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

Lucie Ciencialová

Teoretické základy informatiky

pro Informační studia a knihovnictví

Ústav informatiky
Filozoficko-přírodovědecká fakulta
Slezská univerzita v Opavě

Opava, 2014

Teoretické základy informatiky
pro Informační studia a knihovnictví

RNDr. Lucie Ciencialová Ph.D.

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

Obálka: © Lucie Ciencialová, 2014

Tisk: Z+M Partner, spol. s. r. o., Ostrava

Vydáno v Opavě v prosinci 2014

ISBN: 978 – 80 –7510 – 130 – 3

Tato inovace předmětu Teoretické základy informatiky je spolufinancována Evropským sociálním fondem a Státním rozpočtem ČR, projekt číslo CZ.1.07/2.2.00/28.0014, „Interdisciplinární vzdělávání v ICT s jazykovou kompetencí“. Skripta slouží primárně jako vzdělávací materiál pro cílovou skupinu projektu.

Obsah

Úvod

Mezi klasické části teoretické informatiky patří teorie jazyků a automatů. Tato skripta jsou zamýšlena jako studijní opora pro předmět *Teoretické základy informatiky* studijního oboru *Informační a knihovnická studia*. Skripta jsou samozřejmě použitelná i pro další předměty, jejichž náplní je teorie jazyků a automatů. Daný text je vhodný jak pro prezenční typ studia, tak i pro typ kombinovaný. Cílem je, aby studenti pochopili důležité pojmy z oblasti teorie jazyků a automatů.

Text je členěn logicky do jednotlivých kapitol, ve kterých jsou zřetelně vyznačeny definice, věty, algoritmy a v každé kapitole je nemalé množství řešených příkladů. V závěru jednotlivých kapitol jsou uvedeny neřešené příklady vztahující se k dané kapitole.

V každém vědním oboru máme k dispozici určité nástroje, pomocí kterých zkoumáme objekty našeho zájmu, snažíme se popisovat jejich chování, vlastnosti, modelujeme je. Pokud danými objekty budou počítače (jejich fungování, omezení), dále překladače vyšších programovacích jazyků je jednou z disciplín tzv. Teorie formálních jazyků a automatů.

Podívejme se trochu do historie. Základy této teorie položil v roce 1956 americký matematik Noam Chomsky, který v souvislosti se studiem přirozených jazyků vytvořil matematický model gramatiky jazyka. Původní představa formalizovat popis přirozeného jazyka tak, aby mohl probíhat automatický překlad z jednoho přirozeného jazyka do jiného, nebo aby přirozený jazyk byl prostředek komunikace člověka a počítače, se ukázala jako velmi složitá. Přesto byl tento pokus velmi důležitý, protože vytvoření gramatiky reprezentující formální jazyk (například některý programovací jazyk) se ukázalo jako proveditelné a užitečné.

Jak jsme již uvedli, teorie formálních jazyků našla uplatnění v programovacích jazycích. Podívejme se znovu trochu do historie. Chomsky definoval gramatiky formálního jazyka a provedl klasifikaci formálních jazyků. Backus a Naure použili základní objekty gramatiky pro definici syntaxe programovacího jazyka ALGOL 60. Další vývoj vedl k aplikaci teorie jazyků do oblasti překladačů programovacích jazyků. Musíme si uvědomit tu velikou změnu při konstrukci překladačů umožňující do značné míry automatizovat

náročnou programátorskou práci spojenou s implementací programovacích jazyků.

V úvodu této kapitoly jsme napsali, že teorie jazyků a automatů spadá do oblasti teoretické informatiky. Ale vymezení pojmu teoretická informatika není jednoduché. Jako hlavní části teoretické informatiky z jistého úhlu pohledu můžeme chápat právě teorii jazyků a automatů, teorii vyčíslitelnosti a složitosti, ale také část logiky. V tomto skriptu se dočteme především o dvou matematických formalismech, o gramatikách a automatech, které budou představovat abstraktní modely. Zatímco gramatika umožňuje popsat strukturu vět formálního jazyka, automat dovede tento jazyk rozpoznávat.

Poznatky z teorie formálních jazyků nemají jen význam v oblastech informatiky, informačních technologií. Díky této teorii máme algoritmy, jež jsou základem pro konstrukci reálných programů či zařízení, která zpracovávají informace ve tvaru vět formálního jazyka. Díky poznatkům jsme schopni stanovit možnosti a omezení algoritmických postupů řešení problémů a také nám slouží k odhalování problémů, které jsou algoritmicky nerozhodnutelné.

Nesmíme zapomenout také uvést význam pro kybernetiku a umělou inteligenci, a bez dané teorie jazyků by stěží existovaly tak výkonné a efektivní produkty informatiky jako jsou vývojové nástroje, databázové dotazovací jazyky, značkovací jazyky XML, HTML a další.

Nyní už jen zbývá jediné – popřát čtenáři mnoho zaujetí a trpělivosti při čtení následujících kapitol a zkoumání jednotlivých vlastností jazyků a automatů a jejich vzájemných vztahů.

Kapitola 1

Abeceda, slovo, jazyk

V této kapitole seznámíme čtenáře se základními pojmy teorie formálních jazyků. Terminologie vychází z analogie s přirozenými jazyky. Základním stavebním kamenem jsou symboly, písmena jeho abecedy. Z písmen lze tvořit slova jazyka. Některé posloupnosti písmen tvoří slovo jazyka , jiné ne. Jazyk je množina správně vytvořených slov.

1.1 Abeceda

Základním stavebním kamenem mluveného slova jsou hlásky, psaného textu pak písmena, stavebnice se skládá z jednotlivých kostiček, dům z cihel, oken, trubek atd. V teoretické informatice těmto základním prvkům říkáme **symboly**. Obvykle jsou to písmena latinky, mohou to však být různé značky, geometrické útvary … Pokud sestavíme množinu všech typů symbolů, které se v daném textu, skládance, atd. vyskytují, dostáváme **abecedu**.

Abeceda

Definice 1.1 **Def**

Abeceda je konečná množina. Její prvky nazýváme symboly, znaky nebo písmena.

Abecedou může být jakákoli konečná množina, $\{a\}$ je příklad jednoprvkové abecedy a $\varnothing$ je prázdná abeceda.

Příklad 1.1 P

Uveďte sedm příkladů různých abeced.

1. $A = \{a, b, c, d, e, f, \ldots, y, z\}$ – abeceda tvořená malými písmeny latinky, bez diakritiky;
2. B = { A, Á, E, É, I, Í, O, Ó, U, Ú, Ů, Y, Ý} – abeceda tvořená samohláskami i s diakritikou;
3. $C = \{\alpha, \beta, \ldots, \omega\}$ – abeceda tvořená malými písmeny řecké abecedy;
4. $D = \{a, b, c\}$ – tříprvková abeceda;
5. $E = \{a\}$ - jednoprvková abeceda;
6. $F = \{\blacktriangle, \blacksquare, \blacklozenge, \bigstar, \bullet\}$ – abeceda tvořená grafickými symboly;
7. $G = \{\cdot, -\}$ – abeceda pro morseovku.

1.2 Slovo

Zřetězením určitého počtu symbolů abecedy získáme **slovo**. Protože vzniklo zřetězením, nazýváme ho také někdy **řetězec**. Prázdné slovo, které neobsahuje žádné symboly, značíme ε. Slovo

Definice 1.2 Def

Slovo (řetězec) nad abecedou Σ je posloupnost symbolů $a_1 a_2 \ldots a_n$, kde $a_i \in \Sigma$, $1 \leq i \leq n$.

Slovo nad abecedou Σ je definováno takto:

(1) prázdné slovo ε je slovo nad abecedou Σ,

(2) je-li ω slovo nad Σ a $a \in \Sigma$, pak ωa je slovo nad Σ,

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

O slovech, která jsou složena ze symbolů z abecedy Σ, říkáme, že jsou vytvořena nad abecedou Σ.

Příklad 1.2 P

Vytvořte po jednom slově ze symbolů abeced uvedených v předchozím příkladě.

1. $s_A = sobota$ je slovo vytvořené ze symbolů abecedy A;
2. $s_B = E$ je slovo vytvořené ze symbolů abecedy B;
3. $s_C = \delta\omega\phi$ je slovo vytvořené ze symbolů abecedy C;
4. $s_D = aaabbbaaa$ je slovo vytvořené ze symbolů abecedy D;
5. $s_E = aaaaaaaaaaaa$ je slovo vytvořené ze symbolů abecedy E;
6. $s_F =$ ◆ ★ ◆ ★ ◆ je slovo vytvořené ze symbolů abecedy F;
7. $s_G = --\cdot$ je slovo vytvořené ze symbolů abecedy G.

Počet symbolů, jejichž zřetězením slovo vzniklo, nazýváme **délkou slova**. Prázdné slovo ε má nulovou délku. Délka slova

Definice 1.3 Def

Délka slova je nezáporné celé číslo udávající počet symbolů slova. Délku slova ω značíme symbolicky $|\omega|$. Je-li $\omega = a_1a_2\ldots a_n$, $a_i \in \Sigma$ pro $i = 1,\ldots,n$, pak $|\omega| = n$.

Příklad 1.3 P

Uveďte délky slov z předchozího příkladu.

1. $|s_A| = |sobota| = 6$;
2. $|s_B| = |E| = 1$;
3. $|s_C| = |\delta\omega\phi| = 3$;
4. $|s_D| = |aaabbbaaa| = 9$;
5. $|s_E| = |aaaaaaaaaaaa| = 12$;

6. $|s_F| = |\blacklozenge\bigstar\blacklozenge\bigstar\blacklozenge| = 5$;
7. $|s_G| = |--\cdot| = 3$.

Slovo (nebo podslovo), které sestává právě z k výskytů symbolu a, budeme symbolicky značit a^k. Např. $a^3 = aaa, b^2 = bb, a^0 = \varepsilon$.

Při práci se slovy nás ne vždy zajímá počet všech symbolů, ale potřebujeme znát počet konkrétních symbolů ve slově. Počet výskytů symbolu a ve slově ω značíme $|\omega|_a$.

Příklad 1.4 **P**

Uveďte $|\omega|_a$ pro zadané slovo ω a symbol a:

1. $\omega = xyzzyxxzx,\ a = x$

   - počet symbolů x ve slově $xyzzyxxzx$ je 4 a tedy $|xyzzyxxzx|_x = 4$;
2. $\omega = \pi\beta\delta\alpha\phi\beta\alpha\rho\beta\mu,\ a = \beta$

   - počet symbolů β ve slově $\pi\beta\delta\alpha\phi\beta\alpha\rho\beta\mu$ je 3 a tedy $|\pi\beta\delta\alpha\phi\beta\alpha\rho\beta\mu|_\beta = 3$;
3. $\omega =$ 🌨🌤🌩🌫♋🌨🌦🌫🌩, $a =$ 🌤

   - počet symbolů 🌤 ve slově 🌨🌤🌩🌫♋🌨🌦🌫🌩 je 1 a tedy $|$🌨🌤🌩🌫♋🌨🌦🌫🌩$|_{🌤} = 1$.

Značení V textu budeme používat následující značení:

- malá písmena ze začátku latinské abecedy – $a, b, c, \ldots$ – pro symboly abecedy
- velká písmena řecké abecedy – nejčastěji Σ, Γ – pro abecedy
- malá písmena řecké abecedy – nejčastěji ω, α, β – pro slova

Toto označení používáme jako obecné. Abecedou Σ nebudeme myslet jednu konkrétní abecedu, ale prostě jen „nějakou“ obecnou abecedu. Stejně tak to platí i pro symboly. Například pokud použijeme konstrukci „pro každé $a \in \Sigma$ platí …“ neznamená to, že fyzický symbol a je prvkem abecedy Σ. V této souvislosti a představuje proměnnou, za kterou můžeme dosadit jakýkoli fyzický prvek abecedy.

Množina všech slov

Výrazem Σ^* značíme **množinu všech slov** nad abecedou Σ; Σ^+ je označení pro množinu všech neprázdných slov nad abecedou Σ. Platí tedy

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\} \text{ a } \Sigma^+ = \Sigma^* - \{\varepsilon\}.$$

Dále pro prázdnou množinu platí $\varnothing^* = \{\varepsilon\}$ a $\varnothing^+ = \varnothing$.

Množina všech slov nad konečnou abecedou je sice nekonečná, ale spočetná. Slova v dané abecedě můžeme seřadit (uspořádat): nejprve podle délky a v rámci stejné délky podle abecedy, tj. podle zvoleného uspořádání na prvcích abecedy. Tak jsou slova seřazena do jedné posloupnosti, ve které je lze po řadě očíslovat přirozenými čísly, která tak představují pořadová čísla slov v posloupnosti.

1.3 Jazyk

Lingvisté definují **jazyk** jako znakový systém, pomocí kterého se popisují věci, akce, myšlenky a stavy, případně jako soubor gramaticky správných vyjádření. Druhá definice se blíží definici formálního jazyka nejvíce.

Definice 1.4

Def
Jazyk

Formální jazyk, stručně jazyk, nad abecedou Σ je množina konečných slov (tj. slov konečné délky) vytvořených ze symbolů abecedy Σ.

Jazyk na abecedou Σ je tedy libovolná podmnožina množiny Σ^*. Jazykem nad abecedou Σ, který má nejvíce slov, je právě Σ^*. Naopak jazykem, který má slov nejméně, je jazyk prázdný. Poznamenejme jen, že prázdný jazyk může být nad libovolnou abecedou. Dalším takovým příkladem jazyka je jazyk, který obsahuje jediné slovo (může jít i o prázdné slovo). Pokud bude zřejmé, nad jakou abecedou je jazyk definován, budeme jednoduše používat pojem „jazyk“.

Pro definici toho, jaká slova patří do jazyka, používáme tři způsoby zápisu. Jazyk zapisujeme:

1. Výčtem prvků – uvádíme úplný seznam všech prvků, které do jazyka patří, nebo dostatečné množství slov seřazených tak, aby bylo zřejmé, jaká slova ve výčtu chybí.
   - {leden, únor, březen, duben}
   - {dveře, okno, stůl, postel, židle}
   - $\{ABC, ACB, BAC, BCA, CAB, CBA\}$
   - $\{aa, aaaa, aaaaaa, a^8, a^{10}, \dots\}$

2. Vyjádřením závislosti – definujeme obecný tvar slova a podmínky, které určují, která slova definovaného tvaru do jazyka patří.

   $\{x \mid \text{podmínka}\}$

   Jazyk definovaný v předchozí části $\{aa, aaaa, aaaaaa, a^8, a^{10}, \dots\}$ lze zapsat pomocí závislosti následujícím způsobem:

   $\{a^i \mid i = 2k, i, k \in N\}$

   Čteme „Do jazyka patří slova tvaru a na i-tou, kde i je sudé číslo různé od nuly“. Jiná interpretace je „Do jazyka patří neprázdná slova tvořená symboly a, která mají sudou délku“.

   Vysvětlení zápisu:

   - N je množina přirozených čísel – $\{1, 2, 3, \dots\}$. V textu naleznete také značení N_0 – množina přirozených čísel s nulou.
   - a^i je tvar, který mají slova, která patří do daného jazyka. V našem případě jsou to slova, která se skládají ze symbolů a. Počet symbolů a je dán číslem i.
   - $i = 2k$ definuje právě onu závislost nebo podmínku. Počet symbolů a ve slově musí být roven dvojnásobku jiného čísla, neboli musí být sudé. Protože je definováno, že k je celé číslo větší než nula, pak nejkratším slovem jazyka je a^2.

3. pomocí tzv. regulárních výrazů – tomuto způsobu zápisu se budeme věnovat na konci této kapitoly.

V dalším textu budeme předpokládat, pokud neuvedeme jinak, že proměnné použité v definicích jsou přirozená čísla (mohou nabývat pouze hodnoty z množiny N_0). Místo $\{a^i \mid i = 2k, i, k \in N_0\}$ tedy můžeme zapsat $\{a^i \mid i = 2k, k \geq 0\}$, nebo místo $\{a^i \mid i = 2k, i, k \in N\}$ můžeme zapsat $\{a^i \mid i = 2k, k \geq 1\}$ nebo $\{a^i \mid i = 2k, k > 0\}$.

Příklad 1.5 P

Zadejte dané jazyky výčtem prvků:

1. $L_1 = \{a^i \mid 1 \leq i \leq 4\}$

   Jazyk L_1 obsahuje slova nad abecedou $\{a\}$, která mají počet symbolů od jedné do čtyř. Proto $L_1 = \{a, aa, aaa, aaaa\}$.

2. $L_2 = \{a^i \mid i > 6\}$

   Na rozdíl od jazyka L_1 je počet symbolů slov náležících do jazyka L_2 omezen pouze zdola. Tento jazyk tudíž není konečný.

   $L_2 = \{a^7, a^8, a^9, \dots\}$
3. $L_3 = \{a^i b^i \mid i \leq 2\}$

   Tento jazyk je tvořen slovy, která se skládají ze dvou částí. První část je složena ze symbolů a a druhá ze symbolů b. Protože ve výrazu $a^i b^i$ je použitý stejný index i, je počet symbolů a a počet symbolů b ve slově vždy stejný. Index i je podmínkou omezený shora. Pokud budeme výčet slov, která jazyk L_3 obsahuje, uvádět opět od nejkratšího slova, nesmíme zapomenout na prázdné slovo. $L_3 = \{a^0 b^0 = \varepsilon, ab, aabb\}$.
4. $L_4 = \{a^i b^j \mid i \leq 2; j \in \{3, 4\}$

   Do jazyka L_4 stejně jako do jazyka L_3 patří slova složená z části obsahující pouze symboly a a z na ni navazující části obsahující pouze symboly b. Každá z částí má v definici ($a^i b^j$) použitý jiný index (i, j). Zatímco počet symbolů a je menší nebo roven dvěma, počet symbolů b je roven třem nebo čtyřem. Proto tedy

   $L_4 = \{bbb, bbbb, abbb, abbbb, aabbb, aabbbb\}$.
5. $L_5 = \{a^i b^j c^k \mid i = j; i, j, k > 0\}$

   Jedná se o jazyk nekonečný, slova jsou složena ze tří částí, jako první je část složená ze symbolů a, druhá část je složená ze samých symbolů b a slovo končí symboly c. Protože všechny indexy i, j i k jsou nenulové, ani jedna z částí nemůže chybět. Navíc podmínka $i = j$ určuje, že počet symbolů a a b je ve slově stejný.

   $L_5 = \{abc, abcc, abccc, aabbc, abcccc, aabbcc, abccccc, aabbccc, \dots\}$
6. $L_6 = \{a^i b^j \mid i \neq j; i, j > 0\}$

   Slova jazyka L_6 se opět skládají ze dvou částí – „nejdříve samá a-čka a potom samá b-čka“. Protože jsou indexy nenulové, ani jedna z částí nemůže chybět. Zbylá část podmínky určuje, že indexy musí být různé, tzn. počet symbolů a je různý od počtu symbolů b.

   $L_6 = \{aab, abb, aaab, abbb, aaaab, aaabb, aabbb, abbbb, aaaaab, aaaabb, aabbbb, abbbbb, \dots\}$

I jazyk, kterým hovoříme, čeština, má s formálními jazyky mnoho společného. Uměli byste určit nad jakou abecedou? Jak vypadají slova tohoto jazyka? Na první otázku nejspíš nebude problém odpovědět. Jako abecedu používáme latinku, středoevropský skript. Učíme se ji už v první třídě základní školy. Ale jak vypadají slova jazyka? Stačí opravdu jen naše slova? Čím je tvořen jazyk? Není to jen slovní zásoba, ale také pravidla, která určují, jak skládat slova do vět, aby věty dávaly smysl. Nejblíže slovům formálního jazyka mají opravdu věty jazyka přirozeného.

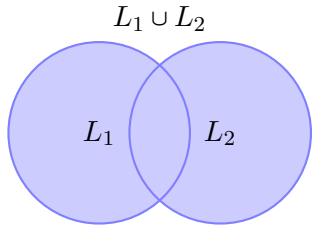
1.4 Operace se slovy a jazyky

Protože jazyky jsou množiny, provádíme s nimi stejné operace jako s ostatními množinami (například číselnými), jako jsou sjednocení, průnik, doplněk a množinový rozdíl. Pokud jsou abecedy jazyků různé, abeceda výsledného jazyka je sjednocením abeced jazyků původních.

Definice 1.5 **Def**

Sjednocení jazyků L_1, L_2 značíme $L_1 \cup L_2$. Výsledkem operace sjednocení je jazyk, který obsahuje všechna slova jazyků L_1 i L_2.
Formálně: $L_1 \cup L_2 = \{\omega \mid \omega \in L_1 \vee \omega \in L_2\}$
Čteme: Do sjednocení jazyků L_1 a L_2 patří slova ω taková, že ω náleží do jazyka L_1 nebo ω náleží do jazyka L_2.

Sjednocení



Obrázek 1.1: Znázornění sjednocení jazyků

Příklad 1.6 **P**

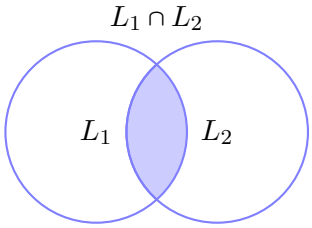
Určete $L_1 \cup L_2$:

- $L_1 = \{a, aa, aaa\}, L_2 = \{b, bb, bbb\} \Rightarrow L_1 \cup L_2 = \{a, b, aa, bb, aaa, bbb\}$;

- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{a, aaa, aaaaa\} \Rightarrow L_1 \cup L_2 = \{\varepsilon, a, aa, aaa, aaaaa\}$;
- $L_1 = \{a^i \mid i = 2k;\ k \geq 0\}, L_2 = \{a^j \mid j = 2k+1;\ k \geq 0\}$
  $\Rightarrow L_1 \cup L_2 = \{a^i \mid i \geq 0\}$.

Definice 1.6 **Def**

Průnik jazyků L_1, L_2 značíme $L_1 \cap L_2$. Výsledkem operace průniku je jazyk, který obsahuje všechna slova, která patří současně do obou jazyků L_1 i L_2. Průnik
Formálně: $L_1 \cap L_2 = \{\omega \mid \omega \in L_1 \wedge \omega \in L_2\}$
Čteme: Do průniku jazyků L_1 a L_2 patří slova ω taková, že ω náleží do jazyka L_1 a současně ω náleží do jazyka L_2.



Obrázek 1.2: Znázornění průniku jazyků

Příklad 1.7 **P**

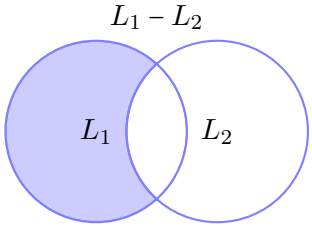
Určete $L_1 \cap L_2$:

- $L_1 = \{a, aa, aaa\}, L_2 = \{a^2, a^4, a^6\} \Rightarrow L_1 \cap L_2 = \{aa\}$;
- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{aaa, aaaaa\} \Rightarrow L_1 \cap L_2 = \varnothing$;
- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{\varepsilon, b, bbb\} \Rightarrow L_1 \cap L_2 = \{\varepsilon\}$;
- $L_1 = \{a^i \mid i = 2k;\ k \geq 0\}, L_2 = \{a^j \mid j = 3k;\ k \geq 0\}$
  $\Rightarrow L_1 \cap L_2 = \{a^i \mid i = 6k;\ k \geq 0\}$;
- $L_1 = \{a^i b^j c^k \mid i = j;\ i, j, k \geq 0\}, L_2 = \{a^i b^j c^k \mid j = k;\ i, j, k \geq 0\}$
  $\Rightarrow L_1 \cap L_2 = \{a^i b^j c^k \mid i = j = k;\ i, j, k \geq 0\}$.

Definice 1.7 Def

Rozdíl jazyků L_1, L_2 značíme $L_1 - L_2$. Výsledkem operace rozdílu je jazyk, který obsahuje všechna slova patřící do jazyka L_1 a nepatřící do jazyka L_2.
Formálně: $L_1 - L_2 = \{\omega \mid \omega \in L_1 \wedge \omega \notin L_2\}$
Čteme: Do rozdílu jazyků L_1 a L_2 patří slova ω taková, že ω náleží do jazyka L_1 a současně ω nenáleží do jazyka L_2.

Rozdíl



Obrázek 1.3: Znázornění rozdílu jazyků

Příklad 1.8 P

Určete $L_1 - L_2$:

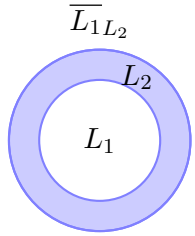
- $L_1 = \{a, aa, aaa\}, L_2 = \{a^2, a^4, a^6\} \Rightarrow L_1 - L_2 = \{a, aaa\}$;
- $L_1 = \{a^2, a^4, a^6\}, L_2 = \{a, aa, aaa\} \Rightarrow L_1 - L_2 = \{a^4, a^6\}$;
- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{aaa, aaaaa\} \Rightarrow L_1 - L_2 = L_1$;
- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{\varepsilon, b, bbb\} \Rightarrow L_1 - L_2 = \{a, aa\}$;
- $L_1 = \{a^i \mid i = 2k;\ k \geq 0\}, L_2 = \{a^j \mid j = 3k;\ k \geq 0\}$
  $\Rightarrow L_1 - L_2 = \{a^i \mid i = 2k \wedge i \neq 6l;\ k, l \geq 0\}$.

Definice 1.8 Def

Doplněk jazyka L_1 do jazyka L_2 značíme $\overline{L_1}_{L_2}$. Výsledkem operace doplňku je jazyk, který obsahuje všechna slova, která nepatří do jazyka L_1 a patří do jazyka L_2. Doplněk je definován pouze pro takové jazyky, pro které platí $L_1 \subseteq L_2$.
Formálně: $\overline{L_1}_{L_2} = \{\omega \mid \omega \notin L_1 \wedge \omega \in L_2\}$

Doplněk

Čteme: Do doplňku jazyka L_1 do jazyka L_2 patří slova ω taková, že ω nenáleží do jazyka L_1 a současně ω náleží do jazyka L_2.



Obrázek 1.4: Znázornění doplňku jazyka

Nechť L je jazyk na abecedou Σ. Místo $\overline{L}_{\Sigma^*}$ budeme psát pouze $\overline{L}$.

Příklad 1.9 **P**

Určete $\overline{L_1}_{L_2}$:

- $L_1 = \{a, aa, aaa\}, L_2 = \{\varepsilon, a, a^2, a^3, a^4, a^5, a^6\} \Rightarrow \overline{L_1}_{L_2} = \{\varepsilon, a^4, a^5, a^6\}$;
- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{a\}^* \Rightarrow \overline{L_1}_{L_2} = \{a^3, a^4, a^5, \dots\}$;
- $L_1 = \{\varepsilon, a, aa\}, L_2 = \{\varepsilon, a, bb, aaa, bbbb, aaaaa\} \Rightarrow$ nelze, protože $L_1 \nsubseteq L_2$.

Dále uvedeme základní operaci se slovy, a to operaci zřetězení. Zřetězení

Definice 1.9 **Def**

Zřetězení slov $u = a_1a_2 \dots a_n, v = b_1b_2 \dots b_m$ označujeme $u \cdot v$, stručněji uv, a definujeme $uv = a_1a_2 \dots a_nb_1b_2 \dots b_m$.

Platí, že $\omega \cdot \varepsilon = \varepsilon \cdot \omega = \omega$, kde $\omega \in \Sigma^*$

Příklad 1.10 **P**

Nechť $u = abba$ a $v = babb$. Uveďte $u \cdot v$ a $v \cdot u$.

- $u \cdot v = abba \cdot babb = abbababb$;
- $v \cdot u = babb \cdot abba = babbabba$.

Operace zřetězení je asociativní, to znamená, že platí $(u \cdot v) \cdot w = u \cdot (v \cdot w)$.

Příklad 1.11 P

Ověřte asociativitu operace zřetězení pro $u = abb$, $v = aaa$, $w = bbb$.

- $(u \cdot v) \cdot w = (abb \cdot aaa) \cdot bbb = abbaaa \cdot bbb = abbaaabbb$;
- $u \cdot (v \cdot w) = abb \cdot (aaa \cdot bbb) = abb \cdot aaabbb = abbaaabbb$.

Již dříve jsme se zmínili o tom, že slova vznikla zřetězením. Teď, když jsme definovali operaci zřetězení, můžeme přejít k dalším pojmům, a to jsou podslovo, prefix (předpona) a sufix (přípona).

Podslovo, prefix a sufix

Definice 1.10 Def

Slovo v je podslovo slova ω, jestliže existují slova u a w taková, že platí:

$$\omega = u \cdot v \cdot w$$

Pokud $u = \varepsilon$, pak v je prefix (předpona).
Pokud $w = \varepsilon$, pak v je sufix (přípona).

Speciálními podslovy (resp. prefixy, sufixy) jsou: celé původní slovo a prázdné slovo. Pokud se podslova (resp. prefixy, sufixy) nerovnají původnímu slovu, říká se jim vlastní. Každé podslovo se dá vyjádřit jako prefix sufixu nebo sufix prefixu.

Příklad 1.12 P

Určete nějaké podslovo, prefix a sufix slova $aabab$.

- vlastní podslova jsou například $ab,\ ba,\ aba$;
- další podslova jsou ε a $aabab$;
- vlastní prefixy jsou například $a,\ aa,\ aab$;
- vlastní sufixy jsou například $b,\ ab,\ bab$.

Příklad 1.13 P

Uveďte všechna podslova slova 0011. Najděte taková slova délky 2 a 3, která nejsou podslovy slova 0011.

- Podslova: $\varepsilon, 0, 1, 00, 01, 11, 001, 011, 0011$.
- Nejsou podslovem: $10, 000, 010, 100, 101, 110, 111$.

Další operací, kterou se budeme zabývat je mocnina slova. Vzniká tehdy, když řetězíme opakovaně jedno slovo. Tuto operaci definujeme rekurzivně.

Definice 1.11 Def

n-tou mocninu slova ω pro přirozené n značíme ω^n a definujeme následovně:

1. $\omega^0 = \varepsilon$,
2. $\omega^n = \omega^{n-1} \cdot \omega$

Platí tedy:

- $\omega^0 = \varepsilon$;
- $\omega^1 = \omega$;
- $\omega^2 = \omega \cdot \omega$;
- $\omega^3 = \omega^2 \cdot \omega = \omega \cdot \omega \cdot \omega$; atd.

Mnohé operace, které jsou nadefinovány nad slovy, je možné rozšířit i na jazyky. Jako první uvedeme operaci zřetězení.

Definice 1.12 Def

Zřetězením jazyků L_1 nad abecedou Σ_1 a L_2 nad abecedou Σ_2 nazýváme jazyk

$$L_1 \cdot L_2 = \{u \cdot v \mid u \in L_1 \wedge v \in L_2\}$$

nad abecedou $\Sigma_1 \cup \Sigma_2$.

Do zřetězení jazyků patří všechna slova vzniklá zřetězením všech kombinací slov obou jazyků v pořadí stanoveném zřetězením jazyků. V praxi vypadá $L_1 \cdot L_2$ tak, že si vytvoříme na obou jazycích L_1 a L_2 uspořádání (seřadíme slova) a zřetězíme první slovo prvního jazyka postupně se všemi slovy druhého jazyka, pak vezmeme druhé slovo prvního jazyka a opět zřetězíme se všemi slovy druhého jazyka, ... Tímto způsobem získáme nejvýše $|L_1| \cdot |L_2|$ slov, kde $|L|$ je mohutnost neboli počet prvků množiny L.

Příklad 1.14 **P**

Definujte výsledek zřetězení jazyků $L_1 \cdot L_2$ a $L_2 \cdot L_1$:

1. $L_1 = \{0^n \mid n \geq 0\}$, $L_2 = \{1\} \Rightarrow L_1 \cdot L_2 = \{0^n 1 \mid n \geq 0\}$ a $L_2 \cdot L_1 = \{10^n \mid n \geq 0\}$;
2. $L_1 = \{1^n \mid n > 0\}$, $L_2 = \{0, 00\} \Rightarrow L_1 \cdot L_2 = \{1^n 0, 1^n 00 \mid n > 0\}$ a $L_2 \cdot L_1 = \{01^n, 001^n \mid n > 0\}$;
3. $L_1 = \{01, 11\}$, $L_2 = \{0, 00\} \Rightarrow L_1 \cdot L_2 = \{010, 0100, 110, 1100\}$ a $L_2 \cdot L_1 = \{001, 011, 0001, 0011\}$.

Definice 1.13 **Def**

Je-li L jazyk nad abecedou Σ, pak L^n značí n-tou mocninu jazyka L a platí: $L^0 = \{\varepsilon\}$, $L^n = L^{n-1} \cdot L$.

Příklad 1.15 **P**

Vytvořte druhou a třetí mocninu jazyka L

1. $L = \{0\}$;
   $L^2 = \{0^2\}$;
   $L^3 = \{0^3\}$;
2. $L = \{\varepsilon, a\}$;
   $L^2 = \{\varepsilon \cdot \varepsilon, \varepsilon \cdot a = a \cdot \varepsilon, a \cdot a\} = \{\varepsilon, a, a^2\}$;
   $L^3 = \{\varepsilon, a, aa, aaa\}$;

3. $L = \{a, ab\}$;
   $L^2 = \{a \cdot a, a \cdot ab, ab \cdot a, ab \cdot ab\} = \{a^2, aab, aba, abab\}$;
   $L^3 = \{aaa, aaab, aaba, abaa, abaab, aabab, ababa, ababab\}$.

Jazyk všech slov, která lze rozdělit na několik částí, z nichž každá patří do jazyka L, nazýváme iterací.

Definice 1.14 **Def**

Iterace jazyka L se značí L^* a je definována jako

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{n=0}^{\infty} L^n$$

Pozitivní iterace jazyka L se značí L^+ a je definována jako

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots = \bigcup_{n=1}^{\infty} L^n$$

Označení pro iteraci resp. pozitivní iteraci již pozorný čtenář zná. Použili jsme je pro množinu všech slov nad abecedou s resp. bez prázdného slova. Nové použití není v rozporu s již dříve uvedeným, protože i abecedu můžeme považovat za jazyk, který obsahuje slova délky jedna. Dále jsme pro množiny všech slov nad danou abecedou uvedli, že platí vztahy: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ a $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. Jak je to s platností těchto vztahů pro jazyky obecně? Odpověď vyplyne z následujícího příkladu.

