

SLEZSKÁ UNIVERZITA V OPAVĚ  
FILOZOFICKO-PŘÍRODOVĚDECKÁ FAKULTA  
ÚSTAV INFORMATIKY



# TEORIE JAZYKŮ A AUTOMATŮ

## STUDIJNÍ TEXTY

*Poslední změny: 24. června 2007*

Mgr. Šárka Vavrečková

sarka.vavreckova@fpf.slu.cz

<http://fpf.slu.cz/~vav10ui>

Opava 2007

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Teoretická informatika</b>	<b>2</b>
1.1 Vznik a vývoj teoretické informatiky . . . . .	2
1.1.1 Matematika . . . . .	2
1.1.2 Jazykověda . . . . .	5
1.1.3 Biologie . . . . .	6
1.2 Možnosti použití teoretické informatiky . . . . .	8
<b>2 Konečné automaty</b>	<b>9</b>
2.1 Konečný automat . . . . .	9
2.2 Nedeterminismus . . . . .	12
2.3 Totální automat . . . . .	15
2.4 Odstranění nepotřebných stavů automatu . . . . .	16
2.4.1 Nedosažitelné stavy . . . . .	16
2.4.2 Nadbytečné stavy . . . . .	17
<b>3 Regulární jazyky</b>	<b>19</b>
3.1 Konečné jazyky . . . . .	19
3.2 Pumping Lemma pro regulární jazyky a nekonečné jazyky . . . . .	20
3.3 Uzávěrové vlastnosti třídy regulárních jazyků . . . . .	22
3.3.1 Sjednocení . . . . .	23
3.3.2 Zřetězení . . . . .	24
3.3.3 Iterace . . . . .	25

3.3.4	Pozitivní iterace . . . . .	26
3.3.5	Zrcadlový obraz . . . . .	26
3.3.6	Průnik . . . . .	27
3.3.7	Homomorfismus . . . . .	29
<b>4</b>	<b>Regulární výrazy</b>	<b>30</b>
4.1	Možnosti využití regulárních výrazů . . . . .	30
4.2	Definice . . . . .	31
4.3	Vztah ke konečným automatům . . . . .	32
4.4	Využití vztahu regulárních výrazů k reg. jazykům . . . . .	36
4.4.1	Důkazy uzávěrových vlastností regulárních jazyků . . . . .	36
4.4.2	Normovaný automat . . . . .	37
4.4.3	Minimalizace konečného automatu . . . . .	38
<b>5</b>	<b>Formální gramatiky</b>	<b>44</b>
5.1	Generování slov jazyka . . . . .	44
5.2	Regulární gramatiky . . . . .	46
5.3	Chomského hierarchie gramatik . . . . .	50
5.3.1	Gramatiky v Chomského hierarchii . . . . .	50
5.3.2	Související typy gramatik . . . . .	51
5.4	Operace nad slovy a jazyky . . . . .	52
<b>6</b>	<b>Bezkontextové jazyky</b>	<b>54</b>
6.1	Bezkontextové gramatiky . . . . .	54
6.2	Vlastnosti bezkontextových gramatik . . . . .	56
6.2.1	Bezkontextová gramatika nezkracující . . . . .	56
6.2.2	Redukovaná gramatika . . . . .	58
6.2.3	Gramatika bez jednoduchých pravidel . . . . .	61
6.2.4	Další typy bezkontextových gramatik . . . . .	62
6.3	Normální formy pro bezkontextové gramatiky . . . . .	63
6.3.1	Chomského normální forma . . . . .	63
6.3.2	Greibachova normální forma . . . . .	65

# Úvod

Tento dokument je zatím v první fázi vývoje, tedy zatím jsou doplňovány kapitoly. Text není ve stavu vhodném pro plné samostudium, vyžaduje komentář vyučujícího a odpovídá přibližně obsahu prezentací z přednášek. Je určen pro studenty prezenčního a kombinovaného studia Slezské univerzity navštěvující kurz Teorie jazyků a automatů.

# Kapitola 1

## Teoretická informatika

*Tato kapitola je především motivační – dovíme se zde, jaký má teoretická informatika význam, jak vznikla a také jaké jsou možnosti jejího praktického použití.*

### 1.1 Vznik a vývoj teoretické informatiky

*Vznik teoretické informatiky – tři kořeny:*

1. matematika,
2. jazykověda,
3. biologie.

#### 1.1.1 Matematika

Počátek 20. století: rozvoj logiky.



Roku 1936 – zjištění: „Nelze cokoliv jednoznačně určit a vypočítat.“

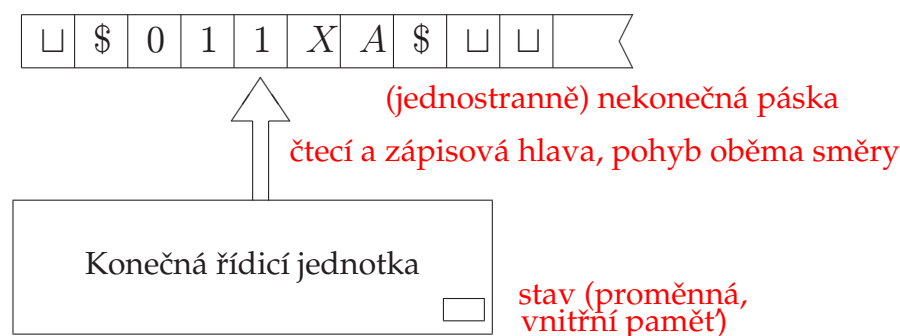
Otázka: „Co lze vypočítat?“

⇒ Turingův stroj a další výpočetní modely

*Alan Turing (1912–1954)*

Vytvořil Turingův stroj jako matematický model

„Na tomto stroji lze vypočítat právě to, co je vypočitatelné.“



Obrázek 1.1: Turingův stroj

Vlastnosti Turingova stroje:

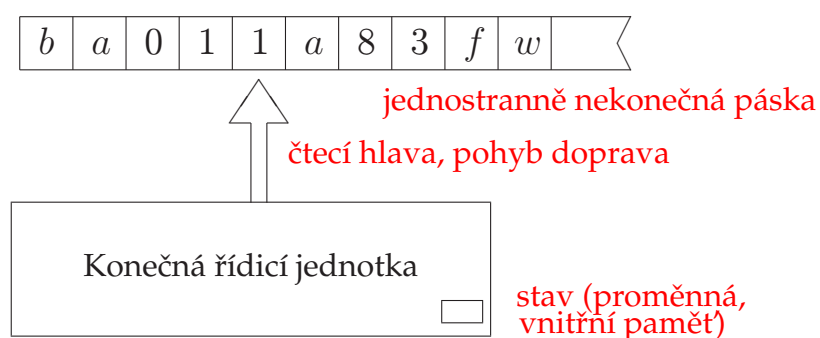
- řídicí jednotka se vždy nachází v některém z předem určených stavů,
- čtecí a zápisová hlava je vždy na některém políčku pásky,
- v každém kroku se stroj řídí podle stavu jednotky a podle obsahu políčka, na které ukazuje hlava,
- podle těchto dvou údajů se rozhodne o akci, která spočívá
  - ve změně stavu jednotky (například ze stavu „načítání“ do stavu „načteno“ nebo ze stavu „pracuji“ do stavu „vypínám“),
  - přepsání obsahu políčka na pásce něčím jiným (nebo může políčko zůstat beze změny), a
  - posun na pásce vpravo, vlevo, a nebo může hlava zůstat na stejném políčku.

Výsledkem činnosti stroje obvykle bývá obsah pásky, důležitou informací může být stav, ve kterém stroj skončil výpočet (je množina koncových stavů).

**Další modely:** Turingův stroj byl moc složitý pro některé jednodušší výpočty, proto vznikly jednodušší modely – konečný automat a zásobníkový automat.

Vlastnosti konečného automatu:

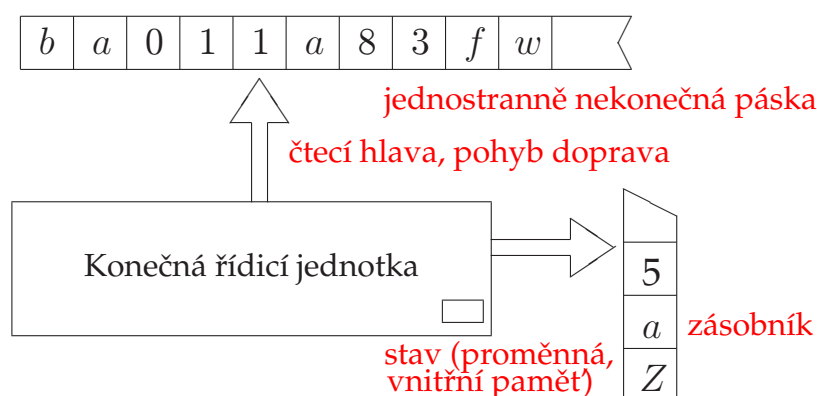
- řídicí jednotka se vždy nachází v některém z předem určených stavů,
- čtecí hlava je vždy na některém políčku pásky,



Obrázek 1.2: Konečný automat

- v každém kroku se stroj řídí podle stavu jednotky a podle obsahu políčka, na které ukazuje hlava,
- podle těchto dvou údajů se rozhodne o akci, která spočívá
  - ve změně stavu jednotky (například ze stavu „načítání“ do stavu „načteno“ nebo ze stavu „pracuji“ do stavu „vypínám“),
  - posunem na pásce o políčko vpravo.

Narozdíl od Turingova stroje se čtecí hlava posouvá v každém kroku o políčko doprava a nemá možnost zapisovat. Výsledkem činnosti automatu je pouze informace o tom, ve kterém stavu stroj skončil, a zda přečetl celý obsah pásky (když nepřičetl a „zasekl se“ – pro daný obsah pole na pásce a momentální stav třeba není definována žádná akce).



Obrázek 1.3: Zásobníkový automat

Vlastnosti zásobníkového automatu:

- řídicí jednotka se vždy nachází v některém z předem určených stavů,
- čtecí hlava je vždy na některém políčku pásky,
- automat má k dispozici zásobník, kde přidává shora a také shora vybírá,
- v každém kroku automat vyjme jeden prvek ze zásobníku, a řídí se podle tohoto vyjmutého symbolu, stavu jednotky a také se může (nemusí) řídit podle obsahu políčka, na které ukazuje hlava,
- podle těchto tří (nebo dvou) údajů se rozhodne o akci, která spočívá
  - ve změně stavu jednotky (například ze stavu „načítání“ do stavu „načteno“ nebo ze stavu „pracuji“ do stavu „vypínám“),
  - posunem na pásce o políčko vpravo, pokud v tomto kroku četl symbol ze vstupní pásky (tj. když čte ze vstupu, zároveň se posune dál),
  - může uložit do zásobníku jakýkoliv počet prvků (i žádný), a to postupně po jednom, ten, který vložil jako poslední, bude hned v dalším kroku vyjmut.

Narozdíl od konečného automatu čtecí hlava nemusí pracovat v každém kroku (buď čte a zároveň se posune, nebo nepracuje vůbec), nemá možnost zapisovat a pohybuje se jen směrem doprava. Výsledkem činnosti je informace o stavu, ve kterém stroj skončil, zda přečetl celý obsah pásky, a případně může být důležité, zda do konce výpočtu stačil vyprázdnit celý zásobník.

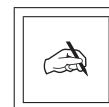
### 1.1.2 Jazykověda

Formální gramatika řeší, jak se slova utvářejí (u matematických kořenů byl řešen problém rozpoznávání již utvořených slov či sekvencí signálů).

*Noam Chomskij* Zkoumal syntaxi jazyka a schopnost lidí mluvit.

*„Všechny jazyky jsou utvořeny přibližně stejným způsobem, mají stejný základ.“*

**Definice 1.1 (Formální gramatika)** *Formální gramatika je soubor obecných pravidel, generujeme větu na základě její struktury (syntaxe).*





**Příklad 1.1**

$$S \rightarrow [begin]A[end]$$

$$A \rightarrow P;$$

$$A \rightarrow P;A$$

$$P \rightarrow [vypis]T$$

$$T \rightarrow [pismenko]$$

$$T \rightarrow [pismenko]T$$

$$S \Rightarrow [begin]A[end] \Rightarrow [begin]P;[end] \Rightarrow [begin]P;P;[end] \Rightarrow$$

$$\Rightarrow [begin][vypis]T;P;[end] \Rightarrow$$

$$\Rightarrow [begin][vypis][pismenko]T;P;[end] \Rightarrow \dots$$

$$\Rightarrow [begin][vypis][pismenko][pismenko][pismenko];$$

$$[vypis][pismenko][pismenko][pismenko][pismenko];[end]$$

**Základní princip:** Věta se skládá ze slov, při skládání částí potřebujeme také pomocná slova („obecné termíny“), za která můžeme dosadit posloupnost skládající se ze slov a pomocných slov – rekurze.

Pomocná slova = *neterminální symboly*, skutečná slova = *terminální symboly*

**Typy formálních gramatik:**

- Gramatika odpovídající Turingovu stroji: pravidla  $\alpha \rightarrow \beta$
- Bezkontextová gramatika: pravidla  $A \rightarrow \beta$
- atd.

**1.1.3 Biologie**

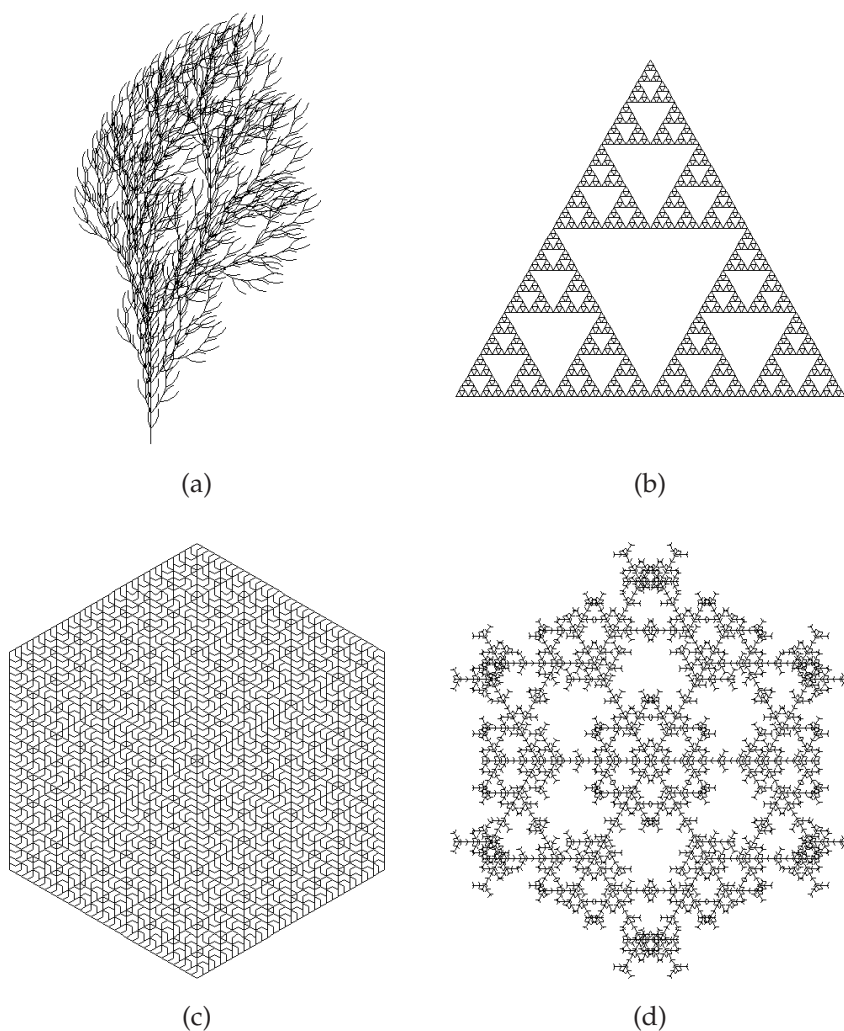
**Aristid Lindenmayer.** Lindenmayerovy systémy (L-Systémy) – zjistil, že když upraví formální gramatiku a pozmění její chování, dokáže například simulovat růst a vývoj rostliny nebo dělení buněk.

Pravidlo:  $b \rightarrow bb$

Výpočet:  $b \Rightarrow bb \Rightarrow b^4 \Rightarrow b^8 \Rightarrow \dots$

Jeho žáci pak přišli na možnost grafické interpretace L-Systemů  $\Rightarrow$  různé typy fraktálů. U fraktálů generovaných L-Systemy jde o to, jak poměrně složitý ná-kres vygenerovat co nejjednodušším řetězcem „obyčejných“ symbolů, které mají význam určité instrukce (vykresli čáru, popojdi bez vykreslení, otoč se doprava, doleva, ulož písmeno do zásobníku, vyjmi písmeno ze zásobníku, apod.).

Uplatňuje se rekurze, tedy totéž pravidlo (nebo tatáž pravidla) se používá pořád dokola tak dlouho, dokud nevznikne dostatečně dlouhá a „složitá“ posloupnost instrukcí.



Obrázek 1.4: Několik fraktálů vygenerovaných pomocí L-Systemů

Například obrázek 1.4c na straně 7 vznikl z řetězce  $F+F+F+F+F+F$  tak, že jsme rekurzivně všechny symboly  $F$  (i ty nově přidávané) zpracovali pravidlem  $F \rightarrow F[+F+F]F$ .

## 1.2 Možnosti použití teoretické informatiky

- stavové programování, překladače
- modelování matematických výpočtů
- analýza přirozeného jazyka (kontrola pravopisu, překlady, atd.)
- L-Systémy: biologie, fraktály (malířství, filmy, apod.), fyzika (teorie chaosu, modelování turbulence, studium tornád, atd.)
- porovnávání složitosti různých algoritmů dělajících totéž
- DNA-výpočty
- fyzika – kvantové počítače
- ...

