evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
Slezská univerzita v Opavě
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Šárka Vavrečková

Teorie jazyků a automatů I

Sbírka úloh pro cvičení

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

Opava, posledníaktualizace 20. února 2017

Anotace: Tato skripta jsou určena pro studenty předmětů Teorie jazyků a automatů I a Základy teoretické informatiky I. Jedná se o sbírku příkladů, která zahrnuje látku probíranou na cvičení daného předmětu. Zde uvedené definice jsou pouze orientační, jejich úkolem je připomenout značení a postupy používané v následných příkladech.

Teorie jazyků a automatů I

Sbírka úloh pro cvičení

RNDr. Šárka Vavrečková, Ph.D.

Dostupné na: http://vavreckova.zam.slu.cz/formal1.html

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

Sázeno v systému LaTeX

Tato inovace předmětu *Teorie jazyků a automatů I, cvičení* je spolufinancována Evropským sociálním fondem a Státním rozpočtem ČR, projekt č. CZ.1.07/2.2.00/28.0014, „Interdisciplinární vzdělávání v ICT s jazykovou kompetencí".

Obsah

Kapitola 1

Jazyky a regulární výrazy

1.1 Řetězec, množina, jazyk

U konečných automatů je nejmenší (atomickou, dále nedělitelnou) jednotkou, se kterou pracujeme, *signál* (*znak*). Množinu signálů, se kterými dokáže konkrétní automat pracovat, nazýváme *abeceda* a značíme symbolem Σ (velké řecké písmeno sigma). Abeceda je vždy konečná množina.

Automaty rozpoznávají (tj. přijímají na svém vstupu) posloupnosti signálů (případně znaků; tyto posloupnosti nazýváme *slova*) – automaty na vstupu postupně čtou signály a zároveň mění svůj vnitřní stav. Jeden automat obvykle dokáže rozpoznávat více takových slov. Množina všech slov, která dokáže automat rozpoznat, je *jazyk rozpoznávaný automatem*. Pokud je automat označen $\mathcal{A}$, pak jazyk jím rozpoznávaný značíme $L(\mathcal{A})$.

Obecně (nejen ve vztahu k automatům) je jazyk *množinou slov nad danou abecedou*.

Protože jazyky mohou být i hodně rozsáhlé množiny, je třeba si zápis jazyka a také jeho slov vhodně zkrátit. Můžeme použít toto značení:

- $disk$ – slovo o délce 4 (skládá se ze čtyř znaků/signálů)
- ε – takto značíme prázdné slovo, tedy řetězec o délce nula (řecké písmeno epsilon)
- a^2, a^3, a^4, ... – počet opakování objektu, který je takto „umocněn“, například a^3 znamená slovo aaa (symbol zřetězení „·“ nemusíme psát), a^0 představuje ε (počet signálů a je nula)
- a^* – operátor * (Kleeneho operátor, iterace) znamená, že objekt, za kterým následuje (signál, prvky množiny, apod.) může být ve slově opakován, a to jakýkoliv počet krát (může být i nula opakování), za hvězdičku můžeme dosadit jakoukoliv nezápornou celočíselnou mocninu, tedy a^* představuje množinu slov $\{\varepsilon, a, aa, aaa, \ldots\}$
- $\{a, b, c\}^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \ldots\}$ (operace hvězdička se provádí přes všechny prvky množiny, kterýkoliv z nich může být použit na kterémkoliv místě)
- $(abc)^4 = abcabcabcabc$ (prvky v posloupnosti na rozdíl od prvků množiny nejsou odděleny čárkou, mezi nimi je vztah zřetězení)
- $(ab)^* = \{\varepsilon, ab, (ab)^2, (ab)^3, \ldots\} = \{\varepsilon, ab, abab, ababab, \ldots\}$
- $\{a, dfg, bb\}^* = \{\varepsilon, a, dfg, bb, aa, adfg, abb, dfga, dfgdfg, dfgbb, aaa, \ldots\}$

- a^+ je pozitivní iterace, uvedený objekt může být v jakémkoliv počtu výskytů podobně jako u operace *, ale nejméně jednou, takže $a^+ = \{a, a^2, a^3, \ldots\}$
- operátor pozitivní iterace se z důvodů snadné zaměnitelnosti s jiným operátorem často přepisuje do tvaru a^*a nebo aa^*
- operátor $+$ může být použit také jako binární: $ab + bc$ například určuje množinu dvou slov, ab a bc, tedy množinu $\{ab, bc\}$
- tento operátor často uplatňujeme na dva výrazy, například $(ab)^* + (ba)^*$ představuje množinu slov, která odpovídají buď předpisu $(ab)^*$ nebo předpisu $(ba)^*$, výsledkem je množina slov $\{\varepsilon, ab, (ab)^2, (ab)^3, \ldots, ba, (ba)^2, (ba)^3, \ldots\}$
- $(a + b)^* = \{a, b\}^*$ – oba zápisy představují množinu všech slov nad abecedou $\{a, b\}$

Protože jazyk je vlastně množina slov, můžeme použít klasický matematický množinový zápis: $\{a^i b a^j \ ; \ i, j \geq 0\}$ je totéž jako a^*ba^*.

Příklad 1.1

Zápis výrazů je často možné minimalizovat (zjednodušit). Například:

- $a^2 \cdot \varepsilon = a^2$ (protože ε je neutrálním prvkem vzhledem k operaci zřetězení, podobně jako číslo 0 u sčítání čísel)
- $a^* + \varepsilon = a^*$ (protože výraz a^* už v sobě zahrnuje slovo ε)
- $(a + \varepsilon)^* = a^*$ (zdůvodněte)
- $(a + aa)^* = a^*$ (zdůvodněte)
- $(ab)^+ + \varepsilon = (ab)^*$ (zdůvodněte)
- $(ab)^* + (a + b)^* = (a + b)^*$
- $(a^2)^3 = a^2a^2a^2 = a^6$

Příklad 1.2

Jazyk může být specifikován také pomocí matematických operací. Například:

- $\{a^{2n} \ ; \ n \geq 0\}$ je totéž jako $(aa)^*$, tedy množina $\{\varepsilon, a^2, a^4, a^6, \ldots\}$
- $\{a^{2n} \ ; \ n \geq 1\}$ je totéž jako $aa(aa)^*$, tedy množina $\{a^2, a^4, a^6, \ldots\}$
- $\{a^{2^n} \ ; \ n \geq 0\}$ je něco jiného (!), jedná se o množinu $\{a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, \ldots\} = \{a^1, a^2, a^4, a^8, \ldots\}$
- $\{w \in \{a, b\}^* \ ; \ |w| > 5\}$ je množina všech slov nad abecedou $\{a, b\}$, jejichž délka je větší než 5
- $\{w \in \{a, b\}^* \ ; \ |w| \leq 5\}$ je množina všech slov nad abecedou $\{a, b\}$, jejichž délka je menší nebo rovna 5, tedy pokud sjednotíme tuto a předchozí množinu, dostaneme: $\{w \in \{a, b\}^* \ ; \ |w| > 5\} \cup \{w \in \{a, b\}^* \ ; \ |w| \leq 5\} = \{a, b\}^*$
- $\{w \in \{a, b\}^* \ ; \ |w|_a = |w|_b\}$ je množina všech slov nad abecedou $\{a, b\}$ takových, že počet znaků a je stejný jako počet znaků b, tj. $\{\varepsilon, ab, ba, aabb, abab, abba, baab, baba, bbaa, \ldots\}$
- $\{a^n b^n \ ; \ n \geq 0\}$ oproti předchozímu přidává podmínku na pořadí (nejdřív znaky a, až pak b), tedy: $\{\varepsilon, ab, aabb, \ldots\}$, do množiny nepatří například slovo aab ani slovo ba

- $\{a^i b^j \ ; \ i, j \geq 0\}$ pro změnu stanovuje pouze podmínku na pořadí: $\{\varepsilon, a, b, aa, bb, ab, aab, abb, \ldots\}$, do této množiny nepatří například slovo ba
- $\{a^i b^j \ ; \ i \geq 2, \ j \geq 1\}$ je definováno podobně jako předchozí, ale máme odlišnou spodní hranici pro hodnotu proměnných i a j, nejkratší slovo je aab

---

Příklad 1.3

Napíšeme všechna slova následujících jazyků kratší než 5.

Postup: nejdřív místo všech „proměnných" indexů * a $^+$ dosadíme nejnižší možnou hodnotu, tj. například ve výrazu a^*b^* vytvoříme slovo $a^0b^0 = \varepsilon$, pak postupně dosazujeme vyšší hodnoty v různých možných kombinacích.

- $ab^* + ba = \{a, ab, ba, abb, abbb, \ldots\}$
- $2 \cdot \{0,1\}^* = \{\, 2, 20, 21, 200, 201, 210, 211, 2000, 2001, 2010, 2011, 2100, 2101, 2110, 2111, \ldots\}$
- $\{a^i bb^j \ ; \ i, j \geq 1\} \cup \{(ab)^i \ ; \ i \geq 0\} = \{abb, aabb, abbb, \ldots\} \cup \{\varepsilon, ab, abab, \ldots\} = \{abb, aabb, abbb, \varepsilon, ab, abab, \ldots\}$

  Zde si všimněme odlišných spodních mezí pro proměnné i, j v obou sjednocovaných jazycích – pokud $i \geq 1$, pak a^i znamená a^+ neboli aa^*.
- $(10)^*01 = \{01, 1001, \ldots\}$
- $(10)^*(01)^* = \{\varepsilon, 10, 01, 1010, 1001, 0101, \ldots\}$
- $\{ba^i cb^j \ ; \ 0 \leq i < j\} = \{bcb, bcbb, \ldots\}$ (všimněte si, že kdyby ve slově bylo jedno a, musela by následovat nejméně dvě b, což by ale znamenalo, že délka slova by nebyla kratší než 5)
- $\{w \in \{a,b\}^* \ ; \ |w|_a < |w|_b\} = \{b, bb, abb, bab, bba, bbb, abbb, babb, bbab, bbba, bbbb, \ldots\}$ (do jazyka však nepatří například slovo $aabb$, protože není splněna podmínka $|w|_a < |w|_b$)
- $\{w \in \{a,b\}^* \ ; \ |w|_a = |w|_b + 1\} = \{a, abb, bab, bba, \ldots\}$ (všimněte si, že „vedlejším důsledkem" podmínky je lichá délka slov jazyka)

Úkoly

Napište všechna slova následujících jazyků kratší než 5.

- $(abc)^*$
- $(a^*b)^*$
- $(ab)^*c$
- $(a^*b)^*c$
- $\{(ab)^i b^j \ ; \ i \geq 0, \ j \geq 1\}$
- $0^*(10)^*1$
- $\{a^i b^j \ ; \ i, j \geq 0, \ i \leq j\}$
- $\{w \in \{a,b\}^* \ ; \ |w|_b = |w|_a + 1\}$

- $\{w \in \{a,b\}^* \ ; \ |w|_a < |w|_b\}$
- $\{w \cdot ba^i \ ; \ w \in \{a,b\}^*, \ i \geq 1\}$
- $\{w \in \{a,b\}^* \ ; \ |w| < 3\}$

1.2 Regulární výrazy

S regulárními výrazy jsme se již setkali, teď se na ně zaměříme formálněji.

Označme pomocnou množinu $\Phi = \{\emptyset, \varepsilon, +, \cdot, *, (,)\}$. Množina $RV(\Sigma)$ všech regulárních výrazů nad abecedou Σ je nejmenší množina slov (řetězců) taková, že

- slova se skládají ze symbolů abecedy $\Sigma \cup \Phi$, Σ a Φ 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 nemusí psát.

Pro operace používané v regulárních výrazech platí podobná pravidla jako pro operace v aritmetických výrazech. Například operátory $\cdot$ a $+$ jsou asociativní a distributivní, $+$ také komutativní. Prvky $\emptyset$ a ε plní v regulárních výrazech podobnou roli jako v aritmetice nula a jednička. Těchto vlastností lze využít při úpravách složitějších regulárních výrazů.

Příklad 1.4

$\emptyset + a = a$	$(a^*b^*)^* = (ab^*)^* = \{a,b\}^* = (a+b)^*$
$\varepsilon \cdot a = a$	$(a+\varepsilon)^* = a^*$
$\varepsilon^* = \varepsilon$	$(a+\varepsilon) \cdot a^* = a^*$
$ab + c = c + ab$	$bb^* + \varepsilon = b^*$
$a(b+c) = ab + ac$	$a(ba)^*b = ab(ab)^* = (ab)^*ab$
$ab^* + ab = a(b^* + b) = ab^*$	$a + (cb)^*a = (\varepsilon + (cb)^*)\,a = (cb)^*a$
$ab^* + a(c+d) = a(b^* + c + d)$	$aba + (ab)^*a = ((ab) + (ab)^*)a = (ab)^*a$

Úkoly

Zjednodušte tyto regulární výrazy:

- $ab^* + a^*(a+\varepsilon)$
- $a^*b(ab)^* + a^*b(ab)^*bb^*$
- $a + bc(bc)^*a$
- $a^* + b(a+\varepsilon) \cdot \emptyset$
- $a^* + a\{a,b\}^*b^*$
- $(ab)^*abaa^* + ab(ab)^*a^*$
- $\{a,b\}^*ba + (ab)^*ba$
- $01(01)^*0 + 0^*(10)^*$
- $01(01)^*0 + 0^*(10)^*\{0,1\}^*$

1.3 Určení jazyka

Když máme sestavit předpis jazyka (tj. množiny řetězců) splňujícího určitou podmínku, musíme vždy zapsat *maximální množinu* vyhovující dané podmínce. Jakýkoliv řetězec vyhovující zadané podmínce musí patřit do takového jazyka.

Příklad 1.5

Sestavíme předpis jazyka L splňujícího následující podmínku:

- L obsahuje všechna slova začínající řetězcem aa a končící symbolem b, je nad abecedou $\{a, b\}$:

  $$\begin{aligned} L &= aa(a+b)^*b \\ &= \{aa\} \cdot \{a,b\}^* \cdot \{b\} \\ &= \{aa \cdot w \cdot b \ ; \ w \in \{a,b\}^*\} \end{aligned}$$

- slova jazyka L obsahují nejméně 2 symboly a a maximálně 10 symbolů b, nad abecedou $\{a, b\}$:

  $$L = \{w \in \{a,b\}^* \ ; \ |w|_a \geq 2, \ |w|_b \leq 10\}$$

- v první části slova jsou pouze symboly a, v druhé části slova jsou pouze symboly b:

  $$\begin{aligned} L &= a^*b^* \\ &= \{a^i b^j \ ; \ i,j \geq 0\} \end{aligned}$$

- v každém slově je stejný počet symbolů a a b, nad abecedou $\{a, b\}$:

  $$L = \{w \in \{a,b\}^* \ ; \ |w|_a = |w|_b\}$$

- ve slovech je symbolů a méně než symbolů b, nad abecedou $\{a, b\}$:

  $$L = \{w \in \{a,b\}^* \ ; \ |w|_a < |w|_b\}$$

- všechna slova mají sudou délku, nad abecedou $\{a, b\}$:

  $$\begin{aligned} L &= ((a+b)^2)^* \\ &= \{w \in \{a,b\}^* \ ; \ |w| = 2n, \ n \in \mathcal{N}_0\} \end{aligned}$$

  ($\mathcal{N}_0$ považujme za množinu všech přirozených čísel, včetně nuly)

- jazyk nad abecedou $\{a\}$, počet symbolů a ve slově je druhou mocninou některého přirozeného čísla nebo nuly:

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

- jazyk nad abecedou $\{a, b\}$, počet symbolů a ve slově je druhou mocninou některého přirozeného čísla nebo nuly:

  $$L = \{w \in \{a,b\}^* \ ; \ |w|_a = n^2, \ n \geq 0\}$$

- jazyk je nad abecedou $\{a, b\}$; slova začínají symbolem a a obsahují sudý počet symbolů b, a nebo začínají symbolem b a obsahují lichý počet symbolů a:

  $$L = \{a \cdot w \ ; \ |w|_b = 2n, \ n \geq 0\} \cup \{b \cdot w \ ; \ |w|_a = 2n+1, \ n \geq 0\}$$

- slova mají méně symbolů a než symbolů b, a zároveň stejný počet symbolů c jako symbolů b, jsou nad abecedou $\{a, b, c\}$:

$$\begin{aligned} L &= \{w \in \{a,b,c\}^* \ ; \ |w|_a < |w|_b\} \cap \{w \in \{a,b,c\}^* \ ; \ |w|_c = |w|_b\} \\ &= \{w \in \{a,b,c\}^* \ ; \ |w|_a < |w|_b, \ |w|_c = |w|_b\} \end{aligned}$$

Úkoly

Sestavte předpis jazyka, kde

1. všechna slova začínají písmenem a a počet symbolů b je větší než 5,
2. počet symbolů a je dvojnásobkem počtu symbolů b,
3. délka slov je některou (přirozená čísla) mocninou čísla 3, jazyk je nad abecedou $\{a, b\}$,
4. počet symbolů b ve slovech je některou (přirozená čísla) mocninou čísla 3, jazyk je nad abecedou $\{a, b\}$,
5. počet symbolů a je větší nebo roven počtu symbolů b ve slově, *a zároveň* délka slov je nejméně 8, jazyk je nad abecedou $\{a, b\}$,
6. počet symbolů a je větší než počet symbolů b ve slově, *a nebo* délka slov je nejméně 8, jazyk je nad abecedou $\{a, b\}$,
7. jedná se o jazyk přirozených binárních čísel nad abecedou $\{0, 1\}$; symbolem 0 může začínat pouze jednociferné číslo (tj. jednoznakový řetězec „0“), všechna ostatní slova začínají symbolem 1, jazyk neobsahuje prázdné slovo,
8. jedná se o jazyk reálných binárních čísel s desetinnou tečkou nad abecedou $\{0, 1, \cdot\}$; pro celou část slova (reálného čísla) platí podmínka z předchozího bodu, je nad abecedou $\{0, 1\}$, dále následuje desetinná tečka s reálnou částí, v reálné části za desetinnou tečkou (také nad abecedou $\{0, 1\}$ musí být alespoň jeden symbol.

1.4 Operace nad jazyky

Protože jazyky jsou množiny řetězců (slov), lze na nich provádět množinové a řetězcové operace. Jazyk reprezentujeme buď množinově nebo výrazem (konkrétněji regulárním výrazem).

Správně bychom měli používat značení operací odpovídající reprezentaci jazyka (tj. množinové operace, jako je například sjednocení či průnik, provádět zásadně na množinových reprezentacích jazyků), ale pokud by způsob reprezentace byl příliš komplikovaný, můžeme od tohoto předpisu upustit. Typicky operaci průniku budeme používat i u reprezentace jazyka regulárním výrazem.

V následujících definicích budeme vždy předpokládat, že jazyky jsou nad abecedou Σ. Nejdřív se budeme zabývat *regulárními operacemi* – sjednocení, zřetězení a iterace, které mají blízký vztah k regulárním výrazům.

1.4.1 Regulární operace

Sjednocení: Slovo patří do sjednocení dvou množin, jestliže je prvkem alespoň jedné z těchto množin. Formální zápis:

$$L_1 \cup L_2 = \{w \in \Sigma^* \ ; \ w \in L_1 \text{ nebo } w \in L_2\} \tag{1.1}$$

Jedná se o jednu z regulárních operací, což znamená, že má svůj obraz v reprezentaci jazyka formou regulárního výrazu. Tam místo symbolu $\cup$ používáme symbol $+$.

Příklad 1.6

- Pokud $L_1 = \{abc, aac, cb\}$ a $L_2 = \{baa, cb\}$, pak
  $L_1 \cup L_2 = \{abc, aac, cb, baa\}$
- Pokud $L_1 = ab^*$ a $L_2 = cb^*$, pak
  $L_1 \cup L_2 = ab^* + cb^* = (a + c)\, b^*$
- Pokud $L_1 = a^*$ a $L_2 = \{a^{2^n} \ ; \ n \geq 0\}$, pak
  $L_1 \cup L_2 = L_1$ (protože zde $L_2 \subset L_1$)
- $L \cup \emptyset = L$ (prázdná množina zde plní roli neutrálního prvku)

Zřetězení: Zřetězení dvou slov není třeba definovat, zapisujeme je $u \cdot v$, kde $u, v \in \Sigma^*$. Zřetězením dvou jazyků je jazyk, jehož slova lze rozdělit na dvě části – první patří do prvního stanoveného jazyka, druhá část do druhého jazyka.

$$L_1 \cdot L_2 = \{u \cdot v \ ; \ u \in L_1, \ v \in L_2\} \tag{1.2}$$

Zřetězení jazyků provedeme jednoduše tak, že po dvojicích zřetězujeme slova těchto jazyků (každé slovo prvního jazyka postupně s každým slovem druhého jazyka).

Příklad 1.7

- Pokud $L_1 = \{abc, aac, cb\}$ a $L_2 = \{baa, cb\}$, pak
  $L_1 \cdot L_2 = \{abcbaa, abccb, aacbaa, aaccb, cbbaa, cbcb\}$
- Pokud $L_1 = \{\varepsilon, ab\}$ a $L_2 = \{bba, cb\}$, pak
  $L_1 \cdot L_2 = \{\varepsilon \cdot baa, \varepsilon \cdot cb, abbba, abcb\} = \{baa, cb, abbba, abcb\}$
- Pokud $L_1 = \{\varepsilon, ab\}$ a $L_2 = \{\varepsilon, cb\}$, pak
  $L_1 \cdot L_2 = \{\varepsilon \cdot \varepsilon, \varepsilon \cdot cb, ab \cdot \varepsilon, abcb\} = \{\varepsilon, cb, ab, abcb\}$
- Pokud $L_1 = a^*$ a $L_2 = b^*$, pak
  $L_1 \cdot L_2 = a^*b^*$
- Pokud $L_1 = a^*$ a $L_2 = \{a^{2^n} \ ; \ n \geq 0\}$, pak
  $L_1 \cdot L_2 = a^*$ (proč?)

- $L \cdot \{\varepsilon\} = L$ (prázdné slovo zde plní úlohu neutrálního prvku)
- $L \cdot \emptyset = \emptyset$ (podobně jako když násobíme nulou)

Iterace: Iterace (Kleeneho uzávěr) je nám již známá operace „hvězdička“. Definujme postupně:

$$L^0 = \{\varepsilon\} \quad \text{(neutrální prvek vzhledem k zřetězení)} \tag{1.3}$$
$$L^1 = L \tag{1.4}$$
$$L^n = \{w_1 \cdot w_2 \cdot \ldots \cdot w_n \ ; \ w_i \in L,\ 1 \leq i \leq n\},\ n \geq 1 \tag{1.5}$$
$$L^{n+1} = L \cdot L^n,\ n \geq 0 \tag{1.6}$$
$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \tag{1.7}$$
$$= \bigcup_{n=0}^{\infty} L^n \tag{1.8}$$

Příklad 1.8

- $L = \{a\}$, pak
  $L^* = a^*$
- Pokud $L = \{abc, aac, cb\}$, pak
  $L^* = (abc + aac + cb)^*$
- Pokud $L = \{a^{2^n} \ ; \ n \geq 0\}$, pak
  $L^* = a^*$ (protože do L patří i slovo a, z něj lze poskládat všechna slova delší než ε)
- $\emptyset^* = \{\varepsilon\}$ (Je to správně? Proč ano/ne? Nápověda: dosaďte si do výše uvedeného vzorce pro iteraci.)
- $\{\varepsilon\}^* = \{\varepsilon\}$

Úkoly

1. Které z výše uvedených regulárních operací jsou binární a která z těchto binárních je komutativní? Jsou asociativní?
2. Jsou dány tyto jazyky:
   $L_1 = \{ab\}$
   $L_2 = a^*$
   $L_3 = (ab)^*$
   $L_4 = \{a^i b^i \ ; \ i \geq 0\}$
   $L_5 = \{\varepsilon, bb\}$.
   Jak vypadají následující jazyky?

- $L_1 \cup L_2$
  $L_1 \cup L_5$
  $L_1 \cup L_3$
  $L_1 \cup L_4$
- $L_1 \cdot L_5$
  $L_1 \cdot L_2$
  $L_1 \cdot L_3$
  $L_3 \cdot L_4$
- $L_1^*, \quad L_3^*$
  L_2^*
  L_4^*
  L_5^*

Jak víme z přednášek, regulární výraz je takové vyjádření (zápis) množiny řetězců, ve kterém používáme operace $+$, zřetězení a iterace. Zde jsme probrali tři operace nad množinami řetězců – sjednocení, zřetězení a iteraci, a nazvali jsme je regulárními operacemi. Shrňme si nyní vztah regulárních operací k regulárním výrazům:

Regulární výraz	**Jazyk v množinovém zápisu**
$\emptyset$	$\emptyset$, tedy prázdný jazyk
ε	$\{\varepsilon\}$ (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)
$\alpha \cdot \beta$	$\{\alpha\} \cdot \{\beta\}$ (zřetězení)
$(\alpha)^*$	$\{\alpha\}^*$ (iterace)

Tabulka 1.1: Vztah mezi zápisem regulárních výrazů a jazyků

1.4.2 Další potřebné operace

Průnik: Do průniku dvou jazyků patří všechna slova, která jsou jak v prvním, tak (zároveň) i v druhém jazyce.

$$L_1 \cap L_2 = \{w \in \Sigma^* \ ;\ w \in L_1 \ \wedge \ w \in L_2\} \tag{1.9}$$

(všimněte si vztahu průniku a konjunkce; v podobném vztahu pro sjednocení a disjunkce)

Příklad 1.9

- Pokud $L_1 = \{abc, aac, cb\}$ a $L_2 = \{baa, cb\}$, pak
  $L_1 \cap L_2 = \{cb\}$
- Pokud $L_1 = \{abc, aac\}$ a $L_2 = \{baa, cb\}$, pak
  $L_1 \cap L_2 = \emptyset$
- Pokud $L_1 = a^*$ a $L_2 = b^*$, pak
  $L_1 \cap L_2 = \{\varepsilon\}$
- Pokud $L_1 = a^*$ a $L_2 = \{a^{2^n} \ ;\ n \geq 0\}$, pak
  $L_1 \cap L_2 = L_2$ (protože zde $L_2 \subset L_1$)
- $L \cap \emptyset = \emptyset$

Rozdíl: Rozdíl dvou jazyků je množina všech slov, která patří do prvního z těchto jazyků a zároveň nepatří do druhého jazyka.

$$L_1 - L_2 = \{w \in \Sigma^* \ ; \ w \in L_1 \ \wedge \ w \notin L_2\} \tag{1.10}$$

Příklad 1.10

- Pokud $L_1 = \{abc, aac, cb\}$ a $L_2 = \{baa, cb\}$, pak
  $L_1 - L_2 = \{abc, aac\}$
- Pokud $L_1 = \{abc, aac\}$ a $L_2 = \{abb, abc, aaa, aab, aac\}$, pak
  $L_1 - L_2 = \emptyset$
- Pokud $L_1 = a^*$ a $L_2 = b^*$, pak
  $L_1 - L_2 = L_1 - \{\varepsilon\} = aa^*$
- $L - \emptyset = L$

Doplněk: Doplněk (komplement) jazyka vzhledem k dané abecedě Σ je množina všech slov nad danou abecedou, která do původního jazyka nepatří. Doplněk jazyka L zapisujeme L^C nebo $\overline{L}$.

$$L^C = \overline{L} = \{w \in \Sigma^* \ ; \ w \notin L\} \tag{1.11}$$

Příklad 1.11

- Pokud $L = \{abc, aac, cb\}$, pak
  $\overline{L} = \Sigma^* - \{abc, aac, cb\}$
- Pokud $L = \{w \in \Sigma^* \ ; \ |w|_a > 10\}$, pak
  $\overline{L} = \{w \in \Sigma^* \ ; \ |w|_a \leq 10\}$
- Pokud $L = \Sigma^*$, pak
  $\overline{L} = \emptyset$ (a naopak)

Také zde platí DeMorganovy zákony:

$$\begin{array}{lcllcl} \overline{L_1 \cup L_2} & = & \overline{L_1} \cap \overline{L_2} & \textit{nebo} & L_1 \cup L_2 & = & \overline{\overline{L_1} \cap \overline{L_2}} \\ \overline{L_1 \cap L_2} & = & \overline{L_1} \cup \overline{L_2} & \textit{nebo} & L_1 \cap L_2 & = & \overline{\overline{L_1} \cup \overline{L_2}} \end{array} \tag{1.12}$$

Množinové operace sjednocení a průniku jsou navzájem *duální* – navzájem na sebe převeditelné (pokud použijeme operaci negace) tak, jak vidíme v DeMorganových zákonech..

