyk obsahující právě ta slova prvního jazyka, která se nenacházejí v druhém jazyce. Jak bylo výše naznačeno, postup bude velmi podobný tomu, který jsme probírali u operace průniku, ale navíc budeme požadovat, aby oba původní automaty byly totální.

Postup

Budeme předpokládat, že dva regulární jazyky L_1 a L_2 jsou rozpoznávány deterministickými úplnými konečnými automaty
$\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1),\ L_1 = L(\mathcal{A}_1)$ a
$\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2),\ L_2 = L(\mathcal{A}_2),\ Q_1 \cap Q_2 = \emptyset.$

Vytváříme jazyk $L = L_1 - L_2$ a automat
$\mathcal{A} = (Q, \Sigma_1 \cup \Sigma_2, \delta, q_0, F),\ L = L(\mathcal{A}).$

Totéž slovo necháme paralelně zpracovávat oba původní automaty, resp. v automatu $\mathcal{A}$ spustíme paralelní simulaci výpočtu původních automatů. Takže pokud v prvním automatu máme přechod $\delta_1(r, a) = s$ a v druhém $\delta_2(u, a) = v$, pak v automatu $\mathcal{A}$ bude přechod $\delta([r, u], a) = [s, v]$. Množinu stavů výsledného automatu bude tvořit množina uspořádaných dvojic takových, že první prvek dvojice je stav z Q_1, druhý prvek je stav z Q_2.

Odlišnost postupu oproti operaci průniku bude v množině koncových stavů. Zatímco u průniku byly v množině koncových stavů všechny takové dvojice, kde oba prvky byly v původních množinách koncových stavů (výsledný automat rozpoznával právě ta slova, která rozpoznávaly oba původní automaty), v případě rozdílu zařadíme do množiny koncových stavů takové dvojice, kde první prvek *je* koncový v prvním automatu a zároveň druhý prvek *není* koncový v druhém automatu.

Shrňme, jak bude výsledný automat vypadat:

- množina stavů: $Q = Q_1 \times Q_2 = \{[x, y]\ ;\ x \in Q_1,\ y \in Q_2\}$
- počáteční stav je uspořádaná dvojice $q_0 = [q_0^1, q_0^2]$
- množina koncových stavů bude podmnožinou množiny Q:
  $F = F_1 \times (Q_2 - F_2) = \{[x, y]\ ;\ x \in F_1,\ y \in (Q_2 - F_2)\}$
- přechodová funkce δ vychází z přechodových funkcí δ_1 a δ_2:
  $\delta([x, y], a) = [u, v],$ kde $\delta_1(x, a) = u,\ \delta_2(y, a) = v,\ a \in \Sigma_1 \cap \Sigma_2$

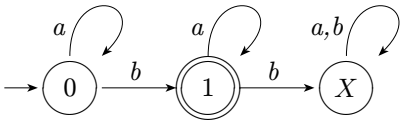
Příklad 1.5

Sestrojíme automat rozpoznávající rozdíl jazyků, k nimž máme *deterministické totální* automaty:
$L_1 = \{a^i b a^j\ ;\ i, j \geq 0\} = a^* b a^*$ $\quad \mathcal{A}_1 = (\{0, 1, X\},\ \{a, b\},\ \delta_1,\ 0,\ \{1\})$
$L_2 = \{(ab)^i a a^j\ ;\ i, j \geq 0\} = (ab)^* a a^*$ $\quad \mathcal{A}_2 = (\{2, 3, 4, Y\},\ \{a, b\},\ \delta_2,\ 2,\ \{3, 4\})$

Úplné deterministické automaty pro jazyky L_1 a L_2:

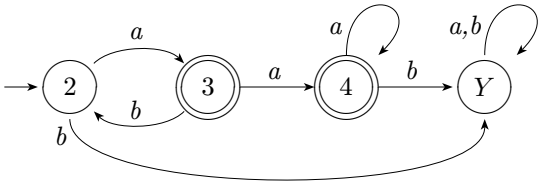
$\mathcal{A}_1$	a	b
$\to 0$	0	1
$\leftarrow 1$	1	X
X	X	X

$\delta_1(0,a)=0$ $\quad$ $\delta_1(1,b)=X$
$\delta_1(0,b)=1$ $\quad$ $\delta_1(X,a)=X$
$\delta_1(1,a)=1$ $\quad$ $\delta_1(X,B)=X$



$\mathcal{A}_2$	a	b
$\to 2$	3	Y
$\leftarrow 3$	4	2
$\leftarrow 4$	4	Y
Y	Y	Y

$\delta_2(2,a)=3$ $\quad$ $\delta_2(4,a)=4$
$\delta_2(2,b)=Y$ $\quad$ $\delta_2(4,b)=Y$
$\delta_2(3,a)=4$ $\quad$ $\delta_2(Y,a)=Y$
$\delta_2(3,b)=2$ $\quad$ $\delta_2(Y,b)=Y$



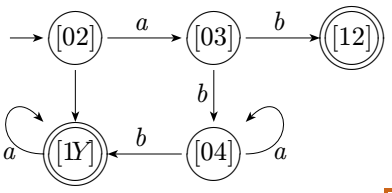
Sestrojíme automat $\mathcal{A}=(Q,\Sigma,\delta,[02],F)$, kde

- $Q=Q_1\times Q_2=\{[02],[03],[04],[0Y],[12],[13],[14],[1Y],[X1],[X2],[X3],[X4],[XY]\}$
- $F=F_1\times(Q_2-F_2)=\{[12],[1Y]\}$
- přechodová funkce δ:

$\mathcal{A}$	a	b
$\to [02]$	$[03]$	$[1Y]$
$[03]$	$[04]$	$[12]$
$[04]$	$[04]$	$[1Y]$
$[0Y]$	$[0Y]$	$[1Y]$
$\leftarrow [12]$	$[13]$	$[XY]$
$[13]$	$[14]$	$[X2]$
$[14]$	$[14]$	$[XY]$
$\leftarrow [1Y]$	$[1Y]$	$[XY]$
$[X2]$	$[X3]$	$[XY]$
$[X3]$	$[X4]$	$[X2]$
$[X4]$	$[X4]$	$[XY]$
$[XY]$	$[XY]$	$[XY]$

$\delta([02],a)=[03]$ $\quad$ $\delta([14],a)=[14]$
$\delta([02],b)=[1Y]$ $\quad$ $\delta([14],b)=[XY]$
$\delta([03],a)=[04]$ $\quad$ $\delta([1Y],a)=[1Y]$
$\delta([03],b)=[12]$ $\quad$ $\delta([1Y],b)=[XY]$
$\delta([04],a)=[04]$ $\quad$ $\delta([X2],a)=[X3]$
$\delta([04],b)=[1Y]$ $\quad$ $\delta([X2],b)=[XY]$
$\delta([0Y],a)=[0Y]$ $\quad$ $\delta([X3],a)=[X4]$
$\delta([0Y],b)=[1Y]$ $\quad$ $\delta([X3],b)=[X2]$
$\delta([12],a)=[13]$ $\quad$ $\delta([X4],a)=[X4]$
$\delta([12],b)=[XY]$ $\quad$ $\delta([X4],b)=[XY]$
$\delta([13],a)=[14]$ $\quad$ $\delta([XY],a)=[XY]$
$\delta([13],b)=[X2]$ $\quad$ $\delta([XY],b)=[XY]$

Diagram celého výsledného automatu nemá smysl kreslit, byl by velmi nepřehledný. Proto je lepší nejdřív provést redukci (postup necháme na čtenáři), po které nám z 12 stavů zbývá pouze 5:



Důkaz (Věta 1.5): Je třeba dokázat, že pro jakékoliv slovo $w\in\Sigma_1^*\cup\Sigma_2^*$ platí: $w\in L(\mathcal{A}_1)-L(\mathcal{A}_2)\iff w\in L(\mathcal{A})$. Budeme používat stejné značení jako ve výše uvedeném popisu konstrukce. Předpokládáme, že automaty $\mathcal{A}_1$ a $\mathcal{A}_2$ jsou deterministické a totální.

Dokazujeme $L(\mathcal{A}_1)-L(\mathcal{A}_2)\subseteq L(\mathcal{A})$:
Nechť $w\in L(\mathcal{A}_1)-L(\mathcal{A}_2),\ |w|=n,\ n\geq 1$.

$\Rightarrow$ v $\mathcal{A}_1$ existuje výpočet $(q_0^1, w) \vdash^* (q_f^1, \varepsilon),\ q_f^1 \in F_1$ o délce n přes posloupnost stavů $q_0^1 = r_0, \ldots, r_n = q_f^1$

$\wedge$ v $\mathcal{A}_2$ existuje výpočet $(q_0^2, w) \vdash^* (q_g^2, \varepsilon),\ q_g^2 \in Q_2 - F_2$ o délce n přes posloupnost stavů $q_0^2 = s_0, \ldots, s_n = q_x^2$

$\wedge$ v množině stavů Q budou dle algoritmu obsaženy obsaženy stavy $[r_i, s_i],\ 0 \leq i \leq n$

$\wedge$ pro přechodovou funkci δ platí $\delta([x, y], a) = [u, v]$, kde $\delta_1(x, a) = u$, $\delta_2(y, a) = v$, $a \in \Sigma_1 \cap \Sigma_2$

$\Rightarrow$ v $\mathcal{A}$ existuje výpočet $([q_0^1, q_0^2], w) \vdash^* ([q_f^1, q_g^2], \varepsilon)$ o délce n přes posloupnost stavů $[q_0^1, q_0^2] = [r_0, s_0], \ldots, [r_n, s_n] = [q_f^1, q_g^2]$

$\Rightarrow$ $w \in L(\mathcal{A})$

Dokazujeme $L(\mathcal{A}_1) - L(\mathcal{A}_2) \supseteq L(\mathcal{A})$:

Nechť $w \in L(\mathcal{A}),\ |w| = n,\ n \geq 1$.

$\Rightarrow$ v $\mathcal{A}$ existuje výpočet $([q_0^1, q_0^2], w) \vdash^* ([q_f^1, q_g^2], \varepsilon),\ q_f^1 \in F_1,\ q_g^2 \in Q_2 - F_2$ o délce n přes posloupnost stavů $[q_0^1, q_0^2] = [r_0, s_0], \ldots, [r_n, s_n] = [q_f^1, q_g^2]$

$\Rightarrow$ vzhledem k algoritmu existuje v automatu $\mathcal{A}_1$ výpočet $(q_0^1, w) \vdash^* (q_f^1, \varepsilon),\ q_f^1 \in F_1$ o délce n přes posloupnost stavů $q_0^1 = r_0, \ldots, r_n = q_f^1$

a v automatu $\mathcal{A}_2$ výpočet $(q_0^2, w) \vdash^* (q_g^2, \varepsilon),\ q_g^2 \in Q_2 - F_2$ o délce n přes posloupnost stavů $q_0^2 = s_0, \ldots, s_n = q_g^2$

$\Rightarrow$ $w \in L(\mathcal{A}_1) \quad \wedge \quad w \notin L(\mathcal{A}_2)$

$\Rightarrow$ $w \in L(\mathcal{A}_1) - L(\mathcal{A}_2)$

Pro $|w| = 0$: platí $[q_0^1, q_0^2] \in F$ právě tehdy a jen tehdy, pokud $q_0^1 \in F_1 \ \wedge \ q_0^2 \in Q_2 - F_2$. Tedy $\varepsilon \in L(\mathcal{A}) \iff \varepsilon \in L(\mathcal{A}_1) - L(\mathcal{A}_2)$. □

1.3 Kritéria regulárnosti jazyka

Jak poznat, zda je daný jazyk regulární? Zatím víme, že jazyk je regulární, jestliže

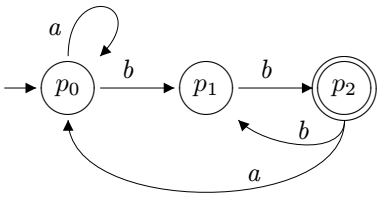
- ho lze reprezentovat pomocí regulárního výrazu,
- k němu lze sestrojit ekvivalentní konečný automat,
- k němu lze sestrojit ekvivalentní regulární gramatiku.

Existují však i jiné možnosti, které obvykle slouží k *popření regulárnosti* – dokázání, že daný jazyk není regulární. To je užitečné zvláště tehdy, když se nám nedaří vytvořit ekvivalentní regulární výraz, konečný automat ani gramatiku, ale zároveň si nejsme jisti, proč – buď jazyk sice je regulární, ale uvedené postupy jsou příliš složité, nebo jazyk doopravdy regulární není.

1.3.1 Pumping lemma pro regulární jazyky

Jednou z možností, jak dokázat, že daný jazyk není regulární, je Pumping lemma pro regulární jazyky. Taky se nazývá *lemma o vkládání* nebo *pumpovací věta* pro regulární jazyky. Jedná se o jednu z tzv. „strukturálních vlastností“ regulárních jazyků, tedy její tvrzení se vztahuje ke struktuře slov jazyka.

Nejdřív si vysvětlíme princip, ze kterého Pumping lemma vychází, potom teprve přejdeme k samotnému lemmatu.



Předpokládejme, že L je regulární jazyk. Víme, že v tom případě je možné k němu sestrojit ekvivalentní konečný automat. Jestliže je tento jazyk nekonečný, bude v diagramu tohoto automatu nejméně jedna smyčka. Na obrázku vpravo vidíme diagram, ve kterém je smyčka přes jeden stav ve stavu p_0 (samotná smyčka generuje množinu slov a^*), dále smyčka přes dva stavy p_1, p_2 a navíc smyčka přes všechny tři stavy. I tak jednoduchý jazyk jako například ba^* by měl v diagramu svého automatu smyčku, protože je nekonečný.

Vezměme z takového jazyka „dostatečně dlouhé slovo“. Co je to dostatečně dlouhé slovo? Jednoduše takové, o kterém se dá říct, že není krátké. Nejlepší je zvolit nikoliv konkrétní slovo, ale reprezentaci množiny slov s podobnou strukturou takovou, kde máme v zápisu index – ten nám zajistí „dostatečnou délku slova“. Například ba^i rozhodně představuje množinu, ve které máme i „dostatečně dlouhá slova“, stačí si pod indexem i představit nějaké hodně velké číslo.

Proč potřebujeme *dostatečně dlouhé slovo*? U takového slova máme jistotu, že při jeho vyhodnocování konečným automatem bude nutné přejít minimálně jednou přes některou smyčku. Když víme, že přes některou smyčku určitě půjdeme, můžeme si s ní trochu pohrát – přes tu smyčku půjdeme opakovaně vícekrát, obecně jiný počet krát než v původním slově. Jestliže jsme pořád v tomtéž konečném automatu, pak musí platit, že slovo, které takto zpracujeme (jiný počet průchodů přes smyčku), taky bude patřit do jazyka automatu.

Například z jazyka automatu na obrázku vpravo si můžeme vybrat slovo $a^k b^2$. Pro různé indexy k získáme tyto posloupnosti výpočtu:

$k = 0$: $(p_0, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$
$k = 1$: $(p_0, abb) \vdash (p_0, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$
$k = 2$: $(p_0, aabb) \vdash (p_0, abb) \vdash (p_0, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$
$k = 3$: $(p_0, aaabb) \vdash (p_0, aabb) \vdash (p_0, abb) \vdash (p_0, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$ atd.

Od indexu 1 vede cesta v grafu vždy minimálně jednou přes smyčku ve stavu p_0, přičemž vždy dané slovo patří do jazyka rozpoznávaného automatem. Opakovaným procházením smyčky „pumpujeme“ danou část slova pořád dokola, proto se probírané větě říká Pumping lemma.

Vyzkoušíme jiné slovo pro tentýž jazyk a automat, například $abb(bb)^k$. Je zřejmé, že teď cílíme na smyčku přes stavy p_1, p_2. Pro různé indexy k získáme tyto posloupnosti výpočtu:

$k = 0$: $(p_0, abb) \vdash (p_0, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$
$k = 1$: $(p_0, abbbb) \vdash (p_0, bbbb) \vdash (p_1, bbb) \vdash (p_2, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$
$k = 2$: $(p_0, abbbbbb) \vdash (p_0, bbbbbb) \vdash (p_1, bbbbb) \vdash (p_2, bbbb) \vdash (p_1, bbb) \vdash (p_2, bb) \vdash (p_1, b) \vdash (p_2, \varepsilon)$
atd.

Opět všechna takto „napumpovaná“ slova patří do jazyka generovaného automatem.

Z toho vyplývá, že typickou vlastností všech regulárních jazyků je: pokud najdeme dostatečně dlouhé slovo, můžeme jeho část (tu, která jde přes jakoukoliv smyčku) pumpovat – opakovat – v jakémkoliv počtu kroků (ovšem vždy celou smyčku, nestačí jen její část), a výsledné slovo bude také patřit do daného jazyka.

Zbývá poslední otázka: jak zvolit „dostatečnou délku“ slova? Podívejme se na náš automat na předchozí straně. Stačí, když zvolíme slovo o délce přesahující počet stavů automatu, a už máme

„dostatečnou délku“, protože v tom případě je jisté, že výpočet půjde přes nejméně jeden stav alespoň dvakrát. V našem případě potřebujeme slovo o délce větší než 3.

Lemma 1.6 (Pumping lemma pro regulární jazyky)

Jestliže L je regulární jazyk, pak existuje přirozené číslo $p > 0$ takové, že pro každé slovo $w \in L$ takové, že $|w| > p$, existuje alespoň jedno rozdělení slova w ve formě $w = x \cdot y \cdot z$, kde

$$y \neq \varepsilon \quad \wedge \quad |x \cdot y| \leq p \quad \wedge \quad \forall k \geq 0 \;\; je \;\; x \cdot y^k \cdot z \in L \tag{1.4}$$

V těchto větách si můžeme všimnout, že se ve skutečnosti střídají existenční a obecný kvantifikátor. Uvedeme si totéž lemma v jiné formulaci, přičemž podmínku $y \neq 0$ zapíšeme jako $|y| > 0$:

Lemma 1.7 (Pumping lemma pro regulární jazyky – jiné znění)

$$L \text{ je regulární jazyk} \quad \Rightarrow \quad \exists p \in \mathbb{N},\ p > 0 \text{ takové, že } \forall w \in L,\ |w| > p \ \exists \text{ rozdělení}$$
$$w = x \cdot y \cdot z \quad \wedge \quad |y| > 0 \quad \wedge \quad |x \cdot y| \leq p \quad \wedge \quad \forall k \geq 0 \;\; je \;\; x \cdot y^k \cdot z \in L \tag{1.5}$$

Poznámka:

Všimněme si ještě jedné důležité věci – ve větě se nic neříká ani o automatu, ani o stavech. Je tam prostě podmínka na „nějaké“ číslo p, ale nic víc. Ve větě totiž vůbec žádný automat nepotřebujeme, stačí, když zvolíme dostatečně velké číslo p (ideálně s využitím vhodného indexu). Zmínka o automatu a stavech by dokonce byla škodlivá, protože (jak si později ukážeme) tuto větu používáme v důkazech, že určitý jazyk *není regulární* (a tedy pro něj vlastně ani žádný konečný automat nelze sestrojit).

Jinými slovy – pokud ve *větě* někdo začne u zkoušky mluvit o automatu či stavech, letí...

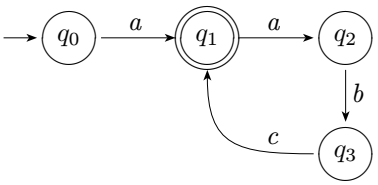
Pumping lemma pro regulární jazyky se dá používat dvěma způsoby:

1. O jazyce L víme, že je regulární, chceme vystihnout strukturu dlouhých slov.
2. Chceme zjistit, zda je jazyk L regulární.

Nejdřív se podíváme na první možnost.

Příklad 1.6

Vezměme jazyk $L = \{a \cdot (abc)^n \ ; \ n \geq 0\}$. Tento jazyk je regulární, protože lze sestrojit ekvivalentní regulární výraz $R = a(abc)^*$, regulární gramatiku a konečný automat, jehož diagram vidíme vpravo. Jazyk L je nekonečný a v diagramu jeho konečného automatu je smyčka přes stavy q_1, q_2, q_3.



Podle Pumping lemma pro regulární jazyky pro tento jazyk existuje $p > 0$ takové, že všechna slova jazyka delší než p dokážeme rozdělit na tři části splňující stanovené podmínky. Můžeme zvolit například $p = 10$, ale lepší je jednoduše se spolehnout na vhodný index a jako základní

slovo zvolit $w = a(abc)^i$ pro dostatečně velký index i (tento předpis mimochodem představuje všechna dlouhá slova jazyka).

Ve větě se píše, že pro každé takové slovo existuje alespoň jedno rozdělení, tedy stačí najít jedno. Můžeme zvolit třeba $w = x \cdot y \cdot z = a \cdot abc \cdot (abc)^{i-1}$, tedy prostřední část $y = abc$.

Ověříme, zda pumpování pro toto rozdělení funguje – označme $w_k = x \cdot y^k \cdot z$:

- $w_0 = a \cdot (abc)^0 \cdot (abc)^{i-1} = a(abc)^{i-1} \in L$
- $w_1 = a \cdot (abc)^1 \cdot (abc)^{i-1} = a(abc)^i \in L$
- $w_2 = a \cdot (abc)^2 \cdot (abc)^{i-1} = a(abc)^{i+1} \in L$
- $w_3 = a \cdot (abc)^3 \cdot (abc)^{i-1} = a(abc)^{i+2} \in L$ atd.

Rozdělení jsme zvolili dobře, pumpované slovo vždy patří do jazyka L.

Pokud bychom zvolili špatné rozdělení (zde například $w = aa \cdot bc \cdot (abc)^{i-1}$), po pumpování bychom získali slova nepatřící do daného jazyka. Ve větě se totiž nepíše, že důsledek platí pro všechna možná rozdělení, ale pro nejméně jedno možné rozdělení.

Možných správných rozdělení ovšem může být víc než jedno, zde bychom například jako prostřední část mohli zvolit třeba $(abc)^2$. Taktéž pokud najdeme víc různých smyček, přidávají se další možnosti rozdělení.

Poznámka:

Uvědomme si, že Pumping lemma má formu *implikace*, nikoliv *ekvivalence*. Říká:

$$\text{jazyk je regulární} \quad \Rightarrow \quad \text{jazyk splňuje danou vlastnost}$$

Proč je to tak důležité? Existují totiž jazyky, které sice nejsou regulární, ale danou vlastnost mají také, a v jejich případě tedy obrácené tvrzení neplatí. Například jazyk $L = \{(ab)^m a^n c^n \ ; \ m, n \geq 1\}$ není regulární (vyžaduje synchronizaci dvou částí slova, což konečný automat nedokáže), ale přesto existuje $p > 0$ takové, že pro každé dostatečně dlouhé slovo jazyka dokážeme najít rozdělení pro pumpování, například $w = \varepsilon \cdot ab \cdot (ab)^{i-1} a^j c^j$.

Užitečnějším využitím Pumping lemma je druhý uvedený způsob – důkaz, že daný jazyk není regulární. Zde využíváme nepřímý důkaz. Původní implikaci

$$\text{jazyk je regulární} \quad \Rightarrow \quad \text{jazyk splňuje danou vlastnost}$$

ekvivalentně upravíme, čímž vznikne tvrzení

$$\begin{aligned} \neg(\text{jazyk splňuje danou vlastnost}) \quad &\Rightarrow \quad \neg(\text{jazyk je regulární}) \\ \text{jazyk nesplňuje danou vlastnost} \quad &\Rightarrow \quad \text{jazyk není regulární} \end{aligned}$$

Poznámka:

Proč to můžeme provést? Vzpomeňte si na ekvivalentní úpravy logických výrazů:

$$(A \to B) \quad \Longleftrightarrow \quad (\neg B \to \neg A)$$

Je třeba si uvědomit, jaký vliv má negace na kvantifikátory. Srovnejte původní tvar podmínky ve větě a tvar po převrácení:

$$L \in \mathscr{L}(REG) \Rightarrow \quad \exists p > 0 \;\; \forall w\text{: } |w| > p \;\; \exists \text{ rozdělení} \ldots \;\; \forall k \geq 0\text{: } x \cdot y^k \cdot z \in L$$

$$\forall p > 0 \;\; \exists w\text{: } |w| > p \;\; \forall \text{ rozdělení} \ldots \;\; \exists k \geq 0\text{: } x \cdot y^k \cdot z \notin L \;\; \Rightarrow L \notin \mathscr{L}(REG)$$

Postup

Pokud tedy pomocí této věty chceme dokázat, že jazyk L není regulární, postupujeme takto:

- vybereme dostatečně dlouhé slovo $w \in L$,
- stanovíme strukturu tohoto slova a určíme možná rozdělení vyhovující podmínkám $w = x \cdot y \cdot z \quad \wedge \quad y \neq \varepsilon \quad \wedge \quad |x \cdot y| \leq p$,
- ukážeme, že žádné z těchto rozdělení neodpovídá podmínce $\forall k \geq 0$ je $x \cdot y^k \cdot z \in L$, tedy pro každé rozdělení najdeme číslo k takové, že $x \cdot y^k \cdot z \notin L$ (obvykle stačí vyzkoušet $k = 0$ nebo $k = 2$),
- pokud se to nepodaří, vracíme se k prvnímu bodu a hledáme další dostatečně dlouhé slovo.

Role čísla p je ve větě dvojí – předně nám říká, že slovo má být opravdu dostatečně dlouhé, a dále ve vztahu $|x \cdot y| \leq p$ stanovuje horní omezení délky prvních dvou částí slova po rozdělení. Omezení je zde proto, abychom v daném slově zvolili vždy první „smyčku“ zleva, má to být první možnost opakování, na kterou ve slově narazíme. Z našeho pohledu je důsledek takový, že pokud při specifikaci dostatečně dlouhého slova použijeme písmenný index, pak by se tento index neměl v prvních dvou částech vůbec vyskytovat.

Příklad 1.7

Ukážeme pomocí Pumping lemma, že jazyk $L = \{a^n b^n \;;\; n \geq 0\}$ není regulární.

Jako dostatečně dlouhé slovo jazyka si zvolíme $w = a^i b^i$ pro (nekonkrétní) velké číslo i. Dále stanovíme různá možná rozdělení tohoto slova a ke každému zjistíme, jestli se dá pumpovat. Rozdělení $w = x \cdot y \cdot z$ má být takové, že prostřední část musí obsahovat alespoň jeden symbol ($y \neq \varepsilon$) a první dvě části mají shora omezenou délku (tj. nemůžeme v nich použít index i).

Postupujeme tak, že určujeme místo ve slově, ve kterém by mohla být prostřední (tedy hraniční) část slova. Jsou tyto možnosti:

- prostřední část y bude na začátku první části slova,
- prostřední část y bude v první části slova, ale ne zcela na začátku.

Další možnosti odpadají, protože musí platit $|x \cdot y| < p$.

Nejdřív tedy prověříme možnost $x = \varepsilon,\; y = a^m,\; z = a^{i-m}b^i$, kde $m > 0$ je číslo $m < p$ (nebudeme zvlášť prověřovat $m = 1,\; m = 2,\; \ldots$, důkaz by se poněkud protáhl). Pak je tedy
$w = \varepsilon \cdot a^m \cdot a^{i-m}b^i$,
po pumpování $w_k = \varepsilon \cdot a^{m \cdot k} \cdot a^{i-m}b^i = a^{m \cdot k + i - m}b^i = a^{m(k-1)+i}b^i$.
Například pro $k = 0$ dostáváme $w_0 = a^{i-m}b^i$, přičemž $m > 0$. Je zřejmé, že $w_0 \notin L$. K témuž závěru bychom došli, kdybychom zvolili jakékoliv číslo $k \neq 1$, dobře se pracuje například s $k = 2$.

Druhá možnost nám dává $x = a^r,\ y = a^s,\ z = a^{i-r-s}b^i$, kde $r, s > 0,\ r + s < p$. Pak platí $w = a^r \cdot a^s \cdot a^{i-r-s}b^i$,
po pumpování $w_k = a^r \cdot a^{s\cdot k} \cdot a^{i-r-s}b^i = a^{r+s\cdot k+i-r-s}b^i = a^{s\cdot k+i-s}b^i = a^{s(k-1)+i}b^i$
Opět můžeme zvolit $k = 0$, dostáváme $w_0 = a^{i-s}b^i \notin L$, protože $s > 0$.

Všimněte si, že první možnost (rozdělení slova w) je vlastně speciálním případem druhé možnosti pro $r = 0$. Tento důkaz by tedy bylo možné ještě zkrátit.

Poznámka:

Pumping lemma by mělo platit pro všechny regulární jazyky. Platí i pro konečné jazyky? Víme, že každý konečný jazyk patří do třídy regulárních jazyků a lze pro něj vytvořit ekvivalentní konečný automat, takže by věta měla platit i pro konečné jazyky.

Podívejme se na znění věty – „pro každé slovo $w \in L$ takové, že $|w| > p \ldots$“ Pokud číslo p položíme rovno počtu stavů automatu, pak v konečném jazyce L neexistuje žádné slovo delší než p (samozřejmě – jinak by v diagramu automatu musela být smyčka a jazyk by nebyl konečný), proto množina slov delších než p je prázdná. Jenže v tom případě můžeme tvrzení „pro každé… existuje…“ považovat za platné, protože pro prvky prázdné množiny vlastně platí jakákoliv podmínka – klidně můžeme každé slovo množiny rozkládat na tři části atd., protože v reálu to provedeme $0\times$ (není co testovat).

Důkaz (Pumping lemma pro regulární jazyky): Tato věta je implikace, tedy důkaz povedeme pouze jedním směrem.

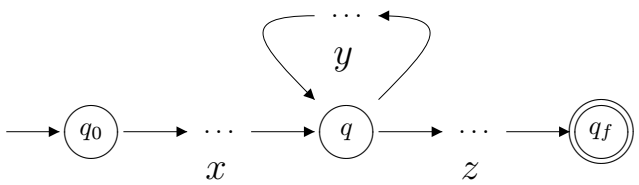
Předpokládejme, že jazyk L je rozpoznáván konečným automatem $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. Máme dokázat, že pro takový jazyk existuje číslo p takové, že každé slovo delší než p lze rozdělit na tři části tak, jak je ve větě popsáno.

Máme dokázat, že takové číslo existuje, proto si můžeme zvolit – volíme $p = \mathrm{card}(Q)$, tedy počet prvků množiny stavů.

Nechť $M = \{w \in L\ ;\ |w| > p\}$ je množina všech slov jazyka L delších než p. Pokud je tato množina prázdná, tvrzení věty platí, jak jsme si přečetli v poznámce před tímto důkazem. Dále tedy předpokládejme, že tato množina není prázdná.

Tvrzení věty má platit pro všechny prvky množiny M, tedy dále budeme pracovat s (jakýmkoliv) slovem $w \in M$, označme $|w| = n$. Protože $w \in L$, musí v automatu $\mathcal{A}$ existovat akceptující výpočet tohoto slova, a to postupně přes $n+1$ stavů. Poněvadž $n > p$ a p je počet stavů automatu, musí existovat minimálně jeden stav (ve skutečnosti minimálně dva stavy) takový, který se v této posloupnosti stavů vyskytuje více než jednou.

Označme $q \in Q$ první takový stav posloupnosti výpočtu, který se v posloupnosti vyskytuje více než jednou. Pak můžeme naši posloupnost stavů rozdělit na tři části, jak je naznačeno na obrázku vpravo – první úsek označený x povede od počátečního stavu q_0 k prv-



nímu výskytu stavu q v posloupnosti výpočtu, pak pokračujeme smyčkou k druhému výskytu q v posloupnosti výpočtu (to je druhý úsek označený y) a zbytek slova je zpracován v třetím úseku označeném z (na nějž neklademe žádné požadavky). Výpočet tedy můžeme zapsat takto:

$$(q_0, x \cdot y \cdot z) \vdash^* (q, y \cdot z) \vdash^+ (q, z) \vdash^* (q_f, \varepsilon)$$

Výpočet slova $w_k = x \cdot y^k \cdot z$ pro všechna $k \geq 0$ bude pak následující:

$$(q_0, x \cdot y^k \cdot z) \vdash^* (q, y^k \cdot z) \quad \underbrace{\vdash^+ \quad \ldots \quad \vdash^+}_{k\text{-krát zpracujeme } y} \quad (q, z) \vdash^* (q_f, \varepsilon)$$

Tím jsme dokázali, že pro jakékoliv slovo $w \in L$ delší než p lze najít rozdělení toho slova $w = x \cdot y \cdot z$ takové, že $w_k = x \cdot y^k \cdot z \in L$ (protože pokud pro slovo w_k dokážeme sestrojit posloupnost výpočtu, pak toto slovo patří do jazyka L). □

1.3.2 Využití uzávěrových vlastností

Třída regulárních jazyků (resp. jazyků rozpoznávaných konečnými automaty) je uzavřena vzhledem mnoha různým operacím, čehož lze využít i v důkazech (ne)regulárnosti. Nejužitečnější operací je v tomto smyslu průnik.

Příklad 1.8

Dokážeme, že jazyk $L = \{w \in \{a,b\}^* \ ; \ |w|_a = |w|_b\}$ není regulární. Tento jazyk obsahuje všechna slova nad abecedou $\{a,b\}^*$ taková, která obsahují stejný počet symbolů a jako b.

Důkaz povedeme nepřímo – budeme předpokládat, že L je regulární, a postupně dojdeme ke sporu.

Jestliže tedy je jazyk L regulární, pak by jeho průnik s jakýmkoliv jiným regulárním jazykem byl taky regulární jazyk (protože třída regulárních jazyků je uzavřena vzhledem k operaci průniku). Víme, že jazyk $R = \{a^i b^j \ ; \ i,j \geq 0\} = a^*b^*$ je regulární. Jazyk $L \cap R$ vypadá takto:

$$L \cap R = \{w \in \{a,b\}^* \ ; \ |w|_a = |w|_b\} \cap \{a^i b^j \ ; \ i,j \geq 0\} = \{a^i b^i \ ; \ i \geq 0\}$$

Jenže jazyk $L \cap R$ není regulární (to jsme na předchozích stránkách dokázali pomocí Pumping lemma). Došli jsme ke sporu a tedy jazyk L není regulární.

1.3.3 Nerodova věta

V této sekci budeme potřebovat, aby konečné automaty, se kterými budeme pracovat, byly deterministické a měly redukovanou množinu stavů – především odstraněné nedosažitelné stavy (ideálně i nadbytečné). Pro připomenutí: nedosažitelné stavy se odstraňují tak, že rekurzivním algoritmem zjistíme, které stavy jsou dosažitelné:

$$\begin{aligned} S_0 &= \{q_0\} \\ S_{i+1} &= S_i \cup \{q \in Q \ ; \ \exists p \in S_i, \ a \in \Sigma: \ \delta(p,a) = q\}, \quad i \geq 0 \end{aligned}$$

A teď trochu algebry – budeme se zabývat ekvivalencemi a kongruencemi. Víme, že ekvivalence je taková relace, která splňuje tyto tři vlastnosti: je reflexivní, symetrická a tranzitivní.

Definice 1.7 (Rozklad na třídy ekvivalence)

Relace ekvivalence $\sim$ *je binární relace na dané množině* M*, která množinu* M *dělí na vzájemně disjunktní podmnožiny. Tyto podmnožiny nazýváme* třídy ekvivalence *a proces jejich vytvoření je rozklad množiny na třídy ekvivalence.*

Třídu ekvivalence $\sim$ *na množině* M *značíme podle kteréhokoliv prvku této třídy:* $[a]_\sim$*, kde* $a \in M$*. Platí* $[a]_\sim = \{b \in M \ ; \ b \sim a\}$*. Faktorovou množinu, tedy rozklad množiny* M *podle ekvivalence* $\sim$*, značíme* $M/\sim$ *a platí* $M/\sim = \bigcup_{a \in M} [a]_\sim$*.*

Příklad 1.9

Aby bylo jasné, jak se používá rozklad na třídy ekvivalence, ukážeme jej na této ekvivalenci:

Nechť $\diamond$ je relace na množině přirozených čísel s nulou $\mathbb{N}_0$ taková, že

$\forall a, b \in \mathbb{N}$: $a \diamond b \ \Leftrightarrow \ (a \bmod 5) = (b \bmod 5)$

Provádíme tedy rozklad množiny na třídy ekvivalence podle zbytku po dělení číslem 5. Jednotlivé třídy vypadají takto:

- $[0]_\diamond = \{0, 5, 10, 15, \ldots\}$
- $[1]_\diamond = \{1, 6, 11, 16, \ldots\}$
- $[2]_\diamond = \{2, 7, 12, 17, \ldots\}$
- $[3]_\diamond = \{3, 8, 13, 18, \ldots\}$
- $[4]_\diamond = \{4, 9, 14, 19, \ldots\}$

Jedná se o ekvivalenci – je reflexivní (každý prvek je ekvivalentní sám se sebou), symetrická (pro každé dva prvky platí $a \diamond b \Leftrightarrow b \diamond a$) a tranzitivní (pro jakékoliv tři prvky platí $(a \diamond b \diamond c) \Rightarrow (a \diamond c)$).

Faktorizací jsme vytvořili celkem pět tříd rozkladu. Tyto třídy jsou navzájem disjunktní (nenajdeme žádný prvek, který by patřil do více než jedné třídy) a zároveň jejich sjednocením je celá původní množina: $\mathbb{N}_0 = [0]_\diamond \cup [1]_\diamond \cup [2]_\diamond \cup [3]_\diamond \cup [4]_\diamond$

Definice 1.8 (Index ekvivalence)

Index ekvivalence $\sim$ *definované na množině* M *je počet tříd rozkladu* $M/\sim$*. Pokud je počet tříd rozkladu nekonečný, je indexem ekvivalence* ∞*.*

Index ekvivalence $\diamond$ z předchozího příkladu je 5, protože rozkladem jsme získali pět tříd rozkladu.

V příkladu jsme si ukázali relaci ekvivalence definovanou na množině přirozených čísel s nulou, dále se zaměříme na ekvivalenci definovanou na množině Σ^* řetězců nad danou abecedou.

Definice 1.9 (Pravá kongruence)

Nechť Σ *je abeceda a nechť* $\sim$ *je relace ekvivalence na množině* Σ^**. Říkáme, že relace* $\sim$ *je* pravou kongruencí *(zprava invariantní), pokud pro každé* $u, v, w \in \Sigma^*$ *platí* $u \sim v \ \Rightarrow \ u \cdot w \sim v \cdot w$*.*

Poznámka:

Všimněte si, že vlastně vůbec neodbočujeme od konečných automatů – v definici pravé kongruence vidíme, že k prvkům (řetězcům) přidáváme zprava další prvky, a co asi provádí konečný automat během výpočtu slova? Postupně zprava přidává další a další rozpoznané symboly.

Lemma 1.8

Nechť $\sim$ je relace ekvivalence na množině Σ^*. Relace $\sim$ je pravou kongruencí právě tehdy, když $\forall u, v \in \Sigma^*,\ a \in \Sigma$ platí $u \sim v \ \Rightarrow\ u \cdot a \sim v \cdot a$.

Důkaz: Jedná se vlastně o variaci předchozí definice, kdy zprava přidáváme místo řetězce právě jeden symbol. Důkaz jedním směrem (definice $\rightarrow$ věta) je triviální, protože tvrzení lemmatu je vlastně okleštěním tvrzení z definice. Důkaz druhým směrem (věta $\rightarrow$ definice) lze provést matematickou indukcí dle délky slova w:

Báze: $|w| = 0$: triviální – $u \sim v \ \Rightarrow\ u \cdot \varepsilon \sim v \cdot \varepsilon$

Předpoklad: Předpokládejme, že vztah platí pro $w \in \Sigma^*$ takové, že $0 \leq |w| \leq n$. Zjistíme, zda vztah platí i pro $w' = w \cdot a,\ a \in \Sigma$, tedy $|w'| = n + 1$.

Krok indukce: Jestliže podle předpokladu $u \sim v \ \Rightarrow\ u \cdot w \sim v \cdot w$, kde $|w| = n$, pak můžeme využít tranzitivitu a tvrzení z lemmatu: $u \cdot w \sim v \cdot w \ \Rightarrow\ u \cdot w \cdot a \sim v \cdot w \cdot a = u \cdot w' \sim v \cdot w'$. □

Věta 1.9 (Nerodova věta)

Nechť L je jazyk nad abecedou L. Pak tato tvrzení jsou ekvivalentní:

1. *L je rozpoznatelný konečným automatem (tj. je to regulární jazyk).*
2. *L je sjednocením některých tříd rozkladu určeného pravou kongruencí $\sim_L$ na Σ^* s konečným indexem.*

Autorem Nerodovy věty je Anil Nerode. Ukážeme si, jakým způsobem je možné větu pro testování regulárnosti použít.

Příklad 1.10

Vezměme jazyk $L = \{a^n b^n\ ;\ n \geq 0\}$. Pomocí Pumping lemma jsme dokázali, že tento jazyk není regulární, teď provedeme důkaz téhož tvrzení pomocí Nerodovy věty.

Předně určíme rozklad tříd ekvivalence vzhledem k abecedě jazyka, která je $\Sigma = \{a, b\}$. Dotyčný rozklad je $\{a, b\}^*/\sim_L$, a kdyby jazyk L byl regulární, pak by podle Nerodovy věty byl sjednocením některých tříd rozkladu určeného pravou kongruencí s konečným indexem. Tento index pracovně označíme k (tj. existuje právě k tříd rozkladu pro ekvivalenci $\sim_L$), tuto konstantu budeme používat i v další části důkazu. Relaci $\sim_L$ nemusíme pro účely důkazu přímo určovat, stačí vědět, že se jedná o pravou kongruenci.

To, že jazyk L není regulární, tedy dokážeme jednoduše tak, že najdeme dvě slova $u, v \in \Sigma^*$ taková, že sice patří do stejné třídy rozkladu, ale jedno z nich patří do L a druhé ne.

Vezmeme slova $ab, a^2b, \ldots, a^kb, a^{k+1}b \in \Sigma^*$. Protože rozklad $\Sigma^*/\sim_L$ má k tříd, budou minimálně dvě z těchto $k+1$ slov v téže třídě – označme je a^ib, a^jb pro nějaké $1 \leq i < j \leq k+1$, tedy $a^ib \sim_L a^jb$. Protože $\sim_L$ je zprava invariantní, mělo by platit $a^ib \cdot b^{i-1} \sim_L a^jb \cdot b^{i-1}$, po zřetězení $a^ib^i \sim_L a^jb^i$, ale zároveň platí $i < j$, tedy $i \neq j$. Proto $a^ib^i \in L$, kdežto $a^jb^i \notin L$.

Z toho vyplývá, že jazyk L není regulární (našli jsme dvě slova, která patří do stejné třídy ekvivalence, ale jedno z nich patří do jazyka L a druhé ne).

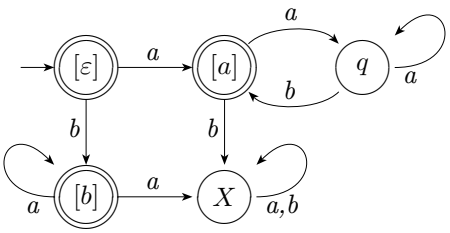
Nerodova věta určuje tvrzení, které je ekvivalencí, stanovuje tedy nutnou a postačující podmínku pro regulárnost jazyka. Jinými slovy – podle toho, zda je či není splněna podmínka 2 věty, můžeme přímo určit, zda daný jazyk je či není regulární. Oproti tomu Pumping lemma neurčuje ekvivalenci (je pouze implikací), tedy jde o nutnou podmínku, nikoliv postačující.

Příklad 1.11

Použijeme opět abecedu $\Sigma = \{a, b\}$. Definujeme ekvivalenci $\sim_L$ na množině Σ přímo určením množin rozkladu:

- $[\varepsilon] = \{\varepsilon\}$
- $[a]$ – do této třídy zařadíme slova odpovídající výrazu $a(aa^*b)^*$
- $[b]$ – do této třídy zařadíme slova odpovídající výrazu ba^*
- X – všechna ostatní slova nad danou abecedou.

Vzhledem k poslední uvedené třídě můžeme tvrdit, že sjednocením všech tříd získáme původní množinu Σ^*, a zároveň jsou všechny vytvořené třídy po dvou navzájem disjunktní (každé slovo nad abecedou Σ patří právě do jedné třídy).



Ověříme, zda se jedná o pravou kongruenci. Nejjednodušší to bude tím, že sestrojíme konečný automat, jehož jednotlivé koncové stavy budou odpovídat prvním třem třídám rozkladu a dále jeden nekoncový bude představovat poslední třídu. Na obrázku vpravo vidíme diagram takto sestrojeného automatu. Tento automat je deterministický a totální (díky stavu X, který plní roli „odpadkového koše“), rozpoznávaný jazyk je $L = \varepsilon + a(aa^*b)^* + ba^*$. Díky determinismu je zpracování jakéhokoliv slova nad abecedou Σ ukončeno právě v jednom stavu (koncovém, pokud slovo patří do jazyka L).

Jak vidíme, Nerodova věta vlastně vystihuje základní strukturální charakteristiku regulárních jazyků, vychází z toho, že regulární jazyk lze reprezentovat sjednocením konečného počtu jednodušších regulárních výrazů. Každá třída rozkladu pak odpovídá regulárnímu výrazu, jehož vyhodnocení v konečném automatu je ukončeno v jednom konkrétním koncovém stavu. Takovou třídu rozkladu můžeme reprezentovat buď některým ze slov patřících do dané třídy, nebo označením příslušného koncového stavu.

Účelem vytvoření následující definice je možnost vytvoření přehlednějšího zápisu výpočtu slova v konečném automatu, zápis využijeme také v důkazu Nerodovy věty.

Definice 1.10 (Rozšířená přechodová funkce)

Nechť $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Rozšířená přechodová funkce v tomto automatu je parciální funkce δ^*: $Q \times \Sigma^* \to Q$, kde

$$\delta^*(q, \varepsilon) = q \tag{1.6}$$

$$\delta^*(q, w \cdot a) = \delta(\delta^*(q, w), a) \tag{1.7}$$

pokud přechody na pravé straně předpisu jsou definovány (pokud je $\mathcal{A}$ totální automat, pak jsou vždy definovány).