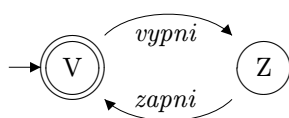
## Konečné automaty

Nejdříve se budeme zabývat nejjednodušším typem modelů, které byly zmíněny v kapitole 1.1.1 o matematických kořenech teoretické informatiky, konečnými automaty, a také se podíváme na základy práce s jednoduchými gramatikami, se kterými jsme se už trochu seznámili v kapitole 1.1.2.

### 2.1 Konečný automat

Konečný automat je vždy v některém (vnitřním) stavu, který si můžeme představit jako proměnnou nabývající předem stanovených hodnot. Pracuje jednoduše tak, že na základě informace o svém momentálním stavu a dále podle signálu, který dostal (symbolu na vstupu) se přesune do některého jiného stavu a zároveň se posune na vstupu, tedy očekává další signál (je připraven načíst další symbol z pásky).

Na obrázcích 2.1, 2.2 a 2.3 vidíme několik jednoduchých diagramů (orientovaných grafů) konečných automatů. Diagram přehledně (u jednodušších automatů) zobrazuje *přechody mezi stavy* (šipky) a *signály* (symboly), které jsou při tomto přechodu načítány a zpracovávány (ohodnocení šipek).

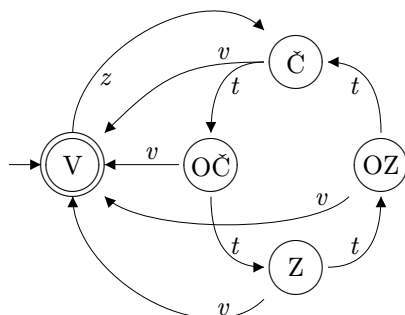


Popis:

V ..... vypnuto

Z ..... zapnuto

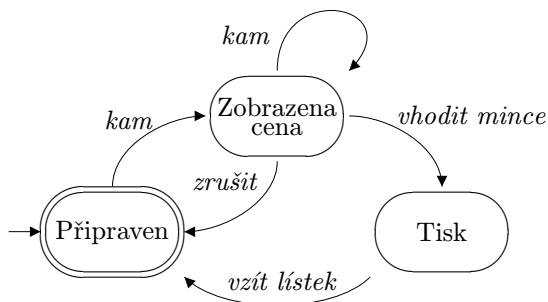
Obrázek 2.1: Elektrický spotřebič jako konečný automat



Popis:

- V ..... vypnuto
- Č, Z ..... červená, zelená
- OČ ..... oranžová od červené
- OZ ..... oranžová od zelené

Obrázek 2.2: Semafor jako konečný automat



Obrázek 2.3: Automat na jízdenky

**Definice 2.1 (Konečný automat)** Konečný automat je uspořádaná pětice

$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ , kde je

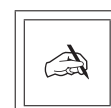
$Q$  ... neprázdná konečná množina stavů

$\Sigma$  ... neprázdná konečná abeceda (množina signálů)

$\delta$  ... přechodová funkce, definovaná níže

$q_0$  ... počáteční stav,  $q_0 \in Q$

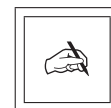
$F$  ... množina koncových stavů,  $F \subseteq Q, F \neq \emptyset$



**Definice 2.2 (Přechodová funkce)** Přechodová funkce  $\delta$  konečného automatu (dále KA)

$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  je zobrazení  $\delta : Q \times \Sigma \rightarrow Q$

$\delta(\text{stav}, \text{signál}) = \text{stav}$



Například:  $\delta(\text{vypnuto}, \text{zapni}) = \text{zapnuto}$

Potřebujeme vědět:

- ve kterém jsme stavu,
- co ještě zbývá přečíst (zpracovat).

Tyto informace jsou uloženy v konfiguraci automatu, která se postupně mění.

**Definice 2.3 (Konfigurace konečného automatu)** Označme  $\Sigma^*$  množinu všech slov, která lze utvořit ze symbolů abecedy  $\Sigma$ . Konfigurace KA je uspořádaná dvojice  $(q, w)$ , kde  $q \in Q$ ,  $w \in \Sigma^*$  (nepřečtená část vstupní pásky).

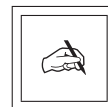


- Počáteční konfigurace:  $(q_0, w_0)$   
(jsme v počátečním stavu a  $w_0$  je celé zpracovávané slovo)
- Koncová konfigurace:  $(q_f, \varepsilon)$ ,  $q_f \in F$   
( $\varepsilon$  je prázdné slovo, tj. řetězec s délkou 0, neboli celé slovo již bylo zpracováno)

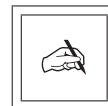
**Definice 2.4 (Přechod mezi konfiguracemi)** Relace přechodu mezi konfiguracemi je definována jako  $\vdash: (Q, \Sigma^*) \times (Q, \Sigma^*)$ , kde platí:

$$(q_i, aw) \vdash (q_j, w) \Leftrightarrow \delta(q_i, a) = q_j$$

(relace přechodu mezi konfiguracemi je tedy závislá na funkci přechodu  $\delta$ ).



**Definice 2.5 (Výpočet slova v konečném automatu)** Výpočet slova v konečném automatu je posloupnost konfigurací spojených relací přechodu, začínající počáteční konfigurací s daným slovem a končící některou koncovou konfigurací.



### Příklad 2.1

Výpočet slova v konečném automatu – semaforu na obrázku 2.2 na straně 10 může být například takový:

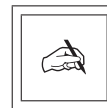
$$(V, ztttttv) \vdash (\check{C}, tttttv) \vdash (\text{itshape } O\check{C}, ttttv) \vdash (Z, tttv) \vdash (OZ, ttv) \vdash (\check{C}, tv) \vdash (O\check{C}, tv) \vdash (Z, v) \vdash (V, \varepsilon)$$

Jiný konečný automat:

$$(q_0, abcd) \vdash (q_3, bcd) \vdash (q_1, cd) \vdash (q_3, d) \vdash (q_3, \varepsilon)$$

kde všechny přechody mezi konfiguracemi jsou určeny funkcí  $\delta: \delta(q_0, a) = q_3$ , atd., a  $q_3$  patří do množiny koncových stavů.

**Definice 2.6 (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.



**Definice 2.7 (Jazyk)** Jazyk  $L$  je množina slov nad danou abecedou  $\Sigma$  (některá podmnožina množiny všech slov utvořených z písmen abecedy  $\Sigma$ ), tedy  $L \subseteq \Sigma^*$ .

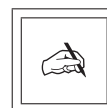
Jazyk může obsahovat také prázdné slovo  $\varepsilon$ .



**Definice 2.8 (Jazyk konečného automatu)** Jazyk konečného automatu  $\mathcal{A}$  je množina všech slov, která automat přijímá, značíme  $L(\mathcal{A})$ .



**Definice 2.9 (Rozpoznání jazyka automatem)** 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).

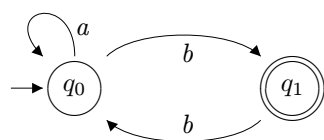


Značíme  $L_j = L(\mathcal{A})$  (jazyk  $L_j$  je rozpoznáván automatem  $\mathcal{A}$ , je jeho jazykem).

### Příklad 2.2

$\mathcal{A} = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\})$

Diagram:



$\delta$  funkce:

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

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

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

Tabulka:

stav \ vstup	a	b
→ q <sub>0</sub>	q <sub>0</sub>	q <sub>1</sub>
← q <sub>1</sub>	-	q <sub>0</sub>

Jeden z možných výpočtů v automatu:

$$(q_0, aab) \vdash (q_0, ab) \vdash (q_0, b) \vdash (q_1, \varepsilon)$$

Jazyk automatu:

$$L(\mathcal{A}) = \{a^*b\} \cdot \{(ba^*b)^i \mid i \geq 0\} = \{(a^*bb)^*a^*b\} = \{a^*(bba^*)^*b\}$$

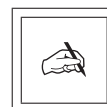
## 2.2 Nedeterminismus

V základní definici konečného automatu existuje pro každé slovo přijímané automatem právě jedna cesta v diagramu automatu, a tedy výpočet je vždy jednoznačný.

Výhodou tohoto postupu je, že takto vytvořený konečný automat se snadněji programuje, protože v klasickém programování je jednoznačnost nutnou podmínkou.

Někdy je však jednodušší vytvořit automat, který tuto vlastnost nemá, tedy pro některá přijímaná slova může existovat více různých cest v diagramu. Zde si takový automat definujeme a ukážeme si také způsob převedení na původní formu.

**Definice 2.10 (Nedeterministický konečný automat)** *Nedeterministický konečný automat (NKA) je takový konečný automat  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ , kde  $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ .*



**Poznámka:**  $\mathcal{P}(Q)$  je potenční množina množiny  $Q$ , je to množina všech jejích podmnožin (včetně prázdné množiny a také samotné množiny  $Q$ ).

V některých stavech na určitý signál může existovat *více než jedna možnost* jak reagovat, dokonce pro některá slova může v grafu automatu existovat více různých cest od počátečního stavu do některého koncového.

V *deterministickém automatu* (DKA) pro jedno slovo existuje právě jedna cesta v grafu (je automatem rozpoznáváno) nebo žádná cesta.

**Definice 2.11 (Jazyk rozpoznávaný NKA)** *Jazyk rozpoznávaný NKA je*

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, w) \vdash^* (q_f, \varepsilon), q_f \in F\}.$$



**Poznámka:** *Jazyk rozpoznávaný nedeterministickým konečným automatem je tedy množina všech slov nad abecedou  $\Sigma$ , pro která existuje alespoň jeden výpočet (cesta v grafu automatu) od počátečního do některého (kteréhokoliv) koncového stavu.*

**Věta 2.1** *Nechť  $\mathcal{A}$  je nedeterministický KA. Potom existuje deterministický KA  $\mathcal{A}'$  takový, že  $L(\mathcal{A}) = L(\mathcal{A}')$  (tj. rozpoznávají stejný jazyk).*



**Důkaz:**  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  nedeterministický (ten máme)

$\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$  deterministický (ten chceme vytvořit)

Stavy nového automatu budou odpovídat množinám stavů původního. Pro každou rovnost  $\delta(q_i, a) = \{q_j, q_k\}$  stavy (množiny)  $\{q_i\}, \{q_j, q_k\}$  budou patřit ke stavům nového automatu.

**Postup:**

- $Q' = \{M \mid M \subseteq Q\} = \mathcal{P}(Q)$  – nová množina stavů bude množinou všech podmnožin původní množiny stavů,





- $q'_0 = \{q_0\}$  – počáteční stav je jednoprvková množina obsahující původní počáteční stav,
- $M \in F' \Leftrightarrow M \cap F \neq \emptyset$  – koncové stavy jsou všechny, které (coby množiny) obsahují alespoň jeden původní koncový stav,
- $\delta'(M, a) = \{q \mid q \in \delta(p, a), p \in M\}$  – všechny prvky množiny zpracujeme podle původního automatu, pak shrneme výsledky do množiny.

□

Pokud pracujeme s reprezentací  $\delta$ -funkce ve tvaru tabulky, můžeme jednoduše postupovat tak, že v tabulce původního automatu „uzávorkujeme“ ohodnocení řádků a buněk do množinových závorek a pak pro každou množinu z buněk, kterou není ohodnocen žádný řádek, přidáme řádek tabulky a doplníme obsah buněk na daném řádku.

Jak určit obsah nové buňky: pokud je řádek ohodnocen množinou  $\{q_i, q_j, \dots\}$ , pak do buňky sepíšeme obsah buněk na řádcích  $\{q_i\}, \{q_j\}, \dots$  v daném sloupci, tedy vlastně sjednocujeme řádky jednotlivých prvků množiny.

**Příklad 2.3**

$$L = \{a, b\}^*bb = \{\{a, b\}^nbb \mid n \geq 0\}$$

$$\mathcal{A} = \{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\}$$

$$\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$$

Nedeterministický:

$\mathcal{A}$	a	b
$\rightarrow q_0$	$q_0$	$q_0, q_1$
$q_1$	-	$q_2$
$\leftarrow q_2$	-	-

Deterministický:

$\mathcal{A}'$	a	b
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\leftarrow \{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$\leftarrow \{q_0, q_1, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$

Odstraníme nepotřebné stavy:

$\mathcal{A}'$	a	b
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$\leftarrow \{q_0, q_1, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$

## 2.3 Totální automat

Obvykle není nutné, aby automat dokázal v každém stavu reagovat na jakýkoliv signál, ale za určitých okolností se tato vlastnost může hodit. Můžeme si představit třeba situaci, kdy programátor chce napsat program dostatečně robustní, který by dokázal reagovat na jakýkoliv vstup, v případě chybného vstupu třeba chybovým hlášením (tedy přechodem do chybového stavu s patřičným ošetřením).

**Definice 2.12 (Totální automat)** *Totální (úplný) konečný automat je deterministický konečný automat, ve kterém lze ve všech stavech reagovat na kterýkoliv symbol abecedy, tj. přechodová funkce  $\delta$  je totální:*

$\forall q \in Q, \forall a \in \Sigma \exists p \in Q : \delta(q, a) = p$  (ke každému stavu a symbolu abecedy existuje stav, do kterého přejdeme z daného stavu na daný symbol).

**Věta 2.2** *Ke každému (deterministickému) konečnému automatu  $\mathcal{A}$  existuje totální automat  $\mathcal{A}'$  takový, že  $L(\mathcal{A}) = L(\mathcal{A}')$ .*

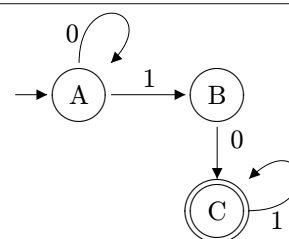
Náznak důkazu – konstrukce:

- převedeme automat na deterministický,
- vytvoříme nový stav  $\emptyset$ , který bude fungovat jako „odpadkový koš“,
- do tohoto stavu nasměrujeme chybějící přechody,
- přidáme smyčku (přechod začínající a končící ve stejném stavu) u stavu  $\emptyset$  pro každý symbol abecedy.

### Příklad 2.4

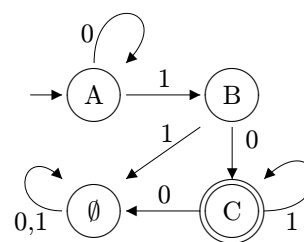
Deterministický:

$\mathcal{A}$	0	1
$\rightarrow A$	A	B
B	C	-
$\leftarrow C$	-	C



Zúplnění:

$\mathcal{A}'$	0	1
$\rightarrow A$	A	B
B	C	$\emptyset$
$\leftarrow C$	$\emptyset$	C
$\emptyset$	$\emptyset$	$\emptyset$



## 2.4 Odstranění nepotřebných stavů automatu

**Definice 2.13 (Nedosažitelný stav)** Nedosažitelný stav je stav  $q_i$  takový, že neexistuje posloupnost přechodů

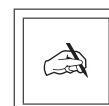
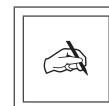
$$(q_0, w) \vdash^* (q_i, w')$$

tedy nelze se do tohoto stavu dostat z počátečního stavu.

**Definice 2.14 (Nadbytečný stav)** Nadbytečný stav je stav  $q_i$  takový, že neexistuje žádná posloupnost přechodů

$$(q_i, w') \vdash^* (q_f, \varepsilon)$$

kde  $q_f \in F$  (koncový stav), tedy z tohoto stavu se nelze dostat do žádného koncového stavu.



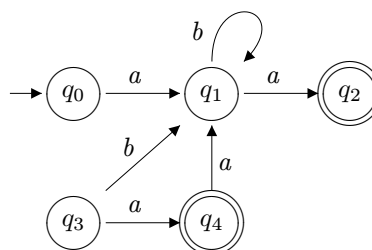
### 2.4.1 Nedosažitelné stavy

Vytvoříme množinu stavů, ke kterým se dá dostat z počátečního stavu – postupujeme od startovacího symbolu.

#### Příklad 2.5

Původní automat:

	$a$	$b$
$\rightarrow q_0$	$q_1$	-
$q_1$	$q_2$	$q_1$
$\leftarrow q_2$	-	-
$q_3$	$q_4$	$q_1$
$\leftarrow q_4$	$q_1$	-



$S_0 = \{q_0\}$ , hledáme prvky  $S_{i-1}$  na označeních řádků, přidáme obsah buněk řádku

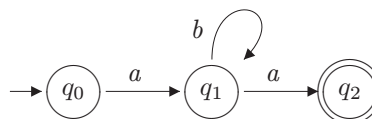
$S_1 = \{q_0, q_1\}$  (z  $q_0$  se dá dostat do  $q_1$ )

$S_2 = \{q_0, q_1, q_2\}$  (z  $q_0$  a  $q_1$  se dá dostat taky do  $q_2$ )

$S_3 = \{q_0, q_1, q_2\} = S_2 \dots$  nová množina stavů

Po úpravě:

	$a$	$b$
$\rightarrow q_0$	$q_1$	-
$q_1$	$q_2$	$q_1$
$\leftarrow q_2$	-	-



**Postup:**

- vytvoříme množinu  $S_0$ , do ní dáme počáteční stav automatu,  $S_0 = \{q_0\}$ ,
- vytvoříme množinu  $S_1$  tak, že do ní dáme prvky množiny  $S_0$  a dále všechny stavy, do kterých vede přechod ze stavů množiny  $S_0$  (tj. zde přidáme všechny stavy, do kterých se dá dostat přímo z  $q_0$ ),
- postupně vytváříme množiny  $S_i$  tak, že do  $S_i$  zařadíme nejdřív obsah množiny  $S_{i-1}$  a pak přidáme všechny stavy, do kterých vede přechod z některého stavu z množiny  $S_{i-1}$ ,
- končíme, když už se do množiny nic nedá přidat, tedy  $S_i = S_{i-1}$ , výsledkem je nová množina stavů.