Zrcadlový obraz: Zrcadlový obraz (reverze) slova vznikne převrácením tohoto slova. V reverzi jazyka provedeme totéž s každým slovem tohoto jazyka.

$$L^R = \overline{L} = \{w^R \ ; \ w \in L\} \tag{1.13}$$

Příklad 1.12

- $L = \{abc, mnp, \varepsilon\}$, pak
  $L^R = \{cba, pnm, \varepsilon\}$
- Pokud $L = a^*$, pak
  $L^R = a^*$ (změna pořadí symbolů není poznatelná, totéž platí pro všechny jazyky nad jednoznakovou abecedou)
- $L = \{a^n b^n \ ; \ n \geq 0\}$, pak
  $L^R = \{b^n a^n \ ; \ n \geq 0\}$
- $L = \{wcw^R \ ; \ w \in \{a, b\}^*\}$, pak
  $L^R = \{wcw^R \ ; \ w \in \{a, b\}^*\}$

Pozitivní iterace: Pozitivní iterace (operace „plus“ je definována podobně jako iterace:

$$L^+ = L^1 \cup L^2 \cup \ldots \tag{1.14}$$

$$= \bigcup_{n=1}^{\infty} L^n \tag{1.15}$$

Homomorfismus: Morfismus (homomorfismus – je to totéž) je zobrazení h zachovávající neutrální prvek (zde ε) a operaci na dané struktuře (u nás zřetězení) v dané abecedě Σ, tj. splňuje *homomorfní podmínky*:

1. $h(\varepsilon) = \varepsilon$
2. $h(a \cdot w) = h(a) \cdot h(w), \quad a \in \Sigma, \ w \in \Sigma^*$

Pokud zobrazení splňuje homomorfní podmínky, lze je definovat jednodušeji – pro jednotlivé symboly jazyka. Platí vzorec:

$$h(L) = \bigcup_{w \in L} h(w) \tag{1.16}$$

Příklad 1.13

Na jazyk $L = \{a^2 b^i (ab)^{i+1} \ ; \ i \geq 1\}$ uplatníme homomorfismy h_1 a h_2 definované následovně:

$h_1(a) = cd$ $\qquad$ $h_2(a) = c$
$h_1(b) = c^3$ $\qquad$ $h_2(b) = c$

Potom platí:
$h_1(L) = \{(cd)^2 c^{3i} (cdc^3)^{i+1} \ ; \ i \geq 1\}$
$h_2(L) = \{c^{i+2} (cc)^{i+1} \ ; \ i \geq 1\} = \{c^{3i+4} \ ; \ i \geq 1\} = c^4 c^3 \left(c^3\right)^*$

Úkoly

1. Které z výše uvedených operací jsou binární a které z těchto binárních jsou komutativní? Jsou asociativní?
2. Jsou dány tyto jazyky:
   $L_1 = \{ab\}$
   $L_2 = a^*$
   $L_3 = (ab)^*$
   $L_4 = \{a^i b^i \ ; \ i \geq 0\}$
   $L_5 = \{\varepsilon, bb\}$.
   Jak vypadají následující jazyky?
   - $L_1 \cap L_3$
     $L_1 \cap L_4$
     $L_2 \cap L_5$
   - $L_2 - L_1$
     $L_1 - L_3$
     $L_5 - L_3$
   - $\overline{L_1}$
     $L_1^+, \quad L_3^+$
     L_5^+
   - $L_1^R, \quad L_3^R$
     L_2^R
     L_4^R
3. Je dán homomorfismus h definovaný následovně: $h(a) = 01, \ h(b) = 10$.
   - Určete $h(L_1)$, pokud $L_1 = \{\varepsilon, aa, ab, aab\}$
   - Určete $h(L_2)$, pokud $L_2 = (ab)^*$
   - Určete $h(L_3)$, pokud $L_3 = a^*abb^*$

1.4.3 Prefixy a postfixy slov

Následující operace nám umožňují pracovat s prefixy (predponami) a postfixy (příponami) slov. Levý a pravý derivát určují množinu slov daného jazyka, jejichž prefixem či postfixem je dané slovo, levý a pravý kvocient používají místo jednoho prefixu či postfixu celou množinu (tedy jiný jazyk).

Levý a pravý derivát: Levý derivát jazyka L podle slova x je množina všech slov takových, že pokud je k nim zleva operací zřetězení přidáno slovo x, patří do jazyka L.

$$\delta_x^l(L) = \{w \in \Sigma^* \ ; \ x \cdot w \in L\} \tag{1.17}$$

Pravý derivát jazyka L podle slova x je množina všech slov takových, že pokud je k nim zprava operací zřetězení přidáno slovo x, patří do jazyka L.

$$\delta_x^r(L) = \{w \in \Sigma^* \ ; \ w \cdot x \in L\} \tag{1.18}$$

Příklad 1.14

Pokud $L = \{(ab)^i(ba)^i \ ; \ i \geq 2\}$, pak

- $\delta_{ab}^l(L) = \{(ab)^{i-1}(ba)^i \ ; \ i \geq 2\}$
- $\delta_{baba}^r(L) = \{(ab)^i(ba)^{i-2} \ ; \ i \geq 2\} = \{(ab)^{i+2}(ba)^i \ ; \ i \geq 0\}$
- $\delta_{(ba)^4}^r(L) = \{(ab)^{i+4}(ba)^i \ ; \ i \geq 0\}$ (všimněte si, že některá slova jazyka L do výsledného jazyka nebyla vůbec zařazena)

Levý a pravý kvocient: Jedná se o zobecnění levého a pravého derivátu. Levý kvocient jazyka L_1 vzhledem k jazyku L_2 je sjednocením levých derivátů jazyka L_1 vzhledem ke všem slovům jazyka L_2. Podobně pravý kvocient jazyka L_1 vzhledem k jazyku L_2 je sjednocením pravých derivátů jazyka L_1 vzhledem ke všem slovům jazyka L_2.

$$\begin{aligned} L_2 \backslash L_1 &= \{w \in \Sigma^* \ ; \ \textit{existuje } x \in L_2 \textit{ tak, že } x \cdot w \in L_1\} \\ &= \{w \in \delta_x^l(L_1) \ ; \ x \in L_2\} \qquad (1.19) \\ L_1 / L_2 &= \{w \in \Sigma^* \ ; \ \textit{existuje } x \in L_2 \textit{ tak, že } w \cdot x \in L_1\} \\ &= \{w \in \delta_x^r(L_1) \ ; \ x \in L_2\} \qquad (1.20) \end{aligned}$$

Mnemotechnická pomůcka: z toho jazyka, který je „níže", bereme prefix (tj. slovo x z výše uvedených vzorců). Ten jazyk, který je „výše", bude zleva nebo zprava „osekán".

Příklad 1.15

Jsou dány tyto jazyky:

$L_1 = \{ab, aab, ac\}$ $\quad$ $L_3 = a^*$ $\quad$ $L_5 = (ab)^*$ $\quad$ $L_7 = \{\varepsilon, abc, ab\}$

$L_2 = \{a^i bc^i \ ; \ i \geq 0\}$ $\quad$ $L_4 = a^*b^*$ $\quad$ $L_6 = (ab)^+$

Stanovíme tyto levé a pravé kvocienty:

- $L_1 \backslash L_2 = \{c, cc\}$ (všimněte si, že opravdu jde o konečný jazyk, třebaže L_2 je nekonečný)
- $L_2 \backslash L_1 = \emptyset$ (protože žádné slovo z jazyka L_2 není prefixem žádného slova jazyka L_1; pozor, nejkratší slovo z jazyka L_2 je b, nikoliv ε)
- $L_3 \backslash L_2 = \{a^m bc^i \ ; \ 0 \leq m \leq i\}$ (postup: vezmeme slovo z L_2, například a^4bc^4, a použijeme jako prefixy postupně slova $\varepsilon, a, a^2, a^3, a^4$ z jazyka L_3 – všechna do délky počtu symbolů a ve slově z jazyka L_2, pak další slovo, atd.)
- $L_4 \backslash L_2 = L_2 \cup \{a^m bc^i \ ; \ 0 \leq m \leq i\} \cup \{c^i \ ; \ i \geq 0\}$ (první část: z L_4 zvolíme ε; druhá část: z L_4 zvolíme slova a^*; třetí část: z L_4 zvolíme a^*b)
  protože $L_2 \subset \{a^m bc^i \ ; \ 0 \leq m \leq i\}$, výsledek můžeme zjednodušit; jak?
- $L_5 \backslash L_2 = L_2 \cup \{c\}$ (pokud $x = \varepsilon$, pak získáme L_2; pokud $x = \{ab\}$, získáme $\{c\}$; další slova z L_4 nelze použít)
- $L_6 \backslash L_2 = \{c\}$
- $L_7 \backslash L_2 = L_2 \cup \{\varepsilon, c\}$
- $L_1 / L_5 = L_1 \cup \{\varepsilon, a\}$ (pozor, teď už počítáme pravý kvocient)
- $L_1 / L_6 = \{\varepsilon, a\}$

1.5 K vlastnostem jazyků

Příklad 1.16

Ještě si trochu procvičíme operace nad jazyky.

- $L_1 = \{a^i b^j \ ; \ i,j \geq 0\}$
  $L_2 = \{w \in \{a,b\}^* \ ; \ |w|_a = |w|_b\}$
  $L_1 \cap L_2 = \{a^n b^n \ ; \ n \geq 0\}$
  $L_1^R = \{b^j a^i \ ; \ i,j \geq 0\}$
  $L_2^R = L_2$ (každé slovo po reverzi opět patří do původního jazyka a naopak)

- $L_1 = (ab)^*$
  $L_2 = (ba)^*$
  $L_1 \cap L_2 = \{\varepsilon\}$
  $L_1 \cup L_2 = (ab)^* + (ba)^*$
  $L_1^R = L_2$, $L_2^R = L_1$

- $L_1 = a^*$
  $L_2 = b^*$
  $L_1 \cap L_2 = \{\varepsilon\}$
  $L_1 \cup L_2 = a^* + b^*$ (pozor, ne $(a+b)^*$, ve slovech budou vždy buď jen symboly a nebo jen symboly b)
  $\{a,b\}^* - L_1 = \{w \in \{a,b\}^* \ ; \ |w|_b \geq 1\}$ (proč?)
  $L_1^R = L_1$, $L_2^R = L_2$

- $L_1 = \{a^n c^n b^n \ ; \ n \geq 0\}$
  $L_2 = \left\{wcw^R \ ; \ w \in \{a,b\}^*\right\}$
  $L_1 \cap L_2 = \{acb\}$
  $L_1 - L_2 = \{a^n c^n b^n \ ; \ n \geq 2\} \cup \{\varepsilon\} = L_1 - \{abc\}$ (všimněte si minima pro hodnotu n)
  $L_1^R = \{b^n c^n a^n \ ; \ n \geq 0\}$
  $L_2^R = L_2$

- $L_1 = a^*$
  $L_2 = \{a^{2^n} \ ; \ n \geq 0\}$
  $L_1 \cup L_2 = L_1$ (protože $L_2 \subset L_1$)
  $L_1 \cap L_2 = L_2$ (z téhož důvodu)
  $L_2 - L_1 = \emptyset$ (ale naopak by to neplatilo, operace $-$ není komutativní)

Z příkladu 1.16 je zřejmé, že

$$a^* + b^* \neq (a+b)^*$$

Také bychom si měli dávat pozor na to, že operace zřetězení není komutativní (nicméně asociativní je). Naproti tomu operace $+$ má stejné vlastnosti jako operace sjednocení množin, včetně vlastnosti komutativity a asociativity.

Příklad 1.17

Najdeme jazyk L, který *splňuje* danou vlastnost:

- $L = L \cdot L$

  například: $L = \{a, b\}^*$, protože platí $\{a, b\}^* \cdot \{a, b\}^* = \{a, b\}^*$
  jiné možnosti: $L = a^*$, $L = (ab)^*$, $L = \{\varepsilon\}$, $L = \{w \in \{a, b\}^* \ ; \ |w|_a = |w|_b\}$, atd.
  špatně by bylo například $L = \{a\}$, $L = \{ab, bc\}$, $L = \{a^{2^n}\}$, $L = \{w \in \{a, b\}^* \ ; \ |w|_a = |w|_b{+}1\}$

- $L = L^*$

  například: $L = a^*$, protože jak L, tak i L^* obsahují právě všechny řetězce nad abecedou $\{a\}$
  jiné možnosti: $L = \{a, b\}^*$, $L = \{w \in \{a, b\}^* \ ; \ |w|_a = |w|_b\}$, $L = \{\varepsilon\}$, atd.
  špatně by bylo například $L = \{ab, aa\}$, $L = \{a^n b^n \ ; \ n \geq 0\}$, $L = a^+$

- $L^* = L^+$

  například: $L = \{a^n b^n \ ; \ n \geq 0\}$, protože L^* i L^+ obsahují tatáž slova včetně ε
  jiné možnosti: $L = a^*$, $L = \{a, b\}^*$, $L = \{w \in \{a, b\}^* \ ; \ |w|_a = |w|_b\}$, atd. (je třeba, aby $\varepsilon \in L$)
  špatně by bylo například $L = \{a^n b^n \ ; \ n \geq 1\}$, $L = \{w \in \{a, b\}^* \ ; \ |w|_a < |w|_b\}$

- $L \cap L^R = \emptyset$

  například: $L = \{a^n b^n \ ; \ n \geq 1\}$ (v reverzi je opačné pořadí a a b, v jazycích není slovo ε)
  jiné možnosti: $L = (ab)^* ab$, $L = \{abc\}$
  špatně by bylo například $L = (ab)^*$, $L = \{a, b\}^*$, $L = \{w \in \{a, b\}^* \ ; \ |w|_a = |w|_b\}$

- $L^* = L \cup \{\varepsilon\}$

  například: $L = \{w \in \{a, b\}^* \ ; \ |w|_a < |w|_b\}$, zde nevadí, že v L není ε
  jiné možnosti: $L = a^*$, $L = \{a, b\}^*$, $L = \{a^n b^n \ ; \ n \geq 1\}$, atd.
  špatně by bylo například $L = \{abc\}$, $L = \{a^n b^n \ ; \ n \geq 1\}$

Příklad 1.18

Najdeme jazyk L, který *nesplňuje* danou vlastnost:

- $L = L^*$

  například: $L = \{abc, ba\}$, protože $L^* = \{\varepsilon, abc, ba, abcabc, abcba, baba, baabc, \ldots\}$
  špatně: $L = a^*$, protože $L^* = (a^*)^* = a^*$ (pozor, v zadání je „nesplňuje“)

- $L \cap L^R = \emptyset$

  například: $L = \{a^n b^n \ ; \ n \geq 0\}$, protože L i L^* obsahují slovo ε
  špatně: $L = \{a^n b^n \ ; \ n \geq 1\}$, zde opravdu L a ^{R} nemají žádné společné slovo, jsou disjunktní

Úkoly

1. Ke každému vztahu napište jazyk L_a, který danou vlastnost splňuje, a jazyk L_n, který danou vlastnost nesplňuje:
   - $L = L^R$
   - $L \cap L^* = \{\varepsilon\}$
   - $L = \{a, b\}^* - L$
2. Ke všem vlastnostem z příkladů 1.16 a 1.18 určete, zda je následující jazyky splňují:
   - $L_1 = \{a, b, c\}^*$
   - $L_2 = \left\{wcw^R \ ; \ w \in \{a, b\}^*\right\}$
   - $L_3 = \left\{ww^R \ ; \ w \in \{a, b\}^*\right\}$
   - $L_4 = \left\{a^{2^n} \ ; \ n \geq 0\right\}$
3. Je dán jazyk $L = (ab)^*b(ba)^*$ a následující homomorfismy:

   $h_1(a) = 01$ $\qquad$ $h_2(a) = 0$
   $h_1(b) = 10$ $\qquad$ $h_2(b) = 00$

   Zjistěte jazyky $h_1(L)$ a $h_2(L)$.
4. Připomeňte si, jak je u algebraických struktur definována vlastnost komutativity, asociativity a distributivity. Zjistěte, zda je operace zřetězení distributivní vzhledem k operaci $+$ a naopak.
5. Je dán jazyk $L = \{\text{vypsat}, \text{vylít}, \text{vymazat}\}$. Vytvořte:

   - $\{\text{vy}\}\backslash^L = \delta^l_{\text{vy}} =$
   - $\{\text{pře}\} \cdot \left(\{\text{vy}\}\backslash^L\right) =$

Kapitola 2

Konečné automaty

Konečný automat je jednoduchý matematický model, který se skládá z řídicí jednotky (včetně stavu), vstupní pásky (vstupní soubor či jiná datová struktura) a čtecí hlavy. V každém kroku tento automat načte jeden symbol ze vstupu a činnost v tomto kroku je určena podle tohoto načteného symbolu a momentálního stavu, spočívá ve změně stavu.

2.1 Vytváříme konečný automat

Konečný automat lze zapsat celkem třemi způsoby:

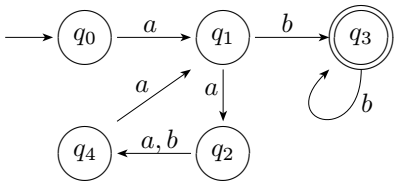
1. použitím plné specifikace s δ-funkcí
2. stavovým diagramem
3. tabulkou přechodů

Ukážeme si všechny tři způsoby. Exaktní postup vytvoření konečného automatu si ukážeme později (kapitola 2.8 na straně 44), ale abychom mohli sestrojovat už teď alespoň menší konečné automaty, podíváme se na zjednodušený postup.

Příklad 2.1

Vytvoříme konečný automat pro jazyk $L = a\,(a\,(a+b)\,a)^*bb^* = a \cdot (a \cdot (a+b) \cdot a)^* \cdot b \cdot b^*$ ve všech třech reprezentacích.

U menších automatů je nejjednodušší vytvořit stavový diagram:



δ-funkce včetně plné specifikace:

$\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4\},\ \{a, b\},\ \delta,\ q_0,\ \{q_3\})$

$$\delta(q_0, a) = q_1 \qquad \delta(q_2, b) = q_4$$
$$\delta(q_1, a) = q_2 \qquad \delta(q_4, a) = q_1$$
$$\delta(q_1, b) = q_3 \qquad \delta(q_3, b) = q_3$$
$$\delta(q_2, a) = q_4$$

Tabulka přechodů:

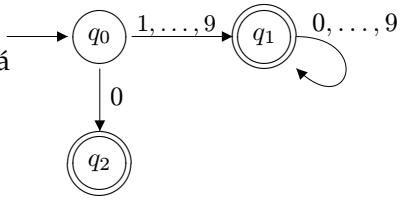
	a	b
$\rightarrow q_0$	q_1	
q_1	q_2	q_3
q_2	q_4	q_4
$\leftarrow q_3$		q_3
q_4	q_1	

Zatímco u stavového diagramu a tabulky přechodů nemusíme uvádět plnou specifikaci – řetězec $\mathcal{A} = (\{q_0, q_1, q_2, q_3\},\ \{a, b\},\ \delta,\ q_0,\ \{q_3\})$, při použití δ-funkce je plná specifikace nutná. Bez ní bychom nepoznali, který stav je počáteční a které stavy jsou koncové.

Příklad 2.2

Sestrojíme konečný automat rozpoznávající celá čísla. Používáme číslice $0 \ldots 9$, číslo se musí skládat z alespoň jedné číslice. Víceciferná čísla nesmí začínat nulou.

Vytvoříme automat se třemi stavy. Oddělíme zpracování jednociferného čísla „0“ od ostatních, která nulou nesmí začínat.



Úkoly

1. Pro následujících šest jazyků sestrojte konečný automat ve všech třech reprezentacích:
   (a) a^*b
   (b) $\{0,1\}^*11$
   (c) $11\{0,1\}^*$
   (d) $(ab)^*ba$
   (e) $\{ab^i \ ;\ i \geq 1\} \cup \{ba^i \ ;\ i \geq 0\}$
   (f) $a^* \cdot \{a, bc\}$

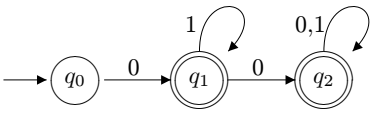
2. Podle zadání vytvořte zbylé dvě reprezentace daného konečného automatu.
   (a) Podle δ-funkce vytvořte stavový diagram a tabulku přechodů:
   $\mathcal{A}_1 = (\{q_0, q_1, q_2\},\ \{a, b, c\},\ \delta,\ q_0,\ \{q_1, q_2\})$
   $\delta(q_0, a) = q_1$ $\quad$ $\delta(q_1, c) = q_1$
   $\delta(q_0, b) = q_2$ $\quad$ $\delta(q_2, c) = q_1$
   (b) Podle každé tabulky přechodů vytvořte stavový diagram a δ-funkci s plnou reprezentací automatu:

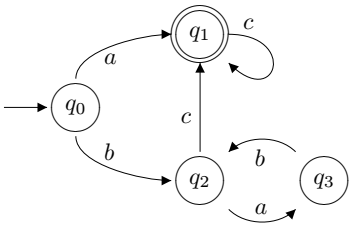
	0	1
$\rightarrow$ A	B	C
B	B	C
$\leftarrow$ C	D	
$\leftarrow$ D		D

	a	b
$\leftrightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$\leftarrow q_2$	q_3	
$\leftarrow q_3$		

Všimněte si, že podle druhé tabulky je počáteční stav zároveň koncový (šipka vede oběma směry).

(c) Podle každého stavového diagramu vytvořte tabulku přechodů a δ-funkci s plnou reprezentací automatu:





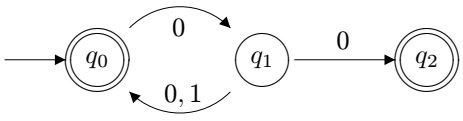
3. Sestrojte konečný automat (stavový diagram) rozpoznávající reálná čísla. Celá část je podle zadání v předchozím příkladu, následuje desetinná čárka a pak opět sekvence číslic (alespoň jedna, může to být i číslice 0).
4. Podle stavového diagramu pro reálná čísla vytvořte tabulku přechodů.

2.2 Nedeterministický konečný automat

Nedeterministický konečný automat je takový automat, kde v alespoň jednom stavu lze na některý signál reagovat více různými způsoby. Protože se tento typ automatu špatně programuje (je těžké vložit do instrukcí náhodnost – zvolit jednu z nabízených cest, a to pokud možno tak, aby vedla „správným směrem“, do koncového stavu), může se hodit postup, jak nedeterministický automat převést na deterministický se zachováním rozpoznávaného jazyka.

Příklad 2.3

Zadaný nedeterministický automat převedeme na deterministický (je dán stavový diagram a tabulka přechodů).



	0	1
$\leftrightarrow q_0$	q_1	
q_1	q_0, q_2	q_0
$\leftarrow q_2$		

Tento převod je nejjednodušší na tabulce přechodů. V některých buňkách je více než jedna položka, proto obsah buněk budeme chápat jako množiny. Vytváříme tedy nový konečný (deterministický) automat takový, že jeho stavy jsou množinami stavů původního automatu.

Novou tabulku přechodů tedy vytvoříme z původní tabulky tak, že nejdřív jak obsah buněk, tak i stavy v označení řádků uzavřeme do množinových závorek. Tím ale v některých buňkách (zde v jedné) dostaneme stav, který není v označení žádného řádku. To napravíme jednoduše tak, že přidáme nový řádek tabulky s tímto označením a obsah buněk zjistíme sjednocením řádků původní tabulky označených prvky množiny, se kterou právě pracujeme:

...			
$\{A\}$	...	$\{B,C\}$	...
...			
$\{B\}$	$\{D,E\}$	...	$\{G\}$
$\{C\}$	$\{F\}$	...	$\{H,I\}$
...			

$\Rightarrow$

...			
$\{A\}$	...	$\{B,C\}$	...
...			
$\{B\}$	$\{D,E\}$	...	$\{G,H\}$
$\{C\}$	$\{F\}$	...	$\{H,I\}$
...			
$\{B,C\}$	$\{D,E,F\}$	...	$\{G,H,I\}$

Může se stát, že sjednocením množin v buňkách opět vznikne množina, která se nenachází v označení žádného řádku. Pak opět vytvoříme nový řádek a pro buňky použijeme operaci sjednocení množin. Postup je tedy rekurzivní.

Takže k příkladu:

	0	1
$\leftrightarrow \{q_0\}$	$\{q_1\}$	
$\{q_1\}$	$\{q_0,q_2\}$	$\{q_0\}$
$\leftarrow \{q_2\}$		

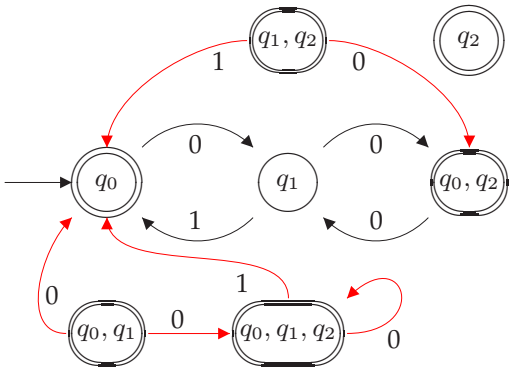
$\Rightarrow$

	0	1
$\leftrightarrow \{q_0\}$	$\{q_1\}$	$\emptyset$
$\{q_1\}$	$\{q_0,q_2\}$	$\{q_0\}$
$\leftarrow \{q_2\}$	$\emptyset$	$\emptyset$
$\leftarrow \{q_0,q_2\}$	$\{q_1\}\cup\emptyset=\{q_1\}$	$\emptyset\cup\emptyset=\emptyset$

Vlastnost „být koncovým stavem“ se také dědí v rámci sjednocení – pokud alespoň jeden z prvků množiny stavů je v původním automatu koncovým stavem, stává se koncovým stavem i celá tato množina.

Uvedený postup je vlastně zkrácený. Ve skutečnosti bychom měli postupovat tak, že bychom jako stavy použili všechny možné kombinace stavů a opět operaci sjednocení množin:

	0	1
$\leftrightarrow \{q_0\}$	$\{q_1\}$	$\emptyset$
$\{q_1\}$	$\{q_0,q_2\}$	$\{q_0\}$
$\leftarrow \{q_2\}$	$\emptyset$	$\emptyset$
$\leftarrow \{q_0,q_1\}$	$\{q_0,q_1,q_2\}$	$\{q_0\}$
$\leftarrow \{q_0,q_2\}$	$\{q_1\}$	$\emptyset$
$\leftarrow \{q_1,q_2\}$	$\{q_0,q_2\}$	$\{q_0\}$
$\leftarrow \{q_0,q_1,q_2\}$	$\{q_0,q_1,q_2\}$	$\{q_0\}$



Ve stavovém diagramu můžeme vidět, že stavy, které jsou oproti předchozímu postupu navíc, jsou nedosažitelné z počátečního stavu a tedy se nenacházejí na žádné cestě při zpracování slov jazyka. Proto je vlastně můžeme bez újmy na obecnosti z automatu odstranit.

Úkoly

Uvedené nedeterministické konečné automaty reprezentované tabulkami převeďte na ekvivalentní deterministické. Ke každému vytvořte stavový diagram původního nedeterministického automatu i vytvořeného deterministického a porovnejte.

	a	b
$\leftrightarrow$ X	Y	
Y	Z	
Z	X	X, Z

	0	1
$\rightarrow$ q_0	q_0, q_1	q_0, q_1
$\leftarrow$ q_1		q_1, q_2
$\leftarrow$ q_2		

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

2.3 Totální automat

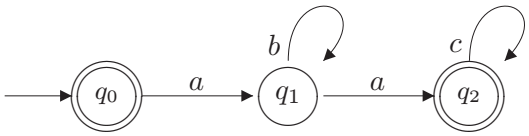
V totálním (úplném) automatu lze v každém stavu reagovat na jakýkoliv symbol. Při převodu (netotálního) automatu na totální vytvoříme nový stav („odpadkový koš“, chybový stav), do kterého přesměrujeme všechny chybějící přechody. Nesmíme zapomenout, že požadavek možnosti reakce na kterýkoliv symbol se vztahuje také na tento nově přidaný stav.