Takto definovaná funkce nám zkrátí zápis, například $\delta^*(q, w) = r$ v deterministickém automatu znamená, že výpočet slova w začínající ve stavu q skončí ve stavu r (po tolika krocích, jaká je délka slova w). Takže jazyk rozpoznávaný automatem $\mathcal{A}$ můžeme zapsat takto:

$$L(\mathcal{A}) = \{w \in \Sigma^* \ ; \ \delta^*(q_0, w) \in F\} \tag{1.8}$$

Důkaz (Nerodova věta): Jedná se o ekvivalenci dvou tvrzení, tedy musíme dokázat dvě k sobě opačné implikace. Tvrzení *(1)* říká, že jazyk L je regulární, tvrzení *(2)* určuje, že L je sjednocením některých tříd rozkladu určeného pravou kongruencí $\sim_L$ na Σ^* s konečným indexem.

(1) $\Rightarrow$ (2): Nechť je jazyk L je regulární, tedy lze sestrojit ekvivalentní deterministický totální konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ bez nedostupných stavů.

Definujeme relaci $\sim_L$ na množině Σ^* předpisem $\forall x, y \in \Sigma^*$: $x \sim_L y \Leftrightarrow_{\text{def}} \delta^*(q_0, x) = \delta^*(q_0, y)$ – tj. dvě slova nad danou abecedou jsou ekvivalentní právě tehdy, když jejich výpočet v automatu $\mathcal{A}$ končí ve stejném stavu (pozor, není řečeno, že končí úspěšně, dotyčný stav nemusí být koncový).

Je zřejmé, že relace ekvivalence $\sim_L$ má konečný index, protože tříd rozkladu je maximálně tolik, kolik je stavů v automatu – $\text{card}(Q)$, přičemž množina Q je konečná. Je to pravá kongruence, což plyne z definice rozšířené přechodové funkce δ^*.

Jazyk L je sjednocením některých tříd rozkladu určeného pravou kongruencí $\sim_L$ na Σ^*, protože může být zapsán následovně:

$$L = \bigcup_{q \in F} \{w \in \Sigma^* \ ; \ \delta^*(q_0, w) = q\} \tag{1.9}$$

(2) $\Rightarrow$ (1): Nechť jazyk L je sjednocením některých tříd rozkladu určeného pravou kongruencí $\sim_L$ na Σ^* s konečným indexem. Jednotlivé třídy označíme $[u]$, kde $u \in [u]$ je některým prvkem třídy $[u]$.

Sestrojíme konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ takto:

- $Q = \Sigma^*/\sim_L$ – stavy automatu budou jednotlivé třídy rozkladu; protože ekvivalence $\sim_L$ má konečný index, také množina Q bude konečná,

- funkci δ určíme jako funkci přechodu mezi třídami: $\delta([u], a) = [ua]$; motivace byla ukázána v příkladu výše, a protože je $\sim_L$ pravou kongruencí, nezávisí náš zápis na volbě reprezentantů třídy,
- $q_0 = [\varepsilon]$,
- F obsahuje právě stavy vzniklé ze tříd rozkladu, jejichž sjednocení dá jazyk L.

Pro jakékoliv slovo $w \in \Sigma^*$ lze indukcí podle délky slova ukázat, že $\delta^*([\varepsilon], w) = [w]$, a platí

$$\forall w \in \Sigma^* \;\Leftrightarrow\; w \in L \;\Leftrightarrow\; [w] \in F \;\Leftrightarrow\; \delta^*([\varepsilon], w) \in F \tag{1.10}$$

Proto můžeme tvrdit, že $L(\mathcal{A}) = L$. □

1.4 Minimalizace konečného automatu

Z minulého semestru víme, jak vytvořit redukovaný automat, tedy redukovat množinu stavů automatu. Ovšem nedá se tvrdit, že takto vytvořený automat je pro daný jazyk nejmenší možný (co se týče množství stavů). Může totiž existovat ekvivalentní automat s ještě menším množstvím stavů (například může být možné ještě dál pracovat s některými cykly či sdružovat konce cest v automatu, čehož redukcí nedosáhneme). Zatímco při redukci pracujeme pouze se „syntaxí“ (grafem, kdy nebereme v úvahu ohodnocení hran), nyní se soustředíme i na *sémantiku* (graf včetně ohodnocení).

Účelem minimalizace nemusí být nutně minimalizace samotná, může být pro nás pouze prostředkem k jinému účelu – například pokud chceme zjistit, zda dva (na pohled různé) konečné automaty rozpoznávají tentýž jazyk, pak oba automaty minimalizujeme a pak jednoduše porovnáme podle stavů.

Následující definice by pro nás měla být triviální, protože pod pojmem ekvivalence automatů jsme i dřív rozuměli jejich jazykovou ekvivalenci:

Definice 1.11 (Jazykově ekvivalentní konečné automaty)

Dva automaty $\mathcal{A}_1$ a $\mathcal{A}_2$ jsou jazykově ekvivalentní, jestliže $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

Nyní můžeme definovat minimální automat:

Definice 1.12 (Minimální automat)

Konečný automat $\mathcal{A}$ je minimální (minimalizovaný), jestliže neexistuje žádný jiný konečný automat $\mathcal{A}'$ s menším množstvím stavů, pro který by platilo $L(\mathcal{A}) = L(\mathcal{A}')$.

Jak lze provést minimalizaci? Algoritmus je založen na myšlence, že pokud cesty vedoucí ze dvou různých stavů (do koncových stavů) určují ekvivalentní množiny rozpoznávaných slov, pak jsou tyto dva stavy zaměnitelné a jeden z nich můžeme odstranit (tedy takové dva stavy ztotožníme/shrneme do jediného). Je zřejmé, že nemůžeme například ztotožnit dva stavy takové, že jeden z nich je koncový a druhý ne. A pokud dva stavy ztotožníme (shrneme), musíme ztotožnit i jejich následníky na cestě grafem diagramu automatu.

Definice 1.13 (Ekvivalence stavů automatu)

Nechť $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Definujeme množiny

$$\forall q \in Q\colon\ L(\mathcal{A}_q) = \{w \in \Sigma^*\ ;\ \delta^*(q_0, w) \in F\} \tag{1.11}$$

jako jazyky pomocných konečných automatů $\mathcal{A}_q = (Q, \Sigma, \delta, q, F)$, kde různé stavy použijeme jako počáteční, s využitím rozšířené přechodové funkce, jak je naznačeno na straně 23.

Stavy $r, s \in Q$ jsou (jazykově) ekvivalentní – zapisujeme $r \equiv s$, jestliže $L(\mathcal{A}_r) = L(\mathcal{A}_s)$, tedy

$$\forall r, s \in Q\colon\quad r \equiv s \quad\Leftrightarrow\quad \forall w \in \Sigma^*\colon\ (\delta^*(r, w) \in F \ \Leftrightarrow\ \delta^*(s, w) \in F) \tag{1.12}$$

Už je zřejmě každému jasné, že proces minimalizace konečného automatu bude spočívat v slučování ekvivalentních stavů. Všimněte si, že vzhledem k platnosti Nerodovy věty můžeme využívat rozklad tříd ekvivalence řetězců vztažený na stavy automatu – definovali jsme konkrétní relaci ekvivalence $\equiv$, kde v označení tříd použijeme přímo ty stavy automatu, v nichž skončí výpočet kteréhokoliv slova z dané třídy.

Definice 1.14 (Podílový automat)

Nechť $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je deterministický totální konečný automat bez nedosažitelných stavů. Podílovým automatem automatu $\mathcal{A}$ je konečný automat $\mathcal{A}/_{\equiv} = (Q/_{\equiv}, \Sigma, \eta, [q_0], F/_{\equiv})$, kde

- *množina stavů $Q/_{\equiv}$ obsahuje třídy rozkladu $[q],\ q \in Q$,*
- *přechodovou funkci definujeme pomocí reprezentantů tříd jako nejmenší funkci takovou, že splňuje vztah*
  $\forall r, s \in Q,\ \forall a \in \Sigma\colon\quad \delta(r, a) = s \quad\Rightarrow\quad \eta([r], a) = [s]$

Mechanismus vytvoření počátečního a koncových stavů je zřejmý, jedná se o třídy rozkladu.

Věta 1.10 (Minimalizace konečného automatu)

Nechť $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je deterministický totální konečný automat bez nedosažitelných stavů a $\mathcal{A}/_{\equiv} = (Q/_{\equiv}, \Sigma, \eta, [q_0], F/_{\equiv})$ jeho podílový automat. Pak $L = L(\mathcal{A}) = L(\mathcal{A}/_{\equiv})$ a $\mathcal{A}/_{\equiv}$ je minimální konečný automat rozpoznávající jazyk L.

Pro vysvětlení postupu minimalizace budeme potřebovat ještě následující:

Definice 1.15

Nechť $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat bez nedosažitelných stavů s totální přechodovou funkcí. Pro každé $i \in \mathbb{N}_0$ stanovíme relaci $\equiv_i$ na množině Q takovou, že

$$\forall p, q \in Q\colon\quad p \equiv_i q \quad\overset{def}{\Longleftrightarrow}\quad \forall w \in \Sigma^*,\ |w| \le i\colon\ \Big(\delta^*(p, w) \in F \ \Leftrightarrow\ \delta^*(q, w) \in F\Big) \tag{1.13}$$

Podle definice platí $p \equiv_i q$ tehdy, když stavy p, q nejsou rozlišitelné ve smyslu ekvivalence stavů pro žádné slovo o délce maximálně i. Zřejmě platí $\quad p \equiv q \quad\Longleftrightarrow\quad \forall i \in \mathbb{N}_0\colon\ p \equiv_i q$.

Na následujícím lemmatu je založen návod na vytvoření minimálního automatu:

Lemma 1.11

Pro konečný deterministický totální automat bez nedosažitelných stavů $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *a výše definovanou relaci* $\equiv_i$ *platí:*

- $\equiv_0 = \{(p,q)\ ;\ p \in F \ \Leftrightarrow\ q \in F\}$
- $\equiv_{i+1} = \{(p,q)\ ;\ p \equiv_i q\ \wedge\ \forall a \in \Sigma\colon\ \delta(p,a) \equiv_i \delta(q,a)\}$

Takže postup minimalizace spočívá ve vyhledání skupin takových stavů, které jsou ekvivalentní, a z lemmatu vyplývá, že tento postup bude iterativní – začneme rozdělením stavů na koncové a nekoncové, a dále budeme členění zjemňovat: půjdeme po cestách v grafu automatu „proti směru“ a budeme oddělovat do různých skupin ty stavy (původně ve společné skupině), které se těmito dosud probranými cestami v dalším kroku liší. Končíme po maximálně tolika krocích, kolik máme stavů, na konci jsou ve společné skupině právě ty stavy, které jsou ekvivalentní a tedy patří do stejné třídy rozkladu.

Postup (Minimalizace konečného automatu)

Je dán deterministický totální konečný automat bez nedosažitelných stavů $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. Pro každý stav $q \in Q$ je třeba vytvořit jazyk $L(\mathcal{A}_q) = \{w \in \Sigma^*\ ;\ \delta^*(q,w) \in F\}$, abychom mohli tyto jazyky pro jednotlivé stavy porovnat a zjistit, které z nich jsou ekvivalentní. Pokud zjistíme, že některé dva stavy jsou ekvivalentní, shrneme je.

Mohli bychom samozřejmě doopravdy ke každému stavu určit jazyk $L(\mathcal{A}_q)$, ale pro naše účely to je zbytečná komplikace. Stačí umět rozlišit, zda stavy jsou či nejsou ekvivalentní, bez nutnosti plného vyčíslování jazyka.

Takže je třeba určit jednotlivé třídy rozkladu podle ekvivalence $\equiv$, třídy pracovně označíme římskými číslicemi. Algoritmus bude iterativní, dané kroky budeme provádět tak dlouho, dokud ještě bude možné provádět změny. Jednotlivé kroky odpovídají sestrojení tříd $\equiv_0, \equiv_1, \equiv_2, \ldots$

1. V prvním kroku (bázi) rozdělíme stavy do dvou skupin – první skupina obsahuje nekoncové stavy: $\mathrm{I} = Q - F$, druhá skupina koncové stavy: $\mathrm{II} = F$. Podle přechodové tabulky sestrojíme pomocnou tabulku skupin, do jejíchž buněk vždy místo cílového stavu zapíšeme označení skupiny, ve které ten stav momentálně je. Dalším krokem začíná rekurze.
2. Každou skupinu rozdělíme na dílčí skupiny tak, aby se ze všech stavů v dílčí skupině přecházelo do téže skupiny, tedy v rámci každé skupiny:
   - srovnáme buňky v řádcích různých stavů skupiny,
   - stavy, jejichž řádky jsou různé (až na označení řádku), oddělíme do různých dílčích skupin, nazveme je dalšími (zatím volnými) římskými číslicemi,

   tedy z jedné skupiny vytvoříme více nových skupin,
3. protože jsme vytvořili nové skupiny a přerozdělili stavy, upravíme podle momentálního stavu obsah buněk (k tomu potřebujeme jak tabulku skupin, tak i původní tabulku přechodů).

Postup končí tehdy, když už nelze provádět žádné změny, po maximálně $\mathrm{card}(Q)$ krocích.

Formálně se postup dá zapsat takto:

Algoritmus 1: Minimalizace konečného automatu

Vstup : $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat bez nedosažitelných stavů s totální přechodovou funkcí

Výstup: $\mathcal{A}/_{\equiv}$ je podílový automat automatu $\mathcal{A}$

$i := 0$;

$\equiv_0 := \{(p, q)\ ;\ p \in F \Leftrightarrow q \in F\}$ (odpovídá kroku 1);

repeat

  $\equiv_{i+1} := \{(p, q)\ ;\ p \equiv_i q \ \wedge\ \forall a \in \Sigma\colon\ \delta(p, a) \equiv_i \delta(q, a)\}$;

  $i := i + 1$;

until $\equiv_i\ = \equiv_{i-1}$;

$\equiv\ := \equiv_i$;

foreach $r, s \in Q,\ a \in \Sigma$ **do**

  urči $[r], [s]$ třídy rozkladu $Q/_{\equiv},\ r \in [r],\ s \in [s]$;

  $\delta(r, a) = s \quad \Rightarrow \quad \eta([r], a) = [s]$

end

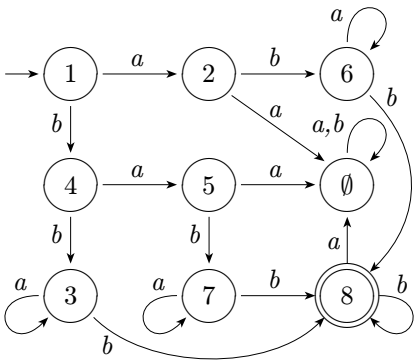
$\mathcal{A}/_{\equiv} = (Q/_{\equiv}, \Sigma, \eta, [q_0], F/_{\equiv})$

V cyklu *repeat* probíhá opakovaně krok 2 postupu, tedy do stejných (pod)skupin vždy sdružíme ty stavy ve skupině, které v následujícím kroku chovají stejně, tedy přecházejí do téže skupiny na tentýž signál na vstupu.

Příklad 1.12

Postup minimalizace automatu si ukážeme na následujícím konečném automatu:
$\mathcal{A} = (Q, \Sigma, \delta, 1, F) = (\{1, 2, \ldots, 8, \emptyset\}, \{a, b\}, \delta, 1, \{8\})$ je deterministický bez nedosažitelných stavů s totální přechodovou funkcí:

$\mathcal{A}$	a	b
$\to 1$	2	4
2	$\emptyset$	6
3	3	8
4	5	3
5	$\emptyset$	7
6	6	8
7	7	8
$\leftarrow 8$	$\emptyset$	8
$\emptyset$	$\emptyset$	$\emptyset$



Naším úkolem je sestrojit automat $\mathcal{A}/_{\equiv}$, který je podílovým automatem automatu $\mathcal{A}$. Všechny stavy jsou dosažitelné a automat je deterministický, tedy ho nemusíme nijak upravovat.

V prvním kroku sestrojíme tabulku skupin, kde zachováme označení řádků a sloupců a do buněk místo cílových stavů zapíšeme skupiny cílových stavů. Pouze jeden stav je koncový, tedy rozdělení do skupin je zatím následující:
skupina I $= \{1, 2, 3, 4, 5, 6, 7, \emptyset\}$
skupina II $= \{8\}$

Po prvním kroku vypadá tabulka skupin takto (vlevo je původní tabulka přechodů, vpravo je tabulka skupin se zdůvodněním obsahu buněk tabulky):

$\mathcal{A}$	a	b
$\to 1$	2	4
2	$\emptyset$	6
3	3	8
4	5	3
5	$\emptyset$	7
6	6	8
7	7	8
$\leftarrow 8$	$\emptyset$	8
$\emptyset$	$\emptyset$	$\emptyset$

skupina		a	b	
I	$\to 1$	I	I	protože $2, 4 \in$ I
	2	I	I	protože $\emptyset, 6 \in$ I
	3	I	II	protože $3 \in$ I, $8 \in$ II
	4	I	I	protože $5, 3 \in$ I
	5	I	I	protože $\emptyset, 7 \in$ I
	6	I	II	protože $6 \in$ I, $8 \in$ II
	7	I	II	protože $7 \in$ I, $8 \in$ II
	$\emptyset$	I	I	protože $\emptyset \in$ I
II	$\leftarrow 8$	I	II	protože $7 \in$ I, $8 \in$ II

Jak vidíme, v první skupině se shoduje obsah buněk v řádcích pro stavy $1, 2, 4, 5, \emptyset$, to bude v dalším kroku podskupina, kterou označíme I, a dále obsah buněk u stavů $3, 6, 7$, což bude nová podskupina III. Takže v dalším kroku budeme mít tři skupiny:
skupina I $= \{1, 2, 4, 5, \emptyset\}$ skupina II $= \{8\}$
skupina III $= \{3, 6, 7\}$

Za tímto odstavcem vlevo je tabulka skupin v první verzi s přehozenými řádky tak, aby u sebe byly řádky se stejným obsahem buněk, uprostřed pro porovnání přechodová tabulka s přehozenými řádky, vpravo druhá verze tabulky skupin s přepsanými buňkami.

skup.		a	b
I	$\to 1$	I	I
	2	I	I
	4	I	I
	5	I	I
	$\emptyset$	I	I
III	3	I	II
	6	I	II
	7	I	II
II	$\leftarrow 8$	I	II

$\mathcal{A}$	a	b
$\to 1$	2	4
2	$\emptyset$	6
4	5	3
5	$\emptyset$	7
$\emptyset$	$\emptyset$	$\emptyset$
3	3	8
6	6	8
7	7	8
$\leftarrow 8$	$\emptyset$	8

skup.		a	b	
I	$\to 1$	I	I	protože $2, 4 \in$ I
	2	I	III	protože $\emptyset \in$ I, $6 \in$ III
	4	I	III	protože $5 \in$ I, $3 \in$ III
	5	I	III	protože $\emptyset \in$ I, $7 \in$ III
	$\emptyset$	I	I	protože $\emptyset \in$ I
III	3	III	II	protože $3 \in$ III, $8 \in$ II
	6	III	II	protože $6 \in$ III, $8 \in$ II
	7	III	II	protože $7 \in$ III, $8 \in$ II
II	$\leftarrow 8$	I	II	protože $7 \in$ I, $8 \in$ II

Tentýž postup použijeme znovu. Vidíme, že ve skupině I je ještě „nekonzistence“, tedy ji rozdělíme na skupiny I a IV a celkem máme čtyři skupiny:
skupina I $= \{1, \emptyset\}$ skupina III $= \{3, 6, 7\}$
skupina IV $= \{2, 4, 5\}$ skupina II $= \{8\}$

Opět následuje trojice tabulek – předchozí tabulka skupin s přehozenými řádky v první skupině, tabulka přechodů a výsledná tabulka skupin pro tento krok.

skup.		a	b
I	$\to 1$	I	I
	$\emptyset$	I	I
IV	2	I	III
	4	I	III
	5	I	III
III	3	III	II
	6	III	II
	7	III	II
II	$\leftarrow 8$	I	II

$\mathcal{A}$	a	b
$\to 1$	2	4
$\emptyset$	$\emptyset$	$\emptyset$
2	$\emptyset$	6
4	5	3
5	$\emptyset$	7
3	3	8
6	6	8
7	7	8
$\leftarrow 8$	$\emptyset$	8

skup.		a	b	
I	$\to 1$	IV	IV	protože $2,4 \in$ IV
	$\emptyset$	I	I	protože $\emptyset \in$ I
IV	2	I	III	protože $\emptyset \in$ I, $6 \in$ III
	4	IV	III	protože $5 \in$ IV, $3 \in$ III
	5	I	III	protože $\emptyset \in$ I, $7 \in$ III
III	3	III	II	protože $3 \in$ III, $8 \in$ II
	6	III	II	protože $6 \in$ III, $8 \in$ II
	7	III	II	protože $7 \in$ III, $8 \in$ II
II	$\leftarrow 8$	I	II	protože $7 \in$ I, $8 \in$ II

Dále je třeba rozdělit skupiny I a IV, což zřejmě už bude poslední krok. Po jejich rozdělení získáme celkem šest skupin:

skupina I $= \{1\}$ skupina IV $= \{2,5\}$ skupina III $= \{3,6,7\}$
skupina V $= \{\emptyset\}$ skupina VI $= \{4\}$ skupina II $= \{8\}$

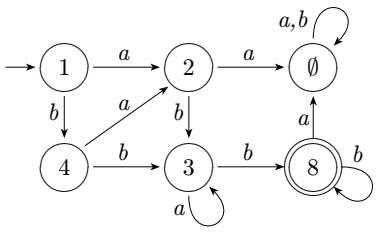
skup.		a	b
I	$\to 1$	IV	IV
V	$\emptyset$	I	I
IV	2	I	III
	5	I	III
VI	4	IV	III
III	3	III	II
	6	III	II
	7	III	II
II	$\leftarrow 8$	I	II

$\mathcal{A}$	a	b
$\to 1$	2	4
$\emptyset$	$\emptyset$	$\emptyset$
2	$\emptyset$	6
5	$\emptyset$	7
4	5	3
3	3	8
6	6	8
7	7	8
$\leftarrow 8$	$\emptyset$	8

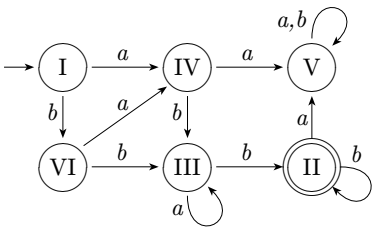
skup.		a	b	
I	$\to 1$	IV	VI	protože $2 \in$ IV, $4 \in$ VI
V	$\emptyset$	V	V	protože $\emptyset \in$ V
IV	2	V	III	protože $\emptyset \in$ V, $6 \in$ III
	5	V	III	protože $\emptyset \in$ V, $7 \in$ III
VI	4	IV	III	protože $5 \in$ IV, $3 \in$ III
III	3	III	II	protože $3 \in$ III, $8 \in$ II
	6	III	II	protože $6 \in$ III, $8 \in$ II
	7	III	II	protože $7 \in$ III, $8 \in$ II
II	$\leftarrow 8$	I	II	protože $7 \in$ I, $8 \in$ II

Algoritmus končí, protože uvnitř každé skupiny je obsah buněk v rámci jednoho sloupce stejný. Z tabulky skupin vyplývá, že stavy 2 a 5 jsou zaměnitelné a dále stavy 3, 6 a 7 jsou zaměnitelné. Výsledný automat bude mít místo 9 stavů jen 6 stavů. Výslednou tabulku přechodů vytvoříme buď s použitím původních stavů (shrneme stavy ze stejné skupiny) nebo stavy označíme přímo skupinami:

$\mathcal{A}/_{\equiv}$	a	b
$\to 1$	2	4
$\emptyset$	$\emptyset$	$\emptyset$
2	$\emptyset$	3
4	2	3
3	3	8
$\leftarrow 8$	$\emptyset$	8



$\mathcal{A}/_{\equiv}$	a	b
$\to$ I	IV	VI
V	V	V
IV	V	III
VI	IV	III
III	III	II
$\leftarrow$ II	I	II



Další informace:

V těchto skriptech nejsou některé důkazy uvedeny. Případné zájemce odkazuji na zdroj [1], kde jsou všechny důkazy ukázány a vysvětleny.

1.5 Vztah mezi konečnými automaty a regulárními výrazy

Zatím jsme jednoduše předpokládali, že třída jazyků rozpoznávaných konečnými automaty je ekvivalentní třídě jazyků reprezentovaných regulárními výrazy (například v tvrzení, že pro každý regulární jazyk lze sestrojit ekvivalentní konečný automat a regulární výraz), ale toto tvrzení jsme si zatím nezdůvodnili ani nedokázali.

Věta 1.12

Třída jazyků rozpoznávaných konečnými automaty je ekvivalentní třídě jazyků reprezentovaných regulárními výrazy.

Důkaz ($\Leftarrow$)**:** Jestliže je dán regulární výraz a úkolem je sestrojit k němu ekvivalentní konečný automat, pak takový úkol zvládneme už se znalostmi, které jsme získali v minulém semestru – stačí si uvědomit, že třída jazyků rozpoznávaných konečnými automaty je uzavřena vzhledem k regulárním operacím (sjednocení, iterace, zřetězení) a tedy existují deterministické postupy pro sestrojení konečného automatu podle daného regulárního výrazu.

Důkaz v tomto směru tvrzení (automat podle reg. výrazu) tedy stojí na důkazech uzavřenosti třídy jazyků rozpoznávaných konečnými automaty vzhledem k operacím sjednocení, zřetězení a iterace. Tyto důkazy již byly provedeny. □

Postup ($\Rightarrow$)

Je dán redukovaný deterministický konečný automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. Úkolem je najít regulární výraz $R = L(\mathcal{A})$.

Na straně 25 jsou v definici Ekvivalence stavů automatu definovány automaty $\mathcal{A}_q$ pro různé stavy $q \in Q$ jako varianty automatu $\mathcal{A}$, přičemž stav q je považován za počáteční stav. Pro zjednodušení označíme stavy čísly, tedy $Q = \{1, 2, \ldots\}$.

Samotný postup zjištění regulárního výrazu je vlastně grafový algoritmus, ve kterém postupně (iterativně) tvoříme množinu řetězců, při jejichž zpracování postupujeme po cestě mezi dvěma konkrétními stavy. V prvních iteracích jsou tyto cesty zatím krátké, ale jejich prodlužováním a spojováním se postupně dostaneme k výsledku, kterým je výraz odpovídající sjednocení cest od počátečního stavu k jednotlivým koncovým stavům. Množinu řetězců pro cestu mezi dvěma stavy $i, j \in Q$ označíme takto:

$$R_{ij} = \{w \in \Sigma^* \ ; \ (i, w) \vdash^*_{\mathcal{A}_i} (j, \varepsilon)\}$$

Tedy pokud v daném automatu postupujeme od stavu i do stavu j, pak na cestě rozpoznáváme právě slova z množiny R_{ij}. Například R_{24} je množina slov rozpoznaných na cestách v grafu auto-

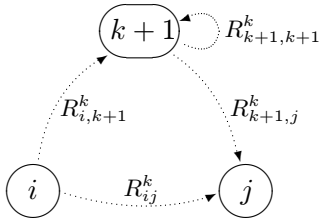
matu vedoucích mezi stavy 2 a 4. Pokud za i dosadíme počáteční stav a za j různé koncové stavy, pak sjednocením vytvořených jazyků získáme jazyk automatu:

$$L(\mathcal{A}) = \bigcup_{f \in F} R_{q_0 f}$$

Budeme postupovat iterativně, zkonstruujeme množiny R_{ij}^k pro $k = 0, 1, \ldots$ postupně pro různé indexy k. R_{ij}^k jsou podmnožiny množiny R_{ij} takové, že na cestě v automatu, která je zpracováním slova z této množiny, se nacházejí pouze stavy s číselným označením menším nebo rovným číslu k (neplatí pro „krajní stavy“ i a j, ty mohou být označeny číslem vyšším než k). Například R_{24}^3 je množina všech slov rozpoznaných na cestě v grafu automatu vedoucí ze stavu 2 do stavu 4, ovšem cesta může vést pouze přes stavy $1, 2, 3$ (může na ní být i smyčka). Formálně:
$R_{ij}^k = \{w \in R_{ij} \ ;\ \text{pokud existuje } m \in Q\text{: } (i, w) \vdash_{\mathcal{A}_i}^+ (m, u) \vdash_{\mathcal{A}_i}^+ (j, \varepsilon),\ w \neq u \neq \varepsilon,\ \text{pak } m \leq k\}$

Bází iterace jsou nejkratší cesty v grafu (tj. přímé), R_{ij}^0 je tedy množina všech slov rozpoznaných na cestě v grafu automatu bez mezilehlých stavů. Takže:

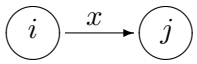
- pokud $\delta(i, a) = j$ pro jakékoliv $a \in \Sigma$, pak $a \in R_{ij}^0$,
- pokud $i = j$, pak $\varepsilon \in R_{ij}^0$.



Nic jiného nelze do R_{ij}^0 zařadit. V tabulce přechodů a v diagramu vypadá vztah z první odrážky takto:

jestliže

	...	x	...
...	...	...	...
i	...	j	...
...	...	...	...



potom platí: $x \in R_{ij}^0$

V dalších krocích tyto cesty skládáme tak, jak je naznačeno na obrázku vpravo, vzorec vypadá následovně:

$$R_{ij}^{k+1} = R_{ij}^k + \left(R_{i,k+1}^k \cdot (R_{k+1,k+1}^k)^* \cdot R_{k+1,j}^k\right), \quad 0 \leq k \leq \mathrm{card}(Q) - 1$$

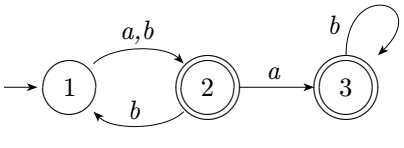
Výsledkem je vždy regulární výraz, který se s každým krokem komplikuje. Z toho vyplývá, že pro krok $k+1$ potřebujeme výsledky kroku k a nic jiného. Vzorec se používá až pro $k + 1 = \mathrm{card}(Q)$, tedy musíme na cestách „dovolit“ postupně všechny stavy, přičemž v každém kroku se zvyšuje složitost získaných regulárních výrazů. Ovšem pro poslední iteraci nám stačí zjistit regulární výrazy pro cesty vedoucí z počátečního stavu do jednotlivých koncových stavů, výsledkem je pak jejich sjednocení.

Příklad 1.13

Zjistíme regulární výraz odpovídající jazyku konečného automatu $\mathcal{A} = (Q, \Sigma, \delta, 1, \{2, 3\})$:

$\mathcal{A}$	a	b
$\to 1$	2	2
$\leftarrow 2$	3	1
$\leftarrow 3$		3

$\delta(1, a) = 2$ $\quad$ $\delta(2, b) = 1$
$\delta(1, b) = 2$ $\quad$ $\delta(3, b) = 3$
$\delta(2, a) = 3$



Tento automat je deterministický a redukovaný, takže není třeba provádět žádné další úpravy. Nejdřív zjistíme regulární výrazy pro bázi alogoritmu, které reprezentují „jednokrokové“ cesty v grafu automatu:

$R^0_{11} = \varepsilon$	protože u R^0_{ij} platí $i = j$
$R^0_{12} = a + b$	protože $\delta(1, a) = 2,\ \delta(1, b) = 2$ určují ohodnocení cesty mezi stavy $1, 2$
$R^0_{13} = \emptyset$	protože mezi stavy $1, 3$ není přímá cesta
$R^0_{21} = b$	protože $\delta(2, b) = 1$ určuje ohodnocení cesty mezi stavy $2, 1$
$R^0_{22} = \varepsilon$	protože u R^0_{ij} platí $i = j$
$R^0_{23} = a$	protože $\delta(2, a) = 3$ určuje ohodnocení cesty mezi stavy $2, 3$
$R^0_{31} = \emptyset$	protože mezi stavy $3, 1$ není přímá cesta
$R^0_{32} = \emptyset$	protože mezi stavy $3, 2$ není přímá cesta
$R^0_{33} = b + \varepsilon$	protože je $\delta(3, b) = 3$ a u R^0_{ij} platí $i = j$

Následuje první iterace, ve které zjišťujeme regulární výrazy R^1_{ij} odpovídající cestám mezi stavy i a j, přičemž tyto cesty mohou vést nejvýše přes stav 1.

R^1_{ij}	=	R^0_{ij}	+	R^0_{i1}	·	$(R^0_{11})^*$	·	R^0_{1j}	=	...
R^1_{11}	=	ε	+	ε	·	ε^*	·	ε	=	ε
R^1_{12}	=	$a + b$	+	ε	·	ε^*	·	$(a + b)$	=	$a + b$
R^1_{13}	=	$\emptyset$	+	ε	·	ε^*	·	$\emptyset$	=	$\emptyset$
R^1_{21}	=	b	+	b	·	ε^*	·	ε	=	b
R^1_{22}	=	ε	+	b	·	ε^*	·	$(a + b)$	=	$\varepsilon + b(a + b)$
R^1_{23}	=	a	+	b	·	ε^*	·	$\emptyset$	=	a
R^1_{31}	=	$\emptyset$	+	$\emptyset$	·	ε^*	·	ε	=	$\emptyset$
R^1_{32}	=	$\emptyset$	+	$\emptyset$	·	ε^*	·	$(a + b)$	=	$\emptyset$
R^1_{33}	=	$b + \varepsilon$	+	$\emptyset$	·	ε^*	·	$\emptyset$	=	$b + \varepsilon$

Využili jsme tyto vztahy: $\varepsilon \cdot X = X,\ \varepsilon^* = \varepsilon,\ \emptyset + X = X,\ \emptyset \cdot X = \emptyset$.

V druhé iteraci zjišťujeme regulární výrazy R^2_{ij} pro cesty vedoucí mezi stavy i a j, a to pouze přes stavy 1 a 2.

R^2_{ij}	=	R^1_{ij}	+	R^1_{i2}	·	$(R^1_{22})^*$	·	R^1_{2j}	=	...
R^2_{11}	=	ε	+	$(a + b)$	·	$(b(a + b))^*$	·	b	=	$((a + b)b)^*$
R^2_{12}	=	$a + b$	+	$(a + b)$	·	$(b(a + b))^*$	·	$(\varepsilon + b(a + b))$	=	$(a + b)\,(b(a + b))^*$
R^2_{13}	=	$\emptyset$	+	$(a + b)$	·	$(b(a + b))^*$	·	a	=	$(a + b)\,(b(a + b))^*\,a$
R^2_{21}	=	b	+	$(\varepsilon + b(a + b))$	·	$(b(a + b))^*$	·	b	=	$(b(a + b))^*\,b$
R^2_{22}	=	$\varepsilon + b(a + b)$	+	$(\varepsilon + b(a + b))$	·	$(b(a + b))^*$	·	$(\varepsilon + b(a + b))$	=	$(b(a + b))^*$
R^2_{23}	=	a	+	$(\varepsilon + b(a + b))$	·	$(b(a + b))^*$	·	a	=	$(b(a + b))^*\,a$
R^2_{31}	=	$\emptyset$	+	$\emptyset$	·	$(b(a + b))^*$	·	b	=	$\emptyset$
R^2_{32}	=	$\emptyset$	+	$\emptyset$	·	$(b(a + b))^*$	·	$(\varepsilon + b(a + b))$	=	$\emptyset$
R^2_{33}	=	$b + \varepsilon$	+	$\emptyset$	·	$(b(a + b))^*$	·	a	=	$b + \varepsilon$

Kromě výše uvedených jsme využili vztah $(\varepsilon + b(a + b))^* = (b(a + b))^*$. Taky je dobré si uvědomit, že platí $X + (\varepsilon + Y) \cdot Y^* X = Y^* X$.

Zbývá třetí iterace, při které však nemusíme zjišťovat všechny výrazy R_{ij}^3, stačí pouze výrazy odpovídající cestám z počátečního stavu do některého koncového.

R_{ij}^3	$=$	R_{ij}^2	$+$	R_{i3}^2	$\cdot (R_{33}^2)^* \cdot$	R_{3j}^2	$=$	$\ldots$
R_{12}^3	$=$	$(a+b)\,(b(a+b))^*$	$+$	$(a+b)\,(b(a+b))^*\,a \cdot$	b^* $\cdot$	$\emptyset$	$=$	$(a+b)\,(b(a+b))^*$
R_{13}^3	$=$	$(a+b)\,(b(a+b))^*\,a$	$+$	$(a+b)\,(b(a+b))^*\,a \cdot$	b^* $\cdot$	$(b+\varepsilon)$	$=$	$(a+b)\,(b(a+b))^*\,ab^*$

Kromě výše uvedených jsme využili vztahy $(b+\varepsilon)^* = b^*$ a $b^* \cdot (b+\varepsilon) = b^*$. Zbývá zapsat výsledný regulární výraz:

$$\begin{aligned} L(\mathcal{A}) = R = R_{12}^3 + R_{13}^3 &= (a+b)\,(b(a+b))^* + (a+b)\,(b(a+b))^*\,ab^* \\ &= (a+b)\,(b(a+b))^* \cdot (\varepsilon + ab^*) \end{aligned}$$

Poznámka:

Konečný automat $\mathcal{A}$ můžeme minimalizovat také tak, že zjistíme přímo regulární výrazy pro jednotlivé pomocné automaty $\mathcal{A}_q$ pro různé (počáteční) stavy q a pak tyto regulární výrazy porovnat. Jak vyplývá z předchozího příkladu, není to ve skutečnosti až tak jednoduché – časově úspornější je postup ukázaný v předchozí sekci o minimalizaci.

Kapitola 2

Bezkontextové gramatiky a jazyky

V této kapitole si zopakujeme jazyky patřící do třídy jazyků $\mathcal{L}(2)$ (resp. CF) v Chomského hierarchii, tedy bezkontextové jazyky, a k nim ekvivalentní bezkontextové gramatiky. Budeme se zabývat pokročilejšími úlohami souvisejícími s bezkontextovými gramatikami, včetně různých konverzí.

2.1 Definice bezkontextové gramatiky

Připomeneme si definici bezkontextové gramatiky:

Definice 2.1 (Bezkontextová gramatika)

Gramatika typu 2 (bezkontextová, context-free – CF) je gramatika, jejíž všechna pravidla jsou v tomto tvaru:

$$A \to \beta, \quad A \in N,\ \beta \in (N \cup T)^* \tag{2.1}$$

Ovšem k definici bezkontextové gramatiky ve skutečnosti patří také související definice, které platí pro gramatiky obecně – relace kroku odvození, reflexivní a tranzitivní uzávěr této relace, jazyk generovaný gramatikou, větná forma, věta.

Příklad 2.1

Vytvoříme gramatiku generující následující jazyk:
$L_1 = \{0^n 1^m \ ;\ 1 \leq n \leq m\}$

Ve slovech jazyka L_1 jsou nejdřív symboly 0 a pak symboly 1. Důležitá je však podmínka, která říká, že počet nul má být menší nebo roven počtu jedniček. Splnění podmínky zajistíme tak, že nejdřív v rekurzivním pravidle generujeme nuly a jedničky tak, že jich je stejný počet, a pak v prostřední části můžeme přidat další jedničky.
$G_1 = (\{A\}, \{0,1\}, P, A)$, kde množina P obsahuje pravidla
$A \to 0A1 \mid A1 \mid 01$
Ukázka derivace:
$A \Rightarrow 0A1 \Rightarrow 0A11 \Rightarrow 00111$

Příklad 2.2

L_2 je jazyk matematických výrazů obsahujících

- celá čísla,
- operátory $+$, $-$, $*$, $/$, tentokrát s ohledem na priority operátorů,
- závorky.

Gramatika generující tento jazyk je
$G_2 = (\{E, F, G\}, \{n, +, -, *, /, (,)\}, P, E)$, kde množina P obsahuje pravidla
$E \to E + F \mid E - F \mid F$
$F \to F * G \mid F/G \mid G$
$G \to (E) \mid n$

Ukázka derivace:
$E \Rightarrow F \Rightarrow F * G \Rightarrow G * G \Rightarrow (E) * G \Rightarrow (E + F) * G \Rightarrow (F + F) * G \Rightarrow (G + F) * G \Rightarrow$
$\Rightarrow (n + F) * G \Rightarrow (n + G) * G \Rightarrow (n + n) * G \Rightarrow (n + n) * n$

Tento jazyk se po určité úpravě používá jako základ pro syntaktickou analýzu běžných programovacích jazyků.

To by pro zopakování mohlo stačit. Připomeňme si ještě, že k derivaci podle bezkontextové gramatiky lze sestrojit její grafickou obdobu – derivační strom. Ovšemže derivační strom můžeme sestrojit i pro derivaci v regulární gramatice, ale vzhledem k jednoduchosti předpisu regulárních pravidel by to bylo zbytečné.

Dále se budeme zabývat vlastnostmi bezkontextových gramatik. Některé vlastnosti jsme si již představili v minulém semestru, ale regulérní důkaz u nich ještě nebyl vyžadován. To v tomto semestru napravíme.

2.2 Transformace bezkontextových gramatik

2.2.1 Nezkracující bezkontextová gramatika

Nezkracující bezkontextová gramatika je taková bezkontextová gramatika $G = (N, T, P, S)$, kde množina pravidel P buď neobsahuje žádné ε-pravidlo, nebo existuje jediné ε-pravidlo $S \to \varepsilon$ a zároveň S není na pravé straně žádného pravidla.

Definice 2.2 (Nezkracující bezkontextová gramatika)

Nezkracující bezkontextovou gramatikou je taková gramatika $G = (N, T, P, S)$, jejíž všechna pravidla jsou v tomto tvaru:

$$A \to \alpha, \quad |\alpha| \geq 1, \quad A \in N, \ \alpha \in (N \cup T)^* \tag{2.2}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Věta 2.1 (Převod na nezkracující gramatiku)

Nechť G je bezkontextová gramatika. Pak existuje bezkontextová gramatika G' nezkracující taková, že $L(G') = L(G)$.

Deterministický (konečný a jednoznačný) exaktní postup si ukážeme v následujícím postupu konstrukce. V něm se jedná především o to, jak určit všechny neterminály, které lze (po různém počtu kroků) přepsat na ε. Tuto informaci dále využijeme tak, že budeme na pravých stranách pravidel postupně vynechávat různé permutace/variace s opakováním symbolů majících tuto vlastnost.

Postup konstrukce (Vytvoření ekvivalentní nezkracující gramatiky): Je dána gramatika $G = (N, T, P, S)$. Sestrojíme k ní ekvivalentní nezkracující gramatiku $G' = (N', T, P', S')$.

Nejdřív vytvoříme nezkracující gramatiku G_p tak, že odstraníme ε-pravidla bez ohledu na to, zda se jedná o startovací symbol (tj. pokud $\varepsilon \in L(G)$, dočasně toto slovo z jazyka odstraníme). Platí, že $L(G_p) = L(G) - \{\varepsilon\}$. To znamená, že pokud v jazyce gramatiky G je slovo ε, budou se jazyky gramatik lišit právě o toto slovo, jinak budou ekvivalentní.

Následně vytvoříme finální gramatiku G' takovou, že $L(G') = L(G)$. Pokud $\varepsilon \notin L(G)$, pak tento krok nemusíme řešit (platilo by $L(G') = L(G_p) = L(G)$), v opačném případě je třeba do jazyka nové gramatiky přidat prázdné slovo.

Vytvoříme množinu N_ε, což je množina všech neterminálů, které lze (po jakémkoliv počtu kroků) přepsat na ε. Tuto množinu budeme tvořit iterativním postupem.

- Jako bázi použijeme prázdnou množinu:
  $N_{\varepsilon,0} = \emptyset$
- V prvním kroku iterace přidáme všechny neterminály, pro které existuje ε-pravidlo:
  $N_{\varepsilon,1} = N_{\varepsilon,0} \cup \{X \in N \ ; \ (X \to \varepsilon) \in P\} = N_{\varepsilon,0} \cup \left\{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in N_{\varepsilon,0}^*\right\}$
- V dalších krocích postupujeme podle tohoto schématu:
  $N_{\varepsilon,i} = N_{\varepsilon,i-1} \cup \left\{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in N_{\varepsilon,i-1}^*\right\}$
- Pokud $N_{\varepsilon,i} = N_{\varepsilon,i-1} = N_\varepsilon$, končíme, máme množinu všech neterminálů, které lze po konečném počtu kroků přepsat na ε.

V každém kroku (kromě báze) přidáváme všechny neterminály, které lze přepsat na řetězec skládající se *pouze* z těch neterminálů, které jsme do množiny přidali v předchozích krocích. V prvním kroku jde přímo o neterminály, pro které existují ε-pravidla, v druhém kroku přidáme neterminály s pravidlem, kde na pravé straně jsou pouze neterminály, pro které existují ε-pravidla, atd.

Množinu N_ε použijeme pro určení pravidel gramatiky G_p – přidáváme nová pravidla podle původních pravidel, ve kterých postupně vypouštíme různý počet a různé kombinace neterminálů umístěných v množině N_ε. Pokud vznikne ε-pravidlo, ignorujeme je.