Vzorec:

$$S_0 = \{q_0\}$$

$$S_i = S_{i-1} \cup \{q \mid \delta(p, a) \ni q, p \in S_{i-1}, a \in \Sigma\}$$

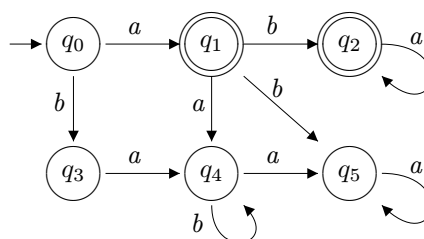
### 2.4.2 Nadbytečné stavy

**Příklad 2.6**

Předpokládáme zde, že nedosažitelné stavy jsou již odstraněny.

Původní automat:

	$a$	$b$
$\rightarrow q_0$	$q_1$	$q_3$
$\leftarrow q_1$	$q_4$	$q_2, q_5$
$\leftarrow q_2$	$q_2$	-
$q_3$	$q_4$	-
$q_4$	$q_5$	$q_4$
$q_5$	$q_5$	-



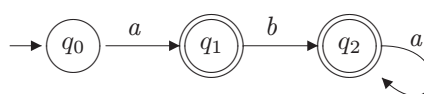
$E_0 = \{q_1, q_2\}$ , hledáme prvky  $E_{i-1}$  v buňkách řádků, přidáme označení řádků

$E_1 = \{q_1, q_2, q_0\}$  (z  $q_0$  se dá dostat do  $q_1$ )

$E_2 = \{q_1, q_2, q_0\} = E_1 \dots$  nová množina stavů

Po úpravě:

	$a$	$b$
$\rightarrow q_0$	$q_1$	$q_3$
$\leftarrow q_1$		$q_2$
$\leftarrow q_2$	$q_2$	-



**Postup:**

- odstraníme nedosažitelné stavy,
- vytvoříme množinu  $E_0$ , do které zařadíme všechny koncové stavy automatu, tj.  $E_0 = F$ ,
- vytvoříme množinu  $E_1$  tak, že do ní dáme prvky množiny  $E_0$  a dále všechny stavy, ze kterých vede přechod do stavů množiny  $E_0$ ,
- postupně vytváříme množiny  $E_i$  tak, že do  $E_i$  zařadíme nejdřív obsah množiny  $E_{i-1}$  a pak přidáme všechny stavy, ze kterých vede přechod do některého stavu z množiny  $E_{i-1}$ ,
- končíme, když už se do množiny nic nedá přidat, tedy  $E_i = E_{i-1}$ , výsledkem je nová množina stavů.



*Vzorec:*

$$E_0 = F$$

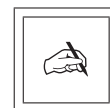
$$E_i = E_{i-1} \cup \{q \mid \delta(q, a) \ni p, p \in E_{i-1}, a \in \Sigma\}$$

# Kapitola 3

## Regulární jazyky

**Definice 3.1 (Regulární jazyk)** *Regulárními jazyky označujeme všechny jazyky, které jsou rozpoznávané konečnými automaty.*

*Jazyk je tedy regulární, pokud lze sestrojít konečný automat, který tento jazyk rozpoznává.*



Ve všech dosud uvedených příkladech byly použity jazyky nekonečné, ale k regulárním jazykům patří také konečné jazyky. Dále se stručně podíváme na možnosti konečných jazyků a potom na nekonečné jazyky a jednu z možností, jak zjistit, zda nekonečný jazyk je či není regulární.

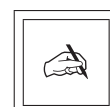
### 3.1 Konečné jazyky

**Definice 3.2 (Konečný jazyk)** *Konečný jazyk je jazyk, který obsahuje konečně mnoho slov.*

**Věta 3.1** *Všechny konečné jazyky jsou regulární.*

**Důkaz:** Pro konečný jazyk sestrojíme konečný automat jednoduše tak, že pro každé slovo jazyka vytvoříme jednu „větev“ výpočtu (nedeterministický automat). Délka větví automatu bude odpovídat délce jednotlivých slov jazyka. □

**Poznámka:** Můžeme mít buď jeden koncový stav společný pro všechna rozpoznávaná slova, a nebo každé slovo bude mít svůj vlastní koncový stav. Toho se využívá například u překladačů, kdy při rozpoznávání konečného množství slov

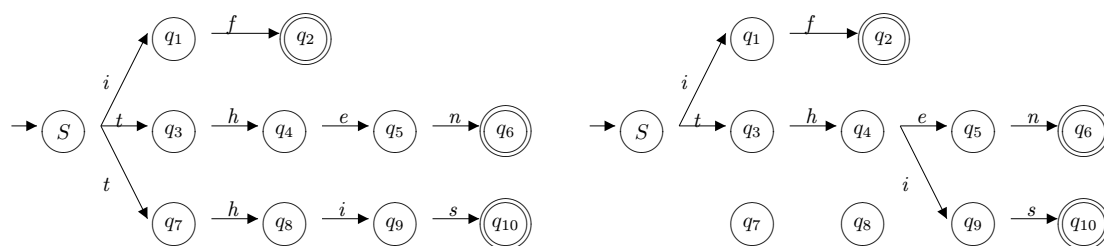


(klíčových slov) pro každé slovo máme zvláštní koncový stav, a podle toho, ve kterém skončí výpočet, určíme, o jaké slovo šlo.

**Příklad 3.1**

$$L = \{if, then, this\}$$

Nedeterministický („otrocký“ postup):      Deterministický:



Reprezentace přechodové funkce tabulkou (chceme, aby byl automat deterministický):

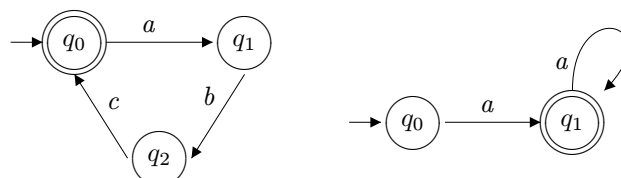
$\mathcal{A}'$	$i$	$f$	$t$	$h$	$e$	$n$	$s$
$\rightarrow S$	$A_1$		$A_2$				
$A_1$		$K_1$					
$A_2$				$A_3$			
$A_3$	$A_5$				$A_4$		
$A_4$						$K_2$	
$A_5$							$K_3$
$\leftarrow K_1$							
$\leftarrow K_2$							
$\leftarrow K_3$							

### 3.2 Pumping Lemma pro regulární jazyky a nekonečné jazyky

**Motivace.** Pumping lemma (pumpovací věta) určuje, že nekonečné regulární jazyky mají jednu konkrétní vlastnost. Proto pokud dokážeme, že daný jazyk tuto vlastnost nemá, můžeme o něm říci, že není regulární.



Abeceda je vždy konečná množina a počet stavů automatu je vždy konečný  $\Rightarrow$  abychom mohli pracovat s dostatečně dlouhými slovy (nekonečného jazyka) s délkou větší než počet stavů automatu, musí být někde smyčka, která způsobí, že se část slova opakuje.



Obrázek 3.1: Ukázky grafů konečných automatů nekonečných jazyků

**Věta 3.2** *Nechť  $L$  je regulární jazyk. Pak existuje celé číslo  $p$ ,  $p > 0$  tak, že každé slovo  $w \in L$ ,  $|w| > p$ , lze rozdělit na tři části  $w = xyz$  tak, že  $y \neq \varepsilon$  ( $|y| > 0$ ) a každé slovo ve tvaru  $xy^kz$ ,  $\forall k \in \mathcal{N}$  je také slovem tohoto jazyka, tedy  $xy^kz \in L$ .*



**Poznámka:** Za číslo  $p$  můžeme dosadit počet stavů automatu, pokud máme sestrojený konečný automat.

### Příklad 3.2

$$L = \{a^{2n} \mid n \geq 0\}$$

Zvolíme  $p = 2$ . Pro nějaké  $i > p$  může být třeba  $a^{2i} = (a^2) \cdot (a^{2(i-1)}) \cdot \varepsilon$  (tedy  $x = a^2$ ,  $y = a^{2(i-1)}$ ,  $z = a^0 = \varepsilon$ )

Použijeme číslo  $k \geq 0$ :

$$k = 0 : xy^0z = a^2 \in L \dots \dots \dots \text{OK}$$

$$k = 1 : xy^1z = a^{2i} \in L \dots \dots \dots \text{OK}$$

$$k = 2 : xy^2z = a^2 (a^{2(i-1)})^2 = a^{2(2i-1)} \in L \dots \dots \dots \text{OK}$$

...

$$k = p : xy^p z = a^2 + p * 2(i - 1) = a^{2(pi-p+1)} \in L \dots \dots \text{OK}$$

**Poznámka:** Pumping lemma se obvykle nepoužívá v základním tvaru, ale spíše pro důkaz, že daný jazyk *není* regulární – větu obrátíme:

Původní: *Když je jazyk regulární, pak existuje  $p \dots$*

Obrátíme: *Když neexistuje  $p \dots$ , pak jazyka není regulární.*



**Postup:** Hledáme dostatečně dlouhé slovo daného jazyka, pro které neexistuje žádné rozdělení  $w = xyz$  takové, že  $xy^kz \in L$ .



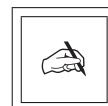
1. zvolíme  $w \in L$  dostatečně dlouhé,
2. zvolíme rozdělení na  $w = xyz$ ,
3. zvolíme  $k$ ,
4. vyrobíme  $w_k = xy^kz$ ,
5. pokud  $w_k \in L \Rightarrow$  návrat k bodu 3, pokud to ještě jde,
6. když to není možné (pro všechna  $k$  slovo  $xy^kz$  patří do jazyka), návrat k bodu 2,
7. když to není možné (všechna rozdělení jsme otestovali a fungují), návrat k bodu 1,
8. jinak: našli jsme slovo  $w$ , pro které neexistuje žádné rozdělení  $xyz$  splňující uvedené podmínky, a tedy jsme dokázali, že  $L$  není regulární jazyk.

Předposlední bod se u regulárního jazyka a také některých jazyků neregulárních provádí do nekonečna, což vyplývá z faktu, že Pumping Lemma je pouze implikace, ne ekvivalence. Tedy věta říká, že každý regulární jazyk má danou vlastnost, ale neříká, že tuto vlastnost nemá žádný neregulární jazyk.

**Poznámka:** Pumping lemma lze ve skutečnosti použít i v případě, že jazyk je konečný – ve větě stojí „... každé slovo  $w \in L, |w| > p$ , lze...“, ovšem když zvolíme  $p$  opravdu dostatečně dlouhé (delší než kterékoliv slovo jazyka), pak vlastně není co testovat a vlastnosti jazyka větě odpovídají.

### 3.3 Uzávěrové vlastnosti třídy regulárních jazyků

**Definice 3.3 (Uzavřenost třídy jazyků vzhledem k operaci  $\varphi$ )** *Daná třída jazyků je uzavřená vzhledem k operaci  $\varphi$ , pokud po uplatnění této operace na jazyky z dané třídy výsledný jazyk patří opět do této třídy.*



**Poznámka:** Možné operace, které z tohoto hlediska zkoumáme, jsou sjednocení, zřetězení, iterace, pozitivní iterace, průnik, doplněk, zrcadlový obraz (reverze), homomorfismus, substituce.

### 3.3.1 Sjednocení

**Věta 3.3** Třída regulárních jazyků je uzavřena vzhledem k operaci sjednocení.

**Důkaz:** Budeme předpokládat, že některé dva regulární jazyky  $L_1$  a  $L_2$  jsou rozpoznávány automaty

$$\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1), L_1 = L(\mathcal{A}_1),$$

$$\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2), L_2 = L(\mathcal{A}_2), \quad Q_1 \cap Q_2 = \emptyset.$$

Vytváříme jazyk  $L = L_1 \cup L_2$  a automat  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ,  $L = L(\mathcal{A})$ :

- Stav  $q_0$  je nový, tedy musí platit  $q_0 \notin Q_1$ ,  $q_0 \notin Q_2$ ,
- $Q = Q_1 \cup Q_2 \cup \{q_0\}$ ,
- $F = F_1 \cup F_2$ ,
- $\Sigma = \Sigma_1 \cup \Sigma_2$ .

$\delta$ -funkce simuluje cesty v původních automatech (přijímá předpisy  $\delta_1$  a  $\delta_2$ ), přidáváme pouze „inicializaci“ – přechod z počátečního stavu  $q_0$ , další přechody již přijímáme. Ze stavu  $q_0$  přecházíme do těch stavů, do kterých se přechází z původních  $q_0^1$  a  $q_0^2$ .

$$\delta(p, a) = \begin{cases} \delta_1(p, a) & | p \in Q_1, \\ \delta_2(p, a) & | p \in Q_2, \\ \delta_1(q_0^1, a) \cup \delta_2(q_0^2, a) & | p = q_0 \end{cases} \quad \square$$

#### Příklad 3.3

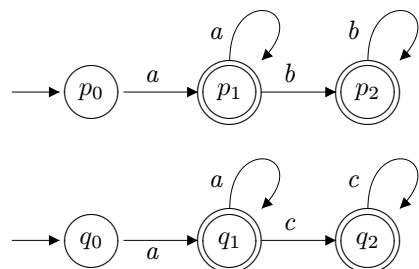
$$L_1 = \{a^i b^j \mid i > 0, j \geq 0\} \quad \mathcal{A}_1 = (\{p_0, p_1, p_2\}, \{a, b\}, \delta_1, p_0, \{p_1, p_2\})$$

$$L_2 = \{a^i c^j \mid i > 0, j \geq 0\} \quad \mathcal{A}_2 = (\{q_0, q_1, q_2\}, \{a, c\}, \delta_2, q_0, \{q_1, q_2\})$$

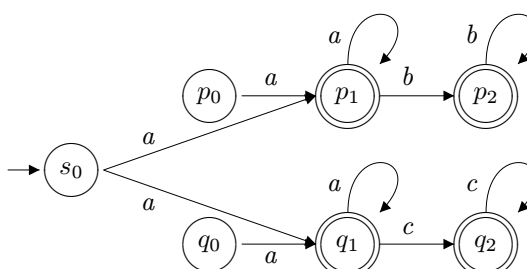
$$L = L_1 \cup L_2 = \{a^i b^j \mid i > 0, j \geq 0\} \cup \{a^i c^j \mid i > 0, j \geq 0\}$$

$$\mathcal{A} = (\{s_0, p_0, p_1, p_2, q_0, q_1, q_2\}, \{a, b, c\}, \delta, s_0, \{p_1, p_2, q_1, q_2\})$$

Původní:



Po sjednocení:



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

$\mathcal{A}_2$	$a$	$b$	$c$
$\rightarrow q_0$	$q_1$		
$\leftarrow q_1$	$q_1$		$q_2$
$\leftarrow q_2$			$q_2$

$\mathcal{A}$	$a$	$b$	$c$
$\rightarrow s_0$	$p_1, q_1$		
$p_0$	$p_1$		
$\leftarrow p_1$	$p_1$	$p_2$	
$\leftarrow p_2$		$p_2$	
$q_0$	$q_1$		
$\leftarrow q_1$	$q_1$		$q_2$
$\leftarrow q_2$			$q_2$

### 3.3.2 Zřetězení

**Věta 3.4** Třída regulárních jazyků je uzavřena vzhledem k operaci zřetězení.

**Důkaz:** Budeme předpokládat, že některé dva regulární jazyky  $L_1$  a  $L_2$  jsou rozpoznávány automaty

$$\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1), L_1 = L(\mathcal{A}_1) \text{ a}$$

$$\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2), L_2 = L(\mathcal{A}_2), Q_1 \cap Q_2 = \emptyset.$$

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

- $q_0 = q_0^1$ ,
- $Q = Q_1 \cup Q_2$ ,
- $\Sigma = \Sigma_1 \cup \Sigma_2$ ,
- pokud  $\varepsilon \notin L_2$ , pak  $F = F_2$ , jinak  $F = F_1 \cup F_2$

$$\delta(p, a) = \begin{cases} \delta_1(p, a) & | p \in Q_1 - F_1, \\ \delta_2(p, a) & | p \in Q_2, \\ \delta_1(p, a) \cup \delta_2(q_0^2, a) & | p \in F_1 \end{cases}$$

□

#### Příklad 3.4

$$L_1 = \{a^i b \mid i \geq 0\}$$

$$\mathcal{A}_1 = (\{p_0, p_1\}, \{a, b\}, \delta_1, p_0, \{p_1\})$$

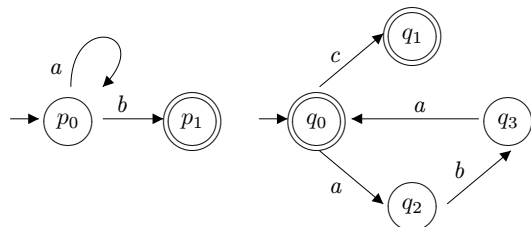
$$L_2 = \{(aba)^i c^{0,1} \mid i \geq 0\}$$

$$\mathcal{A}_2 = (\{q_0, \dots, q_3\}, \{a, b, c\}, \delta_2, q_0, \{q_0, q_1\})$$

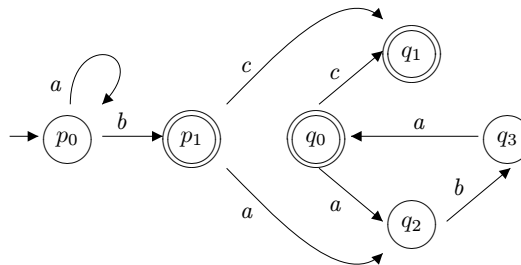
$$\mathcal{A} = (\{p_0, p_1, q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta, p_0, \{p_1, q_0, q_1\})$$



Původní:



Po zřetězení:



### 3.3.3 Iterace

**Věta 3.5** Třída regulárních jazyků je uzavřena vzhledem k operaci iterace.

**Důkaz:** Budeme předpokládat, že některý regulární jazyk  $L_1$  je rozpoznáván automatem

$$\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1), L_1 = L(\mathcal{A}_1).$$

Vytváříme jazyk  $L = L_1^*$  a automat  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ,  $L = L(\mathcal{A})$ .