Postup převodu na totální automat lze použít pouze na deterministický konečný automat, proto u nedeterministického automatu je prvním krokem vždy převod na deterministický.

Příklad 2.4

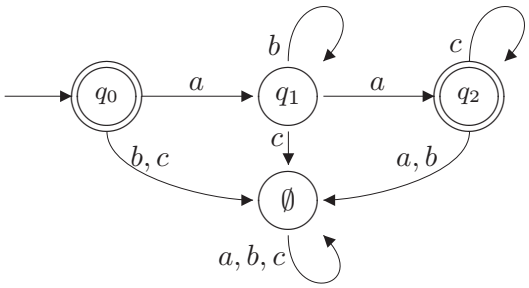
K zadanému deterministickému konečnému automatu vytvoříme ekvivalentní totální automat:

	a	b	c
$\leftrightarrow$ q_0	q_1		
q_1	q_2	q_1	
$\leftarrow$ q_2			q_2



Tento automat určitě není totální – například ve stavu q_0 nelze reagovat hned na dva signály. Vytvoříme nový stav, označíme jej symbolem $\emptyset$ a přesměrujeme do tohoto stavu všechny chybějící přechody:

	a	b	c
$\leftrightarrow$ q_0	q_1	$\emptyset$	$\emptyset$
q_1	q_2	q_1	$\emptyset$
$\leftarrow$ q_2	$\emptyset$	$\emptyset$	q_2
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Příklad 2.5

V příkladu 2.3 jsme k nedeterministickému automatu vytvořili ekvivalentní deterministický. Tento deterministický automat nyní zúplníme (převedeme na totální). Stav $\{q_2\}$ není dosažitelný z počátečního stavu, proto jej také vypustíme.

Abychom trochu zjednodušili značení, budeme místo množin $\{\ldots\}$ používat velká písmena, provedeme přeznačení:

Původní:

	0	1
$\leftrightarrow \{q_0\}$	$\{q_1\}$	$\emptyset$
$\{q_1\}$	$\{q_0, q_2\}$	$\{q_0\}$
$\leftarrow \{q_0, q_2\}$	$\{q_1\}$	$\emptyset$

$\Rightarrow$

Přeznačený:

	0	1
$\leftrightarrow A$	B	$\emptyset$
B	C	A
$\leftarrow C$	B	$\emptyset$

$\Rightarrow$

Totální:

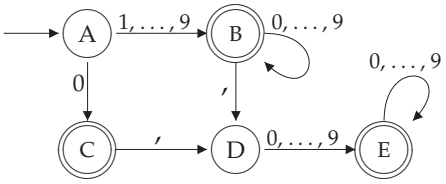
	0	1
$\leftrightarrow A$	B	$\emptyset$
B	C	A
$\leftarrow C$	B	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

Stavové diagramy přeznačeného a zúplněného automatu:

Úkoly

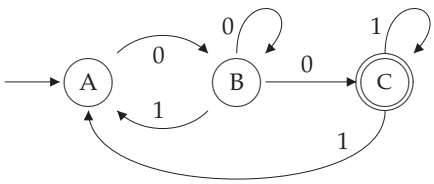
1. Všechny konečné automaty, které jste v úkolu č. 2.2.2 na straně 21 převedli na deterministické, zúplňte (převeďte na totální).
2. Následující konečný automat převeďte na totální a nakreslete jeho stavový diagram (signál v záhlaví posledního sloupce je desetinná čárka):

	0	1, . . . , 9	,
$\rightarrow$ A	C	B	
$\leftarrow$ B	B	B	D
$\leftarrow$ C			D
D	E	E	
$\leftarrow$ E	E	E	



3. Následující (nedeterministický!) konečný automat zúplňte.

	0	1
$\rightarrow$ A	B	
B	B, C	A
$\leftarrow$ C		A, C



2.4 Konečné jazyky

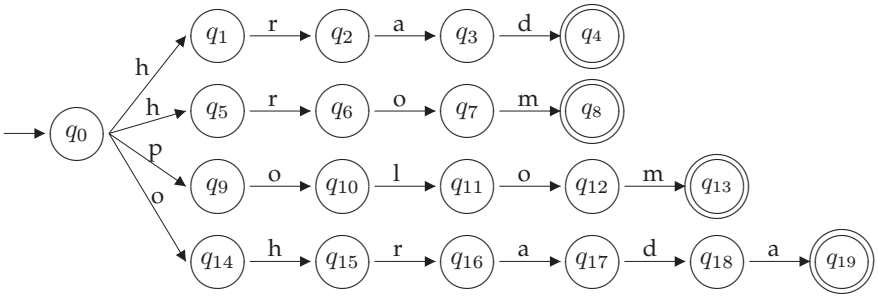
Všechny konečné jazyky jsou regulární, proto pro ně dokážeme sestrojit konečný automat.

U konečného automatu pro konečný jazyk bývá obvyklý požadavek na rozlišitelnost načítaných slov podle koncového stavu, tedy pro každé slovo jazyka by měl existovat samostatný koncový stav. Pak bez nutnosti porovnávání vstupu s jednotlivými slovy jazyka snadno zjistíme, které slovo bylo načteno – podle koncového stavu, ve kterém skončil výpočet.

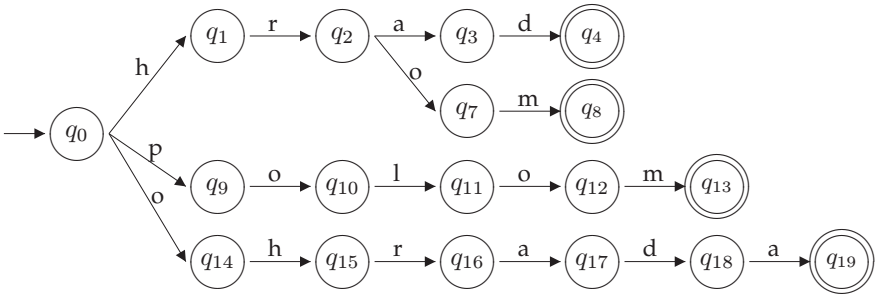
Postup je jednoduchý – pro každé slovo jazyka vytvoříme „větev" ve stavovém diagramu. Jestliže chceme automat deterministický (opět jde o obvyklý požadavek, pokud tento postup používáme při programování), stačí sloučit počátky těch větví, které stejně začínají (konce větví nesmíme sloučit, nebylo by možné rozpoznat slova jazyka podle koncových stavů!).

Příklad 2.6

Podle zadaného konečného jazyka vytvoříme konečný automat.
$L = \{\text{hrad, hrom, polom, ohrada}\}$



Dále chceme, aby automat byl deterministický se zachováním možnosti rozlišit rozpoznávaná slova podle koncového stavu. První dvě slova jazyka začínají stejným podřetězcem, tedy začátky prvních dvou větví sloučíme:



Kdybychom netrvali na podmínce odlišení slov koncovými stavy, bylo by možné sloučit všechny koncové stavy v jeden, a také stavy q_7 a q_{12}.

V konečném automatu pro konečný jazyk nesmí (a ani nemůže) být žádná smyčka (cyklus) – kdyby byla, pak by bylo možné tuto smyčku jakýkolivpočetkrát zopakovat a rozpoznávaný jazyk by byl nekonečný.

Úkoly

1. Vytvořte konečný automat pro následující jazyky (deterministický, a to tak, aby pro každé slovo jazyka existoval jeden konečný stav):
   (a) $L_a = \{$strom, stroj, výstroj$\}$
   (b) $L_b = \{$if, else, elif$\}$
   (c) $L_c = \{$read, write, writeall, matrix$\}$
   (d) $L_d = \{$delfín, ryba, velryba$\}$

   Poznámka: V případě (c) bude koncový stav větve pro druhé slovo součástí větve pro třetí slovo (pokud vytvoříme deterministický automat). To je v pořádku, vlastnost rozpoznávání podle koncového stavu zůstává zachována.
2. Vytvořte konečný automat (stavový diagram), který bude rozpoznávat všechna slova nad abecedou $\Sigma = \{a, b, c\}$, jejichž délka je
   (a) právě 3 znaky,
   (b) nejvýše 3 znaky (tj. 0, 1, 2 nebo 3 znaky).

   Nápověda: Jde o konečné jazyky. Víme, že v automatu pro konečný jazyk nesmí být cyklus, a také víme, že přechody ve stavovém diagramu mohou být označeny více než jedním znakem.

2.5 Odstranění nepotřebných stavů (redukce)

Nepotřebné stavy jsou stavy, které nejsou použity při žádném úspěšném výpočtu (tj. končícím v koncovém stavu). Jedná se o stavy

- *nedosažitelné* – neexistuje k nim cesta z počátečního stavu,
- *nadbytečné* – neexistuje cesta z tohoto stavu do jakéhokoliv koncového stavu.

S odstraňováním (lépe řečeno ignorováním, neuvedením či vypuštěním) prvního typu nepotřebných stavů – nedosažitelných – jsme se setkali už u zkráceného postupu pro převod nedeterministického automatu na deterministický. Pokud nové řádky tabulky přechodů tvoříme jen pro takové množiny původních stavů, které se již vyskytly v některé buňce, většinu nedosažitelných stavů (ale ne vždy všechny) automaticky odstraňujeme.

Když chceme odstranit nepotřebné stavy, *vždy* začínáme nedosažitelnými stavy a až potom odstraníme nadbytečné. V obou případech postupujeme rekurzívně, a to buď podle stavového diagramu nebo podle tabulky přechodů.

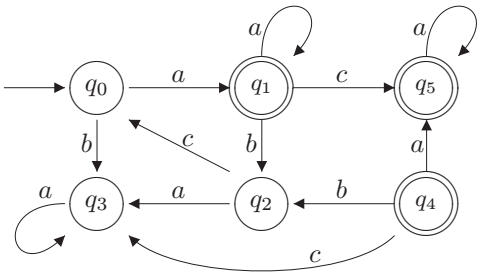
- *nedosažitelné stavy*: jako bázi (základní množinu) zvolíme $S_0 = \{q_0\}$ (obsahuje pouze počáteční stav), další prvky přidáváme „ve směru šipek“ v stavovém diagramu, resp. v tabulce přechodů ve směru označení řádku $\rightarrow$ obsah buněk na řádku, postupně vytváříme množinu všech stavů, do kterých vede cesta z počátečního stavu o délce max. 0, 1, ..., n kroků (toto číslo je dolní index u označení vytvářené množiny S_i);

- *nadbytečné stavy*: jako bázi naopak zvolíme množinu koncových stavů – $E_0 = F$, další prvky přidáváme „proti směru šipek“ v stavovém diagramu, resp. ve směru obsah některé buňky tabulky přechodů $\rightarrow$ označení řádku, vytváříme množinu všech stavů, ze kterých vede cesta do některého koncového stavu o délce max. 0, 1, . . . , n kroků.

Příklad 2.7

V zadaném konečném automatu odstraníme nedosažitelné a nadbytečné stavy.

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



Odstraníme nedosažitelné stavy (do kterých neexistuje cesta z počátečního stavu):

$S_0 = \{q_0\}$

$S_1 = \{q_0\} \cup \{q_1, q_3\} = \{q_0, q_1, q_3\}$ (ze stavu q_0 vede přechod do q_1 a q_3)

$S_2 = \{q_0, q_1, q_3\} \cup \{q_5, q_2\} = \{q_0, q_1, q_3, q_5, q_2\}$

$S_3 = \{q_0, q_1, q_3, q_5, q_2\} = S_2$ (konec, v posledním kroku žádný stav do množiny nepřibyl)

V množině S_3 není stav q_4, je tedy nedosažitelný z počátečního stavu a můžeme ho z automatu odstranit. V každé z množin S_i, které jsme postupně vytvořili, najdeme všechny stavy, které jsou z počátečního stavu dosažitelné po nejvýše i krocích.

Dále zjistíme, ze kterých stavů nevede cesta do koncových stavů. Budeme pracovat již s automatem po odstranění stavu q_4.

$E_0 = F = \{q_1, q_5\}$

$E_1 = \{q_1, q_5\} \cup \{q_0\} = \{q_1, q_5, q_0\}$ (do stavů z E_0 vede přechod pouze ze stavu q_0)

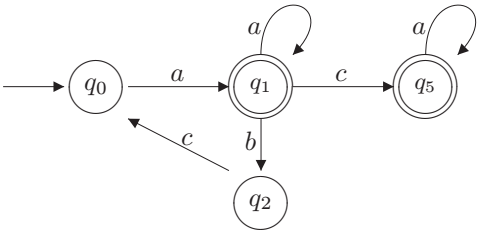
$E_2 = \{q_1, q_5, q_0, q_2\}$

$E_3 = \{q_1, q_5, q_0, q_2\} = E_2$

V množině E_3 není stav q_3, to znamená, že z tohoto stavu neexistuje žádná cesta do koncového a tedy když tento stav odstraníme, neovlivníme výpočet žádného slova, které automat rozpoznává. Odstraníme tento stav a také všechny přechody s ním přímo související.

Po odstranění stavů nepatřících do množin S_i a E_i dostáváme tento konečný automat:

	a	b	c
$\rightarrow q_0$	q_1		
$\leftarrow q_1$	q_1	q_2	q_5
q_2			q_0
$\leftarrow q_5$	q_5		



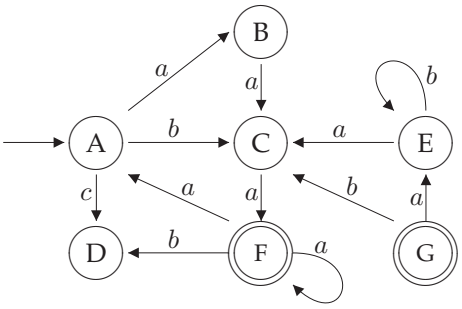
Úkoly

1. Podle zadání následujícího automatu vytvořte tabulku přechodů a stavový diagram. Potom odstraňte nepotřebné stavy. Takto upravený automat pak zúplňte (převeďte na totální automat).

   $\mathcal{A} = (\{A, B, C, D, E, F\},\ \{0, 1, 2\},\ \delta,\ A,\ \{A, C\})$ s funkcí δ:

   $\delta(A,0) = E$ $\quad \delta(B,0) = C$ $\quad \delta(C,0) = C$ $\quad \delta(E,1) = E$ $\quad \delta(F,1) = E$
   $\delta(A,1) = B$ $\quad \delta(B,1) = D$ $\quad \delta(C,1) = C$ $\quad \delta(E,2) = E$ $\quad \delta(F,2) = F$
   $\delta(A,2) = E$ $\quad \delta(B,2) = E$ $\quad \delta(D,2) = C$ $\quad \delta(F,0) = C$

2. K následujícímu konečnému automatu vytvořte zbývající dvě reprezentace – δ-funkci a tabulku přechodů. Potom odstraňte všechny nepotřebné stavy.



3. Podle následujících konečných automatů určených tabulkami přechodů sestrojte stavové diagramy a pak odstraňte všechny nepotřebné stavy.

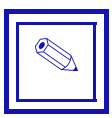
	a	b	c
$\leftrightarrow$ A	A	A	B
B	C		
C	C	B	
D	A	E	B
$\leftarrow$ E	B	C	C

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

4. Zamyslete se nad těmito případy: jak by vypadal jazyk upraveného automatu (bez nepotřebných stavů), kdyby do množin S_i nebyly zařazeny žádné koncové stavy? Jak by vypadal tento jazyk, kdyby do množin E_i nebyl zařazen počáteční stav?

2.6 Uzávěrové vlastnosti – regulární operace

Třída jazyků je množina všech jazyků s danou společnou vlastností. Také regulární jazyky tvoří třídu – třída regulárních jazyků je množina všech jazyků, pro které lze sestrojit konečný automat.



Víme, že na řetězce lze použít operaci zřetězení. Jazyk je množina řetězců, a tedy na jazyk můžeme použít různé množinové operace (sjednocení, průnik, doplněk – zrcadlení, substituci), a také operace odvozené z práce s řetězci a znaky (signály) – zřetězení, dále iteraci (operace „hvězdička“) a další.

Třída jazyků je uzavřena vzhledem k nějaké operaci, pokud po uplatnění operace na jazyky z této třídy vždy vznikne jazyk patřící do stejné třídy.

Postupně si ukážeme všechny základní operace, a to na konečných automatech. Pokud jde o binární operaci (uplatňuje se na dva jazyky), je obvykou podmínkou disjunktnost průniku množin stavů automatů (žádný stav existuje v obou automatech zároveň).

2.6.1 Sjednocení

V případě sjednocení využijeme celou definici původních automatů. Přidáme nový stav a z tohoto stavu povedeme přechody do všech stavů, do kterých vede přechod z počátečního stavu. Nový stav nám tedy simuluje použití počátečních stavů v původních automatech.

Postup si můžeme představit třeba tak, že vezmeme všechny přechody, které vedou z původních počátečních stavů, a *zkopírujeme* (ne přeneseme, ty původní musejí zůstat) jejich začátky (tj. u stavů, ze kterých vycházejí) k novému stavu.

Pokud některý z původních počátečních stavů patří i do množiny koncových stavů, pak také tuto vlastnost přidáme novému počátečnímu stavu.

Označme

- $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ je první automat,
- $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ je druhý automat,

pak automat rozpoznávající jazyk, který je sjednocením jazyků obou automatů, je
$\mathcal{A} = (Q_1 \cup Q_2 \cup \{s_0\},\ \Sigma_1 \cup \Sigma_2,\ \delta,\ s_0,\ F_1 \cup F_2)$, platí, že $s_0 \notin Q_1 \cup Q_2$, přechodová funkce:
$\delta(s_0, x) = \delta_1(q_1, x) \cup \delta_2(q_2, x), \quad x \in \Sigma_1 \cup \Sigma_2$

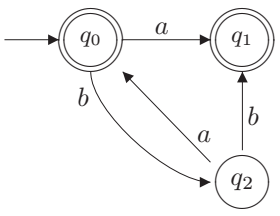
$$\delta(r, x) = \begin{cases} \delta_1(r, x) \ ; \ r \in Q_1, x \in \Sigma_1, \\ \delta_2(r, x) \ ; \ r \in Q_2, x \in \Sigma_2 \end{cases}$$

To znamená, že přechodovou funkci prakticky přejmeme z původních automatů, změna bude jen na začátku.

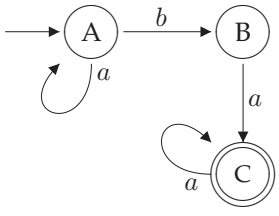
Příklad 2.8

Ukážeme si operaci sjednocení na následujících jazycích a automatech:
$L_1 = \{(ba)^i(a \cup bb) \ : \ i \geq 0\}$
$L_2 = \{a^i baa^j \ ; \ i, j \geq 0\}$

Konečné automaty pro tyto jazyky jsou následující:



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



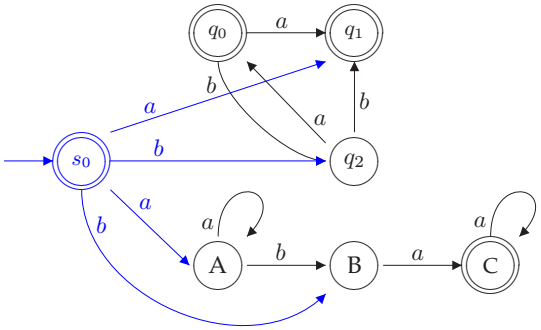
	a	b
$\rightarrow$ A	A	B
B	C	
$\leftarrow$ C	C	

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

$\mathcal{A}_2 = (\{A, B, C\}, \{a, b\}, \delta_2, A, \{C\})$
$\delta_2(A, a) = A$
$\delta_2(A, b) = B$
$\delta_2(B, a) = C$
$\delta_2(C, a) = C$

Přidáme nový stav s_0 a všechny přechody z tohoto stavu nasměrujeme a označíme stejně jako přechody z počátečních stavů původních automatů.

Postup je nejnázornější na stavovém diagramu, ale je jednoduchý také v tabulce přechodů – stačí obě tabulky shrnout do jedné a přidat nový řádek, jehož buňky budou sjednocením buněk na řádcích původních počátečních stavů a na příslušných sloupcích. Protože jeden z původních počátečních stavů (q_0) patří do množiny koncových stavů, také nový počáteční stav s_0 bude zároveň koncovým.



	a	b
$\leftrightarrow$ s_0	q_1, A	q_2, B
$\leftarrow$ q_0	q_1	q_2
$\leftarrow$ q_1		
q_2	q_0	q_1
A	A	B
B	C	
$\leftarrow$ C	C	

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

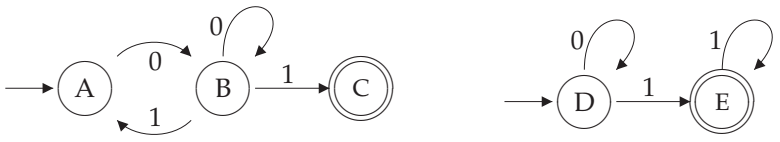
$\delta(s_0, a) = \delta_1(q_0, a) \cup \delta_2(A, a) = \{q_1, A\}$
$\delta(s_0, b) = \delta_1(q_0, b) \cup \delta_2(A, b) = \{q_2, B\}$
$\delta(q_0, a) = \{q_1\}$
$\delta(q_0, b) = \{q_2\}$
$\delta(q_2, a) = \{q_0\}$
$\delta(q_2, b) = \{q_1\}$
$\delta(A, a) = \{A\}$
$\delta(A, b) = \{B\}$
$\delta(B, a) = \{C\}$
$\delta(C, a) = \{C\}$

Je zřejmé, že $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

Pokud by žádný z původních počátečních stavů nepatřil do množiny koncových stavů, pak ani stav s_0 nesmíme zařadit do množiny koncových stavů! Znamenalo by to, že do výsledného jazyka nemá patřit prázdné slovo.

Úkoly

1. Použijte operaci sjednocení na následující konečné automaty:



2. Použijte operaci sjednocení na následující konečné automaty (q_3 zároveň koncový!):

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

	a	b	c
$\leftrightarrow q_3$	q_4	q_6	
q_4		q_5	
q_5			q_3
$\leftarrow q_6$			

3. Pro oba konečné automaty ze zadání předchozího příkladu a také pro výsledný automat vytvořte zbylé dva typy reprezentací – stavový diagram a δ-funkci včetně plné specifikace automatu.
4. Zamyslete se nad tím, jak by vypadalo sjednocení více než dvou konečných automatů.

2.6.2 Zřetězení

Zřetězení dvou jazyků vytvoříme tak, že ve výsledném jazyce jsou všechny možné řetězce, jejichž první část je kterékoliv slovo z prvního jazyka a druhá část zase kterékoliv slovo z druhého jazyka. Záleží na pořadí, části slova nemůžeme přehodit, řídí se pořadím zřetězovaných jazyků.

Příklad 2.9

Princip zřetězení jazyků si ukážeme na těchto konečných jazycích:
$L_1 = \{\varepsilon, abc, ba\}$
$L_2 = \{ddb, dc\}$

Zřetězením těchto dvou jazyků získáme jazyk $L = L_1 \cdot L_2$:
$L = \{\varepsilon \cdot ddb, \varepsilon \cdot dc, abc \cdot ddb, abc \cdot dc, ba \cdot ddb, ba \cdot dc\} = \{ddb, dc, abcccb, abcdc, baddb, badc\}$

Při zřetězení není třeba ve výsledném automatu vytvářet nový stav. Opět využijeme všechny stavy a přechody z původních automatů a budeme přidávat nové přechody, které je propojí.

Při ukončení výpočtu první části slova (tj. v prvním původním automatu) je třeba „plynule“ navázat na výpočet druhého původního automatu. Proto nově přidávané přechody budou vést z koncových stavů prvního automatu, jejich cílový stav (stav, do kterého bude směřovat šipka přechodu) a označení přejmeme (zkopírujeme) od přechodů z počátečního stavu druhého původního automatu.

Počátečním stavem výsledného automatu bude samozřejmě počáteční stav prvního původního automatu.

Do množiny koncových stavů zařadíme

- koncové stavy druhého původního automatu,
- pokud počáteční stav druhého automatu byl zároveň koncovým (to znamená, že do druhého jazyka patřilo slovo ε), pak do množiny koncových stavů výsledného automatu zařadíme také koncové stavy prvního původního automatu.

Označme

- $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ je první automat,
- $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ je druhý automat, zde je již důležité jejich pořadí,

pak automat rozpoznávající jazyk, který je zřetězením jazyků obou automatů, je $\mathcal{A} = (Q_1 \cup Q_2,\ \Sigma_1 \cup \Sigma_2,\ \delta,\ q_1,\ F)$, koncové stavy:

- $F = F_2$, pokud $\varepsilon \notin L(\mathcal{A}_2)$
- $F = F_1 \cup F_2$, pokud $\varepsilon \in L(\mathcal{A}_2)$

(tj. jestliže v jazyce druhého automatu bylo prázdné slovo, tedy počáteční stav patřil do množiny koncových stavů, znamená to, že do výsledného jazyka budou patřit i všechna slova rozpoznávaná prvním automatem – zřetězená s ε, lze skončit také ve stavech z F_1)

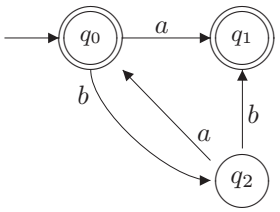
Přechodová funkce:

$$\delta(r,x) = \begin{cases} \delta_1(r,x) \ ; \ r \in Q_1 - F_1, x \in \Sigma_1, \\ \delta_1(r,x) \cup \delta_2(q_2,x) \ ; \ r \in F_1, x \in \Sigma_1 \cup \Sigma_2, \\ \delta_2(r,x) \ ; \ r \in Q_2, x \in \Sigma_2 \end{cases}$$

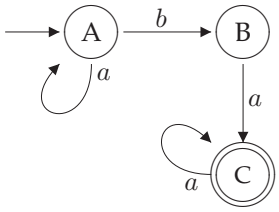
Znamená to, že v koncových stavech prvního původního automatu budeme (kromě existujících původních reakcí) reagovat stejně, jako v počátečním stavu druhého původního automatu (což je podle našeho značení q_2).

Příklad 2.10

Zřetězíme jazyky těchto konečných automatů:



$\mathcal{A}_1$	a	b
$\leftrightarrow\ q_0$	q_1	q_2
$\leftarrow\ q_1$		
q_2	q_0	q_1

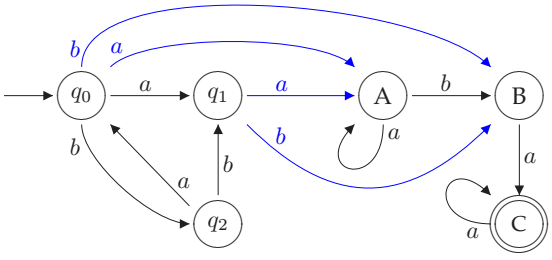


$\mathcal{A}_2$	a	b
$\rightarrow$ A	A	B
B	C	
$\leftarrow$ C	C	

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

$\mathcal{A}_2 = (\{A, B, C\}, \{a, b\}, \delta_2, A, \{C\})$
$\delta_2(A, a) = A$
$\delta_2(A, b) = B$
$\delta_2(B, a) = C$
$\delta_2(C, a) = C$

V automatu $\mathcal{A}_1$ (prvním) jsou dva koncové stavy. Z nich budou vést nové přechody ke stavům automatu $\mathcal{A}_2$ – ke všem stavům, do kterých vede přechod z počátečního stavu A.



	a	b
$\rightarrow\ q_0$	q_1, A	q_2, B
q_1	A	B
q_2	q_0	q_1
A	A	B
B	C	
$\leftarrow$ C	C	

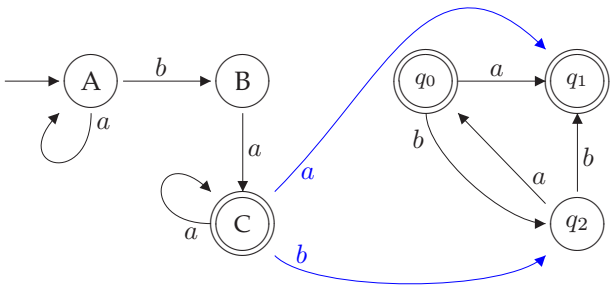
$\mathcal{A} = (\{q_0, q_1, q_2, A, B, C\}, \{a, b\}, \delta, \{C\})$

$\delta(q_0, a) = \{q_1, A\}$ $\delta(q_1, b) = \{B\}$ $\delta(A, a) = \{A\}$ $\delta(C, a) = \{C\}$
$\delta(q_0, b) = \{q_2, B\}$ $\delta(q_2, a) = \{q_0\}$ $\delta(A, b) = \{B\}$
$\delta(q_1, a) = \{A\}$ $\delta(q_2, b) = \{q_1\}$ $\delta(B, a) = \{C\}$

Platí $L(\mathcal{A}) = L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.