Příklad 1.16 **P**

Pro jazyky $L_1 = \{0\}$ a $L_2 = \{\varepsilon, 0\}$ uveďte iteraci a pozitivní iteraci.

Protože platí $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{n=0}^{\infty} L^n$ a $L^+ = L^1 \cup L^2 \cup \cdots = \bigcup_{n=1}^{\infty} L^n$, uveďme nejdříve, jak vypadají mocniny jazyků, poté provedeme jejich sjednocení.

n	L_1^n	L_2^n
$n = 0$	$\{\varepsilon\}$	$\{\varepsilon\}$
$n = 1$	$\{0\}$	$\{\varepsilon, 0\}$
$n = 2$	$\{00\}$	$\{\varepsilon, 0, 00\}$
$n = 3$	$\{000\}$	$\{\varepsilon, 0, 00, 000\}$
$\vdots$		
$n = i$	$\{0^i\}$	$\{\varepsilon, 0, 00, \ldots, 0^i\}$

Nyní si ukážeme, jak vypadají iterace a pozitivní iterace jazyků L_1 a L_2.

$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \bigcup_{n=0}^{\infty} L^n \Rightarrow$
$L_1^* = \{\varepsilon\} \cup \{0\} \cup \{00\} \cup \{000\} \cdots = \{\varepsilon, 0, 00, 000, \ldots\}$
$L_2^* = \{\varepsilon\} \cup \{\varepsilon, 0\} \cup \{\varepsilon, 0, 00\} \cup \{\varepsilon, 0, 00, 000\} \cdots = \{\varepsilon, 0, 00, 000, \ldots\}$

$L^+ = L^1 \cup L^2 \cup \cdots = \bigcup_{n=1}^{\infty} L^n \Rightarrow$
$L_1^+ = \{0\} \cup \{00\} \cup \{000\} \cdots = \{0, 00, 000, \ldots\}$
$L_2^+ = \{\varepsilon, 0\} \cup \{\varepsilon, 0, 00\} \cup \{\varepsilon, 0, 00, 000\} \cdots = \{\varepsilon, 0, 00, 000, \ldots\}$

Zatímco iterace a pozitivní iterace jazyka L_1 se liší o prvek ε, pro jazyk L_2 se jedná o tu samou množinu. Navíc iterace obou jazyků se rovnají.

Ze srovnání výsledných iterací a pozitivních iterací jazyků vyplývá pro platnost vztahů $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ a $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ následující. První vztah je platný. Pokud k množině, která obsahuje ε, přidáme prázdné slovo, množina se nezmění a dostáváme stejný výsledek jako pro jazyk L_2 z příkladu 1.16. Co se týče druhého vztahu, ten platí pouze pro jazyky, které neobsahují prázdné slovo.

Další operací, kterou nadefinujeme, je operace zrcadlového obrazu nazývaná též reverze.

Zrcadlový obraz

Definice 1.15 **Def**

Zrcadlový obraz (reverze) slova $\omega = a_1 a_2 \ldots a_{n-1} a_n$ je slovo
$\omega^R = a_n a_{n-1} \ldots a_2 a_1$.
$\varepsilon^R = \varepsilon$
Zrcadlovým obrazem (reverzí) jazyka L se nazývá jazyk $L^R = \{\omega^R \mid \omega \in L\}$.

Uvedeme několik příkladů reverze jak slov tak jazyků.

Příklad 1.17 **P**

Vytvořte reverzi daných slov.

1. $\omega = aab \Rightarrow \omega^R = baa$;
2. $\omega = 00 \Rightarrow \omega^R = 00 = \omega$;
3. $\omega = abba \Rightarrow \omega^R = abba = \omega$.

Příklad 1.18 **P**

Vytvořte reverzi jazyka.

1. $L = \{a^i b^i c^i \mid i > 0\} \Rightarrow L = \{c^i b^i a^i \mid i > 0\}$;
2. $L = \{00, 11\} \Rightarrow L^R = \{00, 11\} = L$;
3. $L = \{abba, abab, aabb\} \Rightarrow L^R = \{abba, baba, bbaa\}$.

Nyní přistoupíme k poslední definici této kapitoly – definici substituce.

Definice 1.16 **Def**

Substituce

Substituce (nahrazení) je zobrazení $f : \Sigma \to 2^{\Delta^*}$ abecedy Σ do podmnožin množiny Δ^*. Pro každé $a \in \Sigma$ definujeme $f(a) \subseteq \Delta^*$.
Substituce se nazývá **nevypouštějící**, jestliže $\forall a \in \Sigma$ platí,že $\varepsilon \notin f(a)$.
Substituce slova vzniká rozšířením substituce $f : \Sigma \to 2^{\Delta^*}$ na $f : \Sigma^* \to 2^{\Delta^*}$ a definujeme ji takto:

1. $f(\varepsilon) = \varepsilon$;
2. $f(x \cdot a) = f(x) \cdot f(a)$.

Čili substituce prázdného slova je opět prázdné slovo a substituce řetězce odpovídá zřetězení substitucí.
Substituci jazyka definujeme tak, že pro každé $L \in \Sigma^*$ je substituce sjednocením substitucí slov jazyka, čili: $f(L) = \bigcup_{\omega \in L} f(\omega)$. Říkáme, že jazyk $f(L)$ vznikl z L substitucí f.

Jak taková substituce vypadá v praxi, si ukážeme na příkladě.

Příklad 1.19 P

Určete $f(ab)$, jestliže platí:

1. $f(a) = \{1\}, f(b) = \{2\} \Rightarrow f(ab) = f(a) \cdot f(b) = \{12\}$;
2. $f(a) = \{1,3\}, f(b) = \{2\} \Rightarrow f(ab) = f(a) \cdot f(b) = \{12,32\}$;
3. $f(a) = \{1,2\}, f(b) = \{3,4\} \Rightarrow f(ab) = f(a) \cdot f(b) = \{13,14,23,24\}$;
4. $f(a) = \{aa,bb\}, f(b) = \{ab,ba\} \Rightarrow$;
   $f(ab) = f(a) \cdot f(b) = \{aaab,aaba,bbab,bbba\}$;
5. $f(a) = \{\varepsilon,a\}, f(b) = \{aa\} \Rightarrow f(ab) = f(a) \cdot f(b) = \{aa,aaa\}$;
6. $f(a) = \{0^n \mid n \geq 1\}, f(b) = \{1,11\} \Rightarrow f(ab) = f(a) \cdot f(b) = \{0^n 1^m \mid n \geq 1, m \in \{1,2\}\}$;

Jestliže pro každý symbol abecedy je jeho substituce jednoprvková množina, pak substituci nazýváme **homomorfismem**. Je-li h homomorfismus, pak definujeme **inverzní homomorfní obraz** slova ω jako

$$h^{-1}(\omega) = \{x \mid h(x) = \omega\}$$

a pro jazyk L definujeme inverzní homomorfismus jazyka jako

$$h^{-1}(L) = \{\omega \mid h(\omega) \in L\}.$$

1.5 Regulární výrazy

Jak jsme již uvedli na začátku části věnované jazykům, jednou z možností zápisu je použití regulárních výrazů.

Definice 1.17 Def

Regulární výraz v nad abecedou Σ je definován takto:

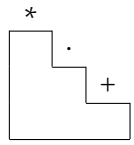
1. $\varnothing$, ε, a jsou regulární výrazy pro všechna $a \in \Sigma$;
2. Jsou-li x, y regulární výrazy, pak $(x + y)$ – sjednocení , $(x \cdot y)$ zřetězení a $(x)^*$ – iterace – jsou regulární výrazy.

Definice 1.18 Def

Hodnota $h(v)$ regulárního výrazu v nad abecedou Σ je definován takto:

1. $h(\varnothing) = \varnothing$, $h(\varepsilon) = \{\varepsilon\}$, $h(a) = \{a\}$, pro všechna $a \in \Sigma$;
2. Jsou-li x, y regulární výrazy, pak $h(x + y) = h(x) \cup h(y)$, $h(x \cdot y) = h(x) \cdot h(y)$ a $h((x)^*) = h(x)^*$.

Hodnotou regulárního výrazu je tedy množina slov – jazyk. Aby bylo možné vynechat některé závorky, byly zavedeny priority regulárních operací. Nejvyšší prioritu má operace iterace a nejnižší priorita byla přidělena operaci sjednocení. Je zvykem znaménko $\cdot$ v jasných případech vypustit.



Obrázek 1.5: Priorita regulárních operací

Nyní uvedeme příklady zápisu jazyků pomocí regulárních výrazů.

Příklad 1.20 P

Určete jazyk, který reprezentuje daný regulární výraz:

- $u = (a+b)^*$ – jedná se o iteraci sjednocení množin $\{a\}$ a $\{b\}$, tedy o jazyk $h(u) = \{a, b\}^*$ – jazyk všech slov vytvořených nad abecedou tvořenou symboly a, b;
- $u = 00(0 + 1)^*111$ – u reprezentuje jazyk tvořený slovy nad abecedou $\{0, 1\}$, která začínají dvěma nulami a končí třemi jedničkami;
- $u = a(aa)^*$ – jazyk reprezentovaný regulárním výrazem u je $h(u) = \{a^i \mid i = 2k + 1; k \geq 0\}$.

Úkoly

1. Vypište všechna slova v abecedě $\{a, b\}$, která mají délku 3.
2. Napište explicitně slovo u (posloupnost písmen), které je určeno výrazem $v^3 \cdot ba \cdot (bba)^2$, kde $v = ab$.
3. Vypište všechna slova jazyků délky menší než 5:
   - $L_1 = \{ab^*\}$,
   - $L_2 = \{a, b\}^* \cdot c$,
   - $L_3 = \{a^i c b^j \mid i, j \geq 1\}$,
   - $L_4 = (ab)^*$,
   - $L_5 = (a^*b)^*$,
   - $L_6 = (ab)^* c$,
   - $L_7 = (a^+b)^* c$,
   - $L_8 = \{(ab)^i b^j \mid i \geq 0, j \geq 1\}$,
   - $L_9 = a^*(ba)^* b$.
4. Vypište všechna slova délky 2, které jsou podslovy slova 00010 (v abecedě $\{0, 1\}$).
5. Vypište všech pět prefixů slova 0100.
6. Vypište všech pět sufixů slova 0100.
7. Vypište prvních deset slov z jazyka $L = \{\omega \in \{a, b\} \mid$ každý výskyt podslova aa je ve ω ihned následován znakem $b\}$.
8. Která slova jsou zároveň prefixem i sufixem slova 101110110? (Najdete všechna tři taková?)
9. Vypište slova ve zřetězení jazyků $\{110, 0111\} \cdot \{01, 000\}$.
10. Uvažujme jazyky $L_1 = \{\omega \in \{0, 1\} \mid \omega$ obsahuje sudý počet výskytů symbolu $0\}$ a $L_2 = \{\omega \in \{0, 1\} \mid \omega$ začíná a končí stejným symbolem $\}$. Vypište prvních sedm slov (uspořádaných podle délky a abecedně) postupně pro jazyky $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, $L_2 - L_1$ a $\overline{L_1}$.
11. Nalezněte dva různé jazyky pro které platí $L_1 \cdot L_2 = L_2 \cdot L_1$.
12. Uvažujme jazyky nad abecedou $\{a, b\}$. Vypište všechna slova ve zřetězení jazyků $L_1 = \{\varepsilon, abb, bba\}$ a $L_2 = \{a, b, abba\}$.

Kapitola 2

Konečná reprezentace jazyků

V dnešní době počítacích strojů vzniká potřeba zpracovávat jazyky mechanicky. Například kontrolovat správnost jejich syntaxe, provádět příkazy, které jsou zakódované v jednotlivých slovech, atd. Vzniká otázka, jak takový jazyk, který může být nekonečný (ale spočetný), můžeme reprezentovat právě pro potřeby počítačů.

Uvedeme si jednoduchý příklad: Mějme jazyk, jehož slova se skládají pouze z jedniček a nul. Podmínky pro to, aby slovo patřilo do jazyka, jsou:

1. stejný počet jedniček a nul,
2. slovo začíná posloupností 001 a
3. délka slova je sudá.

Asi nikomu z nás nedá velkou práci představit si, jak vypadají slova tohoto jazyka a pro slova rozumné délky i ověřit, jestli náleží nebo nenáleží do jazyka. Jak si s takovým úkolem poradí počítač? Obecně je nemožné udělat pro něj slovník slov, která patří do jazyka, a nechat ho jen porovnávat, jestli takové slovo zná. Manuální tvorba takového slovníku ovšem může být nikdy nekončící proces (stejně jako u slovníků používaných pro kontrolu pravopisu v textovém editoru nebo v T9 při psaní SMS v telefonech). Vždy je tu možnost, že nalezneme slovo jazyka, které ve slovníku není. Pro nekonečné jazyky pak dostáváme nekonečné slovníky. Jak uložit takový nekonečný slovník?

Jako kratší se jeví způsob zápisu pomocí vlastností, které slova musí splňovat. Pokud najdeme způsob, jak pravidla zapsat, počítač již může ověřit, jestli slovo pravidla splňuje. Existují dva přístupy k této kontrole:

1. Najdeme takový postup (proceduru), který je schopný vygenerovat všechna slova (následně může zkontrolovat, jestli původní slovo vygeneroval či nikoliv).
2. Najdeme takový algoritmus, který nám dá odpověď, jestli dané slovo patří do jazyka, či nikoliv (rozpozná, jestli dané slovo náleží do jazyka).

Algoritmus je výpočetní proces, který se zastaví pro každý vstup. Procedura se pro některé vstupy zastavit nemusí. Z toho vyplývá, že každý algoritmus je i procedura, ne každá procedura je však algoritmus.

Do první skupiny patří systémy, které pomocí přepisovacích pravidel umí vygenerovat z jednoho speciálního symbolu všechna slova jazyka. Říká se jim generativní gramatiky, zkráceně gramatiky. Speciální symbol použitý pro start generování se nazývá počáteční nebo také startovací. Pomocných symbolů může gramatika používat více (jen konečně mnoho), ale jen jeden z nich může být počáteční.

Generativní gramatiky

Přepisovací pravidla se skládají ze dvou částí – levé a pravé strany. Na obou stranách mohou být řetězce „smíšené“ ze symbolů abecedy i pomocných symbolů. Generování slov jazyka se provádí tak, že na počáteční symbol aplikujeme kterékoliv z existujících pravidel tak, že nahradíme vždy nějakou část řetězce, která odpovídá levé straně přepisovacího pravidla jeho pravou stranou. Tento postup opakujeme na řetězec, který jsme získali, a tak dále, až získáme řetězec tvořený pouze z abecedy jazyka, již bez pomocných symbolů.

Příklad 2.1 P

Nechť gramatika obsahuje dvě pravidla $S \to aAa$ a $A \to bSab$. Pomocné symboly jsou S, A, kde první z nich S je startovací symbol. Pak začátek odvození slova bude probíhat následujícím způsobem. Odvození začneme se symbolem S, na který aplikujeme první pravidlo $S \to aAa$.

$$S \Rightarrow aAa$$

Nově vzniklý řetězec obsahuje pomocný symbol A, který je na levé straně druhého pravidla $A \to bSab$.

$$S \Rightarrow aAa \Rightarrow abSaba$$

Automaty

Jednou z realizací algoritmu, který dá odpověď, jestli dané slovo patří do jazyka, jsou automaty – stroje, které mají vstupní pásku, z níž načítají symbol po symbolu vstupní slovo. Disponují konečnou množinou stavů, ve kterých se může automat nacházet, a pravidly, která určují, co má automat v daném stavu udělat. Pokud automat celé slovo přečte a bude se nacházet v jednom z tzv. koncových stavů, vstupní slovo přijme. Existuje mnoho různých strojů, které mají různé „schopnosti". Budeme se jimi v dalším textu zabývat.

Kapitola 3

Gramatiky

V minulé kapitole jsme čtenáře neformálně seznámili s pojmem gramatiky. Proto nyní začneme rovnou s její formální definicí.

Definice 3.1

Def
Gramatika

Generativní gramatika (zkráceně gramatika) je uspořádaná čtveřice G = (N, T, P, S), kde

- N je neprázdná konečná množina neterminálních symbolů (neterminálů);
- T je neprázdná konečná množina terminálních symbolů (terminálů), je to abeceda generovaného jazyka;

  Množiny N a T musí být disjunktní ($N \cap T = \varnothing$);
- $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ je neprázdná konečná množina přepisovacích pravidel. Každé přepisovací pravidlo přiřazuje nějakému řetězci $\alpha \in (N \cup T)^* N (N \cup T)^*$, který obsahuje alespoň jeden neterminální symbol, nějaký řetězec $\beta \in (N \cup T)^*$ terminálních a neterminálních symbolů. Pravidlo (α, β) obvykle zapisujeme ve tvaru $\alpha \to \beta$ (a čteme jako „α přepiš na β“);
- $S \in N$ je vybraný počáteční symbol, který je vždy neterminálním symbolem.

Nechť $\alpha \to \beta$ je nějaké přepisovací pravidlo gramatiky G. Pak α nazýváme levou stranou pravidla a β stranou pravou. Pravidlům typu $A \to \varepsilon$ říkáme ε-pravidla. Pokud $\beta \in T^*$, tedy pokud je pravá strana pravidla tvořena pouze terminálními symboly, nazýváme toto pravidlo ukončující.

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

Řetězce, které se skládají z terminálních i neterminálních symbolů se nazývají větné formy. Pokud nebude řečeno jinak, větné formy budeme značit malými písmeny řecké abecedy nebo malými písmeny z konce české abecedy.

Větné formy

Mechanizmus generování větných forem a slov se nazývá odvození. Z jedné větné formy můžeme přímo odvodit větnou formu druhou jedním použitím jednoho přepisovacího pravidla. Tomuto také říkáme krok odvození.

Definice 3.2

Def

Každá gramatika $G = (N, T, P, S)$ určuje binární relaci $\Rightarrow_G$ přímého odvození na množině $(N \cup T)^*$ definovanou takto:

Přímé odvození

$$u \Rightarrow_G v$$

právě když existuje pravidlo $\alpha \to \beta \in P$ a slova $\eta, \varrho \in (N \cup T)^*$ taková, že

$$\eta\alpha\varrho = u \quad \text{a} \quad \eta\beta\varrho = v.$$

Říkáme, že δ lze v gramatice G přímo odvodit z γ.

Pokud je zřejmé, které gramatiky se odvození týká, můžeme označení G vynechat a místo $\gamma \Rightarrow_G \delta$ můžeme tedy psát $\gamma \Rightarrow \delta$.

Z definice přímého odvození vyplývá, že pokud $\gamma \Rightarrow_G \delta$, pak γ obsahuje alespoň jeden neterminál. Je-li $\alpha \to \beta$ pravidlo v gramatice G, pak v této gramatice platí $\alpha \Rightarrow \beta$ položíme-li $\eta = \varrho = \varepsilon$.

Příklad 3.1 P

Nechť G je gramatika definovaná takto: $G = (N, T, P, S)$, kde $N = \{A, B, S\}$, $T = \{a, b\}$, $P = \{$

$$\begin{aligned} S &\rightarrow aABb, \\ A &\rightarrow aA, \\ B &\rightarrow Bb, \\ aABb &\rightarrow \varepsilon, \\ aA &\rightarrow b\}. \end{aligned}$$

Neterminální abecedu gramatiky G tvoří neterminály A, B a S. Poslední zmíněný - S - je startovacím (počátečním) neterminálem gramatiky. Terminální symboly jsou a, b.

Uvedeme příklad tří větných forem, které lze vytvořit nad abecedou $N \cup T$, neboli řetězců, které jsou tvořeny terminálními i neterminálními symboly. Platí, že větnými formami jsou i řetězce tvořené výhradně neterminály, nebo slova tvořená výhradně terminálními symboly.

- $\gamma_1 = aSAb$
- $\gamma_2 = AS$
- $\gamma_3 = abaaB$

Větné formy $bASBB$ a $baASBB$ jsou v relaci přímého odvození, protože gramatika obsahuje pravidlo $A \rightarrow aA$ a lze je napsat ve tvaru $bASBB = b{\cdot}A{\cdot}SBB$, $baASBB = b \cdot aA \cdot SBB$ a tedy platí:

$$bASBB \Rightarrow_G baASBB$$

Další příklady větných forem, které jsou v relaci přímého odvození:

- $\gamma = S,\ \delta = aABb$ (pravidlo $S \rightarrow aABb$)
- $\gamma = aaBa,\ \delta = aaBba$ (pravidlo $B \rightarrow Bb$)
- $\gamma = aaABb,\ \delta = a$ (pravidlo $aABb \rightarrow \varepsilon$)

Na množině $(N \cup T)^*$ definujeme také další relace odvození:

Definice 3.3 Def

Odvození

Nechť G je gramatika. Odvození v k krocích pro $k \geq 0$ značíme $\overset{k}{\Rightarrow}_G$ a definujeme je induktivně takto:

- $\overset{0}{\Rightarrow}_G$ je identická relace
- $\overset{k}{\Rightarrow}_G = \overset{k-1}{\Rightarrow}_G \cdot \Rightarrow_G$

Relaci odvození v *nejvýše* k krocích, kterou značíme $\overset{\leq k}{\Rightarrow}_G$, definujeme pro každé $k \in N_0$ předpisem

$$\overset{\leq k}{\Rightarrow}_G = \bigcup_{i=0}^{k} \overset{i}{\Rightarrow}_G$$

Relaci odvození značíme $\overset{*}{\Rightarrow}_G$ a definujeme předpisem

$$\overset{*}{\Rightarrow}_G = \bigcup_{i=0}^{\infty} \overset{i}{\Rightarrow}_G$$

Relace $\overset{*}{\Rightarrow}_G$ je reflexivní a tranzitivní uzávěr relace $\Rightarrow_G$.
Relace netriviálního odvození, kterou značíme $\overset{+}{\Rightarrow}_G$, je definována předpisem

$$\overset{+}{\Rightarrow}_G = \bigcup_{i=1}^{\infty} \overset{i}{\Rightarrow}_G$$

Relace $\overset{+}{\Rightarrow}_G$ je reflexivní uzávěr relace $\Rightarrow_G$.

Aplikace pravidel na jednotlivé větné formy probíhá postupně za sebou – sekvenčně (to znamená, že v každém kroku bude použito právě jedno pravidlo). Do relace odvození v k krocích z větné formy γ patří všechny větné formy, které lze odvodit z této větné formy postupným použitím k ne nutně různých pravidel. Do relace odvození v nejvýše k krocích z větné formy γ patří všechny větné formy, které lze odvodit z této větné formy postupným použitím nejvýše k ne nutně různých pravidel. Poslední tvrzení platí i pro použití pouze jednoho pravidla nebo dokonce i v případě, kdy nepoužijeme pravidlo žádné.

Ve druhé části definice je zmíněn reflexivní a tranzitivní uzávěr relace. Ukážeme si nyní, co si pod pojmem reflexivní a tranzitivní uzávěr představit.

Prvním pojmem, který je potřeba „přeložit", je pojem relace, resp. *binární relace*. Jedná se vlastně o množinu dvojic prvků. Relaci r můžeme interpretovat tak, že do relace patří takové dvojice (a, b) prvků množiny A, které „mezi sebou mají vztah". Dvojice osob patří do relace *rodič-dítě* tehdy, pokud pro ně platí, že druhý z dvojice je potomkem prvního. Dvojice celých

Binární relace

čísel patří do relace *dělitel* právě tehdy, když první číslo je dělitelem druhého. Jiným příkladem je relace „<“, do které patří takové dvojice čísel, kde první je menší než druhé. Neustále zde zmiňujeme dvojice prvků, kde je ale spojení s krokem odvození? Relace lze zapisovat také takovým způsobem, že místo $(a, b) \in r$ napíšeme $a\, r\, b$. Proto $(a, b) \in \Rightarrow_G$ a $a \Rightarrow_G b$ vyjadřují tu samou skutečnost.

Nyní se budeme věnovat vlastnostem binárních relací – reflexivitě a tranzitivitě.

Reflexivní je taková relace, ve které je každý prvek v relaci sám se sebou. Příkladem může být relace rovnost, protože platí, že každý prvek je roven sám sobě, zapíšeme $a = a$ nebo $(a, a) \in =$.

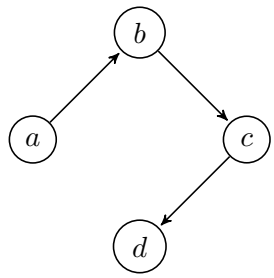
Tranzitivní relace je taková relace pro kterou platí, že pokud $(a, b) \in r$ a současně $(b, c) \in r$, pak musí i $(a, c) \in r$. Příkladem tranzitivní relace je relace „menší rovno“ $\leq$. Pro tři čísla $2, 5, 8$ platí: $2 \leq 5$ a současně $5 \leq 8$ a pak tedy i $2 \leq 8$.

Uzávěrem relace je také relace, která je uzavřená vzhledem ke stanovené vlastnosti. To znamená, že pokud Má být uzávěr relace reflexivní, musí pro každý prvek a množiny, na které je relace definovaná, platit, že (a, a) patří do uzávěru. Pokud bychom hledali tranzitivní uzávěr, pak pro každé tři prvky, pro které platí, že (a, b) a (b, c) patří do relace, pak musí do relace patřit i (a, c). Jak vypadá reflexivní a tranzitivní uzávěr relace ukážeme na příkladě. Nejdříve potřebujeme množinu prvků, např. $\{a, b, c, d\}$, a relaci definovanou nad touto množinou, např. $r = \{(a, b); (b, c); (c, d)\}$. Nyní budeme do relace přidávat takové dvojice, aby výsledná relace byla uzávěrem. Reflexivní uzávěr relace r získáme tak, že přidáme do relace takové prvky, které vyjadřují vztah jednotlivých prvků množiny sebou samými. Znamená to, že přidáme do relace prvky (a, a), (b, b), (c, c) a (d, d). Tranzitivní uzávěr vyjadřuje uzavřenost relace rozšířené vzhledem k vlastnosti tranzitivity.

Řešení našeho příkladu si ukážeme graficky. To, že (a, b) náleží do relace r znázorníme šipkou vedoucí z bodu a do bodu b. Pokud je relace tranzitivní, pak ke každým dvěma šipkám, které tvoří cestu z bodu a do bodu b a odtud do c, přidáme také zkratku – šipku z bodu a rovnou do bodu c.

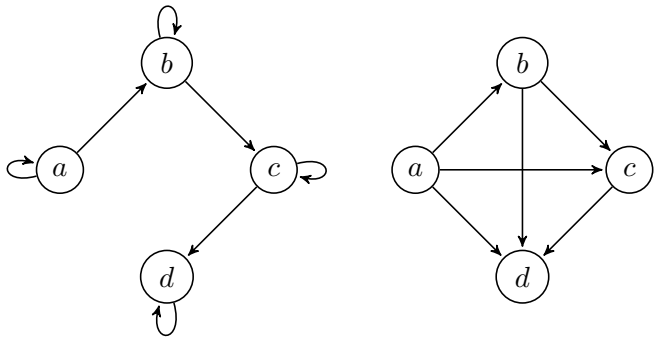
Reflexivní uzávěr relace r získáme tak, že přidáme „šipky“ vedoucí z prvku do sebe samého. Do obrázku tedy přidáme smyčky ke všem prvkům.

Vytvoření tranzitivního uzávěru je složitější. Jde o to zkontrolovat, jestli platí, že pokud vede šipka z bodu a do bodu b a z bodu b do bodu c, tak vede i z bodu a rovnou do bodu c. Tuto kontrolu provedeme pro všechny



Obrázek 3.1: Grafické znázornění relace r.

„cesty“ v grafickém znázornění relace a případně přidáme „zkratky“ mezi jednotlivé body.



(a) Reflexivní uzávěr (b) Tranzitivní uzávěr

Obrázek 3.2: Grafické znázornění reflex. a tranzit. uzávěru relace r

Řekneme, že větná forma v je odvozena z u ($u \Rightarrow^* v$), pokud existuje posloupnost řetězců $u = w_1, w_2, \ldots, w_n = v$ taková, že $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$. Pokud je u startovací neterminál gramatiky a v je slovo (řetězec neterminálů), říkáme, že slovo v patří do jazyka generovaného gramatikou. Jazyk generovaný gramatikou je tedy množina všech slov, kterou lze ze startovacího symbolu pomocí pravidel gramatiky odvodit.

Definice 3.4 **Def**

Nechť $G = (N, T, P, S)$ je gramatika. Jazyk generovaný gramatikou G je množina

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

Gramatiky G_1 a G_2 nazveme jazykově ekvivalentní (zkráceně ekvivalentní), pokud generují stejný jazyk. Tedy platí $L(G_1) = L(G_2)$.

Příklad 3.2 **P**

Mějme gramatiku $G = (\{S, X, Y\}, \{0, 1\}, P, S)$ s množinou pravidel:

$$\begin{aligned} P = \{ \quad S &\to XYS, \\ S &\to \varepsilon, \\ XY &\to YX, \\ YX &\to XY, \\ X &\to 0, \\ Y &\to 1 \ \} \end{aligned}$$

Napíšeme dvě větné formy, které lze, a dvě větné formy, které nelze ze startovacího slova odvodit.

- $X10Y$ lze odvodit: $S \Rightarrow XYS \Rightarrow XYXYS \Rightarrow XYXY \Rightarrow X1XY \Rightarrow \Rightarrow X10Y$
- $X0YX1YS$ lze odvodit: $S \Rightarrow XYS \Rightarrow XYXYS \Rightarrow XYXYXYS \Rightarrow \Rightarrow XXYYXYS \Rightarrow XXYXYYS \Rightarrow X0YXYYS \Rightarrow X0YX1YS$
- $XY0S$ nelze odvodit: $S \Rightarrow XYS \Rightarrow XYXYS \Rightarrow XY0YS$ – jedno Y navíc
- SS nelze odvodit: neexistuje pravidlo, které by vygenerovalo druhý neterminál S

Nyní popíšeme prvky relace $\Rightarrow_G^*$, jinými slovy budeme zkoumat, jakou „funkci“ plní jednotlivá pravidla při generování slov jazyka.

První pravidlo $S \to XYS$ slouží ke generování dvojic neterminálů XY:

$$S \Rightarrow XYS \Rightarrow XYXYS \Rightarrow XYXYXYS \ldots,$$

prvky relace obsahují neterminál S a liší se o dvojici neterminálů XY, které přibudou přímo před neterminálem S.

Pravidlo $S \to \varepsilon$ ukončí fázi generování dvojic XY tím, že smaže neterminál S. Prvky relace při použití tohoto pravidla se liší od sebe právě o jedno S.

Třetí a čtvrté pravidlo zajistí změnu pořadí neterminálů X a Y. Prvky relace obsahují po sobě jdoucí neterminály XY nebo YX. V jednom prvku v obráceném pořadí.

Poslední dvě pravidla přepisují neterminály X a Y na terminální symboly 0 a 1. Prvky relace se liší tím, že místo neterminálu X resp. Y je ve následující větné formě symbol 0 resp. 1.

Nejkratší slovo, které gramatika generuje, je slovo ε.

Pokud se tedy zamyslíme nad tím, která slova patří do jazyka generovaného gramatikou G, zjišťujeme, že jsou to všechna slova, která obsahují stejný počet symbolů 0 jako symbolů 1.

$$L(G) = \left\{\omega \in \{0,1\}^* \mid |\omega|_0 = |\omega|_1\right\}$$

Jak funguje generování slov gramatikami, ukážeme i na dalším příkladě.

Příklad 3.3 P

Mějme gramatiku $G = (\{S\}, \{a,b\}, P, S)$ s množinou pravidel:

$$P = \{\ S \to aSb,\quad S \to ab\ \}$$

Napíšeme dvě větné formy, které lze, a tři větné formy, které nelze ze startovacího neterminálu odvodit.

- $aaSbb$ lze odvodit: $S \Rightarrow aSb \Rightarrow aaSbb$
- $aaaabbbb$ lze odvodit: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaabbbb$
- $aSSb$ nelze odvodit: $S \Rightarrow aSb \Rightarrow aaSbb$ – žádné pravidlo negeneruje dva neterminály S
- $aaSb$ nelze odvodit: každé pravidlo přidá do slova jedno a a jedno b
- $abab$ nelze odvodit: pravidla jsou koncipována tak, že generují vlevo od středu a a na pravou stranu b, nelze vygenerovat „mix“ terminálů.

Nejkratší slovo generované gramatikou G je slovo ab.

Do jazyka generovaného gramatikou G patří taková slova, která se skládají ze dvou stejně dlouhých částí, první část je složena ze symbolů a a druhá část ze symbolů b.

$$L(G) = \left\{a^i b^i \mid i \geq 1\right\}$$

Při určování jazyka, který gramatika generuje, je vhodné zapsat posloupnost prvních pár slov, která gramatika generuje, v pořadí rostoucí délky. Při této činnosti si roztřiďte pravidla na ukončující (ta, která neobsahují na pravé straně neterminál) a ostatní. Přičemž se snažte nalézt postupy, jak se pomocí použití právě těch ostatních pravidel dostanete k aplikaci pravidel ukončujících.

3.1 Omezení tvaru pravidel a Chomského hierarchie gramatik a jazyků

Přepisovací pravidla gramatik tak, jak jsme je nadefinovali v předchozí části, mohou mít téměř libovolné řetězce jak na levé tak na pravé straně. Jediným omezením je přítomnost alespoň jednoho neterminálního symbolu v levé části pravidla. V této části si ukážeme další omezení, která můžeme na tvar pravidel klást.

Otázkou omezení tvaru pravidel se zaobíral lingvista Noam Chomsky, který rozdělil gramatiky do čtyř skupin – typů označených 0, 1, 2 a 3.

Gramatika typu 0

Gramatika typu 0 je libovolná gramatika. Na tvar pravidel není kladen žádný další požadavek (kromě požadavku na existenci minimálně jednoho neterminálu na levé straně pravidla). Tyto gramatiky se někdy nazývají frázové (anglicky phrase grammars) nebo neomezené.