Pro všechna pravidla $(B \to \alpha) \in P, \ |\alpha| > 0$ provedeme:

1. zařadíme toto pravidlo do množiny P_p,
2. určíme na pravé straně pravidla všechny prvky patřící do množiny N_ε:
   $B \to \alpha_0 A_1 \alpha_1 A_2 \alpha_2 A_3 \alpha_3 \ldots A_{n-1} \alpha_{n-1} A_n \alpha_n$, kde $A_i \in N_\varepsilon, \ 1 \leq i \leq n, \ \alpha_i \in ((N \cup T) - N_\varepsilon)^*$

3. přidáme do množiny P_p tato nová pravidla:
   - vynecháme vždy po jednom výskytu symbolů A_i:
     $B \to \alpha_0\ \alpha_1 A_2\alpha_2 A_3\alpha_3 \ldots A_{n-1}\alpha_{n-1}A_n\alpha_n$
     $B \to \alpha_0 A_1\alpha_1\ \alpha_2 A_3\alpha_3 \ldots A_{n-1}\alpha_{n-1}A_n\alpha_n$
     $\ldots$
     $B \to \alpha_0 A_1\alpha_1 A_2\alpha_2 A_3\alpha_3 \ldots A_{n-1}\alpha_{n-1}\ \alpha_n$
   - vynecháme dva výskyty symbolů A_i:
     $B \to \alpha_0\ \alpha_1\ \alpha_2 A_3\alpha_3 \ldots A_{n-1}\alpha_{n-1}A_n\alpha_n$
     $B \to \alpha_0\ \alpha_1 A_2\alpha_2\ \alpha_3 \ldots A_{n-1}\alpha_{n-1}A_n\alpha_n$
     $\ldots$
     $B \to \alpha_0 A_1\alpha_1 A_2\alpha_2 A_3\alpha_3 \ldots\ \ \alpha_{n-1}\ \alpha_n$
   - atd.
   - vynecháme všechny výskyty symbolů A_i:
     $B \to \alpha_0\ \alpha_1\ \alpha_2\ \alpha_3\ \ldots\ \alpha_{n-1}\ \alpha_n$
4. Pravidla $B \to \varepsilon$ (tj. pro $|\alpha| = 0$) ignorujeme, nebudou zařazena do množiny pravidel gramatiky G_p.

Pokud $\varepsilon \notin L(G)$, pak jsme hotovi – výsledná gramatika je $G' = G_p = (N, T, P', S)$ s pravidly $P' = P_p$ sestrojenými podle předchozího postupu.

Jestliže však $\varepsilon \in L(G)$ (to poznáme podle toho, že se do množiny N_ε dostane startovací symbol), pokračujeme následovně.

Předpokládejme tedy, že jsme již odstranili všechna ε-pravidla tak, jak je popsáno výše (i pravidlo $S \to \varepsilon$). Mezivýsledkem je gramatika $G_p = (N, T, P_p, S)$. Potom vytvoříme gramatiku $G' = (N \cup \{S'\},\ T,\ P', S')$ tak, že:

- přidáme nový startovací symbol S' (nově přidaný, tedy $S' \notin N$),
- přidáme pro tento symbol dvě pravidla: $S' \to S \mid \varepsilon$ (při generování neprázdných slov se napojíme na výpočet v původní gramatice a přidáme možnost vygenerování prázdného slova):
  $P' = P_p \cup \{S' \to S \mid \varepsilon\}$ □

Příklad 2.3

Postup si ukážeme na této gramatice:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aBA \mid BB \mid ac$
$A \to BS \mid aA \mid a$
$B \to bB \mid aA \mid \varepsilon$

Sestrojíme množinu neterminálů přepsatelných na ε.
$N_{\varepsilon,0} = \emptyset$
$N_{\varepsilon,1} = \emptyset \cup \{B\} = \{B\}$
$N_{\varepsilon,2} = \{B\} \cup \{S\} = \{B, S\}$ (pravidlo $S \to BB$)
$N_{\varepsilon,3} = \{B, S\} \cup \{A\} = \{B, S, A\}$ (pravidlo $A \to BS$)
$N_{\varepsilon,4} = \{B, S, A\} \cup \emptyset = \{B, S, A\} = N_{\varepsilon,3} = N_\varepsilon$

Všimněte si, že v množině N_ε je i startovací symbol gramatiky S. To znamená, že v jazyce gramatiky je prázdné slovo, třebaže to na původních pravidlech nebylo na první pohled poznat.

Teď nás čeká přidávání nových pravidel – v pravidlech postupně vypouštíme různé variace (s opakováním) neterminálů, které jsme získali v množině N_ε (v tomto případě všech neterminálů). Zatím se nejedná o výsledek, pro gramatiku G_p zatím platí $L(G_p) = L(G) - \{\varepsilon\}$.
$G_p = (\{S, A, B\},\ \{a, b, c\},\ P_p,\ S)$
$S \to aBA \mid aA \mid aB \mid a \mid BB \mid B \mid ac$
$A \to BS \mid S \mid B \mid aA \mid a$ (pravidlo $A \to a$ nemusíme přidávat, už tam je)
$B \to bB \mid b \mid aA \mid a$

Protože je v množině N_ε startovací symbol S, znamená to, že do jazyka gramatiky patří prázdné slovo. Proto je třeba provést ještě jednu úpravu:
$G' = (\{S', S, A, B\},\ \{a, b, c\},\ P',\ S')$
$S' \to S \mid \varepsilon$
$S \to aBA \mid aA \mid aB \mid a \mid BB \mid B \mid ac$
$A \to BS \mid S \mid B \mid aA \mid a$
$B \to bB \mid b \mid aA \mid a$

Důkaz (Věta 2.1): Je zřejmé, že výsledná gramatika G' sestrojená podle výše uvedeného postupu konstrukce je nezkracující (protože neobsahuje žádná ε-pravidla). Ukážeme, že změny, které jsme v gramatice provedli, jsou *ekvivalentní*, tj. nemění jazyk generovaný gramatikou.

Nejdřív probereme případ, kdy $\varepsilon \notin L(G)$. Vycházíme z gramatiky $G = (N, T, P, S)$ a sestrojená gramatika je $G' = (N, T, P', S)$.

Dokazujeme $L(G) \subseteq L(G')$:
Zde si stačí uvědomit, že místo ε-pravidel jsme do gramatiky zařadili pravidla odpovídající původním pravidlům gramatiky, kde jsme odstranili jednotlivé neterminály přepsatelné na ε v různých kombinacích, čímž jsme simulovali použití ε-pravidla na řetězec, jehož podřetězcem je pravá strana některého původního pravidla. Důsledkem je dokonce možné zkrácení derivace.

V každém případě ke každé derivaci v gramatice G dokážeme sestrojit ekvivalentní derivaci téhož slova v gramatice G'.

Dokazujeme $L(G) \supseteq L(G')$:
V množině pravidel gramatiky G' máme kromě původních neepsilonových pravidel nová pravidla, která však respektují původní pravidla v kombinaci s již odstraněnými ε-pravidly. Tedy pro kteroukoliv derivaci v gramatice G' dokážeme sestrojit ekvivalentní derivaci téhož slova v G.

Zbývá případ, kdy $\varepsilon \in L(G)$. Pokud $\varepsilon \in L(G)$, pak podle výše uvedeného postupu máme výslednou gramatiku $G' = (N \cup \{S'\},\ T,\ P_p \cup \{S' \to S \mid \varepsilon\},\ S')$.

Pro derivace slov $w \in L(G)$ takových, že $|w| \geq 1$, platí totéž co v předchozí části důkazu, jen je odvození o jeden krok delší:
$S' \Rightarrow S \Rightarrow^* w$

Zaměřme se tedy na vygenerování slova ε. V gramatice G' použijeme derivaci $S' \Rightarrow \varepsilon$, tedy $\varepsilon \in L(G')$. Proto jestliže $\varepsilon \in L(G)$, pak $\varepsilon \in L(G')$.

Jestliže $\varepsilon \in L(G')$, pak existuje pravidlo $S' \to \varepsilon$ a to se do množiny P' dostalo jen tehdy, pokud $S \in N_\varepsilon$. Do množiny N_ε se S dostalo jen tehdy, pokud v G existuje derivace $S \Rightarrow^* \varepsilon$, a tedy $\varepsilon \in L(G)$. Proto pokud $\varepsilon \in L(G')$, pak také $\varepsilon \in L(G)$. □

2.2.2 Redukovaná gramatika

Redukci bezkontextové gramatiky jsme v minulém semestru také zkoušeli, opět se zde po zopakování pojmů zaměříme na správné provedení formálního důkazu.

Definice 2.3 (Nadbytečný neterminál)

$X \in N$ je nadbytečný neterminál v gramatice $G = (N, T, P, S)$, pokud neexistuje žádné terminální slovo, které lze z tohoto symbolu vygenerovat, tj. neexistuje derivace

$$X \Rightarrow^* w, \quad w \in T^* \tag{2.3}$$

Definice 2.4 (Nedostupný symbol)

Symbol $X \in (N \cup T)$ je nedostupný v gramatice $G = (N, T, P, S)$, jestliže se nemůže objevit v žádné větné formě, tj. neexistuje derivace

$$S \Rightarrow^* \alpha X \beta, \quad \alpha, \beta \in (N \cup T)^* \tag{2.4}$$

Definice 2.5 (Redukovaná gramatika)

Bezkontextová gramatika $G = (N, T, P, S)$ je redukovaná, pokud neobsahuje žádné nadbytečné a nedostupné symboly.

Po větě následují postupně popisy konstrukce, příklady a důkazy k oběma směrům redukce.

Věta 2.2 (Redukce bezkontextové gramatiky)

Ke každé bezkontextové gramatice G existuje redukovaná gramatika (bez nadbytečných a nedostupných symbolů) G' taková, že $L(G) = L(G')$.

Postup konstrukce (Odstranění nadbytečných neterminálů): Sestrojíme množinu neterminálů E_{def}, ze kterých lze vygenerovat terminální řetězec. Nadbytečné neterminály jsou pak právě ty prvky, které do této množiny nepatří. Původní gramatiku označíme $G = (N, T, P, S)$, sestrojíme gramatiku bez nadbytečných neterminálů $G' = (N', T, P', S)$.

- Jako bázi použijeme množinu terminálních symbolů (z výsledné množiny je pak odstraníme): $E_0 = T$

- V prvním kroku iterace přidáme všechny neterminály, pro které existuje terminální nebo ε-pravidlo:
  $E_1 = E_0 \cup \{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in T^*\} = E_0 \cup \{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in E_0^*\}$
- V každém dalším kroku iterace přidáváme další neterminály, pro které existuje pravidlo, na jehož pravé straně jsou pouze symboly zařazené do množiny v předchozích krocích:
  $E_i = E_{i-1} \cup \left\{X \in N \ ; \ (X \to \alpha) \in P, \ \alpha \in E_{i-1}^*\right\}$
- Pokud se už množina nemění (není co přidat), končíme:
  $E_i = E_{i-1} = E_{def}$

V množině E_{def} máme pouze takové symboly, ze kterých je možné vygenerovat terminální slovo (prázdné slovo je taky terminální).

Stanovíme novou množinu neterminálů: $N' = N \cap E_{def}$. Nová množina pravidel bude obsahovat pouze ta pravidla, která mají na levé a pravé straně pouze symboly z množiny E_{def}:
$P' = \left\{(A \to \alpha) \in P \ ; \ A \in E_{def}, \ \alpha \in E_{def}^*\right\}$ □

Příklad 2.4

V následující gramatice odstraníme nadbytečné neterminály:
$G = (\{S, A, B, C, D\}, \ \{a, b, c, d\}, \ P, \ S)$
$S \to aAbC \mid c$
$A \to aA \mid Cc$
$B \to cB \mid dD$
$C \to cB \mid aA \mid b$
$D \to Bd$

Použijeme výše uvedený iterativní postup a sestrojíme množinu E_{def}. Pak určíme novou množinu neterminálů a novou množinu pravidel.
$E_0 = T = \{a, b, c, d\}$ (báze iterace)
$E_1 = \{a, b, c, d, S, C\}$ (podle $S \to c, \ C \to b$)
$E_2 = \{a, b, c, d, S, C, A\}$ (podle $A \to Cc$)
$E_3 = E_2 = E_{def}$

Nová množina neterminálů je $N' = \{S, A, B, C, D\} \cap \{a, b, c, d, S, C, A\} = \{S, A, C\}$. Nová množina pravidel P' bude obsahovat pouze ta pravidla, která mají na levé i pravé straně pouze symboly z množiny E_{def}, tedy celá výsledná gramatika vypadá takto:
$G' = (\{S, A, C\}, \ \{a, b, c, d\}, \ P', \ S)$
$S \to aAbC \mid c$
$A \to aA \mid Cc$
$C \to aA \mid b$

Důkaz (Odstranění nadbytečných neterminálů): Je dána gramatika $G = (N, T, P, S)$. Výše uvedeným postupem konstrukce jsme sestrojili množinu E_{def} a gramatiku $G' = (N', T, P', S)$. Je třeba dokázat, že gramatiky G a G' jsou ekvivalentní.

Algoritmus konstrukce množiny E_{def} je konečný (protože v každém kroku přidáváme nejméně jeden prvek množiny N, přičemž množina N je konečná) a výsledkem je množina symbolů, ze kterých lze vygenerovat terminální slovo (to plyne z faktu, že algoritmus je iterativní, přičemž postupujeme podle pravidel – do množiny se dostanou právě ty symboly, které lze v konečném počtu kroků přepsat na terminální řetězec).

Dokazujeme $L(G) \subseteq L(G')$:
Nechť $w \in L(G)$.

- $\Rightarrow$ existuje derivace slova w v gramatice G:
  $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$
- $\Rightarrow$ do množiny E_{def} se řadí všechny symboly, z nichž lze generovat terminální slovo, tedy všechny symboly v použitých větných formách této derivace do ní patří,
- $\Rightarrow$ z principu konstrukce pravidel gramatiky G' vyplývá, že pravidlo z množiny P použité v kroku derivace $\alpha_{i-1} \Rightarrow \alpha_i,\ 1 \leq i \leq n$ existuje také v množině P' a nebylo algoritmem odstraněno,
- $\Rightarrow$ tatáž derivace slova w existuje i v gramatice G',
- $\Rightarrow w \in L(G')$.

Tatáž úvaha platí i v případě, že $w = \varepsilon$, slovo by bylo odvozeno v jednom kroku derivace.

Dokazujeme $L(G) \supseteq L(G')$:
Tento směr je triviální, protože všechna pravidla z množiny P' existují v původní množině P (při konstrukci množiny P' jsme žádná nová pravidla nepřidávali). Pro každé slovo $w \in L(G')$ existuje derivace v gramatice G' a tatáž derivace existuje i v gramatice G, tedy $w \in L(G)$. □

Poznámka:

Pořadí je důležité – odstraňujeme nejdřív nadbytečné neterminály a až potom nedostupné symboly.

Postup konstrukce (Odstranění nedostupných symbolů): Opět použijeme iterativní metodu – sestrojíme množinu symbolů S_{def}, které jsou dostupné (vyskytují se v některé větné formě v derivaci ze startovacího symbolu). Nedostupné symboly jsou ty, které do této množiny nepatří. Původní gramatiku označíme $G' = (N', T, P, S)$ (předpokládáme, že již byly odstraněny všechny nadbytečné symboly), sestrojíme gramatiku $G'' = (N'', T', P'', S)$.

- Jako bázi použijeme množinu obsahující startovací symbol gramatiky:
  $S_0 = \{S\}$
- V prvním kroku iterace přidáme všechny symboly z pravé strany pravidel pro startovací symbol, protože právě tyto symboly jsou dostupné ze startovacího jedním krokem:
  $S_1 = S_0 \cup \{X \in (N \cup T)\ ;\ (S \to \alpha) \in P,\ |\alpha|_X \geq 1\}$
- V každém dalším kroku iterace přidáváme další symboly, které jsou v jednom kroku dosažitelné ze symbolů množiny z předchozího kroku:
  $S_i = S_{i-1} \cup \{X \in (N \cup T)\ ;\ (A \to \alpha) \in P,\ A \in S_{i-1},\ |\alpha|_X \geq 1\}$

- Pokud se už množina nemění (není co přidat), končíme:
  $S_i = S_{i-1} = S_{def}$

Stanovíme novou množinu neterminálů a terminálů: $N'' = N' \cap S_{def}$, $T' = T \cap S_{def}$. Nová množina pravidel bude obsahovat pouze ta pravidla, která mají na levé a pravé straně pouze symboly z množiny S_{def}:
$P'' = \left\{ (A \to \alpha) \in P' \ ; \ A \in S_{def}, \ \alpha \in S_{def}^* \right\}$ □

Příklad 2.5

Druhou část redukce (odstranění nedostupných symbolů) si ukážeme na této gramatice:
$G = (\{S, A, B, C\}, \ \{a, b, c\}, \ P, \ S)$
$S \to aA \mid bB \mid c$
$A \to cS \mid aA$
$B \to bB \mid cAB \mid b$
$C \to aA \mid b$

Gramatika zjevně neobsahuje žádné nadbytečné symboly, první část redukce tedy nemusíme provádět a pouze odstraníme nedostupné symboly. Sestrojíme množinu S_{def}.
$S_0 = \{S\}$ (báze)
$S_1 = \{S, a, A, b, B, c\}$ (na pravých stranách pravidel pro symbol S)
$S_2 = \{S, a, A, b, B, c, b\}$ (podle pravidel pro A a B)
$S_3 = S_2 = S_{def}$

Nová množina neterminálů je $N \cap S_{def} = \{S, A, B, C\} \cap \{S, a, A, b, B, c, b\} = \{S, A, B\}$, nová množina terminálů je $T \cap S_{def} = \{a, b, c\} \cap \{S, a, A, b, B, c, b\} = \{a, b, c\}$ (zde se nic nemění). Množina pravidel P' bude také protříděna, výsledná gramatika je následující:
$G' = (\{S, A, B\}, \ \{a, b, c\}, \ P', \ S)$
$S \to aA \mid bB \mid c$
$A \to cS \mid aA$
$B \to bB \mid cAB \mid b$

Důkaz (Odstranění nedostupných symbolů): Je dána gramatika $G' = (N', T, P', S)$ bez nadbytečných neterminálů. Výše uvedeným postupem konstrukce jsme sestrojili množinu S_{def} a gramatiku $G'' = (N'', T', P'', S)$.

Algoritmus konstrukce S_{def} je konečný ze stejného důvodu jako E_{def} (maximální počet kroků je roven počtu prvků množiny $N' \cup T$). To, že obsahuje právě ty symboly z původní množiny P', které jsou dosažitelné z počátečního stavu, plyne z algoritmu – v iterativním postupu přidáváme symboly podle pravidel gramatiky.

Zbývá dokázat ekvivalenci jazyků gramatik G' a G''.

Dokazujeme $L(G') \subseteq L(G'')$:
Vezměme jakékoliv slovo $w \in L(G')$.

$\Rightarrow$ existuje derivace slova w v gramatice G':
$S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$

$\Rightarrow$ do množiny S_{def} se řadí všechny dosažitelné symboly, tedy všechny symboly v použitých větných formách této derivace do ní patří,

$\Rightarrow$ z principu konstrukce pravidel gramatiky G' vyplývá, že pravidlo z množiny P' použité v kroku derivace $\alpha_{i-1} \Rightarrow \alpha_i,\ 1 \leq i \leq n$ existuje také v množině P'' a nebylo algoritmem odstraněno,

$\Rightarrow$ tatáž derivace slova w existuje i v gramatice G'',

$\Rightarrow$ $w \in L(G'')$.

Tvrzení platí i v případě, že $w = \varepsilon$, protože pokud $\varepsilon \in L(G')$, pak derivace $S \Rightarrow^* \varepsilon$ existuje v obou gramatikách a $\varepsilon \in L(G'')$.

Dokazujeme $L(G') \supseteq L(G'')$:
Tato část je triviální. V množině pravidel P'' jsou pouze ta pravidla, která existují v množině pravidel P'. Proto každá derivace v gramatice G'' existuje v téže formě i v gramatice G'.

Tím jsme dokázali ekvivalenci gramatik G' a G'' a korektnost a úplnost postupu redukce gramatiky. □

Příklad 2.6

Ukážeme si, proč je třeba nejdřív odstranit nadbytečné a až potom nedostupné symboly.
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to Aa \mid a$
$A \to ABc$
$B \to bB \mid b$
V prvním sloupci následující tabulky je správný postup (nejdřív odstraníme nadbytečné a pak nedostupné symboly), v druhém špatný postup (kdy tyto dva algoritmy zaměníme).

Správně:	**Špatně:**
$E_0 = T$ $E_1 = E_0 \cup \{S, B\}$ $E_2 = E_1,\ N' = \{S, B\}$ *První úprava:* $G' = (\{S, B\},\ \{a, b\},\ P',\ S)$ $S \to a$ $B \to bB \mid b$	$S_0 = \{S\}$ $S_1 = S_0 \cup \{A, a\}$ $S_2 = S_1 \cup \{B, c\}$ $S_3 = S_2 \cup \{b\}$ $S_4 = S_3,\ N' = \{S, A, B\}$ *První úprava:* $G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$ $S \to Aa \mid a$ $A \to ABc$ $B \to bB \mid b$
$S_0 = \{S\}$ $S_1 = S_0 \cup \{a\}$ $S_2 = S_1,\ N'' = \{S\}$ *Druhá úprava:* $G'' = (\{S\},\ \{a\},\ P'',\ S)$ $S \to a$	$E_0 = T$ $E_1 = E_0 \cup \{S, B\}$ $E_2 = E_1,\ N'' = \{S, B\}$ *Druhá úprava:* $G'' = (\{S, B\},\ \{a, b\},\ P'',\ S)$ $S \to a$ $B \to bB \mid b$

Jak vidíme, po úpravách v druhém sloupci nám v gramatice zůstal neterminál B, který je ve skutečnosti nedostupný ze startovacího symbolu.

2.2.3 Gramatika bez jednoduchých pravidel

Jednoduchá pravidla v (jakékoliv Chomského) gramatice jsou taková pravidla, kde na levé a pravé straně je jen jeden symbol, a to v obou případech neterminál.

Definice 2.6 (Jednoduchá pravidla, gramatika bez jednoduchých pravidel)

Jednoduchá pravidla v gramatice $G = (N, T, P, S)$ jsou pravidla ve tvaru $A \to B,\ A, B \in N$ (tedy na pravé i levé straně je jediný neterminál).

Gramatika bez jednoduchých pravidel je taková nezkracující bezkontextová gramatika, která neobsahuje žádná jednoduchá pravidla.

Tento typ pravidel nám v některých případech může vadit (například tehdy, když chceme mít co nejméně pravidel – jak vidíme, tato pravidla vlastně nic negenerují, jen „předávají štafetu“ k jinému neterminálu). Proto formulujeme následující větu:

Věta 2.3

Ke každé bezkontextové gramatice G lze sestrojit gramatiku bez jednoduchých pravidel G' takovou, že $L(G) = L(G')$.

Postup

Je dána *nezkracující* bezkontextová gramatika $G = (N, T, P, S)$, chceme sestrojit ekvivalentní gramatiku $G' = (N', T, P', S)$ bez jednoduchých pravidel.

Intuitivní postup by spočíval v nahrazení pravé strany jednoduchého pravidla pravými stranami pravidel neterminálu, který takto z pravé strany odstraníme, tedy například u pravidla $A \to B$ nahradíme B postupně všemi pravidly pro neterminál B, například jestliže je $B \to \alpha$, pak vznikne pravidlo $A \to \alpha$. Potom místo derivace $A \Rightarrow B \Rightarrow \alpha$ bude existovat derivace $A \Rightarrow \alpha$. Jenže takto bychom mohli za určitých okolností taky přenášet jednoduchá pravidla z jednoho místa na druhé, a tedy intuitivní postup bychom museli provádět rekurzívně tak dlouho, dokud nějaká jednoduchá pravidla existují. Proto budeme používat iterativní postup popsaný v dalších odstavcích.

Pro každý neterminál $A \in N$ sestrojíme množinu N_A takových neterminálů, na které lze tento neterminál přepsat pomocí jednoduchých pravidel. Například jestliže v gramatice máme pravidla $A \to B,\ B \to C$, pak by v množině N_A pro neterminál A byly prvky A, B, C.

Pak probereme postupně všechna pravidla z původní množiny pravidel P a všechna jednoduchá (jejich pravé strany) nahradíme řetězcem, na který se neterminál na levé straně postupně přepisuje (tj. přeskočíme kroky využívající jednoduchá pravidla). Toho docílíme tak, že ke každé

pravidlu $B \to \alpha$ z množiny P přidáme do množiny P' množinu pravidel $A \to \alpha$ pro všechny neterminály A takové, že $B \in N_A$. Postup ukazuje algoritmus 2.

Algoritmus 2: Odstranění jednoduchých pravidel v gramatice

Vstup : $G = (N, T, P, S)$ je nezkracující bezkontextová gramatika
Výstup: G' je gramatika bez jednoduchých pravidel taková, že
$L(G) = L(G')$

foreach $A \in N$ **do**
  $i := 0$;
  $N_{A,0} = \{A\}$;
  repeat
    $i := i + 1$;
    $N_{A,i} = N_{A,i-1} \cup \{X \in N \ ; \ (B \to X) \in P, \ B \in N_{A,i-1}\}$;
  until $N_{A,i} = N_{A,i-1}$;
  $N_A = N_{A,i}$;
end
$P' := \emptyset$;
foreach $(B \to \alpha) \in P$, *které není jednoduché,* **do**
  foreach $C \in N$ *takové, že* $B \in N_C$ **do**
    zařaď $(C \to \alpha) \in P'$;
  end
end
$G' = (N, T, P', S)$ je výsledná gramatika bez jednoduchých pravidel.

Jak vidíme, algoritmus mění pouze množinu pravidel, vše ostatní zůstává. V prvním cyklu **for** sestrojíme množiny N_A postupně pro všechny neterminály, v druhém cyklu **for** tvoříme novou množinu pravidel.

Příklad 2.7

Odstraníme jednoduchá pravidla v gramatice $G = (\{S, A, B\}, \ \{a, b, c\}, \ P, \ S)$ s pravidly
$S \to aAB \mid B \mid bb$
$A \to aA \mid bS \mid aa$
$B \to A \mid ba \mid c$

Podle postupu z algoritmu 2 nejdřív sestrojíme množiny N_A pro každý neterminál $A \in N$.

Množina neterminálů $N = \{S, A, B\}$ je tříprvková, sestrojíme množiny N_S, N_A, N_B.

$N_{S,0} = \{S\}$
$N_{S,1} = \{S\} \cup \{B\}$ podle $S \to B$
$N_{S,2} = \{S, B\} \cup \{A\}$ podle $B \to A$
$N_{S,3} = N_{S,2} = N_S = \{S, B, A\}$

$N_{A,0} = \{A\}$
$N_{A,1} = N_{A,0} = N_A = \{A\}$

$N_{B,0} = \{B\}$
$N_{B,1} = \{B\} \cup \{A\}$ podle $B \to A$
$N_{B,2} = N_{B,1} = N_B = \{B, A\}$

Pomocné množiny máme, teď projdeme všechna původní pravidla v množině P a určíme pravidla do množiny P'. Jednoduchá pravidla ignorujeme.

Pravidlo v P	Pravidla do P'	Protože
$S \to aAB,\ S \to bb$	$S \to aAB,\ S \to bb$	$S \in N_S$
$A \to aA,\ A \to bS,\ A \to aa$	$S \to aA,\ S \to bS,\ S \to aa$	$A \in N_S$
	$A \to aA,\ A \to bS,\ A \to aa$	$A \in N_A$
	$B \to aA,\ B \to bS,\ B \to aa$	$A \in N_B$
$B \to ba,\ B \to c$	$S \to ba,\ S \to c$	$B \in N_S$
	$B \to ba,\ B \to c$	$B \in N_B$

Srovnejme původní gramatiku G a výslednou gramatiku G' bez jednoduchých pravidel:

$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$ $\qquad$ $G' = (\{S, A, B\},\ \{a, b, c\},\ P',\ S)$

$S \to aAB \mid B \mid bb$ $\qquad$ $S \to aAB \mid bb \mid aA \mid bS \mid aa \mid ba \mid c$

$A \to aA \mid bS \mid aa$ $\qquad$ $A \to aA \mid bS \mid aa$

$B \to A \mid ba \mid c$ $\qquad$ $B \to aA \mid bS \mid aa \mid ba \mid c$

Příklad 2.8

Odstraníme jednoduchá pravidla z následující gramatiky:

$G = (\{E, T, F\},\ \{i, n, (,), +, *\},\ P,\ S)$

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid i \mid n$

Sestrojíme pomocné množiny pro jednotlivé neterminály:

$N_{E,0} = \{E\}$ $\qquad$ $N_{T,0} = \{T\}$ $\qquad$ $N_{F,0} = \{F\} = N_F$

$N_{E,1} = \{E, T\}$ $\qquad$ $N_{T,1} = \{T, F\} = N_T$

$N_{E,2} = \{E, T, F\} = N_E$

Neterminál F se nachází ve všech třech množinách, tedy pravidla pro přepis tohoto neterminálu použijeme v gramatice G' pro přepis všech tří neterminálů. Neterminál T najdeme v množinách N_E a N_T, proto pravidla původně přepisující neterminál T budou přepisovat neterminály E a T. Výsledná gramatika je následující:

$G' = (\{E, T, F\},\ \{i, n, (,), +, *\},\ P',\ S)$ s pravidly

$E \to E + T \mid T * F \mid (E) \mid i \mid n$

$T \to T * F \mid (E) \mid i \mid n$

$F \to (E) \mid i \mid n$

Důkaz (Věta 2.3): Vstupem výše uvedeného algoritmu je nezkracující bezkontextová gramatika $G = (N, T, P, S)$, výstupem gramatika bez jednoduchých pravidel $G' = (N, T, P', S)$. Je zřejmé, že algoritmus je konečný, protože počet opakování ve všech uvedených cyklech je limitován

počtem prvků v množinách N a P, přičemž obě množiny jsou konečné. Taky je zřejmé, že v množině pravidel P' nejsou žádná jednoduchá pravidla (algoritmus předepisuje jejich ignorování). Zbývá dokázat, že $L(G') = L(G)$.

Dokazujeme $L(G) \subseteq L(G')$:
Nechť $w \in L(G),\ |w| > 0$:

$\Rightarrow$ v G existuje derivace slova w o délce n: $S \Rightarrow^* w$. Označme
prvky posloupnosti derivace $S = w_0,\ w_1, \ldots,\ w_{n-1},\ w_n = w$,
pravidla použitá v jednotlivých krocích $A_i \to \alpha_i,\ A_i \in N,\ \alpha_i \in (N \cup T)^*,\ 0 \leq i \leq (n-1)$

Nyní sestrojíme ekvivalentní derivaci v gramatice G':

- v posloupnosti indexů $0, \ldots, (n-1)$ najdeme všechny souvislé (pod)posloupnosti indexů $j, \ldots, j+k$ takové, že $\alpha_j, \ldots, \alpha_{j+k} \in N$ (tj. v těch krocích jsou použita jednoduchá pravidla, pravé strany pravidel jsou tvořeny právě jedním neterminálem),
- pro každou související (pod)posloupnost použitých pravidel $A_j \to \alpha_j, \ldots, A_{j+k} \to \alpha_{j+k}$ a derivaci $w_j \Rightarrow \ldots \Rightarrow w_{j+k} \Rightarrow w_{j+k+1}$ najdeme ekvivalentní posloupnost pravidel a derivaci v gramatice G';

je zřejmé, že v takové derivaci se každé dvě sousední větné formy liší právě v jednom symbolu – neterminálu, přičemž minimálně poslední použité pravidlo není jednoduché (je terminální),

$\Rightarrow$ podle algoritmu musí pro každou takovou (pod)posloupnost indexů platit:
$\{\alpha_j, \ldots, \alpha_{j+k}\} = \{A_{j+1}, \ldots, A_{j+k+1}\} \subseteq N_{A_j}$

$\Rightarrow$ pro každou (pod)posloupnost existuje v G' pravidlo $A_j \to \alpha_{j+k+1}$

$\Rightarrow$ pro každou (pod)posloupnost existuje v G' derivace $w_j \Rightarrow w_{j+k+1}$

$\Rightarrow$ v G' existuje derivace slova w, proto $w \in L(G')$.

Dokazujeme $L(G) \supseteq L(G')$:
Nechť $w \in L(G'),\ |w| > 0$:

$\Rightarrow$ v G' existuje derivace slova w: $S \Rightarrow^* w$ ve formě
$S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = w$

označme pravidlo použité v j-tém kroku ($0 \leq j \leq n-1$) pravidlo $A \to \alpha$,

$\Rightarrow$ podle algoritmu existuje $B \in N_A$ (může platit také $B = A$), přičemž v G existuje pravidlo $B \to \alpha$ a derivace $A \Rightarrow^* B$

$\Rightarrow$ v G existuje derivace $A \Rightarrow^* B \Rightarrow \alpha$

$\Rightarrow$ v G existuje derivace slova w, proto $w \in L(G)$.

Zbývá případ, kdy $w = \varepsilon$, a tedy $|w| = 0$. Zde si stačí uvědomit, že po celou dobu pracujeme s nezkracující gramatikou, přičemž tuto vlastnost zachovává i algoritmus. Tedy platí:

$$\varepsilon \in L(G) \quad \Leftrightarrow \quad (S \to \varepsilon) \in P \quad \Leftrightarrow \quad (S \to \varepsilon) \in P' \quad \Leftrightarrow \quad \varepsilon \in L(G)' \qquad \square$$

2.2.4 Necyklické a vlastní gramatiky, substituce

Zatím víme, co je to bezkontextová redukovaná a nezkracující gramatika a gramatika bez jednoduchých pravidel. Dále definujeme ještě další speciální formy bezkontextových gramatik, které budeme využívat v dalších důkazech a postupech.

Definice 2.7 (Gramatika bez cyklu – necyklická)

Gramatika bez cyklu je gramatika, ve které neexistuje žádný symbol $A \in N$ takový, že $A \Rightarrow^+ A$.

Poznámka:

Nezkracující gramatika bez jednoduchých pravidel je vždy bez cyklu (pozor, implikace – mohou existovat gramatiky bez cyklu, které nejsou nezkracující nebo které obsahují jednoduchá pravidla).

Pokud tedy máme k dané gramatice G sestrojit ekvivalentní gramatiku bez cyklu, stačí ji převést do formy nezkracující gramatiky a odstranit jednoduchá pravidla.

Definice 2.8 (Vlastní gramatika)

Bezkontextová gramatika G se nazývá vlastní gramatikou, pokud je bez cyklu, nezkracující, redukovaná (bez nadbytečných symbolů).

Pokud máme k dané gramatice G sestrojit ekvivalentní vlastní gramatiku, převedeme ji do formy nezkracující gramatiky, redukujeme a odstraníme jednoduchá pravidla.

Následující lemma použijeme v důkazech některých dalších vět.

Lemma 2.4 (Lemma o substituci)

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Nechť $A \to \alpha B \beta$ je pravidlo v množině P a dále nechť $B \to \gamma_1 \mid \ldots \mid \gamma_n$ jsou všechna pravidla přepisující neterminál $B \in N$.

Označme gramatiku $G' = (N, T, P', S)$ takovou, kde

$$P' = (P - \{A \to \alpha B \beta\}) \cup \{A \to \alpha\gamma_1\beta \mid \ldots \mid \alpha\gamma_n\beta\} \tag{2.5}$$

Pak $L(G') = L(G)$.

Lemma je založeno na podobném principu, jaký jsme použili při odstraňování jednoduchých pravidel – jen na pravé straně pravidla máme kolem nahrazovaného neterminálu určitý kontext (okolí) navíc, který musí být zachován. V lemmatu o substituci však nemáme žádnou rekurzi, je definován postup pouze pro jeden krok, o to je důkaz jednodušší.

Důkaz: *Dokazujeme $L(G) \subseteq L(G')$:*
Pravidla $A \to \alpha B \beta$ jsou jedinými pravidly, která se nacházejí v P, ale nenacházejí se v G'. Jestliže tedy v gramatice G použijeme v některé derivaci pravidlo $A \to \alpha B \beta$, pak v některém z dalších kroků musí být takto vygenerovaný symbol B přepsán některým pravidlem $B \to \gamma_i$, tedy (až na posloupnost kroků) opět dostáváme derivaci slova w v gramatice G'.

Dokazujeme $L(G) \supseteq L(G')$:
Jestliže v derivaci v gramatice G' použijeme pravidlo $A \to \alpha\gamma_i\beta$, pak na stejném místě v ekvivalentní derivaci gramatiky G použijeme postupně pravidla $A \to \alpha B\beta$ a $B \to \gamma_i$, proto k derivaci $A \Rightarrow \alpha\gamma_i\beta$ v gramatice G' existuje ekvivalentní derivace $A \Rightarrow \alpha B\beta \Rightarrow \alpha\gamma_i\beta$ v gramatice G. □

2.2.5 Rekurze neterminálu v gramatice

V pravidlech bezkontextové gramatiky G máme levou rekurzi přes neterminál A, pokud v gramatice existuje derivace $A \Rightarrow^* A\alpha$. Obdobně, v gramatice máme pravou rekurzi přes neterminál A, pokud v ní existuje derivace $A \Rightarrow^* \alpha A$. Levá nebo pravá rekurze nám může vadit především v praktickém využití při programování, s tím se setkáme v předmětu *Překladače*.

Definice 2.9 (Gramatika bez levé rekurze, gramatika bez pravé rekurze)

Gramatika bez levé rekurze *je gramatika, ve které pro žádný neterminál* $A \in N$ *neexistuje derivace* $A \Rightarrow^+ A\alpha$.

Gramatika bez pravé rekurze *je gramatika, ve které pro žádný neterminál* $A \in N$ *neexistuje derivace* $A \Rightarrow^+ \alpha A$.

Přímá levá rekurze znamená existenci pravidla $A \to A\alpha$, nepřímá existenci pravidla $A \to \beta$, kde $\beta \Rightarrow^+ A\alpha$. Podobně pro pravou rekurzi.

Věta 2.5

Ke každé bc. gramatice G *existuje gramatika bez levé rekurze* G' *taková, že* $L(G) = L(G')$.

Postup (Odstranění přímé levé rekurze)

Je dána bezkontextová gramatika $G = (N, T, P, S)$. Předpokládejme, že množina pravidel

$$A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

je množinou všech pravidel přepisujících symbol $A \in N$, přičemž žádný z řetězců β_i nezačíná symbolem A (tj. rekurzivní zleva jsou pouze pravidla $A \to A\alpha_1 \mid \ldots \mid A\alpha_n$). Pro postup jsou dvě varianty. První variantu použijeme tehdy, když u gramatiky chceme zachovat vlastnost nezkracující gramatiky, druhou variantu použijeme, pokud nám ε-pravidla nevadí.

Varianta 1:

Sadu pravidel $\quad A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$

nahradíme sadou pravidel
$$\begin{aligned} A &\to \beta_1 B \mid \ldots \mid \beta_m B \mid \beta_1 \mid \ldots \mid \beta_m \\ B &\to \alpha_1 B \mid \ldots \mid \alpha_n B \mid \alpha_1 \mid \ldots \mid \alpha_n \end{aligned}$$

Proč to funguje:

- ukázka derivace v gramatice G: $A \Rightarrow A\alpha_3 \Rightarrow A\alpha_1\alpha_3 \Rightarrow A\alpha_8\alpha_1\alpha_3 \Rightarrow \beta_5\alpha_8\alpha_1\alpha_3$
- ukázka ekvivalentní derivace v gramatice G': $A \Rightarrow \beta_5 B \Rightarrow \beta_5\alpha_8 B \Rightarrow \beta_5\alpha_8\alpha_1 B \Rightarrow \beta_5\alpha_8\alpha_1\alpha_3$

Levou rekurzi jsme vlasně převedli na pravou rekurzi (což je v pořádku – „vadí“ nám vždy jen levá nebo pravá rekurze, ne obě zároveň) a přidali jsme jeden neterminál.

Varianta 2:

Sadu pravidel $A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$

nahradíme sadou pravidel
$$A \to \beta_1 B \mid \ldots \mid \beta_m B$$
$$B \to \alpha_1 B \mid \ldots \mid \alpha_n B \mid \varepsilon$$

Druhá varianta je vlastně obdobou první, přičemž díky použití ε-pravidla dostaneme menší počet pravidel, navíc nezhoršujeme nedeterminismus ve výběru pravidel (u první varianty jsme vždy vytvářeli dvojice pravidel přepisujících tentýž symbol, které začínaly stejným podřetězcem). Zatímco první varianta je výhodnější pro důkazy v oblasti teoretické informatiky, druhou variantu opět využijeme při praktickém použití při programování (předmět Překladače).

Algoritmus 3: Odstranění přímé levé rekurze

Vstup : $G = (N, T, P, S)$ je bezkontextová gramatika
Výstup: G' je gramatika bez přímé levé rekurze taková, že $L(G) = L(G')$
$P' := P$; $N' := N$;
foreach $A \in N$ **do**
  nechť $A \to A\alpha_1 \mid \ldots \mid A\alpha_n$ jsou všechna levě rekurzivní pravidla pro A;
  nechť $A \to \beta_1 \mid \ldots \mid \beta_m$ jsou všechna pravidla pro A bez levé rekurze;
  if $n > 0$ **then**
    odstraň z P' pravidla $A \to \alpha_1 \mid \ldots \mid \alpha_n$;
    přidej do P' pravidla $A \to \beta_1 A' \mid \ldots \mid \beta_m A'$;
    přidej do P' pravidla $A' \to \alpha_1 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$;
    $N' := N' \cup \{A'\}$;
  end
end
$G' = (N, T, P', S)$ je výsledná gramatika bez přímé levé rekurze.

Ve výše uvedeném algoritmu je zpracována druhá varianta – přidáváme ε-pravidla. Kdybychom chtěli vytvořit algoritmus pro první variantu, stačilo by upravit pouhé dva řádky.

Důkaz zde neuvedeme, protože je triviální – opět by spočíval v konstrukci derivace téhož slova v gramatikách G a G'.

Příklad 2.9

Odstraníme přímou levou rekurzi v pravidlech této gramatiky:
$G = (\{A, B, C\},\ \{a, b\},\ P,\ S)$
$A \to BC \mid a$
$B \to BaC \mid Ab \mid ba$
$C \to CC \mid b \mid Cb$

Přímá levá rekurze je rozpoznatelná na první pohled – levě rekurzivní jsou pravidla $B \to BaC$, $C \to CC$ a $C \to Cb$. Úprava se tedy bude týkat množin pravidel přepisujících neterminály B a C, vytvoříme dva nové neterminály B' a C'.

Pro neterminál B:			Varianta 1			Varianta 2
Sadu pravidel	B	$\to$	$BaC \mid Ab \mid ba$	B	$\to$	$BaC \mid Ab \mid ba$
nahradíme sadou pravidel	B	$\to$	$AbB' \mid baB' \mid Ab \mid ba$	B	$\to$	$AbB' \mid baB'$
	B'	$\to$	$aCB' \mid aC$	B'	$\to$	$aCB' \mid \varepsilon$
Pro neterminál C:			Varianta 1			Varianta 2
Sadu pravidel	C	$\to$	$CC \mid Cb \mid b$	C	$\to$	$CC \mid Cb \mid b$
nahradíme sadou pravidel	C	$\to$	$bC' \mid b$	C	$\to$	bC'
	C'	$\to$	$CC' \mid bC' \mid C \mid b$	C'	$\to$	$CC' \mid bC' \mid \varepsilon$

Výsledná gramatika pro každou z variant je tato:

$G' = (\{A, B, B', C, C'\},\ \{a, b\},\ P',\ S)$

$A \to BC \mid a$

$B \to AbB' \mid baB' \mid Ab \mid ba$

$B' \to aCB' \mid aC$

$C \to bC' \mid b$

$C' \to CC' \mid bC' \mid C \mid b$

$G'' = (\{A, B, B', C, C'\},\ \{a, b\},\ P'',\ S)$

$A \to BC \mid a$

$B \to AbB' \mid baB'$

$B' \to aCB' \mid \varepsilon$

$C \to bC'$

$C' \to CC' \mid bC' \mid \varepsilon$

Poznámka:

Výše je uveden postup a příklad na odstranění přímé levé rekurze. Pokud bychom chtěli odstranit přímou pravou rekurzi, opět by byly dvě varianty (podle toho, jestli nám víc vadí ε-pravidla nebo stejně *končící* pravidla přepisující tentýž neterminál), a převedli bychom ji na levou rekurzi:

Varianta 1:

Sadu pravidel $A \to \alpha_1 A \mid \ldots \mid \alpha_n A \mid \beta_1 \mid \ldots \mid \beta_m$

nahradíme sadou pravidel $A \to B\beta_1 \mid \ldots \mid B\beta_m \mid \beta_1 \mid \ldots \mid \beta_m$

$B \to B\alpha_1 \mid \ldots \mid B\alpha_n \mid \alpha_1 \mid \ldots \mid \alpha_n$

Varianta 2:

Sadu pravidel $A \to \alpha_1 A \mid \ldots \mid \alpha_n A \mid \beta_1 \mid \ldots \mid \beta_m$

nahradíme sadou pravidel $A \to B\beta_1 \mid \ldots \mid B\beta_m$

$B \to B\alpha_1 \mid \ldots \mid B\alpha_n \mid \varepsilon$

Zatím jsme si ukázali, jak si poradit s přímou rekurzí, zbývá nepřímá rekurze. Postup bude složitější, a musíme zajistit, aby byl konečný.

Postup (Odstranění levé rekurze)

Je dána *vlastní* gramatika $G = (N, T, P, S)$, chceme sestrojit gramatiku $G' = (N', T, P', S)$ bez levé rekurze takovou, že $L(G') = L(G)$. Pokud G není ve formě vlastní gramatiky, provedeme nejdřív transformaci, případně gramatiku redukujeme.

Na množině N definujeme uspořádání (tedy stanovíme pořadí prvků). Pokud $\mathrm{card}(N) = n$, můžeme označit $N = \{A_1, \ldots, A_n\}$ s pořadím dle indexů. Abychom postup nepopletli, je praktické

neterminály prostě přejmenovat na písmena s indexy, abychom měli jejich pořadí neustále na očích.

Postup spočívá v transformaci pravidel do takového tvaru, kde pravá strana pravidla může začínat neterminálem pouze tehdy, když tento neterminál má vyšší index podle stanoveného pořadí než neterminál na levé straně (který přepisujeme).