Příklad 2.11

Nyní použijeme stejné zadání jako v předchozím příkladu, jen obrátíme pořadí zřetězovaných automatů. Musíme brát v úvahu to, že počáteční stav automatu, který je teď druhý v pořadí, je zároveň koncovým stavem, což bude mít vliv také na množinu koncových stavů:



	a	b
$\rightarrow$ A	A	B
B	C	
$\leftarrow$ C	C, q_1	q_2
q_0	q_1	q_2
q_1		
q_2	q_0	q_1

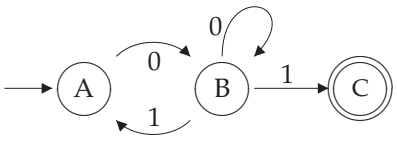
$\mathcal{A}' = (\{q_0, q_1, q_2, A, B, C\}, \{a, b\}, \delta, \{C, q_0, q_1\})$

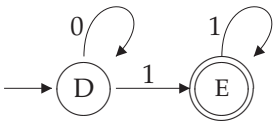
$\delta(A, a) = \{A\}$ $\delta(C, a) = \{C, q_1\}$ $\delta(q_0, b) = \{q_2\}$
$\delta(A, b) = \{B\}$ $\delta(C, b) = \{q_2\}$ $\delta(q_2, a) = \{q_0\}$
$\delta(B, a) = \{C\}$ $\delta(q_0, a) = \{q_1\}$ $\delta(q_2, b) = \{q_1\}$

Platí $L(\mathcal{A}') = L(\mathcal{A}_2) \cdot L(\mathcal{A}_1)$.

Úkoly

1. Proveďte operaci zřetězení jazyků následujících konečných automatů:





2. Proveďte operaci zřetězení jazyků následujících konečných automatů – nejdřív v pořadí podle zadání ($L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$) a potom v opačném pořadí ($L(\mathcal{A}_2) \cdot L(\mathcal{A}_1)$). Sestrojte také stavové diagramy.

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

$\mathcal{A}_2$	a	b	c
$\leftrightarrow\ q_3$	q_4	q_6	
q_4		q_5	
q_5			q_3
$\leftarrow\ q_6$			

2.6.3 Iterace (Kleeneho uzávěr)

Při iteraci zřetězujeme jazyk sám se sebou, a to $0\times$, $1\times$, $2\times$, ..., a pak všechny výsledné jazyky po zřetězení sjednotíme. Formálně to můžeme zapsat následovně:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

kde L^i je i-násobné *zřetězení* jazyka L sama se sebou (například pro $i = 3$ bylo $L^3 = L \cdot L \cdot L$). Zřetězení jsme již probírali.

Příklad 2.12

Princip iterace jazyka si ukážeme na tomto konečném jazyce:
$L_1 = \{abc, ba, cd\}$

Iterací získáme jazyk $L = L_1^*$:

$L = \{\varepsilon,$ ..... $0\times$
$abc, ba, cd,$ ..... $1\times$
$abcabc, abcba, abccd, baabc, baba, bacd, cdabc, cdba, cdcd,$ ..... $2\times$
$abcabcabc, abcabcba, abcabccd, abcbaabc, abcbaba, abcbacd, \ldots\}$ ..... $3\times, \ldots$

Výsledkem iterace je vždy nekonečný jazyk obsahující prázdné slovo ε, i kdyby původní jazyk L_1 byl konečný a i kdyby prázdné slovo neobsahoval.

Úprava konečného automatu při iteraci spočívá v těchto krocích:

1. vytvoříme nový počáteční stav, přechody z něj vedoucí určíme stejné jako ty, které vedou z původního počátečního stavu,
2. vytvoříme nové přechody umožňující „návrat“ z koncových stavů na počátek výpočtu (další slovo z jazyka),
3. nový počáteční stav zařadíme do množiny koncových stavů (tím zařadíme do jazyka slovo ε).

První bod je nevyhnutelný především tehdy, pokud v původním automatu vede některý přechod do počátečního stavu a zároveň tento stav nebyl v množině koncových stavů; je třeba zabránit tomu, aby zařazení počátečního stavu do množiny koncových stavů mělo za následek rozpoznávání takových slov, která do jazyka správně patřit nemají.

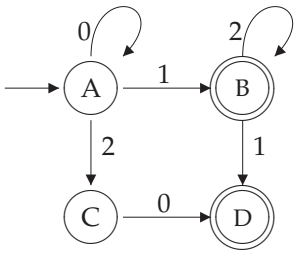
Označme $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ původní automat, pak automat rozpoznávající jazyk, který je iterací původního jazyka, je
$\mathcal{A} = (Q_1 \cup \{s_0\},\ \Sigma,\ \delta,\ s_0,\ F_1 \cup \{s_0\})$, přechodová funkce:

$$\delta(r,x) = \begin{cases} \delta_1(r,x)\ ;\ r \in Q_1 - F_1, x \in \Sigma, \\ \delta_1(r,x) \cup \delta_1(q_0,x)\ ;\ r \in F_1, x \in \Sigma, \\ \delta_1(q_0,x)\ ;\ r = s_0,\ x \in \Sigma \end{cases}$$

Nové přechody povedou z původních koncových stavů, a to do všech stavů, do kterých vede přechod z počátečního stavu. Princip je prakticky stejný jako u dříve probíraných operací, kopírujeme počátky přechodů od počátečního stavu se zachováním jejich cíle i označení (signálu).

Příklad 2.13

Vytvoříme iteraci jazyka následujícího automatu:
$\mathcal{A}_1 = (\{A,B,C,D\}, \{0,1,2\}, \delta_1, A, \{B,D\})$

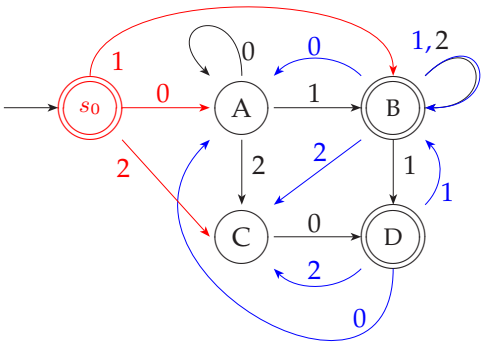


	0	1	2
→ A	A	B	C
← B		D	B
C	D		
← D			

$\delta_1(A,0) = A$
$\delta_1(A,1) = B$
$\delta_1(A,2) = C$
$\delta_1(B,1) = D$
$\delta_1(B,2) = B$
$\delta_1(C,0) = D$

Postup:

- Koncové stavy jsou dva. Z každého přidáme přechody do všech stavů, do kterých směřují přechody z počátečního stavu.
- Vytvoříme nový počáteční stav a přechody z něj vedoucí „zkopírujeme“ z původního počátečního stavu (tj. v novém počátečním stavu se automat bude chovat stejně jako v původním). Tento krok je sice u většiny automatů zbytečný, ale u některých nutný – zamezíme vzniku smyček přes počáteční stav generujících slova nepatřící do jazyka (takové smyčky jsme mohli vytvořit předchozím krokem).
- Potom (nový) počáteční stav označíme jako koncový, aby automat přijímal prázdné slovo ε.



	0	1	2
↔ s_0	A	B	C
A	A	B	C
← B	A	D, B	B, C
C	D		
← D	A	B	C

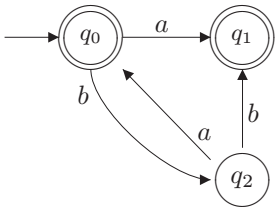
$\mathcal{A} = (\{s_0, A, B, C, D\}, \{0, 1, 2\}, \delta, s_0, \{s_0, B, D\})$

$\delta(s_0, 0) = A$	$\delta(A, 0) = A$	$\delta(B, 1) = D$	$\delta(B, 0) = A$	$\delta(D, 0) = A$
$\delta(s_0, 1) = B$	$\delta(A, 1) = B$	$\delta(B, 2) = B$	$\delta(B, 1) = B$	$\delta(D, 1) = B$
$\delta(s_0, 2) = C$	$\delta(A, 2) = C$	$\delta(C, 0) = D$	$\delta(B, 2) = C$	$\delta(D, 2) = C$

Kdyby stav A v původním automatu patřil do množiny koncových stavů, tak by samozřejmě koncovým stavem zůstal.

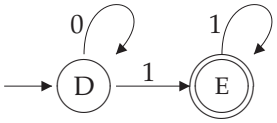
Úkoly

1. Zkonstruujte konečný automat jazyka, který je iterací jazyka následujícího automatu:



$\mathcal{A}_1$	a	b
$\leftrightarrow$ q_0	q_1	q_2
$\leftarrow$ q_1		
q_2	q_0	q_1

2. Zkonstruujte konečný automat jazyka, který je iterací jazyka následujícího automatu:



	0	1
$\rightarrow$ D	D	E
$\leftarrow$ E		E

Pro výsledný automat také vytvořte třetí typ reprezentace – δ-funkci s plnou specifikací.

2.7 Uzávěrové vlastnosti – další operace

2.7.1 Pozitivní iterace

Pozitivní iterace je podobná operace jako předchozí, ale zatímco iterace znamená řetězení $0\times,\ 1\times,\ 2\times,\ \ldots$, při pozitivní iteraci začínáme při řetězení až $1\times,\ 2\times,\ \ldots$. Matematický zápis obojího:

$$L^* = \bigcup_{i=0}^{\infty} L^i \qquad\qquad L^+ = \bigcup_{i=1}^{\infty} L^i$$

Postup je stejný jako u iterace, ale pokud počáteční stav původního automatu nepatřil do množiny koncových stavů (tj. slovo ε nepatřilo do jazyka), nebude koncovým stavem ani po úpravě automatu.

Označme $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ původní automat, pak automat rozpoznávající jazyk, který je

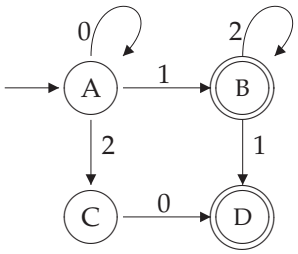
pozitivní iterací původního jazyka, je
$\mathcal{A} = (Q_1 \cup \{s_0\},\ \Sigma,\ \delta,\ s_0,\ F)$, kde $F = F_1 \cup \{s_0\}$, pokud $q_0 \in F_1$, jinak $F = F_1$.

Přechodová funkce:

$$\delta(r,x) = \begin{cases} \delta_1(r,x)\ ;\ r \in Q_1 - F_1, x \in \Sigma, \\ \delta_1(r,x) \cup \delta_1(q_0,x)\ ;\ r \in F_1, x \in \Sigma, \\ \delta_1(q_0,x)\ ;\ r = s_0,\ x \in \Sigma \end{cases}$$

Příklad 2.14

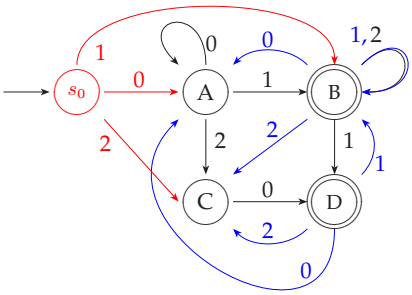
Vytvoříme pozitivní iteraci jazyka následujícího automatu:
$\mathcal{A}_1 = (\{A,B,C,D\}, \{0,1,2\}, \delta_1, A, \{B,D\})$



	0	1	2
→ A	A	B	C
← B		D	B
C	D		
← D			

$\delta_1(A,0) = A$
$\delta_1(A,1) = B$
$\delta_1(A,2) = C$
$\delta_1(B,1) = D$
$\delta_1(B,2) = B$
$\delta_1(C,0) = D$

Přidáváme pouze nové přechody, počáteční stav nebudeme zařazovat do množiny koncových stavů, protože v původním automatu také nebylo rozpoznáváno slovo ε:



	0	1	2
→ s_0	A	B	C
A	A	B	C
← B	A	D, B	B, C
C	D		
← D	A	B	C

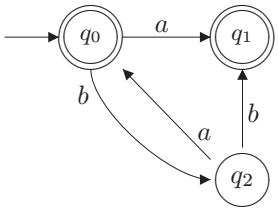
$\mathcal{A} = (\{s_0, A, B, C, D\}, \{0,1,2\}, \delta, s_0, \{B,D\})$

$\delta(s_0,0) = A$ $\quad \delta(A,0) = A$ $\quad \delta(B,1) = D$ $\quad \delta(B,0) = A$ $\quad \delta(D,0) = A$
$\delta(s_0,1) = B$ $\quad \delta(A,1) = B$ $\quad \delta(B,2) = B$ $\quad \delta(B,1) = B$ $\quad \delta(D,1) = B$
$\delta(s_0,2) = C$ $\quad \delta(A,2) = C$ $\quad \delta(C,0) = D$ $\quad \delta(B,2) = C$ $\quad \delta(D,2) = C$

Pokud původní jazyk L obsahuje slovo ε, pak platí $L^* = L^+$, v opačném případě se tyto jazyky nerovnají a platí $L^* = L^+ \cup \{\varepsilon\}$.

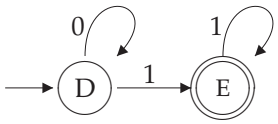
Úkoly

1. Zkonstruujte konečný automat jazyka, který je pozitivní iterací jazyka následujícího automatu:



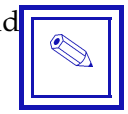
$\mathcal{A}_1$	a	b
$\leftrightarrow$ q_0	q_1	q_2
$\leftarrow$ q_1		
q_2	q_0	q_1

2. Zkonstruujte konečný automat jazyka, který je pozitivní iterací jazyka následujícího automatu:



	0	1
$\rightarrow$ D	D	E
$\leftarrow$ E		E

2.7.2 Zrcadlový obraz (reverze)



Reverze je převrácení. Zrcadlový obraz slova sestrojíme tak, že je zrcadlově „převrátíme“, tj. například ze slova $abcd$ získáme slovo $(abcd)^R = dcba$.

Zrcadlový obraz jazyka je jazyk obsahující všechna slova původního jazyka, ale převrácená. Operace zrcadlení je sama k sobě inverzní – když slovo (nebo jazyk) revertujeme dvakrát, dostáváme původní slovo (jazyk). Zrcadlení můžeme zapsat takto:
$L^R = \{w \ ; \ w^R \in L\}$

Pokud budeme při reverzi pracovat s konečným automatem, především převrátíme všechny cesty, které v automatu existují (tj. převrátíme všechny přechody) a zaměníme počáteční a koncové stavy.

Dále musíme vyřešit jeden problém – počáteční stav musí být vždy jen jeden, ale koncových stavů může být obecně jakýkoliv počet. Proto rozlišíme dva případy – pokud původní automat má jen jediný koncový stav, stačí postup naznačený v předchozím odstavci. Jestliže však má více koncových stavů, musíme zajistit, aby po reverzi existoval jen jediný počáteční stav. Proto vytvoříme nový stav, který v sobě bude shrnovat vlastnosti původních koncových stavů (resp. nových „počátečních“ stavů) týkající se přechodů, které z nich po reverzi vycházejí.

Pokud počáteční stav původního automatu patřil do množiny koncových stavů, bude koncovým stavem také počáteční stav po reverzi.

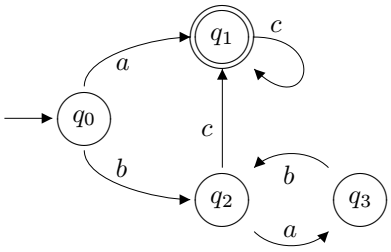
Označme $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ původní automat, pak automat rozpoznávající jazyk, který je reverzí původního jazyka, je
$\mathcal{A} = (Q_1 \cup \{s_0\}, \ \Sigma, \ \delta, \ s_0, \ F)$, množina koncových stavů je $F = \{q_0, s_0\}$, pokud $q_0 \in F_1$, jinak $F = \{q_0\}$ (záleží na tom, zda do původního jazyka patřilo ε). Přechodová funkce:

$$\delta(r,x) = \begin{cases} p \ ; \ \delta_1(p,x) = r, x \in \Sigma, \\ t \ ; \ r = s_0, \delta_1(t,x) \in F_1, x \in \Sigma \end{cases}$$

Na přechodové funkci vidíme, že podle prvního řádku předpisu se všechny přechody obrátí (tj. zamění se zdroj a cíl přechodu), podle druhého řádku vytvoříme přechody vedoucí z nového (počátečního) stavu s_0.

Příklad 2.15

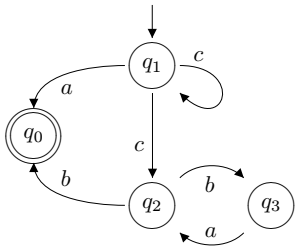
Provedeme operaci zrcadlení jazyka tohoto konečného automatu:
$\mathcal{A}_1 = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta_1, q_0, \{q_1\})$



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

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

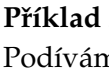
Obrátíme všechny přechody. Protože máme jen jediný koncový stav, stačí pak jen zaměnit počáteční a koncový stav (nebudeme vytvářet nový stav s_0). U tabulky se zaměňují označení řádků s obsahem buněk na tomto řádku.
$\mathcal{A}_R = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \delta, q_1, \{q_0\})$



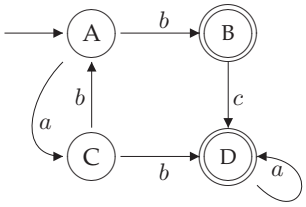
	a	b	c
$\leftarrow q_0$			
$\rightarrow q_1$	q_0		q_1, q_2
q_2		q_0, q_3	
q_3	q_2		

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

Příklad 2.16

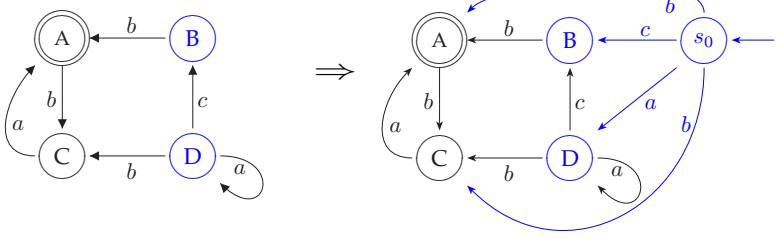


Podíváme se na složitější případ – konečný automat s více koncovými stavy. Postup bude podobný, ale navíc řešíme nutnost existence jediného počátečního stavu. Je dán tento konečný automat:



	a	b	c
$\rightarrow$ A	C	B	
$\leftarrow$ B			D
C		A, D	
$\leftarrow$ D	D		

Jsou zde dva koncové stavy. Nejdřív přesměrujeme všechny přechody a označíme nový koncový stav, a až v druhém kroku vytvoříme nový počáteční stav podobně, jak jsme použili například u sjednocení (tam šlo také o „simulaci“ – nahrazení dvou počátečních stavů jediným).

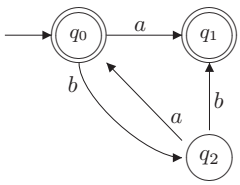


	a	b	c
$\rightarrow s_0$	D	A, C	B
$\leftarrow$ A		C	
B		A	
C	A		
D	D	C	B

Poslední případ, na který se podíváme, je konečný automat s více koncovými stavy, kde také počáteční stav patří do množiny koncových stavů.

Příklad 2.17

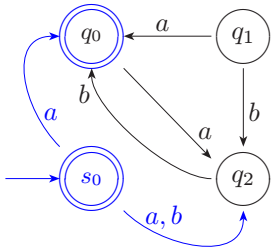
Pro operaci zrcadlení je dán konečný automat $\mathcal{A}_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \delta_1, q_0, \{q_0, q_1\})$



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

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

Narozdíl od předchozích příkladů budou ve výsledném automatu dva koncové stavy. Stav q_0 bude koncový, protože v původním automatu je počátečním stavem, a stav s_0 bude koncový, protože do jazyka původního automatu patří slovo ε, jehož převrácením je opět slovo ε pro jeho rozpoznání musí být počáteční stav (tj. s_0) v množině koncových stavů.
$\mathcal{A}_R = (\{s_0, q_0, q_1, q_2\}, \{a, b\}, \delta, s_0, \{q_0, s_0\})$

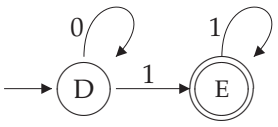


	a	b
$\leftrightarrow$ s_0	q_0, q_2	q_2
$\leftarrow$ q_0	q_2	
q_1	q_0	q_2
q_2		q_0

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

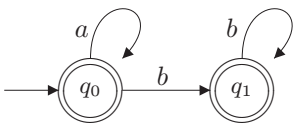
Úkoly

1. Zkonstruujte konečný automat jazyka, který je reverzí (zrcadlovým obrazem) jazyka následujícího automatu:



	0	1
$\rightarrow$ D	D	E
$\leftarrow$ E		E

2. Vytvořte konečný automat jazyka, který je reverzí jazyka následujícího automatu. Pro oba automaty (uvedený i ten, který vytvoříte) sestrojte tabulku přechodů.



3. Vytvořte konečný automat rozpoznávající jazyk $L = \{\varepsilon, \text{tisk}, \text{tis}, \text{síto}\}$ tak, aby jednotlivá slova jazyka byla rozpoznávaná koncovým stavem (tj. pro každé slovo vlastní koncový stav). Potom proveďte reverzi tohoto automatu.
4. Vytvořte zrcadlový obraz jazyka tohoto konečného automatu (má jediný koncový stav, který je zároveň počátečním stavem):

	a	b
$\leftrightarrow\ q_0$	q_1	q_2
q_1		q_2
q_2	q_2	q_0

2.7.3 Průnik

Když vytváříme konečný automat pro průnik dvou jazyků pomocí automatů, které je rozpoznávají, tak vlastně simulujeme (paralelně) činnost obou těchto automatů. Po přečtení každého signálu ze vstupu provedeme jeden krok v obou automatech zároveň. Ve výsledném automatu jsou proto stavy *uspořádanými dvojicemi* stavů z prvního a druhého původního automatu (záleží na pořadí!).

Označme

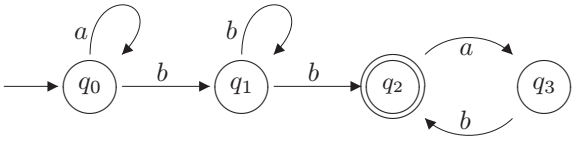
- $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ je první automat,
- $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ je druhý automat,

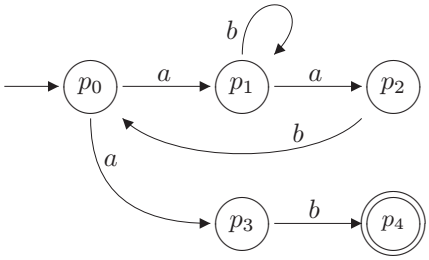
pak automat rozpoznávající jazyk, který je průnikem jazyků obou automatů, je
$\mathcal{A}_P = (Q_1 \times Q_2,\ \Sigma_1 \cap \Sigma_2,\ \delta,\ [q_1, q_2],\ F_1 \times F_2)$, kde operace $\times$ je kartézským součinem uvedených množin. Přechodová funkce:

$\delta([p, r], x) = [\delta_1(p, x), \delta_2(r, x)]$, kde $p \in Q_1, r \in Q_2, x \in \Sigma_2, x \in \Sigma_1 \cap \Sigma_2$

Příklad 2.18

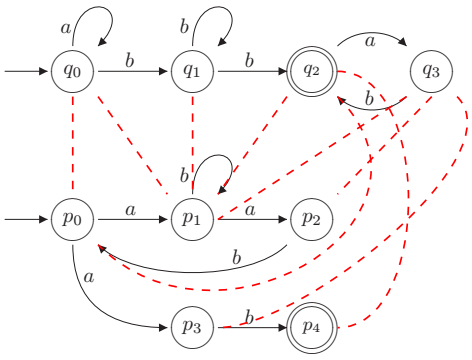
Sestrojíme konečný automat rozpoznávající průnik jazyků následujících dvou automatů:





$\mathcal{A}_1 = (\{q_0, \ldots, q_3\}, \{a, b\}, \delta_1, q_0, \{q_2\})$ $\quad$ $\mathcal{A}_2 = (\{p_0, \ldots, p_4\}, \{a, b\}, \delta_2, p_0, \{p_4\})$

$\delta_1(q_0, a) = \{q_0\}$ $\quad$ $\delta_1(q_2, a) = \{q_3\}$ $\quad$ $\delta_2(p_0, a) = \{p_1, p_3\}$ $\quad$ $\delta_2(p_2, b) = \{p_0\}$
$\delta_1(q_0, b) = \{q_1\}$ $\quad$ $\delta_1(q_3, b) = \{q_2\}$ $\quad$ $\delta_2(p_1, a) = \{p_2\}$ $\quad$ $\delta_2(p_3, b) = \{p_4\}$
$\delta_1(q_1, b) = \{q_1, q_2\}$ $\quad$ $\delta_2(p_1, b) = \{p_1\}$



Nejdřív si vyznačíme průběh „simulace“. Není to nutné (a u složitějšího automatu je to prakticky neproveditelné), ale na diagramu lépe pochopíme paralelnost obou zpracování téhož slova (červeně jsou spojeny stavy, ve kterých jsou automaty ve stejném kroku zpracování slova) – obrázek vpravo. Například pro slovo $abbabab$ paralelně procházíme dvojicemi stavů $[q_0, p_0]$, pak $[q_0, p_1]$, $[q_1, p_1]$, $[q_2, p_1]$, $[q_3, p_2]$, $[q_2, p_0]$, dále $[q_3, p_3]$ a $[q_2, p_4]$.

Protože by bylo hodně náročné (a také nedeterministické a těžko naprogramovatelné) takto vyhledávat všechny dvojice stavů, ve kterých se nachází výpočet v obou automatech ve stejném kroku, použijeme jinou (trochu „otrockou“) metodu:

- jako množinu stavů použijeme množinu všech uspořádaných dvojic, kde první prvek je stav prvního automatu a druhý prvek je stav druhého automatu,
- vytvoříme δ-funkci nebo tabulku přechodů, ve které zachytíme společné přechody na stejný signál v obou automatech,
- odstraníme nepotřebné stavy.

Počátečním stavem bude uspořádaná dvojice obsahující počáteční stavy obou původních automatů, koncové stavy budou všechny uspořádané dvojice, ve kterých jsou oba původní stavy koncovými.

Jednodušší je využití tabulky přechodů. Podle tabulek původních automatů vytvoříme tabulku pro výsledný automat (kombinace řádků ve stejném sloupci).