Příklad 3.4 **P**

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

$$P = \{ \begin{array}{rcl} S & \to & SAB, \\ AB & \to & aABb \mid \varepsilon, \\ Sa & \to & aS, \\ sSb & \to & b \end{array} \}$$

Uvedená gramatika je typu 0.

Gramatika typu 1

Gramatiky typu 1 je taková gramatika pro jejíž pravidla $\alpha \to \beta$ platí $\alpha = uAv$ a $\beta = u\gamma v$, kde $u, v \in (N \cup T)^*$, $A \in N$, $\gamma \in (N \cup T)^+$. Výjimku tvoří pravidlo $S \to \varepsilon$ pod podmínkou, že neterminál S se nevyskytuje na pravé straně žádného z pravidel. Tyto gramatiky jsou nazývány kontextové (anglicky Context-Sensitive grammars, CS).

V literatuře se také můžeme setkat s gramatikami **monotónními**. Jedná se o takové gramatiky pro jejíž pravidla $\alpha \to \beta$ platí $|\alpha| \leq |\beta|$ s výjimkou pro ε-pravidlo $S \to \varepsilon$ stejně jako je tomu uvedeno výše.

Příklad 3.5 **P**

Mějme gramatiku $G = (\{S, A\}, \{a, b\}, P, S)$, kde
$P = \{ \quad S \to aA,$
$\quad aA \to abA,$
$\quad bA \to bAa,$
$\quad aA \to ab \quad \}$
Uvedená gramatika je kontextová.

Gramatiky typu 2 jsou nazývány bezkontextové (Context-Free grammars, CF). Levou stranu pravidel tvoří pouze neterminál. Pravidla jsou tedy tvaru $A \to \beta$, kde $A \in N$, $|\beta| \geq 1$. Pro tento typ gramatiky opět platí výjimka pro pravidlo $S \to \varepsilon$. Gramatika typu 2

Posledním typem gramatik jsou **gramatiky typu 3** neboli gramatiky regulární (Regular grammars). Pravidla regulární gramatiky mohou mít pouze dva tvary: Gramatika typu 3

1. $A \to aB$ nebo
2. $A \to a$, kde $A, B \in N$ a $a \in T$ s výjimkou pro pravidlo $S \to \varepsilon$

Příklad 3.6 **P**

Mějme gramatiku $G = (\{S, A\}, \{a, b\}, P, S)$, kde
$P = \{ \quad S \to aA,$
$\quad A \to bS,$
$\quad A \to b \quad \}$
Uvedená gramatika je typu 0.

Je zřejmé, že regulární gramatika je také bezkontextová, bezkontextová gramatika splňuje podmínky kladené na kontextovou gramatiku a každá

kontextová gramatika je typu 0. Gramatika se označuje podle největšího omezení, která splňují všechna její pravidla.

Příklad 3.7 P

Rozdělte pravidla podle toho, ke gramatice kterého typu by mohli náležet.

1. $aA \to AaB$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, nepatří ani gramatice bezkontextové, protože levá strana je tvořena dvěma symboly. Pravidlo také nesplňuje podmínku pro pravidlo kontextové gramatiky. Protože platí $|aA| \leq |AaB|$, patří toto pravidlo gramatice monotónní.
2. $aA \to B$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, nepatří ani gramatice bezkontextové, protože levá strana je tvořena dvěma symboly. Protože platí $|aA| > |B|$, patří toto pravidlo gramatice typu 0.
3. $AA \to AaB$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, nepatří ani gramatice bezkontextové, protože levá strana je tvořena dvěma i když neterminálními symboly. Protože platí $u = A, v = \varepsilon$ a tím pádem $u \cdot A \cdot v \to u \cdot aB \cdot v|$, patří toto pravidlo gramatice kontextové.
4. $A \to AaB$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, patří gramatice bezkontextové, protože levá strana je tvořena jedním neterminálním symbolem.
5. $A \to Aa$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, patří gramatice bezkontextové, protože levá strana je tvořena jedním neterminálem.
6. $A \to aA$ - pravidlo patří gramatice regulární, protože je typu $A \to aB$.
7. $A \to A$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, patří gramatice bezkontextové, protože levá strana je tvořena jedním neterminálním symbolem. Takové pravidlo můžeme z gramatiky vypustit.
8. $A \to B$ - pravidlo nepatří gramatice regulární, protože není ani typu $A \to aB$ ani $A \to a$, patří gramatice bezkontextové, protože levá strana je tvořena jedním neterminálním symbolem. Takový typ pravidel nazýváme **jednoduchá pravidla**.

9. $A \to \varepsilon$ - pravidlo by mohlo patřit gramatice regulární, bezkontextové i kontextové, pokud je A startovací neterminál. V případě, že A startovacím neterminálem není, pravidlo náleží gramatice typu 0.

Jazyk L se nazývá regulární (bezkontextový, resp. kontextový), pokud existuje regulární (bezkontextová, resp. kontextová) gramatika G taková, že $L(G) = L$. Jazyky, které generují gramatiky typu 0, nazýváme jazyky typu 0 nebo také rekurzivně vyčíslitelné jazyky.

3.2 Prázdné slovo a ε-pravidla

Pozastavme se nyní nad podmínkou ε-pravidla pro startovací neterminál. Pokud bychom v definici typů gramatik vynechali dodatek o tomto pravidle, každý jazyk, který obsahuje prázdné slovo, by byl automaticky typu 0. Jinak řečeno přidání prázdného slova do regulárního jazyka by způsobilo, že nový jazyk by byl neomezený. Jak velká změna v jazyce nastane, pokud do něj toto prázdné slovo přidáme? Pokud existuje konečný popis jazyka L (a to je to, o co se snažíme, viz 2. kapitola), pak přidáním jednoduché věty „ε patří do jazyka“ získáme konečný popis jazyka $L \cup \{\varepsilon\}$.

Věta 3.1

Nechť L je kontextový, bezkontextový resp. regulární jazyk, pak i jazyky $L \cup \{\varepsilon\}$ a $L - \varepsilon$ jsou kontextové, bezkontextové resp. regulární jazyky.

Důkaz: Nejdříve vyšetříme případ, kdy jazyk L neobsahuje prázdné slovo.

Nechť $G = (N, T, P, S)$ je kontextová gramatika, pro kterou platí $L(G) = L$ a S' je neterminál, který nepatří do N.

Navrhneme gramatiku $G' = (N \cup \{S'\}, T, P', S')$ tak, že do množiny pravidel přidáme pravidla $S' \to \alpha$ pro všechna pravidla $S \to \alpha$ z množiny pravidel P. Protože $S' \notin N \cup T$, nemůže se vyskytovat na pravé straně žádného z pravidel gramatiky G'.

Nyní ukážeme, že $L(G) = L(G')$.

Předpokládejme, že v gramatice G existuje odvození $S \Rightarrow_G^* \omega$. Nechť v prvním kroku odvození je použito pravidlo $S \to \alpha$. Pak platí $S \Rightarrow_G \alpha \Rightarrow_G^* \omega$. Podle definice gramatiky G' existuje v množině P' pravidlo $S' \to \alpha$ a tedy v G' existuje odvození $S' \Rightarrow_{G'} \alpha \Rightarrow_{G'}^* \omega$. Dostáváme $L(G) \subseteq L(G')$.

Předpokládejme nyní, že v gramatice G' existuje odvození $S' \Rightarrow_{G'}^* \omega$. První použité pravidlo je $S' \to \alpha$. Pak platí $S' \Rightarrow_{G'} \alpha \Rightarrow_{G'}^* \omega$. Podle definice gramatiky G' se neterminál S' nevyskytuje na pravé straně žádného pravidla a proto nelze v gramatice G' odvodit větnou formu obsahující S'. Také v P existuje pravidlo $S \to \alpha$ a tedy v G existuje odvození $S \Rightarrow_G \alpha \Rightarrow_G^* \omega$. Dostáváme $L(G') \subseteq L(G)$.

Protože platí $L(G) \subseteq L(G')$ a současně $L(G') \subseteq L(G)$, pak $L(G) = L(G')$.

Nalezli jsme tedy gramatiku, která je jazykově ekvivalentní s původní gramatikou, její startovací neterminál se nevyskytuje na pravé straně žádného z pravidel a nová gramatika je stejného typu jako gramatika původní.

Pokud chceme vytvořit kontextovou gramatiku G_1 jazyka $L \cup \{\varepsilon\}$, stačí do gramatiky G' přidat pravidlo $S' \to \varepsilon$. Výsledná gramatika $G_1 = (N', T, P' \cup \{S' \to \varepsilon\}, S')$ generuje slovo ε při použití pravidla $S' \to \varepsilon$ jako jediného při odvození slova, nebo při použití jakéhokoli jiného pravidla generuje slova z jazyka $L(G')$. Platí tedy $L(G_1) = L(G') \cup \{\varepsilon\}$.

Pokud kontextová gramatika $G = (N, T, P, S)$ generuje jazyk, který obsahuje prázdné slovo, pak musí obsahovat pravidlo $S \to \varepsilon$ a S se nesmí vyskytovat na pravé straně žádného pravidla gramatiky G. Vytvoříme gramatiku $G' = (N, T, P - \{S \to \varepsilon\}, S)$. Protože pravidlo $S \to \varepsilon$ nelze použít k odvození jiného než prázdného slova, dostáváme $L(G') = L(G) - \{\varepsilon\}$.

Pokud je jazyk L bezkontextový nebo regulární, důkaz je analogický.

□

Na základě věty 3.1 lze postupovat při tvorbě gramatik tak, že pokud jazyk L obsahuje prázdné slovo, vytvoříme nejdříve gramatiku generující jazyk $L - \{\varepsilon\}$ a pak gramatiku doplníme o pravidlo pro prázdné slovo (a nový startovací neterminál).

Příklad 3.8 **P**

Zapište gramatiku generující aritmetický výraz s identifikátory a operátory + a −.

Množina terminálních symbolů gramatiky generující aritmetické výrazy je tvořena identifikátorem id a operátory + a −. Množinu neterminálů sta-

novíme až po sestrojení pravidel gramatiky. Začneme prvním pravidlem, které bude mít na levé straně startovací neterminál.

Nejkratší aritmetický výraz obsahuje pouze identifikátor. Pravidlo, které jej generuje, má tvar $S \to id$. Pravidla pro generování součtu a rozdílu jsou analogická a mají tvar $S \to S + S$ a $S \to S - S$. Sestrojili jsme gramatiku $G_1 = (\{S\}, \{id, +, -\}, \{S \to id \mid S + S \mid S - S\}, S)$ Tato gramatika je bezkontextová.

Vytvořme odvození v gramatice G_1, které bude generovat slovo $id + id - id + id$ postupně zleva doprava.

$$\begin{aligned} S &\Rightarrow S + S \Rightarrow id + S \Rightarrow id + S - S \Rightarrow id + id - S \Rightarrow id + id - S + S \Rightarrow \\ &\Rightarrow id + id - id + S \Rightarrow id + id - id + id \end{aligned}$$

V každém kroku, kdy gramatika generuje + nebo −, vznikají vždy dva neterminály S, z nichž první v následujícím kroku přepíšeme na terminál id.

Navrhněme teď pravidla mírně změněná: $S \to id \mid id + S \mid id - S$ a opět vytvořme odvození pro slovo $id + id - id + id$.

$$S \Rightarrow id + S \Rightarrow id + id - S \Rightarrow id + id - id + S \Rightarrow id + id - id + id$$

Vidíme, že odvození má menší počet kroků a v každém kroku vzniká nejvýše jeden neterminál. Gramatika $G_2 = (\{S\}, \{id, +, -\}, \{S \to id \mid id + S \mid id - S\}, S)$ je opět bezkontextová.

Protože každé pravidlo, které není ukončující, obsahuje nanejvýš jeden neterminál a ten je situován na konec pravé strany pravidla, je možné pravidla změnit tak, aby byla regulární. Pro odvození $S \Rightarrow_{G_2} id + S$ a $S \Rightarrow_{G_2} id - S$ vytvoříme regulární pravidla taková, aby platilo $S \Rightarrow^*_{G_2} id + S$ a $S \Rightarrow^*_{G_2} id - S$. Nejprve pravidlo typu $A \to aB$, které přepíše neterminál S. Protože odvozujeme slovo zleva doprava, musí pravá strana pravidla začínat terminálem id. Po něm musí následovat neterminál. Nemůže to být již existující neterminál S, protože tím bychom dovolili gramatice generovat slova typu $(id)^i$. Nové pravidlo bude následující: $S \to idA$. V dalším kroku odvození musí být neterminál A přepsán na druhý terminální symbol tzn. + nebo −. Přidáme tedy pravidlo $A \to +S$ a $A \to -S$. Nově vzniklá gramatika $G_3 = (\{S, A\}, \{id, +, -\}, \{S \to id \mid idA,\ A \to +S \mid -S\}, S)$ je jazykově ekvivalentní s předchozími gramatikami (důkaz ponecháme na čtenáři) a je regulární.

K poznatkům teorie formálních jazyků patří tvrzení, které definuje tzv. Chomského hierarchii formálních jazyků.

Chomského hierarchie jazyků

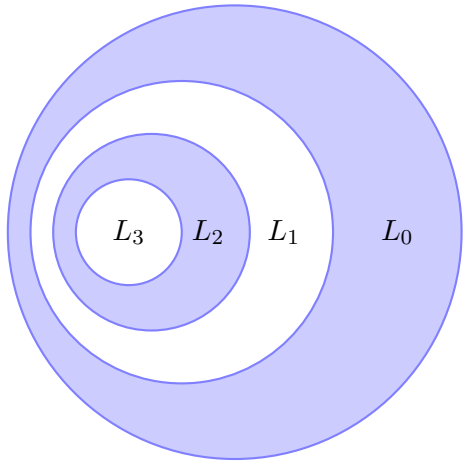
Věta 3.2

V

Nechť $L_i,\ i = 1, 2, 3$ značí třídu všech jazyků typu i. Pak platí

$$L_0 \supset L_1 \supset L_2 \supset L_3.$$

Pokud budeme chtít znázornit vztah z předchozí věty (3.2), dostáváme následující diagram:



Obrázek 3.3: Chomského hierarchie jazyků

Úkoly

1. Zapište gramatiku pro aritmetický výraz:
   (a) s identifikátory a operátory +, $*$
   (b) s identifikátory a operátory +, $*$, $\hat{}$ a závorkami
   (c) s identifikátory a operátory +, $*$, $\hat{}$, unárním +, – a závorkami

2. Uvažujte gramatiku G s množinou neterminálů $N = \{S, A, B\}$, množinou terminálů $T = \{0, 1\}$, startovacím symbolem S a s pravidly:

   $$\begin{array}{lcl} S & \to & 0A1 \mid 1B0 \\ A & \to & 1A0 \mid 1 \\ B & \to & 1B \mid 1. \end{array}$$

   Označme L jazyk generovaný gramatikou G. Rozhodněte o pravdivosti následujících tvrzení:

   (a) L je konečný jazyk.
   (b) L obsahuje prázdné slovo.
   (c) L obsahuje právě dvě slova délky nejvýše 3.
   (d) L obsahuje pouze slova liché délky.
   (e) Každé slovo z L obsahuje znak 0.
   (f) Každé slovo z L obsahuje alespoň dva znaky 0.
   (g) Žádné slovo z L neobsahuje souvislý řetězec 01010.
   (h) Existuje slovo z L, které obsahuje řetězec 11011.
   (i) Pokud dvě slova z L mají stejnou délku, pak začínají stejným symbolem.
   (j) Pokud dvě slova z L začínají stejným symbolem, pak mají stejnou délku.

3. Ověřte, že jazyk $(10)^+$ lze generovat regulární gramatikou.

Kapitola 4

Konečné automaty a regulární jazyky

Zatímco gramatiky představují generativní zařízení, tj. umožňují vygenerovat celý jazyk, opačný přístup, tzv. akceptující, reprezentují automaty: automat „analyzuje“ řetězec předložený na jeho vstupu. Pokud tento řetězec patří do jazyka, automat signalizuje tuto skutečnost dohodnutým způsobem.

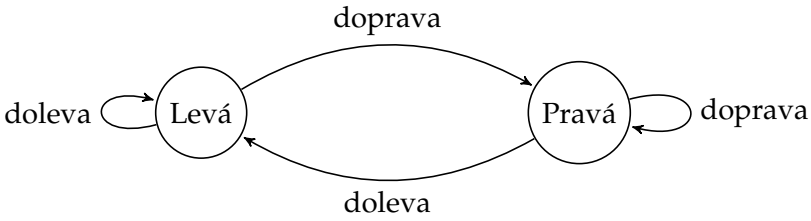
Rozdílnost generativního a akceptujícího přístupu vynikne zejména při řešení úlohy, zda konkrétní slovo ω patří do zadaného jazyka L. Pokud je L zadán např. gramatikou G typu 0, která jej generuje, jediný obecně použitelný postup spočívá v postupném generování slov z jazyka $L(G)$ a jejich porovnávání s ω. Pokud je však L určen automatem A, stačí dát slovo ω automatu A na jeho vstup a čekat, jestli ho automat akceptuje.

Existuje velké množství automatů – nazývaných též také stroje, např. konečné automaty, Mealyho automaty, Mooreovy stroje, zásobníkové automaty, Turingovy stroje, stroje RAM, ... Základní rozdíl mezi jednotlivými typy spočívá v možnosti využívat nějaký druh paměťového zařízení, případně v jeho uspořádání a způsobu práce s uloženými daty.

V této kapitole se budeme zaobírat konečnými automaty a jejich vztahem k regulárním jazykům.

4.1 Konečný automat

Konečný automat je nejjednodušší z formálních modelů počítače. V reálném světě se velmi často setkáváme s takovými přístroji, jejichž funkce sestává z přijímání informací s okolím a reakce na ně. Jednoduchým příkladem může být výhybka. Výhybka je drážní zařízení, které se nachází v místě, kde se dráhy rozcházejí nebo sbíhají, a umožňuje jízdu např. vlaku ve správném směru. Uvažujme ilustrativní situaci, kdy se se dráha rozdvojuje a vlak jedoucí po koleji může pokračovat vlevo nebo vpravo. Stejně tak se výhybka může nacházet ve stavu, kdy umožňuje vjezd na levou nebo pravou kolej. Podle pokynu, na kterou kolej vlak pojede, zůstane výhybka buď ve stávajícím stavu nebo se přestaví na stranu opačnou. Její funkci lze jednoduše popsat následujícím diagramem.



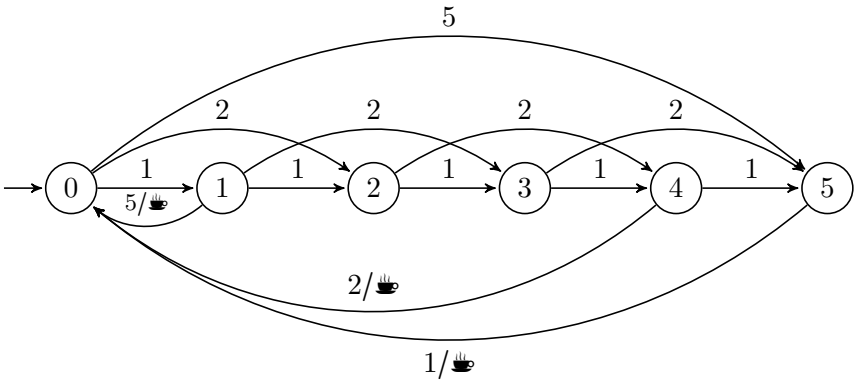
Obrázek 4.1: Grafické znázornění funkce výhybky

Popíšeme nyní konečný automat neformálně. Je to abstraktní stroj, který má konečný počet stavů, v nichž se může nacházet. Konečný automat zpracovává posloupnost symbolů, které jsou umístěné na vstupní pásce. Taková posloupnost se nazývá vstupní slovo. V reálné situaci se nemusí nutně jednat o posloupnost psaných symbolů, ale například o posloupnost vhazovaných mincí a stisknutých tlačítek. Automat pracuje v jednotlivých krocích. V každém kroku přečte jeden symbol vstupního slova a na základě stavu, ve kterém se nachází, a právě čteného symbolu změní svůj stav.

Mezi stavy, kterých je vždy konečně mnoho, existuje jeden, který se nazývá počáteční. Je to stav, ve kterém automat vstupní slovo začíná číst. Dalšími speciálními stavy jsou ty, které se nachází v množině koncových stavů.

Po přečtení posledního symbolu vstupního slova se automat nachází buď v koncovém stavu, pak říkáme, že automat slovo přijímá, nebo ve stavu, který koncový není, kdy slovo nepřijímá.

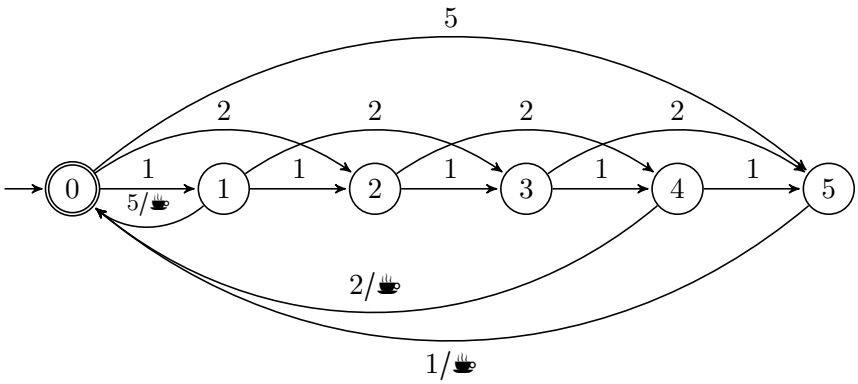
Představme si jednoduchý automat na kávu, který umí připravit jen jeden druh kávy. Protože se jedná o starší model, automat mince nevrací a přijímá pouze přesnou částku. Automat, který je v klidovém režimu, se nachází v počátečním stavu. Označme tento stav 0, protože nebyla vhozena žádná mince. Nechť šálek kávy má hodnotu 6 Kč a automat rozezná mince v hodnotě 1, 2 a 5 Kč – to je jeho vstupní abeceda. Automat bude akceptovat takovou posloupnost vhazovaných mincí, které budou mít celkovou hodnotu rovnu 6 Kč. Vytvořme diagram (obrázek 4.2), který bude zachycovat přechody mezi stavy automatu tak, jak budou postupně vhazovány mince. Stavy automatu budou zachycovat hodnotu již vhozených mincí.



Obrázek 4.2: Grafické znázornění automatu na kávu

Značka ☕ za hodnotou právě vhozené mince, která je umístěná nad šipkami, znamená, že automat vydal kávu. Šipka směřující ke stavu 0 označuje počáteční stav.

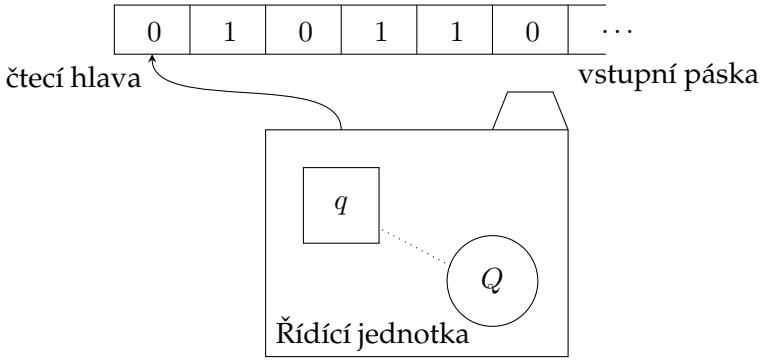
Aby byl automat úplný, je zapotřebí rozhodnout, které stavy budou koncové. Připomeňme, že aby automat slovo přijal, musí se po zpracování celého slova nacházet právě v jednom z koncových stavů. Protože automat vydá kávu pouze pokud součet hodnoty vhozených mincí je roven 6 Kč a automat vydá kávu, bude koncovým stavem stav 0. Do jazyka, který automat přijímá, tak patří i prázdné slovo, což není v rozporu s funkcí automatu na kávu. V diagramu označíme koncový stav dvojitou kružnicí (viz obrázek 4.3).



Obrázek 4.3: Grafické znázornění automatu na kávu s koncovým stavem

Konečný automat je sice abstraktní stroj, můžeme si ho však představit jako zařízení, které obsahuje řídící jednotku (konečněstavovou), čtecí hlavu a vstupní pásku (viz obrázek 4.4).

Neformálně popíšeme činnost konečného automatu: Na začátku výpočtu je na vstupní pásce zapsáno vstupní slovo (to slovo, u kterého potřebujeme určit, jestli patří do daného jazyka). Slovo se skládá za symbolů abecedy, která se nazývá vstupní. Páska je rozdělená na jednotlivá políčka, do každého políčka je zapsán jeden symbol slova. Automat je na počátku výpočtu ve stavu, který se nazývá počáteční, a každý automat může mít pouze jeden počáteční stav. Čtecí hlava je nastavena na první políčko (nejvíce vlevo).



Obrázek 4.4: Zařízení „konečný automat“

Výpočet probíhá v jednotlivých krocích. V každém kroku se opakují následující činnosti: Automat načte symbol ze vstupní pásky, na základě tohoto symbolu a aktuálního stavu změní svůj stav a posune čtecí hlavu o jedno políčko vpravo. To, jaký stav bude novým aktuálním stavem, je určeno přechodovou funkcí, která je součástí definice automatu. Pokud automat přečte celé slovo, čtecí hlava je nyní na prvním prázdném políčku za políčkem, které obsahuje poslední symbol vstupního slova. Nyní zkoumáme, v jakém stavu se nachází automat. Pokud je to jeden ze stavů, které jsou označeny jako koncové, výpočet skončil úspěšně a slovo do jazyka patří – podle předchozího příkladu byla vydána káva. V jiném případě (stav není koncový nebo automat slovo nedočetl) výpočet skončil neúspěšně a vstupní slovo do jazyka nepatří – milovník kávy se tedy svého oblíbeného nápoje nedočká.

Uvedeme nyní formální definici.

Definice 4.1

Def

Konečný automat

Konečný automat (Finite Automaton, FA) M je pětice $(Q, \Sigma, \delta, q_0, F)$, kde

- Q je neprázdná konečná množina stavů;
- Σ je konečná množina vstupních symbolů, nazývaná také vstupní abeceda;
- $\delta : Q \times \Sigma \to Q$ je parciální přechodová funkce;
- $q_0 \in Q$ je počáteční stav;
- $F \subseteq Q$ je množina koncových stavů.

Přechodová funkce δ konečného automatu definuje chování automatu při čtení vstupního symbolu. Na základě aktuálního stavu $q \in Q$ a právě čteného symbolu vstupního slova $a \in \Sigma$ automat změní svůj stav na $q' \in Q$ a posune čtecí hlavu o jedno políčko doprava.

Konečný automat je také často definován stavovým diagramem. Je to orientovaný ohodnocený graf doplněný o speciální znaky. Uzly grafu představují stavy automatu a orientované (směr určuje šipka) ohodnocené (hodnotu udává popisek) hrany zachycují přechodovou funkci. Počáteční stav označíme šipkou, která nemá ohodnocení a směřuje do tohoto stavu z prostředí automatu, koncový stav je označen dvojitým ohraničením. Takový stavový diagram je uveden např. na obrázku 4.3.

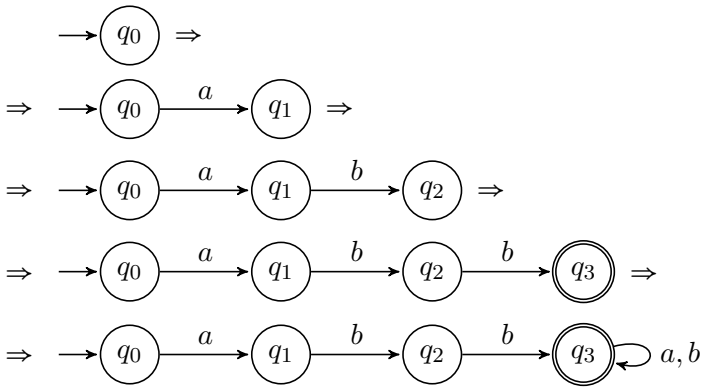
Třetí způsob definice konečného automatu je jeho zadání tabulkou. V záhlaví (označení) sloupců jsou uvedeny symboly vstupní abecedy a v záhlaví řádků pak stavy automatu. Do buněk tabulky jsou doplněny stavy tak, aby kombinace označení sloupce a řádku a obsahu buňky na jejich průsečíku odpovídala přechodové funkci. Počáteční stav je označen šipkou, která ukazuje na označení stavu a šipky vedoucí od označení stavů pak označují stavy koncové.

Příklad 4.1 P

Mějme konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ definovaný následovně:

- $Q = \{q_0, q_1, q_2, q_3\}$;
- $\Sigma = \{a, b\}$;
- $F = \{q_3\}$;
- $\delta(q_0, a) = q_1$
  $\delta(q_1, b) = q_2$
  $\delta(q_2, b) = q_3$
  $\delta(q_3, a) = q_3$
  $\delta(q_3, b) = q_3$

Stavový diagram sestavíme tak, že nejdříve zakreslíme počáteční stav. Dále budeme pokračovat tak, že podle přechodové funkce budeme přidávat šipky a uzly.



Tabulka přechodové funkce je tvořena tolika sloupci, kolik symbolů obsahuje vstupní abeceda, a tolika řádky, kolik stavů má automat.

		a	b
→	q_0	q_1	
	q_1		q_2
	q_2		q_3
←	q_3	q_3	q_3

Situace, ve které se konečný automat nachází, je možné jednoznačně popsat aktuálním stavem a dosud nepřečtenou částí pásky. Tato dvojice tvoří konfiguraci konečného automatu.

Definice 4.2

Def

Konfigurace FA

Konfigurace konečného automatu $M = (Q, \Sigma, \delta, q_0, F)$ je každá dvojice

$$(q, \omega) \in Q \times \Sigma^*,$$

kde q je aktuální stav konečného automatu M a ω je nepřečtená část vstupní pásky.

Počáteční konfigurace je taková konfigurace, která obsahuje počáteční stav: (q_0, w), $w \in \Sigma^*$. Koncová konfigurace je taková, kdy je automat v koncovém stavu a přečetl celé vstupní slovo: (q_f, ε), $q_f \in F$. Na množině všech konfigurací konečného automatu M zavedeme binární relaci krok výpočtu.

Definice 4.3

Def

Krok výpočtu FA

Krok výpočtu konečného automatu $M = (Q, \Sigma, \delta, q_0, F)$ je binární relace $\vdash_M \in (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ definovaná předpisem

$$(q, aw) \vdash_M (p, w) \quad \overset{def}{\Leftrightarrow} \quad \delta(q, a) = p,$$

kde (q, aw) a (p, w) jsou dvě konfigurace konečného automatu M, $q, p \in Q$, $a \in \Sigma$ a $w \in \Sigma^*$.

Symbolem $\vdash_M^k$ označíme k-tou mocninu relace $\vdash_M$, tranzitivní uzávěr relace $\vdash_M$ značíme $\vdash_M^+$. Reflexivní a tranzitivní uzávěr relace kroku výpočtu značíme $\vdash_M^*$.

Pokud je z kontextu zřejmé, o který konečný automat se jedná, budeme u relací $\vdash_M$, $\vdash_M^k$, $\vdash_M^+$ a $\vdash_M^*$ index M vynechávat.

Definice 4.4

Def

Jazyk akceptovaný FA

Jazyk přijímaný (akceptovaný, rozpoznávaný) konečným automatem $M = (Q, \Sigma, \delta, q_0, F)$, označovaný $L(M)$, je tvořen právě takovými slovy, po jejichž přečtení se automat nachází v koncové konfiguraci:

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

Dva konečné automaty nazveme **jazykově ekvivalentní** (zkráceně ekvivalentní), pokud se jazyky jimi akceptované rovnají. Tedy konečné automaty M a M' jsou ekvivalentní, pokud $L(M) = L(M')$.

Ekvivalence konečných automatů

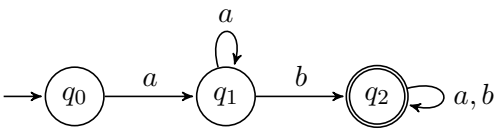
Příklad 4.2

P

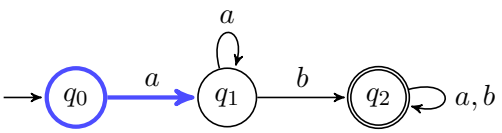
Mějme konečný automat $M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$, kde δ je definováno následovně:

$$\begin{array}{llll} \delta(q_0, a) = q_1 & & \delta(q_1, b) = q_2 \\ \delta(q_1, a) = q_1 & & \delta(q_2, b) = q_2 \\ \delta(q_2, a) = q_2 & & \end{array}$$

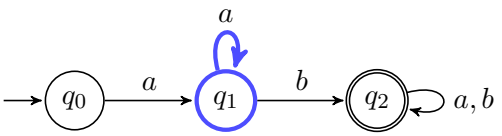
Abychom si lépe představili, jaká slova konečný automat přijímá, uvedeme jeho stavový diagram.



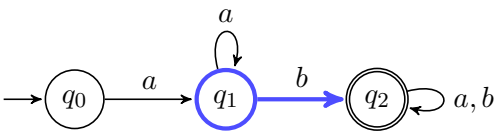
Nyní ukážeme, jak bude vypadat zpracování vstupního slova $aaba$ konečným automatem M. Počáteční konfigurace automatu je $(q_0, aaba)$. Konečný automat se nachází ve stavu q_0 a na vstupu čte první symbol slova $aaba$, symbol a.



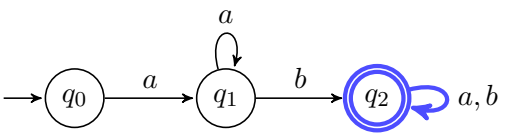
Protože pro stav q_0 je přechodová funkce definovaná $\delta(q_0, a) = q_1$, tak $(q_0, aaba) \vdash (q_1, aba)$ je krok výpočtu konečného automatu M z počáteční konfigurace. Po prvním kroku se automat nachází v konfiguraci (q_1, aba).