Při splnění této podmínky sice může být z neterminálu vygenerován řetězec začínající neterminálem, ale vždy jen takovým, který je „v pořadí dál“ – pokud existuje derivace $A \Rightarrow^* B\alpha$, pak jedině tehdy, jestliže B má vyšší index než A, a proto již pro žádný neterminál $A \in N$ nemůže nastat v derivaci cyklus $A \Rightarrow^* A\alpha$.

Procházíme postupně všechny neterminály $A_i \in \{A_1, \ldots, A_n\}$:

- zbavujeme se všech pravidel přepisujících A_i takových, jejichž pravá strana začíná neterminálem s indexem menším než i, tj. $A_i \to A_j\alpha$, kde $j < i$, a to s využitím lemmatu o substituci (v pravidle $A_i \to A_j\alpha$ nahradíme A_j postupně tím, na co lze A_j v jednom kroku přepsat) – postup je rekurzivní, provádíme ho tak dlouho, dokud existují taková pravidla,

Algoritmus 4: Odstranění levé rekurze

Vstup : $G = (N, T, P, S)$ je *vlastní* bezkontextová gramatika
Výstup: G' je gramatika bez levé rekurze taková, že $L(G) = L(G')$
$P' := P$; $N' := N$;
Urči uspořádání na N: $N = \{A_1, \ldots, A_n\}$;
for $i := 1$ **to** n **do** `// nejdřív odstraníme „levé cykly" v derivacích:`
  for $j := 1$ **to** $(i-1)$ **do**
    foreach $(A_i \to A_j\alpha) \in P'$ **do** `// všimněte si: pracujeme jen s` $j < i$
      nechť $A_j \to \beta_1 \mid \ldots \mid \beta_k$ jsou všechna pravidla pro A_j;
      odstraň $A_i \to A_j\alpha$ z P' ; `// lemma o substituci`
      přidej $A_i \to \beta_1\alpha \mid \ldots \mid \beta_k\alpha$ do P';
    end
  end
  if *pro* A_i *existuje pravidlo s přímou levou rekurzí* **then**
    `// odstraníme přímou rekurzi:`
    nechť $A_i \to \gamma_1 \mid \ldots \mid \gamma_p$ jsou všechna pravidla pro A_i taková, že pravá strana nezačíná symbolem A_i (bez přímé rekurze);
    foreach $(A_i \to A_i\alpha) \in P'$ **do**
      odstraň $A_i \to A_i\alpha$ z P';
      přidej $A'_i \to \alpha A'_i \mid \varepsilon$ do P';
      $N' := N' \cup \{A'_i\}$;
    end
    přidej $A_i \to \gamma_1 A'_i \mid \ldots \mid \gamma_p A'_i$ do P';
  end
end
$G' = (N, T, P', S)$ je výsledná gramatika bez levé rekurze.

- pokud existují nějaká pravidla $A_i \to A_i\alpha$ (tj. přímá levá rekurze), odstraníme rekurzi podle jedné ze dvou variant postupu odstranění přímé rekurze.

Postup je popsán v algoritmu 4, přičemž pro odstranění přímé rekurze je použita varianta 2 (připouštíme ε-pravidla).

2.3 Normální formy pro bezkontextové gramatiky

Normování znamená převod do takového tvaru, který je určitým způsobem standardizovaný. Pro bezkontextové gramatiky můžeme použít tyto normální formy:

1. Chomského normální forma (CNF),
2. Greibachové normální forma (GNF).

Jejich účel je podobný účelu dříve uvedených speciálních typů bezkontextových gramatik – jejich vlastnosti se nám za určitých okolností mohou hodit – například v důkazech nebo u praktického uplatnění při programování.

2.3.1 Chomského normální forma

Nejdřív se podíváme na Chomského normální formu. Pravé strany pravidel dodržujících tuto formu mají buď délku 2 a skládají se jen z neterminálů, nebo mají délku 1 a jsou terminální. Aby bylo možné v gramatice vygenerovat i prázdné slovo, je za určitých okolností povoleno i ε-pravidlo pro startovací symbol (podobně jako u nezkracujících gramatik).

Definice 2.10 (Chomského normální forma)

Bezkontextová gramatika $G = (N, T, P, S)$ je v Chomského normální formě (CNF), jestliže každé pravidlo z množiny P je v některém z těchto tvarů:

- $A \to BC, \quad A, B, C \in N,$
- $A \to a, \quad A \in N,\ a \in T.$

Dále může existovat pravidlo $S \to \varepsilon$ pro startovací symbol gramatiky S, jestliže se S nenachází na pravé straně žádného pravidla.

Věta 2.6 (Převod do Chomského normální formy)

Ke každé bezkontextové gramatice G existuje gramatika G' v CNF taková, že $L(G) = L(G')$.

Postup

Je dána gramatika $G = (N, T, P, S)$. Budeme chtít, aby byla ve tvaru vlastní gramatiky (tj. podle potřeby převedeme na nezkracující gramatiku a odstraníme jednoduchá pravidla).

Pravidla, která již vyhovují předpisu pro CNF (tj. ve tvaru $A \to BC$ nebo $A \to a$, případně ε-pravidlo pro startovací symbol), necháme jak jsou, všechna ostatní je třeba transformovat. Takže

nám zbývají pouze pravidla typu $A \to \alpha$, kde $|\alpha| = k > 1$ (je třeba si uvědomit, že v gramatice nejsou žádná jednoduchá ani ε-pravidla – případně kromě $S \to \varepsilon$).

Označme $A \to x_1 x_2 \dots x_k,\ x_i \in (N \cup T),\ 1 \leq i \leq k,\ k \geq 2$ pravidlo, které právě budeme zpracovávat. Na pravé straně pravidla máme řetězec symbolů (obecně terminálních i neterminálních), nejméně dva. Postupujeme takto:

1. zajistíme, aby na pravé straně pravidla byly pouze neterminály,
2. zajistíme, aby délka pravé strany byla 2 (tj. pravidla „rozsekáme“ do formy $A \to BC$),
3. přidáme nová pravidla podle potřeby.

Nejdřív první krok, paralelně budeme řešit i třetí krok. Všechny terminální symboly $a \in T$ na pravé straně pravidla nahradíme pomocnými neterminálními symboly N_a, které pro tento účel vytvoříme (tj. $N_a \notin N$, musí to být opravdu dosud nepoužité symboly). Takže například původní pravidlo $A \to bBacA$ transformujeme na $A \to N_b B N_a N_c A$. Následně musíme přidat nová pravidla $\forall a \in T\colon\ N_a \to a$, takže dle našeho příkladu původní derivaci $A \Rightarrow bBacA$ nahradíme derivací s více kroky: $A \Rightarrow N_b B N_a N_c A \Rightarrow bBN_a N_c A \Rightarrow bBaN_c A \Rightarrow bBacA$.

Takže zatím máme pravidlo ve formě $A \to X_1 X_2 \dots X_k$, kde pro $X_i \in N',\ 1 \leq i \leq k$ platí:

- pokud $x_i \in N$, pak $X_i = x_i$,
- pokud $x_i \in T$, pak $X_i = N_{x_i}$ vytvořený tak, jak je popsáno výše.

Pokud je $k = 2$, jsme s pravidlem hotovi. Jestliže je však $k > 2$, musíme v druhém kroku postupu toto pravidlo nahradit sadou pravidel, která budou mít na pravé straně právě dva neterminály a přitom budou generovat totéž: pravidlo $A \to X_1 X_2 \dots X_k$ nahradíme touto sadou pravidel:

$$\begin{aligned} A &\to X_1 H_1 \\ H_1 &\to X_2 H_2 \\ H_2 &\to X_3 H_3 \\ &\dots \\ H_{k-2} &\to X_{k-1} X_k \end{aligned}$$

Symboly H_j jsou opět všechny nově přidané, musíme použít odlišné i při zpracovávání různých původních pravidel. Takže derivace, která by po prvním kroku vypadala takto:
$A \Rightarrow X_1 X_2 \dots X_k$
bude po druhém kroku vypadat takto:
$A \Rightarrow X_1 H_1 \Rightarrow X_1 X_2 H_2 \Rightarrow X_1 X_2 X_3 H_3 \Rightarrow^* X_1 X_2 X_3 \dots X_{k-2} H_{k-2} \Rightarrow X_1 X_2 X_3 \dots X_{k-2} X_{k-1} X_k$

Výsledná gramatika G' bude mít pozměněnou množinu pravidel a také rozsáhlejší množinu neterminálů. Postup je podrobně ukázán v algoritmu 5 na straně 55.

Příklad 2.10

Následující gramatiku převedeme do Chomského normální formy.
$G = (\{S, A, B\}, \{0, 1\}, P, S)$
$S \to A \mid 0SA \mid \varepsilon$
$A \to 1A \mid 1 \mid B1$
$B \to 0B \mid 0 \mid 0SBA$

Algoritmus 5: Převod gramatiky do Chomského normální formy

Vstup : $G = (N, T, P, S)$ je *vlastní* bezkontextová gramatika
Výstup: G' je gramatika v Chomského NF taková, že $L(G) = L(G')$
$P' := \emptyset$; $N' := N$;
foreach $(A \to \alpha) \in P$ **do**
  if $(\alpha = a,\ a \in T)$ *nebo* $(\alpha = BC,\ B, C \in N)$ *nebo* $(A \to \alpha) = (S \to \varepsilon)$ **then**
    $P' := P' \cup \{(A \to \alpha)\}$; `// tato pravidla není třeba měnit`
    continue ; `// na začátek cyklu, další pravidlo`
  end
  označme $\alpha = x_1 x_2 \dots x_k,\ x_i \in (N \cup T),\ 1 \le i \le k$;
  for $i = 1$ ***to*** k **do**
    $X_i = \begin{cases} x_i, \text{ pokud } x_i \in N \\ N_{x_i}, \text{ pokud } x_i \in T;\ N_{x_i} \notin N \text{ (nový neterminál)} \end{cases}$
  end
  $\Rightarrow$ vytvořeno pravidlo $A \to X_1 X_2 \dots X_k$;
  if $k = 2$ **then**
    přidej do P' pravidlo $(A \to X_1 X_2)$; `// odpovídá CNF`
    continue ; `// na začátek cyklu, další pravidlo`
  end
  `// rozkouskujeme dlouhá pravidla, kde` $k > 2$:
  použij $H_1, \dots, H_{k-2} \notin N'$; `// tj. dosud nepoužité symboly`
  přidej do P' pravidla $(A \to X_1 H_1)$, $(H_{k-2} \to X_{k-1} X_k)$;
  if $k > 3$ **then**
    for $i = 1$ **to** $k - 3$ **do**
      přidej do P' pravidlo $(H_i \to X_{i+1} H_{i+1})$;
    end
  end
  $N' = N' \cup \{H_1, \dots, H_{k-2}\}$;
end
foreach $a \in T$ **do**
  přidej do P' pravidlo $N_a \to a$; $N' := N' \cup \{N_a\}$;
end
$G' = (N', T, P', S)$ je výsledná gramatika v Chomského normální formě.

Vstupem algoritmu má být gramatika ve formě vlastní gramatiky, což ta naše nesplňuje. Tedy je potřeba nejdřív provést příslušné transformace.

Předběžná úprava 1: převod na nezkracující gramatiku

$G' = (\{S', S, A, B\}, \{0, 1\}, P', S')$
$S' \to S \mid \varepsilon$
$S \to A \mid 0SA \mid 0A$
$A \to 1A \mid 1 \mid B1$
$B \to 0B \mid 0 \mid 0SBA \mid 0BA$

Předběžná úprava 2: odstranění jednoduchých pravidel

$G'' = (\{S', S, A, B\}, \{0, 1\}, P'', S')$

$S' \to 1A \mid 1 \mid B1 \mid 0SA \mid 0A \mid \varepsilon$

$S \to 1A \mid 1 \mid B1 \mid 0SA \mid 0A$

$A \to 1A \mid 1 \mid B1$

$B \to 0B \mid 0 \mid 0SBA \mid 0BA$

Redukovat tato gramatika nepotřebuje, tedy už je ve tvaru vlastní gramatiky a můžeme přikročit k použití algoritmu. Některá pravidla již požadavkům vyhovují, tedy následující pravidla jen převezmeme do výsledné množiny pravidel bez jakýchkoliv dalších úprav:

$S' \to 1 \mid \varepsilon$

$S \to 1$

$A \to 1$

$B \to 0$

Dále se zaměříme na pravidla, jejichž pravá strana má délku 2. Zde pouze provedeme jednoduchou transformaci – všechny terminály nahradíme příslušnými neterminály:

Původní:	Po nahrazení:
$S' \to 1A \mid B1 \mid 0A$	$S' \to N_1A \mid BN_1 \mid N_0A$
$S \to 1A \mid B1 \mid 0A$	$S \to N_1A \mid BN_1 \mid N_0A$
$A \to 1A \mid B1$	$A \to N_1A \mid BN_1$
$B \to 0B$	$B \to N_0B$

Pravidla pro transformaci přidáme do výsledné množiny pravidel.

Zbývají pravidla, jejichž pravá strana je delší než 2. Pravé strany transformujeme jako v předchozím případě (nahradíme terminály příslušnými neterminály) a upravíme je na délku 2.

Původní:	Po nahrazení terminálů:	Pravidla po zkrácení:		
$S' \to 0SA$	$S' \to N_0SA$	$S' \to N_0H_1$	$H_1 \to SA$	
$S \to 0SA$	$S \to N_0SA$	$S \to N_0H_2$	$H_2 \to SA$	
$B \to 0SBA$	$B \to N_0SBA$	$B \to N_0H_3$	$H_3 \to SH_4$	$H_4 \to BA$
$B \to 0BA$	$B \to N_0BA$	$B \to N_0H_5$	$H_5 \to BA$	

Vytvořená pravidla opět přidáme do výsledné množiny pravidel.

Dále vytvoříme pravidla pro nově zavedené neterminály N_a a zařadíme k ostatním:

$N_0 \to 0$

$N_1 \to 1$

Výsledná gramatika v Chomského normální formě:

$G''' = (\{S', S, A, B, N_0, N_1, H_1, H_2, H_3, H_4, H_5\}, \{0, 1\}, P''', S')$ s pravidly

$S' \to 1 \mid \varepsilon \mid N_1A \mid BN_1 \mid N_0A \mid N_0H_1$	$H_1 \to SA$	$H_5 \to BA$
$S \to 1 \mid N_1A \mid BN_1 \mid N_0A \mid N_0H_2$	$H_2 \to SA$	$N_0 \to 0$
$A \to 1 \mid N_1A \mid BN_1$	$H_3 \to SH_4$	$N_1 \to 1$
$B \to 0 \mid N_0B \mid N_0H_3 \mid N_0H_5$	$H_4 \to BA$	

Poznámka:

Na příkladu vidíme, že v konkrétních případech by se postup dal zoptimalizovat – například neterminály H_1 a H_2 generují totéž, a tedy je možné například všude místo H_2 použít H_1 (jsou zaměnitelné, podobně jako mohou být stavy konečného automatu, který je možné minimalizovat). Taktéž neterminály H_4 a H_5 jsou zaměnitelné. Optimálnější gramatika by vypadala takto:

$G''' = (\{S', S, A, B, N_0, N_1, H_1, H_3, H_4\}, \{0, 1\}, P''', S')$ s pravidly

$S' \to 1 \mid \varepsilon \mid N_1A \mid BN_1 \mid N_0A \mid N_0H_1$ $\quad$ $H_1 \to SA$ $\quad$ $N_0 \to 0$

$S \to 1 \mid N_1A \mid BN_1 \mid N_0A \mid N_0H_1$ $\quad$ $H_3 \to SH_4$ $\quad$ $N_1 \to 1$

$A \to 1 \mid N_1A \mid BN_1$ $\quad$ $H_4 \to BA$

$B \to 0 \mid N_0B \mid N_0H_3 \mid N_0H_4$

Nicméně, učíme se uplatňovat algoritmus a vyhýbat se chybám, proto je třeba s optimalizacemi zacházet velmi opatrně.

Důkaz (Převod do Chomského normální formy): Budeme používat značení stejné jako v algoritmu a postupu. Je třeba ukázat, že výsledná gramatika je v CNF, že algoritmus je konečný a že $L(G') = L(G)$.

To, že výsledná gramatika je v CNF, je zřejmé – transformujeme pouze pravidla, která v G nevyhovují CNF. Zaměníme terminály $a \in T$ na pravé straně za neterminály N_a, v dalším postupu je zajištěno, že délka pravé strany těchto pravidel je právě 2. Všechna přidávaná pravidla také vyhovují CNF.

Algoritmus je konečný, protože všechny cykly jsou prováděny v konečném počtu kroků – odvozeném buď z počtu pravidel, délky pravé strany pravidel nebo počtu terminálních symbolů.

Zbývá dokázat, že $L(G') = L(G)$. Zde si stačí uvědomit, jakým způsobem transformujeme pravidla. Pravidlo $A \to x_1 \dots x_k$ z množiny P je transformováno na sadu pravidel

$A \to X_1H_1$
$H_1 \to X_2H_2$
$H_2 \to X_3H_3$
$\dots$
$H_{k-2} \to X_{k-1}X_k$

Uvědomme si, že vlastně pořád *používáme variantu lemmatu o substituci*! Srovnejme derivace v gramatikách G a G':

v gramatice G: $\quad A \Rightarrow x_1x_2 \dots x_k$

v gramatice G': $\quad A \Rightarrow X_1H_1 \Rightarrow X_1X_2H_2 \Rightarrow^* X_1X_2 \dots X_k \Rightarrow x_1X_2 \dots X_k \Rightarrow^* x_1x_2 \dots x_k$

přičemž v posledních krocích používáme pravidla $X_i \to x_i$, pokud $x_i \in T$ (jestliže $x_i \in N$, pak samozřejmě takový krok neprovádíme).

Z toho plyne, že původní pravidla nahrazujeme sadou jiných pravidel takových, která generují tentýž řetězec (jen ve více krocích). Proto $L(G') = L(G)$. □

Poznámka:

Derivační strom gramatiky v Chomského normální formě má jednu velmi důležitou vlastnost – je binární (až na konce větví), každý vnitřní uzel má buď jediného potomka, který je listem, nebo

právě dva potomky, kteří mají potomky. Této vlastnosti se využívá nejen v různých algoritmech (i při programování), ale také v důkazech, protože se snadněji počítají různé číselné parametry s gramatikou související, včetně hloubky rekurze.

Příklad 2.11

Vytvoříme derivační strom uvedené derivace v následující gramatice. Je zřejmé, že gramatika je v CNF.

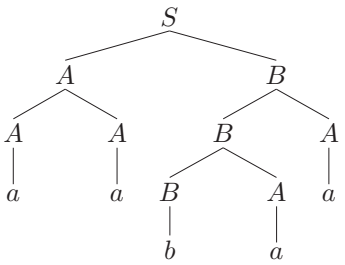
$G = (N, T, P, S)$

$S \to AB \mid a$

$A \to AA \mid a$

$B \to BA \mid b$

$S \Rightarrow AB \Rightarrow AAB \Rightarrow aAB \Rightarrow aaB \Rightarrow aaBA \Rightarrow$
$\Rightarrow aaBAA \Rightarrow aabAA \Rightarrow aabaA \Rightarrow aaabaa$



2.3.2 Greibachové normální forma

Greibachové normální forma (autorkou je Sheila Adele Greibach) předepisuje na pravé straně pravidla právě jeden terminál následovaný jakýmkoliv (konečným) množstvím neterminálů, což v sobě zahrnuje i terminální pravidla. Aby bylo možné do jazyka zařadit i prázdné slovo, připouští se i ε-pravidlo pro startovací symbol v případě, že se startovací symbol nenachází na pravé straně žádného pravidla.

Definice 2.11 (Greibachové normální forma)

Bezkontextová gramatika $G = (N, T, P, S)$ je v Greibachové normální formě (GNF), jestliže každé pravidlo z množiny P je v některém z těchto tvarů:

- $A \to aB_1 \dots B_n$, $n \geq 0$, $A, B_1, \dots B_n \in N$, $a \in T$,
- $S \to \varepsilon$, *jestliže S není na pravé straně žádného pravidla.*

Věta 2.7

Ke každé bezkontextové gramatice G existuje gramatika G' v GNF taková, že $L(G) = L(G')$.

Postup

Je dána bezkontextová gramatika $G = (N, T, P, S)$, která je ve tvaru vlastní gramatiky (tj. je třeba převést do tvaru nezkracující gramatiky, odstranit jednoduchá pravidla a redukovat) a bez levé rekurze (tedy v rámci přípravy odstraníme levou rekurzi. Levá rekurze v pravidlech je nepřípustná, protože by nebylo možné transformovat pravidla do tvaru, kde je prvním symbolem terminál. Sestrojíme gramatiku $G' = (N', T, P', S)$ v GNF takovou, že $L(G') = L(G)$.

Fáze 1 – terminál na začátku pravé strany pravidla:
Zajistíme, aby každé pravidlo začínalo terminálním symbolem. Fáze bude iterační, budeme ho provádět tak dlouho, dokud všechna pravidla nebudou vyhovovat této podmínce.

Označme pravidlo $A \to x_1 x_2 \dots x_k$. Jestliže $k \leq 1$ (pravidlo $X \to \varepsilon$ a pravidla, jejichž pravá strana je tvořena jedním terminálem), pak takové pravidlo můžeme zařadit do množiny P', vyhovuje GNF. V této fázi se tedy zaměříme na pravidla, kde $k \geq 2$ a $x_1 \in N$.

Použijeme lemma o substituci (str. 48) – neterminál x_1 nahradíme pravými stranami pravidel přepisujících tento symbol. Tedy pokud
$x_1 \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m$
jsou všechna pravidla přepisující X_1, pak pravidlo $A \to x_1 x_2 \dots x_k$ nahradíme sadou m pravidel
$A \to \alpha_1 \cdot x_2 \dots x_k \mid \alpha_2 \cdot x_2 \dots x_k \mid \dots \mid \alpha_m \cdot x_2 \dots x_k$
To provádíme tak dlouho, dokud v množině pravidel ještě jsou pravidla, jejichž pravá strana začíná neterminálem.

Algoritmus 6: Převod gramatiky do Greibachové normální formy

Vstup : $G = (N, T, P, S)$ je *vlastní* bezkontextová gramatika bez levé rekurze
Výstup: G' je gramatika v Greibachové NF taková, že $L(G) = L(G')$
$P' := P$; $N' := N$;
while *existuje* $(A \to x \cdot \beta) \in P'$, *kde* $x \in N$, $\beta \in (N \cup T)^*$ **do**
  nechť $x \to \gamma_1 \mid \dots \mid \gamma_m$, $m \geq 1$ jsou všechna pravidla pro x;
  odstraň z P' pravidlo $A \to x \cdot \beta$;
  přidej do P' pravidla $A \to \gamma_1 \cdot \beta \mid \dots \mid \gamma_m \beta$;
end
`// teď pravé strany všech pravidel začínají terminálem`
foreach $(A \to \alpha) \in P'$ **do**
  if $(\alpha = a,\ a \in T)$ *nebo* $(A \to \alpha) = (S \to \varepsilon)$ **then**
    continue ; `// neměníme; na začátek cyklu, další pravidlo`
  end
  označme $\alpha = x_1 x_2 \dots x_k$, $x_i \in (N \cup T)$, $1 \leq i \leq k$;
  for $i = 2$ ***to*** k **do**
    $X_i = \begin{cases} x_i, \text{ pokud } x_i \in N \\ \text{nový neterminál } N_{x_i} \notin N, \text{ pokud } x_i \in T;\ N' = N' \cup \{X_i\} \end{cases}$
  end
  $\Rightarrow$ vytvořeno pravidlo $A \to x_1 X_2 \dots X_k$;
  odstraň z P' pravidlo $A \to x_1 x_2 \dots x_k$;
  přidej do P' pravidlo $A \to x_1 X_2 \dots X_k$;
end
foreach $a \in T$ **do**
  přidej do P' pravidlo $N_a \to a$; $N' := N' \cup \{N_a\}$;
end
$G' = (N', T, P', S)$ je výsledná gramatika v Greibachové normální formě.

Fáze 2 – neterminály na pravých stranách pravidel:
Zatímco první symbol na pravé straně každého pravidla má být terminál, všechny ostatní symboly napravo od něj mají být neterminální. Použijeme stejný postup jako u Chomského normální formy, tedy vytvoříme „pomocné“ neterminály N_a pro všechny terminální symboly $a \in T$.

Všechna pravidla $A \to x_1 x_2 \dots x_k$, kde $k \geq 2$, transformujeme takto – pro všechna $2 \leq i \leq k$:

- pokud $x_i \in N$, pak $X_i = x_i$,
- pokud $x_i \in T$, pak $X_i = N_{x_i}$ vytvořený tak, jak je popsáno výše.

Transformací získáme pravidlo $A \to x_1 X_2 \dots X_k$, které již vyhovuje GNF, tedy je zařadíme do P'.

Zbývá přidat do množiny stavů P' pravidla pro nově vytvořené symboly – $N_a \to a$ pro každé $a \in T$.

Příklad 2.12

Následující gramatiku převedeme do Greibachové normální formy.

$G = (\{E, F\}, \{(,), +, i\}, P, E)$
$E \to E + F \mid F$
$F \to (E) \mid i$

Protože je vstupem algoritmu vlastní gramatika bez levé rekurze, provedeme příslušné transformace – odstraníme levou rekurzi a jednoduchá pravidla.

Předběžná úprava 1: odstranění levé rekurze
$G' = (\{E, E', F\}, \{(,), +, i\}, P', E)$
$E \to FE' \mid F$
$E' \to +FE' \mid +F$
$F \to (E) \mid i$

Předběžná úprava 2: odstranění jednoduchých pravidel
$G'' = (\{E, E', F\}, \{(,), +, i\}, P'', E)$
$E \to FE' \mid (E) \mid i$
$E' \to +FE' \mid +F$
$F \to (E) \mid i$

Začneme provádět algoritmus. Některá pravidla již vyhovují GNF, tedy je bez dalších úprav zařadíme do množiny P':
$E \to i$
$F \to i$

Jedno z pravidel začíná neterminálem, proto provedeme nahrazení podle věty o substituci:

Původní:	Po nahrazení:
$E \to FE'$	$E \to (E)E'$
	$E \to iE'$

Zbývá u všech pravidel, jejichž pravá strana je delší než 1, nahradit terminály příslušnými neterminály (až na první symbol pravé strany pravidla).

Původní:	Po nahrazení:
$E \to (E)E' \mid iE' \mid (E) \mid i$	$E \to (EN_)E' \mid iE' \mid (EN_) \mid i$
$E' \to +FE' \mid +F$	$E' \to +FE' \mid +F$
$F \to (E) \mid i$	$F \to (EN_) \mid i$

Ještě přidáme pravidla pro nový neterminál (použili jsme pouze jeden):
$N_) \to)$

Výsledná gramatika v Greibachové normální formě:
$G''' = (\{E, E', F, N_)\}, \{(,), +, i\}, P''', E)$
$E \to (EN_)E' \mid iE' \mid (EN_) \mid i$
$E' \to +FE' \mid +F$
$F \to (EN_) \mid i$
$N_) \to)$

Důkaz (Převod do Greibachové normální formy): Budeme používat značení stejné jako v algoritmu a postupu. Je třeba ukázat, že výsledná gramatika je v GNF, že algoritmus je konečný a že $L(G') = L(G)$.

To, že výsledná gramatika je v GNF, je zřejmé – transformujeme pouze pravidla, která v G nevyhovují GNF, přičemž podle věty o substituci dostáváme na pravé strany pravidel řetězce začínající terminálem, zbývající symboly (vlastně také s využitím věty o substituci, jen „opačně“) nahrazujeme pro tento účel vytvořenými neterminály. Všechna transformovaná i přidávaná pravidla vyhovují GNF.

Algoritmus je konečný, protože všechny cykly jsou prováděny v konečném počtu kroků: první cyklus (while) je konečný, protože v gramatice nemáme levou rekurzi (kdyby některé pravidlo bylo zleva rekurzivní, pak by tento cyklus šel do nekonečna), počet kroků v dalších cyklech je odvozen buď z počtu pravidel, délky pravé strany pravidel nebo počtu terminálních symbolů.

Zbývá dokázat, že $L(G') = L(G)$. Transformace umísťující terminál na začátek pravé strany pravidla nemění výsledný generovaný řetězec, což plyne z věty o substituci. Následná transformace taktéž nemění generovaný řetězec, pouze prodlužuje derivaci o kroky využívající nová pravidla $N_a \to a,\ a \in T$. Proto platí $L(G') = L(G)$. □

Poznámka:

Zamyslíme se nad tím, jak vlastně vypadá derivační strom některého slova v gramatice, která je v Greibachové normální formě. Sice není binární, ale zato má jinou zajímavou vlastnost – každý vnitřní uzel má právě jednoho potomka ohodnoceného terminálem, a to vždy toho, který je nejvíc vlevo. Zbývající potomci jsou vždy ohodnoceni neterminálem.

2.4 Uzávěrové vlastnosti bezkontextových jazyků

Zatímco třída regulárních jazyků je uzavřena vzhledem k prakticky jakékoliv běžně používané operaci, u třídy bezkontextových jazyků tomu tak v některých případech není. Na druhou stranu bývá konstrukce obvykle jednoduchá, důkaz také.

2.4.1 Sjednocení

Věta 2.8

Třída bezkontextových jazyků je uzavřena vzhledem k operaci sjednocení.

Postup

Jsou dány bezkontextové gramatiky $G_1 = (N_1, T_1, P_1, S_1)$ a $G_2 = (N_2, T_2, P_2, S_2)$. Je třeba, aby $N_1 \cap N_2 = \emptyset$. Pokud tomu tak není, musíme v jedné z gramatik přeznačit neterminály. Vytvoříme bezkontextovou gramatiku $G = (N, T_1 \cup T_2, P, S)$ takovou, že $L(G) = L(G_1) \cup L(G_2)$. Symbol S, který se má stát startovacím symbolem gramatiky, nesmí být použit v původních gramatikách: $S \notin N_1 \cup N_2$.

Výsledku docílíme velmi jednoduše – pravidla z obou původních gramatik přejmeme a přidáme dvě nová, které budou použita vždy na začátku derivace: $S \to S_1 \mid S_2$. Tedy
$P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}$.
Přidáme pouze jediný neterminál: $N = N_1 \cup N_2 \cup \{S\}$.

Požadavek na disjunktnost množin neterminálů původních gramatik je důležitý a je v úlohách typu „rozděl a panuj“ obvyklý – aby se nám přejaté derivační cesty nepomíchaly.

Příklad 2.13

Jsou dány tyto gramatiky:

$G_1 = (\{A, B, C\},\ \{a, b, c\},\ P_1,\ A)$ $\qquad$ $G_2 = (\{M, N, Q\},\ \{a, b, c, u, x\},\ P_2,\ M)$

$A \to aBA \mid CbB$ $\qquad$ $M \to aNc \mid \varepsilon$

$B \to bCb \mid c$ $\qquad$ $N \to PbxM \mid Qu \mid b$

$C \to cA \mid \varepsilon$ $\qquad$ $Q \to MN \mid ab$

Sestrojíme gramatiku G takovou, že $L(G) = L(G_1) \cup L(G_2)$.

$G = (\{S, A, B, C, M, N, Q\},\ a, b, c, u, x\},\ P,\ S)$

$S \to A \mid M$ $\qquad$ $M \to aNc \mid \varepsilon$

$A \to aBA \mid CbB$ $\qquad$ $N \to PbxM \mid Qu \mid b$

$B \to bCb \mid c$ $\qquad$ $Q \to MN \mid ab$

$C \to cA \mid \varepsilon$

Derivace jednoho ze slov v gramatice G_1:

$A \Rightarrow aBA \Rightarrow acA \Rightarrow acCbB \Rightarrow acbB \Rightarrow acbc$

Derivace jednoho ze slov v gramatice G_2:
$M \Rightarrow aNc \Rightarrow aQuc \Rightarrow aabuc$

Derivace obou těchto slov v gramatice G:
$S \Rightarrow A \Rightarrow aBA \Rightarrow acA \Rightarrow acCbB \Rightarrow acbB \Rightarrow acbc$
$S \Rightarrow M \Rightarrow aNc \Rightarrow aQuc \Rightarrow aabuc$

Důkaz (Uzavřenost třídy bezkontextových jazyků na sjednocení): Použijeme stejné značení jako ve výše uvedeném popisu konstrukce. Postup spočívá v přidání nového startovacího symbolu a dvou pravidel pro tento symbol, je tedy zjevně konečný. Dále dokážeme korektnost a úplnost postupu.

Dokazujeme $L(G_1) \cup L(G_2) \subseteq L(G)$:
Bez újmy na obecnosti vezmeme $w \in L(G_1)$ (důkaz pro slovo z jazyka $L(G_2)$ by byl obdobný):

- $\Rightarrow$ pro w existuje v G_1 derivace $S_1 \Rightarrow^* w$
- $\Rightarrow$ podle algoritmu lze v G sestrojit derivaci $S \Rightarrow S_1 \Rightarrow^* w$
- $\Rightarrow$ $w \in L(G)$

Dokazujeme $L(G_1) \cup L(G_2) \supseteq L(G)$:
Pro startovací symbol S existují pouze pravidla $S \to S_1 \mid S_2$. Vezměme některé $w \in L(G)$:

- $\Rightarrow$ pro w existuje v G derivace $S \Rightarrow S_1 \Rightarrow^* w$ nebo $S \Rightarrow S_2 \Rightarrow^* w$
- $\Rightarrow$ v prvním případě existuje v G_1 derivace $S_1 \Rightarrow^* w$ a platí $w \in L(G_1)$,
  v druhém případě existuje v G_2 derivace $S_2 \Rightarrow^* w$ a platí $w \in L(G_2)$
- $\Rightarrow$ $w \in L(G_1) \ \vee \ w \in L(G_2) \quad \Rightarrow \quad w \in L(G_1) \cup L(G_2)$ □

2.4.2 Zřetězení

Věta 2.9

Třída bezkontextových jazyků je uzavřena vzhledem k operaci zřetězení.

Postup

Jsou dány bezkontextové gramatiky $G_1 = (N_1, T_1, P_1, S_1)$ a $G_2 = (N_2, T_2, P_2, S_2)$. Opět budeme vyžadovat, aby $N_1 \cap N_2 = \emptyset$. Vytvoříme bezkontextovou gramatiku $G = (N, T_1 \cup T_2, P, S)$ takovou, že $L(G) = L(G_1) \cdot L(G_2)$. Symbol S, který se má stát startovacím symbolem gramatiky, nesmí být použit v původních gramatikách: $S \notin N_1 \cup N_2$.

Pravidla z obou původních gramatik přejmeme a přidáme jedno nové, které bude použito vždy na začátku derivace: $S \to S_1 \cdot S_2$. Tedy
$P = P_1 \cup P_2 \cup \{S \to S_1 \cdot S_2\}$.
Přidáme pouze jediný neterminál: $N = N_1 \cup N_2 \cup \{S\}$.

Příklad 2.14

Jsou dány tyto gramatiky:

$G_1 = (\{A, B, C\},\ \{a, b, c\},\ P_1,\ A)$ $\qquad$ $G_2 = (\{M, N, Q\},\ \{a, b, c, u, x\},\ P_2,\ M)$

$A \to aBA \mid CbB$ $\qquad$ $M \to aNc \mid \varepsilon$

$B \to bCb \mid c$ $\qquad$ $N \to PbxM \mid Qu \mid b$

$C \to cA \mid \varepsilon$ $\qquad$ $Q \to MN \mid ab$

Sestrojíme gramatiku G takovou, že $L(G) = L(G_1) \cdot L(G_2)$.

$G = (\{S, A, B, C, M, N, Q\},\ a, b, c, u, x\},\ P,\ S)$

$S \to AM$ $\qquad$ $M \to aNc \mid \varepsilon$

$A \to aBA \mid CbB$ $\qquad$ $N \to PbxM \mid Qu \mid b$

$B \to bCb \mid c$ $\qquad$ $Q \to MN \mid ab$

$C \to cA \mid \varepsilon$

Derivace jednoho ze slov v gramatice G_1:

$A \Rightarrow aBA \Rightarrow acA \Rightarrow acCbB \Rightarrow acbB \Rightarrow acbc$

Derivace jednoho ze slov v gramatice G_2:

$M \Rightarrow aNc \Rightarrow aQuc \Rightarrow aabuc$

Derivace zřetězení těchto slov v gramatice G:

$S \Rightarrow AM \Rightarrow aBAM \Rightarrow acAM \Rightarrow acCbBM \Rightarrow acbBM \Rightarrow acbcM \Rightarrow acbcaNc \Rightarrow$

$\Rightarrow acbcaQuc \Rightarrow acbcaabuc$

Důkaz (Uzavřenost třídy bezkontextových jazyků na zřetězení): Značení převezmeme z popisu konstrukce. Postup spočívá v přidání nového startovacího symbolu a jednoho pravidla pro tento symbol, je tedy konečný. Dokážeme korektnost a úplnost postupu.

Dokazujeme $L(G_1) \cdot L(G_2) \subseteq L(G)$:

Vezmeme $w_1 \in L(G_1)$ a $w_2 \in L(G_2)$:

- $\Rightarrow$ pro w_1 existuje v G_1 derivace $S_1 \Rightarrow^* w_1$,
  pro w_2 existuje v G_2 derivace $S_2 \Rightarrow^* w_2$
- $\Rightarrow$ podle algoritmu lze v G sestrojit derivaci $S \Rightarrow S_1 \cdot S_2 \Rightarrow^* w_1 \cdot w_2$
- $\Rightarrow$ $w_1 \cdot w_2 \in L(G)$

Dokazujeme $L(G_1) \cdot L(G_2) \supseteq L(G)$:

Pro startovací symbol S existuje pouze pravidlo $S \to S_1 \cdot S_2$. Vezměme některé $w \in L(G)$:

- $\Rightarrow$ pro w existuje v G derivace $S \Rightarrow S_1 \cdot S_2 \Rightarrow^* w$
  $\wedge$ S se nevyskytuje na pravé straně žádného pravidla
- $\Rightarrow$ v derivačním stromě k této derivaci existují dva oddělené podstromy pro S_1 a S_2, jejichž větné formy (v listech podstromů) jsou w_1 a w_2, tedy $w = w_1 \cdot w_2$
- $\Rightarrow$ lze najít k nim ekvivalentní derivace v původních gramatikách:
  v G_1: $S_1 \Rightarrow^* w_1$
  v G_2: $S_2 \Rightarrow^* w_2$
- $\Rightarrow$ $w_1 \in L(G_1) \ \wedge \ w_2 \in L(G_2) \quad \Rightarrow \quad w \in L(G_1) \cdot L(G_2)$

□

2.4.3 Iterace

Věta 2.10

Třída bezkontextových jazyků je uzavřena vzhledem k operaci iterace.

Postup

Je dána bezkontextová gramatika $G_1 = (N_1, T_1, P_1, S_1)$. Vytvoříme bezkontextovou gramatiku $G = (N, T_1, P, S)$ takovou, že $L(G) = L(G_1)^*$. Symbol S, který se má stát startovacím symbolem gramatiky, nesmí být použit v původní gramatice: $S \notin N_1$.

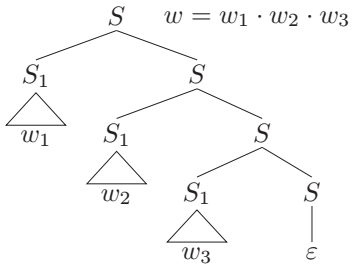
Pravidla z původní gramatiky přejmeme a přidáme dvě nová – jedno rekurzivní, které zajistí opakované řetězení slov původního jazyka, druhé ukončující rekurzi a případně generující prázdné slovo: $S \to S_1 S \mid \varepsilon$. Tedy
$P = P_1 \cup \{S \to S_1 S \mid \varepsilon\}$.
Přidáme pouze jediný neterminál: $N = N_1 \cup \{S\}$.

Příklad 2.15

Je dána tato gramatika:
$G_1 = (\{A, B, C\},\ \{a, b, c\},\ P_1,\ A)$
$A \to aBA \mid CbB \mid abc$
$B \to bCb \mid c$
$C \to cA \mid \varepsilon$
Sestrojíme gramatiku G takovou, že $L(G) = L(G_1)^*$.
$G = (\{S, A, B, C\},\ a, b, c\},\ P,\ S)$
$S \to AS \mid \varepsilon$
$A \to aBA \mid CbB \mid abc$
$B \to bCb \mid c$
$C \to cA \mid \varepsilon$
Derivace několika slov v gramatice G_1:
$A \Rightarrow aBA \Rightarrow acA \Rightarrow acCbB \Rightarrow acbB \Rightarrow acbc$
$A \Rightarrow CbB \Rightarrow cAbB \Rightarrow cCbBbB \Rightarrow cbBbB \Rightarrow cbcbB \Rightarrow cbcbc$
$A \Rightarrow abc$
Derivace sekvence těchto slov v gramatice G:
$S \Rightarrow AS \Rightarrow AAS \Rightarrow AAAS \Rightarrow AAA \Rightarrow$
$\Rightarrow aBAAA \Rightarrow acAAA \Rightarrow acCbBAA \Rightarrow acbBAA \Rightarrow acbcAA \Rightarrow$
$\Rightarrow acbcCbBA \Rightarrow acbccAbBA \Rightarrow acbccCbBbBA \Rightarrow acbccbBbBA \Rightarrow acbccbcbBA \Rightarrow acbccbcbcA \Rightarrow$
$\Rightarrow acbccbcbcabc$
Pro přehlednost jsme nejdřív vygenerovali posloupnost původních startovacích symbolů (pro tři slova jazyka $L(G_1)$) a následně jsme z těchto symbolů vygenerovali řetězce $acbc$, $cbcbc$ a abc.

Derivační strom takto vytvořené gramatiky bude mít směrem vpravo dolů „páteř“ – uzly ohodnocené symbolem S, od nichž půjdou směrem vlevo dolů podstromy pro slova z původní gramatiky. Páteř bude končit použitím pravidla $S \to \varepsilon$.

Vpravo vidíme schéma páteře derivačního stromu pro případ, kdy jsme iterovali tři slova původního jazyka.



Důkaz (Uzavřenost třídy bezkontextových jazyků na iteraci): Značení převezmeme z popisu konstrukce. Postup spočívá v přidání nového startovacího symbolu a dvou pravidel pro tento symbol, je tedy konečný. Dokážeme korektnost a úplnost postupu.

Dokazujeme $L(G_1)^* \subseteq L(G)$:
Vezmeme $w_1, \ldots, w_n \in L(G_1),\ n \geq 1$:

- $\Rightarrow$ pro všechna $w_i,\ 1 \leq i \leq n$, existuje v G_1 derivace $S_1 \Rightarrow^* w_i$
- $\Rightarrow$ podle algoritmu můžeme v G sestrojit (sub)derivaci $S \Rightarrow S_1S \Rightarrow S_1S_1S \Rightarrow^* S_1 \ldots S_1$, kde je ve větné formě celkem n symbolů S_1
- $\wedge\ P_1 \subset P$
- $\Rightarrow$ v derivaci lze pokračovat tak, že vždy i-tý výskyt S_i postupně zpracujeme ekvivalentně k derivaci slova w_i v gramatice G_1
- $\Rightarrow$ v G existuje derivace
  $S \Rightarrow^* S_1 \ldots S_1 \Rightarrow^* w_1 \cdot \ldots \cdot w_n$
- $\Rightarrow\ w_1 \cdot \ldots \cdot w_n \in L(G)$

Tento směr důkazu platí i pro $n = 0$ (tj. $w = \varepsilon$, protože toto slovo samozřejmě patří do $L(G_1)^*$), protože v G existuje pravidlo $S \to \varepsilon$.

Dokazujeme $L(G_1)^* \supseteq L(G)$:
Vezměme některé $w \in L(G)$ takové, že $|w| = k,\ k \geq 1$. Pro startovací symbol S existují pouze pravidla $S \to S_1S \mid \varepsilon$. Proto derivační strom takového slova w lze rozdělit na podstromy takové, že v kořeni každého takového podstromu je S_1 a nadřízeným uzlem ve stromě je některý uzel ohodnocený symbolem S (viz obrázek nahoře).

Počet těchto podstromů označíme číslem $n > 0$ a věty vygenerované v těchto podstromech postupně zleva $w_1, \ldots, w_n$. Z principu konstrukce derivačního stromu vyplývá, že celé vygenerované slovo je $w = w_1 \cdot \ldots \cdot w_n$.

V jednotlivých podstromech a ekvivalentních (pod)derivacích se dle algoritmu vyskytují pouze symboly z gramatiky G_1 (nikoliv symbol S) a jsou používána pouze pravidla z množiny P_1. Proto platí $w_i \in L(G_1),\ 1 \leq i \leq n$, a tedy $L(G) \subseteq L(G_1)^+$.

Zbývá dokázat tento směr tvrzení pro ε. To je triviální, protože vždy $\varepsilon \in L(G_1)^*$. □

2.4.4 Reverze

Věta 2.11

Třída bezkontextových jazyků je uzavřena vzhledem k operaci reverze (zrcadlení).

Postup

Je dána bezkontextová gramatika $G_1 = (N_1, T_1, P_1, S_1)$. Vytvoříme bezkontextovou gramatiku $G = (N_1, T_1, P, S_1)$ takovou, že $L(G) = L(G_1)^R$. Tentokrát nebudeme potřebovat žádný nový symbol, ani nebudeme přidávat pravidla, ale zato všechna pravidla transformujeme.

Budeme vycházet z definice reverze řetězce – v případě řetězce je třeba obrátit pořadí prvků tohoto řetězce. Reverze jazyka znamená provést tuto operaci na všech slovech daného jazyka. A protože slova jazyka $L(G_1)$ jsou generována pravidly z množiny P_1, přičemž tato pravidla určují i pořadí symbolů v generovaném slově, provedeme reverzi jazyka jednoduše reverzí pravých stran pravidel.