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

	a	b
$\rightarrow\ p_0$	p_1, p_3	
p_1	p_2	p_1
p_2		p_0
p_3		p_4
$\leftarrow\ p_4$		

Tabulka přechodů výsledného konečného automatu má $4 \cdot 5 = 20$ řádků:

	a	b
$\rightarrow\ [q_0, p_0]$	$[q_0, p_1], [q_0, p_3]$	
$[q_0, p_1]$	$[q_0, p_2]$	$[q_1, p_1]$
$[q_0, p_2]$		$[q_1, p_0]$
$[q_0, p_3]$		$[q_1, p_4]$
$[q_0, p_4]$		
$[q_1, p_0]$		
$[q_1, p_1]$		$[q_1, p_1], [q_2, p_1]$
$[q_1, p_2]$		$[q_1, p_0], [q_2, p_0]$
$[q_1, p_3]$		$[q_1, p_4], [q_2, p_4]$
$[q_1, p_4]$		

(pokračování)	a	b
$[q_2, p_0]$	$[q_3, p_1], [q_3, p_3]$	
$[q_2, p_1]$	$[q_3, p_2]$	
$[q_2, p_2]$		
$[q_2, p_3]$		
$\leftarrow\ [q_2, p_4]$		
$[q_3, p_0]$		
$[q_3, p_1]$		$[q_2, p_1]$
$[q_3, p_2]$		$[q_2, p_0]$
$[q_3, p_3]$		$[q_2, p_4]$
$[q_3, p_4]$		

Nyní odstraníme nepotřebné (tj. nedosažitelné a nadbytečné) stavy tak, jak jsme se učili v předchozích kapitolách, obsah množin je průběžně tříděn pro usnadnění porovnávání.

$S_0 = \{[q_0, p_0]\}$
$S_1 = \{[q_0, p_0], [q_0, p_1], [q_0, p_3]\}$
$S_2 = \{[q_0, p_0], [q_0, p_1], [q_0, p_3], [q_0, p_2], [q_1, p_1], [q_1, p_4]\}$
$S_3 = \{[q_0, p_0], [q_0, p_1], [q_0, p_2], [q_0, p_3], [q_1, p_1], [q_1, p_4], [q_1, p_0], [q_2, p_1]\}$
$S_4 = \{[q_0, p_0], [q_0, p_1], [q_0, p_2], [q_0, p_3], [q_1, p_0], [q_1, p_1], [q_1, p_4], [q_2, p_1], [q_3, p_2]\}$
$S_5 = \{[q_0, p_0], [q_0, p_1], [q_0, p_2], [q_0, p_3], [q_1, p_0], [q_1, p_1], [q_1, p_4], [q_2, p_1], [q_3, p_2], [q_2, p_0]\}$
$S_6 = \{[q_0, p_0], [q_0, p_1], [q_0, p_2], [q_0, p_3], [q_1, p_0], [q_1, p_1], [q_1, p_4], [q_2, p_0], [q_2, p_1], [q_3, p_2], [q_3, p_1], [q_3, p_3]\}$
$S_7 = \{[q_0, p_0], [q_0, p_1], [q_0, p_2], [q_0, p_3], [q_1, p_0], [q_1, p_1], [q_1, p_4], [q_2, p_0], [q_2, p_1], [q_3, p_1], [q_3, p_2], [q_3, p_3], [q_2, p_4]\}$
$S_8 = \{[q_0, p_0], [q_0, p_1], [q_0, p_2], [q_0, p_3], [q_1, p_0], [q_1, p_1], [q_1, p_4], [q_2, p_0], [q_2, p_1], [q_2, p_4], [q_3, p_1], [q_3, p_2], [q_3, p_3]\} = S_7$

Odstranili jsme stavy $[q_0, p_4], [q_1, p_2], [q_1, p_3], [q_2, p_2], [q_2, p_3], [q_3, p_0], [q_3, p_4]$, které jsou nedosažitelné z počátečního stavu. Dále pracujeme s touto tabulkou přechodů:

	a	b
$\rightarrow$ $[q_0, p_0]$	$[q_0, p_1], [q_0, p_3]$	
$[q_0, p_1]$	$[q_0, p_2]$	$[q_1, p_1]$
$[q_0, p_2]$		$[q_1, p_0]$
$[q_0, p_3]$		$[q_1, p_4]$
$[q_1, p_0]$		
$[q_1, p_1]$		$[q_1, p_1], [q_2, p_1]$
$[q_1, p_4]$		
$[q_2, p_0]$	$[q_3, p_1], [q_3, p_3]$	
$[q_2, p_1]$	$[q_3, p_2]$	
$\leftarrow$ $[q_2, p_4]$		
$[q_3, p_1]$		$[q_2, p_1]$
$[q_3, p_2]$		$[q_2, p_0]$
$[q_3, p_3]$		$[q_2, p_4]$

Odstraníme nadbytečné symboly (ze kterých neexistuje cesta do koncového stavu):

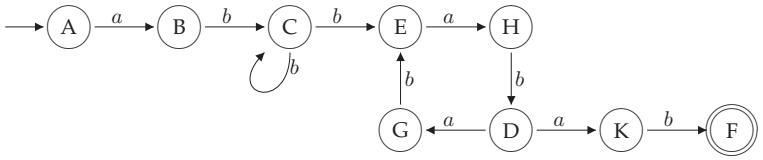
$E_0 = \{[q_2, p_4]\}$
$E_1 = \{[q_2, p_4], [q_3, p_3]\}$
$E_2 = \{[q_2, p_4], [q_3, p_3], [q_2, p_0]\}$
$E_3 = \{[q_2, p_0], [q_2, p_4], [q_3, p_3], [q_3, p_2]\}$
$E_4 = \{[q_2, p_0], [q_2, p_1], [q_2, p_4], [q_3, p_2], [q_3, p_3]\}$
$E_5 = \{[q_2, p_0], [q_2, p_1], [q_2, p_4], [q_3, p_2], [q_3, p_3], [q_1, p_1], [q_3, p_1]\}$
$E_6 = \{[q_1, p_1], [q_2, p_0], [q_2, p_1], [q_2, p_4], [q_3, p_1], [q_3, p_2], [q_3, p_3], [q_0, p_1]\}$
$E_7 = \{[q_0, p_1], [q_1, p_1], [q_2, p_0], [q_2, p_1], [q_2, p_4], [q_3, p_1], [q_3, p_2], [q_3, p_3], [q_0, p_0]\}$
$E_8 = \{[q_0, p_0], [q_0, p_1], [q_1, p_1], [q_2, p_0], [q_2, p_1], [q_2, p_4], [q_3, p_1], [q_3, p_2], [q_3, p_3]\} = E_7$

Po odstranění stavů $[q_0, p_2], [q_0, p_3], [q_1, p_0], [q_1, p_4]$ dostáváme tuto tabulku přechodů (s původním označením stavů a po přeznačení na písmena):

	a	b
$\rightarrow$ $[q_0, p_0]$	$[q_0, p_1]$	
$[q_0, p_1]$		$[q_1, p_1]$
$[q_1, p_1]$		$[q_1, p_1], [q_2, p_1]$
$[q_2, p_0]$	$[q_3, p_1], [q_3, p_3]$	
$[q_2, p_1]$	$[q_3, p_2]$	
$\leftarrow$ $[q_2, p_4]$		
$[q_3, p_1]$		$[q_2, p_1]$
$[q_3, p_2]$		$[q_2, p_0]$
$[q_3, p_3]$		$[q_2, p_4]$

	a	b
$\rightarrow$ A	B	
B		C
C		C, E
D	G, K	
E	H	
$\leftarrow$ F		
G		E
H		D
K		F

Stavový diagram (stavy jsou přeznačené):



Úkoly

1. Vytvořte deterministické konečné automaty pro jazyky $L_1 = \{\text{if, then, else}\}$ a $L_2 = \{\text{if, iff, elif}\}$, v každém z automatů stačí jediný koncový stav pro všechna slova daného jazyka. Potom pomocí automatů vytvořte průnik těchto jazyků.
2. Sestrojte konečný automat, který je průnikem jazyků následujících automatů (pracujte s tabulkami přechodů). Odstraňte všechny nedosažitelné a nadbytečné stavy a sestrojte stavový diagram.

	0	1
$\rightarrow$ A	B	A
$\leftarrow$ B	C	
C		D
D	B	

	0	1
$\rightarrow$ E		F
F	F	H
$\leftarrow$ H	H	H

Pro kontrolu: po odstranění nepotřebných stavů by měl automat mít osm stavů, z toho jeden koncový, jeden cyklus přes jeden stav a jeden cyklus přes tři stavy.

2.7.4 Homomorfismus

Homomorfismus je takové zobrazení, které

- zachovává neutrální prvek (tj. u řetězců prázdný řetězec ε zobrazí opět na ε)
- zachovává zobrazení operace, pro kterou je definován (u řetězců jde o zřetězení, tj. $h(a \cdot b) = h(a) \cdot h(b)$)
- výsledkem zobrazení prvku (signálu, znaku) je vždy řetězec (u substituce, což je zobecnění homomorfismu, je výsledkem zobrazení množina řetězců).

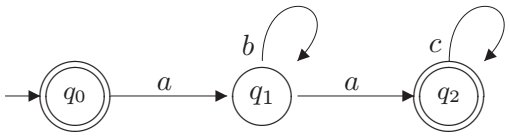
Když konstruujeme konečný automat homomorfismu pro některý jazyk, využíváme plně způsobu definice daného homomorfismu – pokud je v předpisu tohoto zobrazení například $h(a) = xyz$, tj. znak a má být zobrazen na řetězec xyz, vezmeme postupně všechny přechody označené znakem a a nahradíme je cestami rozpoznávajícími slovo xyz. Každá z těch cest musí být samozřejmě samostatná, vždy jeden konkrétní přechod označený signálem a nahradíme jednou samostatnou cestou pro xyz.

Příklad 2.19

Je dán automat $\mathcal{A}$ rozpoznávající jazyk $L(\mathcal{A}) = \{\varepsilon\} \cup \{ab^iac^j \ ; \ i,j \geq 0\}$. Sestrojíme konečný automat $\mathcal{A}_h$ rozpoznávající jazyk vzniklý uplatněním homomorfismu h na jazyk automatu $\mathcal{A}$.

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

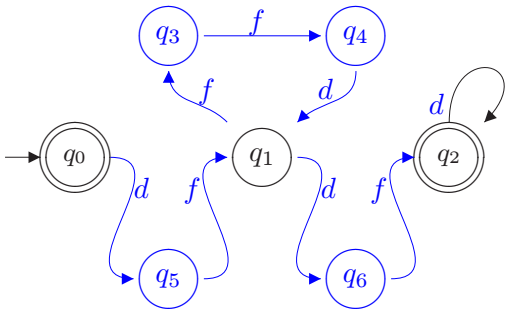
Homomorfismus:
$h(a) = df$,
$h(b) = fffd$,
$h(c) = d$



Po úpravě vypadá konečný automat následovně:

$\mathcal{A}_h$	d	f
$\leftrightarrow\ q_0$	q_5	
q_1	q_6	q_3
$\leftarrow\ q_2$	q_2	
q_3		q_4
q_4	q_1	
q_5		q_1
q_6	q_2	

Původní abeceda:
$\Sigma_1 = \{a, b, c\}$
Nová abeceda:
$\Sigma_2 = \{d, f\}$



Po úpravě dostáváme automat rozpoznávající jazyk $\{\varepsilon\} \cup \{df(ffd)^idfd^j \ ; \ i,j \geq 0\}$.

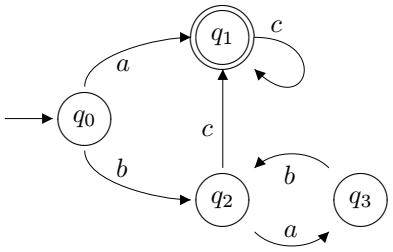
Úkoly

Je dán konečný automat $\mathcal{A}$ s jazykem $L = L(\mathcal{A})$ a homomorfismy h_1 a h_2.

$h_1(a) = 01$
$h_1(b) = 110$
$h_1(c) = 1$

$h_2(a) = dd$
$h_2(b) = g$
$h_3(c) = dg$

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



Zjistěte jazyky $h_1(L)$ a $h_2(L)$ a vytvořte také konečné automaty pro tyto jazyky (úpravou automatu $\mathcal{A}$). Jazyk automatu $\mathcal{A}$ je $L(\mathcal{A}) = \{ac^i \ ; \ i \geq 0\} \cup \{b(ab)^icc^j \ ; \ i,j \geq 0\}$.

Dále pro výsledné automaty sestrojte jejich δ-funkci včetně plné specifikace, nezapomeňte na změnu abecedy.

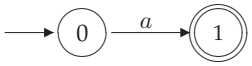
2.8 Sestrojení konečného automatu podle regulárního výrazu

Při sestrojení konečného automatu podle zadaného regulárního výrazu využíváme vztahu mezi prvky regulárního výrazu ($+$, $\cdot$, *) a operacemi nad množinami řetězců ($\cup$, $\cdot$, *). V předchozích příkladech jsme se naučili pracovat s regulárními operacemi (což, jak víme, jsou sjednocení, zřetězení a iterace) a tyto postupy použijeme také zde.

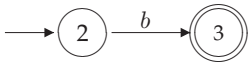
Příklad 2.20

Sestrojíme konečný automat pro regulární výraz $a^*b + (b^*a + bbc)^*$.

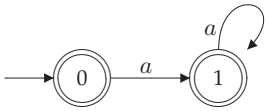
$REG_1 = a$



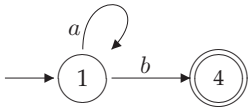
$REG_2 = b$



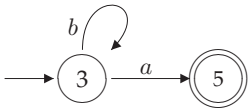
$REG_3 = (REG_1)^* = a^*$



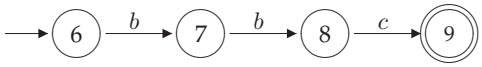
$REG_4 = (REG_3 \cdot REG_2) = a^*b$



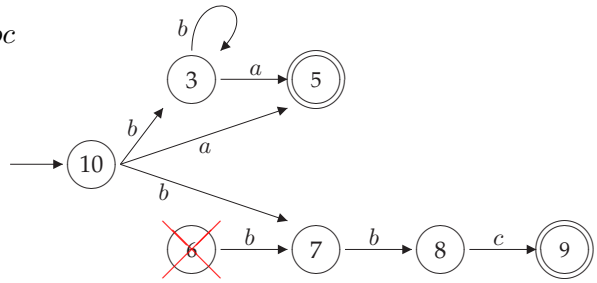
$REG_5 = ((REG_2)^* \cdot REG_1) = b^*a$



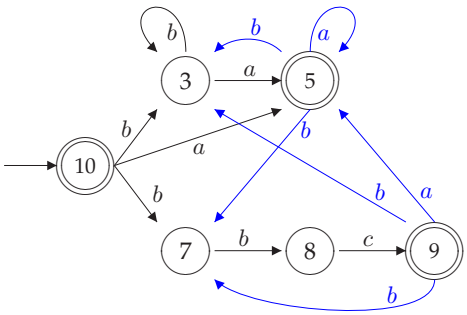
$REG_6 = bbc$



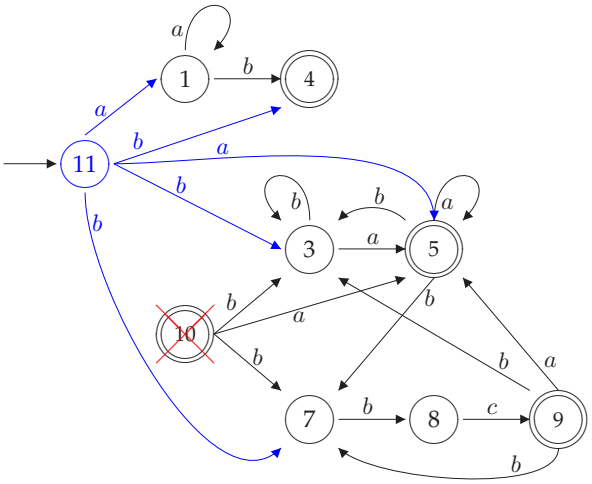
$REG_7 = (REG_5 + REG_6) = b^*a + bbc$



$REG_8 = (REG_7)^* = (b^*a + bbc)^*$



Výsledný automat: $REG_9 = (REG_4 + REG_8) = a^*b + (b^*a + bbc)^*$



	a	b	c
$\rightarrow 11$	1, 5	3, 4, 7	
1	1	4	
3	5	3	
$\leftarrow 4$			
$\leftarrow 5$	5	3, 7	
7		8	
8			9
$\leftarrow 9$	5	3, 7	

Stav 10 je nedosažitelný z počátečního stavu, proto může být odstraněn (v tabulce vpravo se již nenachází).

Úkoly

Sestrojte konečné automaty podle těchto regulárních výrazů:

- $(a + b^*) \cdot b^*$
- $a \cdot (b + (ab)^*) + b$
- $(01 + 10)^*10^*$
- $(abc)^* \cdot a \cdot (ba + c)$

Kapitola 3

Regulární gramatiky

3.1 Vytváříme regulární gramatiku

Nejdřív si ujasníme, co to vlastně je (regulární) gramatika.

Obecně a *neformálně* můžeme říci, že gramatika je konstrukce, která popisuje strukturu daného jazyka a jednotlivých slov tohoto jazyka. Říká nám, co má být dřív, co později, co za čím má následovat, případně jakým způsobem má být co v čem vnořeno, záleží na složitosti samotné gramatiky. Čím je složitější jazyk, který má gramatika popisovat, tím složitější bude samozřejmě i samotná gramatika.

Podle definice je regulární gramatika uspořádaná čtveřice $G = (N, T, P, S)$, kde jednotlivé prvky této posloupnosti (opravdu posloupnosti, záleží na pořadí) znamenají:

- N je konečná neprázdná množina neterminálních symbolů, pro nás představuje obdobu proměnných,
- T je konečná neprázdná množina terminálních symbolů, pro nás je to vlastně abeceda, z jejíchž prvků (znaků) jsou tvořena slova jazyka generovaného naší gramatikou,
- P je konečná neprázdná množina pravidel, jejichž formu probereme dále,
- $S \in N$ je startovací symbol gramatiky (je to jeden z neterminálních symbolů, tak je jasné, proč je ta množina neprázdná).

Všimněte si slov „konečná“ a „neprázdná“, která se v definici často vyskytují. Tato slova jsou velmi důležitá a bez nich by definice byla neúplná.

Pravidla v množině P mohou být v jednom z těchto tvarů:

$$A \rightarrow aB \tag{3.1}$$

$$A \rightarrow a \tag{3.2}$$

kde $A, b \in N$ (jsou to neterminály) a $a \in T$ (terminál). Žádný jiný tvar pravidel není přípustný, už by se nejednalo o regulární gramatiku.

Dále je přípustné pravidlo $S \rightarrow \varepsilon$, ale jen tehdy, když je S startovací symbol gramatiky a nevyskytuje se na pravé straně žádného pravidla. Je zřejmé, že toto pravidlo bude pouze v takové gramatice, která generuje jazyk obsahující prázdné slovo.

Příklad 3.1

Vytvoříme regulární gramatiku, která generuje tento konečný jazyk:
$L = \{\text{vlak},\ \text{drak},\ \text{vlek}\}$
Postupujeme vždy zleva doprava (tak jak čteme běžný text), budeme generovat jedno písmeno za druhým. Musíme se držet tvaru pravidel podle předpisů 3.1 a 3.2. Epsilonové pravidlo nepotřebujeme, protože jazyk neobsahuje prázdné slovo.

Nejdřív se věnujme prvnímu slovu:
$S \to vA \qquad A \to lB \qquad B \to aC \qquad C \to k$
Druhé slovo (musíme začínat zase startovacím symbolem, ten je na začátku generování každého slova gramatiky):
$S \to dD \qquad D \to rE \qquad E \to aF \qquad F \to k$
Třetí slovo:
$S \to vG \qquad G \to lH \qquad H \to eK \qquad K \to k$

Celá gramatika:
$G = (\{S, A, B, C, D, E, F, G, H, K\},\ \{v, l, a, k, d, r, e\},\ P,\ S)$, kde P obsahuje tato pravidla:

$S \to vA \mid dD \mid vG$	$A \to lB$	$D \to rE$	$G \to lH$
	$B \to aC$	$E \to aF$	$H \to eK$
	$C \to k$	$F \to k$	$K \to k$

Všechna pravidla přepisující symbol S jsme shrnuli na jediný řádek, oddělili jsme je svislicí $|$. Nejde o lomítko ani zpětné lomítko, je to svislá čára (na klávesnici ji obvykle najdeme vpravo nad zpětným lomítkem na anglické klávesnici). Tento symbol se v logice používá ve smyslu „nebo“, tentýž význam má i zde – symbol S můžeme přepsat podle prvního nebo druhého nebo třetího pravidla.

Ještě si ověříme, zda v gramatice lze vygenerovat všechna tři slova:
$S \Rightarrow vA \Rightarrow vlB \Rightarrow vlaC \Rightarrow vlak$
$S \Rightarrow dD \Rightarrow drE \Rightarrow draF \Rightarrow drak$
$S \Rightarrow vG \Rightarrow vlH \Rightarrow vleK \Rightarrow vlek$

Všimněte si – jednoduchou šipku používáme u pravidel, kdežto v odvození slov používáme dvojitou šipku. Toto pravidlo budeme bezvýhradně dodržovat, v každém z těchto případů jde o něco jiného.

Příklad 3.2

Gramatika v předchozím příkladu byla zbytečně složitá. Pokusíme se ji zjednodušit tak, aby generovala i nadále tentýž jazyk, ale aby obsahovala menší počet pravidel.

Dvě ze slov jazyka stejně začínají. Proto bychom mohli stejně začínat i při jejich generování.
$G' = (\{S, A, B, C, D, E, F, K\},\ \{v, l, a, k, d, r, e\},\ P',\ S)$, kde P' obsahuje tato pravidla:

$S \to vA \mid dD$	$A \to lB$	$D \to rE$	$K \to k$
	$B \to aC \mid eK$	$E \to aF$	
	$C \to k$	$F \to k$	

Slovo „vlek“ odvodíme takto:
$S \Rightarrow vA \Rightarrow vlB \Rightarrow vleK \Rightarrow vlek$

Dala by se množina pravidel ještě víc zredukovat? Třeba takto:
$G'' = (\{S, A, B, C, D, E\},\ \{v, l, a, k, d, r, e\},\ P'',\ S)$, kde P'' obsahuje tato pravidla:

$S \to vA \mid dD$ $\quad$ $A \to lB$ $\quad$ $D \to rE$ $\quad$ $C \to k$
$\quad$ $B \to aC \mid eC$ $\quad$ $E \to aC$

Ověříme, zda se nám nezměnil generovaný jazyk:
$S \Rightarrow vA \Rightarrow vlB \Rightarrow vlaC \Rightarrow vlak$
$S \Rightarrow dD \Rightarrow drE \Rightarrow draC \Rightarrow drak$
$S \Rightarrow vA \Rightarrow vlB \Rightarrow vleC \Rightarrow vlek$

Je zřejmé, že v gramatice lze vygenerovat opravdu jen tato tři slova, žádné další.

Vždy bychom se měli přesvědčit, jak gramatika „funguje“, vyzkoušet vygenerovat pár typických slov jazyka (alespoň několik nejkratších). Nejde jen o to, aby gramatika generovala všechna slova příslušného jazyka, ale i o to, aby gramatika negenerovala slova do jazyka nepatřící.

Příklad 3.3

Sestrojíme regulární gramatiku generující jazyk $L = ab^* + cd^*$.

Slova jazyka začínají buď písmenem a nebo písmenem c. Tyto dvě „kategorie“ oddělíme hned v pravidlech přepisujících startovací symbol.
$S \to aA \mid cB$
Teď především musíme dát pozor, aby se nám písmena ve slovech „nepomíchala“ – za písmenem a mohou následovat jen písmena b (jakýkoliv počet) a za písmenem c mohou následovat jen písmena d (taky jakýkoliv počet).
$A \to bA \mid b$
$B \to dB \mid d$
Ale ještě jsme neskončili. Do jazyka generovaného touto gramatikou totiž patří i slova o délce jedna:
$S \to a \mid c$
Celá gramatika:
$G = (\{S, A, B\},\ \{a, b, c, d\},\ P,\ S)$
$S \to aA \mid a \mid cB \mid c$
$A \to bA \mid b$
$B \to cB \mid c$
Otestujeme gramatiku vygenerováním několika slov:
$S \Rightarrow aA \Rightarrow abA \Rightarrow abbA \Rightarrow abbb$
$S \Rightarrow a$
$S \Rightarrow cB \Rightarrow cb$

Pravidlo $A \to bA$ je *rekurzvní*, protože se typicky používá v cyklu (je to obdoba cyklu v automatu, ale cyklus probíhá mezi neterminály, ne mezi stavy). Každá rekurze musí nutně skončit, k ukončení rekurze u nás slouží pravidlo $A \to b$. Tato rekurze je přímá (přes jediný neterminál), ale může být i nepřímá, jak uvidíme na následujícím příkladu.

Příklad 3.4

Vytvoříme regulární gramatiku generující jazyk $L = (ab)^*$

Vidíme, že do jazyka patří i prázdné slovo. Proto použijeme pravidlo
$S \to \varepsilon$

Jazyk L je nekonečný, proto zde určitě bude rekurze. Ale ne přímá, budeme potřebovat ještě další neterminál. Musíme střídavě generovat písmena a a b:
$A \to aB$
$B \to bA$
Každá rekurze se někdy musí zastavit:
$B \to b$
Ještě tento cyklus napojíme na startovací symbol:
$S \to aB$
Celá gramatika:
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$, množina P:
$S \to aB \mid \varepsilon$
$A \to aB$
$B \to bA \mid b$
Ověříme, jaká slova jsou gramatikou generována:
$S \Rightarrow \varepsilon$
$S \Rightarrow aB \Rightarrow ab$
$S \Rightarrow aB \Rightarrow abA \Rightarrow abaB \Rightarrow ababA \Rightarrow ababaB \Rightarrow abagab$, atd.

Mohlo by se zdát, že bychom mohli gramatiku sestavit takto:
$G' = (\{S, B\},\ \{a, b\},\ P',\ S)$, množina P':
$S \to aB \mid \varepsilon$
$B \to bS \mid b$
Sice bychom měli o dvě pravidla méně, ale pozor – při použití ε-pravidla se startovací symbol nesmí vyskytovat na pravé straně žádného pravidla, tato gramatika již není regulární!

Je třeba si uvědomit, že gramatika *generuje* slova jazyka, kdežto automat dostane slovo na svůj vstup a ověřuje, zda toto slovo patří do jazyka, který *rozpoznává*. Přesto je mezi gramatikami a automaty velmi úzký vztah – pro daný jazyk můžeme sestrojit gramatiku, která ho generuje (tj. popisuje strukturu všech slov patřících do jazyka) a taky automat, který jeho slova rozpoznává (tj. dokáže určit, zda dané slovo patří či nepatří do daného jazyka). V případě regulárních gramatik a konečných automatů je celkem jednoduché provádět převody mezi nimi, tedy podle gramatiky sestrojit ekvivalentní automat a podle automatu sestrojit ekvivalentní gramatiku.

Úkoly

Sestrojte regulární gramatiky generující tyto jazyky:

- $L_1 = \{0100,\ 1011,\ 1110\}$
- $L_2 = \{\varepsilon,\ 0100,\ 1011,\ 1110\}$
- $L_3 = a^*$
- $L_4 = (abc)^*$
- $L_5 = \{a^m b^n\ ;\ m \geq 1,\ n \geq 0\}$
- $L_6 = (ab)^* + ba^*$
- $L_7 = (ab + cb)^*$

3.2 Konečný automat podle regulární gramatiky

Pokud podle regulární gramatiky vytváříme konečný automat, tak vlastně tento konečný automat simuluje generování svého vstupu v původní gramatice. Jeden krok automatu skládající se z načtení signálu ze vstupu a přechodu do následujícího stavu odpovídá použití jednoho pravidla v gramatice, a to takového, které generuje stejný terminál (signál).

Postupujeme takto:

- množinu terminálů gramatiky použijeme pro *abecedu* automatu,
- množinu neterminálů použijeme pro *stavy*,
- stav vytvořený ze startovacího symbolu gramatiky bude *počátečním stavem*,
- do množiny stavů také přidáme nový stav (obvykle značený X), který se stane *koncovým*,
- pokud je v jazyce generovaném gramatikou slovo ε, pak počáteční stav automatu bude také patřit do množiny koncových stavů, aby automat rozpoznával slovo ε,
- δ-funkci vytvoříme podle pravidel gramatiky.