Automat se tedy nachází ve stavu q_1, na vstupu čte symbol a a podle přechodové funkce zůstává ve stavu q_1. Druhý krok výpočtu: $(q_1, aba) \vdash (q_1, ba)$. Nová konfigurace je (q_1, ba).

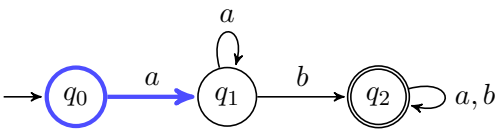


Konečný automat se nachází ve stavu q_1, na vstupu čte symbol b a podle přechodové funkce přechází do stavu q_2. Třetí krok výpočtu: $(q_1, ba) \vdash (q_2, a)$. Nová konfigurace je (q_2, a).

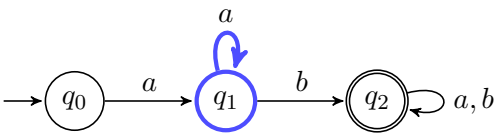


Konečný automat se nachází ve stavu q_2, na vstupu čte symbol a a podle přechodové funkce zůstává ve stavu q_2. Čtvrtý krok výpočtu: $(q_2, a) \vdash (q_2, \varepsilon)$. Nová konfigurace je (q_2, ε). Konečný automat přečetl slovo celé a skončil v koncovém stavu. Slovo $aaba$ tedy patří do jazyka rozpoznávaného konečným automatem M, $aaba \in L(M)$.

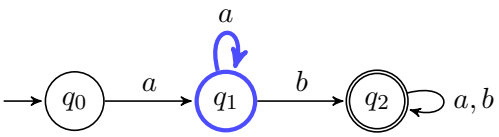
Jako další vstupní slovo použijeme slovo aaa a zjistíme, jestli ho konečný automat M akceptuje. Počáteční konfigurace automatu je (q_0, aaa). Konečný automat se nachází ve stavu q_0 a na vstupu čte první symbol slova aaa, symbol a.



Automat se nachází ve stavu q_0, na vstupu čte symbol a a podle přechodové funkce přechází do stavu q_1. První krok výpočtu: $(q_0, aaa) \vdash (q_1, aa)$. Nová konfigurace je (q_1, aa).

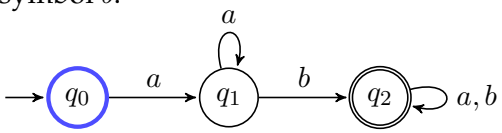


Automat se tedy nachází ve stavu q_1, na vstupu čte symbol a a podle přechodové funkce zůstává ve stavu q_1. Druhý krok výpočtu: $(q_1, aa) \vdash (q_1, a)$. Nová konfigurace je (q_1, a).



Konečný automat se nachází ve stavu q_1, na vstupu čte třetí symbol a a podle přechodové funkce opět zůstává ve stavu q_1. Třetí krok výpočtu: $(q_1, a) \vdash (q_1, \varepsilon)$. Nová konfigurace je (q_1, ε). Konečný automat přečetl slovo celé a skončil ve stavu jiném než koncovém. Slovo aaa tedy nepatří do jazyka rozpoznávaného konečným automatem M, $aaa \notin L(M)$.

Jako poslední vstupní slovo použijeme slovo baa a zjistíme, jestli ho konečný automat M akceptuje. Počáteční konfigurace automatu je (q_0, baa). Konečný automat se nachází ve stavu q_0 a na vstupu čte první symbol slova baa, symbol b.



Protože kkonečný automat M pro tuto situaci nemá nadefinovanou přechodovou funkci, zůstává stát. Výpočet skončil aniž by automat dočetl vstupní slovo do konce a platí $baa \notin L(M)$.

Vraťme se ještě k příkladu 4.1 a konečnému automatu v něm uvedenému. Prázdná místa v tabulce přechodové funkce znamenají, že pro odpovídající kombinaci stavu a vstupního symbolu není definována přechodová funkce. Pokud se při výpočtu dostane automat do stavu q_1, ze vstupní pásky načítá symbol a, nemá definovanou přechodovou funkci. Výpočet skončí neúspěšně aniž by automat dočetl slovo do konce. Tato situace není pro programátory příliš vhodná. Představte si, že máte slovo zapsáno např. na děrné pásce. Pokud naprogramovaný automat nemá pro některou kombinaci stavu a vstupního symbolu nadefinovanou přechodovou funkci, může se stát, že se výpo-

čet zastaví uprostřed slova a páska zůstane uvnitř stroje. Jiným příkladem je načítání jakéhokoli vstupu programem. Uprostřed načítání program zastaví svou činnost a dále nereaguje. Je proto výhodnější definovat přechodovou funkci takovým způsobem, aby byl automat schopen zareagovat vždy (pro každou kombinaci stav, symbol). Taková přechodová funkce se nazývá **totální**. Konečný automat s totální přechodovou funkcí se nazývá **úplný**.

Totální přechodová funkce a úplný FA

Věta 4.1 **V**

Ke každému konečnému automatu M existuje ekvivalentní konečný automat M', který je úplný.

Idea důkazu: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat, který není úplný. Sestrojíme konečný automat $M' = (Q', \Sigma, \delta', q_0, F)$ tak, že do množiny stavů přidáme nový nekoncový stav $p \notin Q$, $Q' = Q \cup \{p\}$. Tímto stavem doplníme prvky přechodové funkce. Přechodovou funkci δ' definujeme pro všechny $q \in Q'$ a $a \in \Sigma$ takto:

$$\delta'(q,a) = \begin{cases} \delta(q,a) \\ p, \text{ pokud } \delta(q,a) \text{ neexistuje} \end{cases}$$

Protože je přechodová funkce δ' definovaná pro všechny dvojice (q, a) je konečný automat M' úplný. Pro každé slovo $w \in L(M)$ existuje posloupnost kroků výpočtu konečného automatu M, která končí v koncové konfiguraci. Pak existuje posloupnost stejných kroků výpočtu v automatu M', která způsobí akceptování slova w a tudíž $w \in L(M')$. Protože neexistuje posloupnost kroků, která by převedla automat M' z nového stavu p do stavu koncového, nevznikla přidáním hodnot přechodové funkce žádná posloupnost kroků taková, aby platilo $w \notin L(M) \wedge w \in L(M')$ a konečné automaty M a M' jsou jazykově ekvivalentní. □

Příklad 4.3 **P**

Ke konečnému automatu z příkladu 4.1 vytvořte ekvivalentní úplný konečný automat. Konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ je definovaný následovně:

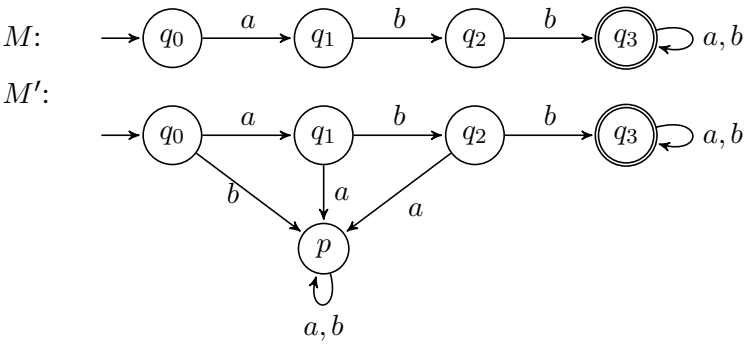
- $Q = \{q_0, q_1, q_2, q_3\}$;
- $\Sigma = \{a, b\}$;

- $F = \{q_3\}$;
- $\delta(q_0, a) = q_1$
  $\delta(q_1, b) = q_2$
  $\delta(q_2, b) = q_3$
  $\delta(q_3, a) = q_3$
  $\delta(q_3, b) = q_3$

Pokud sestrojíme automat M' podle postupu uvedeného ve větě 4.1, dostáváme následující: $M' = (Q \cup \{p\}, \Sigma, \delta', q_0, F)$ je definován následovně:

- $\delta(q_0, a) = q_1$ $\quad$ $\delta(q_0, b) = p$
  $\delta(q_1, b) = q_2$ $\quad$ $\delta(q_1, a) = p$
  $\delta(q_2, b) = q_3$ $\quad$ $\delta(q_2, a) = p$
  $\delta(q_3, a) = q_3$ $\quad$ $\delta(p, a) = p$
  $\delta(q_3, b) = q_3$ $\quad$ $\delta(p, b) = p$

Dodefinování nového stavu a chybějících hodnot přechodové funkce vypadá ve stavovém diagramu následujícím způsobem: (v prvním řádku je původní automat, ve druhém automat úplný)



Pro úplnost uvedeme tabulku přechodové funkce obou automatů:

M:

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

M':

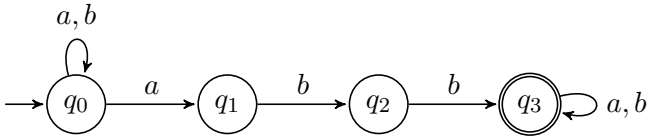
		a	b
$\rightarrow$	q_0	q_1	p
	q_1	p	q_2
	q_2	p	q_3
$\leftarrow$	q_3	q_3	q_3
	p	p	p

4.2 Nedeterministický konečný automat

V této části zavedeme pojem nedeterministický konečný automat. Je to užitečný pojem při návrhu konečných automatů, které rozpoznávají daný jazyk, bude užitečný v důkazech vět.

Konečný automat definovaný v předchozí části se nazývá deterministický, protože v každé konfiguraci existuje nejvýše jeden způsob jakým pokračovat ve výpočtu. Jinak řečeno stavem a čteným symbolem je jednoznačně určen stav, do něhož má automat přejít. Na rozdíl od konečného automatu nedeterministického, který se v některých konfiguracích musí rozhodnout mezi více variantami, jak pokračovat ve výpočtu.

Na obrázku 4.5 uvádíme jednoduchý příklad přechodového diagramu nedeterministického konečného automatu .



Obrázek 4.5: Příklad nedeterministického konečného automatu

Nedeterminismus konečného automatu spočívá v definici přechodové funkce, pomocí které může automat přejít z jednoho stavu přečtením stejného symbolu do různých stavů. V našem případě se jedná o stav q_0, ze kterého je možné přečtením symbolu a přejít do stavu q_1 nebo zůstav ve stavu q_0. Jak se tato skutečnost projeví v tabulce přechodové funkce, si ukážeme na stejném automatu.

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

Jistě si každý všimne, že se v jedné buňce nachází dva stavy. Nyní uvedeme formální definici nedeterministického konečného automatu.

Definice 4.5

Def
Nedeterministický konečný automat

Nedeterministický konečný automat (Nondeterministic Finite Automaton, NFA) M je pětice $(Q, \Sigma, \delta, q_0, F)$, kde

- Q je neprázdná konečná množina stavů;
- Σ je konečná množina vstupních symbolů, nazývaná také vstupní abeceda;
- $\delta : Q \times \Sigma \to 2^Q$ je parciální přechodová funkce;
- $q_0 \in Q$ je počáteční stav;
- $F \subseteq Q$ je množina koncových stavů.

Poznámka: Množina 2^Q je množina všech podmnožin množiny Q, nazývá se potenční množina. $2^Q = \{A \mid A \subseteq Q\}$

Formální definice konečného automatu, jehož tabulku přechodové funkce jsme uvedli výše vypadá takto:
$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\})$ s přechodovou funkcí
$\delta(q_0, a) = \{q_0, q_1\}$
$\delta(q_0, b) = \{q_0\}$
$\delta(q_1, b) = \{q_2\}$
$\delta(q_2, b) = \{q_3\}$
$\delta(q_3, a) = \{q_3\}$
$\delta(q_3, b) = \{q_3\}$

Krok výpočtu nedeterministického konečného automatu definujeme jako binární relaci $\vdash$ (na množině všech konfigurací) takto: Pro všechna $p, q \in Q$, $a \in \Sigma$, $x \in \Sigma^*$ je $(p, ax) \vdash (q, x)$, pokud $\delta(p, a)$ obsahuje q ($q \in \delta(p, a)$), kde x je řetězec označující zbývající část větné formy.

Reflexivní a tranzitivní uzávěr označíme $\vdash^*$.

V případě, že $\delta(q, a) = \{p_1, p_2, p_3\}$, pak v relaci kroku výpočtu jsou následující konfigurace: $(q, ax) \vdash (p_1, x)$, $(q, ax) \vdash (p_2, x)$ a $(q, ax) \vdash (p_3, x)$. NFA v takovém případě může pokračovat ve výpočtu třemi možnými způsoby. Jak se tato situace projeví na akceptování slov a definici jazyka rozpoznávaného NFA? Řekneme, že NFA $M = (Q, \Sigma, \delta, q_0, F)$ přijímá slovo $w \in \Sigma^*$, právě když existuje alespoň jeden výpočet (posloupnost kroků) začínající v konfiguraci (q_0, w), který končí v některé koncové konfiguraci.

Jazyk akceptovaný NFA je tvořen všemi akceptovanými slovy w nad abecedou Σ. Pro jedno slovo, které náleží do jazyka akceptovaného NFA, může tedy existovat výpočet (i více výpočtů), který neskončil úspěšně. Důležitá je ta skutečnost, že alespoň jeden akceptující výpočet existuje.

Věta 4.2 V

Ke každému nedeterministickému konečnému automatu $M = (Q, \Sigma, \delta, q_0, F)$ existuje jazykově ekvivalentní deterministický konečný automat.

Idea důkazu:
Pro sestavení deterministického FA (DFA) $M' = (Q', \Sigma, \delta', q_0', F')$ vytvoříme označení takové, že každý stav DFA bude tvořen složenými množinou stavů původního NFA.

Počáteční stav $q_0' = \{q_0\}$. Budeme nyní tvořit přechodovou funkci DFA a zároveň doplňovat množinu stavů nového automatu. Pro každý nový stav DFA $\{$stavy NFA$\}$, budeme zaznamenávat, jak se změní jednotlivé stavy NFA, které stav DFA obsahuje, načtením vstupního symbolu. Takto vytvořená množina stavů je stavem DFA. Koncovým stavem DFA bude takový stav, který obsahuje alespoň jeden koncový stav původního NFA. □

Pro lepší pochopení principu převodu NFA na DFA je dobré představit si nedeterministický výpočet takto: spustíme výpočet automatu na daném vstupním slově. Pokud nastane situace, kdy automat může zareagovat na vstupní symbol více způsoby, spustíme další automaty tak, aby každý z nich použil jinou možnost, to znamená, každý z nich bude po načtení aktuálního symbolu v jiném stavu. Dále pak budou všechny tyto automaty pracovat paralelně. Pokud v průběhu výpočtu je v aktuální konfiguraci více automatů ve stejném stavu, můžeme z nich vybrat jen jeden, který bude dále zpracovávat vstup, ostatní mohou končit činnost. Stejně tak ukončí činnost i automat, který na daný vstupní symbol neumí zareagovat (neexistuje příslušná přechodová funkce).

V algoritmu použijeme následující označení:

- $Done$ – množina obsahující prozkoumané stavy DFA, tedy jsou takové stavy, pro které je již stanovena přechodová funkce, každý prvek množiny $Done$ je množina stavů NFA, $Done \subseteq Q'$,

- A – právě zpracovávaný stav DFA, jedná se o množinu stavů NFA, $A \in Q'$,
- N – nově vytvořený stav DFA, opět se jedná o množinu stavů NFA, $N \in Q'$,
- p – stav NFA, $p \in Q$.

Nejdříve uvedeme formální popis algoritmu převodu NFA na DFA a poté si převod ukážeme na konkrétním příkladě.

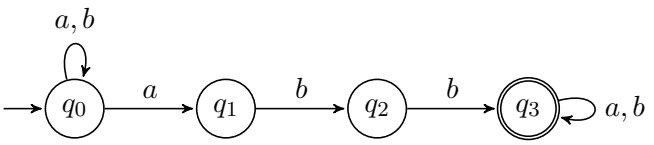
Algoritmus 4.1 Transformace NFA na ekvivalentní DFA

A

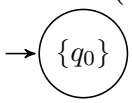
Vstup: Nedetministický konečný automat $M = (Q, \Sigma, \delta, q_0, F)$
Výstup: Ekvivalentní deterministický konečný automat $M' = (Q', \Sigma, \delta', \{q_0\}, F')$

$Q' := \{\{q_0\}\}$; $\delta' := \varnothing$; $F' := \varnothing$; $Done := \varnothing$;
dokud $(Q' - Done) \neq \varnothing$ dělej
  pro $A \in Q' - Done$;
  jestliže $A \cap F \neq \varnothing$ pak
    $F' := F' \cup \{A\}$;
  konec jestliže
  pro všechna $a \in \Sigma$ dělej
    $N := \bigcup_{p \in A} \delta(p, a)$;
    $Q' := Q' \cup \{N\}$;
    $\delta' := \delta' \cup \{((A, a), N)\}$;
  konec pro všechna
  $Done := Done \cup \{A\}$;
konec dokud
$M' := (Q', \Sigma, \delta', \{q_0\}, F')$;

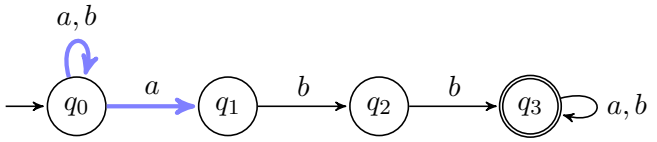
Mějme nedeterministický automat uvedený výše. Znovu zde uvedeme jeho přechodový diagram.



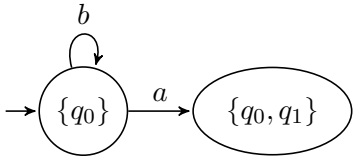
Postupně budeme vytvářet ekvivalentní deterministický konečný automat $M' = (Q', \Sigma, \delta', \{q_0\}, F')$. Nejdříve vytvoříme počáteční stav.



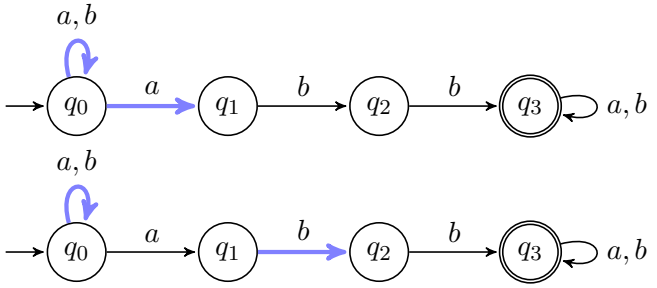
Na základě přechodové funkce vytvoříme seznam stavů NFA M, které patří do $\delta(q_0, a)$.



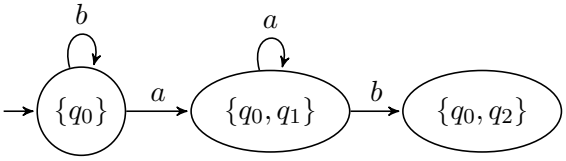
Automat M se může po načtení symbolu a nacházet ve stavu q_0 nebo q_1 a po načtení symbolu b ve stavu q_0. Do množiny N' přidáme stav $\{q_0, q_1\}$, do množiny $Done$ přidáme stav $\{q_0\}$. Změna přechodového zobrazení je znázorněna v následujícím diagramu.



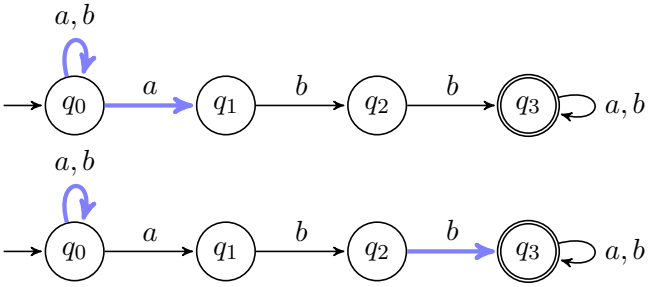
Pro nový stav $\{q_0, q_1\}$ sestavíme opět seznam stavů nedeterministického automatu M pro symbol a i b, ale tentokrát budeme sledovat stavy dva, q_0 a q_1. První obrázek má zesílené šipky pro symbol a a druhý obrázek pro symbol b.



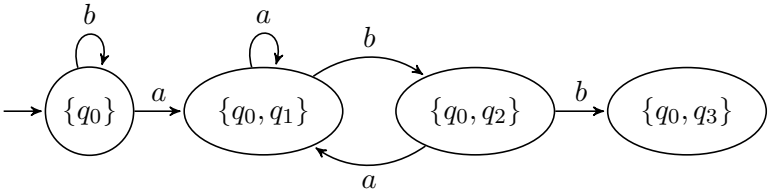
Ze stavu q_1 neexistuje přechod pro symbol a. Tuto skutečnost nemusíme v seznamu nijak zapisovat. Výpočet NFA M, který by obsahoval konfiguraci $(q_1, ax), x \in \Sigma^*$, skončí neúspěšně. Po tomto kroku vypadá DFA M' následovně:



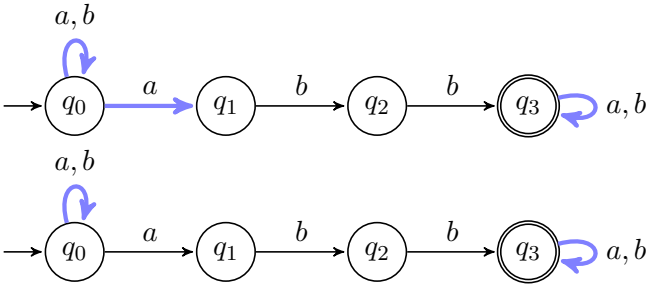
Do množiny N' přidáme stav $\{q_0, q_2\}$ a do množiny $Done$ stav $\{g_0, q_1\}$. Nyní budeme předpokládat, že je DFA M' ve stavu $\{q_0, q_2\}$, a budeme sledovat, do kterých stavů by se dostal NFA M ze stavů q_0 a q_2 načtením symbolů a nebo b.



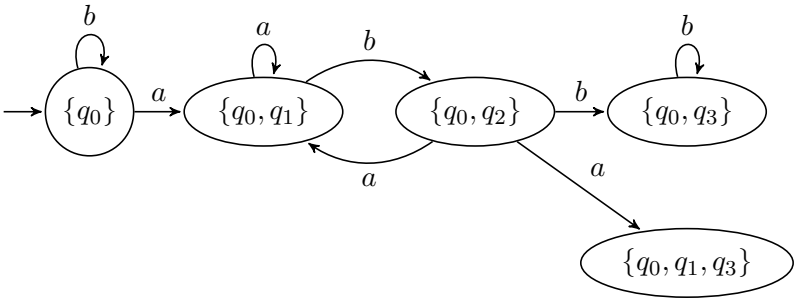
Deterministický konečný automat M' vypadá po dalším kroku konstrukce následovně:



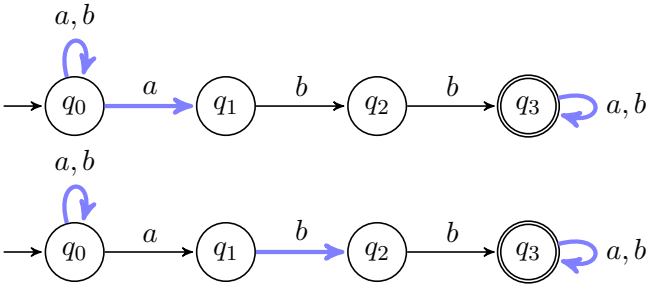
Do množiny N' přidáme stav $\{q_0, q_3\}$ a do množiny $Done$ stav $\{g_0, q_2\}$. Nyní budeme předpokládat, že je DFA M' ve stavu $\{q_0, q_3\}$, a budeme sledovat, do kterých stavů by se dostal NFA M ze stavů q_0 a q_3 načtením symbolů a nebo b.



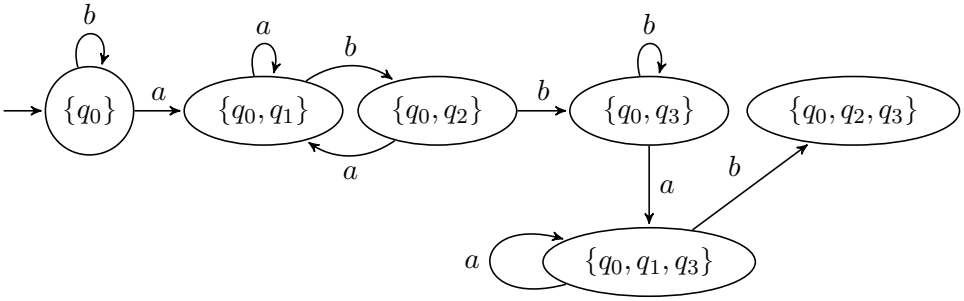
Deterministický konečný automat M' vypadá po dalším kroku konstrukce následovně:



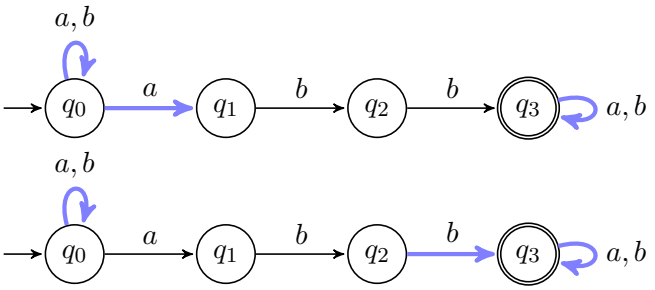
Do množiny N' přidáme stav $\{q_0, q_1, q_3\}$ a do množiny $Done$ stav $\{g_0, q_3\}$. Nyní budeme předpokládat, že je DFA M' ve stavu $\{q_0, q_1, q_3\}$ a budeme sledovat, do kterých stavů by se dostal NFA M ze stavů q_0, q_1 a q_3 načtením symbolů a nebo b.



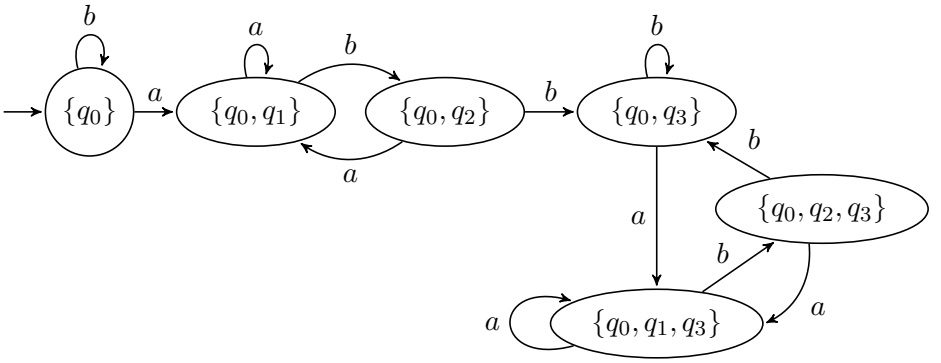
Deterministický konečný automat M' vypadá po dalším kroku konstrukce takto:



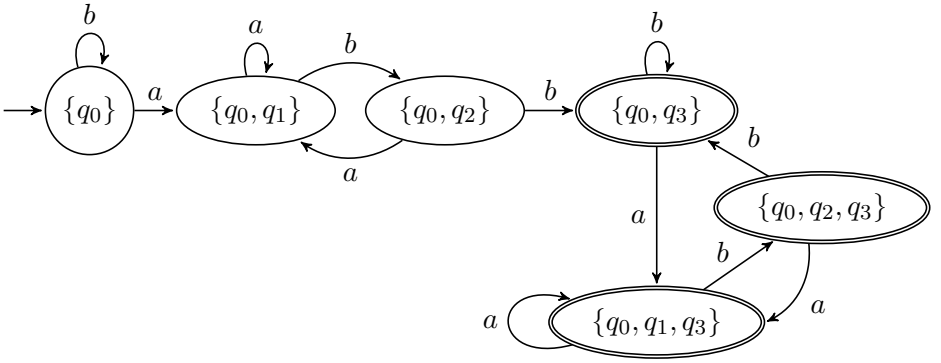
Do množiny N' přidáme stav $\{q_0, q_2, q_3\}$ a do množiny $Done$ přidáme stav $\{q_0, q_1, q_3\}$. Budeme předpokládat, že je DFA M' ve stavu $\{q_0, q_2, q_3\}$, a budeme sledovat, do kterých stavů by se dostal NFA M ze stavů q_0, q_2 a q_3 načtením symbolů a nebo b.



Deterministický konečný automat M' vypadá po dalším kroku konstrukce následovně:



Do množiny $Done$ přidáme stav $\{q_0, q_2, q_3\}$. Protože $N' = Done$, konstrukce přechodové funkce deterministického konečného automatu M' končí. Nyní už zbývá jen určit, které stavy budou koncové. Budou to všechny ty množiny stavů původního NFA, které obsahují alespoň jeden koncový stav NFA M. Do množiny koncových stavů DFA M' tedy patří stavy $\{q_0, q_3\}$, $\{q_0, q_1, q_3\}$ a $\{q_0, q_2, q_3\}$. Kompletní diagram přechodové funkce vypadá takto:



4.3 Konečné automaty a regulární jazyky

V této části budeme studovat vztah regulárních jazyků a jazyků akceptovaných konečnými automaty.

Věta 4.3 V

Ke každé regulární gramatice G existuje nedeterministický konečný automat M takový, že $L(G) = L(M)$.

Idea důkazu: Nechť $G = (N, T, P, S)$ je libovolná regulární gramatika. Sestrojíme nyní NFA $M = (Q, \Sigma, \delta, q_0, F)$, který přijímá jazyk $L(G)$.

Základní idea konstrukce spočívá ve ztotožnění neterminálů gramatiky G se stavy automatu M. Vygenerování terminálního symbolu v jednom kroku gramatiky G odpovídá přečtení téhož (vstupního) symbolu v jednom výpočetním kroku automatu M. Pokud odvozujeme slovo v regulární gramatice, přecházíme od jedné větné formy ke druhé až nakonec vygenerujeme slovo. Charakteristickým rysem větných forem odvozených v regulárních gramatikách ze startovacího neterminálu je to, že všechny obsahují nejvýše jeden neterminál (pokud se nejedná o slovo pak je to právě jeden neterminál). Je zřejmé, že když gramatika G končí odvození slova, není v odvozeném řetězci žádný neterminál. Této situaci však musí u automatu M odpovídat nově zavedený koncový stav.

Položíme

- $Q = N \cup q_f$, kde $q_f \notin N$ (tzn. q_f je nově zavedený stav),
- $q_0 = S$,
- $F = \begin{cases} \{q_f, S\}, \text{ právě když } S \to \varepsilon \in P \\ \{q_f\} \text{ jinak.} \end{cases}$
- Dále pro všechna $A, B \in Q$, $a \in \Sigma$
  1. pro každé pravidlo $A \to aB \in P$ položíme $B \in \delta(A, a)$;
  2. v případě $A \to a \in P$ položíme $q_f \in \delta(A, a)$.

Pro dokončení důkazu by bylo potřeba ještě ukázat, že $L(G) = L(M)$. □

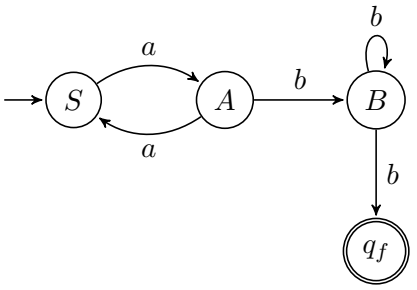
Příklad 4.4 P

Mějme regulární gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$ s množinou pravidel P definovanou takto:

$$\begin{aligned} S &\to aA, \\ A &\to aS \mid bB, \\ B &\to bB \mid b \end{aligned}$$

Vytvoříme konečný automat $M = (Q, \Sigma, \delta, q_0, F)$, který přijímá stejný jazyk, jaký gramatika G generuje.

- $Q = N \cup \{q_f\}$;
- $\Sigma = T$;
- $q_0 = S$;
- $F = \{q_f\}$;
- $\begin{aligned} S \to aA &\Rightarrow A \in \delta(S, a), \\ A \to aS &\Rightarrow S \in \delta(A, a), \\ A \to bB &\Rightarrow B \in \delta(A, b), \\ B \to bB &\Rightarrow B \in \delta(B, b), \\ B \to b &\Rightarrow q_f \in \delta(B, b) \end{aligned}$



Nyní uvedeme tvrzení opačné a opět nastíníme myšlenku důkazu.

Věta 4.4 V

Ke každému konečnému automatu M existuje regulární gramatika G taková, že $L(M) = L(G)$.

Idea důkazu: Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je libovolný konečný automat akceptující jazyk $L(M)$. Sestrojíme nyní regulární gramatiku $G = (N, T, P, S)$ takovou, že $L(G) = L(M)$.

Základní idea konstrukce gramatiky G bude spočívat ve ztotožnění neterminálů gramatiky G se stavy automatu M. Přečtení vstupního symbolu v jednom kroku výpočtu automatu M bude v G odpovídat vygenerování terminálního symbolu v jednom kroku odvození. Přechod do koncového

stavu bude zachycen pomocí ukončovacího pravidla (neobsahuje na pravé straně neterminál). Pokud je v původním automatu počáteční stav i stavem koncovým je pro gramatiku zapotřebí provést posloupnost kroků, které jsou uvedeny v důkazu věty 3.1.

Sestavíme gramatiku $G = (N, T, P, S)$ takovou, že:

- $N = Q$;
- $T = \Sigma$;
- $S = q_0$;
- dále pro všechny prvky $q' \in \delta(q, a)$:
  1. pro všechny prvky $q' \in \delta(q, a)$ vytvoříme pravidlo $q \to aq'$;
  2. v případě $q' \in F$ přidáme **další** pravidlo $q \to a$;
  3. v případě, že $q_0 \in F$ postupujeme podle důkazu věty 3.1 - přidáme pravidlo $S' \to \varepsilon$ a taková pravidla, která mají na levé straně původní startovací symbol, který změníme na S'. Do množiny neterminálů přidáme nový neterminál S', který prohlásíme za nový startovací neterminál. Tedy $S = S'$.