- $q_0 = q_0^1$ ,
- $Q = Q_1$ ,
- $\Sigma = \Sigma_1$ ,
- $F = F_1 \cup \{q_0\}$ ,

$$\delta(p, a) = \begin{cases} \delta_1(p, a) & | p \notin F_1, \\ \delta_1(p, a) \cup \delta_1(q_0^1, a) & | p \in F_1 \end{cases}$$

□

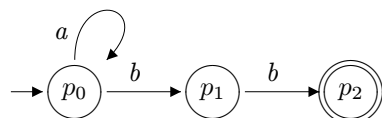
#### Příklad 3.5

$$L_1 = \{a^i bb \mid i \geq 0\}$$

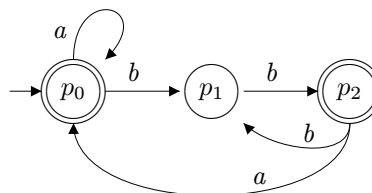
$$\mathcal{A}_1 = (\{p_0, p_1, p_2\}, \{a, b\}, \delta_1, p_0, \{p_2\})$$

$$\mathcal{A} = (\{p_0, p_1, p_2\}, \{a, b\}, \delta, p_0, \{p_0, p_2\})$$

Původní:



Po iteraci:



### 3.3.4 Pozitivní iterace

**Věta 3.6** Třída regulárních jazyků je uzavřena vzhledem k operaci pozitivní iterace.

**Důkaz:** Budeme předpokládat, že některý regulární jazyk  $L_1$  je rozpoznáván automatem  $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$ ,  $L_1 = L(\mathcal{A}_1)$ .

Vytváříme jazyk  $L = L_1^+$  a automat  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ,  $L = L(\mathcal{A})$ .

- $q_0 = q_0^1$ ,
- $Q = Q_1$ ,
- $\Sigma = \Sigma_1$ ,
- $F = F_1$ ,

$$\delta(p, a) = \begin{cases} \delta_1(p, a); & p \notin F_1, \\ \delta_1(p, a) \cup \delta_1(q_0^1, a); & p \in F_1 \end{cases}$$

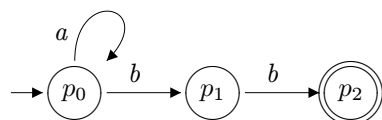
□

#### Příklad 3.6

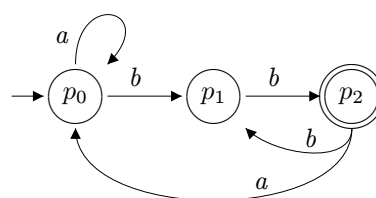
$$L_1 = \{a^i bb \mid i \geq 0\} \quad \mathcal{A}_1 = (\{p_0, p_1, p_2\}, \{a, b\}, \delta_1, p_0, \{p_2\})$$

$$\mathcal{A} = (\{p_0, p_1, p_2\}, \{a, b\}, \delta, p_0, \{p_2\})$$

Původní:



Po pozitivní iteraci:



### 3.3.5 Zrcadlový obraz

**Věta 3.7** Třída regulárních jazyků je uzavřena vzhledem k operaci zrcadlového obrazu (reverze).

**Důkaz:** Budeme předpokládat, že některý regulární jazyk  $L_1$  je rozpoznáván automatem  $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$ ,  $L_1 = L(\mathcal{A}_1)$ .

Vytváříme jazyk  $L = L_1^R$  a automat  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ,  $L = L(\mathcal{A})$ .

- *Princip:* obrátíme všechny „cesty“ tak, aby začínaly tam, kde původně končily a naopak.
- $q_0$  je nově přidán (nelze všechny původní koncové stavy prohlásit za počáteční), nasměrujeme ho tam, odkud původně vedly šipky ke koncovým stavům,
- $Q = Q_1 \cup q_0$ ,
- $\Sigma = \Sigma_1$ ,
- $F = \{q_0^1\}$

$$\delta(p, a) = \begin{cases} \{\bar{p} \mid \delta_1(\bar{p}, a) \ni p\} & \mid p \neq q_0 \\ \{\bar{p} \mid \delta_1(\bar{p}, a) = p\} \cup \{\bar{p} \mid \delta_1(\bar{p}, a) = p \ni q_f, q_f \in F_1\} & \mid p = q_0 \end{cases} \quad \square$$

### Příklad 3.7

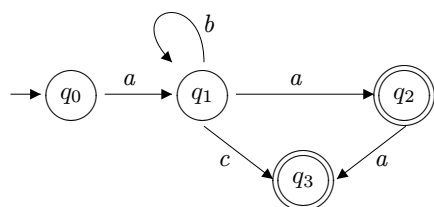
$$L_1 = \{ab^i a^{\{1,2\}} \mid i \geq 0\} \cup \{ab^i c \mid i \geq 0\} = \{ab^i \mid i \geq 0\} \circ \{a, aa, c\}$$

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

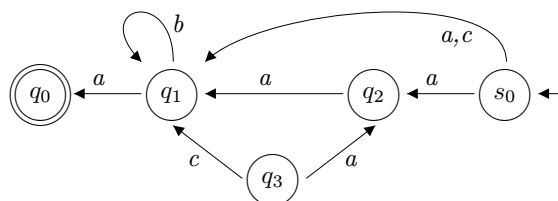
$$\mathcal{A} = (\{q_0, q_1, q_2, q_3, s_0\}, \{a, b, c\}, \delta, s_0, \{q_0\})$$

$$L = L_1^R = \{a, aa, c\} \circ \{b^i a \mid i \geq 0\}$$

Původní:



Po zrcadlení:



### 3.3.6 Průnik

Průnikem dvou jazyků je jazyk obsahující právě ta slova, která jsou v obou zpracovávaných jazycích.

**Věta 3.8** Třída regulárních jazyků je uzavřena vzhledem k operaci průniku.

**Důkaz:** Budeme předpokládat, že některé dva regulární jazyky  $L_1$  a  $L_2$  jsou rozpoznávány 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  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ,  $L = L(\mathcal{A})$ .

Do průniku dvou množin (jazyk není nic jiného než množina slov) řadíme právě ty prvky (slova), které jsou v obou množinách zároveň. Když použijeme konečné automaty, spustíme výpočet daného slova na obou automatech zároveň.

Protože podle definice musí automat v každém kroku zpracovat jeden symbol slova, v obou automatech musí být výpočet téhož slova stejně dlouhý (na cestě v grafu je stejný počet stavů), a tyto stavy si tedy mohou vzájemně odpovídat. Proto ve výsledném – simulujícím – automatu budou stavy reprezentovány uspořádanými dvojicemi, kde první prvek dvojice je stav prvního automatu, druhý prvek je stav druhého automatu.

- $Q = \{[q_i, q_j] \mid q_i \in Q_1, q_j \in Q_2\}$ ,
- $q_0 = [q_0^1, q_0^2]$ ,
- $F = \{[q_i, q_j] \mid q_i \in F_1, q_j \in F_2\}$ ,
- $\Sigma = \Sigma_1 \cup \Sigma_2$  (nebo  $\Sigma = \Sigma_1 \cap \Sigma_2$ , vyjde to nastejno),
- $\delta([q_i, q_j], a) = [\delta_1(q_i, a), \delta_2(q_j, a)]$ ,  $a \in \Sigma$

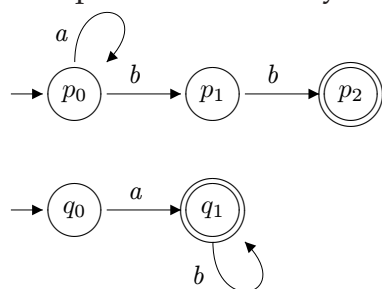
□

### Příklad 3.8

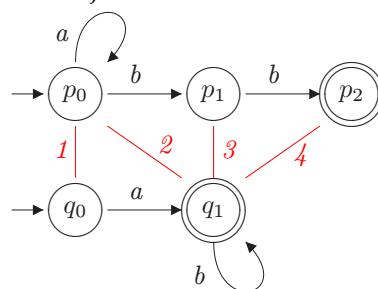
$L_1 = \{a^n b b \mid n \geq 0\}$

$L_2 = \{a b^n \mid n \geq 0\}$

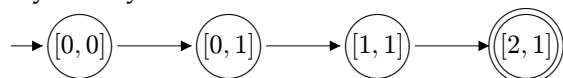
Dva původní automaty:



Pracujeme zároveň v obou automatech:



Výsledný automat:



### 3.3.7 Homomorfismus

**Definice 3.4 (Homomorfismus)** použitý na řetězce znaků s operací zřetězení je jednoznačné zobrazení, které zachovává neutrální prvek (v našem případě prázdný řetězec  $\varepsilon$ ) a také samotnou operaci zřetězení, tedy:

$$h(\varepsilon) = \varepsilon$$

$$h(a \circ w) = h(a) \circ h(w)$$

Zde si pouze ukážeme postup na příkladu, důkaz bude proveden později v jiné kapitole.

**Věta 3.9** Třída regulárních jazyků je uzavřena vzhledem k operaci homomorfismu.

Naznačení postupu důkazu:

Budeme předpokládat, že některý regulární jazyk  $L_1$  je rozpoznáván automatem  $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$ ,  $L_1 = L(\mathcal{A}_1)$ , a dále že existuje homomorfismus  $h$  definovaný pro všechny symboly jazyka  $\Sigma_1$ .

Vytváříme jazyk  $L = h(L_1)$  a automat  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ,  $L = L(\mathcal{A})$ . Postup si ukážeme na příkladu.

#### Příklad 3.9

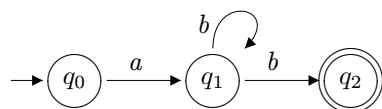
$$L_1 = \{ab^i \mid i > 0\}$$

$$h(a) = ccc,$$

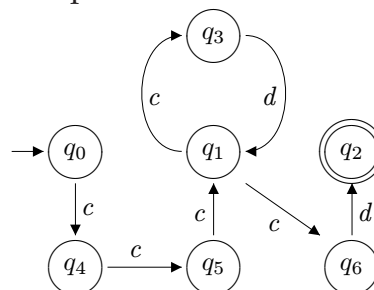
$$h(b) = cd$$

$$h(L_1) = \{c^3(cd)^i \mid i > 0\}$$

Původní automat:



Po aplikaci homomorfismu:





# Kapitola 4

## Regulární výrazy

### 4.1 Možnosti využití regulárních výrazů

Regulární výrazy se v různých podobách využívají v praxi, zejména při vyhledávání (na internetu nebo třeba hledání souboru v počítači), a nebo tehdy, když chceme něco provést s množinou objektů (souborů, textu v souborech, v databázích, apod.) a potřebujeme tuto množinu nějak specifikovat.

*Vyhledávání na internetu (například Google):*

`faktoriál pascal OR C++`

– chceme program pro výpočet faktoriálu v Pascalu nebo C++

`mravenec -Ferda`

– chceme stránky o mravencích, ale ne s Ferdou mravencem

*Vyhledávání na počítači (Windows, DOS, Unixy):*

`*.txt`

– všechny soubory s příponou .txt

`?psa*. *`

– znamená třeba opsané .doc, upsanec .exe, xpsa .xls, atd.

*Vyhledávání na počítači (Unixy):*

`[a-z]*[0-9]`

– všechny řetězce začínající malým písmenem a končící číslicí



`a[!0-9]*.?`

– vše, co začíná malým `a`, přímo za ním může být jakýkoliv řetězec, který nezačíná číslicí, pak je tečka `.` a za ní ještě jeden znak

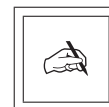
*Možnosti použití:*

- vyhledávání na internetu,
- vyhledávání souborů nebo čehokoliv dalšího textového na počítači,
- prohledávání souborů na počítači (hledáme řetězec) – ve Windows například `findstr`, v Unixech například `grep`,
- databáze,
- elektronické slovníky (cizojazyčné i výkladové),
- podpora v programovacích jazycích,
- atd.

## 4.2 Definice

**Definice 4.1 (Regulární výraz)** *Definujeme pomocnou množinu  $\Phi = \{\emptyset, \varepsilon, +, \circ, *, (, )\}$ .*

*Množina  $RV(\Sigma)$  všech regulárních výrazů nad abecedou  $\Sigma$  je nejmenší množina slov taková, že*



- slova se skládají ze symbolů abecedy  $\Sigma \cup \Phi$ ,  $\Sigma$  a  $\Phi$  jsou disjunktní,
- $\emptyset \in RV(\Sigma)$ ,  $\varepsilon \in RV(\Sigma)$ ,  $a \in RV(\Sigma)$  pro každé  $a \in \Sigma$ ,
- jestliže  $\alpha, \beta \in RV(\Sigma)$ , pak taky
  - $(\alpha + \beta) \in RV(\Sigma)$ ,
  - $(\alpha \cdot \beta) \in RV(\Sigma)$ ,
  - $(\alpha)^* \in RV(\Sigma)$ .

Symbol pro zřetězení se obvykle nemusí psát.

Regulární výraz označuje množinu řetězců s danou vlastností, jazyk je také množina řetězců (slov) s danou vlastností  $\Rightarrow$  každý regulární výraz určuje některý jazyk.

Operace nad jazyky:

$\emptyset$	...	prázdný jazyk
$\varepsilon$	...	$\{\varepsilon\}$ (jazyk obsahující jen slovo s nulovou délkou)
$a, a \in \Sigma$	...	$\{a\}$ (jazyk obsahující jen slovo $a$ s délkou 1)
$\alpha + \beta$	...	$\{\alpha\} \cup \{\beta\}$ (sjednocení)
$\alpha \cdot \beta$	...	$\{\alpha\} \circ \{\beta\}$ (zřetězení)
$(\alpha)^*$	...	$\{\alpha\}^*$ (iterace)

Sjednocení, zřetězení a iteraci označujeme jako *regulární operace*.

#### Příklad 4.1

Ukážeme si několik regulárních jazyků a ekvivalentních regulárních výrazů.

$$L_1 = \{a^i c(ab)^j \mid i, j \geq 0\}$$

$$R_1 = a^* c(ab)^*$$

$$L_2 = \{1^{2i} w \mid i > 0, w \in \{0, 1\}^*\}$$

$$R_2 = (11)(11)^*(0 + 1)^*$$

$$L_3 = \{a^i b \mid i > 0\} \cup \{b^i a \mid i \geq 0\}$$

$$R_3 = aa^*b + b^*a$$

$$L_4 = \{\varepsilon\} \cup (\{ab^4 a^i \mid i \geq 0\} \cup \{b^2 a^{2i+1} \mid i \geq 0\}) \circ \{ca^i \mid i \geq 0\}$$

$$R_4 = \varepsilon + (abbbba^* + bba(aa)^*)ca^*$$

### 4.3 Vztah ke konečným automatům

**Věta 4.1** *Jazyky určené regulárními výrazy jsou právě regulární jazyky, tedy právě jazyky rozpoznávané konečnými automaty.*



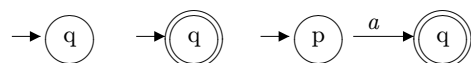
**Důkaz:** „ $\Rightarrow$ “ (podle reg. výrazu sestrojíme konečný automat)

Regulární jazyk je konstruován z množin pomocí operátorů sjednocení, zřetězení a iterace. Všechny tyto operátory mají svůj ekvivalent u regulárních výrazů.



Důkaz lze provést matematickou indukcí:

- *báze:* pro regulární výrazy  $\emptyset, \{\varepsilon\}, \{a\}, a \in \Sigma$  dokážeme jednoduše zkonstruovat konečný automat,



- *indukční krok*: předpokládejme, že pro regulární výrazy  $\alpha$  a  $\beta$  dokážeme sestavit konečné automaty (včetně těch v *bázi*),
- již dříve jsme dokázali (pomocí automatů), že třída regulárních jazyků je uzavřena vzhledem k operacím sjednocení, zřetězení a iterace, tedy použijeme postupy popsané v důkazech těchto operací pro sestavení konečných automatů reprezentujících reg. výrazy  $\alpha + \beta$ ,  $\alpha \cdot \beta$  a  $(\alpha)^*$ .

□

**Důkaz:** „ $\Leftarrow$ “ (podle automatu sestojíme reg. výraz)

Musíme popsat regulárním výrazem všechny cesty vedoucí z počátečního stavu do některého koncového stavu automatu.

Definujeme:

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

Je to množina všech slov takových, která lze v automatu zpracovat tak, že počáteční stav je  $i$  a skončíme ve stavu  $j$ .

Jestliže je množina stavů  $\{1, 2, \dots, n\}$ , pak jazykem automatu je množina

$$\bigcup_{k \in F} R_{1k}$$

(sjednotíme všechny cesty začínající v počátečním stavu a končící v některém z koncových stavů). □

Pro postupnou konstrukci množin  $R_{ij}$  použijeme:

$$R_{ij}^k = \{w \in R_{ij} \mid \text{pokud existuje } m : (i, w) \vdash_{\mathcal{A}_i}^+ (m, u) \vdash_{\mathcal{A}_i}^+ (j, \varepsilon), \quad (4.1)$$

$$w \neq u \neq \varepsilon, \text{ pak } m \leq k\} \quad (4.2)$$

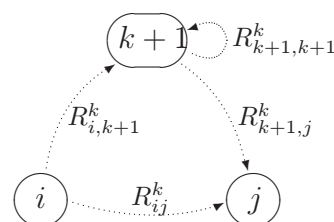


Je to podmnožina množiny  $R_{ij}$  taková, že na cestě v automatu, která je zpracováním slova z této množiny, se nacházejí pouze stavy s indexem *menším nebo rovným* číslu  $k$  (neplatí pro „krajní stavy“  $i$  a  $j$ , ty mohou být i vyšší než  $k$ ).