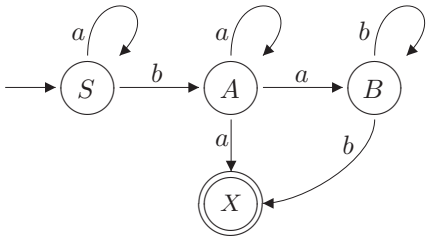
Příklad 3.5

Vytvoříme konečný automat rozpoznávající jazyk generovaný touto gramatikou:
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$ s těmito pravidly:
$S \to aS \mid bA$
$A \to aA \mid aB \mid a$
$B \to bB \mid b$

V gramatice není žádné ε-pravidlo (tj. pravidlo, na jehož pravé straně je řetězec ε), proto počáteční stav nebude patřit do množiny koncových stavů (tato množina bude jednoprvková). Jako abecedu použijeme množinu terminálů, stavy vytvoříme z neterminálů a přidáme jeden koncový stav X.

Přechody tvoříme podle pravidel gramatiky – například podle pravidla $A \to aB$ přidáme do δ-funkce $\delta(A, a) \ni B$. Postup je dobře vidět na δ-funkci:
$\mathcal{A} = (\{S, A, B, X\},\ \{a, b\},\ \delta,\ S,\ \{X\})$
$\delta(S, a) = \{S\}$ *podle* $S \to aS$
$\delta(S, b) = \{A\}$ *podle* $S \to bA$
$\delta(A, a) = \{A, B, X\}$ *podle* $A \to aA \mid aB \mid a$
$\delta(B, b) = \{B, X\}$ *podle* $B \to bB \mid b$

Podobně sestrojíme stavový diagram a tabulku přechodů:



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

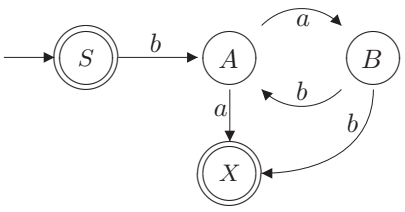
Jazyk generovaný gramatikou a rozpoznávaný automatem je
$L(G) = L(\mathcal{A}) = a^*ba^*a(\varepsilon + b^*b)$

Příklad 3.6

Narozdíl od předchozího příkladu zde máme gramatiku s ε-pravidlem:
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$, v množině P jsou pravidla
$S \rightarrow bA \mid \varepsilon$
$A \rightarrow aB \mid a$
$B \rightarrow bA \mid b$

V jazyce generovaném gramatikou je také prázdné slovo ε a toto slovo musí přijímat i konečný automat, který vytvoříme. To lze zajistit pouze tak, že počáteční stav automatu přidáme do množiny koncových stavů.
$\mathcal{A} = (\{S, A, B, X\},\ \{a, b\},\ \delta,\ S,\ \{S, X\})$

$\delta(S, b) = \{A\}$
$\delta(A, a) = \{B, X\}$
$\delta(B, b) = \{A, X\}$



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

Jazyk generovaný gramatikou a rozpoznávaný automatem je
$L(G) = L(\mathcal{A}) = \varepsilon + b(ab)^*a + b(ab)^*ab = \varepsilon + b(ab)^*a(\varepsilon + b)$

Úkoly

Podle následujících gramatik sestrojte ekvivalentní konečný automat (tj. rozpoznávající jazyk generovaný touto gramatikou).

$G_1 = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \rightarrow aS \mid bS \mid cA$
$A \rightarrow aA \mid cB \mid a$
$B \rightarrow bB \mid cA \mid b$

$G_2 = (\{A, B, C\},\ \{0, 1\},\ P,\ A)$
$A \rightarrow 0B \mid 1C \mid \varepsilon$
$B \rightarrow 0C \mid 1C$
$C \rightarrow 1B \mid 0$

$G_3 = (\{R, Y, Z, W\},\ \{a, b, c\},\ P,\ R)$
$R \to aR \mid bR \mid cZ$
$Y \to cR \mid aY \mid b$
$Z \to aY \mid aW$
$W \to bR \mid b$

$G_4 = (\{S, A, B, C\},\ \{0, 1, 2\},\ P,\ S)$
$S \to 0A \mid 1B \mid 2C \mid \varepsilon$
$A \to 0B \mid 0$
$B \to 1C \mid 1$
$C \to 2A$

3.3 Regulární gramatika podle konečného automatu

Při vytvoření gramatiky generující jazyk, který je rozpoznáván daným konečným automatem, je nejjednodušší obrátit postup, který jsme použili pro vytvoření automatu podle gramatiky.

Když jsme sestrojili konečný automat podle regulární gramatiky, automat měl vždy tyto vlastnosti:

- pokud v jazyce generovaném gramatikou není slovo ε, pak
  - existuje jediný koncový stav (nově přidaný), obvykle pojmenovaný X,
  - ze stavu X nevede žádný přechod (tj. když se výpočet dostane do koncového stavu, nelze dále pokračovat).
- pokud v jazyce generovaném gramatikou je slovo ε, pak
  - kromě nově přidaného koncového stavu X je v množině koncových stavů i počáteční stav,
  - ze stavu X nevede žádný přechod,
  - do počátečního stavu nevede žádný přechod (ekvivalent k faktu, že pokud v gramatice máme pravidlo $S \to \varepsilon$, pak se symbol S nesmí vyskytovat na pravé straně žádného pravidla).

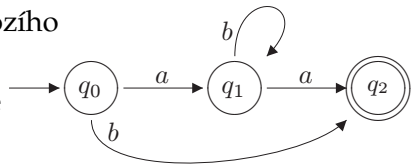
Abychom tedy mohli použít opačný postup k postupu předchozímu, musíme zadaný automat nejdřív upravit do tvaru odpovídajícího výše uvedeným požadavkům. Úprava spočívá v těchto krocích (jejich pořadí je důležité):

1. Jestliže počáteční stav patří do množiny koncových stavů a zároveň do tohoto stavu vede některý přechod:
   - vytvoříme nový počáteční stav (také zařadíme do množiny koncových stavů), do kterého nebudou vést žádné přechody, ale z něho budou vést tytéž přechody jako z původního počátečního stavu,
   - původní počáteční stav zůstane v množině koncových stavů.
2. Pokud je v množině koncových stavů více než jeden stav (případně kromě počátečního stavu, toho se tento bod netýká):
   - vytvoříme nový koncový stav X (nebo jinak označený),
   - ze stavu X nepovede žádný přechod, ale do tohoto stavu povedou tytéž přechody, které vedou do původních koncových stavů,
   - původní koncové stavy sice v automatu necháme, ale vyřadíme je z množiny koncových stavů (netýká se počátečního stavu).
3. Jestliže z koncového stavu vedou přechody, problém vyřešíme stejně jako v předchozím bodě.

Příklad 3.7

Nejdřív si ukážeme jednodušší případ – konečný automat odpovídající uvedeným požadavkům.

Jak vidíme, stav q_2 zde vlastně zastupuje stav X podle předchozího postupu. To znamená, že v gramatice bude množina neterminálů dvouprvková – $\{q_0, q_1\}$, ze stavu q_2 žádný neterminál nevytvoříme a pravidla, která s ním souvisejí, budou sloužit k ukončení generování slova. Můžeme také přejmenovat neterminály.



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

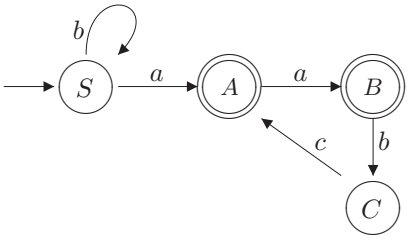
Podle automatu:
$G = (\{q_0, q_1\},\ \{a, b\},\ P,\ q_0)$
$q_0 \rightarrow aq_1 \mid b$
$q_1 \rightarrow a \mid bq_1$

Přejmenované stavy:
$G = (\{A, B\},\ \{a, b\},\ P,\ A)$
$A \rightarrow aB \mid b$
$B \rightarrow a \mid bB$

Gramatika, kterou jsme vytvořili, generuje jazyk $L(G) = ab^*a + b$, což je také jazyk rozpoznávaný zadaným automatem.

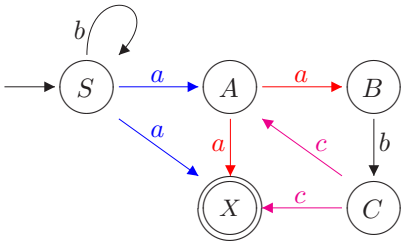
Příklad 3.8

Vytvoříme regulární gramatiku generující jazyk tohoto automatu:



Tento automat má dva koncové stavy, z nichž žádný není počáteční, proto musíme redukovat počet koncových stavů. Navíc z obou koncových stavů vycházejí přechody, to je další důvod pro transformaci.

Vytvoříme stav X, do kterého nasměrujeme všechny přechody mířící do původních koncových stavů. Stav X pak bude jediným koncovým stavem. Po úpravě sestrojíme regulární gramatiku stejně jako u předchozího příkladu.



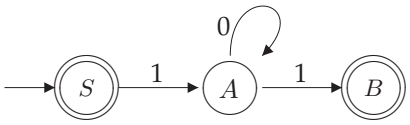
	a	b	c
$\rightarrow S$	A, X	S	
A	B, X		
B		C	
C			A, X
$\leftarrow X$			

$G = (\{S, A, B, C\},$
$\{a, b, c\}, P, S)$
$S \rightarrow aA \mid a \mid bS$
$A \rightarrow aB \mid a$
$B \rightarrow bC$
$C \rightarrow cA \mid c$

Jazyk generovaný gramatikou je $L(G) = b^*a(abc)^*(\varepsilon + a)$.

Příklad 3.9

Následující automat nemusíme upravovat, můžeme přímo napsat regulární gramatiku – startovací symbol patří do množiny koncových stavů, ale do něho nevede žádný přechod.



	0	1
↔ S		A
A	A	B
← B		

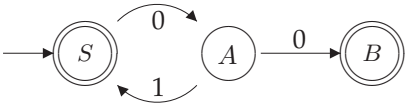
$G = (\{S, A\},\ \{0, 1\},\ P,\ S)$

$S \to 1A \mid \varepsilon$

$A \to 0A \mid 1$

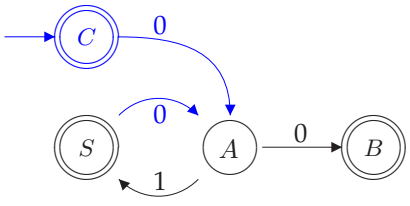
Generovaný (rozpoznávaný) jazyk je $L(G) = 10^*1$.

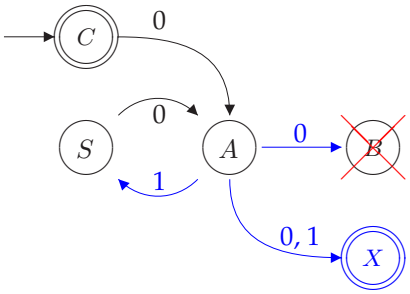
Příklad 3.10



V posledním příkladu si ukážeme převod automatu, jehož počáteční stav je zároveň stavem koncovým a vedou do něho přechody.

Nejdřív automat upravíme. Přidáme nový (počáteční) stav C, který také bude patřit do množiny koncových stavů (aby automat mohl rozpoznat prázdné slovo ε) – tento stav bude využit pouze na začátku výpočtu. Potom přidáme nový koncový stav X, který využijeme na konci každého výpočtu. Původní koncové stavy (kromě počátečního stavu C vyřadíme z množiny koncových stavů.





	0	1
↔ C	A	
S	A	
A	B, X	X
← X		

Ze stavu B nevede cesta do koncového stavu, proto může být odstraněn. Výsledná gramatika je následující:

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

$C \to 0A$

$S \to 0A$

$A \to 0B \mid 0 \mid 1$

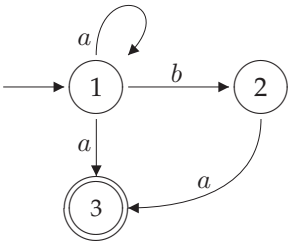
Jazyk generovaný gramatikou G vyjádřený regulárním výrazem je

$L(G) = 0(10)^*(0 + 1)$.

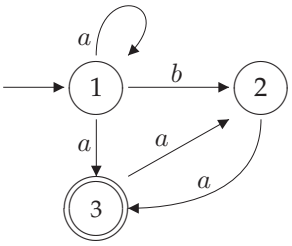
Úkoly

K následujícím konečným automatům vytvořte ekvivalentní regulární gramatiky:

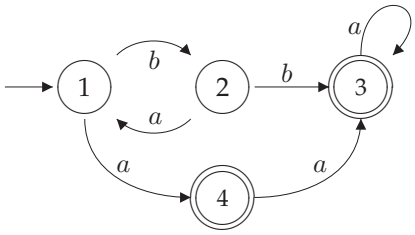
(a)



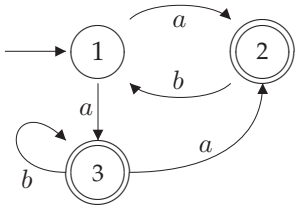
(b)



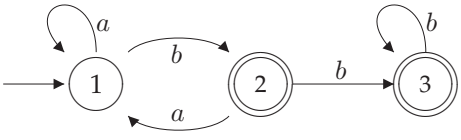
(c)



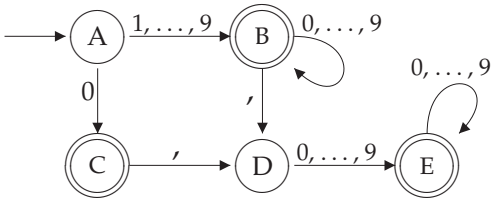
(d)



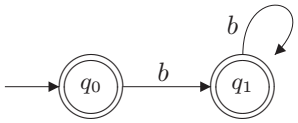
(e)



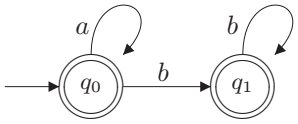
(f)



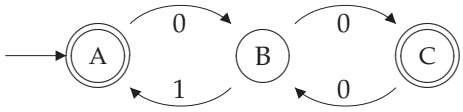
(g)



(h)



(i)



Kapitola 4

Bezkontextové gramatiky

4.1 Vytváříme bezkontextovou gramatiku

Bezkontextové gramatiky mají obecnější tvar pravidel než regulární – na levé straně pravidla je vždy jeden neterminál (jako u regulárních), ale na pravé straně je jakýkoliv řetězec skládající se z terminálů a neterminálů (včetně prázdného řetězce).

Když chceme sestrojit bezkontextovou gramatiku pro zadaný jazyk, pokusíme se popsat strukturu tohoto jazyka s využitím rekurze.

Příklad 4.1

Sestrojíme bezkontextovou gramatiku pro jazyk $L = \{a^n b a^n b \mid n \geq 0\}$ a vytvoříme derivaci (odvození) slova $aabaab$.
$G = (\{S, A\},\ \{a, b\},\ P,\ S)$
$S \to Ab$
$A \to aAa \mid b$

Neterminál A generuje téměř celé slovo, až na poslední symbol b. Část slova generovaná tímto neterminálem je symetrická, proto pravidla jsou vlastně lineární (každá lineární gramatika je zároveň bezkontextová, tedy zadání je v tomto směru splněno). Následuje derivace slova $aabaab$:
$S \Rightarrow Ab \Rightarrow aAab \Rightarrow aaAaab \Rightarrow aabaab$

Příklad 4.2

Sestrojíme bezkontextovou gramatiku generující jazyk
$L = \{wcw^R\ ;\ w \in \{a, b\}^*\}$

Vidíme, že uprostřed slova je symbol c. Dále poloviny slova jsou navzájem symetrické, zrcadlí se. Gramatika tedy bude vypadat takto:
$G = (\{S\},\ \{a, b\},\ P,\ S)$
$S \to aSa \mid bSb \mid c$

Podle gramatiky vygenerujeme několik slov:
$S \Rightarrow c$
$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbcbba$
$S \Rightarrow bSb \Rightarrow bbSbb \Rightarrow bbcbb$

I zde platí – v jednoduchosti je síla. Zde navíc je jednoduchost nezbytná. Kdybychom použili další neterminály a například bychom přidali pravidlo $S \rightarrow AcA$ a pak $A \rightarrow aA \mid bA$ a pak něco na ukončení rekurze, gramatika by generovala úplně jiný jazyk.

Vždy, pokud potřebujeme synchronizovat dvě části slova (například zde použití symbolu a nebo symbolu b), musíme to provést *v jediném pravidle*! Jinak by synchronizace nefungovala.

Úkoly

Sestrojte bezkontextové gramatiky generující tyto jazyky:

- $L_1 = \{a^n b^n \ ; \ n \geq 1\}$
- $L_2 = \{ww^R \ ; \ w \in \{a,b\}^*\}$ (pozn.: v bezkontextových gramatikách můžeme použít ε-pravidlo kdekoliv – u kteréhokoliv neterminálu, a taky tento neterminál klidně může být na pravé straně jakéhokoliv pravidla)
- $L_3 = \{a^n b^n c^k \ ; \ n,k \geq 0\}$
- $L_4 = \{(01)^n 1^{2n} \ ; \ n \geq 1\}$

4.2 Derivační strom

Derivace je odvození slova v gramatice. Každé slovo jazyka generovaného gramatikou má tedy (nejméně jednu) svou derivaci.

Derivační strom je vlastně graf, který je stromem (má jediný kořen, každý uzel kromě kořene má právě jednoho předka a hrany jsou orientované, i když orientaci neznačíme), tvořený podle pravidel gramatiky použitých v derivaci, podle které je derivační strom vytvořen.

Pozor – ke každé derivaci lze sestrojit právě jeden derivační strom, je to vlastně grafické znázornění derivace daného slova. Pokud pro toto slovo existuje víc různých derivací (tj. slovo lze v gramatice odvodit několika různými způsoby), může existovat pro jedno slovo více derivačních stromů.

Příklad 4.3

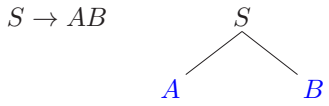
Sestrojíme bezkontextovou gramatiku pro jazyk $L = \{0^n 10^{2n} 1^i \ ; \ i,n \geq 1\}$, vytvoříme derivaci slova 0010000111 a derivační strom k této derivaci.
$G = (\{S,A,B\},\ \{0,1\},\ P,\ S)$
$S \rightarrow AB$
$A \rightarrow 0A00 \mid 0100$
$B \rightarrow 1B \mid 1$

Derivace slova 0010000111 je následující:
$S \Rightarrow AB \Rightarrow 0A00B \Rightarrow 0010000B \Rightarrow 00100001B \Rightarrow 001000011B \Rightarrow 0010000111$

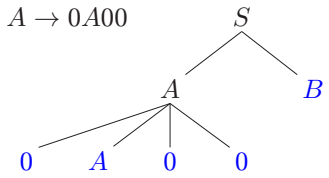
Modře je vyznačena pravá část pravidla, které bylo v daném kroku odvození použito (například v druhém kroku jsme použili pravidlo $A \to 0A00$).

Derivační strom se vždy vztahuje ke konkrétní derivaci. Podle výše uvedené derivace postupně sestrojíme derivační strom takto (jsou uvedeny i mezikroky):

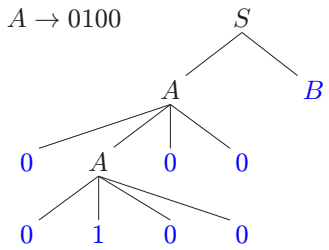
$S \Rightarrow AB$



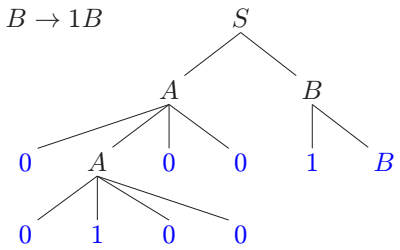
$S \Rightarrow AB \Rightarrow 0A00B$



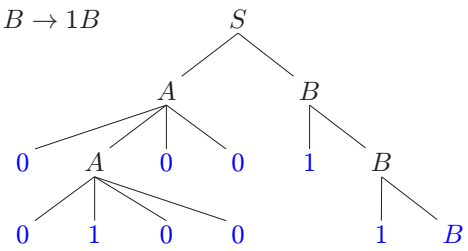
$S \Rightarrow AB \Rightarrow 0A00B \Rightarrow 0010000B$



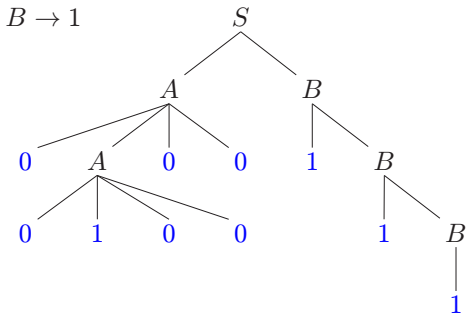
$S \Rightarrow \ldots \Rightarrow 0010000B \Rightarrow 00100001B$



$S \Rightarrow \ldots \Rightarrow 00100001B \Rightarrow 001000011B$



$S \Rightarrow \ldots \Rightarrow 001000011B \Rightarrow 0010000111$



V jednotlivých krocích je kromě samotného stromu uvedeno použité pravidlo a také momentální stav derivace. Modře je vyznačena větná forma, kterou jsme v daném kroku vytvořili. Můžeme si všimnout, že v derivačním stromě vždy jde o obsah listů stromu v daném kroku. Derivační strom dané derivace je poslední v pořadí.

Úkoly

1. Podle gramatiky a uvedené derivace z příkladu 4.1 sestrojte derivační strom.
2. Podle zadané gramatiky vytvořte derivaci slova $ababa$ a k ní derivační strom.

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

   $S \to bAB \mid aBA \mid \varepsilon$
   $A \to abAab \mid \varepsilon$
   $B \to baBba \mid \varepsilon$
3. Podle gramatiky z předchozího příkladu vytvořte derivaci jakéhokoliv slova o délce alespoň 6 začínajícího symbolem b a k této derivaci sestrojte derivační strom.
4. Sestrojte gramatiky generující následující jazyky. V každém jazyce vyberte jakékoliv slovo o délce alespoň 3 symboly, vytvořte jeho derivaci a sestrojte k ní derivační strom.

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

   (b) $L_2 = \{0^n 11^n \ ;\ n \geq 3\} \cup \{\varepsilon\}$

   (c) $L_3 = \{0^i 10^i \ ;\ i \geq 1\} \cup \{1^i 01^i \ ;\ i \geq 1\}$

   (d) $L_4 = \{abc, bad, faa, diff\}$ (nezapomeňte, že na pravé straně pravidla může být jakýkoliv řetězec)

4.3 Úpravy bezkontextových gramatik

4.3.1 Převod na nezkracující bezkontextovou gramatiku

Nezkracující bezkontextová gramatika buď vůbec neobsahuje ε-pravidla (pokud v jazyce generovaném gramatikou není prázdné slovo), a nebo existuje jediné takové pravidlo, a to pro startovací symbol gramatiky, pak ale se startovací symbol nesmí vyskytovat na pravé straně žádného pravidla (to znamená, že v každé derivaci se vyskytuje pouze na jejím začátku, pak už ne).

Při převodu na nezkracující gramatiku odstraňujeme ε-pravidla tak, že „simulujeme“ jejich použití.

Příklad 4.4

K následující gramatice vytvoříme ekvivalentní gramatiku s nezkracujícími pravidly.
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to aaB \mid bAb$
$A \to aBa \mid \varepsilon$
$B \to AbAa \mid a$

Téměř všechna pravidla jsou nezkracující, je zde pouze jediné ε-pravidlo – $A \to \varepsilon$. Přepisovaný symbol A se nachází na pravé straně dvou pravidel – druhého a pátého.

Obě tato pravidla necháme a přidáme jejich varianty se „simulovaným“ použitím ε-pravidla:

$G' = (\{S, A, B\},\ \{a, b\},\ P',\ S)$
$S \to aaB \mid bAb \mid bb$
$A \to aBa$
$B \to AbAa \mid bAa \mid Aba \mid ba \mid a$

U prvního ze zpracovávaných pravidel máme jen jeden výskyt neterminálu A, tedy existuje pouze jediná další varianta (toto jediné A odstraníme – přepíšeme na ε). U druhého zpracovávaného pravidla jsou dva výskyty neterminálu A, proto musíme vytvořit variant více – odstraníme jen první A, odstraníme jen druhé, a nebo odstraníme obě.

Derivace v gramatice G' bude téměř stejná jako v gramatice G, ale kroky s použitím ε-pravidel jsou „přeskočeny“, například
v gramatice G: $S \Rightarrow aaB \Rightarrow aaAbAa \Rightarrow aabAa \Rightarrow aabaBaa \Rightarrow aabaaaa$
v gramatice G': $S \Rightarrow aaB \Rightarrow aabAa \Rightarrow aabaBaa \Rightarrow aabaaaa$

Příklad 4.5

K zadané gramatice vytvoříme ekvivalentní nezkracující gramatiku (tj. generující tentýž jazyk).
$G = (\{S, A\},\ \{a, b, c\},\ P,\ S)$
$S \to aaS \mid bbA \mid \varepsilon$
$A \to cSbSc \mid \varepsilon$

Gramatika generuje také prázdné slovo. Proto nejdřív použijeme stejný postup jako v předchozím příkladu, ale pak ještě musíme zajistit, aby výsledná gramatika také generovala prázdné slovo a přitom zůstala nezkracující. Nejdřív vytvoříme pomocnou gramatiku G', která nemá žádná ε-pravidla, její jazyk je $L(G') = L(G) - \{\varepsilon\}$.
$G' = (\{S, A\},\ \{a, b, c\},\ P',\ S)$
$S \to aaS \mid aa \mid bbA \mid bb$
$A \to cSbSc \mid cbSc \mid cSbc \mid cbc$

Přidáme nový startovací symbol a vytvoříme gramatiku ekvivalentní původní gramatice.
$G'' = (\{S', S, A\},\ \{a, b, c\},\ P'',\ S')$
$S' \to S \mid \varepsilon$
$S \to aaS \mid aa \mid bbA \mid bb$
$A \to cSbSc \mid cbSc \mid cSbc \mid cbc$
Derivace v gramatice G: $S \Rightarrow aaS \Rightarrow aaaaS \Rightarrow aaaa$
Derivace v gramatice G'': $S' \Rightarrow S \Rightarrow aaS \Rightarrow aaaa$

Příklad 4.6

Někdy se může stát, že „simulací“ ε-pravidla vytvoříme další ε-pravidlo. Pak je třeba postup provádět rekurzívně tak dlouho, dokud nejsou všechna ε-pravidla odstraněna.
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to bA \mid aS \mid bB \mid a$
$A \to baB \mid \varepsilon$
$B \to AA \mid b$

Po odstranění pravidla $A \to \varepsilon$ dostaneme tuto gramatiku:
$G' = (\{S, A, B\},\ \{a, b\},\ P',\ S)$
$S \to bA \mid b \mid aS \mid bB \mid a$
$A \to baB$
$B \to AA \mid A \mid \varepsilon \mid b$

Odstraněním obou symbolů A v pravidle $B \to AA$ vzniklo další ε-pravidlo, které je také třeba odstranit:
$G'' = (\{S, A, B\},\ \{a, b\},\ P'',\ S)$
$S \to bA \mid b \mid aS \mid bB \mid b \mid a$
$A \to baB \mid ba$
$B \to AA \mid A \mid b$

Rekurze při řešení pravidel se zbavíme tak, že ji přeneseme jinam. Můžeme použít řešení podobné tomu z redukce gramatiky – vytvoříme (rekurzívně) množinu N_ε všech neterminálů, které lze (třeba i po více než jednom kroku) přepsat na prázdné slovo ε.

V množině $N_{\varepsilon,0}$ bude pouze prázdný řetězec, tj. $N_{\varepsilon,0} = \{\varepsilon\}$. V dalších krocích do této množiny přidáváme neterminály, které lze přepsat na řetězec skládající se pouze ze symbolů, které byly do této množiny přidány v předchozích krocích, tj.
$N_{\varepsilon,i} = N_{\varepsilon,i-1} \cup \{X \in N \mid X \to \alpha,\ \alpha \in N_{\varepsilon,i-1}\}$.