Pravidla, jejichž pravá strana je délky 0 nebo 1, beze změny přejmeme, s delšími pravidly provedeme následující: každé pravidlo $A \to \alpha,\ |\alpha| \geq 2$ nahradíme pravidlem $A \to \alpha^R$.

Příklad 2.16

Je dána tato gramatika:

$G_1 = (\{A, B, C\},\ \{a, b, c\},\ P_1,\ A)$

$A \to aBA \mid CbB \mid abc$

$B \to bCb \mid c$

$C \to cA \mid \varepsilon$

Sestrojíme gramatiku G takovou, že $L(G) = L(G_1)^R$ – revertujeme (zrcadlově převrátíme) všechna pravidla.

$G = (\{A, B, C\},\ a, b, c\},\ P,\ A)$

$A \to ABa \mid BbC \mid cba$

$B \to bCb \mid c$

$C \to Ac \mid \varepsilon$

Derivace dvou slov v gramatice G_1:

$A \Rightarrow aBA \Rightarrow acA \Rightarrow acCbB \Rightarrow acbB \Rightarrow acbc$

$A \Rightarrow abc$

Derivace reverzí těchto slov v gramatice G:

$A \Rightarrow ABa \Rightarrow Aca \Rightarrow BbCca \Rightarrow Bbca \Rightarrow cbca$

$A \Rightarrow cba$

Důkaz (Uzavřenost třídy bezkontextových jazyků na reverzi): Značení převezmeme z popisu konstrukce. Postup je konečný: spočívá v postupné transformaci pravidel (jichž je konečný počet a délka pravých stran pravidel je konečná). Dokážeme korektnost a úplnost postupu.

Dokazujeme $L(G_1)^R \subseteq L(G)$:
Vezmeme $w \in L(G_1)$, tedy $w^R \in L(G_1)^R$:

- $\Rightarrow$ v G_1 existuje derivace $S_1 \Rightarrow^* w$ o délce n (počet použitých pravidel), větné formy v derivaci označme postupně $\beta_0, \ldots, \beta_n$
- $\Rightarrow$ v v kroku derivace $\beta_i \Rightarrow \beta_{i+1},\ 0 \leq i \leq (n-1)$ bylo použito pravidlo, které označíme $A \to \alpha_i$
- $\wedge$ v gramatice G lze sestrojit odpovídající derivaci, ve které jsou používány zrcadlené ekvivalenty původních pravidel – $A \to \alpha_i^R$, jednotlivé větné formy označme $\beta'_0, \ldots, \beta'_n$

$\Rightarrow$ jsou použita pravidla $A \to \alpha_i^R$; na pravých stranách pravidel zůstaly všechny neterminály, jen byly přeuspořádány, tedy posloupnost použitých pravidel (transformovaných) odpovídá, v každém kroku jsou vygenerovány tytéž symboly, jen v opačném pořadí

$\Rightarrow$ $\beta_i' = \beta_i^R,\ 0 \leq i \leq n$

$\Rightarrow$ derivace v G je $S_1 = \beta_0^R \Rightarrow \beta_1^R \Rightarrow^* \beta_n^R = w^R$

$\Rightarrow$ $w^R \in L(G)$

Dokazujeme $L(G_1)^R \supseteq L(G)$:
Tento směr je triviální (resp. úplně stejný jako opačný směr), protože operace reverze řetězců je duální (sama k sobě inverzní): $(w^R)^R = w$. □

Poznámka:

Pokud sestrojíme derivační stromy odpovídajících si derivací v původní a vytvořené gramatice, pak i tyto stromy budou navzájem zrcadlově obrácené.

2.4.5 Průnik a doplněk

Na rozdíl od třídy regulárních jazyků není třída bezkontextových jazyků uzavřena vzhledem k operacím průniku a doplňku (ale je uzavřena vzhledem k průniku s regulárním jazykem, což si dokážeme, až nastudujeme zásobníkové automaty). Uvedeme si důkazy těchto tvrzení – první důkaz bude spočívat v nalezení protipříkladu, v druhém důkazu využijeme De Morganovy zákony.

Věta 2.12

Třída bezkontextových jazyků není *uzavřena vzhledem k operaci průniku.*

Důkaz: Najdeme dva bezkontextové jazyky, jejichž průnik není bezkontextovým jazykem – tím dokážeme, že třída bezkontextových jazyků není uzavřena vzhledem k operaci průniku. Vezměme tyto dva jazyky:

$L_x = \{a^i b^i c^k\ ;\ i,k \geq 0\}$ počet symbolů a a b ve slově je stejný

$L_y = \{a^i b^k c^k\ ;\ i,k \geq 0\}$ počet symbolů b a c ve slově je stejný

Oba tyto jazyky jsou bezkontextové – sestrojíme gramatiky, které je generují:

$G_x = (\{M,N,P\},\ \{a,b,c\},\ P_x,\ M)$

$M \to NP$

$N \to aNb \mid \varepsilon$

$P \to cP \mid \varepsilon$

$G_y = (\{R,S,T\},\ \{a,b,c\},\ P_y,\ R)$

$R \to ST$

$S \to aS \mid \varepsilon$

$T \to bTc \mid \varepsilon$

Průnikem těchto jazyků je $L = L_x \cap L_y = \{a^i b^i c^i\ ;\ i \geq 0\}$, který však není bezkontextový (dá se dokázat například pomocí Pumping lemma pro bezkontextové jazyky). □

Věta 2.13

Třída bezkontextových jazyků není *uzavřena vzhledem k operaci doplňku jazyka v dané abecedě.*

Důkaz: Provedeme důkaz sporem. Předpokládejme, že třída bezkontextových jazyků je uzavřena vzhledem k doplňku v abecedě.

Podle De Morganových zákonů platí pro jakékoliv dva jazyky (množiny řetězců) L_x a L_y

$$L_x \cap L_y = \overline{\overline{L_x} \cup \overline{L_y}}$$

Jestliže jazyky L_x a L_y jsou bezkontextové, pak v případě, že třída $\mathscr{L}$ je uzavřena vzhledem k doplňku, bychom na pravé straně výrazu měli také bezkontextový jazyk. Podívejme se však na levou stranu – víme, že třída bezkontextových jazyků není uzavřena vzhledem k průniku, tedy na levé straně nemáme zaručenu existenci bezkontextového jazyka ⇒ spor. □

2.4.6 Homomorfismus a substituce

Bezkontextová substituce s uplatněná na jazyk L je takové zobrazení, které zobrazuje každý symbol abecedy tohoto jazyka na *bezkontextový jazyk* a přitom platí homomorfní podmínky:

- $s(\varepsilon) = \varepsilon$
- $s(a \cdot v) = s(a) \cdot s(v),\ a \in (N \cup T),\ v \in (N \cup T)^*$

Homomorfismus je speciálním případem substituce, kdy symboly abecedy zobrazujeme na řetězce, nikoliv jazyky (a tedy množiny řetězců). Tedy pokud dokážeme, že třída bezkontextových jazyků je uzavřena vzhledem k substituci, dokážeme tím také, že je uzavřena vzhledem k homomorfismu.

Věta 2.14

Třída bezkontextových jazyků je uzavřena vzhledem k operaci bezkontextové substituce.

Postup

Je dán bezkontextový jazyk L nad abecedou $\Sigma = \{a_1, a_2, \ldots, a_n\}$ a gramatika $G = (N, \Sigma, P, S)$ taková, že $L(G_1) = L_1$.

Dále jsou dány bezkontextové jazyky $L_{a_1}, L_{a_2}, \ldots, L_{a_n}$ nad abecedami $\Sigma_{a_1}, \Sigma_{a_2}, \ldots, \Sigma_{a_n}$ a bezkontextové gramatiky, které je generují: $G_{a_i} = (N_{a_i}, \Sigma_{a_i}, P_{a_i}, a_i),\ 1 \leq i \leq n$. Předpokládejme, že množiny neterminálů N_{a_i} jsou vždy po dvou disjunktní a také jsou disjunktní s množinou N.

Určíme substituci jako zobrazení s: $\Sigma \to \bigcup_{i=1}^{n} \Sigma_{a_i}$, stanovíme $s(a_i) = L_{a_i},\ 1 \leq i \leq n$. Protože zobrazení zachovává homomorfní podmínky, definice plně postačuje k určení jazyka $L' = s(L)$.

Sestrojíme gramatiku $G' = (N', \bigcup_{i=1}^{n} \Sigma_{a_i}, P', S)$ generující jazyk L':

- $N' = N \cup \Sigma \cup \bigcup_{i=1}^{n} N_{a_i}$ – původní terminály se stanou neterminály, také přidáme neterminály z gramatik G_{a_i}
- $P' = P \cup \bigcup_{i=1}^{n} P_{a_i}$ – jednoduše sjednotíme pravidla všech „zúčastněných“ gramatik

Příklad 2.17

Je dán následující jazyk a gramatika, která tento jazyk generuje:

$L = \{a^i b^j c^k \ ; \ i,j,k \geq 1, \ i = j \text{ nebo } j = k\}$

$G = (N, \Sigma, P, S)$, kde je $N = \{S, A, B, X, Y\}$, $\Sigma = \{a, b, c\}$ a množina P:

$S \to AX \mid YB$

$A \to aAb \mid ab$

$B \to bBc \mid bc$

$X \to cX \mid c$

$Y \to aY \mid a$

Dále určíme substituci s, a to stanovením zobrazení jednotlivých prvků abecedy Σ.

$s(a) = L_a = \{1^n \ ; \ n \geq 0\}$ $\quad$ $s(b) = L_b = \{1^n 0^n \ ; \ n \geq 1\}$ $\quad$ $s(c) = L_c = \{0^n \ ; \ n \geq 0\}$

Pro tyto jazyky taky sestrojíme bezkontextové gramatiky:

$G_a = (\{a\}, \ \{1\}, \ P_a, \ a)$ $\quad$ $G_b = (\{b\}, \ \{1,0\}, \ P_b, \ b)$ $\quad$ $G_c = (\{c\}, \ \{0\}, \ P_c, \ c)$

$a \to 1a \mid \varepsilon$ $\quad$ $b \to 1b0 \mid 10$ $\quad$ $c \to 0c \mid \varepsilon$

Vytvoříme gramatiku G' generující jazyk $s(L)$, tedy $L(G') = s(L)$.

$G' = (\{S, A, B, X, Y, a, b, c\}, \ \{0, 1\}, \ P', \ S)$ s množinou P':

$S \to AX \mid YB$ $\quad$ $X \to cX \mid c$ $\quad$ $a \to 1a \mid \varepsilon$

$A \to aAb \mid ab$ $\quad$ $Y \to aY \mid a$ $\quad$ $b \to 1b0 \mid 10$

$B \to bBc \mid bc$ $\quad$ $c \to 0c \mid \varepsilon$

Generovaný jazyk je $L = s(L_1) = 1^* \cdot \{1^n 0^n \mid n \geq 1\}^* \cdot 0^*$

Důkaz (Uzavřenost třídy bezkontextových jazyků na substituci): Značení převezmeme z popisu konstrukce. Postup je konečný, protože prakticky jen sjednocujeme konečné množiny. Dokážeme korektnost a úplnost postupu.

Dokazujeme $s(L(G)) \subseteq L(G')$:

Vezmeme $w \in L(G)$, $|w| = k \geq 1$, tedy $s(w) \subseteq s(L(G))$. Je třeba si uvědomit, že $s(w)$ není slovo, ale jazyk (množina slov). Označme symboly ve slově

$w = a_1 \ldots a_n, \ a_i \in (N \cup T), \ 1 \leq i \leq k.$

$\Rightarrow$ v G existuje derivace slova w: $S \Rightarrow^* w$

$\wedge$ zobrazení s stanovuje $s(a_i) = L_{a_i}, \ 1 \leq i \leq k$, přičemž jazyky L_{a_i} jsou generovány bezkontextovými gramatikami G_{a_i}

$\Rightarrow$ sestrojíme jazyk $s(w) = s(a_1) \cdot s(a_2) \cdot \ldots \cdot s(a_n)$; podle algoritmu lze v gramatice G' všechna slova tohoto jazyka generovat následovně:

$S \Rightarrow^* a_1 a_2 \ldots a_k \Rightarrow$ $\quad$ použijeme pravidla převzatá z G

$\Rightarrow^* s(a_1) a_2 \ldots a_k \Rightarrow$ $\quad$ použijeme pravidla převzatá z G_{a_1}

$\Rightarrow^* s(a_1) s(a_2) \ldots a_k \Rightarrow$ $\quad$ použijeme pravidla převzatá z G_{a_2}

$\vdots$

$\Rightarrow^* s(a_1) s(a_2) \ldots s(a_k) = s(w)$ $\quad$ použijeme pravidla převzatá z G_{a_k}

$\Rightarrow$ $s(w) \subseteq L(G')$

Poznámka:

Zápis $\ldots \Rightarrow^* s(a_1)\ldots$ apod. není ve skutečnosti zcela korektní, protože $s(a_1)$ není symbol ani řetězec, ale množina, tento způsob zápisu byl zvolen pouze za účelem přehlednosti a zdůraznění vztahu k odvození těchto slov.

Dokazujeme $s(L(G)) \supseteq L(G')$:
Vezměme slovo $w \in L(G'),\ |w| \geq 1$:

- $\Rightarrow$ v G' existuje derivace $S \Rightarrow^* w$, přičemž podle algoritmu vypadá derivační strom příslušný k této derivaci následovně:
  - každou cestu v tomto stromě vedoucí od kořene (ohodnoceného S) k listu (ohodnocenému některým terminálem z množiny $\bigcup_{i=1}^{n} \Sigma_{a_i}$) lze rozdělit na dvě části tak, že
    - na hranici těchto dvou částí je uzel ohodnocený některým symbolem a z množiny Σ (terminál v původní gramatice G),
    - v první části cesty jsou pouze uzly ohodnocené neterminály gramatiky G,
    - v druhé části cesty jsou pouze uzly ohodnocené symboly z množiny $N_a \cup \Sigma_a$ (list je ohodnocen symbolem z Σ_a, ostatní uzly druhé části symboly z množiny N_a)
  - v derivačním stromě určíme podstromy takové, že v každém podstromě jsou sdruženy všechny výše popsané cesty derivačního stromu, které se shodují v první části cesty a hraničním uzlu, v kořeni podstromu je (hraniční) uzel ohodnocený symbolem z Σ
- $\Rightarrow$ označení listů kteréhokoliv takto označeného podstromu (jehož kořen je ohodnocen $a \in \Sigma$) čtené zleva doprava dává slovo jazyka L_a
- $\Rightarrow$ pokud tyto podstromy odstraníme (necháme jen jejich původní kořenové uzly), získáme derivační strom, jehož uzly jsou ohodnoceny pouze symboly z množiny $N \cup \Sigma$
- $\Rightarrow$ ze způsobu konstrukce pravidel gramatiky G' (pravidla přejatá z G) vyplývá, že takto upravený derivační strom odpovídá derivaci některého slova $u \in L$ ve tvaru $S \Rightarrow^* u$
- $\wedge$ každý symbol ze slova u je startovacím symbolem v některé z gramatik $G_{a_i},\ 1 \leq i \leq n$
- $\Rightarrow$ $s(u) \subseteq s(L(G))$

Pro $|w| = 0$ je důkaz triviální – jestliže $\varepsilon \in L(G)$, pak podle homomorfních podmínek musí také platit $\varepsilon \in L(G')$, a naopak. □

2.5 Kritéria bezkontextovosti

2.5.1 Využití uzávěrových vlastností bezkontextových jazyků

Podobně jako u regulárních jazyků, i u bezkontextových jazyků můžeme uzávěrové vlastnosti využít jako kritérium příslušnosti jazyka do dané třídy.

Příklad 2.18

Jazyk $L = \left\{a^{i_1}b^{i_1}a^{i_2}b^{i_2}\ldots a^{i_k}b^{i_k}\ ;\ i_1,\ldots,i_k \geq 0,\ k \geq 1\right\}$ je možné reprezentovat jako k-násobné zřetězení jazyka $L = \{a^n b^n\ ;\ n \geq 0\}$. Proto je tento jazyk bezkontextový.

Příklad 2.19

Jazyk $L = \{a^i b^j c^k \ ; \ i, j, k \geq 1, \ i = j \text{ nebo } j = k\}$ je sjednocením těchto dvou jazyků:

- $L_1 = \{a^i b^j c^k \ ; \ i, j, k \geq 1, \ i = j\}$
- $L_2 = \{a^i b^j c^k \ ; \ i, j, k \geq 1, \ j = k\}$

Jazyky L_1 a L_2 jsou bezkontextové, proto i jazyk L je bezkontextový.

Jako kritérium nepříslušnosti do třídy bezkontextových jazyků se hodně používá operace průniku s regulárním jazykem, se kterou se setkáme až v následující kapitole u zásobníkových automatů.

2.5.2 Pumping lemma pro bezkontextové jazyky

Dále budeme vycházet z toho, že už ovládáme Pumping lemma pro regulární jazyky. U bezkontextových jazyků bude princip podobný.

U regulárních jazyků jsme si smysl lemmatu vysvětlili na rekurzi (cyklech) v konečném automatu, v případě bezkontextových jazyků budeme hledat rekurzi v pravidlech gramatiky. Pokud existuje derivace
$A \Rightarrow^+ yAu \Rightarrow^* yzu$, kde $y, u, z \in T^*$,
je zřejmé, že neterminál A je rekurzivní a s jeho použitím lze vygenerovat i velmi dlouhá slova. „Pumpování“ bude v takovém případě následující:
$A \Rightarrow^+ y_1 A u_1 \Rightarrow^* y_1 y_2 A u_2 u_1 \Rightarrow^* y_1 y_2 y_3 A u_3 u_2 u_1 \Rightarrow^* \ldots \Rightarrow^* y_1 y_2 y_3 \ldots z \ldots u_3 u_2 u_1$,
tedy „pumpujeme“ neterminál a jeho podstrom.

Vezměme si jakýkoliv neterminál $A \in N$, který je rekurzivní. Pak derivace, v jejíž některé větné formě se vyskytuje symbol A a používá rekurzi, bude vypadat takto:
$S \Rightarrow^* xAv \Rightarrow^* xyAuv \Rightarrow^* xyzuv$

V poslední větné formě je vidět rozdělení celého generovaného slova na pět částí:

- části x a v, které jsou v druhé uvedené větné formě před a za symbolem A,
- části y a u, které jsou v třetí uvedené větné formě před a za symbolem A a zjevně pochází právě z rekurzivní subderivace symbolu A: $A \Rightarrow^* yAu$,
- část z, která je vygenerována ze subderivace symbolu A až po rekurzi: $A \Rightarrow^* z$

Také zde si určíme, co budeme rozumět pod pojmem *dostatečně dlouhé* slovo – pokud se jedná o bezkontextový jazyk a my umíme sestrojit ekvivalentní bezkontextovou gramatiku, pak dostatečně dlouhé slovo bude slovo delší než počet neterminálů vynásobený délkou pravé strany toho pravidla gramatiky, jehož pravá strana je nejdelší.

Bez újmy na obecnosti předpokládejme, že gramatika generující bezkontextový jazyk L je v *Chomského normální formě*.

Lemma 2.15

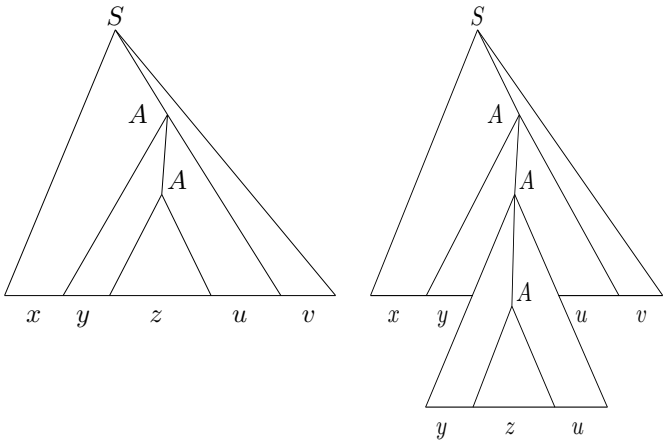
Pokud $G = (N, T, P, S)$ je bezkontextová gramatika v Chomského normální formě, pak derivační strom každého slova $w \in L(G)$ je binární až na hrany k listům na koncích větví stromu a platí $|w| \leq 2^{h-1}$, kde h je počet uzlů na nejdelší cestě z kořene do některého listu v derivačním stromě.

Důkaz: To, že derivační strom gramatiky v CNF je binární až na hrany k listům, je zřejmé, protože všechna pravidla přepisující neterminály jsou ve tvaru $A \to BC,\ A, B, C \in N$ (tj. každý vnitřní uzel má právě dva potomky).

Následující plyne právě z toho, že se jedná o binární strom (kromě listů): pokud by byl derivační strom slova w vyvážený, tj. všechny cesty v jeho grafu by byly stejně dlouhé (označme počet uzlů na takové cestě h), pak by délka slova w byla přesně 2^{h-1} (jedničku odečteme, protože při vytváření posledního patra používáme „nebinární“ pravidla $A \to a,\ A \in N,\ a \in T$). Obvykle však derivační strom nebývá vyvážený, tedy pokud je h délka nejdelší větve, pak je číslo 2^{h-1} horním omezením délky slova w. □

Označme $n = \mathrm{card}(N)$ je počet neterminálů gramatiky. Potřebujeme, aby některá cesta od kořene k listu byla delší než n, aby se na této cestě vyskytoval některý neterminál více než jednou, proto budeme chtít, aby naše „dostatečně dlouhé“ slovo w bylo delší než 2^{n-1}. Tím zajistíme, že alespoň jeden neterminál se v derivaci bude chovat rekurzívně (právě na této cestě).

Situace je znázorněna na dvojici obrázků vpravo. Symbol $A \in N$ je rekurzivní (ať už jde o přímou nebo nepřímou rekurzi), přičemž jsme při odvozování použili (nejméně) jednou rekurzivní posloupnost pravidel přepisující symbol A a druhý zobrazený výskyt symbolu A je zpracován nerekurzivní posloupností pravidel.



Na druhém obrázku je ukázán případ, kdy jsme při přepsání druhého znázorněného symbolu A použili opět rekurzivní posloupnost pravidel a až na následný vygenerovaný výskyt symbolu A nerekurzivní posloupnost.

V prvním případě je možné rozdělit vygenerované slovo na pět částí $w = x \cdot y \cdot z \cdot u \cdot v$ tak, jak je naznačeno, v druhém případě je druhá a čtvrtá část „pumpována“ díky opětovnému použití rekurze.

Věta 2.16 (Pumping lemma pro bezkontextové jazyky)

Nechť L je bezkontextový jazyk. Pak existují přirozená čísla p a q taková, že pro každé slovo $w \in L$, $|w| > p$ existuje alespoň jedno rozdělení na pět částí $w = x \cdot y \cdot z \cdot u \cdot v$, přičemž

- $|y \cdot u| > 0$ *(v alespoň jedné z těchto dvou částí musí být alespoň jeden symbol),*
- $|y \cdot z \cdot u| \leq q$ *(prostřední část má omezenou délku),*
- $x \cdot y^k \cdot z \cdot u^k \cdot v \in L$ *pro každé $k \geq 0$.*

Všimněte si, že i zde se nám střídají kvantifikátory:
$\exists$ čísla $p, q \ldots \quad \forall w \in L,\ |w| > p \quad \exists$ rozdělení $\ldots \quad \forall k \geq 0\ x \cdot y^k \cdot z \cdot u^k \cdot v \in L.$

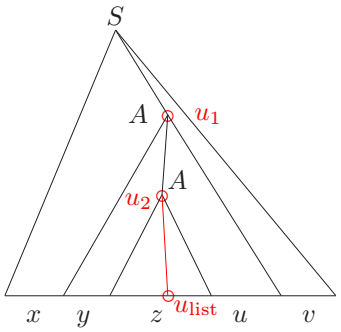
U regulárních jazyků jsme dělili slovo na tři části a jedna (prostřední) část byla pumpována, u bezkontextových jazyků dělíme slovo na pět částí, přičemž druhá a čtvrtá část jsou pumpovány.

Jinak je smysl podobný, včetně daných omezujících podmínek – dostatečná délka slova, neprázdnost pumpované části, horní omezení délky té části slova, které se týká rekurze (pozor: nejen pumpované části).

Podívejme se na význam podmínek určených v Pumping lemma vzhledem k formě pravidel gramatiky a tvaru derivačního stromu. Připomeňme, že pracujeme s gramatikou v CNF.

- $|w| > p$ jsme si vysvětlili, budeme chtít, aby p zajišťovalo existenci cesty v derivačním stromě delší než n, proto můžeme stanovit $p = 2^{n-1}$,
- $|y \cdot u| > 0$ je důležité, abychom měli co „pumpovat“ – kdybychom připustili $y \cdot u = \varepsilon$, pak bychom dovolili používat jednoduchá pravidla, která však nevyhovují CNF,
- $|y \cdot z \cdot u| \leq q$ – v derivačním stromě na té cestě od kořene k listu, na níž se opakuje některý neterminál, vybereme ten výskyt dotyčného neterminálu, který je nejblíže listu; můžeme použít například $q = 2^n$.

Důkaz (Pumping lemma pro bezkontextové jazyky): Jedná se o implikaci. Vycházíme z předpokladu, že L je bezkontextový jazyk, máme dokázat, že existují čísla p a q vyhovující uvedeným podmínkám.



Důkaz opět povedeme tak, že zvolíme hodnoty těchto čísel a dokážeme, že tyto hodnoty odpovídají požadavku Pumping lemmatu. Vybereme si v derivačním stromě dostatečně dlouhého slova takovou cestu, která bude obsahovat (ke svému konci blízko listu) dva uzly označené tímtéž neterminálem, což znamená, že na první výskyt byla uplatněna rekurze, kdežto na druhý ne. Pak provedeme úpravu derivačního stromu – na druhý výskyt použijeme stejnou posloupnost pravidel jako na první, čímž zajistíme pumpování (pro $k = 2$).

L je bezkontextový jazyk:

$\Rightarrow$ existuje bezkontextová gramatika $G = (N, T, P, S)$ v Chomského NF taková, že $L = L(G)$; určíme $p = 2^{\text{card}(N)-1}$, $q = 2^{\text{card}(N)}$ a vezmeme jakékoliv slovo $w \in L$ takové, že $|w| > p$

$\Rightarrow$ v derivačním stromě slova w existuje cesta delší než $\text{card}(N)$, na níž označíme tyto uzly (jako na obrázku vpravo nahoře):

- u_{list} je list na konci této cesty, je to list ohodnocený terminálem,
- u_1 a u_2 jsou označeny neterminálem $A \in N$,
- u_1 je blíže kořenu, u_2 je blíže listu,
- cesta z u_1 do u_{list} má délku nejvýše $\text{card}(N) + 1$ (tedy na cestě mezi nimi zřejmě není žádná dvojice uzlů, které by byly stejně označené, a jediným uzlem na této cestě s ohodnocením A je u_2).

Je zřejmé, že uzel u_2 je kořenem podstromu, jehož listy tvoří podslovo z, tedy reprezentuje derivaci $A \Rightarrow^* z$, uzel u_1 je kořenem podstromu, jehož listy tvoří podslovo yzu, a reprezentuje derivaci $A \Rightarrow^* yAu \Rightarrow^* yzu$. Derivace z kořene stromu je $S \Rightarrow^* xAv \Rightarrow^* xyAuv \Rightarrow^* xyzuv$.

$\Rightarrow$ Protože je G v CNF a víme, že cesta od u_1 do u_{list} má délku nejvýše $\text{card}(N) + 1$, je délka slova yzu shora omezena číslem $2^{\text{card}(N)}$ (od délky cesty jsme odečetli 1, list), čímž máme zdůvodněnu volbu q.

$\wedge$ v gramatice G, která je v CNF, nejsou žádná jednoduchá ani ε-pravidla, tedy ve slově derivovaném z A v uzlu u_1 vždy získáme slovo, kde $|y \cdot u| > 0$.

$\Rightarrow$ Nyní upravíme derivační strom a příslušnou derivaci, vytvoříme následující varianty:

1. podstrom uzlu u_1 nahradíme podstromem uzlu u_2, tedy na symbol A v uzlu u_1 uplatníme místo rekurzivního zpracování nerekurzivní zpracování jako v uzlu u_2,
   odpovídající derivace: $S \Rightarrow^* xAv \Rightarrow^* xzv = xy^0zu^0v$,
2. podstrom uzlu u_2 nahradíme kopií podstromu uzlu u_1, tedy na symbol A v uzlu u_2 uplatníme rekurzivní zpracování stejně jako v uzlu u_1,
   odpovídající derivace: $S \Rightarrow^* xAv \Rightarrow^* xyAuv \Rightarrow^* xy^2Au^2v \Rightarrow^* xy^2zu^2v$,
3. výsledek předchozího kroku zpracujeme naprosto stejným způsobem, tedy nerekurzivní podstrom nahradíme kopií rekurzivního podstromu z nejbližšího vyššího uzlu označeného neterminálem A,
   odpovídající derivace: $S \Rightarrow^* xAv \Rightarrow^* xyAuv \Rightarrow^* xy^2Au^2v \Rightarrow^* xy^3Au^3v \Rightarrow^* xy^3zu^3v$,

k. atd., celkem k-krát – derivace pak mají formu $S \Rightarrow^* xy^kAu^kv \Rightarrow^* xy^kzu^kv$.

$\Rightarrow$ pokud dokážeme zkonstruovat derivaci a derivační strom, pak výsledné slovo také patří do jazyka: $xy^kzu^kv \in L(G),\ k \geq 0$. □

Příklad 2.20

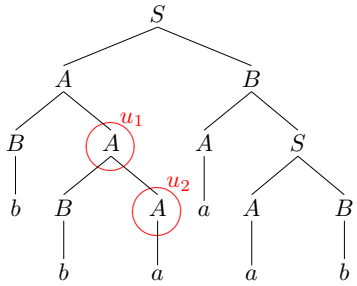
Význam tvrzení v Pumping lemmatu si ukážeme na následující gramatice (jak vidíme, je v CNF):

$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$, v množině P jsou pravidla

$S \to AB$

$A \to BA \mid a$

$B \to AS \mid b$



$n = \text{card}(N) = 3$, proto stanovíme $p = 2^2 = 4$, $q = 2^3 = 8$. Potřebujeme slovo delší než 4, například $bbaaab$. Derivace tohoto slova je následující:

$S \Rightarrow AB \Rightarrow BAB \Rightarrow bAB \Rightarrow bBAB \Rightarrow bbAB \Rightarrow bbaB \Rightarrow bbaAS \Rightarrow bbaaS \Rightarrow bbaaAB \Rightarrow$
$\Rightarrow bbaaaB \Rightarrow bbaaab$

Vpravo je derivační strom této derivace. Uzly u_1 a u_2 bychom mohli vybrat více způsoby, zde jsme se rozhodli pro rekurzi v neterminálu A (na cestě v grafu stromu jsme našli dva „poslední“ výskyty tohoto neterminálu). Jinou volbou by mohly být uzly označené neterminálem B nebo S (obojí na cestě z kořenového uzlu zcela vpravo).

V našem případě jsme zvolili rozdělení na pět částí $w = b \cdot b \cdot a \cdot \varepsilon \cdot aab$, ve zkráceném zápisu derivace: $S \Rightarrow^* b \cdot A \cdot aab \Rightarrow^* b \cdot b \cdot A \cdot \varepsilon \cdot aab \Rightarrow^* b \cdot b \cdot a \cdot \varepsilon \cdot aab$

Upravíme derivační strom a derivaci a vytvoříme varianty podle Pumping lemmatu:

$k = 0$: $S \Rightarrow^* b \cdot A \cdot aab \Rightarrow^* b \cdot a \cdot aab = b \cdot b^0 \cdot a \cdot \varepsilon^0 \cdot aab$

$k = 1$: $S \Rightarrow^* b \cdot A \cdot aab \Rightarrow^* b \cdot b \cdot A \cdot \varepsilon \cdot aab \Rightarrow^* b \cdot b \cdot a \cdot \varepsilon \cdot aab$

$k = 2$: $S \Rightarrow^* b \cdot A \cdot aab \Rightarrow^* b \cdot b \cdot A \cdot \varepsilon \cdot aab \Rightarrow^* b \cdot b^2 \cdot A \cdot \varepsilon^2 \cdot aab \Rightarrow^* b \cdot b^2 \cdot a \cdot \varepsilon^2 \cdot aab$

$k = 3$: $S \Rightarrow^* b \cdot A \cdot aab \Rightarrow^* b \cdot b \cdot A \cdot \varepsilon \cdot aab \Rightarrow^* b \cdot b^2 \cdot A \cdot \varepsilon^2 \cdot aab \Rightarrow^* b \cdot b^3 \cdot A \cdot \varepsilon^3 \cdot aab \Rightarrow^* b \cdot b^3 \cdot a \cdot \varepsilon^3 \cdot aab$

obecně: $S \Rightarrow^* b \cdot A \cdot aab \Rightarrow^* b \cdot b^k \cdot A \cdot \varepsilon^k \cdot aab \Rightarrow^* b \cdot b^k \cdot a \cdot \varepsilon^k \cdot aab, \quad k \geq 0$

Příklad 2.21

Je dán jazyk $L = \{a^n b^n \ ; \ n \geq 1\}$. Ukážeme, že tento jazyk splňuje podmínky Pumping lemmatu pro bezkontextové jazyky. Nebudeme zde uvádět gramatiku a ani s ní nebudeme nijak počítat. Hodnoty p a q nebudeme stanovovat přesně, jen budeme počítat s tím, že jde o „dostatečně velká“ čísla.

Vybereme „dostatečně dlouhé“ slovo $w \in L$, například $w = a^i b^i$ pro nějaké dostatečně velké číslo i. Je třeba najít nějaké rozdělení tohoto slova na pět částí splňující podmínky Pumping lemmatu.

Část y ani část u musí obsahovat buď jen a nebo jen b, protože jinak bychom pumpováním získalí střídavě symboly a a b, což neodpovídá předpisu jazyka. Víme, že celá prostřední část yzu má délku shora omezenou, tedy se zde nebude vyskytovat index i. Protože slova jazyka zachovávají určitou symetrii, musíme tutéž symetrii použít i při určování rozdělení (yzu zasahuje do obou polovin slova w). V následující tabulce je naznačeno vhodné rozdělení:

x	y	z	u	v	**Podmínky**	**Pumpování**
a^{i-r-s}	a^r	$a^s b^s$	b^r	b^{i-s-r}	$2 \cdot r > 0, \quad 2 \cdot r + 2 \cdot s \leq q$	$a^{i-r+k \cdot r} b^{i-r+k \cdot r}$

Podařilo se nám najít takové rozdělení slova w na části $w = x \cdot y \cdot z \cdot u \cdot v$, které splňuje podmínky Pumping lemmatu, včetně možnosti pumpování $x \cdot y^k \cdot z \cdot u^k \cdot v \in L \quad \forall k \geq 0$.

Poznámka:

Také tvrzení v Pumping lemmatu pro bezkontextové jazyky je implikací, proto se jedná pouze o *nutnou*, nikoliv *postačující* podmínku bezkontextovosti. Proto lemma typicky používáme ve formě „když jazyk nemá danou vlastnost, není bezkontextový“, stejně jako u Pumping lemmatu pro regulární jazyky.

Příklad 2.22

Je dán jazyk $L = \left\{c^{2^m} a^n b^n \ ; \ m, n \geq 1\right\}$. Tento jazyk sice není bezkontextový, ale přesto splňuje podmínku stanovenou v Pumping lemmatu. Pro slovo $w = c^{2^i} a^j b^j$ pro dostatečně velká čísla i, j existuje například rozdělení $c^{2^i} a^{j-r} \cdot a^r \cdot \varepsilon \cdot b^r \cdot b^{j-r}$, přičemž $c^{2^i} a^{j-r} \cdot a^{k \cdot r} \cdot \varepsilon \cdot b^{k \cdot r} \cdot b^{j-r} \in L \ \forall k \geq 0$. Proto (stejně jako u regulárních jazyků) nemůžeme Pumping lemma použít jako *postačující* podmínku bezkontextovosti.

Postup (Důkaz, že jazyk není bezkontextový pomocí Pumping lemma)

Postup vychází z možností ekvivalentních úprav logických výrazů:

$$(A \to B) \quad \Longleftrightarrow \quad (\neg B \to \neg A)$$

Podrobněji k části výrazu B s kvantifikátory:

$L \in \mathscr{L}(CF) \Rightarrow \exists p, q \in \mathbb{N} \ \ \forall w: \ |w| > p \ \ \exists \text{ rozdělení} \ldots \ \ \forall k \geq 0: \ x \cdot y^k \cdot z \cdot u^k \cdot v \in L$

$\forall p, q \in \mathbb{N} \ \ \exists w: \ |w| > p \ \ \forall \text{ rozdělení} \ldots \ \ \exists k \geq 0: \ x \cdot y^k \cdot z \cdot u^k \cdot v \notin L \Rightarrow L \notin \mathscr{L}(CF)$

Z toho vyplývá, že budeme postupovat takto:

- vybereme dostatečně dlouhé slovo $w \in L$,
- stanovíme strukturu tohoto slova a určíme možná rozdělení vyhovující podmínkám $w = x \cdot y \cdot z \cdot u \cdot v \quad \wedge \quad y \cdot u \neq \varepsilon \quad \wedge \quad |y \cdot z \cdot u| \leq q$,
- ukážeme, že žádné z těchto rozdělení neodpovídá podmínce $\forall k \geq 0$ je $x \cdot y^k \cdot z \cdot u^k \cdot v \in L$, tedy pro každé rozdělení najdeme číslo k takové, že $x \cdot y^k \cdot z \cdot u^k \cdot v \notin L$ (obvykle stačí vyzkoušet $k = 0$ nebo $k = 2$),
- pokud se to nepodaří, vracíme se k prvnímu bodu a hledáme další dostatečně dlouhé slovo.

Příklad 2.23

Ukážeme, že zadaný jazyk není bezkontextový: $L = \{a^n b^n c^n \ ; \ n \geq 0\}$

Zvolíme slovo $w = a^i b^i c^i$ pro dostatečně velké číslo i. Možná rozdělení tohoto slova jsou v tabulce. q je konečné číslo, v části yzu se proto nesmí vyskytovat „potenciálně nekonečné“ i a tato část se bude pohybovat buď v první nebo třetí třetině slova, nebo na rozhraních mezi třetinami. Pokud je na rozhraní, pak v rámci část y opět nemůžeme míchat dva různé typy symbolů, protože při pumpování by se tyto symboly střídaly, totéž platí o části u.

x	y	z	u	v	**Podmínky**	xy^izu^iv	**pro** $k = 0$
xyz je v první třetině slova: $a^{i-r-s-t-m}$	a^r	a^s	a^t	$a^m b^i c^i$	$r+t > 0$, $r+s+t \leq q$	$a^{i+k(r+t)-r-t} b^i c^i$	$a^{i-r-t} b^i c^i \notin L$
xyz je v třetí třetině slova: $a^i b^i c^m$	c^r	c^s	c^t	$c^{i-m-r-s-t}$	$r+t > 0$, $r+s+t \leq q$	$a^i b^i c^{i+k(r+t)-r-t}$	$a^i b^i c^{i-r-t} \notin L$
xyz je na rozhraní první a druhé třetiny: a^{i-r-s}	a^r	$a^s b^t$	b^m	$b^{i-t-m} c^i$	$r+m > 0$, $r+s+t+m \leq q$	$a^{i+kr-r} b^{i+kt-t} c^i$	$a^{i-r} b^{i-t} c^i \notin L$
xyz je na rozhraní druhé a třetí třetiny: $a^i b^{i-r-s}$	b^r	$b^s c^t$	c^m	c^{i-t-m}	$r+m > 0$, $r+s+t+m \leq q$	$a^i b^{i+kr-r} c^{i+kt-t}$	$a^i b^{i-r} c^{i-t} \notin L$

Ve variantě uvedené v prvním řádku bychom také mohli dosadit $r = 0$ nebo $s = 0$ (jen jedno z toho, jinak by rozdělení nevyhovovalo podmínkám Pumping lemmatu), případně $m = 0$, ale na výsledku by se tím nic nezměnilo. Podobně lze tvořit „podvarianty“ i pro další řádky, ale byla by to práce navíc, pro důkaz naprosto dostačují tyto čtyři varianty.

V každé variantě jsme našli číslo k, pro které je $x \cdot y^k \cdot z \cdot u^k \cdot v \notin L$, proto $L \notin \mathscr{L}(CF)$.

Příklad 2.24

Ukážeme, že zadaný jazyk není bezkontextový: $L = \left\{a^{n^2} \ ; \ n \geq 0\right\}$ Tentokrát zvolíme výpočetní postup s řešením soustavy (ne)rovnic.

Zvolíme $w = a^{i^2} \in L$ s dostatečně velkým indexem i. Slovo rozdělíme: $a^{i^2} = a^{x_1} a^{x_2} a^{x_3} a^{x_4} a^{x_5}$. Aby toto rozdělení splňovalo podmínky Pumping lemmatu, musí platit tyto vztahy:

$$x_1 + x_2 + x_3 + x_4 + x_5 \quad = \quad i^2 \tag{2.6}$$

$$x_2 + x_4 > 0 \tag{2.7}$$

$$x_2 + x_3 + x_4 \leq q \tag{2.8}$$

$$\forall k \geq 0 \;\; \exists r > 0: \quad x_1 + k \cdot x_2 + x_3 + k \cdot x_4 + x_5 = r^2 \tag{2.9}$$

Úpravou poslední rovnice získáme $k(x_2 + x_4) + x_1 + x_3 + x_5 = r^2$. Je třeba si uvědomit, že v této rovnici jsou proměnnými k a r, se vším ostatním zacházíme jako s konstantami, protože se pro dané slovo w nemění. Podívejme se, co je na levé a pravé straně této rovnice:

- $k(x_2 + x_4) + x_1 + x_3 + x_5$ je lineární funkce o proměnné k,
- r^2 je funkce s exponentem, která roste výrazně rychleji než lineární.

Dále víme, že číslo $x_2 + x_4$ u lineárního členu levé strany rovnice je určitě různé od nuly (to vyplývá ze vztahu (2.7), tedy se rozhodně nejedná o konstantní funkci.

Levá a pravá strana se sice mohou rovnat pro některá konkrétní čísla, která bychom dosadili za k a r, ale obecně vztah (2.9) nemůže být platný (představte si graf lineární a kvadratické funkce – v kolika bodech se protínají?), navíc definičním oborem obou funkcí je množina $\mathbb{N}$ přirozených čísel. Z toho vyplývá, že $L \notin \mathscr{L}(CF)$.

Kapitola 3

Zásobníkový automat

V této kapitole se budeme zabývat zásobníkovými automaty, který jsme si stručně popsali již na začátku minulého semestru. Po definici zásobníkového automatu se budeme zabývat jeho některými vlastnostmi a vztahem k bezkontextovým gramatikám.

3.1 Definice zásobníkového automatu

Jak víme, *zásobník* je dynamická struktura podobná frontě nebo seznamu, která se používá stylem LIFO (Last-in First-out): ten prvek, který jsme vložili jako poslední, jako první vyjmeme.

Zásobníkový automat získáme tak, že konečný automat obohatíme o zásobníkovou pásku a zajistíme, aby byl výpočet řízen především obsahem zásobníku. Tento model pracuje takto:

- vyjme symbol na vrcholu zásobníku,
- může nebo nemusí přečíst symbol ze vstupní pásky, pokud přečte, posune se o pole dál,
- dále se rozhoduje podle
  - svého vnitřního stavu,
  - symbolu, který vyndal ze zásobníku,
  - pokud četl ze vstupní pásky, pak i podle přečteného symbolu,
- činnost automatu v jednom kroku spočívá v přechodu do některého dalšího stavu a v uložení řetězce znaků do zásobníku.

Definice 3.1 (Zásobníkový automat)

Zásobníkový automat *je uspořádaná sedmice* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, kde je*

Q ... *neprázdná konečná množina stavů*
Σ ... *neprázdná konečná abeceda automatu*
Γ ... *neprázdná konečná zásobníková abeceda*
δ ... *přechodová funkce, definovaná níže*
q_0 ... *počáteční stav,* $q_0 \in Q$
Z_0 ... *počáteční zásobníkový symbol,* $Z_0 \in \Gamma$
F ... *množina koncových stavů,* $F \subseteq Q$ *(může být i prázdná)*

Přechodová funkce δ *zásobníkového automatu* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je definována takto:*
$\delta : \ Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$ *(zápis pomocí množin)*
$\delta(q, a, Z) \ni (r, \gamma), \quad$ *kde* $q, r \in Q, \ a \in (\Sigma \cup \{\varepsilon\}), \ Z \in \Gamma, \ \gamma \in \Gamma^*$ *(symbolický zápis)*

Zásobníkový automat je obecně nedeterministický. Oproti konečnému automatu máme navíc zásobníkovou abecedu, počáteční zásobníkový symbol a bohatší přechodovou funkci.

Poznámka:

Počáteční zásobníkový symbol můžeme brát jako „zarážku“ na dně zásobníku. Je to obdoba ukazatele null (příp. nil) v dynamických datových strukturách programovacích jazyků.

Definice 3.2 (Konfigurace, počáteční a koncová konfigurace)

Konfigurace *zásobníkového automatu* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je uspořádaná trojice* (q, w, γ), *kde* $q \in Q, \ w \in \Sigma^*$ *(nepřečtená část vstupní pásky) a* $\gamma \in \Gamma^*$ *(momentální obsah zásobníku).*

Počáteční konfigurace *je konfigurace* (q_0, w_0, Z_0), *kde* q_0 *je počáteční stav automatu,* w_0 *je celé zpracovávané slovo a* Z_0 *je počáteční zásobníkový symbol.*

Na začátku výpočtu tedy máme na vstupu celé slovo, začínáme v počátečním stavu a v zásobníku máme pouze počáteční zásobníkový symbol. Definice koncové konfigurace závisí na typu zásobníkového automatu, definujeme ji proto až později.