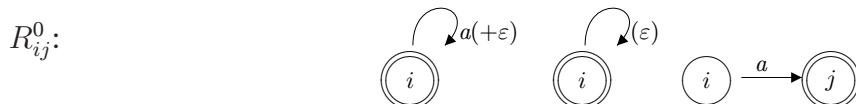
Postupujeme dle následujícího vzorce:

$$R_{ij}^{k+1} = R_{ij}^k + (R_{i,k+1}^k \cdot (R_{k+1,k+1}^k)^* \cdot R_{k+1,j}^k)$$

To je rekurzivní vzorec pro výpočet množin  $R_{ij}^k$  – nejdřív vezmeme všechna slova, která lze rozpoznat cestou přes stavy  $\leq k$ , a pak přidáme slova, která jsou zpracovávána cestou obsahující nejméně jednou stav  $k+1$  (tedy nejdřív cesta z  $i$  do stavu  $k+1$ , pak případně smyčka přes tento stav a stavy  $\leq k$ , a potom zpátky ze stavu  $k+1$  do  $j$ ).



Bází rekurze jsou jednoduché automaty, kde zachycujeme přímý přechod ze stavu  $i$  do  $j$ :



- první případ použijeme, když ve stavu  $i$  existuje smyčka přes tento stav:  $R_{ii}^0 = a + \varepsilon$
- druhý případ použijeme pro stav, ve kterém neexistuje smyčka (máme pouze „prázdný přechod“):  $R_{ii}^0 = \varepsilon$
- třetí případ použijeme pro dva různé stavy  $i$  a  $j$ , mezi kterými vede přímý přechod:  $R_{ij}^0 = a$
- pokud mezi stavy  $i$  a  $j$  nevede přímý přechod, bude  $R_{ij}^0 = \emptyset$ .

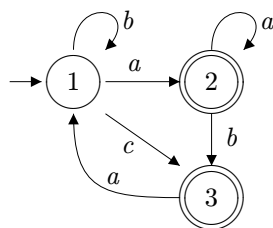
Postup:

- vytvoříme  $R_{ij}^0$  pomocí definice automatu (z tabulky nebo diagramu),
- podle rekurzivního vzorce vypočteme další množiny  $R_{ij}^k$  až ke  $k = n$ ,
- platí  $R_{ij}^n = R_{ij}$ , dostaneme pro  $i = 1$  a  $j \in F$  všechny cesty v automatu vedoucí od počátečního stavu ( $i = 1$ ) do koncových stavů,
- výsledný regulární výraz pro celý automat je  $\bigcup_{j \in F} R_{1j}$ .

#### Příklad 4.2

Uvedený postup si ukážeme na automatu se třemi stavy. Upozorňujeme, že složitost postupu narůstá s množstvím stavů geometrickou řadou.

	$a$	$b$	$c$
$\rightarrow 1$	2	1	3
$\leftarrow 2$	2	3	–
$\leftarrow 3$	1	–	–



$$R_{11}^0 = b + \varepsilon$$

$$R_{11}^1 = (b + \varepsilon) + ((b + \varepsilon)(b + \varepsilon)^*(b + \varepsilon)) = b^*$$

$$R_{12}^0 = a$$

$$R_{12}^1 = a + ((b + \varepsilon)(b + \varepsilon)^*a) = b^*a$$

$$R_{13}^0 = c$$

$$R_{13}^1 = c + ((b + \varepsilon)(b + \varepsilon)^*c) = b^*c$$

$$R_{21}^0 = \emptyset$$

$$R_{21}^1 = \emptyset + \emptyset = \emptyset$$

$$R_{22}^0 = a + \varepsilon$$

$$R_{22}^1 = (a + \varepsilon) + \emptyset = a + \varepsilon$$

$$R_{23}^0 = b$$

$$R_{23}^1 = b + \emptyset = b$$

$$R_{31}^0 = a$$

$$R_{31}^1 = a + (a(b + \varepsilon)^*(b + \varepsilon)) = ab^*$$

$$R_{32}^0 = \emptyset$$

$$R_{32}^1 = \emptyset + (a(b + \varepsilon)^*a) = ab^*a$$

$$R_{33}^0 = \varepsilon$$

$$R_{33}^1 = \varepsilon + (a(b + \varepsilon)^*c) = \varepsilon + ab^*c$$

$$R_{11}^2 = b^* + (b^*a(a + \varepsilon)^*\emptyset) = b^*$$

$$R_{12}^2 = b^*a + (b^*a(a + \varepsilon)^*(a + \varepsilon)) = b^*aa^*$$

$$R_{13}^2 = b^*c + (b^*a(a + \varepsilon)^*b) = b^*c + b^*aa^*b$$

$$R_{21}^2 = \emptyset + ((a + \varepsilon)(a + \varepsilon)^*\emptyset) = \emptyset$$

$$R_{22}^2 = (a + \varepsilon) + ((a + \varepsilon)(a + \varepsilon)^*(a + \varepsilon)) = a^*$$

$$R_{23}^2 = b + ((a + \varepsilon)(a + \varepsilon)^*b) = a^*b$$

$$R_{31}^2 = ab^* + (ab^*a(a + \varepsilon)^*\emptyset) = ab^*$$

$$R_{32}^2 = ab^*a + (ab^*a(a + \varepsilon)^*(a + \varepsilon)) = ab^*aa^*$$

$$R_{33}^2 = (\varepsilon + ab^*c) + (ab^*a(a + \varepsilon)^*b) = \varepsilon + ab^*c + ab^*aa^*b$$

$$R_{12}^3 = b^*aa^* + ((b^*c + b^*aa^*b)(\varepsilon + ab^*c + ab^*aa^*b)^*ab^*aa^*) =$$

$$= b^*aa^* + ((b^*c + b^*aa^*b)(ab^*c + ab^*aa^*b)^*ab^*aa^*)$$

$$R_{13}^3 = (b^*c + b^*aa^*b) + ((b^*c + b^*aa^*b)(\varepsilon + ab^*c + ab^*aa^*b)^*(\varepsilon + ab^*c + ab^*aa^*b)) =$$

$$= (b^*c + b^*aa^*b) + ((b^*c + b^*aa^*b)(+ab^*c + ab^*aa^*b)^*(\varepsilon + ab^*c + ab^*aa^*b))$$

$$R_{12} = R_{12}^3, R_{13} = R_{13}^3$$

$$R(\mathcal{A}) = R_{12} + R_{13}$$

## 4.4 Využití vztahu regulárních výrazů k reg. jazykům

### 4.4.1 Důkazy uzávěrových vlastností regulárních jazyků

Dokážeme uzavřenost třídy regulárních jazyků vzhledem k substituci, a protože homomorfismus je vlastně speciálním případem substituce, tak i uzavřenost třídy regulárních jazyků vzhledem k homomorfismu (zatím jsme si tuto operaci ukázali jen na příkladu).

**Definice 4.2 (Substitute)** *použitá na řetězce znaků s operací zřetězení je zobrazení, které zachovává neutrální prvek (v našem případě prázdný řetězec  $\varepsilon$ ) a také samotnou operaci zřetězení, tedy:*

$$s(\varepsilon) = \varepsilon$$

$$s(a \circ w) = s(a) \circ s(w)$$

*Rozdíl oproti homomorfismu:*

- homomorfismus přiřazuje každému znaku původní abecedy právě jeden řetězec,
- substituce přiřazuje každému znaku původní abecedy množinu řetězců, v případě regulární substituce jde o množinu, která je reprezentovaná regulárním jazykem,
- homomorfismus je tedy speciální případ substituce.

**Věta 4.2** *Třída regulárních jazyků je uzavřena vzhledem k operaci substituce.*

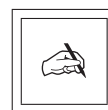
**Důkaz:** Máme jazyk  $L_1$  nad abecedou  $\Sigma_1$  reprezentovaný regulárním výrazem.

Substituce  $\sigma$  je určena tak, že pro každý prvek abecedy  $\Sigma_1$ ,  $a \in \Sigma_1$ , máme regulární výraz  $\sigma(a)$  nad abecedou  $\Delta_a$ .

Po uplatnění substituce vznikne jazyk  $\sigma(L_1)$  nad abecedou

$$\bigcup_{a \in \Sigma_1} \Delta_a$$

tak, že v regulárním výrazu původního jazyka nahradíme všechny symboly abecedy  $\Sigma_1$  příslušnými regulárními výrazy  $\sigma(a)$ .  $\square$



**Příklad 4.3**

$$L = a^*b + (b^*a)^*$$

$$\sigma_1(a) = m^*, \sigma_1(b) = p^*q + p$$

$$\Rightarrow \sigma_1(L) = m^*(p^*q + p) + ((p^*q + p)m^*)^*$$

$$\sigma_2(a) = c, \sigma_2(b) = c^*$$

$$\Rightarrow \sigma_2(L) = c^*c^* + (c^*c)^* = c^*$$

$$\sigma_3(a) = d^*, \sigma_3(b) = d$$

$$\Rightarrow \sigma_3(L) = d^*d + (d^*d^*)^* = d^*$$

$$\sigma_4(a) = e^*, \sigma_4(b) = f^*$$

$$\Rightarrow \sigma_4(L) = e^*f^* + (e^*f^*)^* = (e^*f^*)^* = (e + f)^*$$

**4.4.2 Normovaný automat**

**Definice 4.3 (Normovaný automat)** *Normovaný automat je deterministický automat bez nedosažitelných stavů, jehož stavy jsou označeny jednoznačně (jednoznačným postupem).*



*Účel:*

O dvou ekvivalentních automatech, které nejsou normované, můžeme říci, že jsou stejné až na označení stavů. Normováním si zjednodušíme porovnávání automatů, protože dva ekvivalentní normované automaty jsou shodné i včetně označení stavů (nemusíme prověřovat různé kombinace uspořádaných dvojic stavů).

**Postup:** Stav  $i$  automatu uspořádáme podle nejkratších slov z jazyků  $L_i$  (budeme indexovat sestupně, „největší“ slovo dostane nejmenší index). Porovnáváme podle délky, stejně dlouhá slova podle abecedy.



- u každého stavu  $i$  automatu zjistíme nejkratší slovo příslušného jazyka  $L_i$ , je možné, že budeme potřebovat více takových slov,
- seřadíme podle délky sestupně,
- pokud vyjdou dvě stejně dlouhá slova u více stavů, porovnáme je podle abecedy,



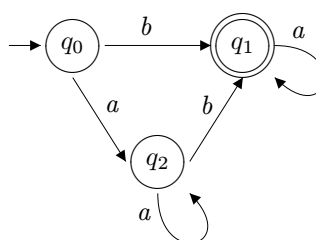
- pokud vyjdou stejná slova u více stavů, použijeme u každého druhé nejkratší slovo (atd. dokud nenajdeme rozdíl, ten najdeme určitě – jsou to různé jazyky),
- seřazené stavy oindexujeme (pojmenujeme) od 1.

**Příklad 4.4**

Znormujeme automat  $\mathcal{A} = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_1\})$

Původní automat:

	a	b
$\rightarrow q_0$	$q_2$	$q_1$
$\leftarrow q_1$	$q_1$	
$q_2$	$q_2$	$q_1$



$L(q_0) = ba^* + aa^*b$ , nejmenší slova jsou  $b, ab, ba, aab, \dots$

$L(q_1) = a^*$ , nejmenší slova jsou  $\varepsilon, a, aa, \dots$

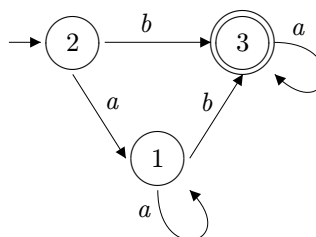
$L(q_2) = a^*b$ , nejmenší slova jsou  $b, ab, aab, \dots$

Jak vidíme, nejkratší slovo obsahuje jazyk  $L(q_1)$ , proto tomuto jazyku přiřadíme nejvyšší index (3).

Zbývající dva jazyky mají první dvě nejmenší slova stejná, až v třetím slově se liší –  $ba < aab$  a proto jazyku  $L(q_0)$  přiřadíme druhý nejvyšší index (2).

Po znormování:

	a	b
$\rightarrow 2$	1	3
$\leftarrow 3$	3	
2	1	3



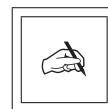
Jak si můžeme všimnout na příkladu 4.4 a jak můžeme také zjistit úvahou, koncové stavy většinou mívají přiřazeny nejvyšší indexy a proti směru výpočtu (tedy směrem od koncových k počátečnímu stavu) se indexy obvykle snižují. To však není tak úplně pravidlem – třeba v příkladu 4.4 nemá počáteční stav nejnížší index.

**4.4.3 Minimalizace konečného automatu**

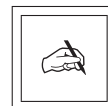
Minimalizací budeme rozumět především snížení počtu stavů automatu. Tento postup může být užitečný při porovnávání jazyků rozpoznávaných dvěma (napo-

hled) různými automaty, ale také při optimalizaci programů vytvořených stavovým programováním podle konečného automatu.

**Definice 4.4 (Ekvivalentní automaty)** Dva konečné automaty  $\mathcal{A}_1$  a  $\mathcal{A}_2$  jsou ekvivalentní, pokud rozpoznávají tentýž jazyk, tj.  $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ .



**Definice 4.5 (Minimální automat)** Konečný automat je minimální, jestliže neexistuje žádný s ním ekvivalentní automat, který má menší počet stavů.



**Věta 4.3** Ke každému konečnému automatu lze sestavit ekvivalentní minimální konečný automat.



#### Postup:

- vytvoříme ekvivalentní deterministický automat,
- odstraníme nedosažitelné stavy,
- vytvoříme ekvivalentní redukovaný automat,
- vhodně přeznačíme stavy – znormujeme automat.



Využijeme konstrukci podílového automatu (viz podílová grupa apod.).

#### Redukce stavů automatu

Redukce stavů automatu  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ :

- pro všechny stavy  $q$  automatu vytvoříme „pomocné“ automaty  $\mathcal{A}_q$  tak, že vezmeme původní automat  $\mathcal{A}$  a stav  $q$  použijeme jako počáteční, tedy

$$\forall q \in Q : \mathcal{A}_q = (Q, \Sigma, \delta, q, F),$$

označíme  $L_q = L(\mathcal{A}_q)$ ,

- zavedeme relaci ekvivalence  $\sim$  na množině stavů automatu  $Q$ , definujeme ji takto: pro jakékoliv dva stavy  $p, q \in Q$  platí:  $p \sim q \iff L_p = L_q$  (dva stavy jsou ekvivalentní, pokud se rovnají jazyky příslušných pomocných automatů),
- když při postupu ze dvou různých stavů dostáváme totéž, pak jsou tyto stavy zaměnitelné  $\Rightarrow$  můžeme je „shrnout“ do jediného stavu, tedy všechny cesty mířící do  $q$  přesměrujeme do  $p$  a stav  $q$  odstraníme,

- toto odstraňování provádíme tak dlouho, dokud v automatu existují ekvivalentní stavy.

**Definice 4.6 (Podílový automat)** Necht'  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  je konečný automat bez nedosažitelných stavů.

Vytvoříme třídy ekvivalence  $\sim$  obsahující některý stav  $q \in Q$ :

$$[q] = \{p \mid p \in Q, p \sim q\}$$

Podílový automat  $\mathcal{A}_\sim$  podle ekvivalence  $\sim$  je  $\mathcal{A}_\sim = (Q_\sim, \Sigma, \delta_\sim, [q_0], F_\sim)$ , kde

- $Q_\sim = \{[q] \mid q \in Q\}$ ,
- $\delta_\sim([q], a) = [\delta(q, a)]$ ,
- $F_\sim = \{[f] \mid f \in F\}$

Konstrukci podílového automatu použijeme pro redukci stavů původního automatu, proto o takto vytvořeném automatu budeme hovořit také jako o *redukovaném*.

Je poměrně snadné dokázat, že podílový automat je ekvivalentní s původním automatem („shrnujeme“ cesty, které jsou shodné). Na příkladu si ukážeme vytvoření podílového automatu včetně hledání ekvivalentních stavů.

**Postup:** Vytvoříme automaty  $\mathcal{A}_i = (Q, \Sigma, \delta, i, F)$  (jako počáteční stav použijeme  $i \in Q$ ), zajímají nás jazyky  $L_i = L(\mathcal{A}_i)$ . Ty pak porovnáme a zpracujeme ekvivalentní.

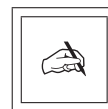
Budeme postupovat následovně:

- vytvoříme množiny  $R_{ij}^0$ , rekurzivně vypočteme další množiny  $R_{ij}^k$  až ke  $k = 8$ ,
- zjistíme jazyky

$$L(i) = \bigcup_{f \in F} R_{if}^8$$

a porovnáme je, pokud některé dva stavy rozpoznávají stejný jazyk, jeden z nich odstraníme s přesměrováním všech přímých cest vedoucích do tohoto stavu,

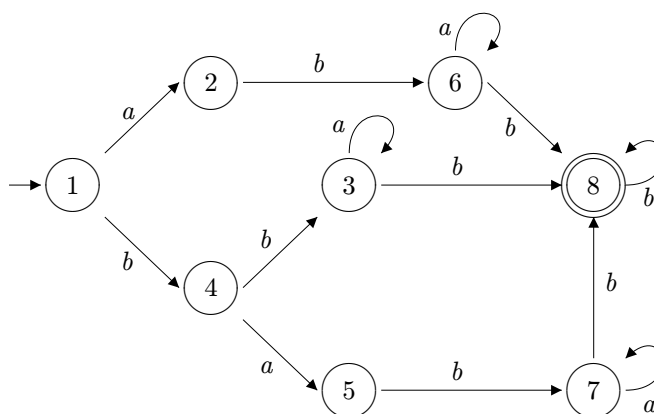
- automat převedeme do normálního tvaru (znormujeme).