Pro dokončení důkazu by bylo potřeba ještě ukázat, že $L(M) = L(G)$. □

Příklad 4.5 **P**

Mějme konečný automat $M = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_0\})$ s přechodovou funkcí definovanou následovně:

$$\begin{aligned}
\delta(q_0, a) &= \{q_0\},\\
\delta(q_0, b) &= \{q_1\},\\
\delta(q_1, a) &= \{q_0\},\\
\delta(q_1, b) &= \{q_1\}.
\end{aligned}$$

Sestrojíme regulární gramatiku $G = (N, T, P, S)$ takovou, že $L(G) = L(M)$:

- $N = \{q_0, q_1\}$;
- $T = \{a, b\}$;
- $P = \{$
$$\begin{aligned}
&\delta(q_0, a) = \{q_0\} \Rightarrow q_0 \to a\, q_0,\\
&\delta(q_0, b) = \{q_1\} \Rightarrow q_0 \to b\, q_1,\\
&\delta(q_1, a) = \{q_0\} \Rightarrow q_1 \to a\, q_0,\\
&\delta(q_1, b) = \{q_1\} \Rightarrow q_1 \to b\, q_1,\\
&\\
&\delta(q_0, a) = \{q_0\} \Rightarrow q_0 \to a,\\
&\delta(q_1, a) = \{q_0\} \Rightarrow q_1 \to a\}
\end{aligned}$$

Protože q_0 je stavem počátečním i koncovým, musíme přidat pravidlo, které vygeneruje prázdné slovo. Upravíme tedy množinu pravidel následujícím způsobem:

- $P = \{\ q_0 \to a\ q_0,\ q_0 \to b\ q_1,\ q_1 \to a\ q_0,\ q_1 \to b\ q_1,\ q_0' \to a,\ q_1 \to a,\ S' \to \varepsilon,\ S' \to a\ q_0,\ S' \to b\ q_1,\ S' \to a\ \}$
- upravíme množinu neterminálů – $N = N \cup \{S'\}$,
- i startovací symbol – $S = S'$.

Jako důsledek uvedených vět získáme následující:

Věta 4.5 V

Třída regulárních jazyků je ekvivalentní s třídou jazyků akceptovaných konečnými automaty.

4.4 Lemma o vkládání pro regulární jazyky

Představme si, že jsme postaveni před úkol zjistit, jestli je zadaný jazyk regulární. Nejspíš se pokusíme sestrojit konečný automat, který jej bude akceptovat, příp. navrhnout regulární gramatiku, která bude tento jazyk generovat. Pokud se nám ani při nejlepší snaze nevede automat či gramatiku navrhnout, je možné, že takový automat resp. gramatika neexistuje a zadaný

jazyk regulární není. Nyní ale potřebujeme nějaký nástroj, pomocí kterého daný fakt dokážeme. Právě takovým nástrojem je lemma o vkládání (někdy nazývané pumping lemma) pro regulární jazyky vyjadřující nutnou (nikoli postačující) podmínku pro regularitu jazyka.

Věta 4.6 V

Lemma o vkládání. Nechť L je regulární jazyk. Pak existuje $n \in N$ takové, že libovolné slovo $w \in L$, jehož délka je alespoň n, lze psát ve tvaru $w = xyz$, kde

1. $|xy| \leq n$,
2. $y \neq \varepsilon$ a
3. $xy^i z \in L$ pro každé $i \in N_0$.

Jinými slovy: Každé dostatečně dlouhé slovo regulárního jazyka lze rozdělit na tři části. První podmínka omezuje délku prvních dvou částí, je podstatná pro delší slova. Podle druhé podmínky nesmí být prostřední část slova rovna prázdnému slovu. Třetí podmínka dala název tomuto lemmatu. Vztahuje se totiž k pumpování - jedná se o to, že pokud prostřední část slova zopakujeme (i vícekrát), dostaneme nové slovo, které opět bude patřit do jazyka. Protože číslo i může mít i hodnotu nula, bude do jazyka L patřit i slovo, které vznikne vypuštěním prostřední části.

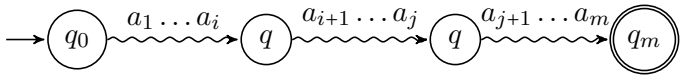
Protože se lemma poskytuje podmínku nutnou, využívá se ho právě k důkazům toho faktu, že zadaný jazyk regulární není. Dříve než ukážeme na příkladě, jak dokazujeme neregularitu jazyka, uvedeme ideu důkazu lemmatu o vkládání.

Důkaz. Jelikož L je regulární, existuje deterministický KA $M = (Q, \Sigma, q_0, F)$ rozpoznávající jazyk L. Položme $n = |Q|$. Pro libovolné slovo $w \in L$ délky alespoň n (tj. $w = a_1 \dots a_m, m \geq n$) platí, že automat M projde při akceptování slova w $m + 1$ stavy. Konečný automat M provede následující výpočet:

$$(q_0, a_1 \dots a_m) \vdash (q_1, a_2 \dots a_m) \vdash \cdots \vdash (q_m, \varepsilon), \text{kde } q_m \in F.$$

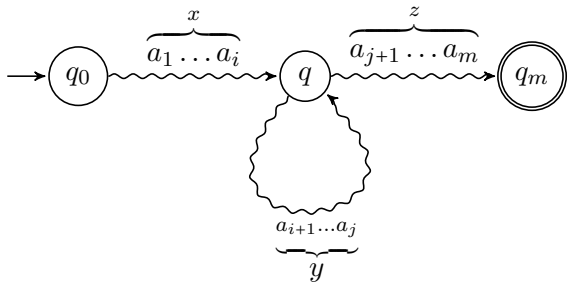
Protože automat projde $m + 1$ konfiguracemi, kterých je více než n, musí projít jedním ze stavů alespoň dvakrát (podle Dirichletova principu). Existují

tedy dva indexy i a j, takové, že $0 \leq i < j \leq m$ a $q_i = q_j = q$. Graficky můžeme znázornit výpočet konečného automatu způsobem uvedeným na obrázku 4.6.



Obrázek 4.6: Zobrazení výpočtu KA

Protože jsou v této linii dva stejné stavy, můžeme výpočet zobrazit i způsobem jako na obrázku 4.4.



Obrázek 4.7: Zobrazení výpočtu KA se smyčkou

Zpracovávané slovo je tedy rozděleno na tři části $w = xyz$ tak, že $y \neq \varepsilon$ (mezi dvěma průchody tím samým stavem automat načetl alespoň jeden znak ze vstupní pásky). K tomu, aby konečný automat prošel jedním stavem dvakrát potřebuje načíst nejvýše n znaků z pásky – připomeňme, že po načtení n znaků je automat ve své $n + 1$ konfiguraci. Z výše uvedeného znázornění je vidět, že konečný automat při zpracování slova xz vynechá a při zpracování slova $xy^i z,\ i \geq 1$ naopak i-krát zopakuje průchod smyčkou a nakonec skončí v koncovém stavu – tedy taková slova přijme. □

Jak jsme již zmínili lemma o vkládání obsahuje nutnou podmínku pro to, aby jazyk byl regulární. Proto se používá spíše k tomu, abychom dokázali, že „jestliže nelze dostatečně dlouhé slovo rozdělit na tři části tak, aby byla

splněny tři podmínky v lemmatu, pak jazyk není regulární.“ Ilustrujeme použití lemmatu na příkladě.

Příklad 4.6 **P**

Dokažte, že jazyk $L = \{a^{i+3}b^i | i \geq 0\}$ není regulární.

Budeme postupovat následujícím způsobem: Nalezneme dostatečně dlouhé slovo (delší než $n \in N_0$ - konstanta pro daný jazyk). Vyzkoušíme všechny možné způsoby jeho rozdělení a pro každé z nich nalezneme takové k, pro které $xy^kz \notin L$. Není nutné zkoušet exaktně všechna rozdělení slova, ale pouze typy rozdělení.

Nechť takovým dostatečně dlouhým slovem je $a^{j+3}b^j$ pro nějaké dostatečně velké $j \in N_0$.

Při rozdělení slova $a^{p+3}b^p$ na tři části x, y, z nás bude zajímat hlavně struktura slova y.

1. Nechť je $y = a^p, p \leq j, 0 < p \leq n$. Pak rozdělené slovo lze napsat jako $a^r a^p a^s b^j$, kde $r + p + s = j + 3$. Platí tedy, že $x = a^r$, $y = a^p$ a $z = a^s b^j$. Pokud zvolíme ve slově xy^kz $k = 0$, pak dostáváme slovo $xz = a^r a^s b^j$, kde platí $r + s < j + 3$ a tudíž $xz \notin L$.
2. Obdobná situace nastává v okamžiku, kdy je y složeno pouze ze symbolů b. Nechť je $y = b^p, p \leq j, 0 < p \leq n$. Pak rozdělené slovo lze napsat jako $a^{j+3}b^r b^p b^s$, kde $r + p + s = j$. Platí tedy, že $x = a^{j+3}b^r$, $y = b^p$ a $z = b^s$. Pokud zvolíme ve slově xy^kz $k = 0$, pak dostáváme slovo $xz = a^{j+3}a^r b^s$, kde platí $r + s < j$ a tudíž $xz \notin L$.
3. Poslední možností je vytvořit prostřední část slova jak ze symbolů a tak ze symbolů b. Nechť je $y = a^p b^q$, $x = a^{j+3-p}$ a $z = b^{j-q}$. Je zřejmé, že opakováním podslova y (to znamená, pokud zvolíme k větší než jedna) dostáváme slovo $xy^kz = a^{j+3-p}(a^p b^q)^k b^{j-q}$, které strukturou neodpovídá slovům z jazyka L. U tohoto rozdělení slova narážíme ještě na jeden problém. Pokud prozkoumáme první podmínku lemmatu o vkládání uvědomíme si, že střed „dlouhých“ slov se mění z jejich délkou, čím delší je slovo, tím vzdálenější je místo, kde končí úsek složený ze samých a-ček a začíná úsek složený ze symbolů b. Toto ovšem odporuje podmínce, která tvrdí, že délka slova xy je omezena konstantou, která je dána pro celý jazyk.

Jiné rozdělení není možné, proto není jazyk L regulární.

4.5 Vlastnosti regulárních jazyků

V této části se budeme zaobírat uzavřeností třídy regulárních jazyků na různé operace nad jazyky. Také ukážeme, že existuje podmínka, podle níž poznáme, že zadaný jazyk nemůže být regulární.

Třída jazyků je uzavřena vzhledem k dané operaci, pokud při použití této operace na jazyky z dané třídy získáme vždy opět jazyk z této třídy.

Důležitost uzávěrových vlastností spočívá nejen ve zkoumání vlastností dané třídy jazyků, ale má i řadu praktických aspektů. Daný jazyk lze často vyjádřit pomocí několika jazyků jednodušších, pro každý z nich navrhnout konečný automat a na jejich základě sestrojit žádaný výsledný automat.

Věta 4.7

V

Třída regulárních jazyků je uzavřená na doplněk.

Idea důkazu: Ke každému regulárnímu jazyku L existuje úplný deterministický konečný automat M, pro který platí $L = L(M)$. Sestrojíme nyní konečný automat, který přijímá doplněk jazyka L. Protože M je úplný a deterministický, pro každý vstupní řetězec existuje výpočet, který přečte vstupní slovo až do konce a výpočet skončí v koncovém stavu, pokud slovo náleží do jazyka L, a v jiném stavu, pokud slovo do jazyka L nepatří. Pokud se zamyslíme nad předchozí větou, docházíme k závěru, že pokud máme sestrojit konečný automat, který akceptuje doplněk jazyka L, stačí z koncových stavů vytvořit stavy nekoncové a stavy, které nejsou koncové, přiřadíme do množiny koncových stavů nového FA.

Poznámka: Konečný automat, který rozpoznává jazyk L musí být deterministický. A to proto, aby pro každé slovo existoval pouze jeden výpočet, buď úspěšný nebo neúspěšný. Pokud by automat byl nedeterministický, mohl by pro slovo, které do jazyka patří, existovat úspěšný i neúspěšný výpočet. Po změně stavů z koncových na nekoncové a obráceně, by takové slovo bylo automatem opět akceptováno.

Věta 4.8

V

Třída regulárních jazyků je uzavřená na sjednocení.

Idea důkazu: Mějme dva regulární jazyky L_1 a L_2, ke kterým existují úplné deterministické konečné automaty M_1 a M_2, které je akceptují (platí tedy $L_1 = L(M_1)$ a $L_2 = L(M_2)$). Sestrojíme konečný automat, který akceptuje jazyk $L_1 \cup L_2$. Postup konstrukce konečného automatu, který akceptuje sjednocení dvou daných jazyků, je podobný postupu hledání deterministického konečného automatu ke konečnému automatu nedeterministickému. Představme si, že máme k dispozici oba konečné automaty M_1 i M_2, a oba tyto automaty dostanou na společnou vstupní pásku slovo, o kterém mají rozhodnout, jestli patří do jazyka jimi akceptovaného. Oba automaty budou synchronně načítat symbol po symbolu a měnit stavy, ve kterých se nachází. Pokud na konci výpočtu bude alespoň jeden automat v koncovém stavu, pak dané slovo náleží do jazyka $L_1 \cup L_2$.

Jak tedy takový automat M_3 akceptující sjednocení dvou jazyků sestrojit? Nechť $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ a $M_2 = (Q_2, \Sigma_2, \delta_2, p_0, F_2)$ jsou konečné automaty akceptující L_1 a L_2. Zkonstruujeme $M_3 = (Q_3, \Sigma_3, \delta_3, \boxed{q_0, p_0}, F_3)$. Každý stav automatu M_3 je tvořen dvojicí stavů – první stav automatu M_1, druhý stav automatu M_2. Vstupní abeceda Σ_3 je tvořena sjednocením abeced Σ_1 a Σ_2. Množina koncových stavů je tvořena takovými dvojicemi stavů, kde alespoň jeden z nich je koncový v původním automatu.

Konstrukce přechodové funkce probíhá následujícím způsobem. Nejdříve zakreslíme počáteční stav. Pro všechny vstupní symboly zapíšeme dvojice stavů, do kterých se původní automaty dostaly. Protože jsme na počátku předpokládali, že oba konečné automaty jsou deterministické a úplné, jedná se vždy o dvojici stavů, která tvoří stav nový. V případě neúplnosti automatu by se mohlo stát, že daný vstup umí zpracovat pouze jeden automat – nový stav by byl tvořen pouze jediným stavem automatu původního, nebo dokonce žádný stav – nový automat by neměl definovanou přechodovou funkci pro tuto kombinaci stavu a vstupního symbolu. Pokud bychom nepožadovali deterministický konečný automat, bylo by možné, aby nový stav tvořilo více stavů původních automatů než pouze dvojice. Pokračujeme v přidávání nových stavů a konstruování přechodů mezi nimi, dokud nemáme

vyřešeny přechody pro všechny nové stavy a všechny vstupní symboly z Σ_3. To, že konstrukce přechodové funkce automatu M_3 je proces konečný, je dáno konečným počtem různých dvojic stavů původních automatů, který je roven $|\Sigma_1| \cdot |\Sigma_2|$. □

Algoritmus 4.2 FA akceptující sjednocení regulárních jazyků

A

Vstup: Úplné deterministické konečné automaty
$M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ a $M_2 = (Q_2, \Sigma_2, \delta_2, p_0, F_2)$
Výstup: Úplný deterministický konečný automat
$M_3 = (Q_3, \Sigma_3, \delta_3, \boxed{q_0, p_0}, F_3)$

$Q_3 := \{\boxed{q_0, p_0}\}$; $\delta_3 := \varnothing$; $F_3 := \varnothing$; $Done := \varnothing$; $\Sigma_3 := \Sigma_1 \cup \Sigma_2$;
dokud $(Q' - Done) \neq \varnothing$ dělej
  $M :=$ prvek množiny $Q' - Done$;
  jestliže $M = \boxed{q, p}$, kde $q \in F_1 \vee p \in F_2$, pak
    $F_3 := F_3 \cup \{M\}$;
  konec jestliže
  pro všechna $a \in \Sigma$ dělej
    $N := \boxed{\delta(q, a), \delta(p, a)}$;
    $Q_3 := Q_3 \cup \{N\}$;
    $\delta_3 := \delta_3 \cup \{((M, a), N)\}$;
  konec pro
  $Done := Done \cup \{M\}$;
konec dokud
$M_3 := (Q_3, \Sigma_3, \delta_3, \boxed{q_0, p_0}, F_3)$;

Nyní uvedeme příklad konstrukce konečného automatu pro sjednocení jazyků.

Příklad 4.7

P

Mějme dva úplné konečné automaty M_1 a M_2. Tyto automaty akceptují jazyky:

$$L_1 = \{abaw \mid w \in \{a, b\}^*\} \text{ a } L_2 = \{w \in \{a, b\}^* \mid |w|_a \bmod 3 = 0\}.$$

Sestavíme konečný automat, který akceptuje sjednocení těchto dvou jazyků, tedy $L_3 = L_1 \cup L_2 = \{w \in \{a,b\}^* \mid w = abax,\ x \in \{a,b\}^* \vee |w|_a \bmod 3 = 0\}$.

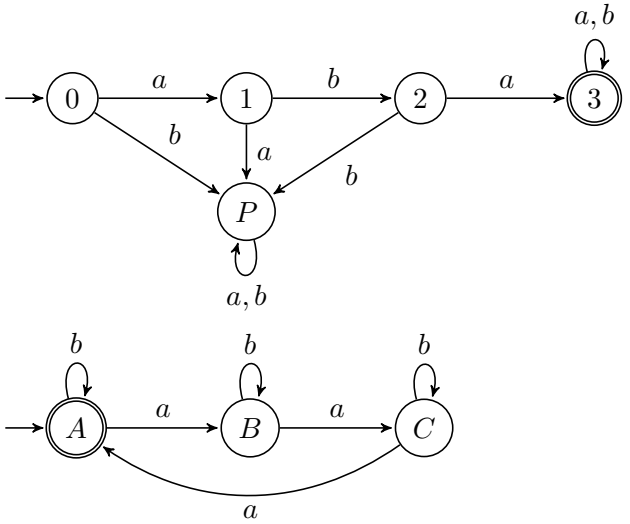
$M_1 = (\{0,1,2,3,P\}, \{a,b\}, \delta_1, 0, \{3\})$, kde:

δ_1 : $\delta_1(0,a) = 1$, $\delta_1(0,b) = P$,
$\delta_1(1,a) = P$, $\delta_1(1,b) = 2$,
$\delta_1(2,a) = 3$, $\delta_1(2,b) = P$,
$\delta_1(3,a) = 3$, $\delta_1(3,b) = 3$,
$\delta_1(4,a) = P$, $\delta_1(4,b) = P$

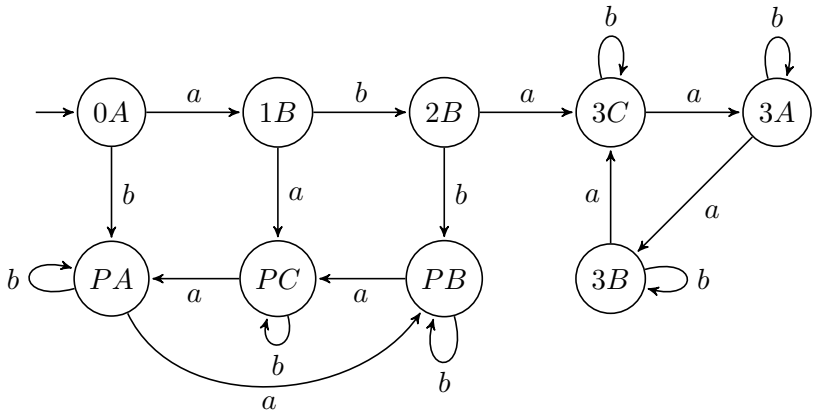
$M_2 = (\{A,B,C\}, \{a,b\}, \delta_2, A, \{A\})$, kde:

δ_2 : $\delta_2(A,a) = B$, $\delta_2(A,b) = A$,
$\delta_2(B,a) = C$, $\delta_2(B,b) = B$,
$\delta_2(C,a) = A$, $\delta_2(C,b) = C$

Znázorníme automaty graficky.



Postupně konstruujeme přechodový diagram na základě přechodových funkcí jednotlivých konečných automatů. Postupně pro dvojici stavů a všechny symboly ze vstupní abecedy přidáváme nové přechody do stávajících nebo nových dvojic stavů.



Nyní stanovíme, které stavy nového automatu budou koncové. Protože do sjednocení dvou množin patří právě ta slova, která jsou v jedné nebo ve druhé množině, bude postačující, pokud stav automatu M_3 bude obsahovat označení alespoň jednoho koncového stavu původních konečných automatů. Výsledkem je tedy konečný automat:

$M_3 = (\{0A, 1B, 2B, 3A, 3B, 3C, PA, PB, PC\}, \{a, b\}, \delta_3, 0A, \{3A\})$, kde:

$$
\begin{aligned}
\delta_3: \quad & \delta_3(0A, a) = 1B, && \delta_3(0A, b) = PA,\\
& \delta_3(1B, a) = PC, && \delta_3(1B, b) = 2B,\\
& \delta_3(2B, a) = 3C, && \delta_3(2B, b) = PB,\\
& \delta_3(3A, a) = 3B, && \delta_3(3A, b) = 3A,\\
& \delta_3(3B, a) = 3C, && \delta_3(3B, b) = 3B,\\
& \delta_3(3C, a) = 3A, && \delta_3(3C, b) = 3C,\\
& \delta_3(PA, a) = PB, && \delta_3(PA, b) = PA,\\
& \delta_3(PB, a) = PC, && \delta_3(PB, b) = PB,\\
& \delta_3(PC, a) = PA, && \delta_3(PC, b) = PC.
\end{aligned}
$$

Věta 4.9 V

Třída regulárních jazyků je uzavřená na průnik.

Idea důkazu: Přístup k tvorbě konečného automatu pro průnik regulárních jazyků je obdobný jako pro sjednocení regulárních jazyků. Jediná změna

nastává v konstrukci množiny koncových stavů. Do množiny koncových stavů konečného automatu akceptujícího průnik dvou regulárních jazyků patří takové stavy, které se skládají ze dvou koncových stavů. Jinak řečeno nový konečný automat je v koncovém stavu právě tehdy, když jsou v koncovém stavu oba dílčí konečné automaty.

Algoritmus pro konstrukci FA akceptujícího průnik dvou regulárních jazyků je následující:

Algoritmus 4.3 FA akceptující průnik regulárních jazyků

A

Vstup: Úplné deterministické konečné automaty
$M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$a $M_2 = (Q_2, \Sigma_2, \delta_2, p_0, F_2)$
Výstup: Úplný deterministický konečný automat
$M_4 = (Q_4, \Sigma_4, \delta_4, \boxed{q_0, p_0}, F_4)$

$Q_4 := \{\boxed{q_0, p_0}\}$; $\delta_4 := \varnothing$; $F_4 := \varnothing$; $Done := \varnothing$; $\Sigma_4 := \Sigma_1 \cup \Sigma_2$;
dokud $(Q' - Done) \neq \varnothing$ dělej
  $M :=$ prvek množiny $Q' - Done$;
  jestliže $M = \boxed{q, p}$, kde $q \in F_1 \wedge p \in F_2$ pak
    $F_4 := F_4 \cup \{M\}$;
  konec jestliže
  pro všechna $a \in \Sigma$ dělej
    $N := \boxed{\delta(q, a), \delta(p, a)}$;
    $Q_4 := Q_4 \cup \{N\}$;
    $\delta_4 := \delta_4 \cup \{((M, a), N)\}$;
  konec pro
  $Done := Done \cup \{M\}$;
konec dokud
$M_4 := (Q_4, \Sigma_4, \delta_4, \boxed{q_0, p_0}, F_4)$;

Protože postup konstrukce konečného automatu akceptujícího průnik regulárních jazyků je obdobný tomu pro konstrukci konečného automatu akceptujícímu sjednocení, uvedeme zde příklad, který navazuje na příklad uvedený k větě 4.8.

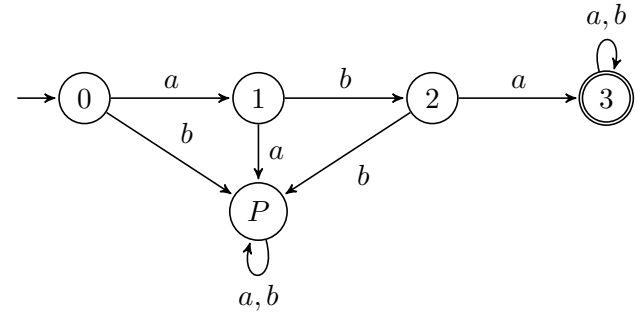
Příklad 4.8 P

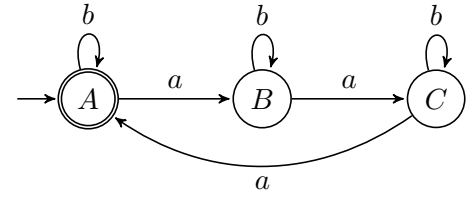
Mějme dva úplné konečné automaty M_1 a M_2 z příkladu 4.7. Tyto automaty akceptují jazyky:

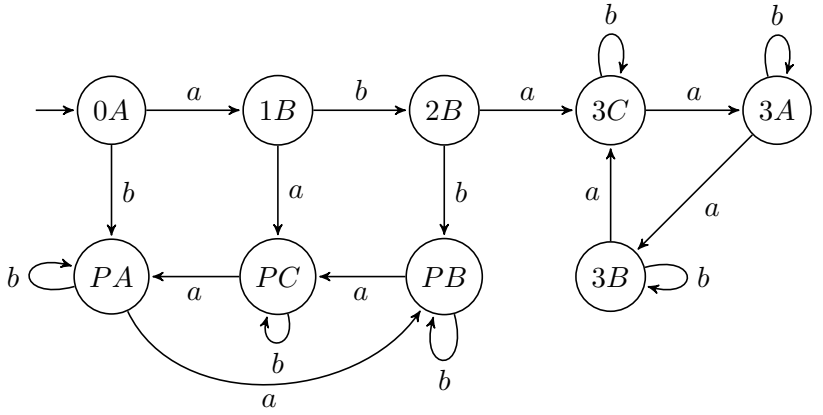
$$L_1 = \{abaw \mid w \in \{a,b\}^*\} \text{ a } L_2 = \{w \in \{a,b\}^* \mid |w|_a \bmod 3 = 0\}.$$

Sestavíme konečný automat, který akceptuje průnik těchto dvou jazyků, tedy $L_4 = L_1 \cup L_2 = \{w \in \{a,b\}^* \mid w = abax,\ x \in \{a,b\}^* \wedge |w|_a \bmod 3 = 0\}$.

Připomeneme grafické znázornění obou automatů a výsledný automat bez označených koncových stavů.



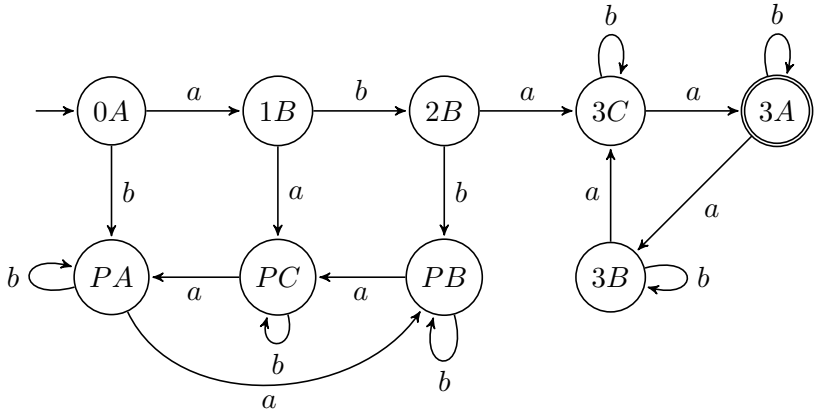




Nyní stanovíme, které stavy nového automatu budou koncové. Protože do průniku dvou množin patří právě ta slova, která jsou v současně v bou množinách, bude nutné, aby stav automatu M_4 bude obsahovat označení koncových stavů obou původních konečných automatů. Koncovým stavem je tedy stav $3A$. Výsledkem je tedy konečný automat:

$M_4 = (\{0A, 1B, 2B, 3A, 3B, 3C, PA, PB, PC\}, \{a, b\}, \delta_4, 0A, \{3A\})$, kde:

δ_4 : $\delta_4(0A, a) = 1B$, $\delta_4(0A, b) = PA$,
$\delta_4(1B, a) = PC$, $\delta_4(1B, b) = 2B$,
$\delta_4(2B, a) = 3C$, $\delta_4(2B, b) = PB$,
$\delta_4(3A, a) = 3B$, $\delta_4(3A, b) = 3A$,
$\delta_4(3B, a) = 3C$, $\delta_4(3B, b) = 3B$,
$\delta_4(3C, a) = 3A$, $\delta_4(3C, b) = 3C$,
$\delta_4(PA, a) = PB$, $\delta_4(PA, b) = PA$,
$\delta_4(PB, a) = PC$, $\delta_4(PB, b) = PB$,
$\delta_4(PC, a) = PA$, $\delta_4(PC, b) = PC$



Věta 4.10

Třída regulárních jazyků je uzavřená na rozdíl.

Idea důkazu: Konstrukce konečného automatu akceptujícího rozdíl dvou regulárních jazyků $L_1 - L_2$ je obdobná, jako v předchozích dvou větách. Rozdíl

je pouze v určení koncových stavů. Do množiny koncových stavů patří takové stavy (p, q), kde p je koncový stav FA akceptujícího jazyk L_1, a q nepatří do množiny koncových stavů FA akceptujícího jazyk L_2. □

Budeme pokračovat s konečnými automaty z příkladu 4.7.

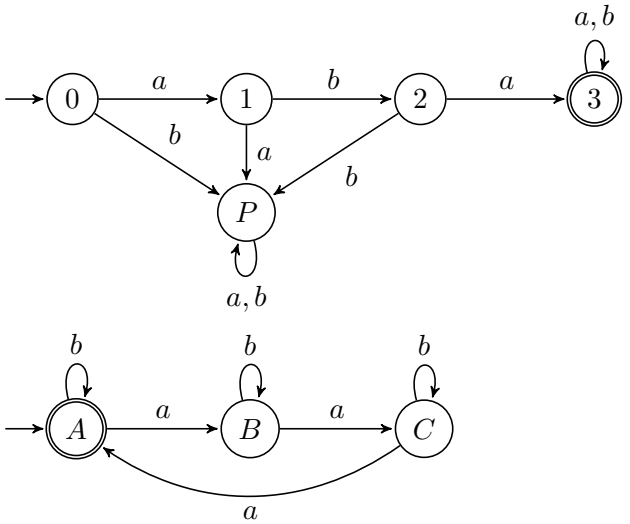
Příklad 4.9 **P**

Mějme dva úplné konečné automaty M_1 a M_2 z příkladu 4.7. Tyto automaty akceptují jazyky:

$$L_1 = \{abaw \mid w \in \{a,b\}^*\} \text{ a } L_2 = \{w \in \{a,b\}^* \mid |w|_a \bmod 3 = 0\}.$$

Sestavíme konečný automat, který akceptuje rozdíl těchto dvou jazyků, tedy $L_5 = L_1 - L_2 = \{w \in \{a,b\}^* \mid w = abax,\ x \in \{a,b\}^* \wedge |w|_a \bmod 3 \neq 0\}$.

Připomeneme grafické znázornění obou automatů.

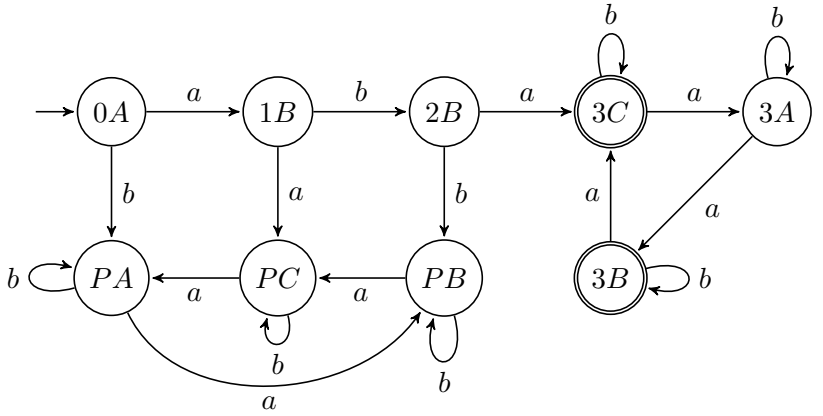


Nyní stanovíme, které stavy nového automatu budou koncové. Protože do rozdílu dvou množin patří právě ta slova, která náleží do první množiny a současně nenáleží do množiny druhé, bude nutné, aby stav automatu M_5 obsahoval označení původních koncových stavů prvního automatu a označení takových stavů druhého automatu, které nejsou koncové.

Výsledkem je tedy konečný automat:

$M_5 = (\{0A, 1B, 2B, 3A, 3B, 3C, PA, PB, PC\}, \{a, b\}, \delta_5, 0A, \{3B, 3C\})$, kde:

$$
\begin{aligned}
\delta_5: \quad & \delta_5(0A, a) = 1B, && \delta_5(0A, b) = PA,\\
& \delta_5(1B, a) = PC, && \delta_5(1B, b) = 2B,\\
& \delta_5(2B, a) = 3C, && \delta_5(2B, b) = PB,\\
& \delta_5(3A, a) = 3B, && \delta_5(3A, b) = 3A,\\
& \delta_5(3B, a) = 3C, && \delta_5(3B, b) = 3B,\\
& \delta_5(3C, a) = 3A, && \delta_5(3C, b) = 3C,\\
& \delta_5(PA, a) = PB, && \delta_5(PA, b) = PA,\\
& \delta_5(PB, a) = PC, && \delta_5(PB, b) = PB,\\
& \delta_5(PC, a) = PA, && \delta_5(PC, b) = PC
\end{aligned}
$$



Necháme na čtenáři, aby sestrojil konečný automat akceptující jazyk $L_2 - L_1$.

Věta 4.11 V

Třída regulárních jazyků je uzavřená na zřetězení.