Definice 3.3 (Přechod mezi konfiguracemi)

Relaci přechodu mezi konfiguracemi zásobníkového automatu $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *značíme symbolem* $\vdash$ *a definujeme ji takto:*

$$(q_i, a\,\omega, Z\beta) \ \vdash \ (q_j, \omega, \gamma\beta) \quad \overset{def}{\iff} \quad \delta(q_i, a, Z) \ni (q_j, \beta), \tag{3.1}$$

kde $q_i, q_j \in Q, \quad a \in (\Sigma \cup \{\varepsilon\}), \quad \omega \in \Sigma^*, \quad Z \in \Gamma, \quad \beta, \gamma \in \Gamma^*$

Symbol $\vdash^*$ značí reflexivní a tranzitivní uzávěr relace $\vdash$, symbol $\vdash^+$ je tranzitivní uzávěr této relace, symbol $\vdash^n$ znamená přesně n přechodů mezi konfiguracemi.

Rozeznáváme tři základní typy zásobníkových automatů:

- *Zásobníkový automat končící přechodem do koncového stavu* – způsob ukončení výpočtu je podobný jako u konečného automatu: je třeba
  - přečíst celý vstup a zároveň
  - přesunout se do některého ze stavů z množiny F (do některého koncového stavu).
- *Zásobníkový automat končící s prázdným zásobníkem*: je třeba
  - přečíst celý vstup a zároveň
  - vyprázdnit celý zásobník (včetně počátečního zásobníkového symbolu).
- *Zásobníkový automat končící přechodem do koncového stavu a s prázdným zásobníkem*: je třeba splnit vše, co je v předchozích odrážkách (přečtený vstup, koncový stav, prázdný zásobník).

Definice 3.4 (Typy ZA, koncová konfigurace a rozpoznávaný jazyk ZA)

Zásobníkový automat končící přechodem do koncového stavu *je* $\mathcal{A}_F = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *s koncovou konfigurací* $(q_f, \varepsilon, \gamma),\ q_f \in F,\ \gamma \in \Gamma^*$ *a rozpoznávaný jazyk je*

$$L(\mathcal{A}_F) = \{w \in \Sigma^*\ ;\ (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \gamma),\ q_f \in F,\ \gamma \in \Gamma^*\}\,. \tag{3.2}$$

Zásobníkový automat končící s prázdným zásobníkem *je* $\mathcal{A}_\emptyset = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ *s koncovou konfigurací* $(q, \varepsilon, \varepsilon),\ q \in Q$ *a rozpoznávaný jazyk je*

$$L(\mathcal{A}_\emptyset) = \{w \in \Sigma^*\ ;\ (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon),\ q \in Q\} \tag{3.3}$$

Zásobníkový automat končící přechodem do koncového stavu a s prázdným zásobníkem *je* $\mathcal{A}_{F,\emptyset} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *s koncovou konfigurací* $(q_f, \varepsilon, \varepsilon),\ q_f \in F$ *a rozpoznávaný jazyk je*

$$L(\mathcal{A}_{F,\emptyset}) = \{w \in \Sigma^*\ ;\ (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \varepsilon),\ q_f \in F\} \tag{3.4}$$

Když vytváříme zásobníkový automat, musíme předem vědět, kterého typu bude, podle toho konstruujeme přechodovou funkci. Nicméně – pokud vytvoříme jeden z těchto typů, není problém sestrojit ekvivalentní zásobníkový automat jiného typu.

Poznámka:

Při sestavování zásobníkového automatu postupujeme poněkud systematičtěji než u konečného automatu. Všímáme si *struktury slov* jazyka. Rozdělíme (obecné) slovo na části a stanovíme, jak se v jednotlivých částech má automat chovat. Průběh zpracování v těchto částech odlišíme stavem a určíme, co v kterém stavu má být v zásobníku a jak se má se zásobníkem zacházet.

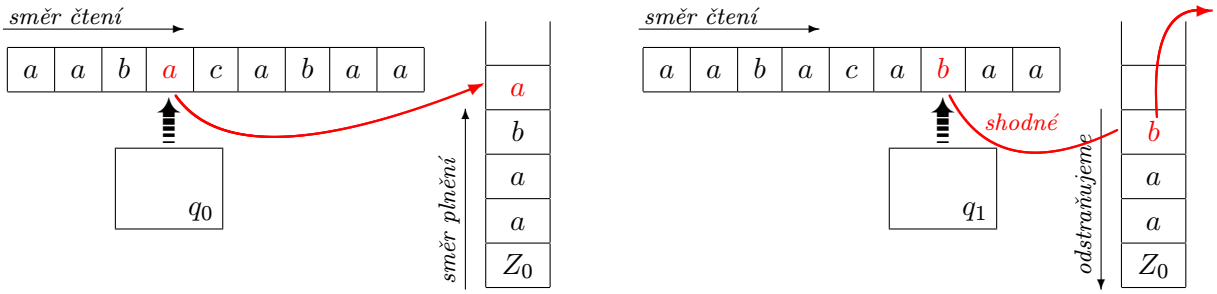
Příklad 3.1

Sestrojíme zásobníkový automat rozpoznávající jazyk $L = \{wcw^R\ ;\ w \in \{a, b\}^*\}$

Vytvoříme zásobníkový automat rozpoznávající prázdným zásobníkem:
$\mathcal{A}_\emptyset = (Q,\ \{a, b\},\ \Gamma,\ q_0,\ Z_0,\ \delta,\ \emptyset)$
Slova tohoto jazyka se skládají ze dvou částí oddělených symbolem c. Pro každou část určíme stav, tedy potřebujeme dva stavy: $Q = \{q_0, q_1\}$. Jak se náš automat má v těchto stavech chovat?

- Ve stavu q_0 načítáme první část slova, pouze ukládáme do zásobníku:
  - načteme symbol ze vstupu,
  - sice vyzvedneme symbol z vrcholu zásobníku, ale vrátíme ho zpátky beze změny,
  - načtený symbol uložíme také do zásobníku.
- Ve stavu q_1 načítáme druhou část slova a kontrolujeme s obsahem zásobníku:
  - načteme symbol ze vstupu,
  - vyzvedneme symbol z vrcholu zásobníku,
  - pokud jsou tyto dva symboly shodné, pak je vše v pořádku.

Přechod mezi stavy q_0 a q_1 nastává při načtení „oddělujícího“ symbolu c.



Vytvoříme přechodovou funkci. Nejdřív budeme předpokládat, že je na vstupu slovo obsahující i jiné symboly než jen c (tj. délka ≥ 2), pak přidáme i možnost zpracování slova o délce 1.

- Začneme ve stavu q_0, na vrcholu zásobníku je symbol Z_0 (v zásobníku zatím nic jiného nemáme); zásobníkový symbol sice vyjmeme ze zásobníku (musíme), ale opět ho tam vrátíme a přidáme symbol, který jsme načetli ze vstupu. Na vstupu může být buď a nebo b.
  $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  $\delta(q_0, b, Z_0) = (q_0, bZ_0)$
- Stejně budeme reagovat i v dalších krocích (jsme pořád v první části slova). Ať minimalizujeme počet řádků, použijeme zástupný symbol X znamenající a nebo b:
  $\delta(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
  $\delta(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
- Jsme na hranici mezi dvěma částmi slova, načteme c, změníme stav (zásobník necháme jak je, vrátíme vyjmutý symbol a nebudeme přidávat c):
  $\delta(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
- V druhé části slova provádíme synchronizaci obou částí – první část máme v zásobníku (v přesně opačném pořadí než jak tento řetězec byl na vstupu), druhou na vstupu, budeme kontrolovat, jestli je na obou místech totéž, do zásobníku nebudeme nic ukládat:
  $\delta(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
- Zpracovali jsme celou druhou část slova ze vstupu, v zásobníku by měl zůstat už jen symbol Z_0, tedy ukončíme výpočet:
  $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$
- Ještě ošetříme případ, kdy bude na vstupu nejkratší slovo jazyka, c:
  $\delta(q_0, c, Z_0) = (q_1, \varepsilon)$

Celá definice tohoto zásobníkového automatu je následující:

$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \Gamma,\ q_0,\ Z_0,\ \delta,\ \emptyset)$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
$\delta(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
$\delta(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
$\delta(q_0, c, Z_0) = (q_1, \varepsilon)$
$\delta(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

Zápis by se dal ještě více zkrátit, například u řádků z prvního bodu postupu. Zásobníková abeceda je $\Gamma = \{Z_0, a, b\}$, řadíme tam všechny symboly, které se mohou dostat do zásobníku.

Podíváme se na zpracování několika slov patřících do jazyka L. Nejdřív slovo $abcba$:

- V prvním kroku použijeme předpis $\delta(q_0, a, Z_0) = (q_0, aZ_0)$:
  $(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0)$

- V druhém kroku použijeme předpis $\delta(q_0, b, a) = (q_0, ba)$:
  $(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0) \vdash (q_0, cba, baZ_0)$
- V pátém kroku to bude předpis $\delta(q_0, c, b) = (q_1, b)$:
  $\ldots \vdash (q_0, cba, baZ_0) \vdash (q_1, ba, baZ_0)$
- V dalším kroku začneme synchronizovat – srovnávat, předpis $\delta(q_1, b, b) = (q_1, \varepsilon)$:
  $\ldots \vdash (q_1, ba, baZ_0) \vdash (q_1, a, a, Z_0)$
- Pokračujeme předpisem $\delta(q_1, a, a)$:
  $\ldots \vdash (q_1, a, a, Z_0) \vdash (q_1, \varepsilon, Z_0)$
- V posledním kroku uklidíme zásobník předpisem $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$:
  $\ldots \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Celý výpočet:
$(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0) \vdash (q_0, cba, baZ_0) \vdash (q_1, ba, baZ_0) \vdash (q_1, a, a, Z_0) \vdash (q_1, \varepsilon, Z_0) \vdash$
$\vdash (q_1, \varepsilon, \varepsilon)$

Teď trochu delší slovo $aabacabaa$:
$(q_0, aabacabaa, Z_0) \vdash (q_0, abacabaa, aZ_0) \vdash (q_0, bacabaa, aaZ_0) \vdash (q_0, acabaa, baaZ_0) \vdash$
$\vdash (q_0, cabaa, abaaZ_0) \vdash (q_1, abaa, abaaZ_0) \vdash (q_1, baa, baa, Z_0) \vdash (q_1, aa, aaZ_0) \vdash$
$\vdash (q_1, a, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Nejkratší slovo jazyka c – stačí nám jeden krok:
$(q_0, c, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

Dáme na vstup několik slov nepatřících do jazyka $L(\mathcal{A})$:
$(q_0, ab, Z_0) \vdash (q_0, b, aZ_0) \vdash (q_0, \varepsilon, baZ_0) \vdash$ nelze pokračovat, $ab \notin L(\mathcal{A})$

$(q_0, ca, Z_0) \vdash (q_1, a, Z_0) \vdash (q_1, a, \varepsilon) \vdash$ nelze pokračovat, $ca \notin L(\mathcal{A})$
Všimněte si posledního kroku – můžeme použít $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$, třebaže vstup není prázdný.

$(q_0, \varepsilon, Z_0) \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A})$

Příklad 3.2

Sestrojíme zásobníkový automat končící v koncovém stavu pro jazyk $L = \{a^n b^n \ ; \ n \geq 0\}$.

Promyslíme si, jak má vypadat přechodová funkce a které stavy budeme potřebovat. Pro první polovinu slova budeme mít stav q_0, pro druhou stav q_1. První polovinu slova budeme jen načítat do zásobníku, kdežto při načítání druhé poloviny budeme naopak zásobník vyprazdňovat a zároveň srovnávat se vstupem (na jeden symbol b na vstupu musí být jeden symbol a v zásobníku).

- Pro první polovinu slova:
  $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  $\delta(q_0, a, a) = (q_0, aa)$ dohromady: $\delta(q_0, a, X) = (q_0, aX)$, kde $X \in \{Z_0, a\}$
- Přechod do druhé poloviny slova:
  $\delta(q_0, b, a) = (q_1, \varepsilon)$
- Druhá polovina slova:
  $\delta(q_1, b, a) = (q_1, \varepsilon)$

- Ukončení:
  $\delta(q_1, \varepsilon, Z_0) = (q_f, \varepsilon)$
- Automat má rozpoznávat i prázdné slovo:
  $\delta(q_0, \varepsilon, Z_0) = (q_f, \varepsilon)$

Výsledný automat:
$\mathcal{A}_F = (\{q_0, q_1, q_f\},\ \{a, b\},\ \{Z_0, a\},\ \delta,\ q_0,\ Z_0,\ \{q_f\})$ s přechodovou funkcí:

$\delta(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_0, \varepsilon, Z_0) = (q_f, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_f, \varepsilon)$

Výpočet několika slov patřících nebo nepatřících do jazyka L:
$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0) \vdash (q_0, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_f, \varepsilon, \varepsilon)$
$(q_0, \varepsilon, Z_0) \vdash (q_f, \varepsilon, \varepsilon)$
$(q_0, aab, Z_0) \vdash (q_0, ab, aZ_0) \vdash (q_0, b, aaZ_0) \vdash (q_1, \varepsilon, aZ_0) \vdash$ nelze pokračovat, $aab \notin L(\mathcal{A})$
$(q_0, abb, Z_0) \vdash (q_0, bb, aZ_0) \vdash (q_1, b, Z_0) \vdash (q_f, b, \varepsilon) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A})$
$(q_0, b, Z_0) \vdash (q_f, b, \varepsilon) \vdash$ nelze pokračovat, $b \notin L(\mathcal{A})$
$(q_0, a, Z_0) \vdash (q_0, \varepsilon, aZ_0) \vdash$ nelze pokračovat, $a \notin L(\mathcal{A})$

V tomto případě jsme sestrojili zásobníkový automat, který nejen rozpoznává koncovým stavem (tj. pokud jsme v koncovém stavu, zde q_f, a zároveň je vstup přečtený, je výpočet úspěšný), ale zároveň je v každé koncové konfiguraci prázdný zásobník. Tedy jsme sestrojili zásobníkový automat končící zároveň v koncovém stavu a s prázdným zásobníkem.

3.2 Vztah mezi typy zásobníkových automatů

Nejčastěji vytváříme zásobníkové automaty rozpoznávající prázdným zásobníkem, ale obecně je jedno, který typ použijeme. Každý z výše uvedených typů zásobníkových automatů totiž dokážeme převést na kterýkoliv jiný typ.

Věta 3.1 (Vztah mezi typy zásobníkových automatů)
Pro zásobníkové automaty končící s prázdným zásobníkem, v koncovém stavu a kombinované (s prázdným zásobníkem a v koncovém stavu) platí následující:

$$\mathscr{L}(\mathcal{A}_\emptyset) \cong \mathscr{L}(\mathcal{A}_F) \cong \mathscr{L}(\mathcal{A}_{F,\emptyset}) \tag{3.5}$$

Postup konstrukce ($\mathscr{L}(\mathcal{A}_\emptyset) \subseteq \mathscr{L}(\mathcal{A}_F)$)**:** Původní zásobníkový automat končící prázdným zásobníkem označíme $\mathcal{A}_\emptyset = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$, nový automat končící v koncovém stavu označíme $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F)$.

K zásobníkovému automatu končícímu s prázdným zásobníkem sestrojíme ekvivalentní zásobníkový automat končící v koncovém stavu takto:

- vytvoříme nový stav q_f, který bude novým koncovým stavem,
- v konfiguraci, ve které bychom v původním automatu končili, provedeme ještě jeden přechod – právě do stavu q_f,
- aby tento přechod vůbec byl možný, potřebujeme mít vlastní zásobníkový symbol ještě pod zásobníkovým symbolem použitým v původním automatu, a na to je třeba brát ohled i na začátku výpočtu.

Uvnitř nového automatu budeme simulovat ten původní. Nový automat bude mít vlastní symbol konce zásobníku, a v prvním kroku výpočtu naváže na výpočet původního automatu tím, že

- přejde do stavu, ve kterém začíná původní automat,
- do zásobníku přidá symbol konce zásobníku původního automatu (svůj tam nechá).

Srovnáme průběh výpočtu v původním a novém zásobníkovém automatu:

Původní automat: $q_0\,[Z_0] \vdash \ldots \vdash q_k\,[Z_0] \vdash q_m\,[\,]$

Nový automat: $q_0'\,[Z_0'] \vdash q_0\,[Z_0 / Z_0'] \vdash \ldots \vdash q_k\,[Z_0 / Z_0'] \vdash q_m\,[Z_0'] \vdash q_f\,[\,]$

Tedy potřebujeme, aby první přechod mezi konfiguracemi vypadal takto: $(q_0', w, Z_0') \vdash (q_0, w, Z_0 Z_0')$ (vstup se v prvním kroku nezmění), čímž se napojíme na výpočet původního automatu. Na závěr výpočtu musíme provést následující: $(q_m, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$, čímž se dostaneme do koncové konfigurace nového automatu. Stavy q_0' a q_f jsou nově přidané, tedy musí platit $q_0' \notin Q,\ q_f \notin Q$.

Přechodovou funkci původního automatu přejmeme a přidáme tyto přechody:
$\delta'(q_0', \varepsilon, Z_0') = (q_0, Z_0 Z_0')$
$\delta'(q, \varepsilon, Z_0') = (q_f, \varepsilon)$ pro všechny stavy $q \in Q$

Výsledný automat vypadá takto:
$\mathcal{A}_F = (Q \cup \{q_0', q_f\},\ \Sigma,\ \Gamma \cup \{Z_0'\},\ \delta',\ q_0',\ Z_0',\ \{q_f\})$ □

Příklad 3.3

K zásobníkovému automatu z prvního příkladu této kapitoly (začíná na straně 81) sestrojíme ekvivalentní zásobníkový automat končící v koncovém stavu. Původní automat je
$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \{Z_0, a, b\},\ q_0,\ Z_0,\ \delta,\ \emptyset)$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
$\delta(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
$\delta(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
$\delta(q_0, c, Z_0) = (q_1, \varepsilon)$
$\delta(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

Přidáme nový stav q_0', ve kterém bude začínat výpočet, stav q_f, ve kterém bude končit výpočet, a nový zásobníkový symbol Z_0'. Automat bude následující:
$\mathcal{A}_F = (\{q_0, q_1, q_0', q_f\},\ \{a, b\},\ \{Z_0, a, b, Z_0'\},\ q_0',\ Z_0',\ \delta',\ \{q_f\})$, přechodová funkce δ' je

$\delta'(q_0', \varepsilon, Z_0') = (q_0, Z_0 Z_0')$

$\delta'(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta'(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta'(q_0, a, X) = (q_0, aX), \quad X \in \{a, b\}$
$\delta'(q_0, b, X) = (q_0, bX), \quad X \in \{a, b\}$
$\delta'(q_0, \varepsilon, Z_0') = (q_f, \varepsilon)$

$\delta'(q_0, c, X) = (q_1, X), \quad X \in \{a, b\}$
$\delta'(q_0, c, Z_0) = (q_1, \varepsilon)$
$\delta'(q_1, X, X) = (q_1, \varepsilon), \quad X \in \{a, b\}$
$\delta'(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$
$\delta'(q_1, \varepsilon, Z_0') = (q_f, \varepsilon)$

Ukážeme si zpracování několika slov patřících nebo nepatřících do jazyka původního automatu:

$(q_0', abcba, Z_0') \vdash (q_0, abcba, Z_0 Z_0') \vdash (q_0, bcba, aZ_0 Z_0') \vdash (q_0, cba, baZ_0 Z_0') \vdash (q_1, ba, baZ_0 Z_0') \vdash$
$\vdash (q_1, a, a, Z_0 Z_0') \vdash (q_1, \varepsilon, Z_0 Z_0') \vdash (q_1, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$

$(q_0', c, Z_0') \vdash (q_0, c, Z_0 Z_0') \vdash (q_1, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$

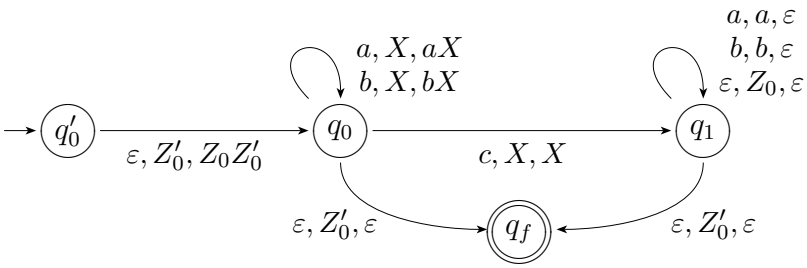
$(q_0', ab, Z_0') \vdash (q_0, ab, Z_0 Z_0') \vdash (q_0, b, aZ_0 Z_0') \vdash (q_0, \varepsilon, baZ_0 Z_0') \vdash$ nelze pokračovat, $ab \notin L(\mathcal{A}_F)$

$(q_0', ca, Z_0') \vdash (q_0, ca, Z_0 Z_0') \vdash (q_1, a, Z_0 Z_0') \vdash (q_1, a, Z_0') \vdash (q_f, a, \varepsilon) \vdash$ nelze pokračovat, $ca \notin L(\mathcal{A}_F)$

$(q_0', \varepsilon, Z_0') \vdash (q_0, \varepsilon, Z_0 Z_0') \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A}_F)$

Příklad 3.4

Reprezentaci zásobníkového automatu diagramem obvykle nepoužíváme – důvodem je horší přehlednost diagramu (v zásobníkových automatech musíme někam zapsat i práci se zásobníkem, také máme přechody bez zpracování slova na vstupu). Nicméně vytvoření diagramu je možné, například pro automat z předchozího příkladu by vypadal takto:



$X \in \{a, b, Z_0\}$

Ke každému přechodu píšeme čtený symbol ze vstupu, čtený symbol ze zásobníku a řetězec k uložení na zásobník.

Poznámka:

Všimněte si, že zásobníkový automat, který jsme v druhém příkladu vytvořili, ve skutečnosti končí jak v koncovém stavu, tak i s prázdným zásobníkem, tedy jsme zároveň dokázali vztah $\mathscr{L}(\mathcal{A}_\emptyset) \subseteq \mathscr{L}(\mathcal{A}_{F,\emptyset})$.

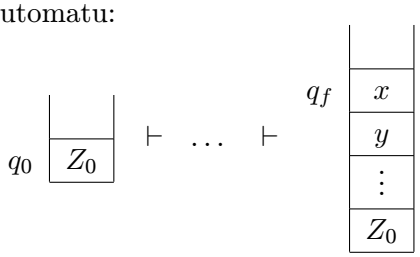
Postup konstrukce ($\mathscr{L}(\mathcal{A}_F) \subseteq \mathscr{L}(\mathcal{A}_\emptyset)$)**:** Původní zásobníkový automat končící v koncovém stavu označíme $\mathcal{A}_\emptyset = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, nový automat končící s prázdným zásobníkem označíme $\mathcal{A}' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', \emptyset)$.

K zásobníkovému automatu končícímu v koncovém stavu sestrojíme ekvivalentní zásobníkový automat končící s prázdným zásobníkem takto:

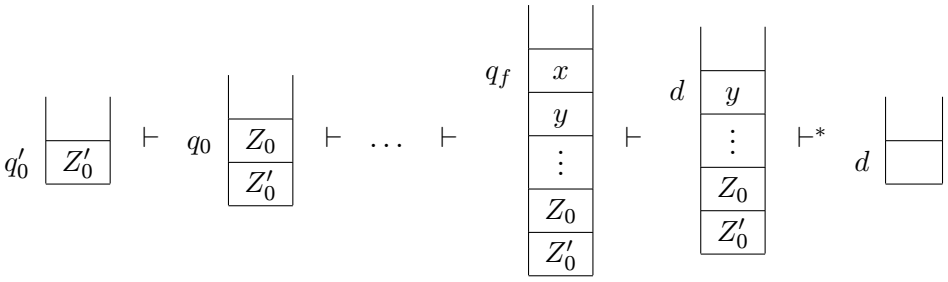
- budeme mít vlastní symbol konce zásobníku,
- na konci výpočtu (až budeme v koncovém stavu původního automatu) přejdeme do nově přidaného stavu d („delete“), ve kterém následovně rekurzívně mažeme obsah zásobníku.

Uvnitř nového automatu budeme opět simulovat ten původní. Srovnáme průběh výpočtu v původním a novém zásobníkovém automatu:

Původní automat:



Nový automat:



První přechod mezi konfiguracemi má vypadat takto: $(q'_0, w, Z'_0) \vdash (q_0, w, Z_0 Z'_0)$, čímž se napojíme na výpočet původního automatu.

Na závěr výpočtu musíme provést následující: $(q_f, \varepsilon, Z\gamma Z'_0) \vdash (d, \varepsilon, \gamma Z'_0) \vdash^* (d, \varepsilon, \varepsilon)$, čímž se dostaneme do koncové konfigurace nového automatu. Stavy q'_0 a d jsou nově přidané, tedy musí platit $q'_0 \notin Q, \ d \notin Q$.

Přechodovou funkci původního automatu přejmeme a přidáme tyto přechody:

- $\delta'(q'_0, \varepsilon, Z'_0) = (q_0, Z_0 Z'_0)$ (napojíme se na původní výpočet v $\mathcal{A}$),
- $\delta'(q_f, \varepsilon, Z) = (d, \varepsilon)$ pro všechny původní koncové stavy $q_f \in F$ a zásobníkové symboly $Z \in \Gamma \cup \{Z'_0\}$ (přesun do „mazacího“ stavu d),
- $\delta'(d, \varepsilon, Z) = (d, \varepsilon)$ pro všechny $Z \in \Gamma \cup \{Z'_0\}$ (mazání).

Výsledný automat: $\mathcal{A}_\emptyset = (Q \cup \{q'_0, d\}, \ \Sigma, \ \Gamma \cup \{Z'_0\}, \ \delta', \ q'_0, \ Z'_0, \ \emptyset)$. □

Poznámka:

Pokud bychom předchozí popsaný postup obohatili o jeden koncový stav: $F' = \{d\}$, získáme zásobníkový automat končící s prázdným zásobníkem a zároveň v koncovém stavu. Tím jsme si ukázali, že platí $\mathscr{L}(\mathcal{A}_F) \subseteq \mathscr{L}(\mathcal{A}_{\emptyset,F})$, čímž jsme dokončili postup konstrukce pro všechny vztahy zahrnuté v uvedené větě.

Příklad 3.5

Sestrojíme zásobníkový automat končící v koncovém stavu pro jazyk $L = \{a^n b^m \ ; \ 1 \leq m \leq n\}$. Symbolů b má být ve slově buď stejně nebo méně než symbolů a. Automat má pracovat takto:

- Ve stavu q_0 načítáme symboly a ze vstupu a ukládáme do zásobníku:
  $\delta(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
- Přecházíme do druhé části slova:
  $\delta(q_0, b, a) = (q_1, \varepsilon)$
- V druhé části slova postupně odstraňujeme symboly a ze zásobníku:
  $\delta(q_1, b, a) = (q_1, \varepsilon)$
- Pokud máme celý vstup přečtený, ukončíme výpočet, třebaže v zásobníku ještě mohou být symboly a:
  $\delta(q_1, \varepsilon, X) = (q_f, X),$ kde $X \in \{Z_0, a\}$

Výsledný automat:
$\mathcal{A} = (\{q_0, q_1, q_f\},\ \{a, b\},\ \{Z_0, a\},\ \delta,\ q_0,\ Z_0,\ \{q_f\})$, přechodová funkce:
$\delta(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, X) = (q_f, X),$ kde $X \in \{Z_0, a\}$

Ukázka zpracování několika slov patřících či nepatřících do jazyka $L(\mathcal{A})$:

$(q_0, aab, Z_0) \vdash (q_0, ab, aZ_0) \vdash (q_0, b, aaZ_0) \vdash (q_1, \varepsilon, aZ_0) \vdash (q_f, \varepsilon, aZ_0)$

$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0) \vdash (q_0, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_f, \varepsilon, Z_0)$

$(q_0, abb, Z_0) \vdash (q_0, bb, aZ_0) \vdash (q_1, b, Z_0) \vdash (q_f, b, Z_0) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A})$

$(q_0, \varepsilon, Z_0) \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A})$

Sestrojíme ekvivalentní zásobníkový automat končící s prázdným zásobníkem:
$\mathcal{A}_\emptyset = (\{q_0', q_0, q_1, q_f, d\},\ \{a, b\},\ \{Z_0', Z_0, a\},\ q_0',\ Z_0',\ \emptyset)$ s přechodovou funkcí:
$\delta'(q_0', \varepsilon, Z_0') = (q_0, Z_0 Z_0')$
$\delta'(q_0, a, X) = (q_0, aX),$ kde $X \in \{Z_0, a\}$
$\delta'(q_0, b, a) = (q_1, \varepsilon)$
$\delta'(q_1, b, a) = (q_1, \varepsilon)$
$\delta'(q_1, \varepsilon, X) = (q_f, X),$ kde $X \in \{Z_0, a\}$
$\delta'(q_f, \varepsilon, X) = (d, \varepsilon),$ kde $X \in \{Z_0', Z_0, a\}$
$\delta'(d, \varepsilon, X) = (d, \varepsilon),$ kde $X \in \{Z_0', Z_0, a\}$

Ukázka zpracování stejných slov jako u automatu $\mathcal{A}$:

$(q_0', aab, Z_0') \vdash (q_0, aab, Z_0 Z_0') \vdash (q_0, ab, aZ_0 Z_0') \vdash (q_0, b, aaZ_0 Z_0') \vdash (q_1, \varepsilon, aZ_0 Z_0') \vdash (q_f, \varepsilon, aZ_0 Z_0') \vdash$
$\vdash (d, \varepsilon, Z_0 Z_0') \vdash (d, \varepsilon, Z_0') \vdash (d, \varepsilon, \varepsilon)$

$(q_0', aabb, Z_0') \vdash (q_0, aabb, Z_0 Z_0') \vdash (q_0, abb, aZ_0 Z_0') \vdash (q_0, bb, aaZ_0 Z_0') \vdash (q_1, b, aZ_0 Z_0') \vdash$
$\vdash (q_1, \varepsilon, Z_0 Z_0') \vdash (q_f, \varepsilon, Z_0 Z_0') \vdash (d, \varepsilon, Z_0') \vdash (d, \varepsilon, \varepsilon)$

$(q_0', abb, Z_0') \vdash (q_0, abb, Z_0 Z_0') \vdash (q_0, bb, aZ_0 Z_0') \vdash (q_1, b, Z_0 Z_0') \vdash (q_f, b, Z_0 Z_0') \vdash (d, b, Z_0') \vdash$
$\vdash (d, b, \varepsilon) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A}_\emptyset)$

$(q_0', \varepsilon, Z_0') \vdash (q_0, \varepsilon, Z_0 Z_0') \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A}_\emptyset)$

Poznámka:

Kdybychom v předchozím případě chtěli vytvořit zásobníkový automat rozpoznávající jak s prázdným zásobníkem, tak i v koncovém stavu, jen bychom mírně pozměnili zápis:
$\mathcal{A}_\emptyset = (\{q_0', q_0, q_1, q_f, d\},\ \{a, b\},\ \{Z_0', Z_0, a\},\ q_0',\ Z_0',\ \{d\})$. Přechodová funkce by byla stejná.

3.3 Vztah zásobníkových automatů a bezkontextových gramatik

3.3.1 Vytvoření automatu podle bezkontextové gramatiky

Nadále budeme počítat s tím, že zásobníkové automaty všech tří základních typů jsou navzájem ekvivalentní, a tedy generují stejné třídy jazyků. Proto v důkazech budeme volně tyto typy zaměňovat.

Věta 3.2 (Bezkontextová gramatika $\longrightarrow$ zásobníkový automat)

Ke každé bezkontextové gramatice G lze vytvořit zásobníkový automat $\mathcal{A}$ tak, že $L(G) = L(A)$:

$$\mathscr{L}(CF) \subseteq \mathscr{L}(ZA) \tag{3.6}$$

Postup konstrukce: Je dána bezkontextová gramatika $G = (N, T, P, S)$. Chceme sestrojit k ní ekvivalentní zásobníkový automat $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ končící s prázdným zásobníkem.

Princip: potřebujeme rozpoznávat právě ta slova, která jsou generována gramatikou. Proto vytvářený automat bude na svém zásobníku provádět simulaci derivace pro slovo, které dostane na svůj vstup. Pokud zjistí, že slovo lze v původní gramatice derivovat, pak je přijme.

Postup bude odlišný od postupu pro regulární gramatiky a konečné automaty. Máme k dispozici zásobník a ten budeme využívat. Naopak stavy pro nás nebudou důležité, vystačíme si s jediným stavem, který můžeme označit q. Abeceda je $\Sigma = T$. Přechodová funkce bude mít dvě části:

- První část odpovídá pravidlům původní gramatiky:

Pravidlo gramatiky	*Předpis v δ-funkci*
$A \to \alpha$	$\delta(q, \varepsilon, A) \ni (q, \alpha)$

  Jak vidíme, vstupu si nevšímáme (ε), ze zásobníku vyjmeme neterminál (zde A) a nahradíme řetězcem z pravé strany pravidla pro tento neterminál (α). Vše se odehrává pouze na zásobníku, nemění se stav a nepohybujeme se na vstupu.

- Druhá část gramatiky určuje, že pokud je na vrcholu zásobníku terminál, pak zjistíme, jestli je tentýž terminál na vstupu – když ano, posuneme se jak na vstupu, tak i na zásobníku:

  $\delta(q, a, a) = (q, \varepsilon)$ pro všechny symboly $a \in T$

  Tato část je také důležitá, protože v zásobníku máme samozřejmě kromě neterminálů i terminály (dostávají se tam se zápisem pravých stran pravidel, α). Zároveň kontrolujeme, jestli simulace derivace podle gramatiky probíhá správně (tedy zda simulujeme odvozování toho slova, které je na vstupu, a ne jiného).

Počátečním stavem bude stav q, počátečním zásobníkovým symbolem bude S (startovací symbol gramatiky), protože pokud máme na zásobníku provádět simulaci derivace, musíme tam na začátku mít počáteční větnou formu, což je právě jednoprvkový řetězec obsahující startovací symbol.

Protože cokoliv, co se nachází v pravidlech gramatiky, se může objevit v zásobníku, zásobníková abeceda obsahuje všechny neterminály i terminály původní gramatiky: $\Gamma = N \cup T$. □

Příklad 3.6

Sestrojíme zásobníkový automat ekvivalentní gramatice $G = (\{S, A\},\ \{a, b, c\},\ P, S)$, kde P obsahuje tato pravidla:
$S \to aSbb \mid cAa$
$A \to cAa \mid \varepsilon$

Vytvoříme zásobníkový automat $\mathcal{A} = (\{q\},\ \{a, b, c\},\ \{S, A, a, b, c\},\ \delta,\ q,\ S,\ \emptyset)$ s přechodovou funkcí určenou takto:

$\delta(q, \varepsilon, S) = \{(q, aSbb), (q, cAa)\}$ podle pravidel $S \to aSbb \mid cAa$
$\delta(q, \varepsilon, A) = \{(q, cAa), (q, \varepsilon)\}$ podle pravidel $A \to cAa \mid \varepsilon$
$\delta(q, a, a) = \{(q, \varepsilon)\}$ protože $a \in T$
$\delta(q, b, b) = \{(q, \varepsilon)\}$ protože $b \in T$
$\delta(q, c, c) = \{(q, \varepsilon)\}$ protože $c \in T$

Jazyk generovaný gramatikou G a rozpoznávaný automatem $\mathcal{A}$ je
$L(G) = L(\mathcal{A}) = \{a^n c^k a^k b^{2n}\ ;\ n \geq 0,\ k \geq 1\}$.

Ukážeme si vždy derivaci některého slova v gramatice G a ekvivalentní zpracování slova v zásobníkovém automatu $\mathcal{A}$:

$S \Rightarrow cAa \Rightarrow ccAaa \Rightarrow ccaa$

$(q, ccaa, S) \vdash (q, ccaa, cAa) \vdash (q, caa, Aa) \vdash (q, caa, cAaa) \vdash (q, aa, Aaa) \vdash (q, aa, aa) \vdash (q, a, a) \vdash$
$\vdash (q, \varepsilon, \varepsilon)$

$\Longrightarrow$ $ccaa \in L(G)$ a zároveň $caa \in L(\mathcal{A})$

$S \Rightarrow aSbb \Rightarrow aaSbbbb \Rightarrow aacAabbbb \Rightarrow aacabbbb$

$(q, aacabbbb, S) \vdash (q, aacabbbb, aSbb) \vdash (q, acabbbb, Sbb) \vdash (q, acabbbb, aSbbbb) \vdash (q, cabbbb, Sbbbb) \vdash$
$\vdash (q, cabbbb, cAabbbb) \vdash (q, abbbb, Aabbbb) \vdash (q, abbbb, abbbb) \vdash (q, bbbb, bbbb) \vdash (q, bbb, bbb) \vdash$
$\vdash (q, bb, bb) \vdash (q, b, b) \vdash (q, \varepsilon, \varepsilon)$

$\Longrightarrow$ $aacabbbb \in L(G)$ a zároveň $aacabbbb \in L(\mathcal{A})$

Na vstup automatu dáme některá slova nepatřící do jazyka $L(G)$:

$(q, abb, S) \vdash (q, abb, aSbb) \vdash (q, bb, Sbb) \vdash (q, bb, cAabb) \vdash$ nelze pokračovat, $abb \notin L(\mathcal{A})$

$(q, \varepsilon, S) \vdash (q, \varepsilon, cAa) \vdash$ nelze pokračovat, $\varepsilon \notin L(\mathcal{A})$

$(q, acab, S) \vdash (q, acab, aSbb) \vdash (q, cab, Sbb) \vdash (q, cab, cAabb) \vdash (q, ab, Aabb) \vdash (q, ab, abb) \vdash$
$\vdash (q, b, bb) \vdash (q, \varepsilon, b) \vdash$ nelze pokračovat, $acab \notin L(\mathcal{A})$

Příklad 3.7

Sestrojíme zásobníkový automat jazykově ekvivalentní k této gramatice:
$G = (\{S\},\ \{a,b,c\},\ P,\ S)$, kde $P = \{S \to aSa \mid bSb \mid c\}$

Vytvoříme zásobníkový automat $\mathcal{A} = (\{q\},\ \{a,b,c\},\ \{S,a,b,c\},\ q,\ S,\ \emptyset)$.
$\delta(q,\varepsilon,S) = \{(q,aSa),\ (q,bSb),\ (q,c)\}$
$\delta(q,a,a) = \{(q,\varepsilon)\}$
$\delta(q,b,b) = \{(q,\varepsilon)\}$

Jazyk generovaný gramatikou G a rozpoznávaný automatem $\mathcal{A}$ je $L = \{wcw^R\ ;\ w \in \{a,b\}^*\}$.

Ukázka rozpoznání slova *abcba*:

$(q,abcba,S) \vdash (q,abcba,aSa) \vdash (q,bcba,Sa) \vdash (q,bcba,bSba) \vdash (q,cba,Sba) \vdash (q,cba,cba) \vdash$
$\vdash (q,ba,ba) \vdash (q,a,a) \vdash (q,\varepsilon,\varepsilon)$

Ukázka neúspěšného výpočtu (slovo *acb* bude odmítnuto):

$(q,acb,S) \vdash (q,acb,aSa) \vdash (q,cb,Sa) \vdash (q,cb,ca) \vdash (q,b,a) \vdash$ nelze pokračovat, $acb \notin L(\mathcal{A})$

3.3.2 Vytvoření gramatiky podle zásobníkového automatu

Následující lemma využijeme v důkazu další věty o vztahu mezi bezkontextovými gramatikami a zásobníkovými automaty.

Lemma 3.3

Ke každému zásobníkovému automatu $\mathcal{A}$ existuje jednostavový zásobníkový automat $\mathcal{A}'$ takový, že $L(\mathcal{A}') = L(\mathcal{A})$.

Postup

Původní a vytvářený automat jsou:
$\mathcal{A} = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,\emptyset)$ $\qquad$ $\mathcal{A}' = (\{s\},\Sigma,\Gamma',\delta',s,Z_0',\emptyset)$

Zásobníkový automat se v každém kroku obvykle rozhoduje podle tří kritérií – stavu vstupní pásky, stavu zásobníku a svého vnitřního stavu, ale když má k dispozici jen jediný vnitřní stav, musí se umístění této informace nahradit něčím jiným. Vstupní pásku nesmíme pozměnit, zbývá jen zásobník.

Tedy informaci původně uloženou ve vnitřním stavu přesuneme do zásobníku tak, že místo „jednoduchých“ zásobníkových symbolů budeme používat uspořádané trojice, jejichž druhý prvek je některý symbol původní zásobníkové abecedy, první a třetí prvek jsou stavy:

$$\Gamma' = \{[q_i,Z,q_j]\ ;\ q_i,q_j \in Q,\ Z \in \Gamma\}$$

- q_i je stav, ve kterém je Z na vrcholu zásobníku původního automatu (a tedy se v tom stavu vybírá ze zásobníku),
- q_j je stav, do kterého přecházíme při vyjmutí všeho, co může být vygenerováno pomocí Z, ze zásobníku v původním automatu.

V zásobníku tedy kromě původních zásobníkových symbolů ukládáme také informaci o tom, v jakém stavu jsou tyto symboly zpracovávány a do jakého stavu přecházíme po jejich plném zpracování v původním automatu.

Nyní definujeme přechodovou funkci:

1. Na začátku výpočtu připravíme simulaci původního automatu:
   $\delta'(s,\varepsilon,Z_0') = \left\{\left(s,[q_0,Z_0,p]\right)\ ;\ p\in Q\right\}$
2. Pro všechny předpisy $\mathcal{A}$ ve tvaru $\delta(p,a,Z)\ni(q,\varepsilon)$ (do zásobníku nic nevkládají) vytvoříme
   $\delta'\left(s,a,[p,Z,q]\right)\ni(s,\varepsilon),\quad a\in\Sigma\cup\{\varepsilon\}$
3. Pro všechny ostatní funkce ve tvaru $\delta(p,a,Z)\ni(q,B_1B_2\dots B_n)$:
   $\delta'\left(s,a,[p,Z,u_n]\right)\ \supseteq\ \left\{\left(s,[q,B_1,u_1][u_1,B_2,u_2][u_2,B_3,u_3]\dots[u_{n-1}B_nu_n]\right)\ ;\right.$
   $\left.\forall\text{ kombinace stavů } u_i\in Q, 1\le i\le n,\ a\in\Sigma\cup\{\varepsilon\}\right\}$

Nejnáročnější je poslední bod. Nové zásobníkové symboly zde určují všechny možné posloupnosti, jakými lze v původním automatu dojít ze stavu q, ve kterém vybíráme ze zásobníku symbol Z, do některého stavu u_n, do kterého přecházíme po zpracování posledního zde vkládaného symbolu, B_n. Musí zde být všechny možné kombinace n-tic původních stavů, protože nemůžeme předvídat, jaká posloupnost zpracovávaných stavů bude v těchto n krocích použita.

V původním automatu je symbol Z vyjmut ve stavu p a hned přecházíme do stavu q; zároveň vkládáme do zásobníku symboly $B_1,B_2,\dots,B_n$, a to počínaje symbolem B_n (symbol B_1 pak bude na vrcholu zásobníku). Proto v novém automatu ze stavu q po vyjmutí symbolu B_1 přecházíme do stavu u_1, atd. Až zpracujeme vše, co bylo vygenerováno ze symbolu Z (tj. všechny symboly $B_1,\dots,B_n$), dostaneme se do stavu u_n.

Příklad 3.8

Postup ukážeme na jazyku $L=\left\{a^nc\,b^nc^k\ ;\ n\ge0,\ k>0\right\}$

Tento jazyk rozpoznává zásobníkový automat $\mathcal{A}=(\{0,1\},\ \{a,b,c\},\ \{Z,a\},\ \delta,\ 0,\ Z,\ \emptyset)$

$\delta(0,a,X)=(0,aX),\quad X\in\{a,Z\}$
$\delta(0,c,X)=(1,X),\quad X\in\{a,Z\}$
$\delta(1,b,a)=(1,\varepsilon)$
$\delta(1,c,Z)=\{(1,Z),\ (1,\varepsilon)\}$

Vytvoříme automat $\mathcal{A}'$:

$\delta'\left(s,\varepsilon,Z'\right)=\left\{\left(s,[0,Z,0]\right),\ \left(s,[0,Z,1]\right)\right\}$
$\delta'\left(s,a,[0,X,0]\right)=\left\{\left(s,[0,a,0][0,X,0]\right),\ \left(s,[0,a,1][1,X,0]\right)\right\},\ X\in\{a,Z\}$
$\delta'\left(s,a,[0,X,1]\right)=\left\{\left(s,[0,a,0][0,X,1]\right),\ \left(s,[0,a,1][1,X,1]\right)\right\}$
$\delta'\left(s,c,[0,X,0]\right)=\left\{\left(s,[1,X,0]\right)\right\}$
$\delta'\left(s,c,[0,X,1]\right)=\left\{\left(s,[1,X,1]\right)\right\}$
$\delta'\left(s,b,[1,a,1]\right)=\{(s,\varepsilon)\}$

$$\delta'\Big(s,c,[1,Z,0]\Big) = \Big\{\Big(s,[1,Z,0]\Big)\Big\}$$
$$\delta'\Big(s,c,[1,Z,1]\Big) = \Big\{\Big(s,[1,Z,1]\Big),\ (s,\varepsilon)\Big\}$$

Nové zásobníkové symboly jsou možná přehledné z hlediska vytvoření, ale bude lepší nahradit je kratšími variantami podle následující tabulky.