**Příklad 4.5**

$\mathcal{A} = (Q, \Sigma, \delta, 1, F) = (\{1, 2, \dots, 8\}, \{a, b\}, \delta, 1, \{8\})$  deterministický bez nedosažitelných stavů

	a	b
→ 1	2	4
2	∅	6
3	3	8
4	5	3
5	∅	7
6	6	8
7	7	8
← 8	∅	8



Jazyky  $L_i$  vytvoříme postupem popsaným výše:

$$L_1 = R_{18}^8 = bba^*bb^* + aba^*bb^* + baba^*bb^*$$

$$L_2 = R_{28}^8 = ba^*bb^*$$

$$L_3 = R_{38}^8 = a^*bb^*$$

$$L_4 = R_{48}^8 = ba^*bb^* + aba^*bb^*$$

$$L_5 = R_{58}^8 = ba^*bb^* = L_2$$

$$L_6 = R_{68}^8 = a^*bb^* = L_3$$

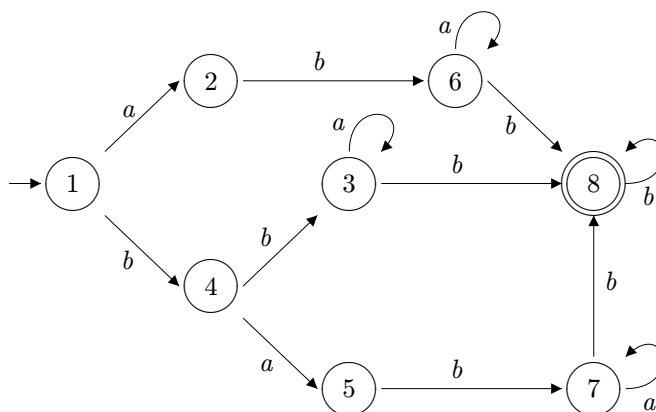
$$L_7 = R_{78}^8 = a^*bb^* = L_3$$

$$L_8 = R_{88}^8 = b^*$$

⇒ zpracujeme stavy 5, 6, 7.

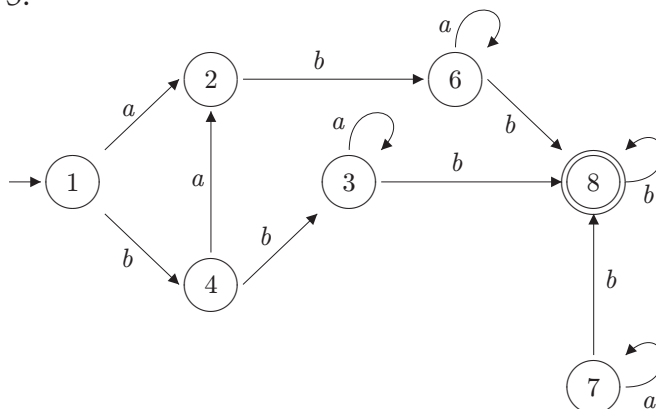
Původní automat:

	a	b
→ 1	2	4
2	∅	6
3	3	8
4	5	3
5	∅	7
6	6	8
7	7	8
← 8	∅	8



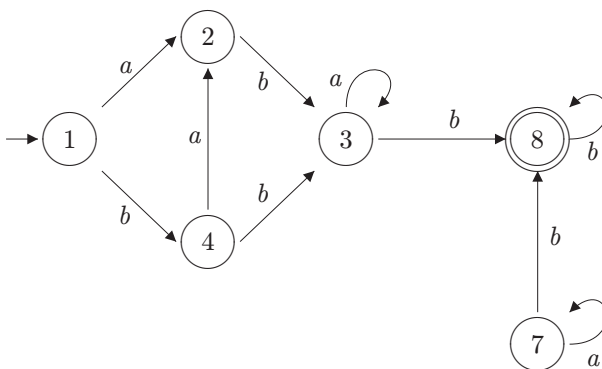
Po odstranění stavu 5:

	$a$	$b$
$\rightarrow 1$	2	4
2	$\emptyset$	6
3	3	8
4	2	3
6	6	8
7	7	8
$\leftarrow 8$	$\emptyset$	8



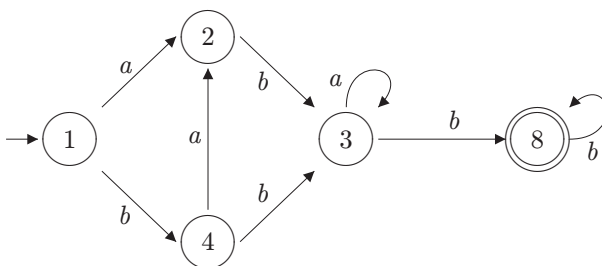
Po odstranění stavu 6:

	$a$	$b$
$\rightarrow 1$	2	4
2	$\emptyset$	3
3	3	8
4	2	3
7	7	8
$\leftarrow 8$	$\emptyset$	8



Po odstranění stavu 7:

	$a$	$b$
$\rightarrow 1$	2	4
2	$\emptyset$	3
3	3	8
4	2	3
$\leftarrow 8$	$\emptyset$	8



Takto vytvořený automat je už sice redukovaný, ale ještě ne minimální. Abychom mohli splnit podmínku snadného porovnávání automatů, musíme náš automat ještě normovat.

Využijeme jazyky dílčích automatů, které jsme zjistili dříve:

$$L_1 = R_{18}^8 = bba^*bb^* + aba^*bb^* + baba^*bb^*$$

$$L_2 = R_{28}^8 = ba^*bb^*$$

$$L_3 = R_{38}^8 = a^*bb^*$$

$$L_4 = R_{48}^8 = ba^*bb^* + aba^*bb^*$$

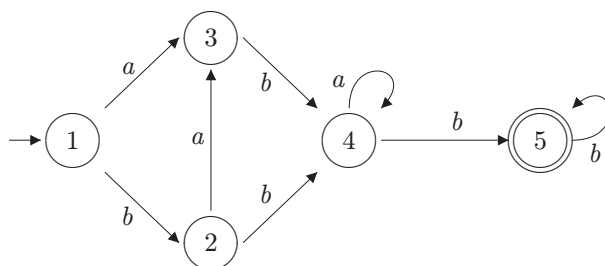
$$L_8 = R_{88}^8 = b^*$$

Z těchto jazyků vybereme nejkratší slova a ta porovnáme:

Stav $i$	Nejkratší slovo jazyka $L_i$	Nové označení stavu
1	$abb$	1
2	$bb, bab, \dots$	2
3	$b$	4
4	$bb, abb, \dots$	3
8	$\varepsilon$	5

Výsledný automat po znormování:

	$a$	$b$
$\rightarrow 1$	2	3
2	$\emptyset$	4
3	2	4
4	4	5
$\leftarrow 5$	$\emptyset$	5



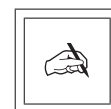
# Kapitola 5

## Formální gramatiky

*Až dosud jsme se zabývali možnostmi zjišťování, zda zadané slovo nebo posloupnost signálů vyhovuje daným podmínkám, tedy zda patří do určitého jazyka. V této kapitole se budeme zabývat možnostmi generování slov vyhovujících daným podmínkám, tedy patřících do určitého jazyka.*

### 5.1 Generování slov jazyka

**Definice 5.1 (Základní pojmy formálních gramatik)**



- Abeceda je neprázdná konečná množina, jejíž prvky nazýváme znaky, symboly, signály.
- Slovo nad abecedou  $\Sigma$  je konečná posloupnost symbolů abecedy  $\Sigma$ , tj.  $w \in \Sigma^*$ .
- Jazyk na abecedou  $\Sigma$  je množina slov  $L$  nad abecedou  $\Sigma$ , tj.  $L \subseteq \Sigma^*$ .
- Automat rozpoznává slovo, tj. dostane již existující slovo na vstup a rozhodne, zda patří do daného jazyka.
- Gramatika vytváří (generuje) slova daného jazyka, popisuje strukturu jazyka.

Použití gramatik při generování slov si nejdřív ukážeme na příkladu.

**Příklad 5.1**

$$L = a^*(bc + cb^*c)$$

$$S \rightarrow aS \quad \textcircled{1}$$

$$S \rightarrow bA \quad \textcircled{2}$$

$$S \rightarrow cB \quad \textcircled{3}$$

$$A \rightarrow c \quad \textcircled{4}$$

$$B \rightarrow bB \quad \textcircled{5}$$

$$B \rightarrow c \quad \textcircled{6}$$

Generujeme slovo  $cbbc$ :  $S \Rightarrow cB \Rightarrow cbB \Rightarrow cbbB \Rightarrow cbbc$  **Není co přepisovat  $\Rightarrow$  konec**

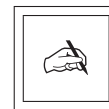
Úspornější a přehlednější způsob zápisu (shrňme pravidla se stejnou levou stranou na jeden řádek):

$$S \rightarrow aS \mid bA \mid cB \quad \textcircled{1}, \textcircled{2}, \textcircled{3}$$

$$A \rightarrow c \quad \textcircled{4}$$

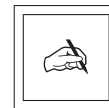
$$B \rightarrow bB \mid c \quad \textcircled{5}, \textcircled{6}$$

**Definice 5.2 (Formální gramatika)** Formální gramatika je uspořádaná čtveřice (posloupnost)  $G = (N, T, P, S)$ , kde



- $N$  je neprázdná konečná množina neterminálních symbolů (neterminální abeceda),
- $T$  je neprázdná konečná množina terminálních symbolů (terminální abeceda), platí  $N \cap T = \emptyset$ ,
- $P$  je neprázdná konečná množina pravidel,  $P \subseteq ((N \cup T)^* N (N \cup T)^*) \times (N \cup T)^*$  jinak:  $\alpha A \beta \rightarrow \gamma$ , kde  $A \in N$ ,  $\alpha, \beta, \gamma \in (N \cup T)^*$
- $S$  je startovací symbol gramatiky,  $S \in N$ .

**Definice 5.3 (Relace kroku odvození  $\Rightarrow$ )** Necht'  $w_1, w_2 \in (N \cup T)^*$  pro gramatiku  $G = (N, T, P, S)$ .



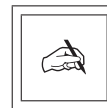
Slovo  $w_2$  lze přímo (v jednom kroku) odvodit ze slova  $w_1$  (píšeme  $w_1 \Rightarrow_G w_2$ , obvykle stačí  $\Rightarrow$ ), jestliže existuje pravidlo  $(\alpha \rightarrow \beta) \in P$ ,  $w_1 = x_1 \alpha x_2$ ,  $w_2 = x_1 \beta x_2$  (podřetězec  $\alpha$  nahradíme řetězcem  $\beta$ ).

Symbol  $\Rightarrow^*$  je reflexivním tranzitivním uzávěrem relace  $\Rightarrow$ , tj. například zápis  $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow w_4 \Rightarrow w_5$  lze zkrátit na  $w_1 \Rightarrow^* w_5$ .

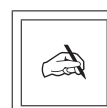


**Definice 5.4 (Jazyk)** *Jazyk generovaný gramatikou  $G = (N, T, P, S)$  je množina*

$$L(G) = \{w \in T^* \mid S \Rightarrow_G^* w\}$$



**Definice 5.5 (Větná forma, věta)** *Větná forma gramatiky  $G = (N, T, P, S)$  je kterékoliv slovo  $\alpha$  takové, že  $S \Rightarrow_G^* \alpha$ , tedy je to kterékoliv slovo, které lze odvodit ze startovacího symbolu pomocí pravidel gramatiky.*



*Věta gramatiky  $G = (N, T, P, S)$  je taková větná forma  $w$ , která se skládá pouze z terminálních symbolů, tedy  $S \Rightarrow_G^* w$ ,  $w \in T^*$ . Můžeme říci, že jazyk generovaný gramatikou je množina všech vět této gramatiky.*

**Definice 5.6 (Ekvivalence gramatik)** *Gramatiky  $G_1$  a  $G_2$  jsou ekvivalentní, pokud generují tentýž jazyk, tedy  $L(G_1) = L(G_2)$ .*



**Definice 5.7 (Derivace)** *Derivace slova  $\alpha$  délky  $n$  v gramatice  $G = (N, T, P, S)$  je posloupnost slov  $\alpha_1, \alpha_2, \dots, \alpha_n$  taková, že*



- $\alpha_1 = S$ ,
- $\alpha_n = \alpha$ ,
- $\alpha_i \Rightarrow_G \alpha_{i+1} \forall i : 1 \leq i \leq n - 1$ .

## 5.2 Regulární gramatiky

**Definice 5.8 (Regulární gramatika)** *Regulární gramatika je taková formální gramatika  $G = (N, T, P, S)$ , kde  $P$  je množina pravidel ve tvaru*



*$A \rightarrow a$  nebo  $A \rightarrow aB$ , kde  $A, B \in N$ ,  $a \in T$ ,*

*pokud se  $S$  nevyskytuje na pravé straně žádného pravidla, může existovat i pravidlo  $S \rightarrow \varepsilon$ .*

**Věta 5.1** *Jazyky generované regulárními gramatikami jsou právě regulární jazyky.*



Postup a princip důkazu si nejdřív ukážeme na příkladech.

### Příklad 5.2

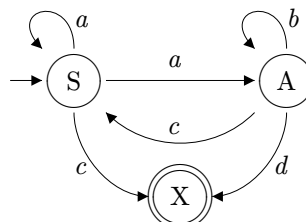
$G = (\{S, A\}, \{a, b, c, d\}, P, S)$ , kde v  $P$  jsou pravidla

$S \rightarrow aS \mid aA \mid c$

$A \rightarrow bA \mid cS \mid d$

Vytvořený konečný automat:

	$a$	$b$	$c$	$d$
$\rightarrow S$	$S, A$		$X$	
$A$		$A$	$S$	$X$
$\leftarrow X$				



$$L = a^*(ab^*ca^*)^*(c + ab^*d) = a^*ab^*(ca^*ab^*)^*d + a^*(ab^*ca^*)^*c$$

**Příklad 5.3**

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

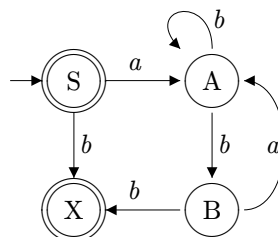
$$S \rightarrow aA \mid b \mid \varepsilon$$

$$A \rightarrow bA \mid bB$$

$$B \rightarrow aA \mid b$$

Vytvořený konečný automat:

	$a$	$b$
$\leftrightarrow S$	$A$	$X$
$A$		$A, B$
$B$	$A$	$X$
$\leftarrow X$		



$$L = \varepsilon + b + ab^*b(ab^*b)^*b$$

**Důkaz:** „ $\Rightarrow$ “ ( $G = (N, T, P, S) \longrightarrow \mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ )

- abeceda  $\Sigma = T$ , počáteční stav  $q_0 = S$ ,
- stavy  $Q = N \cup \{X\}$ , musí být  $X \notin N$  ( $X$  je nově přidáný),
- koncové stavy:
  - pokud  $(S \rightarrow \varepsilon) \in P$ , pak  $F = \{X, S\}$ ,
  - jinak  $F = \{X\}$ ,
- $\delta$  funkce:
  - pro každé pravidlo  $(U \rightarrow aV) \in P$  je  $\delta(U, a) \ni V$ ,
  - pro každé pravidlo  $(U \rightarrow a) \in P$  je  $\delta(U, a) \ni X$ .

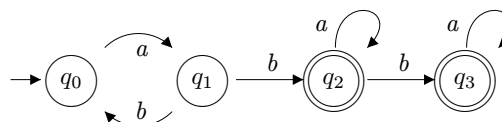


□

**Příklad 5.4**

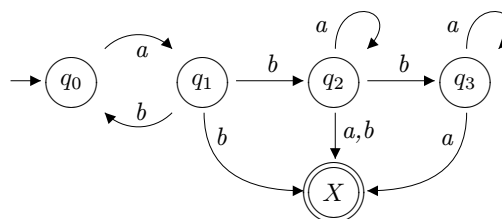
$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_2, q_3\})$$

	a	b
→ q <sub>0</sub>	q <sub>1</sub>	
q <sub>1</sub>		q <sub>0</sub> , q <sub>2</sub>
← q <sub>2</sub>	q <sub>2</sub>	q <sub>3</sub>
← q <sub>3</sub>	q <sub>3</sub>	



Automat upravíme, abychom mohli použít postup přesně opačný postupu převodu gramatiky na automat:

	a	b
→ q <sub>0</sub>	q <sub>1</sub>	
q <sub>1</sub>		q <sub>0</sub> , q <sub>2</sub> , X
q <sub>2</sub>	q <sub>2</sub> , X	q <sub>3</sub> , X
q <sub>3</sub>	q <sub>3</sub> , X	
← X		



Výsledná gramatika a její úprava (jen nahradíme neterminály velkými písmeny):

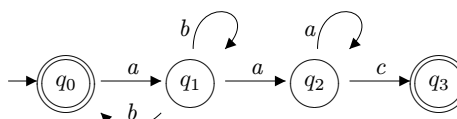
$$\begin{aligned} q_0 &\rightarrow aq_1 & S &\rightarrow aA \\ q_1 &\rightarrow bq_0 \mid bq_2 \mid b & A &\rightarrow bS \mid bB \mid b \\ q_2 &\rightarrow aq_2 \mid bq_3 \mid a \mid b & B &\rightarrow aB \mid bC \mid a \mid b \\ q_3 &\rightarrow aq_3 \mid a & C &\rightarrow aC \mid a \end{aligned}$$

$$L = (ab)^*aba^* + (ab)^*aba^*ba^* = (ab)^*aba^*(\varepsilon + ba^*)$$

**Příklad 5.5**

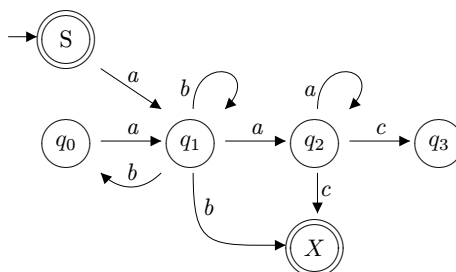
$$\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta, q_0, \{q_0, q_3\})$$

	a	b	c
↔ q <sub>0</sub>	q <sub>1</sub>		
q <sub>1</sub>	q <sub>2</sub>	q <sub>0</sub> , q <sub>1</sub>	
q <sub>2</sub>	q <sub>2</sub>		q <sub>3</sub>
← q <sub>3</sub>			



Upravíme automat, tentokrát musíme řešit i počáteční stav:

	a	b	c
$q_0$	$q_1$		
$q_1$	$q_2$	$q_0, q_1, X$	
$q_2$	$q_2$		$X$
$q_3$			
$\leftarrow X$			
$\leftrightarrow S$	$q_1$		



Stav  $q_3$  se stává nadbytečným, neexistuje žádná cesta vedoucí od něho do koncového stavu.