Příklad 4.7

Podle pravidel gramatiky
$G = (\{S, A, B, C, D\},\ \{a, b, c, d\},\ P,\ S)$
$S \to aAa \mid aa$
$A \to BC \mid b$
$B \to aC \mid cD \mid \varepsilon$
$C \to AAa \mid \varepsilon$
$D \to AAB \mid d$
vytvoříme tyto množiny:
$N_{\varepsilon,0} = \{\varepsilon\}$
$N_{\varepsilon,1} = \{B, C\}$ podle $B \to \varepsilon,\ C \to \varepsilon$, prázdné slovo už nezařazujeme
$N_{\varepsilon,2} = \{B, C, A\}$ podle $A \to BC$
$N_{\varepsilon,3} = \{B, C, A, D\}$ podle $D \to AAB$

V dalším kroku již nelze další neterminál přidat (ostatně všechny už v množině jsou), proto $N_\varepsilon = N_{\varepsilon,3} = \{B, C, A, D\}$.

Množiny použijeme takto: pokud je na pravé straně pravidla některý z neterminálů obsažených v množině N_ε, pak vytvoříme další pravidla se „simulovaným“ použitím ε-pravidel na tento neterminál, ε-pravidla odstraníme.

Postup je prakticky stejný jako předchozí, ale zpracování gramatiky již není třeba provádět rekurzívně (rekurze byla použita při generování množiny N_ε). Fialovou barvou jsou zvýrazněny neterminály obsažené v množině N_ε (a tedy vytvoříme pravidlo, ve kterém je odstraníme), modrou barvou pak nová pravidla.

$S \to aAa \mid aa \mid aa$ (vlastně máme dvě stejná pravidla, jedno může být odstraněno)
$A \to BC \mid C \mid B \mid b$
$B \to aC \mid a \mid cD \mid c$
$C \to AAa \mid Aa \mid a$
$D \to AAB \mid AB \mid B \mid d$

Příklad 4.8

Postup předvedený v předchozím příkladu použijeme na gramatiku z příkladu 4.6.
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to bA \mid aS \mid bB \mid a$
$A \to baB \mid \varepsilon$
$B \to AA \mid b$

Opět rekurzívně vytvoříme množinu N_ε:
$N_{\varepsilon,0} = \{\varepsilon\}$
$N_{\varepsilon,1} = \{A\}$
$N_{\varepsilon,2} = \{A, B\} = N_{\varepsilon,3} = N_\varepsilon$

Vychází nám tato gramatika:
$G' = (\{S, A, B\},\ \{a, b\},\ P',\ S)$
$S \to bA \mid b \mid aS \mid bB \mid b \mid a$
$A \to baB \mid ba$
$B \to AA \mid A \mid b$

Oproti původnímu řešení není třeba několikrát přepisovat pravidla gramatiky.

Úkoly

K následujícím gramatikám vytvořte ekvivalentní nezkracující gramatiky.

$G_1 = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to aSbA \mid ab$
$A \to aAbA \mid \varepsilon$
$B \to bAbB \mid b$

$G_2 = (\{S, A\},\ \{0, 1\},\ P,\ S)$
$S \to S1S1 \mid A \mid \varepsilon$
$A \to 0A0 \mid 1$

$G_3 = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to BAa \mid aS \mid \varepsilon$
$A \to aA \mid \varepsilon$
$B \to bBA \mid \varepsilon$

$G_4 = \{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to AB \mid BA \mid b$
$A \to aaA \mid \varepsilon$
$B \to AA \mid bS \mid a$

4.3.2 Redukce gramatiky

Účelem redukce gramatiky je snížení počtu neterminálů při zachování generovaného jazyka. Odstraňujeme

- *nadbytečné neterminály* (tj. neterminály, ze kterých nelze vygenerovat žádné terminální slovo),
- *nedostupné symboly* (terminální i neterminální – nelze je vygenerovat ze startovacího symbolu).

Zachováváme toto pořadí – nejdřív nadbytečné a pak teprve nedostupné. Používáme podobný postup jako při redukci množiny stavů konečného automatu.

Příklad 4.9

Redukujeme následující gramatiku:
$G = (\{S, A, B, C, D\},\ \{a, b, c, d\},\ P,\ S)$
$S \to bS \mid bD \mid \varepsilon$
$A \to aCB \mid bAD$
$B \to dB \mid d$
$C \to bCC \mid aAbB$
$D \to bA \mid cDb \mid aS \mid \varepsilon$

Nejdřív odstraníme nadbytečné neterminály. Rekurzívně vytvoříme množinu všech symbolů, které jsou buď terminální a nebo z nich lze vygenerovat terminální řetězec (v případě, že jde o neterminál). Bází rekurze je množina všech terminálů. V dalších krocích přidáváme neterminály z levých stran pravidel, na jejichž pravé straně jsou pouze symboly, které jsme do naší množiny přidali v předchozích krocích (a nebo pro ně existuje ε-pravidlo).
$N_0 = \{a, b, c, d\}$
$N_1 = \{a, b, c, d, S, B, D\}$ (podle pravidel $S \to \varepsilon,\ B \to d,\ D \to \varepsilon$)
$N_2 = \{a, b, c, d, S, B, D\}$ (podle pravidel $S \to bS \mid bD,\ B \to dB,\ D \to cDb \mid aS$)

Protože jsme do množiny N_2 oproti množině N_1 nic dalšího nepřidali, rekurze končí. Množina neterminálů bude $N \cap N_2 = \{S, B, D\}$, vyřadíme neterminály A a C včetně všech pravidel, která je obsahují na levé nebo pravé straně. Gramatika po odstranění nadbytečných symbolů je následující:
$G' = (\{S, B, D\},\ \{a, b, c, d\},\ P',\ S)$
$S \to bS \mid bD \mid \varepsilon$
$B \to dB \mid d$
$D \to cDb \mid aS \mid \varepsilon$

Zbývá odstranit nedostupné symboly. Postupujeme opět rekurzívně. Vytvoříme množinu symbolů, které se nacházejí v nějaké větné formě v derivaci ze startovacího symbolu. Začneme množinou obsahující pouze startovací symbol, v každém kroku přidáváme symboly, které se nacházejí na pravých stranách pravidel přepisujících symboly zařazené v předchozích krocích.
$V_0 = \{S\}$
$V_1 = \{S, b, D\}$ (podle pravidel $S \to bS \mid bD$)
$V_2 = \{S, b, D, c, a\}$ (podle pravidel $D \to cDb \mid aS$)
$V_3 = V_2$

Z postupu vyplývá, že pokud v některém kroku přidáme pouze terminální symboly, v následujícím kroku již nelze nic přidat (protože terminály se nepřepisují žádným pravidlem) a rekurze končí. Je

zřejmé, že nedostupný neterminál je jeden (B) a terminál taktéž jeden (d). Redukovaná gramatika (tj. již bez nadbytečných i nedostupných symbolů) je následující:
$G'' = (\{S, D\},\ \{a, b, c\},\ P'',\ S)$
$S \to bS \mid bD \mid \varepsilon$
$D \to cDb \mid aS \mid \varepsilon$

Úkoly

Redukujte tyto gramatiky:

$G_1 = (\{S, A, B, C, D\},\ \{a, b, c, d\},\ P,\ S)$
$S \to aBa \mid bA \mid c \mid ABc$
$A \to aA \mid dD \mid bDa$
$B \to aB \mid bA \mid bS$
$C \to aSc \mid cC \mid c$
$D \to dD \mid aAb$

$G_2 = (\{S, A, B, C, D, E\},\ \{0, 1, 2, 3\},\ P,\ S)$
$S \to AB0 \mid 2E \mid A1 \mid \varepsilon$
$A \to 0A0 \mid 1B \mid 1S$
$B \to 01B \mid 1D \mid EB$
$C \to 2A \mid 1S \mid 0$
$D \to D1 \mid 3B$
$E \to 0E1 \mid 1B$

4.3.3 Odstranění jednoduchých pravidel

Jednoduchá pravidla jsou taková pravidla, na jejichž pravé straně máme jen jediný symbol, a to neterminál, například $A \to B$. Jednoduchá pravidla zbytečně prodlužují výpočet a v některých algoritmech mohou být překážkou při programování.

Pokud tento typ pravidel chceme odstranit, můžeme jednoduše nahradit symbol na pravé straně jednoduchého pravidla celou množinou pravidel, na která se tento symbol přepisuje. Například pokud existují pravidla $B \to \alpha \mid \beta \mid \gamma$, pak jednoduché pravidlo $A \to B$ nahradíme pravidly $A \to \alpha \mid \beta \mid \gamma$ (v derivaci to bude znamenat zkrácení odvození o jeden krok – zpracování jednoduchého pravidla).

Protože touto náhradou můžeme opět pro daný neterminál dostat jednoduché pravidlo, postup je rekurzivní a lze ho zjednodušit přenesením rekurze na množiny neterminálů podobně jako v předchozích postupech. Vytváříme množiny N_A postupně pro všechny neterminály $A \in N$, které inicializujeme jednoprvkovou množinou obsahující neterminál v označení množiny (tj. v tomto případě A). Jde o množiny neterminálů, jejichž pravidla se pomocí jednoduchých pravidel mají postupně připsat do množiny pravidel pro daný neterminál A.

Příklad 4.10

Odstraníme jednoduchá pravidla v následující gramatice:
$G = (\{S, A, B, C\},\ \{a, b\},\ P,\ S)$
$S \to aBa \mid A$
$A \to aA \mid B$
$B \to bB \mid AB \mid C \mid \varepsilon$
$C \to bA \mid b$

Rekurzívně vytvoříme množiny $N_S,\ N_A,\ N_B,\ N_C$. Na začátku rekurze, v bázi, bude v dané množině pouze ten symbol, pro který množinu tvoříme, tj. například $N_{S,0} = \{S\}$.
$N_{S,0} = \{S\}$
$N_{S,1} = \{S, A\}$ (podle $S \to A$)
$N_{S,2} = \{S, A, B\}$ (podle $A \to B$)
$N_{S,3} = \{S, A, B, C\} = N_{S,4} = N_S$ (podle $B \to C$)
$N_{A,0} = \{A\}$
$N_{A,1} = \{A, B\}$ (podle $A \to B$
$N_{A,2} = \{A, B, C\} = N_{A,3} = N_A$ (podle $B \to C$)
$N_{B,0} = \{B\}$
$N_{B,1} = \{B, C\} = N_{B,2} = N_A$ (podle $B \to C$)
$N_{C,0} = \{C\} = N_{C,1} = N_C$

Z gramatiky odstraníme všechna jednoduchá pravidla. Po jejich odstranění pak použijeme vytvořené množiny – pokud například pro neterminál A je $N_A = \{A, B, C\}$, pak v gramatice pro neterminál A použijeme všechna pravidla, která v původní gramatice byla pro neterminály A, B, C.
$G' = (\{S, A, B, C\},\ \{a, b\},\ P',\ S)$
$S \to aBa \mid aA \mid bB \mid AB \mid \varepsilon \mid bA \mid b$
$A \to aA \mid bB \mid AB \mid \varepsilon \mid bA \mid b$
$B \to bB \mid AB \mid \varepsilon \mid bA \mid b$
$C \to bA \mid b$

Pokud nám vadí pouze jednoduchá pravidla jako taková, předchozí postup je dostačující. Jestliže však je překážkou samotné chování jednoduchých pravidel (například prodlužování derivace), je třeba předem odstranit ε-pravidla. Například v gramatice z příkladu 4.11 po odstranění jednoduchých pravidel existuje derivace $S \Rightarrow AB \Rightarrow A \Rightarrow \ldots$, kde nacházíme ekvivalent použití jednoduchého pravidla $S \to A$ z původní gramatiky.

Příklad 4.11

Gramatiku z příkladu 4.11 předem upravíme – převedeme na nezkracující. Pak teprve odstraníme jednoduchá pravidla.
$G = (\{S, A, B, C\},\ \{a, b\},\ P,\ S)$
$S \to aBa \mid A$
$A \to aA \mid B$
$B \to bB \mid AB \mid C \mid \varepsilon$
$C \to bA \mid b$

Po převodu na nezkracující gramatiku:
$N_\varepsilon = \{B, A, S\}$
$G' = (\{S, A, B, C\},\ \{a, b\},\ P',\ S)$
$S \to aBa \mid aa \mid A \mid \varepsilon$
$A \to aA \mid a \mid B$

$B \to bB \mid b \mid AB \mid A \mid B \mid C$
$C \to bA \mid b$

Pravidlo $S \to \varepsilon$ můžeme nechat, protože se S nevyskytuje na pravé straně žádného pravidla (není nutné vytvářet nový startovací symbol). Odstraníme jednoduchá pravidla:
$N_S = \{S, A, B, C\}$
$N_A = \{A, B, C\}$
$N_B = \{B, A, C\}$ (všimněte si rozdílu oproti podobné množině v příkladu 4.11)
$N_C = \{C\}$

$G'' = (\{S, A, B, C\},\ \{a, b\},\ P'',\ S)$
$S \to aBa \mid aa \mid aA \mid a \mid \varepsilon \mid aA \mid a \mid bB \mid b \mid AB \mid bA \mid b$
$A \to aA \mid a \mid bB \mid b \mid AB \mid bA \mid b$
$B \to bB \mid b \mid AB \mid aA \mid a \mid bA \mid b$
$C \to bA \mid b$

Úkoly

Odstraňte jednoduchá pravidla z následujících gramatik:

$G_1 = (\{S, A, B\},\ \{0, 1\},\ P,\ S)$
$S \to 0A1 \mid B \mid \varepsilon$
$A \to 1S \mid S \mid B$
$B \to 1A \mid S \mid 0$

$G_2 = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to aA \mid bB \mid A$
$A \to bAb \mid S \mid \varepsilon$
$B \to aBa \mid A \mid a$

4.3.4 Gramatika bez cyklu a vlastní gramatika

Gramatika bez cyklu je taková gramatika, kde neexistuje žádná derivace $A \Rightarrow^+ A$ pro jakýkoliv neterminál A. Už z definice je zřejmé, jak gramatiku upravit, aby splňovala tuto vlastnost – stačí ji převést na nezkracující gramatiku (bez ε-pravidel) a odstranit jednoduchá pravidla.

Vlastní gramatika je gramatika bez cyklu, nezkracující a bez nadbytečných symbolů. Postup je tedy následující:

- převedeme gramatiku na nezkracující,
- odstraníme jednoduchá pravidla,
- odstraníme nadbytečné symboly postupem, který již známe z redukce gramatiky.

4.3.5 Levá a pravá rekurze

Gramatika je rekurzivní zleva, pokud v ní existuje derivace $A \Rightarrow^+ A\alpha$, gramatika je rekurzívní zprava, pokud v ní existuje derivace $A \Rightarrow^+ \alpha A$, A je některý neterminál gramatiky, α je jakýkoliv řetězec z terminálů a neterminálů.

Levá nebo pravá rekurze nám může bránit v některých dalších úpravách a nebo v naprogramování postupu popsaného gramatikou. Obvykle nám vadí jen levá a nebo jen pravá rekurze,

proto jejich odstranění řešíme jednoduše převodem na tu, která nám nevadí. Při odstraňování rekurze vycházíme z toho, že k témuž řetězci lze dojít více různými způsoby. Ukážeme si odstranění *přímé rekurze*, jejíž existence je patrná přímo z pravidla.

Příklad 4.12

Následující gramatiku zbavíme přímé levé rekurze.
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aAb \mid b$
$A \to Aa \mid bB \mid ab$
$B \to Ba \mid Bb \mid ca$

Levá rekurze je v pravidlech $A \to Aa$ a dále $B \to Ba \mid Bb$. Nejdřív vyřešíme první z těchto pravidel – pro neterminál A. Množinu pravidel $A \to Aa \mid bB \mid ab$ nahradíme jinou množinou, která generuje tentýž řetězec. Jak může vypadat derivace z neterminálu A?
$A \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow^* Aa^i \Rightarrow bBa^i$ (generujeme zprava doleva, a to nejdřív rekurzívním pravidlem, rekurze je pak ukončena vlevo některým nerekurzívním pravidlem).

Stejný řetězec lze generovat pravou rekurzí, a to přidáním nového neterminálu a úpravou pravidel (původní pravidla jsou $A \to Aa \mid bB \mid ab$):
$A \to bBA' \mid abA' \mid bB \mid ab$
$A' \to aA' \mid a$

Derivace ze symbolu A pak vypadá následovně:
$A \Rightarrow bBA' \Rightarrow bBaA' \Rightarrow bBaaA' \Rightarrow^* bBa^{i-1}A' \Rightarrow bBa^i$

Zbývá upravit pravidla pro neterminál B – $B \to Ba \mid Bb \mid ca$, nahradíme je pravidly
$B \to caB' \mid ca$
$B' \to aB' \mid bB' \mid a \mid b$

Vytvořili jsme tedy ekvivalentní gramatiku bez přímé levé rekurze:
$G' = (\{S, A, A', B, B'\},\ \{a, b, c\},\ P',\ S)$
$S \to aAb \mid b$
$A \to bBA' \mid abA' \mid bB \mid ab$
$A' \to aA' \mid a$
$B \to caB' \mid ca$
$B' \to aB' \mid bB' \mid a \mid b$

Uvedený postup je použitelný pouze na *vlastní gramatiku* (tj. gramatiku bez cyklu, nezkracující, bez nadbytečných symbolů). Gramatika z předchozího příkladu je vlastní, proto nebylo třeba ji předem upravovat.

Příklad 4.13

Odstraníme přímou levou rekurzi v následující gramatice:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to bAa \mid c \mid A$
$A \to Ba \mid b \mid Abb$
$B \to aBb \mid Bbc \mid ca \mid \varepsilon$

Tato gramatika není vlastní. Je třeba nejdřív převést ji na nezkracující, odstranit jednoduchá pravidla a teprve potom můžeme odstranit levou rekurzi.

1. Odstraníme ε-pravidla (resp. jediné, $B \to \varepsilon$):

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

   $S \to bAa \mid c \mid A$
   $A \to Ba \mid a \mid b \mid Abb$
   $B \to aBb \mid ab \mid Bbc \mid bc \mid ca$
2. Odstraníme jednoduché pravidlo $S \to A$:

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

   $S \to bAa \mid c \mid Ba \mid a \mid b \mid Abb$
   $A \to Ba \mid a \mid b \mid Abb$
   $B \to aBb \mid ab \mid Bbc \mid bc \mid ca$
3. Odstraníme přímou levou rekurzi. Pravidla $A \to Ba \mid a \mid b \mid Abb$ nahradíme pravidly

   $A \to BaA' \mid aA' \mid bA' \mid Ba \mid a \mid b$
   $A' \to bbA' \mid bb$

   Dále pravidla $B \to aBb \mid ab \mid Bbc \mid bc \mid ca$ nahradíme pravidly

   $B \to aBbB' \mid abB' \mid aB' \mid |bcB' \mid caB' \mid aBb \mid ab \mid bc \mid ca$
   $B' \to bcB' \mid bc$

Vytvořili jsme tuto gramatiku:
$G''' = (\{S, A, A', B, B'\},\ \{a, b, c\},\ P''',\ S)$
$S \to bAa \mid c \mid Ba \mid a \mid b \mid Abb$
$A \to BaA' \mid aA' \mid bA' \mid Ba \mid a \mid b$
$A' \to bbA' \mid bb$
$B \to aBbB' \mid abB' \mid aB' \mid |bcB' \mid caB' \mid aBb \mid ab \mid bc \mid ca$
$B' \to bcB' \mid bc$

Příklad 4.14

Odstraníme přímou pravou rekurzi v této gramatice:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aAb \mid AB$
$A \to abA \mid c \mid Sc$
$B \to bbB \mid cB \mid a$

Jde o vlastní gramatiku, nemusíme ji předem do tohoto tvaru upravovat. Postupujeme stejně jako u odstranění levé rekurze, jen zaměníme levou a pravou stranu. Přidáme nové neterminály a upravíme pravidla.

Z neterminálu A může být například tato derivace:
$A \Rightarrow abA \Rightarrow ababA \Rightarrow^* (ab)^i A \Rightarrow (ab)^i c$

Pravidla $A \to abA \mid c \mid Sc$ nahradíme pravidly
$A \to A'c \mid A'Sc \mid c \mid Sc$
$A' \to A'ab \mid ab$

Derivace z neterminálu A se změní:
$A \Rightarrow A'c \Rightarrow A'abc \Rightarrow A'ababc \Rightarrow^* A'(ab)^{i-1}c \Rightarrow (ab)^i c$

Vygenerovaný řetězec je tentýž, ale zatímco u pravé rekurze byl generován zleva doprava, u levé rekurze je generován opačně – zprava doleva.

Podobně upravíme pravidla $B \to bbB \mid cB \mid a$:
$B \to B'a \mid a$
$B' \to B'bb \mid B'c \mid bb \mid c$

Vychází nám tato gramatika:
$G' = (\{S, A, A', B, B'\},\ \{a, b, c\},\ P',\ S)$
$S \to aAb \mid AB$
$A \to A'c \mid A'Sc \mid c \mid Sc$
$A' \to A'ab \mid ab$
$B \to B'a \mid a$
$B' \to B'bb \mid B'c \mid bb \mid c$

Rekurze může být také nepřímá, například v pravidlech
$A \to Bba \mid a$
$B \to Aab \mid b$

Levou rekurzi, včetně nepřímé, odstraníme tak, že pravidla převedeme do tvaru, kdy pravá strana pravidla začíná neterminálem pouze za přesně určených okolností. Postup:

1. Stanovíme pořadí neterminálů. Můžeme si je například označit indexy, přejmenovat nebo jednoduše určit jejich pořadí. Nechť pořadí neterminálů je $(A_1, A_2, \ldots, A_n)$. Účelem algoritmu je upravit pravidla tak, aby mohla začínat pouze neterminály s vyšším indexem než je index přepisovaného neterminálu. Tím zcela odstraníme levou rekurzi.
2. Pro neterminály $A_i, A_j,\ 1 \leq i, j \leq n$ postupně transformujeme pravidla. Pro první krok stanovíme $i = 1, j = 1$.
   - Pokud $j < i$ a existuje pravidlo $A_i \to A_j\alpha$, kde α je jakýkoliv řetězec (tj. pravidlo pro A_i začíná neterminálem A_j), pak symbol A_j v pravidle nahradíme postupně všemi pravými stranami pravidel pro A_j, například
     $A_i \to A_j abB$
     $A_j \to bDA_i a \mid CA_j a$
     První z uvedenených pravidel nahradíme pravidly $A_i \to bdA_i aabB \mid CA_j aabB$. Provedeme $j = j + 1$.
   - Pokud $j = i$, pak odstraníme přímou levou rekurzi podle postupu, který už známe. Provedeme $j = j + 1$.
   - Pokud $j > i$, provedeme $i = i + 1,\ j = 1$.
3. Bod 2 postupu provedeme pro všechna i, $1 \leq i \leq n$.

Aby byl postup použitelný, musí být uplatněn pouze na *vlastní gramatiku*. Postup pro pravou rekurzi je obdobný, jen zaměníme levou za pravou stranu.

Příklad 4.15

Odstraníme levou rekurzi v následující gramatice:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to aA \mid AB \mid b$
$A \to SBa \mid a$
$B \to Bb \mid Aa \mid c$

Přímá levá rekurze je zde jen v jednom pravidle, ale nepřímá rekurze se týká více pravidel. Gramatika je vlastní, není třeba ji do tohoto tvaru upravovat. Stanovíme pořadí neterminálů: $(A_1, A_2, A_3) = (S, A, B)$.

- $i = 1,\ j = 1$:
  není třeba upravovat, pravidla pro S neobsahují přímou rekurzi.
- $i = 2,\ j = 1$:
  jedno z pravidel pro A začíná neterminálem s „nižším indexem“, S; upravíme:

  $A \to aABa \mid ABBa \mid bBa \mid a$
- $i = 2,\ j = 2$:
  řešíme přímou rekurzi (protože $i = j$):

  $A \to aABaA' \mid bBaA' \mid aA' \mid aABa \mid bBa \mid a$
  $A' \to BBaA' \mid BBa$
- $i = 3,\ j = 1$:
  žádné z pravidel pro B nezačíná symbolem S
- $i = 3,\ j = 2$:
  změna se týká druhého pravidla pro symbol B (začíná symbolem A, jehož index je 2). Při nahrazení symbolu A v tomto pravidle použijeme již upravená pravidla pro A:

  $B \to Bb \mid aABaA'a \mid bBaA'a \mid aA'a \mid aABaa \mid bBaa \mid aa \mid c$
- $i = 3,\ j = 3$:
  odstraníme přímou rekurzi:

  $B \to aABaA'aB' \mid bBaA'aB' \mid aA'aB' \mid aABaaB' \mid bBaaB' \mid aaB' \mid cB' \mid$
  $\quad aABaA'a \mid bBaA'a \mid aA'a \mid aABaa \mid bBaa \mid aa \mid c$
  $B' \to bB' \mid b$

Celá gramatika bez levé rekurze:
$G' = (\{S, A, A', B, B'\},\ \{a, b, c\},\ P',\ S)$
$S \to aA \mid AB \mid b$
$A \to aABaA' \mid bBaA' \mid aA' \mid aABa \mid bBa \mid a$
$A' \to BBaA' \mid BBa$
$B \to aABaA'aB' \mid bBaA'aB' \mid aA'aB' \mid aABaaB' \mid bBaaB' \mid aaB' \mid cB' \mid$
$\quad aABaA'a \mid bBaA'a \mid aA'a \mid aABaa \mid bBaa \mid aa \mid c$
$B' \to bB' \mid b$

Odstraněním přímé rekurze rozšiřujeme množinu neterminálů, proto je nutné *dynamicky* upravovat také nadefinovanou posloupnost neterminálů určující pořadí. Neterminály s pravidly, které takto vytvoříme, je třeba zahrnout do algoritmu.

Příklad 4.16

Odstraníme levou rekurzi v následující gramatice:
$G = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to aA \mid bB$
$A \to BaA \mid ab$
$B \to BaA \mid b$

Postupujeme podle algoritmu. Stanovíme pořadí neterminálů na (S, A, B). Dále:

- $i = 1,\ j = 1$: není třeba upravovat, pravidla pro S neobsahují přímou rekurzi.
- $i = 2,\ j = 1, 2$: není třeba upravovat, pravidla pro A nezačínají symboly S a A.
- $i = 3,\ j = 1, 2$: není třeba upravovat, pravidla pro B nezačínají symboly S a A.
- $i = 3,\ j = 3$: odstraníme přímou rekurzi.

  $B \to bB' \mid b$
  $B' \to AaB' \mid Aa$

  Měníme posloupnost: $(A_1, A_2, A_3, A_4) = (S, A, B, B')$
- $i = 4,\ j = 1$: není třeba upravovat, pravidla pro B' nezačínají symbolem S.
- $i = 4,\ j = 2$:

  $B' \to BaAaB' \mid abaB' \mid BaAa \mid aba$
- $i = 4,\ j = 3$:

  $B' \to bB'aAaB' \mid baAaB' \mid abaB' \mid bB'aAa \mid baAa \mid aba$
- $i = 4,\ j = 4$: není třeba upravovat.

Výsledná gramatika:
$G' = (\{S, A, B, B'\},\ \{a, b\},\ P',\ S)$
$S \to aA \mid bB$
$A \to BaA \mid ab$
$B \to bB' \mid b$
$B' \to AaB' \mid Aa$
$B' \to bB'aAaB' \mid baAaB' \mid abaB' \mid bB'aAa \mid baAa \mid aba$

Úkoly

1. Odstraňte levou rekurzi v těchto gramatikách (zvažte, zda není třeba gramatiku předem upravit):

$G_1 = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$

$S \to aAB \mid b \mid SB \mid \varepsilon$
$A \to bA \mid Aac \mid AB \mid a$
$B \to BbA \mid c$

$G_2 = (\{S, A, B\},\ \{a, b\},\ P,\ S)$

$S \to AB \mid BA$
$A \to ABbA \mid bA \mid \varepsilon$
$B \to SA \mid a \mid BbA$

2. Odstraňte pravou rekurzi v těchto gramatikách (příp. gramatiku předem upravte):

$G_1 = (\{S, A, B\},\ \{0, 1\},\ P,\ S)$

$S \to 11A \mid 01B \mid \varepsilon$
$A \to 11A \mid B01 \mid 10$
$B \to 01B \mid 00AB \mid 1A1 \mid 01$

$G_2 = (\{S, A, B\},\ \{a, b\},\ P,\ S)$

$S \to aAb \mid abB \mid ASA$
$A \to bA \mid B \mid \varepsilon$
$B \to bbB \mid abAB \mid abb$

Kapitola 5

Zásobníkový automat

5.1 Co je to zásobníkový automat

Konečné automaty jsou užitečné (to zjistíme především v předmětu Překladače, kdy se naučíme používat stavové programování), nicméně nejsou zdaleka všemocné. Na některé úkoly nestačí, proto potřebujeme i složitější formy automatů/strojů. O něco složitější než konečný automat je automat zásobníkový.