Dlouhé označení	$\longrightarrow$	*Krátké označení*
$[0,Z,0]$	$\longrightarrow$	A
$[0,Z,1]$	$\longrightarrow$	B
$[1,Z,0]$	$\longrightarrow$	C
$[1,Z,1]$	$\longrightarrow$	D
$[0,a,0]$	$\longrightarrow$	E
$[0,a,1]$	$\longrightarrow$	F
$[1,a,0]$	$\longrightarrow$	G
$[1,a,1]$	$\longrightarrow$	H

Upravíme δ' funkci, uvedeme plnou specifikaci automatu:
$\mathcal{A}' = (\{s\},\ \{a,b,c\},\ \{Z',A,B,C,D,E,F,G,H\},\ \delta',\ s,\ Z',\ \emptyset)$

$\delta'(s,\varepsilon,Z') = \{(s,A),\ (s,B)\}$

$\delta'(s,a,A) = \{(s,EA),\ (s,FC)\}$
$\delta'(s,a,E) = \{(s,EE),\ (s,FG)\}$
$\delta'(s,a,B) = \{(s,EB),\ (s,FD)\}$
$\delta'(s,a,F) = \{(s,EF),\ (s,FH)\}$

$\delta'(s,b,H) = \{(s,\varepsilon)\}$

$\delta'(s,c,A) = \{(s,C)\}$
$\delta'(s,c,E) = \{(s,G)\}$
$\delta'(s,c,B) = \{(s,D)\}$
$\delta'(s,c,F) = \{(s,H)\}$

$\delta'(s,c,C) = \{(s,C)\}$
$\delta'(s,c,D) = \{(s,D),\ (s,\varepsilon)\}$

Ukázka zpracování slova *aacbbc* automatem $\mathcal{A}$:

$$(0,aacbbc,Z) \vdash (0,acbbc,aZ) \vdash (0,cbbc,aaZ) \vdash (1,bbc,aaZ) \vdash (1,bc,aZ) \vdash (1,c,Z) \vdash (1,\varepsilon,\varepsilon)$$

Ukázka zpracování slova *aacbbc* automatem $\mathcal{A}'$:

$$(s,aacbbc,Z') \vdash (s,aacbbc,B) \vdash (s,acbbc,FD) \vdash (s,cbbc,FHD) \vdash (s,bbc,HHD) \vdash$$
$$\vdash (s,bc,HD) \vdash (s,c,D) \vdash (s,\varepsilon,\varepsilon)$$

Totéž, ale bez nahrazení zásobníkových symbolů kratšími verzemi:

$$(s,aacbbc,Z') \vdash \Big(s,aacbbc,[0,Z,1]\Big) \vdash \Big(s,acbbc,[0,a,1][1,Z,1]\Big) \vdash$$
$$\vdash \Big(s,cbbc,[0,a,1][1,a,1][1,Z,1]\Big) \vdash (s,bbc,[1,a,1][1,a,1][1,Z,1]\Big) \vdash$$
$$\vdash \Big(s,bc,[1,a,1][1,Z,1]\Big) \vdash \Big(s,c,[1,Z,1]\Big) \vdash (s,\varepsilon,\varepsilon)$$

Věta 3.4 (Zásobníkový automat $\longrightarrow$ bezkontextová gramatika)

Ke každému zásobníkovému automatu $\mathcal{A}$ lze vytvořit bezkontextovou gramatiku G takovou, že $L(G) = L(\mathcal{A})$, tedy

$$\mathscr{L}(ZA) \subseteq \mathscr{L}(CF) \tag{3.7}$$

Postup

Když jsme dokazovali, že ke každé bezkontextové gramatice lze sestrojit ekvivalentní zásobníkový automat, vytvořili jsme jednostavový zásobníkový automat. Zde využijeme přesně opačný postup – podle jednostavového automatu vytvoříme gramatiku.

Prvním krokem tedy bude vytvoření jednostavového zásobníkového automatu $\mathcal{A}'$ k automatu $\mathcal{A}$ podle postupu popsaného v důkazu předchozího lemmatu. V druhém kroku vytvoříme gramatiku, jejíž neterminály vytvoříme ze zásobníkových symbolů.

Když oba kroky shrneme, postup je následující:

1. Inicializujeme výpočet:
   $\forall q \in Q: \quad S \to [q_0, Z_0, q]$
2. Pro všechny předpisy ve tvaru $\delta(p, a, Z) \ni (q, \varepsilon)$ (do zásobníku nic nevkládají) vytvoříme
   $[p, Z, q] \to a$
3. Pro všechny ostatní předpisy ve tvaru $\delta(p, a, Z) \ni (q, B_1 B_2 \dots B_n)$:
   $[p, Z, u_n] \to a[q, B_1, u_1][u_1, B_2, u_2] \dots [u_{n-1}, B_n, u_n]$
   pro každou kombinaci stavů $u_i \in Q,\ 1 \leq i \leq n$.

Příklad 3.9

Budeme pokračovat v předchozím příkladu. V zadání příkladu byl automat
$\mathcal{A} = (\{0,1\},\ \{a,b,c\},\ \{Z,a\},\ \delta,\ 0,\ Z,\ \emptyset)$
$\delta(0,a,X) = (0,aX), \quad X \in \{a,Z\}$
$\delta(0,c,X) = (1,X), \quad X \in \{a,Z\}$
$\delta(1,b,a) = (1,\varepsilon)$
$\delta(1,c,Z) = \{(1,Z),\ (1,\varepsilon)\}$

Podle popsaného postupu vytvoříme gramatiku $G = (N,T,P,S),\ T = \Sigma$. V tabulce níže jsou uvedena všechna pravidla:

V automatu:	V gramatice:
	$S \to [0,Z,0] \mid [0,Z,1]$
$\delta(0,a,Z) = (0,aZ)$	$[0,Z,0] \to a[0,a,0][0,Z,0] \mid a[0,a,1][1,Z,0]$ $[0,Z,1] \to a[0,a,0][0,Z,1] \mid a[0,a,1][1,Z,1]$
$\delta(0,a,a) = (0,aa)$	$[0,a,0] \to a[0,a,0][0,a,0] \mid a[0,a,1][1,a,0]$ $[0,a,1] \to a[0,a,0][0,a,1] \mid a[0,a,1][1,a,1]$
$\delta(0,c,Z) = (1,Z)$	$[0,Z,0] \to c[1,Z,0]$ $[0,Z,1] \to c[1,Z,1]$
$\delta(0,c,a) = (1,a)$	$[0,a,0] \to c[1,a,0]$ $[0,a,1] \to c[1,a,1]$
$\delta(1,b,a) = (1,\varepsilon)$	$[1,a,1] \to b$
$\delta(1,c,Z) = \{(1,Z),\ (1,\varepsilon)\}$	$[1,Z,0] \to c[1,Z,0]$ $[1,Z,1] \to c[1,Z,1] \mid c$

Zjednodušíme neterminály podle tabulky z předchozího příkladu a shrneme pravidla přepisující stejný neterminál:

$S \to A \mid B$
$A \to aEA \mid aFC \mid cC$
$B \to aEB \mid aFD \mid cD$
$C \to cC$
$D \to cD \mid c$
$E \to aEE \mid aFG \mid cG$
$F \to aEF \mid aFH \mid cH$
$H \to b$

Odstraníme pravidla, kde je na pravé straně G (protože pro tento symbol není žádné pravidlo) a redukujeme:

$S \to B$
$B \to aFD \mid cD$
$D \to cD \mid c$
$F \to aFH \mid cH$
$H \to b$

Ukázka generování slova *aacbbc*:

$S \Rightarrow B \Rightarrow aFD \Rightarrow aaFHD \Rightarrow aacHHD \Rightarrow aacbHD \Rightarrow aacbbD \Rightarrow aacbbcD \Rightarrow aacbbcc$

Důsledkem předchozích vět je ekvivalence tříd jazyků:

Důsledek 3.5

Třída jazyků rozpoznávaných zásobníkovými automaty ($\mathscr{L}(ZA)$) je ekvivalentní třídě bezkontextových jazyků ($\mathscr{L}(CF)$).

3.4 Zásobníkové automaty a uzávěrové vlastnosti bezkontextových jazyků

V sekci 2.4 na straně 62 jsme probrali téměř všechny operace, vzhledem k nimž je nebo není třída bezkontextových jazyků uzavřena, a dokázali jsme, že třída bezkontextových jazyků není uzavřena vzhledem k operaci průniku (str. 68). To však neplatí pro průnik s regulárním jazykem:

Věta 3.6

Třída bezkontextových jazyků je uzavřena vzhledem k průniku s regulárním jazykem.

Postup

Na rozdíl od jiných uzávěrových vlastností bezkontextových jazyků, zde konstrukci nebudeme provádět na gramatikách, ale na automatech. Postup bude podobný tomu, který jsme použili v předchozím semestru pro průnik dvou regulárních jazyků.

Je dán bezkontextový jazyk reprezentovaného zásobníkovým automatem $\mathcal{A}_1$ a regulární jazyk reprezentovaný konečným automatem $\mathcal{A}_2$:

$\mathcal{A}_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_0^{(1)}, Z_0^{(1)}, F_1)$ (rozpoznává koncovým stavem)
$\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^{(2)}, F_2)$

Sestrojíme $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F), \quad L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

Položme $\Sigma = \Sigma_1 \cup \Sigma_2,\ \Gamma = \Gamma_1,\ Z_0 = Z_0'$.

Výpočet v automatu $\mathcal{A}$ má být simultánní simulací obou původních automatů – slovo w, které má být rozpoznáno, dáme zároveň na vstup obou původních automatů. Automat $\mathcal{A}$ přijme slovo w, pokud v obou automatech bude existovat úspěšný výpočet od počáteční k některé koncové konfiguraci.

Stavy automatu $\mathcal{A}$ budou uspořádané dvojice stavů původních automatů, první prvek je stav automatu $\mathcal{A}_1$ a druhý je stav automatu $\mathcal{A}_2$. Uspořádaná dvojice zachycuje, v jakém stavu v původních automatech je právě simulovaný výpočet.

$Q = Q_1 \times Q_2 = \{[q_1, q_2]\ ;\ q_1 \in Q_1,\ q_2 \in Q_2\}$

Množina koncových stavů: $F = F_1 \times F_2 = \{[q_1, q_2]\ ;\ q_1 \in F_1,\ q_2 \in F_2\}$

Definujeme přechodovou funkci:

1. V každém kroku, ve kterém je čten symbol ze vstupu, se posouváme v obou simulovaných automatech, v zásobníkovém automatu také pracujeme se zásobníkem – pro každé $a \in \Sigma$, $q_1, p_1 \in Q_1,\ q_2, p_2 \in Q_2,\ Z \in \Gamma,\ \gamma \in \Gamma_1^*$:

   $\delta([q_1, q_2], a, Z) \ni ([p_1, p_2], \gamma) \quad \Longleftrightarrow \quad \delta_1(q_1, a, Z) \ni (p_1, \gamma),\ \delta_2(q_2, a) \ni p_2$

2. Vyřešíme odlišnost původních automatů při práci se vstupní abecedou. Zásobníkový automat nemusí v každém kroku číst ze vstupní pásky, kdežto konečný ano. Proto umožníme simulovanému konečnému automatu dělat ε-kroky, při kterých nebude číst ze vstupní pásky ani provádět změnu stavu – pro každé $q_1, p_1 \in Q_1,\ q_2 \in Q_2,\ Z \in \Gamma,\ \gamma \in \Gamma_1^*$:

   $\delta([q_1, q_2], \varepsilon, Z) \ni ([p_1, q_2], \gamma) \quad \Longleftrightarrow \quad \delta_1(q_1, \varepsilon, Z) \ni (p_1, \gamma)$

Příklad 3.10

Vezmeme bezkontextový jazyk $L_1 = \{ww^R\ ;\ w \in \{a, b\}^*\}$ a regulární jazyk $R = a^*$. Jejich průnikem je jazyk $L_2 = \{a^n a^n\ ;\ n \geq 0\} = \{a^{2n}\ ;\ n \geq 0\}$

Sestrojíme automat $\mathcal{A}_1,\ L(\mathcal{A}_1) = L_1$:

$\mathcal{A}_1 = (\{q_0, q_1, q_2\},\ \{a, b\},\ \{Z_0, a, b\},\ \delta_1,\ q_0,\ Z_0,\ \{q_2\})$, kde

$\delta_1(q_0, a, Z_0) = \{(q_0, aZ_0)\}$
$\delta_1(q_0, b, Z_0) = \{(q_0, bZ_0)\}$
$\delta_1(q_0, a, b) = \{(q_0, ab)\}$
$\delta_1(q_0, b, a) = \{(q_0, ba)\}$
$\delta_1(q_0, a, a) = \{(q_0, aa), (q_1, \varepsilon)\}$
$\delta_1(q_0, b, b) = \{(q_0, bb), (q_1, \varepsilon)\}$
$\delta_1(q_1, a, a) = \{(q_1, \varepsilon)\}$
$\delta_1(q_1, b, b) = \{(q_1, \varepsilon)\}$
$\delta_1(q_1, \varepsilon, Z_0) = \{(q_2, \varepsilon)\}$

Konečný automat pro jazyk $R = a^*$ je velmi jednoduchý:

$\mathcal{A}_2 = (\{r\}\ \{a\},\ \delta_2,\ r,\ \{r\})$, kde $\delta_2(r, a) = r$

Vytvoříme $\mathcal{A} = (Q,\ \{a, b\},\ \{Z_0, a, b\},\ \delta,\ [q_0, r],\ Z_0,\ F)$.

$\delta([q_0, r], a, Z_0) = \{[q_0, r], aZ_0])\}$
$\delta([q_0, r], a, a) = \{([q_0, r], aa), ([q_1, r], \varepsilon)\}$
$\delta([q_1, r], a, a) = \{[q_1, r], \varepsilon)\}$
$\delta([q_1, r], \varepsilon, Z_0) = \{([q_2, r], \varepsilon)\}$

Ještě doplníme:

$Q = \{[q_0, r],\ [q_1, r],\ [q_2, r]\}$
$F = \{[q_2, r]\}$

Jak vidíme, je jazyk L_2 průnikem bezkontextového a regulárního jazyka, proto (i bez konstrukce automatu rozpoznávajícího tento jazyk) můžeme říci, že je to bezkontextový jazyk.

Příklad 3.11

Pomocí uzávěrových vlastností dokážeme, že následující jazyk není bezkontextový:
$L = \{w \in \{a,b,c\}^* \ ; \ |w|_a = |w|_b = |w|_c\}$ (stejný počet a, b a c)

Důkaz povedeme sporem. Předpokládejme, že jazyk L je bezkontextový. Pak by průnikem tohoto jazyka s jakýmkoliv regulárním jazykem byl také bezkontextový jazyk. Vezmeme regulární jazyk $R = a^*b^*c^*$. Jejich průnikem je

$L \cap R = L' = \{a^n b^n c^n \ ; \ n \geq 0\}$

O tomto jazyce však víme, že není bezkontextový, proto $L \notin \mathscr{L}(CF)$.

3.5 Deterministické bezkontextové jazyky

3.5.1 Deterministický zásobníkový automat

U regulárních jazyků platí, že ke každému (nedeterministickému) konečnému automatu lze sestrojit ekvivalentní deterministický. U bezkontextových jazyků tomu tak není.

Definujeme deterministický zásobníkový automat a následně ukážeme, že třída jazyků rozpoznávaných deterministickými zásobníkovými automaty je vlastní podmnožinou třídy bezkontextových jazyků.

Definice 3.5 (Deterministický zásobníkový automat)

Zásobníkový automat $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *je deterministický, jestliže pro každé* $q \in Q$, $Z \in \Gamma$ *platí zároveň*

- $\delta(q, a, Z)$ *má nejvýše jeden prvek pro každé* $a \in \Sigma \cup \{\varepsilon\}$.
- *je-li* $\delta(q, \varepsilon, Z) \neq \emptyset$, *pak* $\delta(q, a, Z) = \emptyset \ \forall a \in \Sigma$.

To znamená, že v deterministickém zásobníkovém automatu máme v každém kroku právě jednu možnost, jak reagovat (i včetně rozhodování, zda máme nebo nemáme číst ze vstupní pásky).

Definice 3.6 (Deterministický bezkontextový jazyk)

Jazyk L *je deterministickým bezkontextovým jazykem, jestliže existuje deterministický zásobníkový automat (DZA)* $\mathcal{A}_D$ *takový, že* $L(\mathcal{A}_D) = L$.

Z definice vyplývá, že pro každé slovo patřící do jazyka existuje právě jeden výpočet v $\mathcal{A}_D$. Deterministické bezkontextové jazyky budeme značit DCF a třídu jazyků, které generují, $\mathscr{L}(DCF)$.

Věta 3.7

Třída jazyků generovaných deterministickými zásobníkovými automaty je vlastní podmnožinou třídy bezkontextových jazyků:

$$\mathscr{L}(DCF) \subset \mathscr{L}(CF) \tag{3.8}$$

Důkaz: To, že platí $\mathscr{L}(DCF) \subseteq \mathscr{L}(CF)$, je zřejmé – vyplývá to z toho, že deterministický zásobníkový automat je vlastně speciálním případem (obecného) zásobníkového automatu, a víme, že třída jazyků generovaných bezkontextovými jazyky je ekvivalentní třídě jazyků rozpoznávaných zásobníkovými automaty.

Vlastní inkluzi (tj. $\mathscr{L}(DZA) \subsetneq \mathscr{L}(CF)$) lze dokázat tak, že najdeme jazyk patřící do druhé, ale nepatřící do první třídy. Bezkontextovým jazykem, který není deterministickým bezkontextovým, je například
$L = \{ww^R \ ; \ w \in \{a,b\}^*\}$.

Zásobníkový automat pro tento jazyk je v příkladu na straně 96, v každém případě jde automat nedeterministický – nevíme, ve kterém okamžiku vlastně přecházíme do druhé poloviny rozpoznávaného slova (nemáme možnost si předem tuto informaci zjistit a obě poloviny slova mají podobnou strukturu), a proto v každém kroku při načítání první poloviny slova potřebujeme možnost nedeterministicky zvolit buď pokračování v první polovině slova, anebo přechod do druhé. □

3.5.2 Uzávěrové vlastnosti deterministických bezkontextových jazyků

Věta 3.8

Pro třídu jazyků $\mathscr{L}(DCF)$ platí následující:

- *je uzavřena vzhledem k operaci průniku s regulárním jazykem,*
- *není uzavřena vzhledem k operaci průniku.*

Postup i důkaz je stejný jako u (obecně) bezkontextových jazyků.

Dále budeme potřebovat tuto pomocnou větu:

Lemma 3.9

Ke každému DZA $\mathcal{A}$ lze zkonstruovat ekvivalentní DZA $\mathcal{A}'$, který každý vstup dočte do konce.

Důkaz je složitý, je třeba vyřešit problém zacyklení v epsilonových krocích (práce se zásobníkem). Ovšem důsledek lemmatu je pro nás důležitý – znamená to, že DZA lze zkonstruovat tak, aby dokázal v každé konfiguraci reagovat, tedy se jedná o konstrukci totální přechodové funkce (jako zúplnění u konečných automatů), přičemž zároveň řešíme problém zacyklení.

Věta 3.10

Třída jazyků $\mathscr{L}(DCF)$ je uzavřena vzhledem k operaci doplňku.

Postup

Podle předchozího lemmatu lze pro jakýkoliv DZA sestrojit takový ekvivalentní DZA, který vstup dočte do konce, a tedy dokáže v konečném počtu kroků rozhodnout, zda slovo patří nebo nepatří do jazyka rozpoznávaného tímto automatem. Proto je postup následující:

- sestrojíme k původnímu automatu A deterministický zásobníkový automat A', který čte každý vstup až do konce (také má totální přechodovou funkci),
- zaměníme koncové a nekoncové stavy.

Věta 3.11

Třída jazyků $\mathscr{L}(DCF)$ není uzavřena vzhledem k operaci sjednocení.

Důkaz: Vyplývá z De Morganových zákonů:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \tag{3.9}$$

Předpokládejme, že třída jazyků $\mathscr{L}(DCF)$ je uzavřena vzhledem k operaci sjednocení. Pak by na pravé straně vztahu (3.9) byl jazyk ze třídy deterministických bezkontextových jazyků (protože podle předchozích vět je $\mathscr{L}(DCF)$ uzavřena vzhledem k operaci doplňku), jenže na pravé straně vztahu se může vyskytnout i jazyk, který není deterministický bezkontextový (tato třída jazyků není uzavřena vzhledem k operaci průniku). Spor $\Rightarrow$ třída jazyků $\mathscr{L}(DCF)$ nemůže být uzavřena vzhledem k operaci sjednocení. □

Důsledek 3.12

$$\mathscr{L}(DCF) \subset \mathscr{L}(CF) \tag{3.10}$$

Kapitola 4

Jazyky typu 0

V této kapitole se budeme zabývat nejvyšší třídou jazyků Chomského hierarchie s (téměř) obecným tvarem pravidel, jazyky typu 0. Stručně se podíváme na gramatiky typu 0 a pak se budeme věnovat především základní variantě Turingova stroje.

4.1 Gramatiky typu 0

Gramatiky typu 0 (frázové gramatiky) mají v Chomského hierarchii nejobecnější tvar pravidel. Jediným požadavkem je existence alespoň jednoho neterminálu na levé straně pravidla.

Definice 4.1 (Gramatika typu 0)

Gramatika typu 0 je taková gramatika, jejíž všechna pravidla jsou ve tvaru

$$\alpha \rightarrow \beta, \quad \alpha \in (N \cup T)^* N (N \cup T)^*, \ \beta \in (N \cup T)^* \tag{4.1}$$

4.2 Stroje rozpoznávající jazyky typu 0

Existují dva typy strojů (matematických modelů), které rozpoznávají jazyky typu 0 – zásobníkový automat rozšířený o další zásobník a Turingův stroj.

4.2.1 Zásobníkový automat se dvěma zásobníky

Můžeme mít jakýkoliv počet zásobníků, ale dá se dokázat, že k rozpoznávání jazyků typu 0 stačí dva zásobníky.

Definice 4.2 (Zásobníkový automat se dvěma zásobníky)

Zásobníkový automat se dvěma zásobníky je $\mathcal{A} = (Q, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, Z_1, Z_2, F)$*, kde*

- Γ_1 *je abeceda prvního zásobníku,* $Z_1 \in \Gamma_1$ *je počáteční zásobníkový symbol prvního zásobníku,*
- Γ_2 *je abeceda druhého zásobníku,* $Z_2 \in \Gamma_2$ *je počáteční zásobníkový symbol druhého zásobníku,*

- *δ funkce je definována takto:*
  $\delta\colon\ Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma_1 \times \Gamma_2 \ \to\ Q \times \Gamma_1^* \times \Gamma_2^*$
  $\delta(q_1, a, b_1, b_2) \in (q_2, \gamma_1, \gamma_2),\ a \in \Sigma \cup \{\varepsilon\},\ b_1 \in \Gamma_1,\ b_2 \in \Gamma_2,\ \gamma_1 \in \Gamma_1^*,\ \gamma_2 \in \Gamma_2^*.$

Vše ostatní se přejímá z definice zásobníkového automatu (s jedním zásobníkem).

Definice 4.3 (Konfigurace a přechod mezi konfiguracemi)

Konfigurace *zásobníkového automatu se dvěma zásobníky* $\mathcal{A} = (Q, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, Z_1, Z_2, F)$ *je*

$$(q, w, \gamma_1, \gamma_2) \in Q \times \Sigma^* \times \Gamma_1^* \times \Gamma_2^* \tag{4.2}$$

(stav, nepřečtená část vstupu, obsah prvního zásobníku, obsah druhého zásobníku). Počáteční konfigurace je (q_0, w_0, Z_1, Z_2)*, konečnou konfiguraci určujeme podle typu zásobníkového automatu (ukončení s prázdnými zásobníky nebo koncovým stavem).*

Relaci přechodu mezi konfiguracemi *zásobníkového automatu se dvěma zásobníky značíme* $\vdash$ *a definujeme takto:*

$$(q_i, a\mu, b_1\gamma_1, b_2\gamma_2) \vdash (q_j, \mu, \beta_1\gamma_1, \beta_2\gamma_2) \quad \Longleftrightarrow \quad \delta(q_i, a, b_1, b_2) \ni (q_j, \beta_1, \beta_2) \tag{4.3}$$

kde $q_i, q_j \in Q,\ a \in \Sigma,\ \mu \in \Sigma^*,\ b_1 \in \Gamma_1,\ b_2 \in \Gamma_2,\ \beta_1 \in \Gamma_1,\ \beta_2 \in \Gamma_2.$

Podobně jako u zásobníkových automatů s jedním zásobníkem je definován také reflexivní a tranzitivní uzávěr relace a jazyk rozpoznávaný automatem, to necháváme na čtenáři, definice budou prakticky stejné.

Příklad 4.1

Vytvoříme zásobníkový automat se dvěma zásobníky pro jazyk, o kterém víme, že není bezkontextový: $L = \{a^n b^n c^n\ ;\ n \geq 0\}$

$\mathcal{A} = (\{q_0, q_1, q_2\},\ \{a, b, c\},\ \{a, Z_1\},\ \{b, Z_2\},\ \delta,\ Z_1,\ Z_2,\ \emptyset)$

Přechodová funkce pracuje takto:

- v první fázi (stav q_0) načítáme symboly a a ukládáme je do prvního zásobníku, druhý zásobník zatím není používán,
- v druhé fázi (stav q_1) načítáme symboly b, přitom vyjímáme z prvního zásobníku symboly a (tak je zajištěn stejný počet a a b) a zároveň ukládáme symboly b do druhého zásobníku,
- v třetí fázi (stav q_2) musí již být první zásobník prázdný, načítáme ze vstupu symboly c a zároveň vyjímáme z druhého zásobníku symboly b (tak je zajištěn stejný počet b a c).

$$\begin{array}{llll}
\delta(q_0, a, Z_1, Z_2) = (q_0, aZ_1, Z_2) & \qquad & \delta(q_1, c, Z_1, b) = (q_2, Z_1, \varepsilon) \\
\delta(q_0, a, a, Z_2) = (q_0, aa, Z_2) & & \delta(q_2, c, Z_1, b) = (q_2, Z_1, \varepsilon) \\
\delta(q_0, b, a, Z_2) = (q_1, \varepsilon, bZ_2) & & \delta(q_0, \varepsilon, Z_1, Z_2) = (q_0, \varepsilon, \varepsilon) \\
\delta(q_1, b, a, b) = (q_1, \varepsilon, bb) & & \delta(q_2, \varepsilon, Z_1, Z_2) = (q_2, \varepsilon, \varepsilon)
\end{array}$$

Ukázka zpracování slova *aabbcc*:

$(q_0, aabbcc, Z_1, Z_2) \vdash (q_0, abbcc, aZ_1, Z_2) \vdash (q_0, bbcc, aaZ_1, Z_2) \vdash (q_1, bcc, aZ_1, bZ_2) \vdash$
$\vdash (q_1, cc, Z_1, bbZ_2) \vdash (q_2, c, Z_1, bZ_2) \vdash (q_2, \varepsilon, Z_1, Z_2) \vdash (q_2, \varepsilon, \varepsilon, \varepsilon)$

4.2.2 Turingův stroj

Činnost Turingova stroje jsme si již trochu (zatím neformálně) osvětlili v předchozím semestru. Shrneme si základní vlastnosti:

- konečná (konečněstavová) řídicí jednotka,
- nekonečná páska (obvykle směrem doprava nekonečná),
- čtecí a zápisová hlava může číst symbol z pásky, přepsat ho jiným symbolem, pohybuje se o jedno pole doleva nebo doprava,
- výpočet končí při přechodu do některého koncového stavu, nemusí být přečtena celá páska.

Formální definice Turingova stroje je trochu podobná definici konečného automatu, ale máme zvlášť vstupní abecedu a páskovou abecedu (symboly ze vstupní abecedy mohou být v počáteční konfiguraci), a ovšem je jiná definice přechodové funkce.

Zatímco v konečném automatu je třeba v každém kroku zpracovat jeden symbol, nelze zapisovat a vždy se posouváme o jedno pole doprava, v Turingově stroji sice také v každém kroku zpracováváme jeden symbol, ale můžeme ho přepsat na jiný a možnosti pohybu jsou různé.

Definice 4.4 (Turingův stroj)

Turingův stroj je uspořádaná šestice $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$*, kde*

- Q *je konečná neprázdná množina stavů,*
- Σ *je konečná neprázdná vstupní abeceda (symboly, ze kterých se skládá vstupní slovo),* $\Sigma \subseteq \Gamma$*,*
- Γ *je konečná neprázdná pásková abeceda (symboly, které se mohou vyskytovat na pásce),*
- $q_0 \in Q$ *je počáteční stav,*
- $F \subseteq Q$ *je množina koncových stavů,*
- δ *je přechodová funkce:*
  $\delta\colon\ Q \times \Gamma \ \to\ Q \times \Gamma \times \{-1, 0, 1\}$,
  $\delta(q_i, a) = (q_j, b, P), \quad q_i, q_j \in Q, \quad a, b \in \Gamma, \quad P \in \{-1, 0, 1\}$

Z definice přechodové funkce vyplývá, že v každém kroku, kdy je použito $\delta(q_i, a) = (q_j, b, P)$, $q_i, q_j \in Q,\ a, b \in \Gamma,\ P \in \{-1, 0, 1\}$, jsou provedeny následující akce:

- jsme ve stavu q_i a na pásce čtecí a zápisová hlava právě ukazuje na políčko označené a,
- přejdeme do stavu q_j,
- symbol a na pásce přepíšeme symbolem b,
- posuneme čtecí a zápisovou hlavu podle předpisu P.

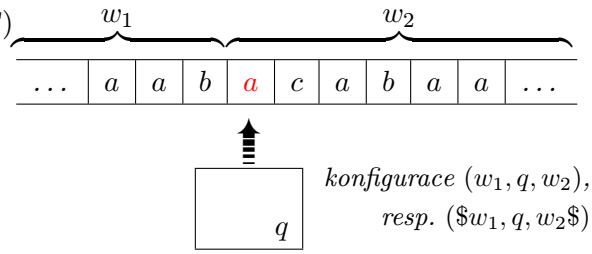
Takto definovaný Turingův stroj je deterministický.

Prázdná políčka pásky se obvykle označují symbolem $\sqcup$ (nebo písmenem B, Blank), slovo je od prázdných políček odděleno (tedy obklopeno) symboly \$, tyto symboly v přechodové funkci pomáhají zjistit, zda jsme na začátku či konci slova. Je možné stanovit také dva různé symboly – jeden pro vymezení začátku a druhý pro vymezení konce slova (například \$ a #). Hraniční symbol (-y) je také součástí páskové abecedy.

Množina F koncových stavů bývá často tvořena dvěma stavy, jedním pro přijetí a jedním pro odmítnutí slova: $F = \{q_{\text{accept}}, q_{\text{reject}}\}$, případně místo q_{reject} je někdy použito q_{error}.

Definice 4.5 (Konfigurace Turingova stroje)

Konfigurace Turingova stroje $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *je* (w_1, q, w_2)*, kde* $w_1 \in \Gamma$ *je část pásky před čtecí a zápisovou hlavou,* $q \in Q$ *je stav, ve kterém se řídicí jednotka nachází, a* $w_2 \in \Gamma$ *je část pásky za čtecí a zápisovou hlavou, čtecí a zápisová hlava ukazuje na první symbol řetězce* w_2*.*



Počáteční konfigurace *je* (ε, q_0, w_0) *nebo* $(\$, q_0, w_0\$)$ *(pokud chceme zahrnout do konfigurace i hraniční symboly),* $w_0 \in \Sigma^*$ *je vstupní slovo, které má být zpracováno.*

Koncová konfigurace *je* (w_1, q_f, w_2) *(resp.* $(\$w_1, q_f, w_2\$)$*),* $q_f \in F$*,* $w_1, w_2 \in \Gamma^*$*.*

Příklad 4.2

Například konfigurace $(abbca, q, daab)$ znamená, že se stroj nachází ve stavu q, na pásce je slovo $abbcadaab$ a čtecí a zápisová hlava ukazuje na šestý symbol slova – d.

Alternativně bychom tuto konfiguraci zapsali $(\$abbca, q, daab\$)$.

Definice 4.6 (Relace přechodu mezi konfiguracemi)

Relace přechodu mezi konfiguracemi je určena takto:

$$(\alpha, q_i, a\beta) \vdash (\alpha b, q_j, \beta) \iff \delta(q_i, a) = (q_j, b, 1) \tag{4.4}$$

$$(\alpha, q_i, a\beta) \vdash (\alpha, q_j, b\beta) \iff \delta(q_i, a) = (q_j, b, 0) \tag{4.5}$$

$$(\alpha c, q_i, a\beta) \vdash (\alpha, q_j, cb\beta) \iff \delta(q_i, a) = (q_j, b, -1) \tag{4.6}$$

V prvním případě se čtecí a zápisová hlava posunuje doprava, v druhém zůstává na místě (tj. v dalším kroku bude číst totéž políčko jako v tomto) a v třetím případě se posunuje doleva.

Příklad 4.3

Sestrojíme Turingův stroj rozpoznávající jazyk $L = \{a^n b^n c^n \ ; \ n \geq 0\}$

Budeme postupovat takto: označíme první a (tj. přepíšeme symbolem $\bar{a}$), najdeme první b, označíme ho, pak najdeme první c, taktéž označíme, potom přejdeme na začátek (postupujeme doleva, dokud nenajdeme nejbližší označené $\bar{a}$), posuneme se o jedno pole doprava na první neoznačené a, označíme ho, atd. Tedy slovo zpracováváme v „rundách“ (průchodech) – v každé rundě zpracujeme jedno a, jedno b a jedno c. Tím zajišťujeme synchronizaci zpracování všech tří částí slova.

Plná specifikace bude následující (její části si dále postupně vysvětlíme):

$\mathcal{M} = (\{q_0, q_P, q_A, q_B, q_C, q_f, q_{\text{accept}}\},\ \{a, b, c\},\ \{a, \bar{a}, b, \bar{b}, c, \bar{c}, \sqcup, \$\},\ \delta,\ \{q_{\text{accept}}\})$

V různých stavech bude TS provádět odlišnou činnost, takže podobně jako u zásobníkových automatů, i zde dělíme různé módy činnosti automatu pomocí různých stavů. Jednotlivé stavy znamenají:

- q_0 – začátek výpočtu a začátek první „rundy“; označíme nejbližší a (přepíšeme na $\bar{a}$) a přejdeme do stavu q_A; pokud nenajdeme žádné a, pak je zřejmě na vstupu prázdné slovo, tj. přecházíme do finálního módu (stavu) q_f,
- q_P – začátek „rundy“; má podobnou funkci jako q_0 v první rundě, tedy označíme nejbližší a, přejdeme do stavu q_A a posuneme se vpravo; pokud nenajdeme žádné a, pak jsou všechna označena (za posledním $\bar{a}$ bude $\bar{b}$) a přejdeme do finálního módu q_f,
- q_A – označili jsme a, přeskakujeme symboly $a, \bar{b}$, hledáme první neoznačené b,
- q_B – označili jsme b, přeskakujeme symboly $b, \bar{c}$, hledáme první neoznačené c,
- q_C – označili jsme c, vracíme se na začátek k poslednímu označenému $\bar{a}$, při pohybu doleva přeskakujeme všechny symboly $\bar{c}, b, \bar{b}, a$,
- q_f – finální mód, ve kterém musíme zkontrolovat, jestli nám v druhé nebo třetí části nezůstaly nějaké neoznačené symboly (kdyby zůstaly, pak slovo na vstupu je chybné) – procházíme všechna $\bar{b}$ a $\bar{c}$, a až narazíme na koncovou zarážku (\$ na konci slova), končíme výpočet.

Definujeme δ funkci:

$\delta(q_0, \$) = (q_{accept}, 0)$ (přijali jsme prázdné slovo)

$\delta(q_0, a) = (q_A, \bar{a}, 1)$
$\delta(q_P, a) = (q_A, \bar{a}, 1)$
$\delta(q_A, a) = (q_A, a, 1)$
$\delta(q_A, \bar{b}) = (q_A, \bar{b}, 1)$
$\delta(q_A, b) = (q_B, \bar{b}, 1)$

$\delta(q_B, b) = (q_B, b, 1)$
$\delta(q_B, \bar{c}) = (q_B, \bar{c}, 1)$
$\delta(q_B, c) = (q_C, \bar{c}, -1)$
$\delta(q_C, X) = (q_C, X, -1),$
$X \in \{a, b, \bar{b}, \bar{c}\}$

$\delta(q_C, \bar{a}) = (q_P, \bar{a}, 1)$
$\delta(q_P, \bar{b}) = (q_f, \bar{b}, 1)$
$\delta(q_f, \bar{b}) = (q_f, \bar{b}, 1)$
$\delta(q_f, \bar{c}) = (q_f, \bar{c}, 1)$
$\delta(q_f, \$) = (q_{accept}, \$, 0)$

Ukázka zpracování slova abc:

$(\varepsilon, q_0, abc) \vdash (\bar{a}, q_A, bc) \vdash (\bar{a}\bar{b}, q_B, c) \vdash (\bar{a}, q_C, \bar{b}\bar{c}) \vdash (\varepsilon, q_C, \bar{a}\bar{b}\bar{c}) \vdash (\bar{a}, q_0, \bar{b}\bar{c}) \vdash (\bar{a}\bar{b}, q_f, \bar{c}) \vdash$
$\vdash (\bar{a}\bar{b}\bar{c}, q_f, \varepsilon) \vdash (\bar{a}\bar{b}\bar{c}, q_{accept}, \varepsilon)$

Jestliže zaznamenáme i hraniční symboly, zpracování bude vypadat takto:

$(\$, q_0, abc\$) \vdash (\$\bar{a}, q_A, bc\$) \vdash (\$\bar{a}\bar{b}, q_B, c\$) \vdash (\$\bar{a}, q_C, \bar{b}\bar{c}\$) \vdash (\$, q_C, \bar{a}\bar{b}\bar{c}\$) \vdash (\$\bar{a}, q_0, \bar{b}\bar{c}\$) \vdash$
$\vdash (\$\bar{a}\bar{b}, q_f, \bar{c}\$) \vdash (\$\bar{a}\bar{b}\bar{c}, q_f, \$) \vdash (\$\bar{a}\bar{b}\bar{c}, q_{accept}, \$)$

Ukázka zpracování slova $aabbcc$:

$(\varepsilon, q_0, aabbcc) \vdash (\bar{a}, q_A, abbcc) \vdash (\bar{a}a, q_A, bbcc) \vdash (\bar{a}a\bar{b}, q_B, bcc) \vdash (\bar{a}a\bar{b}b, q_B, cc) \vdash (\bar{a}a\bar{b}, q_C, b\bar{c}c) \vdash$
$\vdash (\bar{a}a, q_C, \bar{b}b\bar{c}c) \vdash (\bar{a}, q_C, a\bar{b}b\bar{c}c) \vdash (\varepsilon, q_C, \bar{a}a\bar{b}b\bar{c}c) \vdash (\bar{a}, q_0, a\bar{b}b\bar{c}c) \vdash (\bar{a}\bar{a}, q_A, \bar{b}b\bar{c}c) \vdash$
$\vdash (\bar{a}\bar{a}\bar{b}, q_A, b\bar{c}c) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}, q_B, \bar{c}c) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}\bar{c}, q_B, c) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}, q_C, \bar{c}\bar{c}) \vdash (\bar{a}\bar{a}\bar{b}, q_C, \bar{b}\bar{c}\bar{c}) \vdash (\bar{a}\bar{a}, q_C, \bar{b}\bar{b}\bar{c}\bar{c}) \vdash (\bar{a}, q_C, \bar{a}\bar{b}\bar{b}\bar{c}\bar{c}) \vdash$
$\vdash (\bar{a}\bar{a}, q_0, \bar{b}\bar{b}\bar{c}\bar{c}) \vdash (\bar{a}\bar{a}\bar{b}, q_f, \bar{b}\bar{c}\bar{c}) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}, q_f, \bar{c}\bar{c}) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}\bar{c}, q_f, \bar{c}) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}\bar{c}\bar{c}, q_f, \varepsilon) \vdash (\bar{a}\bar{a}\bar{b}\bar{b}\bar{c}\bar{c}, q_{accept}, \varepsilon)$

Ukázka zpracování slova ac:

$(\varepsilon, q_0, ac) \vdash (\bar{a}, q_A, c)$ chyba $\Rightarrow$ slovo ac není přijato.

Ukázka zpracování slova ε:

$(\varepsilon, q_0, \varepsilon) \vdash (\varepsilon, q_{accept}, \varepsilon)$

Poznámka:

Všimněme si rozdílu mezi činností Turingova stroje a dříve definovaných automatů:

- Turingův stroj nejen čte vstupní pásku, může ji i zapisovat,
- čtecí a zápisová hlava se může (nemusí) pohybovat různými směry, nejen doprava,
- nepotřebujeme zásobník, ale přesto můžeme uchovávat i jinou informaci než označení stavu, ve kterém právě jsme – kamkoliv na pásku si můžeme cokoliv poznamenat a později tuto informaci využít,
- pomocí Turingova stroje lze provádět i výpočty.

Turingovými stroji se budeme podrobněji zabývat v předmětu Teorie vyčíslitelnosti.

4.2.3 Varianty Turingova stroje

Řekli jsme si, že Turingův stroj je v základní definici deterministický. Proto varianty můžeme především rozdělit podle tohoto kritéria:

- *deterministický* – základní varianta,
- *nedeterministický* – pro tentýž stav a obsah pásky lze definovat více různých akcí.

Podobně, jako zásobníkový automat může mít více zásobníků, Turingův stroj může mít více pásek, přičemž s každou páskou pracuje nezávisle:

- *jednopáskový* – základní varianta,
- *vícepáskový* – máme více pásek, každá má vlastní čtecí a zápisovou hlavu, tyto hlavy se mohou pohybovat navzájem nezávisle (například první doprava, druhá doleva a třetí třeba zůstane na místě v tomtéž kroku zpracování).

Na jedné pásce může být i více stop, ale všechny stopy na jedné pásce mají společnou čtecí a zápisovou hlavu:

- *jednostopý* – na (každé) pásce je jen jedna stopa,
- *vícestopý* – na pásce může být více stop.

Vícestopý Turingův stroj tedy svou čtecí a zápisovou hlavou z pásky nečte jen jeden symbol, ale přímo celý vektor (symboly ze všech stop na stejném místě najednou).

Dále můžeme stanovit možnost pohybu čtecí a zápisové hlavy:

- *možnost pohybu* $\{-1, 0, 1\}$ – čtecí a zápisová hlava se může pohybovat doleva nebo doprava, a nebo zůstat na místě,
- *možnost pohybu* $\{-1, 1\}$ – čtecí a zápisová hlava se musí pohybovat v každém kroku, a to doleva nebo doprava, nesmí zůstat na místě.

Páska může být

- *jednostranně nekonečná* – vstupní slovo je na začátku výpočtu umístěno na začátek pásky, před ně již není možné nic napsat,
- *oboustranně nekonečná* – základní varianta.

Lze dokázat, že všechny výše uvedené varianty jsou navzájem ekvivalentní, všechny varianty lze převést na základní variantu – deterministický jednopáskový jednostopý automat, jehož čtecí a zápisová hlava se může pohybovat oběma směry nebo zůstat na místě a lze zapisovat i před vstupní slovo.

4.3 Vztah Turingových strojů k jazykům typu 0

Zde ukážeme, že jazyky rozpoznávané Turingovým strojem jsou právě jazyky typu 0. Pro tuto vlastnost se také jazykům typu 0 říká *rekurzívně spočetné jazyky*, protože Turingův stroj vlastně pracuje na principu rekurze.

Definice 4.7 (Rekurzívně spočetný jazyk)

Jazyk nazveme rekurzívně spočetný (rekurzívně vyčíslitelný, částečně rekurzivní), pokud je přijímán *nějakým Turingovým strojem (tento Turingův stroj se na slovo patřící do jazyka zastaví v akceptujícím stavu, na slovo nepatřící do jazyka se buď zastaví v odmítajícím stavu nebo se dostane do nekonečné smyčky).*

Definice 4.8 (Rekurzívní jazyk)

Jazyk nazveme rekurzívní, pokud je rozhodován *nějakým Turingovým strojem (tento Turingův stroj se pro jakékoliv slovo zastaví, a to: na slovo jazyka v akceptujícím stavu a na slovo nepatřící do jazyka v odmítajícím stavu, pro žádný vstup nepřejde do nekonečné smyčky).*

Takže tu máme následující vztahy:

- rekurzívně spočetný — přijímán TS — na slova patřící do jazyka TS se zastaví
- rekurzívní — rozhodován TS — zastaví se na všechna slova (na slova patřící do jazyka TS v přijímajícím stavu, na zbylá v odmítajícím stavu).

4.3.1 Vytvoření Turingova stroje podle gramatiky

Věta 4.1 (Gramatika $\longrightarrow$ Turingův stroj)

Ke každé gramatice G typu 0 lze sestroji Turingův stroj $\mathcal{M}$ takový, že platí $L(\mathcal{M}) = L(G)$.

Postup

Podle gramatiky $G = (N, T, P, S)$ typu 0 sestrojíme nedeterministický dvoustopý Turingův stroj $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

Postup stojí na tom, že Turingův stroj bude simulovat derivaci slova v gramatice – první stopa obsahuje vstupní slovo, nemění se, slouží pro kontrolu; druhá stopa pak bude „pracovní“: obsahuje

větnou formu v derivaci v daném kroku (tj. na začátku to bude pouze S). Každý krok derivace v gramatice bude simulován sekvencí kroků Turingova stroje (jednou rundou kroků) takto:

1. Nedeterministicky zvolíme některé pravidlo $\alpha \to \beta$ v gramatice.
2. Pokud se α nachází ve větné formě, zvolíme některý výskyt α a přepíšeme ho na β; pokud $|\beta| \neq |\alpha|$, nejdřív vhodně posuneme všechny symboly za řetězcem α doleva nebo doprava.
3. Porovnáme vstup na první stopě s obsahem druhé stopy – pokud je stejný, vstup přijmeme, jinak zpět k bodu 1.