Výsledná gramatika a převedení neterminálů na velká písmena:

$$S \rightarrow aq_1 \mid \varepsilon$$

$$S \rightarrow aB \mid \varepsilon$$

$$q_0 \rightarrow aq_1$$

$$A \rightarrow aB$$

$$q_1 \rightarrow aq_2 \mid bq_0 \mid bq_1 \mid b$$

$$B \rightarrow aC \mid bA \mid bB \mid b$$

$$q_2 \rightarrow aq_2 \mid c$$

$$C \rightarrow aC \mid c$$

$$L = \varepsilon + (ab^*b)^*ab^*(b + aa^*c)$$

**Důkaz:** „ $\Leftarrow$ “ ( $\mathcal{A} = (Q, \Sigma, \delta, q_0, F) \longrightarrow G = (N, T, P, S)$ )

- upravíme automat – upravený je  $\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$ 
  - $F' = \{X\}$ ,
  - duplikujeme přímé přechody vedoucí do původních koncových stavů, vytvořené přesměrujeme do  $X$ ,
  - pokud  $q_0 \in F$ , vytvoříme nový počáteční stav  $S$ ,  $F' = \{X, S\}$ ,  $q'_0 = S$ ,  $\forall a \in \Sigma : \delta'(S, a) = \delta(q_0, a)$ ,
- $T = \Sigma$ ,  $N = Q' - \{X\}$ , startovací symbol je  $q'_0$ ,
- pravidla  $P$ :
  - pro všechny přechody  $\delta(q_i, a) \ni q_j \quad (q_i \rightarrow aq_j) \in P$ , pokud  $q_j \notin F'$ ,
  - jinak  $(q_i \rightarrow a) \in P$ ,
  - jestliže  $S \in F'$ , pak  $(S \rightarrow \varepsilon) \in P$ .



□

## 5.3 Chomského hierarchie gramatik

Po jazykovědci Noamu Chomském je pojmenována hierarchie základních typů formálních gramatik. Gramatiky v této hierarchii mají jedno společné – *sekvenční* způsob generování slova. To znamená, že v každém kroku odvození je použito právě jedno pravidlo na právě jednom místě v přepisovaném slově.

V hierarchii najdeme čtyři typy gramatik označených čísly 0, 1, 2, 3. *Třídy* (tedy množiny) *jazyků* generované těmito gramatikami označujeme  $\mathcal{L}(0)$ ,  $\mathcal{L}(1)$ ,  $\mathcal{L}(2)$ ,  $\mathcal{L}(3)$ . Mimo hierarchii, ale v těsné vazbě na ni, existují další typy gramatik, na které se také podíváme.

### 5.3.1 Gramatiky v Chomského hierarchii

Chomského hierarchie zahrnuje tedy čtyři typy gramatik. U každého typu si uvedeme případný slovní název a dále označení, které se kromě  $\mathcal{L}$ (číslo) používá pro označení související třídy jazyků, tvar pravidel a ekvivalentní stroj, který rozpoznává stejnou třídu jazyků.

#### Definice 5.9 (Gramatiky Chomského hierarchie)

##### 1. Gramatiky typu 0 (*rekurzivně vyčíslitelné, RE – Recursively Enumerable*)

- žádné zvláštní podmínky na tvar pravidel (jen na levé straně musí být alespoň jeden neterminál):

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

*jiný zápis:*  $\alpha A \beta \rightarrow \gamma$ ,  $A \in N$ ,  $\alpha, \beta, \gamma \in (N \cup T)^*$

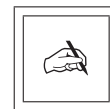
- ekvivalentní stroj: Turingův stroj (TS)

##### 2. Gramatiky typu 1 (*nezkracující, monotónní*)

- pro pravidla musí navíc kromě podmínek gramatik typu 0 platit:

$$\alpha \rightarrow \beta, \quad |\alpha| \leq |\beta|, \quad \alpha, \beta \in (N \cup T)^*$$

- může existovat pravidlo  $S \rightarrow \varepsilon$ , pokud se  $S$  nenachází na pravé straně žádného pravidla



- gramatiky se nazývají nezkracující, protože generované slovo se po jednotlivých krocích buď prodlužuje, nebo se jeho délka nemění, ale nezkracuje se
- ekvivalentní stroj: lineárně ohraničený automat (LOA, LBA – Linearly Bounded Automaton)

### 3. Gramatiky typu 2 (bezkontextové, CF – Context-free)

- pravidla ve tvaru

$$A \rightarrow \alpha, \quad A \in N, \alpha \in (N \cup T)^*$$

- ekvivalentní stroj: zásobníkový automat (ZA)

### 4. Gramatiky typu 3 (regulární, REG – Regular)

- pravidla ve tvaru

$$A \rightarrow aB, \quad A \rightarrow a, \quad A, B \in N, a \in T$$

- může existovat pravidlo  $S \rightarrow \varepsilon$ , pokud se  $S$  nenachází na pravé straně žádného pravidla
- ekvivalentní stroj: konečný automat (KA)

**Věta 5.2** Mezi třídami jazyků generovaných gramatikami v Chomského hierarchii jsou tyto vztahy:

$$\mathcal{L}(3) \subset \mathcal{L}(2) \subset \mathcal{L}(1) \subset \mathcal{L}(0)$$

Pro korektní důkaz tohoto tvrzení ještě nemáme dostatek znalostí, proto si jen uvedeme jazyky, které lze použít v důkazu prvních dvou inkluzí:

$$L_{CF} = \{a^n b^n \mid n \geq 1\} \in (\mathcal{L}(2) - \mathcal{L}(3))$$

$$L_{MON} = \{a^n b^n c^n \mid n \geq 1\} \in (\mathcal{L}(1) - \mathcal{L}(2))$$

## 5.3.2 Související typy gramatik

Dále v textu budeme také pracovat s těmito typy gramatik:

### 1. Kontextové gramatiky (CS – Context Sensitive)

- pravidla ve tvaru

$$\alpha A \beta \rightarrow \alpha \gamma \beta, \quad \gamma \neq \varepsilon, A \in N, \alpha, \beta, \gamma \in (N \cup T)^*$$



- může existovat pravidlo  $S \rightarrow \varepsilon$ , pokud se  $S$  nenachází na pravé straně žádného pravidla
- tato třída jazyků je ekvivalentní třídě jazyků typu 1 (nezkracujícím) –  $CS \cong \mathcal{L}(1)$

## 2. Lineární gramatiky ( $LIN$ – Linear)

- pravidla ve tvaru

$$A \rightarrow \alpha B \beta, A \rightarrow \alpha, \quad A, B \in N, \alpha, \beta \in T^*$$

- speciální případy lineárních gramatik:
  - pravolineární ( $RLIN$ )  $A \rightarrow \beta B, A \rightarrow \beta$ , stejně definované jako regulární
  - levolineární ( $LLIN$ )  $A \rightarrow B \beta, A \rightarrow \beta$
- $LIN$  je speciální případ  $CF$  (bezkontextových) gramatik, dá se dokázat, že  $LIN \subset CF$
- $RLIN = LLIN = REG$ , dá se dokázat, že  $REG \subset LIN$
- $L = \{a^n b^n \mid n \geq 1\}^+ \in (CF - LIN)$
- $L = \{a^n b^n \mid n \geq 1\} \in (LIN - REG)$

## 3. Gramatiky generující konečné jazyky ( $FIN$ – Finite languages)

- konečné jazyky lze generovat regulárními gramatikami:
  - $S \rightarrow$  první slovo  $\mid$  druhé slovo  $\mid$  třetí slovo  $\mid \dots$
- platí  $FIN \subset REG$
- $L = a^* \in (REG - FIN)$

## 5.4 Operace nad slovy a jazyky

Následující definice se budou týkat jazyků  $L_1$  a  $L_2$  nad abecedou  $\Sigma$ , případně slov  $w_1$  a  $w_2$  nad touto abecedou.

**Definice 5.10 (Operace zřetězení)** Operace zřetězení slov je definována následovně: necht'  $w_1 = a_1 a_2 \dots a_n, w_2 = b_1 b_2 \dots b_m$ . Jejich zřetězením je slovo  $w = w_1 \cdot w_2 = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$  o délce  $n + m$ .

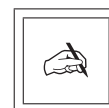


Operace zřetězení jazyků je definována následovně: Zřetězením jazyků  $L_1$  a  $L_2$  je jazyk

$$L = L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$

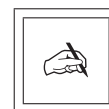
**Definice 5.11 (Operace sjednocení jazyků)** Sjednocením jazyků  $L_1$  a  $L_2$  je jazyk

$$L = L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$$



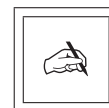
**Definice 5.12 (Operace průniku jazyků)** Průnikem jazyků  $L_1$  a  $L_2$  je jazyk

$$L = L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$$



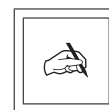
**Definice 5.13 (Operace komplementu (doplňku) jazyka)** Komplementem jazyka  $L_1$  v abecedě  $\Sigma$  je jazyk

$$L = \overline{L_1} = \{w \in \Sigma^* \mid w \notin L_1\}$$



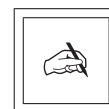
**Definice 5.14 (Operace uzávěru jazyka)** (iterace, nekonečné sjednocení) uzávěrem jazyka  $L_1$  je jazyk

$$L = L_1^* = \bigcup_{i=0}^{\infty} L_1^i, \text{ kde } L_1^i = \underbrace{L_1 \cdot L_1 \cdot \dots \cdot L_1}_{\text{celkem } i \times}$$



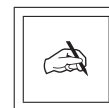
**Definice 5.15 (Operace bez  $\varepsilon$ -nového uzávěru jazyka)** Je definována podobně:

$$L = L_1^+ = \bigcup_{i=1}^{\infty} L_1^i, \text{ kde } L_1^i = \underbrace{L_1 \cdot L_1 \cdot \dots \cdot L_1}_{\text{celkem } i \times}$$



(rozdíl je v indexu pod sumou, zde je  $i > 0$ )

**Definice 5.16 (Operace zrcadlového obrazu)** Operace zrcadlového obrazu slova je definována následovně: necht'  $w_1 = a_1 a_2 \dots a_n$ . Jeho zrcadlovým obrazem je slovo  $w = w_1^R = a_n a_{n-1} \dots a_2 a_1$ .



Operace zrcadlového obrazu jazyka je definována následovně: Zrcadlovým obrazem jazyka  $L_1$  je jazyk

$$L = L_1^R = \{w^R \mid w \in L_1\}$$



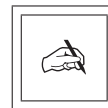
# Kapitola 6

## Bezkontextové jazyky

V této kapitole se budeme zabývat jazyky odpovídajícími třídě jazyků  $\mathcal{L}(2)$  (resp. CF) v Chomského hierarchii, tedy bezkontextovým gramatikám.

### 6.1 Bezkontextové gramatiky

**Definice 6.1 (Bezkontextová gramatika)** Bezkontextová gramatika je gramatika, jejíž pravidla jsou ve tvaru  $A \rightarrow \alpha$ , kde  $A \in N$ ,  $\alpha \in (N \cup T)^*$



#### Příklad 6.1

$$L_1 = \{ww^R \mid w \in \{a, b\}^*\}$$

$G_1 = (\{S\}, \{a, b\}, P, S)$ , kde množina  $P$  obsahuje pravidla  $S \rightarrow aSa \mid bSb \mid \varepsilon$

Derivace:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbbba$$

#### Příklad 6.2

$$L_2 = \{a^n b^m c^n \mid m, n > 0\}$$

$G_2 = (\{S, A\}, \{a, b, c\}, P, S)$ , kde množina  $P$  obsahuje pravidla

$$S \rightarrow aSc \mid aAc$$

$$A \rightarrow bA \mid b$$

Derivace:

$$S \Rightarrow aSc \Rightarrow aaAcc \Rightarrow aabAcc \Rightarrow aabbAcc \Rightarrow aabbcc$$

**Příklad 6.3**

$$L_3 = \{0^n 1^m \mid 1 \leq n \leq m\}$$

$G_3 = (\{A\}, \{0, 1\}, P, A)$ , kde množina  $P$  obsahuje pravidla

$$A \rightarrow 0A1 \mid A1 \mid 01$$

Derivace:

$$A \Rightarrow 0A1 \Rightarrow 0A11 \Rightarrow 00111$$

**Příklad 6.4**

$L_4 =$  jazyk matematických výrazů obsahujících

- celá čísla
- operátory  $+$ ,  $*$  (bez ohledu na prioritu)
- závorky

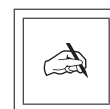
$G_4 = (\{E\}, \{n, +, *, \cdot, /, \}, \{(), P, E\})$ , kde množina  $P$  obsahuje pravidla

$$E \rightarrow E + E \mid E * E \mid (E) \mid n$$

Derivace:

$$\begin{aligned} E &\Rightarrow E * E \Rightarrow E + E * E \Rightarrow n + E * E \Rightarrow n + (E) * E \Rightarrow \\ &\Rightarrow n + (E) * n \Rightarrow n + (n) * n \end{aligned}$$

**Definice 6.2 (Derivační strom)** *Derivační strom některé derivace v bezkontextové gramatice  $G = (N, T, P, S)$  je uspořádaný kořenový strom (tj. souvislý acyklický graf), jehož uzly jsou ohodnoceny symboly z množiny  $N \cup T \cup \{\varepsilon\}$  a platí:*



- všechny uzly kromě kořene mají jednoho předchůdce, kořen nemá žádného,
- pokud je některý uzel ohodnocen symbolem  $A$ , jeho následníci po řadě symboly  $a_1, a_2, \dots, a_n$ , pak v množině pravidel  $P$  gramatiky existuje pravidlo  $A \rightarrow a_1 a_2 \dots a_n$ ,
- jestliže uzel ohodnocený symbolem  $A$  má jediného následníka ohodnoceného symbolem  $\varepsilon$ , pak v  $P$  existuje pravidlo  $A \rightarrow \varepsilon$ ,
- kořen stromu je ohodnocen  $S$ , listy jsou ohodnoceny symboly  $\in T \cup \{\varepsilon\}$ , ostatní symboly  $\in N$ .

V každém kroku vytvoření derivačního stromu tvoří listy větnou formu v gramatice.

**Příklad 6.5**

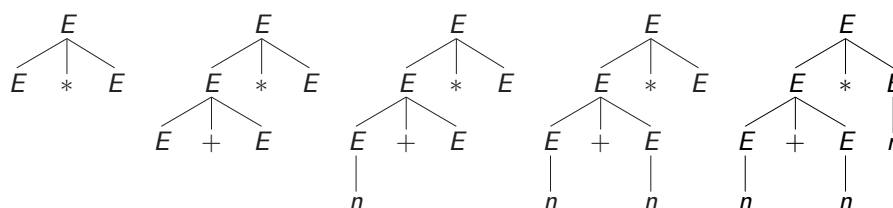
Pravidla:

$$E \rightarrow E + E \mid E * E \mid (E) \mid n$$

Derivace:

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow n + E * E \Rightarrow n + n * E \Rightarrow n + n * n$$

Budeme postupně vytvářet derivační strom této derivace:



Obrázek 6.1: Postupné vytvoření derivačního stromu

## 6.2 Vlastnosti bezkontextových gramatik

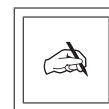
Zde se podíváme především na některé speciální tvary bezkontextových gramatik (u každého tvaru si uvedeme i vztah k obecné definici).

Tyto speciální tvary mají smysl především tehdy, když navrhujeme gramatiku pro určitý účel a potřebujeme, aby měla určité vlastnosti – převedeme do některého speciálního tvaru, který tyto vlastnosti má (například z důvodu snadnější programovatelnosti, třeba u syntaktické analýzy překladače).

### 6.2.1 Bezkontextová gramatika nezkracující

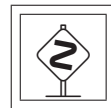
**Definice 6.3 (Nezkracující bezkontextová gramatika)** *Nezkracující bezkontextovou gramatikou (nevypouštějící) nazýváme takovou gramatiku, kde množina pravidel  $P$  buď neobsahuje žádné  $\varepsilon$ -pravidlo, a nebo existuje jediné  $\varepsilon$ -pravidlo  $S \rightarrow \varepsilon$  a zároveň  $S$  není na pravé straně žádného pravidla.*

**Věta 6.1** *Nechť  $G$  je bezkontextová gramatika. Pak existuje gramatika  $G'$  bez  $\varepsilon$ -pravidel taková, že  $L(G') = L(G) - \{\varepsilon\}$ .*



**Důkaz:** Pro každý symbol  $A \in N$ , který lze přepsat na  $\varepsilon$ :

- pro každé pravidlo  $B \rightarrow \beta_0 A \beta_1 A \dots A \beta_n$ , kde je  $A$  na pravé straně pravidla, simulujeme použití pravidla  $A \rightarrow \varepsilon$  na různých místech – přidáme pravidla



$$B \rightarrow \beta_0 \beta_1 A \beta_2 \dots \beta_{n-1} A \beta_n$$

$$B \rightarrow \beta_0 A \beta_1 \beta_2 \dots \beta_{n-1} A \beta_n$$

...

$$B \rightarrow \beta_0 A \beta_1 A \beta_2 \dots \beta_{n-1} \beta_n$$

$$B \rightarrow \beta_0 \beta_1 \beta_2 \dots \beta_{n-1} A \beta_n$$

...

$$B \rightarrow \beta_0 \beta_1 \beta_2 \dots \beta_{n-1} \beta_n$$

- odstraníme pravidlo  $A \rightarrow \varepsilon$

□

**Věta 6.2** Necht'  $G$  je bezkontextová gramatika. Pak existuje gramatika  $G'$  bez  $\varepsilon$ -pravidel nezkracující taková, že  $L(G') = L(G)$ .