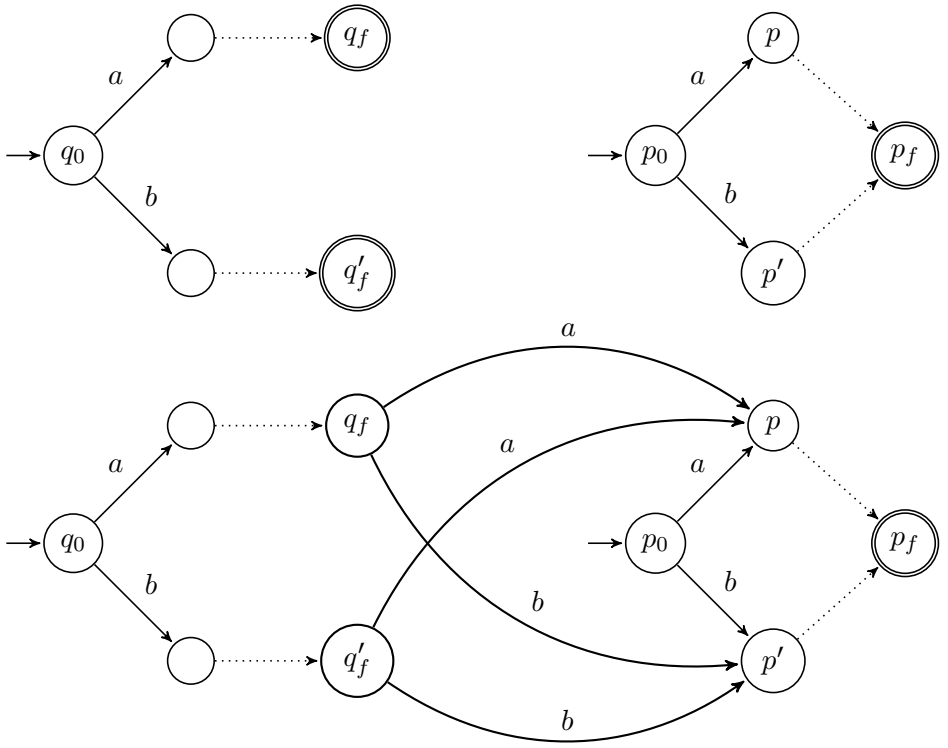
Idea důkazu: Mějme dva regulární jazyky L_1 a L_2. Ke každému z nich existuje úplný deterministický konečný automat M_1 a M_2, který jej akceptuje. Zkonstruujeme konečný automat M_3, který akceptuje jazyk $L_1 \cdot L_2$.

Nechť $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ a $M_2 = (Q_2, \Sigma_2, \delta_2, p_0, F_2)$ takové, že platí $Q_1 \cap Q_2 = \emptyset$.

Konečný automat $M_3 = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_3, q_0, F_2)$, kde

$$\delta_3 = \delta_1 \cup \delta_2 \cup \delta'$$

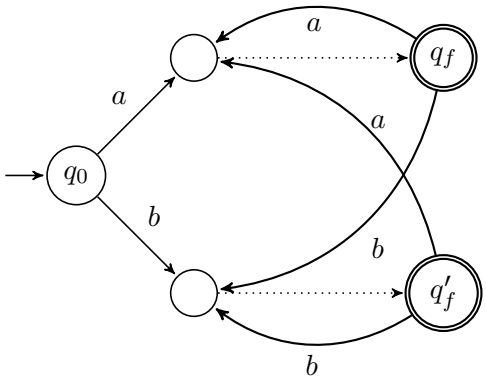
Část přechodové funkce označená δ' zodpovídá za „spojení" obou automatů a je definována takto: $\forall \delta_2(p_0, a) = p$ přidej $\delta'(q_f, a) = p, \forall q_f \in F_1$. Pro všechny přechody, které vedou z počátečního stavu druhého automatu, přidáme přechody ze všech koncových stavů tam, kam vedou ty z počátečního stavu. Graficky znázorněna by situace vypadala následovně:



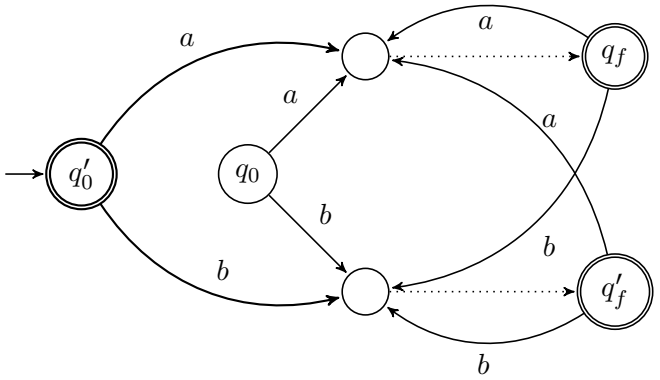
Věta 4.12 V

Třída regulárních jazyků je uzavřená na iteraci.

Idea důkazu: Nechť L je regulární jazyk a $M = (Q, \Sigma, \delta, q_0, F)$ konečný automat, který jej akceptuje. Sestrojíme konečný automat $M' = (Q', \Sigma, \delta', q_0, F)$, který bude akceptovat iteraci jazyka L, tedy L^*. První část konstrukce je obdobná konstrukci automatu přijímajícího zřetězení jazyků, jen budeme zřetězovat automat se sebou samým, a to v cyklu. Opět přidáme přechody od koncových stavů takové a do těch stavů, jaké vedou z počátečního stavu.



Tento automat pak přijímá pozitivní iteraci jazyka L. Je potřeba dořešit situaci, kdy automat, pro který platí $q_0 \notin F$, má přijmout jazyk L^0, tedy prázdné slovo. Situaci vyřešíme přidáním nového počátečního stavu q'_0, který bude zároveň patřit i do množiny koncových stavů, a přidáme také přechody z tohoto nového počátečního stavu takové, jaké má původní počáteční stav.



Sestrojili jsme tedy konečný automat $M' = (Q \cup \{q_0'\}, \Sigma, \delta', q_0', F \cup \{q_0'\})$, kde:

- $\forall \delta(q, a) = p$ přidej $\delta'(q, a) = p$,
- $\forall \delta(q_0, a) = p$ přidej $\delta'(q_f, a) = p$, $\forall q_f \in F \cup \{q_0'\}$.

□

Věta 4.13 V

Třída regulárních jazyků je uzavřená na zrcadlový obraz.

Idea důkazu: Nechť L je regulární jazyk, který je rozpoznáván nějakým konečným automatem M. Sestrojíme s ním ekvivalentní nedeterministický konečný automat M'. FA M' obdržíme tak, že:

- v přechodovém grafu M obrátíme orientaci přechodů,
- přidáme nový počáteční stav,
- z nového počátečního stavu povedeme takové přechody, které vedou ze všech koncových stavů automatu M
- počáteční stav automatu M bude koncovým stavem konečného automatu M'.

Sestrojili jsme tedy konečný automat $M' = (Q \cup \{q_0'\}, \Sigma, \delta', q_0', \{q_0\})$, kde:

- $\forall \delta(q, a) = p$ přidej q do $\delta'(p, a)$,
- $\forall \delta(p, a) = q_f$ přidej do $\delta'(q_0', a)$ stav p, $\forall q_f \in F$.

□

4.6 Redukce a minimalizace konečného automatu

Při návrhu konečného automatu různými způsoby může být výsledkem na pohled zcela odlišné automaty, které se mohou lišit nejen vlastnostmi přechodové funkce, ale i počty stavů. Za předpokladu, že všechny akceptují stejný jazyk, který z nich je lepší? Zachycuje některý z nich zcela jiný přístup nebo jsou všechny založené na stejném principu? Odpověď na tuto otázku nám může poskytnout tzv. redukce a minimalizace konečného automatu.

Při redukci odstraňujeme nedosažitelné a nadbytečné stavy. Minimalizace konečného automatu pak spočívá ve snižování počtu stavů takovým způsobem, že budeme nacházet ekvivalentní stavy.

Nyní přistoupíme k redukci konečného automatu.

Definice 4.6

Def

Dosažitelný stav

Stav q konečného automatu $M = (Q, \Sigma, \delta, q_0, F)$ se nazývá dosažitelný, jestliže existuje slovo $w \in \Sigma^*$ takové, že pro které existuje výpočet konečného automatu $(q_0, w) \vdash^* (p, \varepsilon)$.

Stavy, které nejsou dosažitelné, se nazývají nedosažitelné.

Postup, jakým odstraníme nedosažitelné stavy, je poměrně jednoduchý.

Algoritmus 4.4 Odstranění nedosažitelných stavů FA

A

Vstup: Konečný automat $M = (Q, \Sigma, \delta, q_0, F)$
Výstup: Ekvivalentní konečný automat $M' = (Q', \Sigma, \delta', q_0, F')$ bez nedosažitelných stavů

$S_0 = \{q_0\}$; i = 1;
dělej
$S_i = S_{i-1} \cup \{q \mid \delta(p, a) \ni q,\ p \in S_{i-1}, a \in \Sigma\}$
dokud platí $S_i \neq S_{i-1}$;
$Q' = S_i$;
$F' = F \cap S_i$;
pro všechna $\delta(p, a)$ pokud $p \in Q'$ polož $\delta'(p, a) = \delta(p, a) \cap S_i$;

Popíšeme jednotlivé kroky vytvoření množiny S_i.

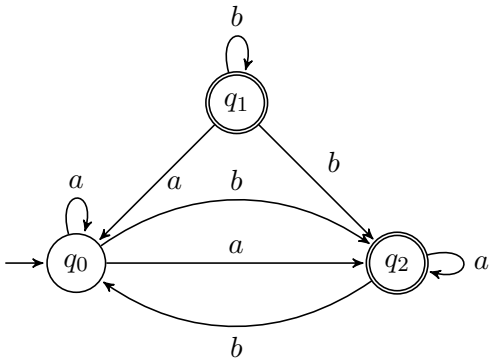
- Vytvoříme množinu S_0, do ní přiřadíme počáteční stav automatu, $S_0 = \{q_0\}$;
- vytvoříme množinu S_1 tak, že do ní dáme prvky množiny S_0 a dále všechny stavy, do kterých vede přechod ze stavů množiny S_0, tzn. přidáme všechny stavy, do kterých se dá dostat přímo z q_0;
- postupně vytváříme množiny $S_2, \ldots, S_i$ tak, že do S_i zařadíme nejdříve obsah množiny S_{i-1} a pak přidáme všechny stavy, do kterých vede přechod z některého stavu z množiny S_{i-1};
- Vytváření nových množin ukončíme tehdy, když se do množiny nedá přidat žádný nový stav, tedy $S_i = S_{i-1}$.
- Nový konečný automat získáme tak, že z původního konečného automatu odstraníme stavy (z množiny Q a F), které nenáleží do množiny S_i – jsou nedosažitelné, a dále odstraníme všechny přechody, které vedou z nebo do těchto stavů.

Příklad 4.10 P

Mějme konečný automat $M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_1, q_2\})$ s přechodovou funkcí definovanou následovně:

$$\begin{aligned}
\delta(q_0, a) &= \{q_0, q_2\} \\
\delta(q_0, b) &= \{q_2\} \\
\delta(q_1, a) &= \{q_0\} \\
\delta(q_1, b) &= \{q_1, q_2\} \\
\delta(q_2, a) &= \{q_2\} \\
\delta(q_2, b) &= \{q_0\}
\end{aligned}$$

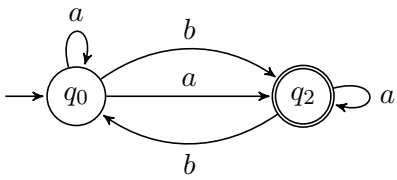
Pokud znázorníme automat M graficky, dostáváme následující schéma:



Zkonstruujeme ekvivalentní konečný automat M' bez nedosažitelných stavů.

1. $S_0 = \{q_0\}$;
2. $S_1 = \{q_0\} \cup \{q_2\}$, přidáme stav q_2, protože pokud je automat ve stavu q_0 a načítá vstupní symbol b, dostává se do stavu q_2;
3. $S_2 = \{q_0, q_2\}$, nepřidáváme žádný stav, protože ze stavu q_0 i q_2 se přečtením a nebo b dostává automat zase do stavu q_0 nebo q_2;
4. $Q' = S_2$; $F' = \{q_2\}$;
5. $\delta(q_0, a) = \{q_0, q_2\}$
   $\delta(q_0, b) = \{q_2\}$
   ~~$\delta(q_1, a) = \{q_0\}$~~
   ~~$\delta(q_1, b) = \{q_1, q_2\}$~~
   $\delta(q_2, a) = \{q_2\}$
   $\delta(q_2, b) = \{q_0\}$

Výsledný automat má následující přechodový diagram:



V dalším kroku je zapotřebí odstranit nadbytečné stavy.

Definice 4.7

Def

Nadbytečný stav

Stav q konečného automatu $M = (Q, \Sigma, \delta, q_0, F)$ se nazývá nadbytečný, jestliže neexistuje slovo $w \in \Sigma^*$ takové, pro které existuje výpočet konečného automatu $(q, w) \vdash^* (q_f, \varepsilon), q_f \in F$.

Postup, jakým odstraníme nadbytečné stavy, je uveden v následujícím algoritmu.

Algoritmus 4.5 Odstranění nadbytečných stavů FA

A

Vstup: Konečný automat $M = (Q, \Sigma, \delta, q_0, F)$
Výstup: Ekvivalentní konečný automat $M' = (Q', \Sigma, \delta', q_0, F')$ bez nadbytečných stavů

$R_0 = F$; i = 1;
dělej
$R_i = R_{i-1} \cup \{p \mid \delta(p, a) \ni q,\ q \in R_{i-1}, a \in \Sigma\}$
dokud platí $R_i \neq R_{i-1}$;
$Q' = R_i$;
$F' = F \cap R_i$;
pro všechna $\delta(p, a)$ pokud $p \in Q'$ polož $\delta'(p, a) = \delta(p, a) \cap R_i$;

Popíšeme jednotlivé kroky vytvoření množiny R_i.

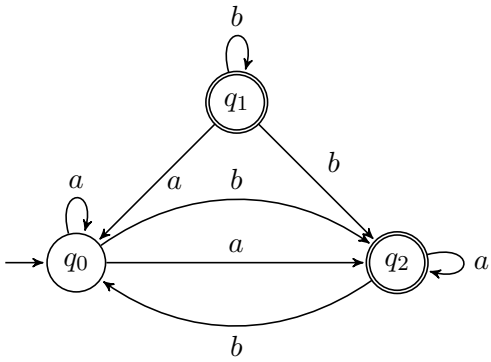
- Vytvoříme množinu R_0, do ní přiřadíme všechny konečné stavy automatu,
  $R_0 = F$;
- vytvoříme množinu R_1 tak, že do ní dáme prvky množiny R_0 a dále všechny stavy, ze kterých vede přechod do stavů množiny R_0, tzn. přidáme všechny stavy, ze kterých se dá dostat přímo do koncových stavů;
- postupně vytváříme množiny $R_2, \ldots, R_i$ tak, že do R_i zařadíme nejdříve obsah množiny R_{i-1} a pak přidáme všechny stavy, ze kterých vede přechod do některého stavu z množiny R_{i-1};
- Vytváření nových množin ukončíme tehdy, když se do množiny nedá přidat žádný nový stav, tedy $R_i = R_{i-1}$.
- Nový konečný automat získáme tak, že z původního konečného automatu odstraníme stavy (z množiny Q a F), které nenáleží do množiny R_i – jsou nadbytečné, a dále odstraníme všechny přechody, které vedou z nebo do těchto stavů.

Příklad 4.11 P

Mějme konečný automat $M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_1, q_2\})$ s přechodovou funkcí definovanou následovně:

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

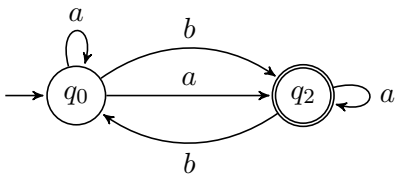
Pokud znázorníme automat M graficky, dostáváme následující schéma:



Zkonstruujeme ekvivalentní konečný automat M' bez nedosažitelných stavů.

1. $S_0 = \{q_0\}$;
2. $S_1 = \{q_0\} \cup \{q_2\}$, přidáme stav q_2, protože pokud je automat ve stavu q_0 a načítá vstupní symbol b, dostává se do stavu q_2;
3. $S_2 = \{q_0, q_2\}$, nepřidáváme žádný stav, protože ze stavu q_0 i q_2 se přečtením a nebo b dostává automat zase do stavu q_0 nebo q_2;
4. $Q' = S_2$; $F' = \{q_2\}$;
5. $\delta(q_0, a) = \{q_0, q_2\}$
   $\delta(q_0, b) = \{q_2\}$
   ~~$\delta(q_1, a) = \{q_0\}$~~
   ~~$\delta(q_1, b) = \{q_1, q_2\}$~~
   $\delta(q_2, a) = \{q_2\}$
   $\delta(q_2, b) = \{q_0\}$

Výsledný automat má následující přechodový diagram:



Nyní se pustíme do druhé části redukce konečných automatů.

Definice 4.8

Def

Minimální FA

Úplný deterministický konečný automat je minimální, jestliže neexistuje žádný jazykově ekvivalentní úplný konečný automat, který má menší počet stavů.

V každém konečném automatu je dáno základní rozlišení stavů na koncové a ostatní stavy. Dva stavy je dále možné rozlišit pomocí řetězce vstupních symbolů a to tak, že automat začíná zpracovávat řětězec v jednom nebo druhém stavu, po přečtení řetězce se pak v prvním případě nacází automat ve stavu, který patří do cílové množiny a ve druhém případě je ve stavu mimo tuto množinu. Stavy, které takto nelze rozlišit, se nazývají ekvivalentní.

Definice 4.9

Def

Ekvivalence stavů

Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat a $p, q \in Q$. Řekneme, že stavy p, q jsou ekvivalentní ($p \sim q$), jestliže pro každé $w \in \Sigma^*$ platí:

$$(p, w) \vdash (q_f, \varepsilon) \Leftrightarrow (q, w) \vdash (q'_f, \varepsilon), q_f, q'_f \in F.$$

Relace $\sim$ je relací ekvivalence na množině Q. Ukážeme nyní, jak pro libovolné dva stavy určit, zda jsou ekvivalentí.

Pro každý konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ definujeme posloupnost relací $\overset{0}{\sim}, \overset{1}{\sim}, \overset{2}{\sim} \ldots$ na množině Q takto:

- $p \overset{0}{\sim} q \Longleftrightarrow_{df} p \in F \Leftrightarrow q \in F$;

- $p \overset{i}{\sim} q \Longleftrightarrow_{df} p \overset{i-1}{\sim} q$ a pro všechna $a \in \Sigma \quad \delta(p,a) \overset{i-1}{\sim} \delta(q,a), \quad i \geq 1.$

Z předchozího vyplývá, že $p \overset{i}{\sim} q$ právě tehdy, když libovolné slovo délky nepřesahující i převede automat ze stavů p, q v obou případech do koncového nebo v obou případech do nekoncového stavu.

Věta 4.14 **V**

Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Potom platí následující tvrzení:

1. Každá z relací $\overset{i}{\sim}$ je relací ekvivalence na Q. Označme $R_i = Q/\overset{i}{\sim}$ rozklad Q podle $\overset{i}{\sim}$.
2. Pro každé $i \geq 0$ je R_{i+1} zjemněním rozkladu R_i, tzn. pokud pro $p, q \in Q$ platí $p \overset{i+1}{\sim} q$, pak platí také $p \overset{i}{\sim} q$.
3. Pokud pro nějaké $i \geq 0$ platí $R_{i+1} = R_i$, pak také $R_{i+t} = R_i$ pro libovolné $t \geq 1$.
4. Jestliže $|Q| = n$, pak existuje $k \leq n - 1$ takové, že $R_k = R_{k+1}$.
5. Pro každé k takové, že $R_k = R_{k+1}$, je relace $\overset{k}{\sim}$ totožná s relací $\sim$, tzn. pro libovolné $p, q \in Q$ je $p \overset{k}{\sim} q \Leftrightarrow p \sim q$.

Podle předchozí věty sestavíme algoritmus rozkladu množiny stavů Q na $\sim$-třídy.

Algoritmus 4.6 Rozklad množiny stavů Q na $\sim$-třídy **A**

Vstup: Konečný automat $M = (Q, \Sigma, \delta, q_0, F)$
Výstup: $Q/\sim$

$R_0 = \{\{q_f | q_f \in F\}, \{q \in Q | q \notin F\}$;
dělej

R_i : dva stavy patří do stejné množiny rozkladu $Q/\overset{i}{\sim}$, pokud náležely do stejné množiny rozkladu $Q/\overset{i-1}{\sim}$ a pro každé $a \in \Sigma$ platí $\delta(p,a)$ patří do stejné množiny rozkladu $Q/\overset{i-1}{\sim}$ jako $\delta(q,a)$, jinak patří do dvou nových množin, které vzniknou rozdělením množiny původní.

dokud platí $R_i \neq R_{i-1}$;
$Q/\sim = R_i$;

Nyní již můžeme přejít k samotné minimalizaci.

Věta 4.15 **V**

Ke každému konečnému automatu lze sestrojit jazykově ekvivalentní minimální konečný automat.

Ukážeme, jak takový automat zkonstruovat. Budeme předpokládat, že máme libovolný konečný automat (nedeterministický, neúplný).

Postup:

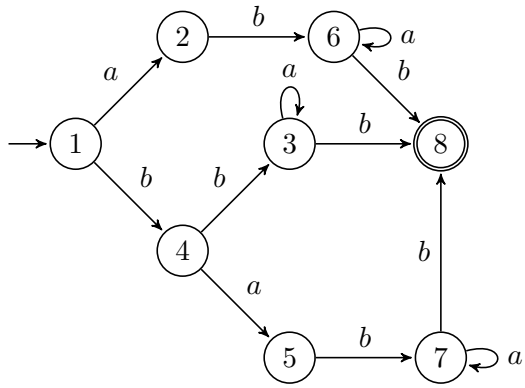
1. Vytvoříme ekvivalentní deterministický automat;
2. Zbavíme ho nedosažitelných stavů;
3. Pokud výsledný automat není úplný, převedeme konečný automat na úplný;
4. Postupně vytváříme rozklady množiny stavů podle relace ekvivalence $\overset{i}{\sim}, i \geq 0$;
5. Pokud $\overset{i-1}{\sim} = \overset{i}{\sim}$, získali jsme finální rozklad. Jednotlivé množiny rozkladu jsou novými stavy minimálního automatu. Vhodně je přeznačíme přeformulujeme přechodové zobrazení.

Tento postup demonstrujeme na následujícím příkladě:

Příklad 4.12 **P**

Nechť $M = (\{1, 2, \ldots, 8\}, \{a, b\}, \delta, 1, \{8\})$ je konečný automat deterministický a bez nedosažitelných stavů.

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



Při konstruci minimálního automatu je výhodné pracovat s tabulkou přechodové funkce.

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

Doplníme automat na úplný

Nyní rozdělíme množinu stavů na koncové a ostatní. Obě množiny označíme I a II. Zapíšeme, do které z množin vedou přechody z jednotlivých stavů. Pokud jsou hodnoty navzájem různé, vytvoříme množiny z těch stavů, které mají hodnoty stejné navzájem. Množiny nikdy neslučujeme!

			a	b
I	→	1	2	4
		2	P	6
		3	3	8
		4	5	3
		5	P	7
		6	6	8
		7	7	8
		P	P	P
II	←	8	P	8

			a	b
I	→	1	I	I
		2	I	I
		3	I	II
		4	I	I
		5	I	I
		6	I	II
		7	I	II
		P	I	I
II	←	8	II	II

			a	b
I	→	1	I	I
		2	I	II
		4	I	II
		5	I	II
		P	I	I
II		3	II	III
		6	II	III
		7	II	III
III	←	8	III	III

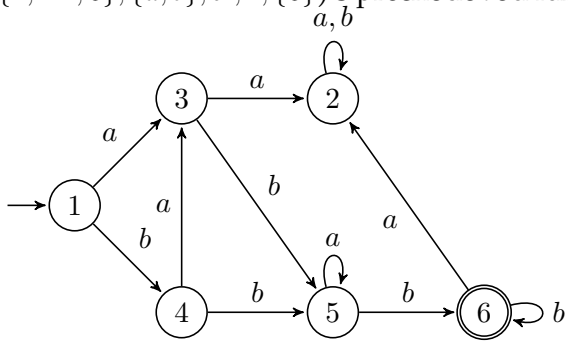
			a	b
I	$\rightarrow$	1	II	II
		P	I	I
II		2	I	III
		4	II	III
		5	I	III
III		3	III	IV
		6	III	IV
		7	III	IV
IV	$\leftarrow$	8	I	IV

			a	b
I	$\rightarrow$	1	III	IV
II		P	II	II
III		2	II	V
		5	II	V
IV		4	III	V
V		3	V	VI
		6	V	VI
		7	V	VI
VI	$\leftarrow$	8	II	VI

Toto byl poslední krok. Další rozdělení množin stavů není nutné, protože uvnitř množin stavy přechází vždy do stejných množin stavů.

Konečný automat přeznačíme a dostáváme výsledný minimalizovaný konečný automat $M' = (\{1,\ldots,6\},\{a,b\},\delta',1,\{6\})$ s přechodovou funkcí:

		a	b
$\rightarrow$	1	3	4
	2	2	2
	3	2	5
	4	3	5
	5	5	6
$\leftarrow$	6	2	6



Úkoly

1. Mějme konečný automat $M_1 = (\{q_0,q_1,q_2,q_F\},\{0,1\},\delta,q_0,\{q_F\})$ s přechodovou funkcí:

$$\begin{array}{llll} \delta(q_0,0) = \{q_0,q_1\} & & \delta(q_0,1) = \{q_0,q_2\} \\ \delta(q_1,0) = \{q_1,q_F\} & & \delta(q_1,1) = \{q_1\} \\ \delta(q_2,0) = \{q_2\} & & \delta(q_2,1) = \{q_2,q_F\} \\ \delta(q_F,0) = \varnothing & & \delta(q_F,1) = \varnothing \end{array}$$

   O jaký konečný automat se jedná? Uveďte alternativní způsoby zápisu přechodové funkce.

2. Mějme konečný automat $M_1 = (\{q_1,q_2,q_3\},\{a,b,c\},\delta,q_1,\{q_3\})$ s přechodovou funkcí:

$$\begin{array}{llll} \delta(q_1,a) = q_2 & \delta(q_1,b) = 3 & \delta(q_1,c) = q_2 \\ \delta(q_2,a) = q_3 & \delta(q_2,b) = 1 & \delta(q_2,c) = q_2 \\ \delta(q_3,a) = q_1 & \delta(q_3,b) = 2 & \delta(q_3,c) = q_1 \end{array}$$

(a) Napište příklad tří počátečních konfigurací. Jak vypadá koncová konfigurace?
(b) Vypište 10 nejkratších slov, která automat přijímá.
(c) Vypište pět slov nad abecedou $\{a,b,c\}$, která automat nepřijme.

3. Uvažujme nedeterministický konečný automat

$M_2 = (\{S,A,B,C\},\{a,b,c\},\delta,S,\{A\})$: $\begin{array}{rcl} \delta : \delta(S,a) & = & \{A\}, \\ \delta(S,c) & = & \{B\}, \\ \delta(B,b) & = & \{B,C\}, \\ \delta(C,a) & = & \{B\}, \\ \delta(C,c) & = & \{A\}. \end{array}$

Sestrojte deterministický úplný konečný automat, který bude ekvivalentní s automatem M_2.

4. Sestrojte konečný automat generující jazyk:
   (a) $L = \{w \in \{0,1\}^*\}$
   (b) $L = \{w \in \{0,1\}^* \mid w$ je sudé délky $\}$
   (c) $L = \{w \in \{0,1\}^* \mid$ počet symbolů 1 ve slově w je lichý $\}$
   (d) $L = \{w \in \{0,1\}^* \mid$ za každým podslovem 11 ve slově w ihned následuje $0\}$
   (e) $L = \{w \in \{0,1\}^* \mid w$ je sudé délky a počet 1 je dělitelný třemi $\}$
   (f) $L = \{w \in \{0,1\}^* \mid w$ začíná podslovem 1101 nebo končí $0000\}$
   (g) $L = \{u \in \{0,1\}^* \cdot v \in \{0,1\}^* \mid u$ je liché délky a obsahuje podslovo 0110, $v = (100)^i,\ i \geq 1\}$
5. Navrhněte konečný automat rozpoznávající jazyk všech slov nad abecedou $\{a,b,c\}$, ve kterých je součet počtů výskytů znaků a a b sudý.
6. Navrhněte konečný automat přijímající právě ta slova nad abecedou $\{a,b,c,d\}$, která nemají jako první znak a, nemají jako druhý znak b, nemají jako třetí znak c a nemají jako čtvrtý znak d. Jejich délka může být $i < 4$.
7. Minimalizuje konečné automaty zadané tabulkou:

(a)

		0	1
→	1	1	2
←	2	3	2
	3	2	2

(b)

		a	b
$\rightarrow$	1	1	3
	2	1	3
$\leftarrow$	3	6	5
	4	1	5
	5	2	4
$\leftarrow$	6	2	6

(c)

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

(d)

		a	b
$\rightarrow$	1	2	3
	2	2	4
$\leftarrow$	3	3	5
	4	2	7
$\leftarrow$	5	6	3
$\leftarrow$	6	6	6
	7	7	4
	8	2	3
	9	9	4

Kapitola 5

Zásobníkové automaty

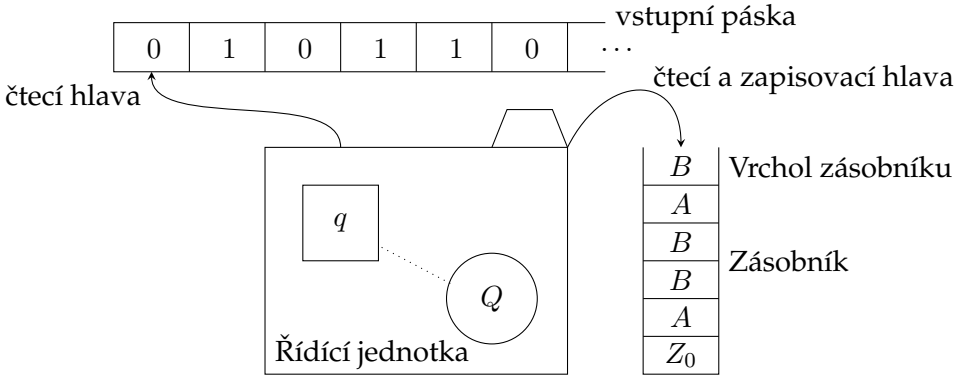
Dalším automatem, se kterým se seznámíme, je zásobníkový automat. Jedná se o stroj, který si můžeme představit jako konečný automat rozšířený o paměť, tzv. zásobník.

Zásobník představuje takový typ paměti, jenž umožňuje ukládat symboly z jisté abecedy; při čtení je však přístupný pouze naposledy uložený symbol. Jedná se tedy o paměť typu LIFO - last in, first out. Přístup k libovolnému symbolu v zásobníku získáme, odebereme-li ze zásobníku všechny později uložené symboly. Co se týče čtení symbolů ze zásobníku, jedná se o způsob podobný destruktivnímu čtení paměti počítače - čtením se uložená hodnota zničí - nabitý kondenzátor uchovávající hodnotu 1 se čtením vybije, po přečtení má hodnotu 0. Jinak řečeno, aby symbol v zásobníku mohl být přečten, je potřeba jej ze zásobníku odebrat.

V souladu s běžně užívanou konvencí budeme nazývat poslední obsazené políčko vrcholem zásobníku – vycházíme z představy svislého uspořádání paměťových buněk, přičemž je zaplňujeme odspodu nahoru. Schéma zásobníkového automatu je na obrázku 5.1.

Zásobníkový automat můžeme chápat jako zařízení, které se skládá z následujících částí:

- řídicí jednotka, která se může nacházet v některém z konečně mnoha stavů,
- vstupní páska stejné délky jako vstupní slovo, které je na ní zapsáno,
- čtecí hlava, která umožňuje řídící jednotce číst symbol z políčka vstupní pásky, nad nímž se nachází čtecí hlava,



Obrázek 5.1: Zařízení „zásobníkový automat“

- zásobník představovaný potenciálně nekonečnou páskou (dělenou na jednotlivá políčka, přičemž na každém políčku může být nejvýše jeden symbol z takzvané zásobníkové abecedy),
- čtecí a zapisovací halva, která umožňuje řídicí jednotce číst i odstraňovat symbol z vrcholu zásobníku a zapisovat symboly „nad vrchol zásobníku“.

V následujícím se budeme věnovat neformálnímu popisu funkce zásobníkového automatu. Nejdříve probereme výchozí předpoklady:

1. Na vstupní pásce bude zapsáno slovo tvořené ze symbolů vstupní abecedy zásobníkového automatu. Páska (dělená na jednotlivá políčka) bude mít na každém políčku právě jeden symbol a délka pásky bude stejná jako délka vstupního slova, tj. páska je konečné délky a na jejím konci nejsou žádná nevyužitá políčka.
2. Čtecí hlava je umístěna nad prvním (tj. nejlevějším) políčkem vstupní pásky (a je nachystána ke čtení 1. symbolu).
3. V zásobníku je uložen jediný symbol – takzvaný počáteční zásobníkový symbol.
4. Čtecí a zapisovací hlava je umístěna nad vrcholem zásobníku.
5. Zásobníkový automat se nachází v počátečním stavu.

Nyní popíšeme jednotlivé fáze průběhu výpočtu. Vycházíme z předpokladu, že řídicí jednotka má (díky čtecí i čtecí a zapisovací hlavě) vždy k dispozici informace o vstupním symbolu a vrcholu zásobníku. Výpočet probíhá

nedeterministicky, přičemž v každém okamžiku má zásobníkový automat na výběr několik možností kroků výpočtu.

1. V první řadě má možnosti plynoucí ze znalosti vstupního symbolu, aktuálního stavu a symbolu z vrcholu zásobníku. Každá z těchto možností zahrnuje nový stav řídicí jednotky a (případně prázdný) řetěz zásobníkových symbolů, jimiž bude „přepsán" vrchol zásobníku.