Příklad 4.4

Vytvoříme gramatiku typu 0 pro jazyk $L = \{a^n b^n c^n \ ; \ n \geq 1\}$. Gramatika generující jazyk L je $G = (\{S, A, B, X\},\ \{a, b, c\},\ P,\ S)$, kde v P jsou pravidla

$S \to aAbX$
$Ab \to bA$
$AX \to BXc$
$AX \to c$
$bB \to Bb$
$aB \to aaAb$

Ukázka odvození slova *aabbcc*:

$S \Rightarrow aAbX \Rightarrow abAX \Rightarrow abBXc \Rightarrow aBbXc \Rightarrow aaAbbXc \Rightarrow aabAbXc \Rightarrow aabbAXc \Rightarrow aabbcc$

Podle této gramatiky sestrojíme Turingův stroj. Každý stav bude reprezentovat konkrétní „mód“ činnosti stroje – v každém stavu bude stroj určitým způsobem reagovat na dané vstupy. Význam jednotlivých stavů:

q_0 nedeterministicky vybereme pravidlo,

$q_1, q_2, \ldots, q_6$ vybrali jsme 1., 2., ..., 6. pravidlo, teď nedeterministicky vybereme, kde ho chceme použít,

q_{11} už jsme si vybrali místo pro uplatnění prvního pravidla, zpracovali jsme první symbol pravidla (první znak α přepíšeme na první znak β),

q_{12} přepisujeme druhý symbol pravidla ...

$\vdots$

q_{21} už jsme si vybrali místo pro uplatnění druhého pravidla, zpracovali jsme první symbol pravidla

$\vdots$

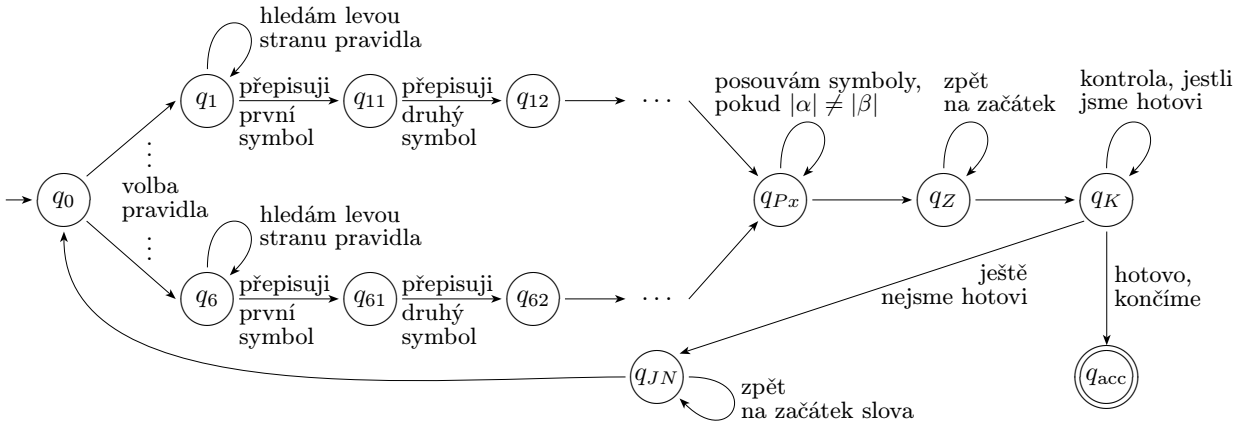
q_Z přepsali jsme celé pravidlo, vracíme se na začátek větné formy, bude další pravidlo,

q_K kontrola, jestli máme skončit,

q_{JN} ještě ne (ještě neskončit, stopy mají různý obsah),

q_{acc} akceptujeme.

Přecházení mezi těmito stavy je schematicky naznačeno na diagramu na obrázku 4.1.



Obrázek 4.1: Turingův stroj podle gramatiky

Postup si nejdřív ukážeme na průběhu výpočtu slova, které bylo v gramatice pro ukázku odvozeno. Na průběhu výpočtu vidíme, že horní stopa se nemění, zatímco na dolní probíhá simulace generování slova v gramatice.

$$\left(\begin{array}{c}\$\\\$\end{array}, q_0, \begin{array}{cccccccc} a & a & b & b & c & c & \$ \\ S & \$ & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash \left(\begin{array}{c}\$\\\$\end{array}, q_1, \begin{array}{cccccccc} a & a & b & b & c & c & \$ \\ S & \$ & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash$$

$$\vdash \left(\begin{array}{cc}\$ & a\\\$ & a\end{array}, q_{11}, \begin{array}{ccccccc} a & b & b & c & c & \$ \\ \$ & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash \left(\begin{array}{cc}\$ & a\\\$ & a\end{array}, q_{12}, \begin{array}{ccccc} a & b & b & c & c & \$ \\ A & \sqcup & \sqcup & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash$$

$$\vdash \left(\begin{array}{ccc}\$ & a & a\\\$ & a & A\end{array}, q_{13}, \begin{array}{ccccc} b & c & c & \$ \\ b & \sqcup & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash \left(\begin{array}{ccc}\$ & a & a\\\$ & a & A\end{array}, q_{14}, \begin{array}{ccccc} b & b & c & c & \$ \\ b & X & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash$$

$$\vdash \left(\begin{array}{ccc}\$ & a & a\\\$ & a & A\end{array}, q_Z, \begin{array}{ccccc} b & c & c & \$ \\ b & \$ & \sqcup & \sqcup & \sqcup \end{array}\right) \vdash \left(\begin{array}{ccc}\$ & a & a\\\$ & a & A\end{array}, q_Z, \begin{array}{ccccc} b & b & c & c & \$ \\ b & X & \$ & \sqcup & \sqcup \end{array}\right) \vdash$$

$$\vdash \left(\begin{array}{cc}\$ & a\\\$ & a\end{array}, q_Z, \begin{array}{ccccc} a & b & b & c & c & \$ \\ A & b & X & \$ & \sqcup & \sqcup \end{array}\right) \vdash \left(\begin{array}{c}\$\\\$\end{array}, q_Z, \begin{array}{cccccc} a & a & b & b & c & c & \$ \\ a & A & b & X & \$ & \sqcup & \sqcup \end{array}\right) \vdash$$

$$\vdash \left(\begin{array}{c}\$\\\$\end{array}, q_K, \begin{array}{cccccc} a & a & b & b & c & c & \$ \\ a & A & b & X & \$ & \sqcup & \sqcup \end{array}\right) \vdash \ldots \vdash \left(\begin{array}{c}\$\\\$\end{array}, q_0, \begin{array}{cccccc} a & a & b & b & c & c & \$ \\ a & A & b & X & \$ & \sqcup & \sqcup \end{array}\right) \quad \vdash \ldots$$

$$\vdash \ldots \vdash \left(\begin{array}{cccccccc}\$ & a & a & b & b & c & c & \$\\\$ & a & a & b & b & c & c & \$\end{array}, q_{\text{acc}}\right)$$

Definujeme přechodovou funkci – pro každé $U \in \Sigma \cup \{\sqcup\},\ V \in \Gamma$ potřebujeme následující předpisy:

Nedeterministicky vybereme pravidlo gramatiky, které chceme uplatnit na druhou stopu:

$$\delta\left(q_0, \begin{array}{c}U\\V\end{array}\right) = \left\{\left(q_1, \begin{array}{c}U\\V\end{array}, 0\right), \left(q_2, \begin{array}{c}U\\V\end{array}, 0\right), \left(q_3, \begin{array}{c}U\\V\end{array}, 0\right), \ldots, \left(q_6, \begin{array}{c}U\\V\end{array}, 0\right)\right\}$$

Zpracování prvního pravidla – nejdřív nedeterministicky vybereme, na kterém místě řetězce pravidlo uplatníme (najdeme některý výskyt α), a pak pro $\alpha \to \beta$ začneme přepisovat α na β:

$$\delta\left(q_1, \begin{array}{c}U\\S\end{array}\right) = \left\{\left(q_1, \begin{array}{c}U\\S\end{array}, 1\right), \left(q_{11}, \begin{array}{c}U\\a\end{array}, 1\right)\right\}$$

$$\delta\left(q_1, \begin{array}{c}U\\M\end{array}\right) = \left(q_1, \begin{array}{c}U\\M\end{array}, 1\right),\ M \neq S$$

$$\delta\left(q_{11}, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_{12}, \begin{matrix} U \\ A \end{matrix}, 1\right) \qquad \delta\left(q_{12}, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_{13}, \begin{matrix} U \\ b \end{matrix}, 1\right)$$

$$\delta\left(q_{13}, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_{14}, \begin{matrix} U \\ X \end{matrix}, 1\right) \qquad \delta\left(q_{14}, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_Z, \begin{matrix} U \\ \$ \end{matrix}, -1\right)$$

$$\delta\left(q_Z, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_Z, \begin{matrix} U \\ V \end{matrix}, -1\right),\ V \neq \$ \text{ (jdeme na začátek větné formy)}$$

$$\delta\left(q_Z, \begin{matrix} \$ \\ \$ \end{matrix}\right) = \left(q_K, \begin{matrix} \$ \\ \$ \end{matrix}, 1\right) \text{ (jsme na začátku)}$$

$$\delta\left(q_K, \begin{matrix} U \\ U \end{matrix}\right) = \left(q_K, \begin{matrix} U \\ U \end{matrix}, 1\right) \text{ (kontrolujeme, jestli už jsme na konci odvození slova)}$$

$$\delta\left(q_K, \begin{matrix} \$ \\ \$ \end{matrix}\right) = \left(q_{\text{acc}}, \begin{matrix} \$ \\ \$ \end{matrix}, 0\right) \text{ (obě stopy mají stejný obsah } \Rightarrow \text{ konec odvození, konec práce)}$$

$$\delta\left(q_K, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_{JN}, \begin{matrix} U \\ V \end{matrix}, -1\right),\ V \neq U \text{ (ještě není konec, přejdeme na začátek pásky a zvolíme další pravidlo)}$$

$$\delta\left(q_{JN}, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_{JN}, \begin{matrix} U \\ V \end{matrix}, -1\right) \qquad \delta\left(q_{JN}, \begin{matrix} \$ \\ \$ \end{matrix}\right) = \left(q_0, \begin{matrix} \$ \\ \$ \end{matrix}, 1\right)$$

$$\delta\left(q_2, \begin{matrix} U \\ A \end{matrix}\right) = \left\{\left(q_2, \begin{matrix} U \\ A \end{matrix}, 1\right), \left(q_{21}, \begin{matrix} U \\ b \end{matrix}, 1\right)\right\} \text{ (zpracováváme druhé pravidlo)}$$

$$\delta\left(q_2, \begin{matrix} U \\ M \end{matrix}\right) = \left(q_2, \begin{matrix} U \\ M \end{matrix}, 1\right),\ M \neq A$$

$$\delta\left(q_{21}, \begin{matrix} U \\ b \end{matrix}\right) = \left(q_Z, \begin{matrix} U \\ A \end{matrix}, -1\right)$$

Třetí pravidlo je typu $|\alpha| < |\beta|$, vše za α posuneme doprava, aby se β vešla:

$$\delta\left(q_3, \begin{matrix} U \\ A \end{matrix}\right) = \left\{\left(q_3, \begin{matrix} U \\ A \end{matrix}, 1\right), \left(q_P, \begin{matrix} U \\ 3 \end{matrix}, 1\right)\right\} \text{ (symbol 3 je zarážka, abychom po posouvání věděli, kde máme začít psát řetězec } \beta)$$

$$\delta\left(q_3, \begin{matrix} U \\ M \end{matrix}\right) = \left(q_3, \begin{matrix} U \\ M \end{matrix}, 1\right),\ M \neq A$$

Funkce pro posun následujícího řetězce o 1 políčko doprava:

$$\delta\left(q_P, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_P, \begin{matrix} U \\ V \end{matrix}, 1\right),\ V \neq \$ \text{ (nejdřív se přesuneme na konec řetězce)}$$

$$\delta\left(q_P, \begin{matrix} U \\ \$ \end{matrix}\right) = \left(q_{P\$}, \begin{matrix} U \\ \$ \end{matrix}, 1\right)$$

$$\delta\left(q_{PM}, \begin{matrix} U \\ V \end{matrix}\right) = \left(q_{PZ}, \begin{matrix} U \\ M \end{matrix}, -1\right),\ M \in \{a, b, c, S, A, B, X, \$\}$$

$$\delta\left(q_{PZ}, \begin{matrix} U \\ M \end{matrix}\right) = \left(q_{PM}, \begin{matrix} U \\ M \end{matrix}, 1\right) \text{ (zapamatujeme si } M\text{, předchozí částí } \delta \text{ fce ho pak zkopírujeme vpravo)}$$

$\delta\left(q_{PZ}, \begin{matrix}U\\3\end{matrix}\right) = \left(q_{31}, \begin{matrix}U\\B\end{matrix}, 1\right)$ (zarážka 3 znamená, že jsme při uplatňování 3. pravidla gramatiky posunuli vše za tímto místem o 1 políčko doprava, teď můžeme zapsat pravou stranu pravidla, už se vejde)

$\delta\left(q_{31}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{32}, \begin{matrix}U\\X\end{matrix}, 1\right)$ $\quad$ $\delta\left(q_{32}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{Z}, \begin{matrix}U\\c\end{matrix}, -1\right)$

$\delta\left(q_{PZ}, \begin{matrix}U\\6\end{matrix}\right) = \left(q_{P}, \begin{matrix}U\\6'\end{matrix}, 1\right)$ (6. pravidlo gramatiky vyžaduje posun o 2 políčka, tedy musíme funkci pro posun volat 2×)

$\delta\left(q_{PZ}, \begin{matrix}U\\6'\end{matrix}\right) = \left(q_{61}, \begin{matrix}U\\a\end{matrix}, 1\right)$ $\quad$ $\delta\left(q_{61}, \begin{matrix}U\\B\end{matrix}\right) = \left(q_{62}, \begin{matrix}U\\a\end{matrix}, 1\right)$

$\delta\left(q_{62}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{63}, \begin{matrix}U\\A\end{matrix}, 1\right)$ $\quad$ $\delta\left(q_{63}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{Z}, \begin{matrix}U\\b\end{matrix}, -1\right)$

Čtvrté pravidlo gramatiky vyžaduje posun následujících symbolů doleva (β je kratší než α):

$\delta\left(q_{4}, \begin{matrix}U\\A\end{matrix}\right) = \left(q_{41}, \begin{matrix}U\\4\end{matrix}, 1\right)$

$\delta\left(q_{41}, \begin{matrix}U\\X\end{matrix}\right) = \left(q_{R}, \begin{matrix}U\\X\end{matrix}, 1\right)$ (kontrolujeme, zda je ve slově přítomna celá α)

Funkce pro posun následujícího řetězce o 1 políčko doleva:

$\delta\left(q_{R}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{RV}, \begin{matrix}U\\V\end{matrix}, 1\right)$ $\quad$ $\delta\left(q_{RV}, \begin{matrix}U\\M\end{matrix}\right) = \left(q_{RD}, \begin{matrix}U\\V\end{matrix}, 1\right)$

$\delta\left(q_{RD}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{R}, \begin{matrix}U\\V\end{matrix}, 1\right)$ $\quad$ $\delta\left(q_{RD}, \begin{matrix}U\\\$\end{matrix}\right) = \left(q_{RZ}, \begin{matrix}U\\\sqcup\end{matrix}, -1\right)$

$\delta\left(q_{RZ}, \begin{matrix}U\\V\end{matrix}\right) = \left(q_{RZ}, \begin{matrix}U\\V\end{matrix}, -1\right)$ $\quad$ $\delta\left(q_{RZ}, \begin{matrix}U\\4\end{matrix}\right) = \left(q_{Z}, \begin{matrix}U\\c\end{matrix}, -1\right)$

Pokud by β byla delší než 1 znak, museli bychom pro zpracování celého pravidla použít stavy $q_{42}, q_{43}, \ldots$

atd. pro další pravidla gramatiky.

4.3.2 Vytvoření gramatiky podle Turingova stroje

Věta 4.2 (Turingův stroj ⟶ gramatika)

Ke každému Turingovu stroji $\mathcal{M}$ lze sestrojit gramatiku G typu 0 takovou, že $L(\mathcal{M}) = L(G)$.

Postup

Může se sice zdát zvláštní nechat gramatiku simulovat činnost Turingova stroje, ale u gramatik typu 0 to jde.

Původní Turingův stroj funguje tak, že převezme svůj vstup, který je nad abecedou Σ, a ten určitým způsobem zpracuje. Pokud je na vstupu slovo patřící do přijímaného jazyka, stroj

toto slovo přijme, jinak je slovo odmítnuto nebo výpočet bude pokračovat nekonečnou smyčkou (neskončí).

Sestrojíme gramatiku, která bude pracovat podobně. Přidělený vstup nasimulujeme jednoduše tak, že v první sadě kroků jednoduše vygenerujeme jakékoliv slovo nad abecedou Σ (ve dvou kopiích), bez ohledu na to, zda patří nebo nepatří do daného jazyka. To bude náš „přidělený" vstup. V dalším postupu budeme toto slovo postupně zpracovávat pravidly odvozenými z přechodové funkce Turingova stroje, po simulaci každého předpisu zkontrolujeme, zda má být slovo (simulovaným strojem) přijato. Budeme postupovat takto:

- vygenerujeme dvě kopie téhož slova,
- první (horní) na konci generování použijeme jako výstup gramatiky, na druhé (spodní) budeme simulovat činnost TS,
- výstup gramatiky bude složen pouze z terminálních symbolů (a tedy získáme samotné výsledné slovo) jen tehdy, pokud simulace na druhé kopii skončí ve stavu q_{acc}.

Neterminály máme několika typů:

- neterminály ve tvaru sloupcového vektoru, jehož horní prvek $\in T$ nebo je ε,
- neterminály pro stav (zač. q_0) a začátek a konec řetězce \$,
- další neterminály bez přímého vztahu k TS (S, S').

Pro lepší představu se podíváme na fragment ukázky odvození:

$$S \Rightarrow^* q_0 \begin{bmatrix} \varepsilon \\ \$ \end{bmatrix} \begin{bmatrix} a \\ a \end{bmatrix} \begin{bmatrix} a \\ a \end{bmatrix} \begin{bmatrix} b \\ b \end{bmatrix} \begin{bmatrix} b \\ b \end{bmatrix} \begin{bmatrix} c \\ c \end{bmatrix} \begin{bmatrix} c \\ c \end{bmatrix} \$ \Rightarrow^* aabbcc$$

Postupně vytvoříme pravidla gramatiky.

1. Vygenerujeme dvě kopie téhož slova:

   $$S \to q_0 \begin{bmatrix} \varepsilon \\ \$ \end{bmatrix} S'$$
   $$S' \to \begin{bmatrix} a \\ a \end{bmatrix} S' \quad \text{pro každé } a \in \Sigma$$
   $$S' \to \$$$

2. Simulujeme činnost Turingova stroje:

   (a) pro $\delta(q_i, a) = (q_j, b, 1)$, $a, b \in \Gamma$, $q_i, q_j \in Q$ vytvoříme pravidla
   $$q_i \begin{bmatrix} x \\ a \end{bmatrix} \to \begin{bmatrix} x \\ b \end{bmatrix} q_j, \quad \text{pro } x \in \Sigma \cup \{\varepsilon\}$$

   (b) pro $\delta(q_i, a) = (q_j, b, 0)$, $a, b \in \Gamma$, $q_i, q_j \in Q$ vytvoříme pravidla
   $$q_i \begin{bmatrix} x \\ a \end{bmatrix} \to q_j \begin{bmatrix} x \\ b \end{bmatrix}, \quad \text{pro } x \in \Sigma \cup \{\varepsilon\}$$

   (c) pro $\delta(q_i, a) = (q_j, b, -1)$, $a, b \in \Gamma$, $q_i, q_j \in Q$ vytvoříme pravidla
   $$\begin{bmatrix} y \\ c \end{bmatrix} q_i \begin{bmatrix} x \\ a \end{bmatrix} \to q_j \begin{bmatrix} y \\ c \end{bmatrix} \begin{bmatrix} x \\ b \end{bmatrix}, \quad \text{pro } x, y \in \Sigma \cup \{\varepsilon\}, c \in \Gamma$$

   (d) pro $\delta(q_i, \sqcup) = (q_j, b, 1)$, příp. $\delta(q_i, \$) = (q_j, b, 1)$, $b \in \Gamma$, $q_i, q_j \in Q$ vytvoříme pravidla
   $$q_i\$ \to \begin{bmatrix} \varepsilon \\ b \end{bmatrix} q_j\$$$

(e) pro $\delta(q_i, \sqcup) = (q_j, b, 0)$, příp. $\delta(q_i, \$) = (q_j, b, 0)$, $b \in \Gamma,\ q_i, q_j \in Q$ vytvoříme pravidla

$$q_i\$ \ \rightarrow \ q_j \begin{bmatrix} \varepsilon \\ b \end{bmatrix} \$$$

(f) pro $\delta(q_i, \sqcup) = (q_j, b, -1)$, příp. $\delta(q_i, \$) = (q_j, b, -1)$, $b \in \Gamma,\ q_i, q_j \in Q$ vytvoříme pravidla

$$\begin{bmatrix} y \\ c \end{bmatrix} q_i\$ \ \rightarrow \ q_j \begin{bmatrix} y \\ c \end{bmatrix} \begin{bmatrix} \varepsilon \\ b \end{bmatrix} \$, \quad \text{pro } y \in \Sigma \cup \{\varepsilon\}, c \in \Gamma$$

3. Zakončíme derivaci (vytvoří se terminální slovo):

(a) Posuneme q_{acc} na začátek větné formy:

$$Mq_{\text{acc}} \ \rightarrow \ q_{\text{acc}}M \quad \text{pro všechny neterminály } M \in N$$

(b) Tvoříme terminály:

$$q_{\text{acc}} \begin{bmatrix} a \\ x \end{bmatrix} \ \rightarrow \ a\, q_{\text{acc}}, \quad \text{pro } x \in \Sigma \cup \{\varepsilon\}$$

(c) Mažeme vše, co nevytvoří terminál:

$$q_{\text{acc}} \begin{bmatrix} \varepsilon \\ x \end{bmatrix} \ \rightarrow \ q_{\text{acc}}, \quad \text{pro } x \in \Sigma \cup \{\varepsilon\}$$

(d) Konec derivace: $q_{\text{acc}}\$ \ \rightarrow \ \varepsilon$

Z posledních dvou uvedených vět plyne následující důsledek:

Důsledek 4.3

Třídy jazyků generovaných gramatikami typu 0 a přijímaných Turingovými stroji jsou ekvivalentní, odpovídají jazykům typu 0.

Kapitola 5

Jazyky typu 1

V této kapitole se budeme zabývat třídou jazyků generovaných gramatikami typu 1 Chomského hierarchie. Nejdřív se podíváme na tvar gramatik, které mohou generovat tyto jazyky a vztah mezi nimi, dále se budeme zabývat modely – stroji rozpoznávajícími tyto gramatiky, a také se budeme věnovat uzávěrovým vlastnostem jazyků typu 1 Chomského hierarchie.

5.1 Gramatiky typu 1

V předchozím semestru jsme si vysvětlili, že třebaže gramatiky typu 1 jsou nezkracující, existuje jiný typ gramatik generujících tutéž třídu jazyků – gramatiky kontextové.

Definice 5.1 (Gramatiky typu 1 v Chomského hierarchii)

Gramatiky typu 1 (nezkracující, monotónní, rekurzivní) jsou gramatiky, jejichž všechna pravidla zachovávají omezení pro pravidla gramatik typu 0 (min. neterminál vlevo) a zároveň jsou v tomto tvaru:

$$\alpha \to \beta, \quad |\alpha| \leq |\beta|, \quad \alpha, \beta \in (N \cup T)^* \tag{5.1}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla. Jazyky rozpoznávané gramatikami typu 1 jsou jazyky typu 1, třídu jazyků typu 1 značíme $\mathscr{L}(1)$.

Definice 5.2 (Kontextové gramatiky)

Kontextové gramatiky (Context-sensitive, CS) jsou gramatiky, jejichž všechna pravidla odpovídají předpisu

$$\alpha A \beta \to \alpha \gamma \beta, \quad A \in N, \ \alpha, \beta, \gamma \in (N \cup T)^*, \ \gamma \neq \varepsilon \tag{5.2}$$

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Jazyky rozpoznávané kontextovými gramatikami nazýváme kontextovými jazyky, třídu těchto jazyků značíme $\mathscr{L}(CS)$.

Je zřejmé, že každá kontextová gramatika je zároveň nezkracující, vyplývá to přímo z tvaru pravidel. Platí však i opačná relace?

Věta 5.1

Ke každé nezkracující gramatice G (typu 1) lze sestrojit kontextovou gramatiku G' takovou, že platí $L(G') = L(G)$.

Postup

Je dána nezkracující gramatika typu 1 $G = (N, T, P, S)$, chceme sestrojit kontextovou gramatiku $G' = (N', T, P', S)$.

Ta pravidla, která již splňují podmínku zachování kontextu, zařadíme do P' bez další konverze, zbývající pravidla musíme transformovat. Protože se jedná o nezkracující gramatiku, můžeme předpokládat, že pravá strana pravidel je stejně dlouhá nebo delší než jejich levá strana, což nám trochu zjednoduší práci. Každé pravidlo typu

$$A_1 A_2 \dots A_m \to B_1 B_2 \dots B_n,$$

které nemá kontextový tvar (přičemž $m \leq n$), nahradíme množinou pravidel:

$\boxed{A_1}\ A_2\ A_3 \dots A_m \to \boxed{C_1}\ A_2\ A_3 \dots A_m$
$C_1\ \boxed{A_2}\ A_3 \dots A_m \to C_1\ \boxed{C_2}\ A_3 \dots A_m$
$C_1\ C_2\ \boxed{A_3} \dots A_m \to C_1\ C_2\ \boxed{C_3} \dots A_m$
$\vdots$
$C_1\ C_2 \dots C_{m-1}\ \boxed{A_m} \to C_1\ C_2 \dots C_{m-1}\ \boxed{C_m\ B_{m+1} \dots B_n}$
$C_1\ C_2 \dots C_{m-1} \boxed{C_m} B_{m+1} \dots B_n \to C_1 \dots C_{m-1} \boxed{B_m} \dots B_n$
$C_1\ C_2 \dots \boxed{C_{m-1}} B_m\ B_{m+1} \dots B_n \to C_1 \dots \boxed{B_{m-1}} B_m \dots B_n$
$\vdots$
$\boxed{C_1}\ B_2 \dots B_n \to \boxed{B_1} \dots B_n$

Kde C_i jsou nově přidané neterminály, které se jinde nevyskytují. Původní derivaci

$A_1 A_2 \dots A_m \Rightarrow B_1 B_2 \dots B_n$

nahrazujeme derivací

$A_1A_2A_3 \dots A_{m-1}A_m \Rightarrow C_1A_2A_3 \dots A_{m-1}A_m \Rightarrow C_1C_2A_3 \dots A_{m-1}A_m \Rightarrow C_1C_2C_3 \dots A_{m-1}A_m \Rightarrow$
$\Rightarrow A_{m-1}A_m \Rightarrow C_1C_2C_3 \dots C_{m-1}A_m \Rightarrow^* C_1C_2C_3 \dots C_{m-1}C_mB_{m+1} \dots B_n \Rightarrow$
$\Rightarrow C_1C_2C_3 \dots C_{m-1}B_mB_{m+1} \dots B_n \Rightarrow C_1C_2C_3 \dots B_{m-1}B_mB_{m+1} \dots B_n \Rightarrow^*$
$\Rightarrow^* C_1B_2B_3 \dots B_{m-1}B_mB_{m+1} \dots B_n \Rightarrow B_1B_2B_3 \dots B_{m-1}B_mB_{m+1} \dots B_n$

Poznámka:

Na následujícím příkladu si ukážeme nejen samotný převod nezkracující gramatiky na kontextovou, ale také postup při vytvoření (nezkracující) gramatiky. U nezkracující gramatiky můžeme využít *princip jezdce* – neterminálu, který je v generovaném slově pomocí pravidel postupně „posouván“ v potřebném směru.

Příklad 5.1

Vytvoříme nezkracující gramatiku pro jazyk $L = \{ww\ ;\ w \in \{a,b\}^*\}$. Tento jazyk není bezkontextový (pozor, nepleťte si ho s jazykem, kde je ww^R), druhá polovina slova je přesnou kopií první poloviny.

Postupně sestavíme nezkracující pravidla gramatiky G.

$S \to XZ_aZ_a \mid XZ_bZ_b \mid aa \mid bb \mid \varepsilon$

pokud w končí na a, začínáme prvním pravidlem – $\ldots Z_a \ldots Z_a$

pokud w končí na b, začínáme druhým pravidlem – $\ldots Z_b \ldots Z_b$

$XZ_a \to aZ_aX_a \mid bZ_aX_b \mid aaX_a \mid baX_b$

v první polovině jsme vygenerovali a, pošleme info do druhé poloviny – $\ldots aZ_aX_a \ldots Z_a$

v první polovině jsme vygenerovali b, pošleme info do druhé poloviny – $\ldots bZ_aX_b \ldots Z_a$

$X_aa \to aX_a$ symbol X_a nebo X_b posíláme doprava

$X_ab \to bX_a$

$X_ba \to aX_b$

$X_bb \to bX_b$

$X_aZ_a \to YaZ_a \mid aa$ info doputovalo k zarážce na konci slova – $\ldots aZ_a \ldots YaZ_a$

$X_bZ_a \to YbZ_a \mid ba$ $\ldots bZ_a \ldots YbZ_a$

$aY \to Ya$ symbol Y posíláme doleva – zpráva „pokračuj v první polovině“

$bY \to Yb$

$Z_aY \to XZ_a$ překročili jsme zarážku na konci první poloviny slova

Dále totéž pro Z_b místo Z_a:

$XZ_b \to aZ_bX_a \mid bZ_bX_b \mid abX_a \mid bbX_b$

$X_aZ_b \to YaZ_b \mid ab$

$X_bZ_b \to YbZ_b \mid bb$

$Z_bY \to XZ_b$

Ukázka odvození:

$S \Rightarrow XZ_aZ_a \Rightarrow aZ_aX_aZ_a \Rightarrow aZ_aYaZ_a \Rightarrow aXZ_aaZ_a \Rightarrow abZ_aX_baZ_a \Rightarrow abZ_aaX_bZ_a \Rightarrow$
$\Rightarrow abZ_aaYbZ_a \Rightarrow abZ_aYabZ_a \Rightarrow abXZ_aabZ_a \Rightarrow abbX_babZ_a \Rightarrow abbaX_bbZ_a \Rightarrow abbabX_bZ_a \Rightarrow$
$\Rightarrow abbabba$

5.2 Kurodova normální forma pro gramatiky typu 1

Kurodova normální forma (KNF, Kuroda Normal Form) je obdobou Chomského normální formy (ovšem pro gramatiky typu 1), a postup převodu gramatiky do KNF je taky velmi podobný. Jediný rozdíl je v tom, že v KNF potřebujeme reprezentovat i taková pravidla, kde je na levé straně více než jeden symbol.

V postupu transformace nezkracující gramatiky typu 1 tedy vyjdeme z postupu pro CNF a přidáme variantu pro další typ pravidel.

Definice 5.3 (Kurodova normální forma pro gramatiky typu 1)

Gramatika typu 1 $G = (N, T, P, S)$ je v KNF, jestliže všechna její pravidla jsou v některém z těchto tvarů:

$$\begin{aligned} A &\to BC \\ AB &\to CD \\ A &\to a \end{aligned}$$

kde $A, B, C, D \in N$, $a \in T$.

Dále může existovat pravidlo $S \to \varepsilon$, pokud je S startovacím symbolem gramatiky a zároveň se nenachází pravé straně žádného pravidla.

Věta 5.2

Ke každé nezkracující (i kontextové) gramatice G lze sestrojit gramatiku G' v Kurodově normální formě takovou, že $L(G') = L(G)$.

Postup

Je dána nezkracující gramatika typu 1 $G = (N, T, P, S)$. Vytvoříme k ní ekvivalentní gramatiku v KNF $G' = (N', T, P', S)$. Budeme postupovat takto:

- *pravidla v bezkontextovém tvaru:* použijeme algoritmus pro převod na Chomského NF,
- *pravidla, která nejsou bezkontextového typu:* upravíme pravidla podle následujícího algoritmu.

Vstup: nezkracující gramatika bez jednoduchých pravidel typu $X \to Y, \quad X, Y \in N$ (pro případnou úpravu použijeme stejný algoritmus jako u bezkontextových gramatik).

Celý postup:

1. Pro všechna pravidla neodpovídající předpisu v KNF:
   - všechny terminály (na levé i pravé straně pravidla) $a \in T$ nahradíme „pomocnými“ neterminály N_a,
   - pro všechny neterminály vytvořené v předchozím bodu přidáme pravidlo $N_a \to a$.
2. Podle CNF: pravidla $A \to B_1 B_2 \dots B_n, \ n > 2$ nahradíme pravidly

$$\begin{aligned} A &\to B_1 X_1 \\ X_1 &\to B_2 X_2 \\ &\vdots \\ X_{n-2} &\to B_{n-1} B_n \end{aligned}$$

3. Pravidla $A_1 A_2 \dots A_m \to B_1 B_2 \dots B_m, \ m > 2$ nahradíme pravidly

$$\begin{aligned} A_1 A_2 &\to B_1 X_1 \\ X_1 A_3 &\to B_2 X_2 \\ &\vdots \\ X_{m-2} A_m &\to B_{m-1} B_m \end{aligned}$$

4. Nezkracující pravidla $A_1A_2 \dots A_m \rightarrow B_1B_2 \dots B_n,\ 2 < m \leq n$ nahradíme pravidly

$$\begin{array}{rcl@{\qquad}rcl} A_1A_2 & \rightarrow & B_1X_1 & X_{m-1} & \rightarrow & B_mX_m \\ X_1A_3 & \rightarrow & B_2X_2 & X_m & \rightarrow & B_{m+1}X_{m+1} \\ \vdots & & & \vdots & & \\ X_{m-2}A_m & \rightarrow & B_{m-1}X_{m-1} & X_{n-2} & \rightarrow & B_{n-1}B_n \end{array}$$

5.3 Lineárně ohraničený automat

Tak jako jazykům typu 0 můžeme přiřadit Turingův stroj a například konečný automat rozpoznává právě regulární jazyky, k jazykům typu 1 můžeme přiřadit lineárně ohraničený automat (LOA).

Definici LOA přejímáme z definice Turingova stroje, jen přidáváme zákaz přepisu hraničních symbolů.

Definice 5.4 (Lineárně ohraničený automat)

Lineárně ohraničený automat (LOA, LBA – Linear Bounded Automaton) je jednopáskový (nedeterministický) Turingův stroj $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, ve kterém čtecí a zápisová hlava nesmí během výpočtu přepsat hraniční symboly \$ pásky (tj. slovo nesmí být během výpočtu prodlužováno).

Definice 5.5 (Konfigurace lineárně ohraničeného automatu)

Konfigurace lineárně ohraničeného automatu je (α, q, β), kde q je stav, na pásce je řetězec $\alpha\beta$, čtecí a zápisová hlava ukazuje na první symbol řetězce β.

Přechod mezi konfiguracemi je definován stejně jako u Turingova stroje, jen je třeba zohlednit nemožnost přepisu hraničních symbolů.

Věta 5.3

Pro konkrétní lineárně ohraničený automat $\mathcal{M}$ a daný vstup w_0 existuje konečný počet různých konfigurací.

Důkaz: Protože máme konečný počet stavů, konečný počet páskových symbolů a omezenou část vstupní pásky, platí: jestliže označíme

d ... délka používané části pásky,
s ... počet stavů v Q,
g ... počet prvků páskové abecedy Γ,

pak počet všech možných různých konfigurací je $d \cdot s \cdot g^d$ (hlava může být na d různých pozicích, nabývá hodnot s různých stavů, počet všech řetězců nad abecedou Γ o délce d je g^d). □

Poznámka:

Rekurzívní jazyky jsme definovali na začátku kapitoly o jazycích typu 0, na straně 106. Na rozdíl od rekurzívně spočetných jazyků je lze zpracovat Turingovým strojem tak, že pro jakýkoliv vstup výpočet skončí přechodem do některého koncového stavu, bez přechodu do nekonečné smyčky, a výpočet probíhá na principu rekurze (rekurzívně uplatňujeme δ funkci).

Důsledek 5.4

LOA lze vždy navrhnout tak, aby výpočet skončil nad jakýmkoliv vstupem. Proto jazyky, které jsou přijímány nějakým LOA, jsou právě rekurzívní jazyky.

Důkaz: Stačí při každém kroku výpočtu LOA zkontrolovat, jestli se nenachází v konfiguraci, ve které už byl dříve. Pokud ano, výpočet v původním automatu se dostal do smyčky a my můžeme skončit v chybovém stavu q_{reject} (vstup nepřijmeme).

Pokud pro každý vstup LOA najdeme počet políček pásky, se kterými během výpočtu bude pracovat čtecí a zápisová hlava (tj. délka části pásky použité při výpočtu, prostorová složitost výpočtu automatu nad daným vstupem), zjistíme, že tato hodnota je lineárně závislá na délce vstupu, tedy existuje přirozené číslo k takové, že délka použité části pásky je menší než $k \cdot |w|$.

Odtud je odvozen také název tohoto automatu – automat s lineárně ohraničeným pracovním prostorem. □

Poznámka:

V některých zdrojích je LOA definován přímo jako Turingův stroj s lineárně ohraničeným pracovním prostorem, a je tedy dovoleno používat pro výpočet každého slova w nejvýše $k \cdot |w|$ políček pásky (tedy nejen pro $k = 1$).

Věta 5.5

Jazyky typu 1 jsou právě jazyky přijímané lineárně ohraničenými automaty.

Důkaz ($\Rightarrow$): Máme nezkracující gramatiku $G = (N, T, P, S)$, která generuje jazyk L typu 1. Derivace v této gramatice je ve tvaru
$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots w_n$, kde $|w_i| \leq |w_j|$ pro $i < j$.

Sestrojíme LOA $\mathcal{M}$ takto:

1. Na vstupu je slovo, o kterém chceme zjistit, zda je generováno gramatikou G.
2. Na tento vstup budeme uplatňovat pravidla gramatiky takto:
   (a) nedeterministicky zvolíme pravidlo gramatiky $(\alpha \to \beta)$,

(b) najdeme v řetězci na pásce některý výskyt pravé strany tohoto pravidla (β) – pokud je těchto výskytů více, nedeterministicky mezi nimi jeden zvolíme,

(c) výskyt β přepíšeme řetězcem α, pokud je α kratší, posuneme to, co následuje za β, doleva, aby zbytek pracovního slova navazoval na α.

3. Výpočet ukončíme:
   - ve stavu akceptování (q_{accept}), pokud na pásce bude pouze startovací symbol gramatiky a nic jiného,
   - ve stavu odmítnutí slova (q_{reject}, q_{error}), pokud je na pásce něco jiného a již nelze použít žádné pravidlo gramatiky.

Je zřejmé, že tento LOA vytváří derivaci slova podle gramatiky G zprava doleva (a tedy konstruuje derivační strom pro toto slovo zdola nahoru ke kořeni stromu ohodnocenému startovacím symbolem) a ve stavu akceptování skončí právě na ta slova, která generuje gramatika G. □

Důkaz ($\Leftarrow$)**:** Je dán LOA $\mathcal{M}$. Vytvoříme nezkracující gramatiku G podobně jako v obdobném důkazu pro Turingovy stroje a gramatiky typu 0, jen musíme pravidla upravit tak, aby byla nezkracující.

1. V gramatice G nejdřív vygenerujeme řetězec neterminálů, který představuje dvojici slov na vstupu simulovaného LOA. Oproti gramatice typu 0 musíme symboly pro začátek a konec pracovní části pásky a také symbol pro stav automatu umístit dovnitř neterminálů pro symboly slova, abychom nebyli nuceni na konci výpočtu použít epsilonová pravidla.

   $$S \to \left[\begin{array}{c} a \\ \$, q_0, a \end{array}\right] S' \;\middle|\; \left[\begin{array}{c} \varepsilon \\ \$, q_0, \$ \end{array}\right]$$

   $$S' \to \left[\begin{array}{c} a \\ a \end{array}\right] S' \;\middle|\; \left[\begin{array}{c} a \\ a, \$ \end{array}\right] \text{ pro každé } a \in \Sigma$$

   Dostaneme řetězec neterminálů

   $$\left[\begin{array}{c} a_1 \\ \$, q_0, a_1 \end{array}\right] \left[\begin{array}{c} a_2 \\ a_2 \end{array}\right] \left[\begin{array}{c} a_3 \\ a_3 \end{array}\right] \cdots \left[\begin{array}{c} a_k \\ a_k, \$ \end{array}\right]$$

2. Simulujeme průběh výpočtu LOA:

   Např. pro každou část δ funkce typu $\delta(q_i, a) \ni (q_j, b, 1)$

   $$\left[\begin{array}{c} x \\ q_i, a \end{array}\right] \left[\begin{array}{c} y \\ c \end{array}\right] \to \left[\begin{array}{c} x \\ b \end{array}\right] \left[\begin{array}{c} y \\ q_j, c \end{array}\right] \text{ pro každé } c \in \Gamma - \{\$\}$$

   Podobně pro ostatní směry pohybu čtecí a zápisové hlavy, navíc musíme ošetřit práci s neterminály, které obsahují hraniční znaky \$.

3. Ukončení výpočtu:

   $$\left[\begin{array}{c} x \\ a \end{array}\right] \left[\begin{array}{c} y \\ q_{\text{acc}}, b \end{array}\right] \to \left[\begin{array}{c} x \\ q_{\text{acc}}, a \end{array}\right] \left[\begin{array}{c} y \\ b \end{array}\right]$$

   $$\left[\begin{array}{c} x \\ \$, a \end{array}\right] \left[\begin{array}{c} y \\ q_{\text{acc}}, b \end{array}\right] \to x \left[\begin{array}{c} y \\ K, b \end{array}\right]$$

$$\begin{bmatrix} x \\ K,a \end{bmatrix} \begin{bmatrix} y \\ b \end{bmatrix} \to x \begin{bmatrix} y \\ K,b \end{bmatrix}$$

$$\begin{bmatrix} x \\ K,a \end{bmatrix} \begin{bmatrix} y \\ b,\$ \end{bmatrix} \to xy$$

Posuneme q_{acc} dopředu na začátek řetězce, pak přejdeme do ukončujícího stavu K a postupně všechny neterminály přepíšeme na terminály. □

5.4 Uzávěrové vlastnosti jazyků typu 1

Zatímco u jazyků typu 0 nemělo smysl probírat jakékoliv uzávěrové vlastnosti (třída jazyků typu 0 je totiž uzavřena na absolutně vše), u jazyků typu 1 nemusí být tento fakt až tak zřejmý.

Věta 5.6

Třída jazyků typu 1 je uzavřena vzhledem k operacím sjednocení, zřetězení, iterace, pozitivní iterace, homomorfismu, substituce.

Důkaz: Pro jazyky typu 1 je důkaz stejný jako u bezkontextových jazyků, provádí se drobnou úpravou gramatiky. Například pro sjednocení přidáme navíc pravidlo $S \to S_1 \mid S_2$, kde S je nově přidaný symbol, S_1 je startovací symbol první gramatiky a S_2 je startovací symbol druhé gramatiky. □

Věta 5.7

Třída jazyků typu 1 je uzavřena vzhledem k operaci průniku.

Důkaz: Máme dva LOA $\mathcal{M}_1$, $\mathcal{M}_2$. Sestrojíme LOA $\mathcal{M}$, který bude postupně simulovat výpočet obou těchto automatů, a pokud oba skončí s akceptováním vstupního slova, toto slovo také akceptuje.

1. Na vstupní pásce pro akceptování slova w bude řetězec $\$w\#w\$$.
2. Nejdřív $\mathcal{M}$ simuluje na první kopii slova w výpočet stroje $\mathcal{M}_1$ s tím, že hraniční symboly jsou \$ a #.
3. Pokud simulovaný výpočet skončí ve stavu akceptování slova, posune čtecí a zápisovou hlavu na první symbol za znakem # (tj. na první symbol druhé kopie slova w) a simuluje výpočet stroje $\mathcal{M}_2$.
4. Pokud tento výpočet skončí ve stavu akceptování, $\mathcal{M}$ akceptuje slovo w.

V důkazu je využito i faktu, že pro jakýkoliv jazyk typu 1 lze sestrojit LOA – TS, který nebude překračovat hranice dané vstupním slovem. □

Věta 5.8

Třída jazyků typu 1 je uzavřena vzhledem k operaci doplňku.

Důkaz lze provést pomocí De Morganových pravidel nebo konstrukčně. Důkaz pomocí De Morganových pravidel můžeme nechat na čtenáři, konstrukční důkaz je jednoduchý:

Důkaz: LOA lze vždy sestrojit tak, aby jeho výpočet byl konečný (tedy aby nemoho dojít k zacyklení). Pokud chceme vytvořit LOA takový, který by přijímal doplněk původního jazyka, pouze zaměníme akceptující a chybový stav. □

Literatura

[1] Černá, Ivana, a kol. Automaty a formální jazyky I. Učební text FI MU. Fakulta informatiky, Masarykova univerzita, Brno: 2012. Dostupné z: http://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf

[2] Chytil, M.: *Automaty a gramatiky*. Praha: SNTL, 1984.

[3] Meduna, Alexander. *Automata and languages: theory and applications*. London: Springer, 2000, xv, 916 s. ISBN 18-523-3074-0
Dostupné na Google Books: http://books.google.cz, jako klíčová slova zadejte celý název knihy [cit. 2008-7-1]

[4] Melichar, Bořivoj. *Jazyky a překlady*. Vyd. 1. Praha: ČVUT, 1996. ISBN 80-010-1511-4

[5] Paleta, Petr. *Co programátory ve škole neučí: aneb Softwarové inženýrství v reálné praxi*. Vyd. 1. Brno: Computer Press, 2003, 337 s. ISBN 80-251-0073-1

[6] Robic, Florent. A real Turing machine. *The Alan Turing Year* [online]. 2012 [cit. 2015-01-27]. Dostupné z: http://www.turing2012.fr/?p=530&lang=en