**Důkaz:** Postup závisí na tom, zda  $\varepsilon \notin L(G)$ .

- Pokud  $\varepsilon \notin L(G)$ , postupujeme dle předchozího důkazu.
- Pokud  $\varepsilon \in L(G)$ , postupujeme dle následujícího schématu (v  $G'$  bude jediné  $\varepsilon$ -ové pravidlo  $S' \rightarrow \varepsilon$ ):



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

↓

$$G_1 = (N, T, P_1, S)$$

(vytvoříme podle postupu v předchozím důkazu)

↓

$$G' = (N \cup \{S'\}, T, P', S'), \text{ kde } P' = P_1 \cup \{S' \rightarrow \varepsilon \mid S\}$$

□

Převod obecné bezkontextové gramatiky na nezkracující si ukážeme na gramatice v příkladu 6.6.

**Příklad 6.6**

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

$S \rightarrow AaB \mid \varepsilon$

$A \rightarrow AbbA \mid aBc \mid \varepsilon$

$B \rightarrow bAB \mid aS$

Po prvním kroku:

$S \rightarrow AaB \mid aB$

$A \rightarrow AbbA \mid bbA \mid Abb \mid bb \mid aBc$

$B \rightarrow bAB \mid bB \mid aS \mid a$

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

$S' \rightarrow S \mid \varepsilon$

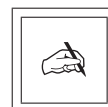
$S \rightarrow AaB \mid aB$

$A \rightarrow AbbA \mid bbA \mid Abb \mid bb \mid aBc$

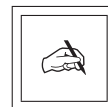
$B \rightarrow bAB \mid bB \mid aS \mid a$

**6.2.2 Redukovaná gramatika**

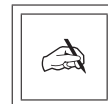
**Definice 6.4 (Nadbytečný neterminál)** (zbytečný)  $X$  je nadbytečný neterminál, pokud neexistuje žádné terminální slovo, které lze z tohoto symbolu vygenerovat, tj. neexistuje derivace  $X \Rightarrow^* w$ ,  $w \in T^*$ .



**Definice 6.5 (Nedostupný symbol)** Symbol  $X \in (N \cup T)$  je nedostupný, jestliže se nemůže objevit v žádné větě formě, tj. neexistuje derivace  $S \Rightarrow^* \alpha X \beta$ ,  $\alpha, \beta \in (N \cup T)^*$ .



**Definice 6.6 (Redukovaná gramatika)** Bezkontextová gramatika je redukovaná, pokud neobsahuje žádné nadbytečné a nedostupné symboly.



**Postup:**

- odstraníme nadbytečné symboly,
- *potom* odstraníme nedostupné symboly.



**Věta 6.3** Ke každé bezkontextové gramatice  $G$  existuje redukovaná gramatika  $G'$  taková, že  $L(G) = L(G')$ .



*Algoritmus odstranění nadbytečných symbolů*

1.  $N_0 = T$
2.  $N_i = N_{i-1} \cup \{A \in N \mid (A \rightarrow \alpha) \in P, \alpha \in N_{i-1}^*\}$
3.  $N_i \neq N_{i-1} \implies \text{inc}(i), \text{přechod na bod 2}$
4.  $N_i = N_{i-1} \implies \text{konec algoritmu, } N' = N_i \cap N$

**Příklad 6.7**

Odstranění nadbytečných symbolů:

$$S \rightarrow aAbC \mid c$$

$$A \rightarrow aA \mid Cc$$

$$B \rightarrow cB \mid dD$$

$$C \rightarrow cB \mid aA \mid b$$

$$D \rightarrow Bd$$

$$N_0 = T$$

$$N_1 = \{S, C\} \cup T$$

$$N_2 = \{S, C, A\} \cup T$$

$$N_3 = N_2 \implies N' = N_3 \cap N = \{S, A, C\} \quad P' = \{S \rightarrow aAbC \mid c, A \rightarrow aA \mid Cc, C \rightarrow aA \mid b\}$$

*Algoritmus odstranění nedosažitelných symbolů*

1.  $V_0 = \{S\}$
2.  $V_i = V_{i-1} \cup \{X \in (N \cup T) \mid (A \rightarrow \alpha X \beta) \in P, A \in V_{i-1}^*\}$
3.  $V_i \neq V_{i-1} \implies \text{inc}(i), \text{přechod na bod 2}$
4.  $V_i = V_{i-1} \implies \text{konec algoritmu,}$   

$$N' = V_i \cap N,$$

$$T' = V_i \cap T$$

**Příklad 6.8**

$$P: S \rightarrow aA \mid bB \mid c$$

$$A \rightarrow cS \mid aA$$

$$B \rightarrow bB \mid cAB$$

$$C \rightarrow aA \mid b$$

$$P': S \rightarrow aA \mid bB \mid c$$

$$A \rightarrow cS \mid aA$$

$$B \rightarrow bB \mid cAB$$

$$C \rightarrow aA \mid bS \rightarrow aA \mid bB \mid c$$

$$A \rightarrow cS \mid aA$$

$$B \rightarrow bB \mid cAB$$

$$C \rightarrow aA \mid b$$

$$P'': N_0 = T$$

$$N_1 = \{S, C\} \cup T$$

$$N_2 = \{S, C, A\} \cup T$$

$$N_3 = N_2$$

$$\implies N' = N_3 \cap N = \{S, A, C\}$$

$$G_{REDUK} = (N'', T'', P'', S)$$

$$V_0 = \{S\}$$

$$V_1 = \{S, A, a, c\}$$

$$V_2 = V_1$$

$$\implies N'' = V_2 \cap N' = \{S, A\}$$

$$\implies T'' = V_2 \cap T = \{a, c\}$$

**Důkaz:**

1. sestrojíme množiny  $N_\varepsilon = \{A \in N \mid A \Rightarrow^* \varepsilon\}$

(podle postupu pro nadbytečné symboly,  $N_0 = \{\varepsilon\}$ )

2. sestrojíme novou množinu pravidel  $P'$ :

(a) pro každé pravidlo ve tvaru

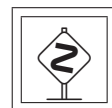
$$A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \dots B_k \alpha_k \in P, k \geq 0, B_i \in N_\varepsilon \forall i, 1 \leq i \leq k, \forall a_j \in \alpha_i : a_j \notin N_\varepsilon$$

do  $P'$  přidáme všechna pravidla ve tvaru  $A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \dots X_k \alpha_k$ , kde  $X_i \in \{B_i, \varepsilon\}$ , pokud vznikne  $A \rightarrow \varepsilon$ , nepřidáme ho,

$$(b) S \in N_\varepsilon \implies (S' \rightarrow \varepsilon \mid S) \in P', N' = N \cup \{S'\}$$

$$(c) S \notin N_\varepsilon \implies N' = N, S' = S$$

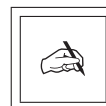
3.  $G' = (N', T, P', S')$



□

### 6.2.3 Gramatika bez jednoduchých pravidel

**Definice 6.7 (Jednoduchá pravidla)** Jednoduchá pravidla jsou pravidla ve tvaru  $A \rightarrow B$ ,  $A, B \in N$  (tedy na pravé i levé straně je jediný neterminál).



**Věta 6.4** Ke každé bezkontextové gramatice  $G$  lze sestavit gramatiku bez jednoduchých pravidel  $G'$  takovou, že  $L(G) = L(G')$ .



**Důkaz:**



1.  $\forall A \in N$  sestojíme množinu  $N_A$ :

$$(a) N_{A,0} = \{A\},$$

$$(b) N_{A,i} = N_{A,i-1} \cup \{X \in N \mid (B \rightarrow X) \in P, B \in N_{A,i-1}\}$$

$$(c) N_{A,i} \neq N_{A,i-1} \implies inc(i), \text{ přesun na b),}$$

$$N_{A,i} = N_{A,i-1} \implies \text{konec algoritmu, } N_A = N_{A,i}$$

2. sestojíme  $P'$ :

$\forall (B \rightarrow \alpha) \in P$ , které není jednoduché,  $\forall A \in N$  takové, že  $B \in N_A$ , dáme do  $P'$  pravidlo  $A \rightarrow \alpha$

□

#### Příklad 6.9

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid i$$

$$N_{E,0} = \{E\}$$

$$N_{T,0} = \{T\}$$

$$N_{E,1} = \{E, T\}$$

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

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

$$N_{F,0} = \{F\} = N_F$$

$$E \rightarrow E + T \mid T * F \mid (E) \mid i$$

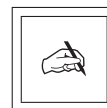
$$T \rightarrow T * F \mid (E) \mid i$$

$$F \rightarrow (E) \mid i$$



### 6.2.4 Další typy bezkontextových gramatik

**Definice 6.8 (Gramatika bez cyklu)** *Gramatika bez cyklu je taková gramatika, ve které neexistuje derivace  $A \Rightarrow^+ A$  pro žádné  $A \in N$ .*

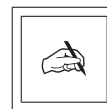


**Poznámka:** Nezkracující gramatika bez jednoduchých pravidel je vždy bez cyklu (pozor, pouze implikace, protože mohou existovat gramatiky bez cyklu, které nejsou nezkracující nebo které nejsou bez jednoduchých pravidel).

**Definice 6.9 (Vlastní gramatika)** *Vlastní gramatika je gramatika bez cyklu, nezkracující, bez nadbytečných symbolů.*

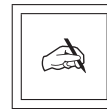


**Definice 6.10 (Gramatika bez levé rekurze)** *Gramatika bez levé rekurze je gramatika, ve které pro žádný neterminál  $A \in N$  neexistuje derivace  $A \Rightarrow^+ A\alpha$ .*



*Přímá levá rekurze znamená existenci pravidla  $A \rightarrow A\alpha$ , nepřímá existenci pravidla  $A \rightarrow \beta A\alpha$ , kde  $\beta \Rightarrow^* \varepsilon$ .*

**Definice 6.11 (Gramatika bez pravé rekurze)** *Obdobně jako u levé rekurze, vyměníme slovo „levá“ za „pravá“.*



**Věta 6.5** *Ke každé bezkontextové gramatice  $G$  existuje gramatika bez levé rekurze  $G'$  taková, že  $L(G) = L(G')$ .*



**Důkaz:** (Odstranění levé rekurze)

1. převedeme gramatiku na vlastní (bez cyklu, nezkracující, bez nadbytečných symbolů),
2. každou sadu pravidel s levou rekurzí

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

nahradíme sadou pravidel

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A' \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

Proč: původní sada pravidel generuje sekvenci řetězců  $\alpha_i$ , rekurze je ukončena tak, že před všechny řetězce  $\alpha_i$  je vygenerován jeden z řetězců  $\beta_j$ , tedy vlastně vygenerujeme totéž, jen jinými pravidly.

$$A \Rightarrow A\alpha_3 \Rightarrow A\alpha_1\alpha_3 \Rightarrow A\alpha_7\alpha_1\alpha_3 \Rightarrow \beta_2\alpha_7\alpha_1\alpha_3$$

□

**Příklad 6.10**

Odstranění levé rekurze:

$$A \rightarrow BC \mid a$$

$$B \rightarrow BaC \mid Ab \mid ba$$

$$C \rightarrow CC \mid b \mid CbA \rightarrow BC \mid a$$

$$B \rightarrow BaC \mid Ab \mid ba$$

$$C \rightarrow CC \mid b \mid Cb$$

$$A \rightarrow BC \mid a$$

$$B \rightarrow AbB' \mid baB' \mid ab \mid ba$$

$$B' \rightarrow aCB' \mid aC$$

$$C \rightarrow bC' \mid b$$

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

## 6.3 Normální formy pro bezkontextové gramatiky

Pro bezkontextové gramatiky můžeme použít tyto normální formy:

1. Chomského normální forma
2. Greibachova normální forma

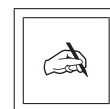
Jejich účel je podobný jako účel dříve uvedených speciálních typů bezkontextových gramatik – jejich vlastnosti se nám za určitých okolností mohou hodit.

### 6.3.1 Chomského normální forma

**Definice 6.12 (Chomského normální forma)** *Bezkontextová gramatika je v Chomského normální formě (CNF), jestliže každé pravidlo z množiny  $P$  je v některém z těchto tvarů:*

- $A \rightarrow BC, A, B, C \in N,$
- $A \rightarrow a, A \in N, a \in T,$
- $S \rightarrow \varepsilon,$  jestliže  $S$  není na pravé straně žádného pravidla.

**Věta 6.6** *Ke každé bezkontextové gramatice  $G$  existuje gramatika  $G'$  v CNF taková, že  $L(G) = L(G')$ .*



**Důkaz:**

1. Převědeme gramatiku do tvaru vlastní gramatiky (nezkracující, odstraníme jednoduchá pravidla).
2.  $\forall A \rightarrow \alpha$ , kde  $|\alpha| \geq 2$ , nahradíme každý výskyt jakéhokoliv terminálu  $a$  neterminálem  $N_a \Rightarrow$  v pravidlech s pravou stranou delší než 1 se vyskytují pouze neterminály.
3. Přidáme pravidla  $N_a \rightarrow a$ .
4. pro každé pravidlo  $A \rightarrow Y_1 Y_2 \dots Y_n$ ,  $n \geq 3$ ,  $A, Y_i \in N$  :

$$A \rightarrow Y_1 Z_1, Z_1 \rightarrow Y_2 Z_2, \dots, Z_{n-3} \rightarrow Y_{n-2} Z_{n-2}, Z_{n-2} \rightarrow Y_{n-1} Y_n$$

(rozkouskujeme dlouhá pravidla)

□

**Příklad 6.11**

*Původní gramatika*

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

$$S \rightarrow A \mid 0SA \mid \varepsilon$$

$$A \rightarrow 1A \mid 1 \mid B1$$

$$B \rightarrow 0B \mid 0 \mid 0SBA$$

*Úprava 1: Nezkracující gramatika*

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

$$S \rightarrow A \mid 0SA \mid 0A$$

$$A \rightarrow 1A \mid 1 \mid B1$$

$$B \rightarrow 0B \mid 0 \mid 0SBA \mid 0BA$$

$$S' \rightarrow S \mid \varepsilon$$

*Úprava 2: Odstranění jednoduchých pravidel*

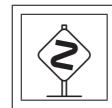
$$G'' = (\{S', S, A, B\}, \{0, 1\}, P'', S')$$

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

$$A \rightarrow 1A \mid 1 \mid B1$$

$$B \rightarrow 0B \mid 0 \mid 0SBA \mid 0BA$$

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



Úprava 3: Terminály v pravidlech s pravou stranou delší než 1

$$G''' = (\{S', S, A, B, N_0, N_1\}, \{0, 1\}, P''', S')$$

$$S \rightarrow N_1A \mid 1 \mid BN_1 \mid N_0SA \mid N_0A$$

$$A \rightarrow N_1A \mid 1 \mid BN_1$$

$$B \rightarrow N_0B \mid 0 \mid N_0SBA \mid N_0BA$$

$$S' \rightarrow N_1A \mid 1 \mid BN_1 \mid N_0SA \mid N_0A \mid \varepsilon$$

$$N_0 \rightarrow 0$$

$$N_1 \rightarrow 1$$

Úprava 4: Rozkouskování pravidel

$$G'''' = (\{S', S, A, B, N_0, N_1, Z_1, Z_2, Z_3\}, \{0, 1\}, P'''' , S')$$

$$S \rightarrow N_1A \mid 1 \mid BN_1 \mid N_0Z_1 \mid N_0A$$

$$Z_1 \rightarrow SA$$

$$A \rightarrow N_1A \mid 1 \mid BN_1$$

$$B \rightarrow N_0B \mid 0 \mid N_0Z_2 \mid N_0Z_3$$

$$Z_2 \rightarrow SZ_3$$

$$Z_3 \rightarrow BA$$

$$S' \rightarrow N_1A \mid 1 \mid BN_1 \mid N_0Z_1 \mid N_0A \mid \varepsilon$$

$$N_0 \rightarrow 0$$

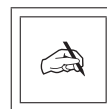
$$N_1 \rightarrow 1$$

### 6.3.2 Greibachova normální forma

**Definice 6.13 (Greibachova normální forma)** *Bezkontextová gramatika je v Greibachově normální formě (GNF), jestliže každé pravidlo z množiny  $P$  je v některém z těchto tvarů:*

- $A \rightarrow aB_1B_2 \dots B_n, n \geq 0, A, B_1, B_2, \dots B_n \in N, a \in T,$
- $S \rightarrow \varepsilon,$  jestliže  $S$  není na pravé straně žádného pravidla.

**Věta 6.7** *Ke každé bezkontextové gramatice  $G$  existuje gramatika  $G'$  v GNF taková, že  $L(G) = L(G')$ .*



**Důkaz:**

1. Převědeme gramatiku do tvaru vlastní gramatiky (nezkracující, odstraníme jednoduchá pravidla), odstraníme levou rekurzi.
2. V každém pravidle za nejlevější neterminál dosazujeme všechna jeho pravidla rekurzivně tak dlouho, dokud řetězec nezačíná terminálem.
3. Terminály  $a$ , které nejsou na začátku některého pravidla, nahradíme pomocnými neterminály  $N_a$ .
4. Přidáme pravidla  $N_a \rightarrow a$ .



□

**Příklad 6.12***Původní gramatika*

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

$$E \rightarrow E + F \mid F$$

$$F \rightarrow (E) \mid i$$

*Úprava 1: Odstranění levé rekurze*

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

$$E \rightarrow FE' \mid F$$

$$E' \rightarrow +FE' \mid +F$$

$$F \rightarrow (E) \mid i$$

*Úprava 2: Odstranění jednoduchých pravidel*

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

$$E \rightarrow FE' \mid (E) \mid i$$

$$E' \rightarrow +FE' \mid +F$$

$$F \rightarrow (E) \mid i$$

*Úprava 3: Rekurzivní nahrazování*

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

$$E \rightarrow (E)E' \mid iE' \mid (E) \mid i$$

$$E' \rightarrow +FE' \mid +F$$

$$F \rightarrow (E) \mid i$$

*Úprava 4: Terminály*

$G''' = (\{E, E', F, N\}, \{(\,), +, i\}, P''', E)$

$E \rightarrow (EN)E' \mid iE' \mid (EN) \mid i$

$E' \rightarrow +FE' \mid +F$

$F \rightarrow (EN) \mid i$

$N) \rightarrow )$

---