   V uvedeném případě automat přesune čtecí hlavu nad vstupní páskou o jedno políčko doprava, přejde do nového stavu, odstraní symbol z vrcholu zásobníku a do zásobníku zapíše příslušný řetěz, tj. je-li prázdný, nezapisuje nic, je-li délky 1, zapíše jej namísto odstraněného symbolu, a je-li délky alespoň 2, jeho první symbol zprava zapíše do zásobníku nejdříve, pak zapíše druhý symbol zprava atd. (tzn. po provedení zápisu se nejlevější symbol ocitne na vrcholu zásobníku).
2. V druhé řadě má možnosti obdobné předchozímu bodu, ale lišící se tím, že není vyžadována žádná znalost vstupního symbolu. Tzn. automat pro svůj výpočetní krok potřebuje znát pouze aktuální stav a symbol z vrcholu zásobníku. Výsledkem je pak přechod do nového stavu a „přepsání" vrcholu zásobníku – jinými slovy, automat v daném kroku nečetl vstupní symbol, neposunuje hlavu po vstupní pásce, pouze mění svůj stav a obsah zásobníku.

Jako poslední část neformálního popisu funkce zásobníkového automatu uvedeme, jakým způsobem je vyhodnocen výpočet. Uvedeme si dva způsoby, jak může zásobníkový automat přijímat vstupní slovo.

1. První způsob je analogický přijetí vstupního slova nedeterministickým konečným automatem, tj. je vyžadována existence výpočtu, který končí po přečtení celého vstupního slova v některém z vyznačených koncových stavů.
2. Druhý způsob využívá přítomnosti zásobníku a vyžaduje existenci takového výpočtu, který končí po přečtení celého vstupního slova a vyprázdněním zásobníku.

U každého zásobníkového automatu pochopitelně výslovně uvedeme, který z popsaných způsobů přijímání vstupních slov používá.

Nyní přejdeme k formální definici zásobníkového automatu.

Definice 5.1

Def
Zásobníkový automat

Zásobníkovým automatem (zkráceně ozn. ZA) nazýváme uspořádanou sedmici
$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

- Q je konečná neprázdná množina stavů,
- Σ je vstupní abeceda,
- Γ je zásobníková abeceda,
- δ je přechodové zobrazení; $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$,
- $q_0 \in Q$ je počáteční stav,
- $Z_0 \in \Gamma$ je počáteční symbol,
- $F \subseteq Q$ je množina koncových stavů.

Stejně jkao u konečného automatu nadefinujeme dále konfiguraci a krok výpočtu.

Definice 5.2

Def
Konfigurace ZA

Mějme libovolný ZA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Každou uspořádanou trojici $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ nazveme konfigurací zásobníkového automatu M.

Definice 5.3

Def
Krok výpočtu ZA

Krok výpočtu M definujeme jako binární relaci $\vdash$ na množině všech konfigurací takto: Pro všechna $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in \Sigma^*$, $Z \in \Gamma$, $\alpha, \beta \in \Gamma^*$ je

$$(p, ax, Z\alpha) \vdash (q, x, \beta\alpha),$$

právě když $\delta(p, a, Z)$ obsahuje (q, β).
Symbolem $\vdash^*$ budeme označovat reflexivní a tranzitivní uzávěr relace $\vdash$, symbolem $\vdash^+$ tranzitivní uzávěr relace $\vdash$.

Výpočet zásobníkového automatu definujeme prostřednictvím relace $\vdash^*$.

Podobně, jako tomu bylo u konečných automatů a nedeterministických konečných automatů, i v případě zásobníkových automatů konfigurace jednoznačně popisuje situaci, v níž se stroj nachází v průběhu výpočtu. Tuto situaci zřejmě určuje aktuální stav, dosud nepřečtená část vstupního slova a aktuální obsah zásobníku. Tj. konfiguraci (q, w, α) odpovídá situace, kdy se daný zásobníkový automat nachází ve stavu q, dosud nepřečtená část vstupního slova je rovna w a v zásobníku je zapsán řetěz α, přičemž vrchol zásobníku obsahuje nejlevější symbol řetězu α.

V definici kroku výpočtu si všimněte požadavku na neprázdný obsah zásobníku. Jinak řečeno, dojde-li v průběhu výpočtu k vyprázdnění zásobníku, není další výpočetní krok definován.

Pokud si dobře prohlédneme definici zásobníkového automatu, docházíme k faktu, že je definovaný jako nedeterministický. Nedeterminismus spočívá nejen v definici přechodové funkce - podobně jako tomu bylo u konečného automatu, ale také v možnosti nečíst vstupní symbol a zpracovávat „pouze“ symboly ze zásobníku a měnit aktuální stav.

Příklad 5.1 P

$\delta(p, a, Z) = \{(q_1, a), (q_2, \varepsilon)\}$ a $\delta(p, \varepsilon, Z) = \{(q_3, Z), (q_4, aZ)\}$ máme v binární relaci $\vdash$ následující dvojice konfigurací:

$(p, a, Z) \vdash (q_1, \varepsilon, a)$,
$(p, a, Z) \vdash (q_2, \varepsilon, \varepsilon)$,
$(p, a, Z) \vdash (q_3, a, Z)$,
$(p, a, Z) \vdash (q_4, a, aZ)$.

Výpočet obsahující konfiguraci (p, a, Z) bude tedy zahrnovat čtyři různé výpočty, z nichž každý pokračuje po konfiguraci (p, a, Z) jinou z výše uvedených konfigurací.

Jak bylo již dříve předesláno, pro zásobníkový automat lze zavést dva různé způsoby přijetí vstupního slova, což obecně vede ke dvěma různým jazykům, jež přijímá tentýž zásobníkový automat. Budeme mezi nimi roz-

lišovat tak, že v definici zásobníkového automatu, který přijímá prázdným zásobníkem budeme uvádět prázdnou množinu koncových stavů.

Definice 5.4

Def

Jazyk přijímaný koncovým stavem

Nechť $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Množinu

$$L(M) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \alpha) \text{ pro nějaké } q_f \in F, \alpha \in \Gamma^*\}$$

nazveme jazykem přijímaným (rozpoznávaným) zásobníkovým automatem M koncovým stavem.

Definice 5.5

Def

Jazyk přijímaný prázdným zásobníkem

Nechť $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$ je zásobníkový automat. Množinu

$$L_\varepsilon(M) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \text{ pro nějaké } q \in Q\}$$

nazveme jazykem přijímaným (rozpoznávaným) zásobníkovým automatem M prázdným zásobníkem.

Uvedeme nyní příklady zásobníkových automatů a jazyků, které akceptují.

Příklad 5.2

P

Sestrojte zásobníkový automat, který přijímá jazyk $L = \{0^n 1^n \mid n \leq 1\}$ koncovým stavem.

Nejdříve navrhneme činnost zásobníkového automatu neformálně. Dokud bude ZA číst symboly 0, setrvá v počátečním stavu q_0 a za každý přečtený symbol 0 uloží do zásobníku jeden symbol 0. Tím bude zajištěno zkopírování všech symbolů 0 do zásobníku. Při přečtení prvního symbolu 1 přejde ZA do stavu q_1 a ze zásobníku odebere jeden symbol 0. Ve stavu q_1 ZA setrvá, dokud bude číst symboly 1 – přitom bude odstraňovat symboly 0 ze

zásobníku. Zůstane-li v zásobníku pouze symbol Z_0, přejde do koncového stavu, neboť právě dočetl řetězec ve tvaru 0^n1^n.

Definice příslušného zásobníkového automatu vypadá následovně:
$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde:

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{Z_0, 0\}$,
- $F = \{q_2\}$,
- 1 $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$
  2 $\delta(q_0, 0, 0) = \{(q_0, 00)\}$
  3 $\delta(q_0, 1, 0) = \{(q_1, \varepsilon)\}$
  4 $\delta(q_1, 1) = \{(q_1, \varepsilon)\}$
  5 $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$

Nyní uvedeme příklady výpočtu zásobníkového automatu s různými slovy na vstupní pásce.

Symbolem $\overset{i}{\vdash}, 1 \geq i \geq 5$ označíme krok výpočtu a přechodovou funkci v něm použitou.

Nechť $(q_0, 000111, Z_0)$ je počáteční konfigurace ZA M. Pak další výpočet probíhá následujícím způsobem:
$(q_0, 000111, Z_0) \overset{1}{\vdash} (q_0, 00111, 0Z_0) \overset{2}{\vdash} (q_0, 0111, 00Z_0) \overset{2}{\vdash} (q_0, 111, 000Z_0) \overset{3}{\vdash}$
$\overset{3}{\vdash} (q_1, 11, 00Z_0) \overset{4}{\vdash} (q_1, 1, 0Z_0) \overset{4}{\vdash} (q_1, \varepsilon, Z_0) \overset{5}{\vdash} (q_2, \varepsilon, Z_0)$

Pokud slovo na vstupní pásce nepatří do jazyka akceptovaného ZA M, zásobníkový automat skončí výpočet v jiné konfiguraci než koncové. Uvedeme výpočty se slovy 000110, 00111, 00011 a 1100.

1. 000110:
   $(q_0, 000110, Z_0) \overset{1}{\vdash} (q_0, 00110, 0Z_0) \overset{2}{\vdash} (q_0, 0110, 00Z_0) \overset{2}{\vdash} (q_0, 110, 000Z_0) \overset{3}{\vdash}$
   $\overset{3}{\vdash} (q_1, 10, 00Z_0) \overset{4}{\vdash} (q_1, 0, 0Z_0)$
2. 00111:
   $(q_0, 00111, Z_0) \overset{1}{\vdash} (q_0, 0111, 0Z_0) \overset{2}{\vdash} (q_0, 111, 00Z_0) \overset{3}{\vdash} (q_1, 11, 0Z_0) \overset{4}{\vdash}$
   $\overset{4}{\vdash} (q_1, 1, Z_0)$
3. 00011:
   $(q_0, 00011, Z_0) \overset{1}{\vdash} (q_0, 0011, 0Z_0) \overset{2}{\vdash} (q_0, 011, 00Z_0) \overset{2}{\vdash} (q_0, 11, 000Z_0) \overset{3}{\vdash}$
   $\overset{3}{\vdash} (q_1, 1, 00Z_0) \overset{4}{\vdash} (q_1, \varepsilon, 0Z_0)$
4. 1100:
   $(q_0, 1100, Z_0)$

Z definice přechodového zobrazení vyplývá, že vytvořený zásobníkový automat je deterministický.

Příklad 5.3 **P**

Sestrojte zásobníkový automat, který přijímá jazyk $L = \{ww^R \mid w \in a, b^*\}$ prázdným zásobníkem.

Nejdříve navrhneme činnost zásobníkového automatu neformálně.

Hlavní princip zpracování slova, které se skládá ze dvou částí w a w^R bude spočívat v zapsání první části do zásobníku (zápis „šablony“) a umazávání symbolů ze zásobníku v průběhu čtení druhé části slova v případě, že se bude shodovat symbol čtený na vstupní pásce a symbol na vrcholu zásobníku (kontrola zrcadlového obrazu). ZA bude načítat symboly a zapisovat je do zásobníku. V momentě, kdy narazí na dva stejné symboly za sebou, bude mít možnost volby, jestli druhý symbol zapsat do zásobníku, nebo začít zásobník umazávat. Výpočet skončí úspěšně, pokud ZA po zpracování celého slova umaže v posledním kroku výpočtu zásobníkový symbol Z_0. Protože do jazyka patří i prázdné slovo, musí mít automat možnost umazat tento zásobníkový symbol i hned v prvním kroku výpočtu.

Definice příslušného zásobníkového automatu vypadá následovně:

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$, kde:

- $Q = \{q_0, q_1\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{Z_0, a, b\}$,
- 1 $\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$
  2 $\delta(q_0, b, Z_0) = \{(q_0, b, Z_0)\}$
  3 $\delta(q_0, a, a) = \{(q_0, aa), (q_1, \varepsilon)\}$
  4 $\delta(q_0, a, b) = \{(q_0, ab)\}$
  5 $\delta(q_0, b, a) = \{(q_0, ba)\}$
  6 $\delta(q_0, b, b) = \{(q_0, bb), (q_1, \varepsilon\}$
  7 $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$
  8 $\delta(q_1, b, b) = \{(q_1, \varepsilon)\}$
  9 $\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$
  10 $\delta(q_0, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

Nyní uvedeme příklady výpočtu zásobníkového automatu s různými slovy na vstupní pásce.

Symbolem $\overset{i}{\vdash}, 1 \geq i \geq 10$ označíme krok výpočtu a přechodovou funkci v něm použitou.

Nechť $(q_0, abaaba, Z_0)$ je počáteční konfigurace ZA M. Pak další výpočet probíhá následujícím způsobem:

$(q_0, abaaba, Z_0) \overset{1}{\vdash} (q_0, baaba, aZ_0) \overset{5}{\vdash} (q_0, aaba, baZ_0) \overset{4}{\vdash} (q_0, aba, abaZ_0) \overset{3}{\vdash}$
$\overset{3}{\vdash} (q_1, ba, baZ_0) \overset{8}{\vdash} (q_1, a, aZ_0) \overset{7}{\vdash} (q_1, \varepsilon, Z_0) \overset{9}{\vdash} (q_1, \varepsilon, \varepsilon)$

Protože je ZA M nedeterministický, může proběhnout s jedním vstupním slovem více výpočtů. Pro slovo $abaaba$ existují výpočty dva. První, akceptující jsme již uvedli. Nyní ukážeme, jak vypadá druhý, neakceptující výpočet.

$(q_0, abaaba, Z_0) \overset{1}{\vdash} (q_0, baaba, aZ_0) \overset{5}{\vdash} (q_0, aaba, baZ_0) \overset{4}{\vdash} (q_0, aba, abaZ_0) \overset{3}{\vdash}$
$\overset{3}{\vdash} (q_0, ba, aabaZ_0) \overset{5}{\vdash} (q_0, a, baabaZ_0) \overset{4}{\vdash} (q_0, \varepsilon, abaabaZ_0)$

Věta 5.1 V

Jazyk $L = L_\varepsilon(M)$ pro nějaký ZA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing), F \neq \varnothing$ právě tehdy, když $L = L(M')$ pro nějaký ZA $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F')$.

Idea důkazu:

1. $\Longleftarrow$: k danému ZA M' zkonstruujeme ZA M simulující jeho činnost. Kdykoli M' přejde do koncového stavu, bude mít M možnost volby, zda pokračovat v simulaci automatu M' nebo přejít do svého, nově přidaného stavu q_ε, v němž vyprázdní svůj zásobník.

V průběhu výpočtu ZA M' může nastat situace, kdy dojde k vymazání zásobníku. Takový vstup by nový ZA M přijal bez ohledu na to, že automat M' nebyl v koncovém stavu a vstupní slovo by neakceptoval. Musíme tedy zabránit tomu, aby M v tomto bodě vstup (prázdným zásobníkem) akceptoval. Řešení spočívá v tom, že ještě před zahájením simulace výpočtu ZA M' uložíme na dno zásobníku ZA M nový symbol, který nedovolíme odstanit jinak, než když stroj M' přejde koncovém stavu $q \in F$ nebo do stavu q_ε.

Nechť $M' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0, F')$ je ZA takový, že $L = L(M')$. Sestrojme $M = (Q, \Sigma, \Gamma, \delta, q_0, Z', \varnothing)$ takový, že $L = L_\varepsilon(M)$.

Položme $M = (Q' \cup \{q_0, q_\varepsilon\}, \Sigma, \Gamma' \cup \{Z'\}, \delta, q_0, \varnothing)$, kde $Z' \notin \Gamma'$ a $q_0', q_\varepsilon \notin Q$ a kde δ je definována takto:

1. $\delta(q_0, \varepsilon, Z') = \{(q_0, Z_0 Z')\}$,
2. jestliže $\delta'(q, a, Z)$ obsahuje (r, γ), pak $\delta(q, a, Z)$ obsahuje (r, γ), pro všechna $q \in Q, a \in \Sigma \cup \{\varepsilon\}$ a $Z \in \Gamma$,
3. $\forall q \in F, \forall Z \in \Gamma \cup \{Z'\} : \delta(q, \varepsilon, Z)$ obsahuje $(q_\varepsilon, \varepsilon)$,
4. $\forall Z \in \Gamma\{Z'\} : \delta(q_\varepsilon, \varepsilon, Z) = \{q_\varepsilon, \varepsilon)\}$.

2. $\Longrightarrow$: M' simuluje činnost M a má opět nově přidaný symbol jako své dno zásobníku. Jakmile je M' schopen číst tento symbol (tj. zásobník automatu M je prázdný), pak M' přejde do nově přidaného stavu q_f, který je koncovým stavem.

Nechť $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$ je zásobníkový automat takový, že $L = L_\varepsilon(M)$. Sestrojme M' takový, že $L = L(M')$.

Položme $M' = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{Z'\}, \delta', q_0', Z', \{q_f\})$, kde δ' je definována takto:

1. $\delta'(q_0', \varepsilon, Z') = \{(q_0, Z_0 Z')\}$,
2. jestliže $\delta(q, a, Z)$ obsahuje (r, γ), pak $\delta'(q, a, Z)$ obsahuje (r, γ) pro všechna $q \in Q, a \in \Sigma \cup \{\varepsilon\}$ a $Z \in \Gamma$,
3. $\forall q \in Q : \delta'(q, \varepsilon, Z') = \{(q_f, \varepsilon)\}$;

□

Úkoly

1. Mějme zásobníkový automat $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, Z_0\}\delta, q_0, \varnothing)$ s přechodovým zobrazením definovaným takto:

$$\begin{aligned} \delta(q_0, a, Z_0) &= \{(q_0, aa)\}, \\ \delta(q_0, a, a) &= \{(q_0, aa), (q_1, a)\}, \\ \delta(q_1, b, a) &= \{(q_2, \varepsilon)\}, \\ \delta(q_2, a, a) &= \{(q_1, \varepsilon)\} \end{aligned}$$

   - Určete, zda se jedná o deterministický nebo nedeterministický zásobníkový automat.
   - Jakým způsobem M akceptuje jazyk?
   - Určete jazyk akceptovaný zásobníkovým automatem.

2. Sestrojte zásobníkový automat, který bude akceptovat jazyk $L = \{a^i b^j \mid i > 0, j = 2i\}$ prázdným zásobníkem. Upravte tento zásobníkový automat tak, aby přijímal jazyk L koncovým stavem.
3. Sestrojte zásobníkový automat, který bude akceptovat jazyk $L = \{a^i b^j \mid 0 \leq i < j\}$ koncovým stavem. Upravte tento zásobníkový automat tak, aby přijímal jazyk L prázdným zásobníkem.
4. Sestrojte zásobníkový automat, který bude akceptovat jazyk $L = \{wcw^R \mid w \in \{a, b\}^*, |w| \leq 3\}$ prázdným zásobníkem. Upravte tento zásobníkový automat tak, aby přijímal jazyk L koncovým stavem.

Kapitola 6

Bezkontextové gramatiky a jazyky

V této kapitole se budeme podrobněji zabývat bezkontextovými gramatikami (CF gramatikami) a jazyky jimi generovanými – bezkontextovými jazyky (CF jazyky). Připomeneme definici bezkontextových gramatik.

Definice 6.1

Def

Bezkontextová gramatika

Bezkontextová gramatika je gramatika $G = (N, T, P, S)$, jejíž pravidla jsou ve tvaru $A \to \beta$,kde $A \in N,\ \beta \in (N \cup T)^*,\ |\beta| \leq 1$. Výjimku tvoří pravidlo $S \to \varepsilon$ pod podmínkou, že neterminál S se nevyskytuje na pravé straně žádného z pravidel.

Odvození slov v bezkontextové gramatice probíhá stejným způsobem, jak jsme popsali u gramatik regulárních. V odvození v regulární gramatice, bylo v každém kroku odvození vybíráno pravidlo z množiny pravidel, které měly na levé straně totožný neterminál. Každá větná forma totiž obsahovala neterminál pouze jeden. Nyní u bezkontextových gramatik může větná forma obsahovat více neterminálů (i různých) a je proto nejdříve nutné provést výběr neterminálu, na který budeme aplikovat přepisovací pravidlo a teprve poté vybírat pravidlo. Z uvedeného vyplývá, že i pro jedno slovo

může existovat více odvození, která se liší právě tím, v jakém pořadí jsou vybírány neterminály pro aplikaci pravidel. Osvětlíme si tuto problematiku na příkladě.

Příklad 6.1 **P**

Mějme gramatiku $G = (\{S, A\}, \{a, b\}, \{S \to aSA \mid a,\ A \to bAS \mid b\}, S)$. Uvedeme dvě různá odvození téhož slova $aabbba$.

$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaaAA \Rightarrow aaabA \Rightarrow aabbAS \Rightarrow aabbAa \Rightarrow aabbba$

$S \Rightarrow aSA \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aaSAbAa \Rightarrow aaaAbAa \Rightarrow aaabbAa \Rightarrow aabbba$

Speciálním případem odvození jsou tzv. kanonické odvození. Jsou to taková odvození, kde v každém kroku je přepsán vždy nejlevější příp. nejpravější neterminál. Takové odvození se nazývá levé příp. pravé.

Definice 6.2 **Def**

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Řekneme, že řetězec α lze přepsat na řetězec β v gramatice G levým přepsáním, jestliže existuje v gramatice G pravidlo $A \to \gamma \in P$ takové, že $\alpha = \xi A \eta$, $\beta = \xi \gamma \eta$, přičemž $\xi \in T^*$, $\eta \in (N \cup T)^*$. Jestliže platí $\xi \in (N \cup T)^*$ a $\eta \in T^*$, pak hovoříme o pravém přepsání.

Levé a pravé přepsání

Odvození se nazývá levé (příp. pravé), pokud je každý krok odvození levým (příp. pravým) přepsáním.

Levé a pravé odvození

Odvození slova $aabbba$ v gramatice G z příkladu 6.1 nejsou ani levá ani pravá. Ukážeme, jak vypadá levé a pravé odvození.

$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaaAA \Rightarrow aaabA \Rightarrow aabbAS \Rightarrow aabbbS \Rightarrow aabbba$

$S \Rightarrow aSA \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aSbba \Rightarrow aaSAbba \Rightarrow aaSbbba \Rightarrow aabbba$

6.1 Strom odvození, nejednoznačné gramatiky

Abychom mohli porovnat, do jaké míry jsou si tato odvození podobná, příp. jak moc se liší, zavádíme grafické zobrazení odvození, které se nazývá derivační strom. Grafické znázornění svou strukturou připomíná graf, který se nazývá strom a přívlastek derivační pochází ze slova derivace, jak se také někdy nazývá odvození.

Definice 6.3 **Def**

Derivační strom

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Strom H nazveme derivačním stromem v G právě, když platí tyto podmínky:

1. každý uzel má návěští, které je symbolem z $N \cup T \cup \{\varepsilon\}$,
2. kořen má návěští S,
3. má-li vnitřní uzel návěští A, pak $A \in N$,
4. má-li uzel n návěští A a jeho všichni potomci $n_1, \ldots, n_k$ mají v uspořádání zleva doprava návěští $X_1, \ldots, X_k$, pak $A \to X_1 \ldots X_k \in P$,
5. má-li uzel n návěští ε, pak n je list a je jediným potomkem svého otce.

Výsledkem derivačního stromu H nazveme slovo vzniklé zřetězením návěští listů v uspořádání zleva doprava.

V následujícím příkladě vytvoříme derivační strom.

Příklad 6.2 **P**

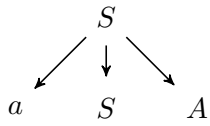
Vytvořme derivační strom k prvnímu odvození slova $aabbba$ v bezkontextové gramatice $G = (\{S, A\}, \{a, b\}, \{S \to aSA \mid a,\ A \to bAS \mid b\}, S)$, které je uvedeno před definicí derivačního stromu.

$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaaAA \Rightarrow aaabA \Rightarrow aabbAS \Rightarrow aabbAa \Rightarrow aabbba$

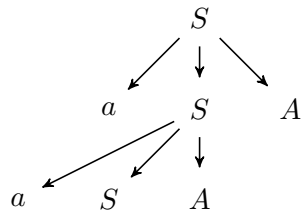
Podle odvození sestavíme derivační strom. Nejdříve umístíme kořen stromu - uzel s návěštím startovacího symbolu.

S

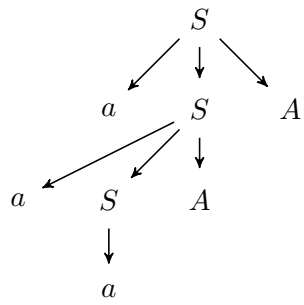
V prvním kroku odvození je startovací neterminál přepsán podle pravidla $S \to aSA$.



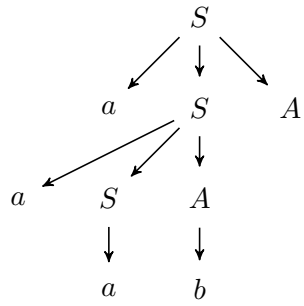
Ve druhém kroku je použito stejné pravidlo $S \to aSA$ tentokrát na nově vzniklý neterminál S.



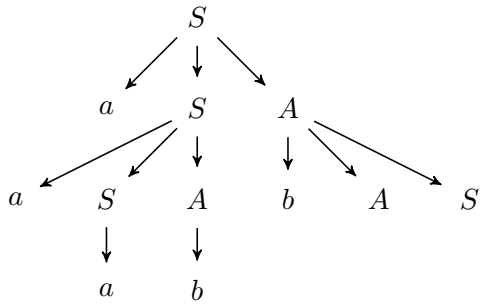
Ve třetím kroku je opět přepsán neterminál S. Tentokrát je použito pravidlo $s \to a$.



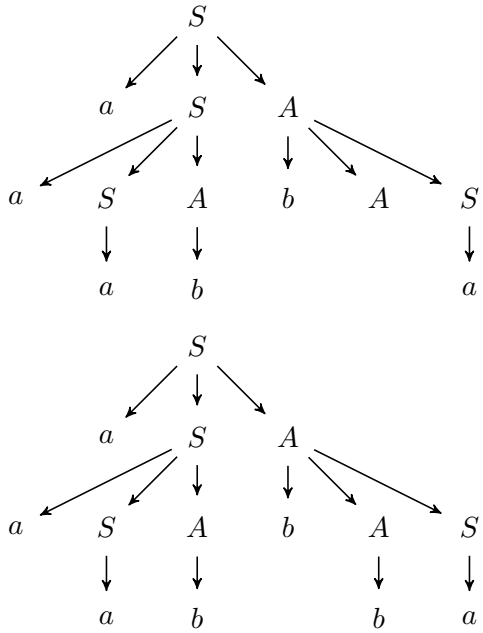
Ve čtvrtém kroku je první výskyt neterminálu A přepsán pomocí pravidla $A \to b$.



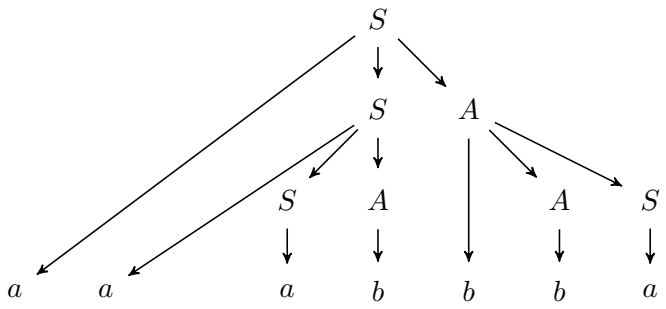
V pátém kroku je pravidlo $A \to bAS$ aplikováno na jediný zbývající neterminál A.



V šestém (předposledním) kroku je přepsán neterminál S pomocí pravidla $S \to a$ a v posledním, sedmém kroku je odvození dokončeno aplikací pravidla $A \to b$.



Odvozené slovo získáme přečtením terminálních symbolů postupně zleva doprava. Aby bylo zřejmé, jaké slovo vzniklo odvozením, které znázorňuje derivační strom, můžeme listy stromu „protáhnout“ až na spodní hranici stromu.



Toto znázornění se používá pouze tehdy, pokud potřebujeme zdůraznit existenci terminálních symbolů nebo symbolu ε.

Pokud vytvoříme derivační strom i pro druhé odvození slova z příkladu 6.1, dostáváme stejný derivační strom jako pro odvození první. Mohou v gramatice existovat pro jedno slovo tak rozdílná odvození, že mají každé má jiný strom odvození? Odpověď nalezneme v následujícím příkladě.

Příklad 6.3 P

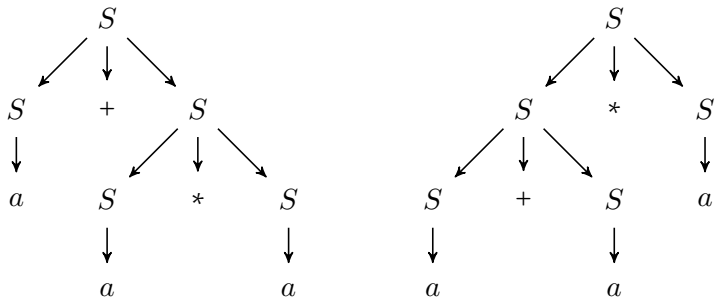
Mějme gramatiku $G = (\{S\}, \{a, +, *\}, \{S \to S + S \mid S * S \mid a\}, S)$. Odvodíme v gramatice G slovo $a + a * a$.

$S \Rightarrow S + S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * a$

Vidíme, že jsme jako první pravidlo použili pravidlo $S \to S + S$. Je možné začít odvozování jiným pravidlem?

$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * a$

Vytvoříme k těmto dvěma odvozením derivační stromy. Vlevo je derivační strom prvního odvození a vpravo pak derivační strom druhého odvození.



Pokud budeme studovat derivační stromy dojdeme k poznatku, že ke každému derivačnímu stromu existuje jediné levé odvození a ke každému levému odvození existuje jediný derivační strom. Na základě této myšlenky můžeme vyslovit následující definici.

Definice 6.4

Def

Nejednoznačná gramatika

Bezkontextová gramatika G se nazývá nejednoznačná, jestliže existuje slovo $w \in L(G)$ takové, že v gramatice G existují alespoň dvě různá levá odvození tohoto slova. Jinak se gramatika G nazývá jednoznačná.

Z uvedeného vyplývá, že ke slovu $w \in L(G)$ v nejednoznačné gramatice G existuje více derivačních stromů. Nejednoznačnost gramatiky s sebou nese praktické problémy. Například nejednoznačná gramatika umožňuje provést více různých rozborů jednoho řetězce a pokud by rozbor umožňoval interpretaci významu řetězce, vzniká problém, kterou interpretaci zvolit. Proto se vždy, pokud to jde, snažíme sestrojit gramatiku jednoznačnou. Existují však jazyky, které žádná jednoznačná gramatika negeneruje.

Příklad 6.4

P

Mějme gramatiku $G = (\{S, A, C, X, Y\}, \{a, b, c\}, P, S)$ s množinou pravidel:

$$\begin{aligned} S &\to XC|X|C|AY|Y|A|\varepsilon \\ X &\to aXb|ab \\ Y &\to bYc|bc \\ A &\to aA|a \\ C &\to cC|c \end{aligned}$$

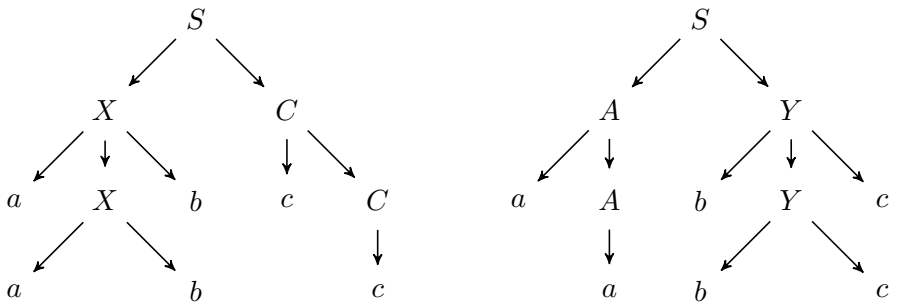
Bezkontextová gramatika G generuje jazyk

$$L = \{a^i b^j c^k | i, j, k \geq 0, i = j \text{ nebo } j = k\}.$$

Tato gramatika je nejednoznačná protože slova tvaru $a^n b^n c^n, n \geq 1$ mohou být generována dvěma různými způsoby - dvěma různými levými odvozeními. Předvedeme tuto skutečnost na slově $a^2b^2c^2$.

$S \Rightarrow XC \Rightarrow aXbC \Rightarrow aabbC \Rightarrow aabbcC \Rightarrow aabbcc$

$S \Rightarrow AY \Rightarrow aAY \Rightarrow aaY \Rightarrow aabYc \Rightarrow aabbcc$



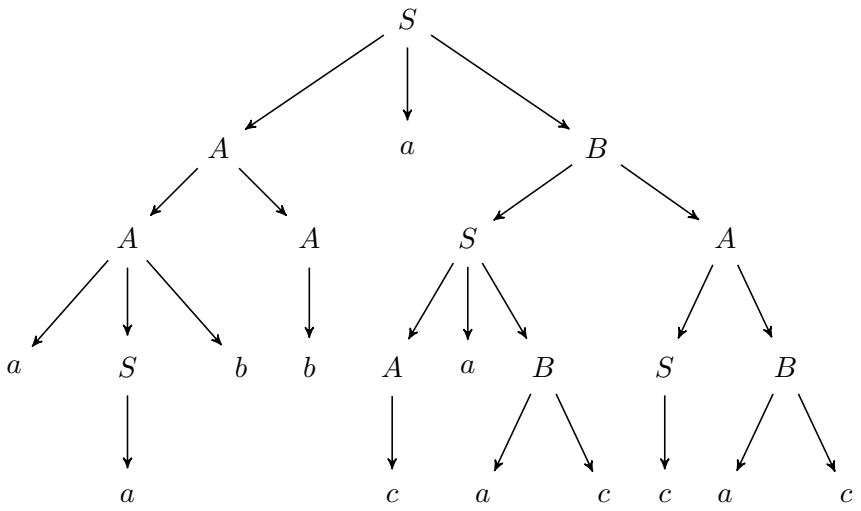
I když jsou odvození na první pohled různá, pro úplnost jsme zde uvedli i stromy odvození.