5.1.1 Definice

Zásobníkový automat dostaneme tak, že

- konečný automat obohatíme o zásobníkovou pásku,
- zajistíme, aby výpočet byl řízen především podle této zásobníkové pásky,
- netrváme na tom, aby byl v každém kroku čten vstup.

Zásobníkový automat je uspořádaná sedmice $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

- Q je konečná neprázdná množina stavů,
- Σ je konečná neprázdná abeceda,
- Γ je konečná neprázdná *zásobníková* abeceda,
- δ je přechodová funkce definovaná níže,
- q_0 je počáteční stav automatu, $q_0 \in Q$,
- Z_0 je počáteční zásobníkový symbol, $Z_0 \in \Gamma$,
- F je množina koncových stavů, $F \subseteq Q$ (může být i prázdná).

Přechodová funkce δ, která popisuje činnost automatu, je zobrazení $\delta : \ Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow Q \times \Gamma^*$. To můžeme také zapsat jako
$\delta(q_i, a, Z) \ni (q_j, \gamma),\ q_i, q_j \in Q,\ a \in (\Sigma \cup \{\varepsilon\}),\ Z \in \Gamma,\ \gamma \in \Gamma^*$

Konfigurace výše definovaného zásobníkového automatu $\mathcal{A}$ je $Q \times \Sigma^* \times \Gamma^*$, také můžeme zapsat ve tvaru $(q, \alpha, \gamma),\ q \in Q,\ \alpha \in \Sigma^*,\ \gamma \in \Gamma^*$.

Počáteční konfigurace je (q_0, w, Z_0), kde w je slovo, které bylo dáno na vstup automatu. Koncová konfigurace závisí na typu zásobníkového automatu.

Přechod mezi konfiguracemi zásobníkového automatu je relace

$$(q_i, a\alpha, Z\beta) \vdash (q_j, \alpha, \gamma\beta) \quad \Longleftrightarrow \quad (q_j, \beta) \in \delta(q_i, a, Z)$$

kde $a \in (\Sigma \cup \{\varepsilon\}),\ Z \in (\Gamma \cup \{\varepsilon\}),\ \alpha \in \Sigma^*$.

Zásobníkový automat pracuje takto:

1. vyjme symbol na vrcholu zásobníku,
2. může/nemusí přečíst jeden symbol ze vstupní pásky, pokud přečte, posune se o políčko dál,
3. dále se rozhoduje podle
   - svého vnitřního stavu,
   - symbolu, který vyndal ze zásobníku,
   - pokud četl ze vstupní pásky, pak i podle přečteného symbolu,
4. akce automatu spočívá v
   - přechodu do některého dalšího stavu
   - a v uložení řetězce znaků do zásobníku.

5.1.2 Typy zásobníkových automatů

Rozeznáváme tyto základní typy zásobníkových automatů:

1. zásobníkový automat končící přechodem do koncového stavu,
2. zásobníkový automat končící prázdným zásobníkem.

Existuje také jejich kombinace – zásobníkový automat končící přechodem do koncového stavu při prázdném zásobníku.

Zásobníkový automat končící přechodem do koncového stavu značíme $\mathcal{A}_F$. Jeho koncová konfigurace má tento tvar:

$$(q_f, \varepsilon, \gamma),\ q_f \in F,\ \gamma \in \Gamma^*$$

Rozpoznávaný jazyk je

$$L(\mathcal{A}_F) = \{w \in \Sigma^* \ ;\ (q_0, w, Z_0) \vdash {}^*(q_f, \varepsilon, \gamma),\ q_f \in F,\ \gamma \in \Gamma^*\}$$

To znamená, že abychom mohli skončit, musíme přečíst celý vstup a dostat se do některého koncového stavu. Množina koncových stavů zde nebývá prázdná (kdyby byla, automat by rozpoznával prázdný jazyk).

Zásobníkový automat končící s prázdným zásobníkem značíme $\mathcal{A}_\emptyset$. Koncová konfigurace vypadá následovně:

$$(q, \varepsilon, \varepsilon),\ q \in Q$$

Rozpoznávaný jazyk je

$$L(\mathcal{A}_\emptyset) = \{w \in \Sigma^* \ ; \ (q_0, w, Z_0) \vdash {}^*(q, \varepsilon, \varepsilon), \ q \in Q\}$$

Abychom mohli skončit, musíme přečíst celý vstup a vyprázdnit zásobník. Koncové stavy nepotřebujeme, proto je množina koncových stavů obvykle prázdná.

Zásobníkový automat končící přechodem do koncového stavu při prázdném zásobníku značíme $\mathcal{A}_{F,\emptyset}$. Jeho koncová konfigurace je

$$(q_f, \varepsilon, \varepsilon), \ q_f \in F$$

Rozpoznávaný jazyk je

$$L(\mathcal{A}_{F,\emptyset}) = \{w \in \Sigma^* \ ; \ (q_0, w, Z_0) \vdash {}^*(q_f, \varepsilon, \varepsilon), \ q_f \in F\}$$

Je třeba splnit podmínky obou předchozích typů zároveň – abychom mohli skončit výpočet (úspěšně), je třeba přečíst celý vstup, vyprázdnit zásobník a navíc být v koncovém stavu.

Všechny tři typy zásobníkových automatů končí samozřejmě výpočet i tehdy, když nejsou v žádné koncové konfiguraci, ale do žádné další se nemohou dostat (přechodová funkce nedává možnost reagovat v daném stavu s daným obsahem zásobníku a vstupní pásky). V tomto případě však končíme s tím, že zpracovávané slovo nepatří do jazyka rozpoznávaného automatem.

Příklad 5.1

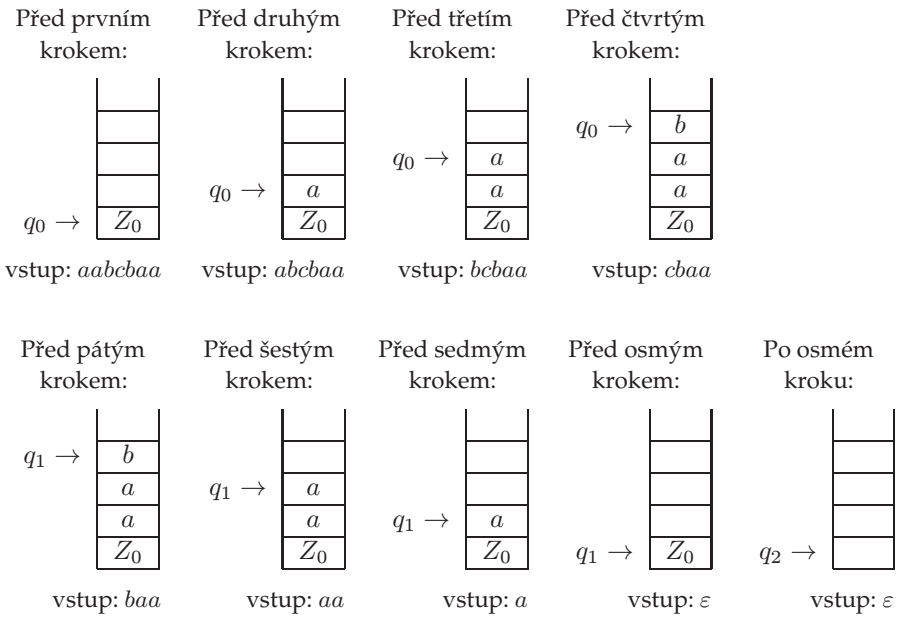
Sestrojíme zásobníkový automat (dále ZA) končící s prázdným zásobníkem rozpoznávající jazyk $L = \left\{wcw^R \ ; \ w \in \{a, b\}^*\right\}$

Vytvoříme zásobníkový automat rozpoznávající prázdným zásobníkem. To znamená, že končit budeme tehdy, když celý vstup bude přečtený a celý zásobník bude vyprázněný (včetně symbolu konce zásobníku Z_0). Množina koncových stavů bude prázdná, protože žádný koncový stav nepotřebujeme.

Automat bude pracovat takto:

- v první fázi bude číst obsah vstupu (první polovina slova) a ukládat do zásobníku (vždy co v každém kroku vyjmeme, vrátíme do zásobníku zároveň se symbolem ze vstupu, tedy ukládáme dva symboly), jsme ve stavu q_0,
- díky principu zásobníku (čteme v opačném pořadí, než jak byly symboly uloženy) je ukládaná první polovina slova zároveň zrcadlově převrácena,
- když na vstupu narazíme na c (hranice, polovina slova), přejdeme do stavu q_1 a tím změníme způsob práce automatu,
- když jsme ve stavu q_1, nic do zásobníku neukládáme, symbol, který v každém kroku vyjmeme, porovnáme s tím, co je na vstupu – když souhlasí, můžeme pokračovat (tj. v každém kroku se posuneme na vstupu a zároveň ubereme symbol ze zásobníku).

Následuje nákres s obsahem zásobníku, vstupu a stavem v jednotlivých krocích výpočtu:



V plné specifikaci uvedeme stavy, abecedu, zásobníkovou abecedu, počáteční stav, symbol konce zásobníku, δ-funkci a množinu koncových stavů (zde prázdnou), dále upřesníme předpis δ-funkce:
$\mathcal{A}_\emptyset = (\{q_0, q_1\},\ \{a, b\},\ \{a, b, Z_0\},\ q_0,\ Z_0,\ \delta,\ \emptyset)$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$	na začátku výpočtu, slovo začíná a
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$	na začátku výpočtu, slovo začíná b
$\delta(q_0, a, X) = (q_0, aX),\ X \in \{a, b\}$	zatím jen načítáme a ukládáme do zásobníku
$\delta(q_0, c, X) = (q_1, X),\ X \in \{a, b\}$	jsme na hranici
$\delta(q_1, a, a) = (q_1, \varepsilon)$	shoda a v obou polovinách slova
$\delta(q_1, b, b) = (q_1, \varepsilon)$	shoda b v obou polovinách slova
$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$	skončili jsme na vstupu i v zásobníku

Ukázka výpočtu automatu na slovo $abcba$:

$(q_0, abcba, Z_0) \vdash\ (q_0, bcba, aZ_0) \vdash$	q_0 : přenášíme do zásobníku obsah vstupu
$\vdash\ (q_0, cba, baZ_0) \vdash$	hraniční bod, přejdeme do módu q_1
$\vdash\ (q_1, ba, baZ_0) \vdash\ (q_1, a, aZ_0) \vdash$	q_1 : jen vybíráme ze zásobníku a porovnáváme
$\vdash\ (q_1, \varepsilon, Z_0) \vdash\ (q_1, \varepsilon, \varepsilon)$	konec

---

Zásobníkový automat končící v koncovém stavu (a vlastně i „hybridní“ typ) bychom z předešlého vytvořili jednoduše – stačí poslední část definice δ funkce přepsat takto:
$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$ s tím, že $Q = \{q_0, q_1, q_2\},\ F = \{q_2\}$.

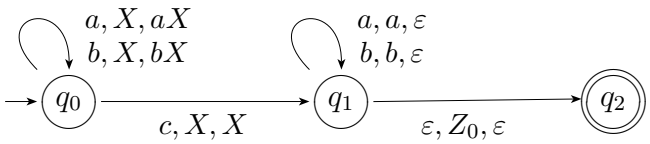
Příklad 5.2

Vytvoříme zásobníkový automat končící v koncovém stavu reprezentovaný stavovým diagramem pro jazyk z příkladu 5.1
$L = \left\{wcw^R\ ;\ w \in \{a, b\}^*\right\}$.

Narozdíl od konečných automatů, u zásobníkových nám nestačí ohodnotit šipky pouze symbolem načítaným ze vstupní pásky. Musíme vždy zadat tři údaje (ve správném pořadí!)

- symbol načtený ze vstupu (nebo ε, když ze vstupu nic nenačítáme),
- symbol, který vyjímáme ze zásobníku (nesmí zde být ε!, v každém kroku musíme nějaký symbol vyndat),
- řetězec, který ukládáme do zásobníku.

Diagram vytvoříme podle δ funkce v příkladu 5.1, jen navíc zakomponujeme změnu zahrnutou v poznámce nad tímto příkladem. Výsledný diagram vidíme na obrázku níže.



$X \in \{a, b, Z_0\}$

U konečných automatů byl diagram ještě celkem použitelný, ale u zásobníkových automatů ho již nepoužíváme (ve složitějších případech je naprosto nepřehledný). Typicky používáme zápis pomocí δ-funkce.

Úkoly

1. Podle postupů v předchozích příkladech sestrojte δ-funkci automatu *končícího v koncovém stavu* pro jazyk $L = \left\{wcw^R \ ; \ w \in \{a, b\}^*\right\}$.
2. Sestrojte zásobníkový automat pro jazyk $L = \{a^n b^n \ ; \ n \geq 1\}$.

   Nápověda: první polovinu slova (stav q_0) čtěte a přitom ukládejte do zásobníku, v případě druhé poloviny slova (stav q_1, tam přejdeme, když narazíme na první b) ze zásobníku vybírejte uložené symboly a jejich počet srovnávejte s čtenými symboly b (za každé a v zásobníku musí být jedno b na vstupu). Pozor na ukončení – vyzkoušejte, zda automat spolehlivě funguje (musí přijmout například slova ab a $aabb$, ale odmítnout prázdné slovo).

5.2 Vytváříme zásobníkový automat

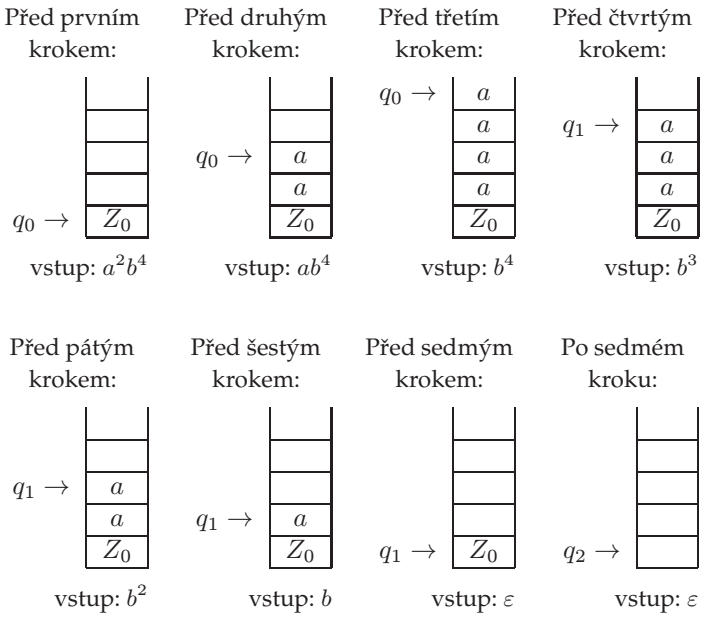
Při konstrukci zásobníkového automatu si především musíme promyslet, jakým způsobem bude ve kterém kroku zacházeno se zásobníkem.

Příklad 5.3

Sestrojíme zásobníkový automat rozpoznávající jazyk $L = \{a^n b^{2n} \ ; \ n \geq 0\}$

Můžeme postupovat podobně, jak bylo doporučeno v předchozím úkolu. Ale pozor – počet symbolů b je dvojnásobný. Aby nám „nenadbývaly“ symboly na vstupu, zvolíme δ-funkci následovně:

- stav q_0: pokud je na vstupu a (a v zásobníku cokoliv), uložíme do zásobníku původní vyjmutý symbol a dále dva symboly a (tj. řetězec aa),
- stav q_0: pokud je na vstupu b (na vrcholu zásobníku musí být a, protože jinak by slovo nepatřilo do rozpoznávaného jazyka), symbol a již do zásobníku nebudeme vracet, „spárujeme“ ho s přečteným symbolem b, přejdeme do stavu q_1,
- stav q_1: pokud je na vstupu b (na vrcholu zásobníku musí být a), reagujeme stejně jako v předchozím případě,
- končíme s prázdným zásobníkem, do jazyka patří i prázdné slovo.



$\mathcal{A} = (\{q_0, q_1\},\ \{a, b\},\ \{Z_0, a\},\ \delta,\ q_0,\ Z_0,\ \emptyset)$
$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$
$\delta(q_0, a, a) = (q_0, aaa)$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

Poslední dva předpisy přechodové funkce určují chování vedoucí k ukončení výpočtu. V okamžiku, kdy ze zásobníku vyjmeme Z_0 (za kterým už žádný symbol být nemůže), nelze dál pokračovat, protože v každém kroku výpočtu je nutno vyjmout symbol ze zásobníku (není co vyjmout).

Podle sestrojeného automatu zpracujeme několik slov (to bychom měli vždy udělat, abychom měli jistotu, že automat pracuje správně):
$(q_0, aabbbb, Z_0) \vdash (q_0, abbbb, aaZ_0) \vdash (q_0, bbbb, aaaaZ_0) \vdash (q_1, bbb, aaaZ_0) \vdash (q_1, bb, aaZ_0) \vdash$
$\vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$
$(q_0, abb, Z_0) \vdash (q_0, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_1, \varepsilon, \varepsilon)$

$(q_0, \varepsilon, Z_0) \vdash (q_0, \varepsilon, \varepsilon)$

$(q_0, ab, Z_0) \vdash (q_0, b, aaZ_0) \vdash (q_1, \varepsilon, aZ_0)$ konec, slovo $ab \notin L$, tato konfigurace není koncová

$(q_0, abbb, Z_0) \vdash (q_0, bbb, aaZ_0) \vdash (q_1, bb, aZ_0) \vdash (q_1, b, Z_0) \vdash (q_1, b, \varepsilon)$ opět nejde o koncovou konfiguraci, proto $abbb \notin L$ (všimněte si posledního provedeného kroku odvození)

Úkoly

1. Sestrojte zásobníkový automat rozpoznávající jazyk

   $L = \{a^n b^n c^k \ ; \ n, k \geq 1\}$.

   Nápověda: slova se skládají ze tří částí. První dvě části musejí být stejně dlouhé, tedy pro ně musíme použít synchronizaci zásobníkem (stav q_0: symboly a čteme a ukládáme do zásobníku, stav q_1: symboly b čteme a srovnáváme se zásobníkem, stav q_2: už zásobník nepotřebujeme, jen čteme symboly c tak dlouho, dokud jsou na vstupu).

   Nezapomeňte ověřit, zda automat přijímá slova abc, a^2b^2c, abc^2 (to jen pro kontrolu), a jestli nepřijímá slova ε, a, ab, b, c, ac (ta nepatří do jazyka rozpoznávaného automatem).
2. Sestrojte zásobníkový automat rozpoznávající jazyk

   $L = \{a^n b^k c^n \ ; \ n, k \geq 0\}$.
3. Sestrojte zásobníkový automat rozpoznávající jazyk

   $L = \{(01)^n (10)^n \ ; \ n \geq 0\}$.
4. Sestrojte zásobníkový automat rozpoznávající jazyk

   $L = \{0^n 1^{n+3} \ ; \ n \geq 0\}$.

   Pozor, budeme potřebovat dostatek stavů. Ověřte, zda automat náhodou nepřijímá i ta slova, která nepatří do jazyka L.
5. Sestrojte zásobníkový automat rozpoznávající jazyk

   $L = \{a^n b^n c a^k b^k \ ; \ n \geq 0, \ k \geq 1\}$.

5.3 Nedeterminismus

Jaký je rozdíl mezi deterministickým a nedeterministickým zásobníkovým automatem? Deterministický v každém roku reaguje jednoznačně. Narozdíl od konečného automatu se to trochu hůře pozná v krocích, ve kterých automat nečte ze vstupu.

Podívejme se na automaty, které jsme zkonstruovali v příkladech v této kapitole.

Příklad 5.4

Automat v příkladu 5.1 je deterministický, protože v každém kroku dokážeme reagovat zcela jednoznačně. Jedná se o deterministický zásobníkový automat. Podobně i v následujícím příkladě.

Pokud dokážeme pro jazyk zkonstruovat deterministický automat, víme, že ten jazyk je deterministický. Proto jazyk
$L = \left\{ wcw^R \ ; \ w \in \{a,b\}^* \right\}$
je deterministický bezkontextový jazyk.

Příklad 5.5

Zásobníkový automat z příkladu 5.3 naproti tomu deterministický není. Proč?

Podívejme se na tyto dva řádky předpisu δ-funkce:
$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$
$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$
V obou případech reagujeme ve stavu q_0 a ze zásobníku jsme vytáhli symbol Z_0. V prvním případě sice čteme ze vstupu symbol a a v druhém případě vstup nečteme („nevšímáme" si ho), ale musíme si uvědomit, že v žádném kroku vlastně *nejsme nuceni číst vstup*.

Takže například v konfiguraci (q_0, abb, Z_0) jsou ve skutečnosti dvě možnosti jak reagovat:
$(q_0, abb, Z_0) \vdash (q_0, bb, aZ_0)$
$(q_0, abb, Z_0) \vdash (q_0, abb, \varepsilon)$
Druhá možnost sice nevede ke koncové konfiguraci (vlastně ani následující krok již není možné provést), ale to u nedeterministického automatu nehraje roli, je důležité, že daný krok lze provést. Z toho vyplývá, že tento automat je nedeterministický.

Kdyby bylo možné pro daný jazyk sestrojit alespoň jeden deterministický automat, byl by tento jazyk deterministický, ale to nelze (proč?). Proto můžeme říct, že jazyk
$L = \{a^n b^{2n} \ ; \ n \geq 0\}$
je nedeterministický bezkontextový jazyk.

Úkoly

1. Jazyk $L = \left\{ ww^R \ ; \ w \in \{a,b\}^* \right\}$ není deterministický. Sestrojte zásobníkový automat, který ho rozpoznává, a zdůvodněte, proč není deterministický.

   Nápověda: v příkladu 5.1 je automat pro podobný jazyk, jen chybí „hraniční" symbol c. Do stavu q_1 tedy musíme přejít někde na hranici, kterou ale nedokážeme detekovat jednoznačně. Rozhodně to bude v místě, kde na vstupu čteme tentýž symbol jako je ten, který jsme vyndali ze zásobníku, v těchto případech se tedy musí automat ve stavu q_0 chovat nedeterministicky.
2. Ukažte, že jazyk $L = \{a^n b^{2n} \ ; \ n \geq 1\}$ je deterministický bezkontextový jazyk.

   Nápověda: sestrojte zásobníkový automat a všimněte si rozdílu oproti automatu z příkladu 5.3.

5.4 Zásobníkový automat podle bezkontextové gramatiky

Mezi zásobníkovými automaty a bezkontextovými gramatikami existuje podobný vztah jako mezi konečnými automaty a regulárními gramatikami. Existuje algoritmus, jak podle bezkontextové

gramatiky sestrojit ekvivalentní zásobníkový automat (ten si za chvíli ukážeme) a naopak jak podle zásobníkového automatu sestrojit bezkontextovou gramatiku (ten je složitější, necháme si ho na další semestr).

Postup bude založen na úplně jiném principu než u regulárních jazyků. Cokoliv se bude dít, to se bude dít na zásobníku. Budeme potřebovat jen jediný stav (resp. stav pro nás nebude představovat žádnou relevantní informaci, ten jeden máme jen proto, že nějaký stav být musí). Podle zadané gramatiky sestrojíme zásobníkový automat končící prázdným zásobníkem.

Postupujeme takto:

- jediný stav q, je zároveň počáteční,
- abecedu automatu vezmeme z množiny terminálních symbolů,
- zásobníkovou abecedu utvoříme z množin terminálních a neterminálních symbolů (vše se děje na zásobníku, tj. musíme sem zařadit všechny „stavební kameny“),
- δ-funkci sestrojíme především podle pravidel, jak uvidíme o něco dále,
- jako symbol konce zásobníku použijeme startovací symbol gramatiky,
- množina koncových stavů bude prázdná (končíme prázdným zásobníkem).

Zbývá určit δ-funkci. Její předpis se bude skládat ze dvou částí:

1. první část sestrojíme podle pravidel gramatiky, použijeme tehdy, když je na vrcholu zásobníku neterminál:

   $A \to \alpha \qquad \Longrightarrow \qquad \delta(q, \varepsilon, A) \ni (q, \alpha)$

   znamená:

   pokud je na vrcholu zásobníku neterminál A, nebudeme si všímat vstupu, vyjmeme A ze zásobníku a místo něj tam dáme α (pravou stranu pravidla)

2. druhá část se použije tehdy, když bude na vrcholu zásobníku terminální symbol:

   $\delta(q, a, a) = (q, \varepsilon)$

   znamená:

   pokud je na vrcholu zásobníku terminál a a na vstupu bude tentýž terminál, vyjmeme a ze zásobníku a nic tam už dávat nebudeme.

Příklad 5.6

Podle zadané gramatiky sestrojíme zásobníkový automat:
$G = (\{S, A, B\},\ \{a, b, c\},\ P,\ S)$
$S \to abSba \mid A$
$A \to cAc \mid aB$
$B \to aB \mid \varepsilon$

Použijeme jediný stav q, zásobníková abeceda bude obsahovat všechny terminály a neterminály. Výsledný automat bude následující:
$\mathcal{A} = (\{q\},\ \{a, b, c\},\ \{S, A, B, a, b, c\},\ \delta,\ q,\ S,\ \emptyset)$

První část – podle pravidel:
$\delta(q, \varepsilon, S) = \{(q, abSba),\ (q, A)\}$
$\delta(q, \varepsilon, A) = \{(q, cAc),\ (q, aB)\}$
$\delta(q, \varepsilon, B) = \{(q, aB),\ (q, \varepsilon)\}$

Druhá část – podle terminálů:
$\delta(q, a, a) = \{(q, \varepsilon)\}$
$\delta(q, b, b) = \{(q, \varepsilon)\}$
$\delta(q, c, c) = \{(q, \varepsilon)\}$

V gramatice odvodíme slovo a totéž slovo rozpoznáme ve vytvořeném automatu:
$S \Rightarrow abSba \Rightarrow abAba \Rightarrow abaBba \Rightarrow abaaBba \Rightarrow abaaba$

Ekvivalentní zpracování (téhož) slova v automatu:
$(q, abaaba, S) \vdash (q, abaaba, abSba) \vdash (q, baaba, bSba) \vdash (q, aaba, Sba) \vdash (q, aaba, Aba) \vdash$
$\vdash (q, aaba, aBba) \vdash (q, aba, Bba) \vdash (q, aba, aBba) \vdash (q, ba, Bba) \vdash (q, ba, ba) \vdash (q, a, a) \vdash (q, \varepsilon, \varepsilon)$

Co se vlastně děje v takto zkonstruovaném automatu? Všimněte si obsahu zásobníku, srovnejte ho s odvozením slova v gramatice. Když si odmyslíme manipulaci s terminály v zásobníku (zleva je „umazáváme“), je jasné, že vlastně provádíme simulaci. V zásobníku simulujeme odvození slova, a pokud se podaří v simulaci dojít v zásobníku ke slovu, které jsme měli na vstupu (přesněji celé ho v zásobníku po odvození „zlikvidovat“), můžeme tvrdit, že slovo ze vstupu lze vygenerovat gramatikou, podle které byl automat sestrojen.

Úkoly

1. Podle odvození (derivace) na konci příkladu 5.6 sestrojte derivační strom. V tomto derivačním stromě sledujte, co se děje v souvislosti s rozpoznáváním slova v ekvivalentním automatu. Dá se říct, že automat prochází určitým způsobem tento derivační strom?
2. Sestrojte zásobníkové automaty ekvivalentní s těmito gramatikami:

$G_1 = (\{S\},\ \{a, b\},\ P,\ S)$
$S \to aSa \mid bSb \mid c$

$G_2 = (\{S, A, B\},\ \{a, b\},\ P,\ S)$
$S \to bAB \mid aBA \mid \varepsilon$
$A \to abAab \mid \varepsilon$
$B \to baBba \mid \varepsilon$