Vzhledem k charakteristice jazyka - sjednocení dvou jednodušších jazyků - nemůže existovat jednoznačná gramatika, která tento jazyk generuje. Problémem jsou slova, která patří do průniku obou jazyků, nemůže být jednoznačně dáno, jestli má být generováno částí generující první nebo druhý jazyk.

Úkoly

1. Mějme strom odvození zobrazený na obrázku 6.1.
   - Zjistěte, odvození kterého slova strom zachycuje.

- Zapište pravidla gramatiky G, která byla k odvození tohoto slova zapotřebí.
- Napište levé a pravé odvození tohoto slova v gramatice G.



Obrázek 6.1: Strom odvození v gramatice G

2. Napište tři slova generovaná gramatikou $G = (\{S, A, B\}, \{a, b, c\}, P, S)$ s množinou pravidel

$$\begin{array}{lllll} P = \{ & S & \to & aAB \mid aBa \mid bABa \\ & A & \to & bAa \mid bab \mid SS \\ & B & \to & cbc \mid aBaa \mid bSSc \quad \} \end{array}$$

Vytvořte pro tato slova stromy odvození.

6.2 Úpravy bezkontextových gramatik

Při tvorbě gramatik akceptujících určitý jazyk zejména využitím různých automatizovaných činností může vzniknout gramatika, která obsahuje velké množství pravidel a symbolů. Ne všechny musí být k akceptování slov daného jazyka potřebné, ne použitím všech neterminálů musí jít vygenerovat terminální slovo. Taková pravidla a množství jak terminálních tak neterminálních symbolů gramatiku pouze znepřehledňují.

Definice 6.5 **Def**

Řekneme, že symbol $X \in N \cup T$ je nepoužitelný v bezkontextové gramatice $G = (N, T, P, S)$ právě tehdy, když v gramatice G neexistuje odvození $S \Rightarrow^* wXy \Rightarrow^* wxy$, pro nějaké $w, x, y \in T^*$. Řekneme, že gramatika G je redukovaná, jestliže neobsahuje žádný nepoužitelný neterminál.

Výše uvedená definice postihuje dvě podmínky nepoužitelnosti:

1. neexistence první části uvedeného odvození má za následek, že symbol $X \in N \cup T$ se nevyskytuje v žádné větné formě odvoditelné z S (jedná se o tzv. nedosažitelný symbol),
2. neexistence druhé části odvození vyjadřuje skutečnost, že z neterminálního symbolu X nelze odvodit žádný terminální řetězec (tzv. redundantní neterminál).

Poznamenejme ještě, že existence obou výše zmíněných odvození ještě není postačující podmínkou pro použitelnost symbolu. Vždy musíme gramatiku zkoumat jako celek. Symbol X se může objevit třeba jen v takové větné formě, která obsahuje jiný redundantní neterminál.

Lemma 6.1 **L**

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pak existuje algoritmus, který zkonstruuje bezkontextovou gramatiku G' bez redundantních symbolů takovou, že $L(G) = L(G')$.

Idea algoritmu spočívá v sestrojení množiny neredundantních symbolů a z následné úpravy gramatiky.

Algoritmus 6.1 Odstranění redundantních symbolů

A

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$
Výstup: Ekvivalentní bezkontextová gramatika $G' = (N', T', P', S)$ bez redundantních symbolů

$i = 0; N_0 = \varnothing;$
dělej
  $i = i + 1;$
  $N_i = N_{i-1} \cup \{A \mid A \to \alpha \in P; \alpha \in (N_{i-1} \cup T)^*\};$
dokud platí $N_i \neq N_{i-1}$;
$G' = (N \cap N_i, T, P \cap (N_i \times (N_i \cup T)^*), S);$

Funkci algoritmu Odstranění redundantních neterminálů ukážeme na příkladě. Pro lepší pochopení iteračního vytváření množin neredundantních neterminálů budeme postupovat krok po kroku i vysvětlivkami.

Příklad 6.5

P

Mějme gramatiku $G = (\{S, A, B, C, D\}, \{a, b, c, d\}, P, S)$ s množinou pravidel:

$S \to aACb|c$
$A \to bA|dC$
$B \to bB|Dc$
$C \to Bc|aA|d$
$D \to Ba$

Při sestavování množin N_i budeme postupovat přesně podle algoritmu.

1. inicializace – $i = 0;\ N_0 = \varnothing;$
2. první průchod cyklem - $i = 1;\ N_1 = \{A \mid A \to \alpha; \alpha \in T^*\} = \{S, C\}$. Procházíme pravé strany pravidel a zkoumáme, které umíme sestavit pouze s terminálních symbolů. Pokud takovou pravou stranu nalezneme, levou stranu téhož pravidla přidáme do množiny N_1;
3. test na rovnost – $N_1 \neq N_0$ - pokračujeme v cyklu;

4. druhý průchod cyklem – $i = 2$;

   $N_2 = \{S, C\} \cup \{A \mid A \to \alpha; \alpha \in (T \cup \{S, C\})^*\} = \{S, C, A\}$

   Procházíme pravé strany pravidel a zkoumáme, které umíme sestavit s terminálních symbolů a neterminálů S a C. Pokud takovou pravou stranu nalezneme, levou stranu téhož pravidla přidáme do množiny N_2;
5. test na rovnost – $N_2 \neq N_1$ – pokračujeme v cyklu;
6. druhý průchod cyklem – $i = 3$;

   $N_3 = \{S, C, A\} \cup \{A \mid A \to \alpha; \alpha \in (T \cup \{S, C, A\})^*\} = \{S, C, A\}$

   Procházíme pravé strany pravidel a zkoumáme, které umíme sestavit s terminálních symbolů a neterminálů S a C. Pokud takovou pravou stranu nalezneme, levou stranu téhož pravidla přidáme do množiny N_3.
7. test na rovnost – $N_3 \neq N_2$ – ukončení cyklu;
8. sestavení gramatiky – výsledná gramatika obsahuje pouze takové neterminály, které jsou v množině N_3, a taková pravidla, na která „máme stavební materiál" – $G' = (\{S, A, C\}, \{a, b, c, d\}, P', S)$, kde množina pravidle obsahuje pravidla:

   $$\begin{array}{lcl} S & \to & aACb|c \\ A & \to & bA|dC \\ C & \to & aA|d \end{array}$$

Následující věta se týká důležité otázky a to jestli je gramatika schopná vygenerovat byť i jediný terminální řetězec.

Věta 6.2

V

Existuje algoritmus, který pro každou bezkontextovou gramatiku ověří, jestli generuje neprázdný jazyk.

Algoritmus, který dá odpověď na otázku, jestli jazyk, který generuje daná bezkontextová gramatiky je prázdný nebo ne, vychází z algoritmu odstranění redundantních symbolů. Pokud po skončení cyklu množina N_i neobsahuje startovací neterminál, pak je tento neterminál redundantní a nelze z něj odvodit žádné terminální slovo a tudíž jazyk generovaný touto gramatikou je prázdný.

Algoritmus 6.2 $L(G)\ ?\ \varnothing$

A

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$
Výstup: „ANO“ jestliže $L(G) \neq \varnothing$, „NE“ jinak

$i = 0; N_0 = \varnothing$;
dělej
  $i = i + 1$;
  $N_i = N_{i-1} \cup \{A \mid A \to \alpha \in P; \alpha \in (N_{i-1} \cup T)^*\}$;
dokud platí $N_i \neq N_{i-1}$;
jestliže $S \in N_i$, pak výstup= „ANO“, jinak výstup= „NE“;

Protože algoritmus je až na poslední krok totožný s algoritmem odstranění redundantních symbolů, nebudeme jeho funkci rozebírat na příkladě.

Lemma 6.3

L

Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pak existuje algoritmus, který zkonstruuje bezkontextovou gramatiku G' bez nedosažitelných symbolů takovou, že $L(G) = L(G')$.

Idea algoritmu spočívá v iterativním sestrojení množiny dosažitelných symbolů a s následné úpravy gramatiky.

Algoritmus 6.3 Odstranění nedosažitelných symbolů

A

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$
Výstup: Ekvivalentní bezkontextová gramatika $G' = (N', T', P', S)$ bez nedosažitelných symbolů

$i = 0; V_0 = \{S\};$
dělej
  $i = i + 1;$
  $V_i = V_{i-1} \cup \{X \mid \exists A : A \to \alpha X \beta \in P \wedge A \in V_{i-1}\};$
dokud platí $V_i \neq V_{i-1}$;
$G' = (N \cap V_i, T \cap v_i, P \cap (V_i \times V_i^*), S);$

Funkci algoritmu opět ilustrujeme na příkladě.

Příklad 6.6

P

Mějme gramatiku $G = (\{S, A, B, C, D\}, \{a, b, c, d\}, P, S)$ s množinou pravidel:

$S \to aB$
$A \to bS|dDa$
$B \to bSa|Cb$
$C \to Bc|cSC|c$
$D \to aAd|d$

Při sestavování množin V_i budeme postupovat přesně podle algoritmu.

1. inicializace – $i = 0;\ V_0 = \{S\};$
2. první průchod cyklem – $i = 1;$

   $V_1 = \{X \mid \exists A : A \to \alpha X \beta \wedge A \in V_0\} = \{S, a, B\}$

   Procházíme tentokrát levé strany pravidel a hledáme takovou, která obsahuje nějaký neterminál z množiny V_i. Pokud nalezneme pravidlo s takovou levou stranou, symboly tvořící pravou stranu přidáme do množiny V_1 a to jak terminály tak neterminály;
3. test na rovnost – $V_1 \neq V_0$ – pokračujeme v cyklu;
4. druhý průchod cyklem – $i = 2;$

   $V_2 = \{S, a, B\} \cup \{X \mid \exists A : A \to \alpha X \beta \wedge A \in \{S, a, B\}\} = \{S, a, B, b, C\}$

Množinu V_2 tvoří symboly z pravých stran těch pravidel, které mají na levé straně neterminály S a B.

5. test na rovnost – $V_2 \neq V_1$ – pokračujeme v cyklu;
6. druhý průchod cyklem – $i = 3$;

   $V_3 = \{S, B, C, a, b\} \cup \{X \mid \exists A : A \to \alpha X \beta \wedge A \in \{S, B, C\}\} =$
   $= \{S, B, C, a, b, c\}$

   Množinu V_3 tvoří symboly z pravých stran těch pravidel, které mají na levé straně neterminály S, B a C.
7. test na rovnost – $V_3 \neq V_2$ – pokračujeme cyklu;
8. třetí průchod cyklem – $i = 4$;

   $V_4 = \{S, B, C, a, b, c\} \cup \{X \mid \exists A : A \to \alpha X \beta \wedge A \in \{S, B, C\}\} =$
   $= \{S, B, C, a, b, c\}$
9. test na rovnost – $V_4 = V_3$ – ukončení cyklu;
10. sestavení gramatiky – výsledná gramatika obsahuje pouze takové symboly, které jsou v množině V_4 a taková pravidla, na která „máme stavební materiál".

    $G' = (\{S, B, C\}, \{a, b, c\}, P', S)$, kde množina pravidel obsahuje pravidla:

$$\begin{array}{lcl} S & \to & aB \\ B & \to & bSa \mid Cb \\ C & \to & Bc \mid cSC \mid c \end{array}$$

Věta 6.4 V

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

Z gramatiky G nejdříve podle lemmatu 6.1 odstraníme neterminály, ze kterých nelze odvodit terminální slovo. Získáme tak gramatiku G'. Z gramatiky G' podle lemmatu 6.3 odstraníme nedosažitelné symboly. Tvrdíme, že takto získaná gramatika G'' je redukovaná.

Každý neterminál $X \in N''$ je v gramatice G' dosažitelný v nějakém odvození $A \Rightarrow \alpha X \beta$. Toto odvození je také odvozením v G'', protože jsme odstranili jen nedosažitelné neterminály, které se v odvození nevyskytují. Pak X je v gramatice G'' dosažitelný.

Analogicky libovolný neterminál $A \in N''$ patří i do N'. To znamená, že v G pro něj existovalo odvození terminálního slova. Toto odvození je podle lemmatu 6.1 i odvozením v G'. Pak ze stejného důvodu, jako v předchozím odstavci je to i odvození v G''. Proto i z A lze odvodit v G'' terminální slovo. □

Nyní již víme, že ke každé bezkontextové gramatice existuje gramatika, která neobsahuje nadbytečné a nedosažitelné symboly. velmi často se stává, že při návrhu gramatiky sledujeme určité myšlenkové postupy a pravidla, která vytvoříme jsou sice bezkontextová, ale gramatika nesplňuje podmínku použití ε-pravidel. zamysleme se nyní nad tím, jak se projeví použití ε-pravidel v odvození slova.

Uvažuje například gramatiku s pravidly $S \to Ab \mid AbA$, $A \to a \mid \varepsilon$. Odvození slova může vypadat následujícím způsobem:

$$S \Rightarrow AbA \Rightarrow bA \Rightarrow b \text{ nebo } S \Rightarrow AbA \Rightarrow Ab \Rightarrow b$$

Ve druhém a třetím kroku odvození je použito pravidlo $A \Rightarrow \varepsilon$. Pokud bychom chtěli odstranit toto pravidlo z gramatiky, musíme zabezpečit, že odvození věty bA, Ab a slova b bude v gramatice stále možné. Proto doplníme gramatiku o pravidla $S \Rightarrow bA$, $S \Rightarrow Ab$ a $S \Rightarrow b$. Srovnáním prvního použitého pravidla a nových tří pravidel dostáváme návod, jak upravit pravidla gramatiky, kterou zbavujeme ε-pravidel. Z pravé strany pravidel postupně vypouštíme všechny výskyty neterminálů, které mohou „zmizet“ použitím ε-pravidel.

Nejdříve definujeme pojem nevypouštějící gramatiky.

Definice 6.6 **Def**

Bezkontextová gramatika se nazývá nevypouštějící (ε-free), jestliže neobsahuje žádné pravidlo typu $A \to \varepsilon$ neboli žádné ε-pravidlo (s výjimkou pro pravidlo $S \to \varepsilon$).

Následuje algoritmus pro odstranění ε-pravidel.

Algoritmus 6.4 Odstranění ε-pravidel

A

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$
Výstup: Bezkontextová gramatika $G' = (N', T, P', S')$ taková, že $L(G') = L(G)$

$i = 0; E_0 = \varnothing;$
dělej
  $i = i + 1;$
  $E_i = E_{i-1} \cup \{A \mid A \to \alpha \in P; \alpha \in E_{i-1}^*\};$
dokud platí $E_i \neq E_{i-1}$;
$E = E_i;$
pro všechna pravidla $A \to \alpha_1 \ldots \alpha_n$ dělej:
  přidej do P' všechna pravidla ve tvaru $A \to \beta_1 \ldots \beta_n$ odvozená
  z $A \to \alpha_1 \ldots \alpha_n$, která splňují následující podmínky:
    1) pokud $\alpha_j \notin E$, pak $\beta_j = \alpha_j$;
    2) pokud $\alpha_j \in E$, pak $\beta_j \in \{\alpha_j, \varepsilon\}$;
    3) $\beta_j \neq \varepsilon$ alespoň pro jedno $j, 1 \leq j \leq n$;
jestliže $S \in E$
  pak přidej do P' pravidla $S' \to S \mid \varepsilon$ a $N' = N \cup \{S'\}$;
  jinak $N' = N;\ S' = S;$

V první fázi provádění algoritmu vyberete všechny neterminály, které lze v rámci odvození přepsat na ε, po té projdete všechna pravidla původní gramatiky a pokud jejich pravá strana obsahuje vybraný neterminál, vložíte taková pravidla, jejichž pravé strany tento neterminál obsahují i neobsahují. Pokud je takových neterminálů na pravé straně pravidla více, je potřeba obsáhnout všechny kombinace vypouštěných neterminálů. Výjimku tvoří taková kombinace, kdybychom vypustili všechny neterminály a vzniklo ε-pravidlo. (Ne vždy musí vypuštěním všech neterminálů vzniknout ε-pravidlo!)

Jak probíhá aplikace algoritmu si opět ukážeme na příkladu.

Příklad 6.7 **P**

Upravte bezkontextovou gramatiku $G = (N, T, P, S)$ tak, aby neobsahovala ε-pravidla, když:

1. $N = \{S, A, B, C\}$;
2. $T = \{a, b, c\}$;
3. $P =$ $S \to aA \mid aAB \mid AAC$
   $A \to aB \mid CaA \mid CC$
   $B \to aS \mid bB \mid AB$
   $C \to c \mid \varepsilon$

Nejdříve sestrojíme množinu E. Začneme tím, že zvolíme $E_0 = \varnothing$. V prvním kroku přidáme do množiny E_1 všechny neterminály, které se objevují na levé straně ε-pravidel. Dostáváme tedy $E_1 = \{C\}$. V dalších krocích přidáváme do množiny E_i takové neterminály, které jsou na levé straně takových pravidel, jejichž pravá strana obsahuje pouze prvky množiny E_{i-1}.

$E_0 = \varnothing$
$E_1 = \{C\}$
$E_2 = \{C, A\}$
$E_3 = \{C, A, S\}$
$E_4 = \{C, A, S\}$
$E_4 = E_3 = E$

Tímto jsme získali seznam neterminálů, které lze po konečném počtu kroků přepsat na prázdné slovo ε.

Nyní pro každé pravidlo určíme, jestli obsahuje na pravé straně pravidla neterminály z množiny E. Jestliže ano, přidáme do množiny pravidel taková pravidla, která vzniknou vypuštěním některého z těchto neterminálů. Vynecháme všechna ε-pravidla.

$P' =$ $S \to aA \mid a$
$S \to aAB \mid aB$
$S \to AAC \mid AC \mid AA \mid A \mid C$
$A \to aB$
$A \to CaA \mid aA \mid Ca \mid a$
$A \to CC \mid C$
$B \to aS \mid a$
$B \to bB$
$B \to AB \mid B$
$C \to c$

Pokud do množiny E patří i startovací neterminál, vytvoříme nový startovací neterminál S' a přidáme do množiny pravidel $S' \to \varepsilon$ a $S' \to S$.

$$\begin{aligned} P' = P' \cup \; & S' \to \varepsilon \\ & S' \to S \end{aligned}$$

Pak dostáváme gramatiku $G' = (N', T, P', S')$ bez ε-pravidel, pro kterou platí $L(G) = L(G')$ a dále:

1. $N' = \{S', S, A, B, C\}$;
2. $T = \{a, b, c\}$;
3. $P' = \begin{array}{lll} S' & \to & S \mid \varepsilon \\ S & \to & aA \mid a \mid aAB \mid aB \mid AAC \mid AC \mid AA \mid A \mid C \\ A & \to & aB \mid CaA \mid aA \mid Ca \mid a \mid CC \mid C \\ B & \to & aS \mid a \mid bB \mid AB \mid B \\ C & \to & c \end{array}$

Přejděme nyní k poslední úpravě bezkontextových gramatik. Jedná se o odstranění takzvaných jednoduchých pravidel. Jsou to pravidla typu $A \to B$, kde A a B jsou neterminály. Proč se zbavovat jednoduchých pravidel? Ve své podstatě totiž nepřináší žádnou přidanou hodnotu. Provedením pravidla $A \to B$ dojde k záměně neterminálu A za neterminál B, nevyprodukuje se žádný symbol „navíc“.

Algoritmus 6.5 Eliminace jednoduchých pravidel CF gramatiky

A

Vstup: Bezkontextová gramatika bez ε-pravidel $G = (N, T, P, S)$
Výstup: Bezkontextová gramatika $G' = (N, T, P', S)$ bez jednoduchých pravidel

pro každé $A \in N$ dělej:
  $i = 0; N_0 = \{A\}$;
  dělej
    $i = i + 1$;
    $N_i = N_{i-1} \cup \{C \mid B \to C \in P; B \in N_{i-1}\}$;
  dokud platí $N_i \neq N_{i-1}$;
  $N_A = N_i$;
konec dělej pro každé
pro každé $A \in N$ dělej:
  přidej do P' všechna nejednoduchá pravidla $A \to \alpha$ taková, že $B \to \alpha \in P \wedge B \in N_A$;
konec dělej pro každé;

Princip algoritmu odstranění jednoduchých pravidel je poměrně jednoduchý. Nejprve pro každý neterminál vytvoříme množinu neterminálů N_A, na které lze použitím jednoduchých pravidel tento neterminál přepsat. Pak nahradíme jednoduchá pravidla (s neterminálem A na levé straně) pravidly, která mají na levé straně zkoumaný neterminál A a na pravé straně všechny pravé strany pravidel neterminálů, které patří do vytvořené množiny N_A.

Realizaci algoritmu představíme na jednoduchém příkladě:

Příklad 6.8 **P**

Upravte bezkontextovou gramatiku $G = (N, T, P, S)$ tak, aby neobsahovala jednoduchá pravidla, když:

1. $N = \{S, A, B, C\}$;
2. $T = \{a, b, c\}$;
3. $P = \begin{array}{lcl} S & \to & aA \mid aAB \mid C \\ A & \to & aB \mid CaA \mid CC \\ B & \to & aS \mid bB \mid A \\ C & \to & c \mid A \end{array}$

Nejdříve sestrojíme ke každému neterminálu A jeho množinu N_A. Začneme neterminálem S a vytvářením množiny N_S.

V prvním kroku položíme $N_0 = \{S\}$. Ve druhém kroku přidáme do množiny N_1 všechny neterminály, které se objevují na pravé straně jednoduchých pravidel pro startovací neterminál (mají S na levé straně). Dostáváme tedy $N_1 = \{S, C\}$. V dalších krocích přidáváme do množiny N_i další neterminály, které jsou na pravé straně takových jednoduchých pravidel, které mají na levé straně neterminály z množiny N_{i-1}. Pokračujeme tak dlouho, dokud dvě za sebou vytvořené množiny nejsou ekvivalentní.

$$\begin{array}{lcl} N_0 & = & \{S\} \\ N_1 & = & \{S, C\} \\ N_2 & = & \{S, C, A\} \\ N_3 & = & \{S, C, A\} \\ N_3 & = & N_2 = N_S \end{array}$$

Tímto způsobem vytvoříme i další množiny neterminálů.

$$\begin{array}{lcl} N_0 & = & \{A\} \\ N_1 & = & \{A\} \\ N_1 & = & N_0 = N_A \end{array} \qquad \begin{array}{lcl} N_0 & = & \{B\} \\ N_1 & = & \{B, A\} \\ N_2 & = & \{B, A\} \\ N_2 & = & N_1 = N_B \end{array} \qquad \begin{array}{lcl} N_0 & = & \{C\} \\ N_1 & = & \{C, A\} \\ N_2 & = & \{C, A\} \\ N_2 & = & N_1 = N_C \end{array}$$

Nyní začneme sestavovat novou množinu pravidel. Vyjdeme z množiny pravidel P, ze které odstraníme všechna jednoduchá pravidla.

$$\begin{array}{lcl} S & \to & SaA \mid aAB \\ A & \to & aB \mid CaA \mid CC \\ B & \to & aS \mid bB \\ C & \to & c \end{array} \qquad P' = \begin{array}{lcl} S & \to & SaA \mid aAB \mid c \mid aB \mid CaA \mid CC \\ A & \to & aB \mid CaA \mid CC \\ B & \to & aS \mid bB \mid aB \mid CaA \mid CC \\ C & \to & c \mid aB \mid CaA \mid CC \end{array}$$

Tím je vytvořena množina pravidel nové gramatiky.

Úkoly

1. Vytvořte redukovanou gramatiku k bezkontextové gramatice $G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$ s množinou pravidel

$$\begin{array}{rcl} P = \{S & \to & aSb \mid aAbb \mid \varepsilon \\ A & \to & aAB \mid bB \\ B & \to & aAb \mid BDB \\ C & \to & CC \mid DcS \\ D & \to & aba \mid cS\} \end{array}$$

2. Vytvořte redukovanou gramatiku k bezkontextové gramatice $G = (\{S, A, B, C\}, \{a, b, c, d\}, P, S)$ s množinou pravidel

$$\begin{array}{rcl} P = \{S & \to & ABc \mid AA \mid BB \\ A & \to & aB \mid aA \mid aa \\ B & \to & bB \\ C & \to & dB \mid dd\} \end{array}$$

3. Vytvořte redukovanou gramatiku k bezkontextové gramatice $G = (\{S, A, B, C\}, \{a, b\}, P, S)$ s množinou pravidel

$$\begin{array}{rcl} P = \{S & \to & AB \mid CA \mid \varepsilon \\ A & \to & a \\ B & \to & BC \mid AB \\ C & \to & aB \mid b\} \end{array}$$

4. Vytvořte nevypouštějící gramatiku k bezkontextové gramatice $G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$ s množinou pravidel

$$\begin{aligned} P = \{S &\rightarrow AB \mid \varepsilon \\ A &\rightarrow aAAb \mid SB \mid CA \\ B &\rightarrow BbA \mid CaC \mid \varepsilon \\ C &\rightarrow aBB \mid bS\} \end{aligned}$$

5. Vytvořte nevypouštějící gramatiku k bezkontextové gramatice $G = (\{S, A, B\}, \{a, b\}, P, S)$ s množinou pravidel

$$\begin{aligned} P = \{S &\rightarrow AAa \mid BBb \\ A &\rightarrow aaB \mid AA \mid \varepsilon \\ B &\rightarrow bbA \mid BB \mid \varepsilon\} \end{aligned}$$

6. Vytvořte nevypouštějící gramatiku k bezkontextové gramatice $G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$ s množinou pravidel

$$\begin{aligned} P = \{S &\rightarrow ABC \\ A &\rightarrow Ab \mid BC \mid CA \\ B &\rightarrow bB \mid b \mid Ab \mid \varepsilon \\ C &\rightarrow cC \mid c \mid Ac \mid \varepsilon\} \end{aligned}$$

6.3 Normální tvary bezkontextových gramatik

Protože definice bezkontextových pravidel neklade žádná omezení na pravou stranu pravidel (kromě omezení pro ε), mohou se gramatiky generující jeden a týž jazyk poměrně značně nejen v počtu pravidel, ale také v délce pravých stran. Jak porovnat takové gramatiky? Jak porovnat jazyky z hlediska velikosti gramatik, které jej generují? Je možné nalézt omezení pro pravou stranu pravidel takové, které by nezúžilo množinu jazyků, kterou jsou schopny vygenerovat gramatiky s takto omezenými pravidly? Odpověď na tyto otázky je založená na nalezení tzv. normálních tvarů gramatik. Každou gramatiku lze na takový tvar převést. Normální tvary pak také slouží jako výchozí předpoklad pro vedení důkazů, odrazový můstek pro programování algoritmů, které využívají existenci bezkontextových gramatik.

V tomto textu se zmíníme o dvou normálních tvarech: Chomského a Greibachové normálním tvaru.

Definice 6.7

Def

Chomského normální tvar

Řekneme, že bezkontextová gramatika $G = (N, T, P, S)$ je v Chomského normálním tvaru (Chomsky normal form – CNF), pokud každé pravidlo z P má jeden z těchto tvarů:

1. $A \to BC$,
2. $A \to a$, kde $A, B, C \in N, a \in T$, s výjimkou pro pravidlo $S \to \varepsilon$ diskutovanou v části 3.2.

Všimněme si, jak vypadá derivační strom odvození slova v gramatice, která je v CNF. Každému odvození v gramatice v CNF odpovídá binární strom odvození – buď použijeme pravidlo typu $A \to BC$ a dostaneme dva podstromy, nebo pravidlem $A \to a$ větev ukončíme.

Jak přetransformujeme bezkontextovou gramatiku na gramatiku v CNF je uvedeno v následujícím algoritmu.

Algoritmus 6.6 Transformace CF gramatiky na CNF

A

Vstup: Bezkontextová gramatika bez jednoduchých pravidel $G = (N, T, P, S)$
Výstup: Bezkontextová gramatika $G' = (N', T, P', S)$ v CNF

$N' = N$;
přiřaď do množiny P' všechna pravidla typu $A \to a$, kde $a \in T, A \in N$;
pro každé pravidlo $A \in \alpha$, kde $|\alpha| \geq 2$ dělej:
  pro všechna $a \in T$, která jsou obsažena v α;
    nahraď všechny výskyty a za $\boxed{a}$;
    přiřaď $\boxed{a}$ do množiny N';
  pro pravidlo $A \to N_1 \dots N_n$ dělej:
    $X = A,\ \beta = N_1 \dots N_2$;
    dokud je $|\beta| > 2$
      pravidlo $X \to N_i \boxed{N_{i+1} \dots N_n}$ přidej do P';
      $\boxed{N_{i+1} \dots N_n}$ přidej do N';
      $X = \boxed{N_{i+1} \dots N_n}$;
      $\beta = N_{i+1} \dots N_n$;
    konec dokud
    pravidlo $X \to \beta$ přidej do P';
pro všechny neterminály $\boxed{a} \in N'$ takové, že $a \in T$
  přidej do P' pravidlo $\boxed{a} \to a$.

Funkce algoritmu je poměrně jednoduchá. Z bezkontextové gramatiky bez jednoduchých pravidel vytvoříme novou bezkontextovou gramatiku, která bude v Chomského normálním tvaru a to na základě faktu, že pokud pravá strana pravidla má délku jedna (na pravé straně je jen jeden symbol), pak to musí být terminál. Tato pravidla a eventuálně ε-pravidlo přidáme do množiny pravidel nové gramatiky beze změny. Ostatní pravidla (z délkou pravé strany větší než jedna) mají na pravé straně větnou formu. V této větné formě provedeme nahrazení všech terminálních symbolů za „zabalené“ terminály, které jsou novými neterminálními symboly. Tato pravidla mají po úpravě pravou stranu složenou ze samých neterminálů. Další krok pro úpravu pravidel spočívá v „balíčkování“ neterminálů tak, aby pravá strana byla tvořena vždy dvěma neterminály. Pokud je délka pravé strany

větší než dvě, zabalíme všechny neterminály kromě prvního. Vznikne tak nový neterminál, ke kterému vytvoříme „odbalovací“ pravidlo. Pokud má obsah balíčku délku větší než dvě, postup opakujeme. Pokud je řetězec, který rozbalujeme, délky dvě, můžeme pravidlo přidat do množiny pravidel beze změny. Tímto způsobem pokračujeme, dokud nezpracujeme všechna pravidla původní gramatiky.

Nakonec přidáme pravidla, která slouží k rozbalení terminálních symbolů, to je pravidla typu $\boxed{a} \to a$.

Opět ilustrujeme funkci algoritmu na příkladě.

Příklad 6.9 **P**

Upravte bezkontextovou gramatiku $G = (N, T, P, S)$ na Chomského normální tvar, když platí:

1. $N = \{S, A, B, C\}$;
2. $T = \{a, b, c\}$;
3. $P =$ $S \to aAB \mid \varepsilon$
   $A \to CaAb \mid CC$
   $B \to bB \mid aba$
   $C \to c \mid cAcc$

Převedeme tedy gramatiku G, která je bezkontextová, bez jednoduchých pravidel, na gramatiku $G' = (N', T, P', S)$ v CNF. Vidíme, že pravidlo $S \to \varepsilon$ a $C \to c$ můžeme přiřadit do množiny pravidel gramatiky P'. Nyní budeme brát jedno pravidlo po druhém a aplikovat na ně postup z algoritmu.

Začneme s pravidlem $S \to aAB$. Nejdříve „zabalíme“ terminál a. Dostáváme upravené pravidlo $S \to \boxed{a}AB$, které budeme dále „balíčkovat“ a vytvářet nová pravidla.

původní pravidlo	S	$\to$	aAB
nová pravidla	S	$\to$	$\boxed{a}\,\boxed{AB}$
	$\boxed{AB}$	$\to$	AB

Podobně pokračujeme pro další pravidla:

původní pravidlo	$A \to CaAb$
nová pravidla	$A \to C\boxed{a}A\boxed{b}$
	$\boxed{a}A\boxed{b} \to a\boxed{A}\boxed{b}$
	$\boxed{A}\boxed{b} \to A\boxed{b}$

Pravidlo $A \to CC$ již v požadovaném tvaru je.

původní pravidlo	$B \to bB$
nové pravidlo	$A \to \boxed{b}B\boxed{b}$

původní pravidlo	$B \to aba$
nová pravidla	$B \to a\boxed{b}\boxed{a}$
	$\boxed{b}\boxed{a} \to b\boxed{a}$
nové pravidlo	$A \to \boxed{b}B\boxed{b}$

původní pravidlo	$C \to cAcc$
nová pravidla	$C \to c\boxed{A}\boxed{c}\boxed{c}$
	$A\boxed{c}\boxed{c} \to A\boxed{c}\boxed{c}$
	$\boxed{c}\boxed{c} \to c\boxed{c}$

V poslední fázi transformace přidáme pravidla pro „rozbalení“ terminálů.

$\boxed{a} \to a, \qquad \boxed{b} \to b, \qquad \boxed{c} \to c$

Tímto krokem jsme dokončili transformaci bezkontextové gramatiky na Chomského normální tvar.

Druhým zmiňovaným normálním tvarem je Greibachové normální tvar.

Definice 6.8

Def

Řekneme, že bezkontextová gramatika $G = (N, T, P, S)$ je v Greibachové normálním tvaru (Greigach normal form – GNF), pokud každé pravidlo z P má tvar :

Greibachové normální tvar

1. $A \to aB_1B_2 \dots B_n; n \geq 0, A, B_1, B_2, \dots, B_n \in N, a \in T$,
2. $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla.

Uvedeme příklady bezkontextových gramatik, u kterých určíme, jestli jsou v Greibachové normálním tvaru.

Příklad 6.10

P

$$
\begin{array}{lcl}
G_1 & = & (N_1, T_1, P_1, S) \\
N_1 & = & \{S, A, B\} \\
T_1 & = & \{a, b, c\} \\
P_1 & = & \{S \to aBC \\
 & & \ A \to bBB \mid b \\
 & & \ B \to cS \mid c \ \}
\end{array}
$$

Všechny pravé strany pravdilel gramatiky G_1 začínají terminálním symbolem. Navíc je tento terminální symbol na pravé straně jediný. Gramatika G_1 je tedy v GNF.

$$
\begin{array}{lcl}
G_2 & = & (N_2, T_2, P_2, E) \\
N_2 & = & \{E, F, G\} \\
T_2 & = & \{(,), i\} \\
P_2 & = & \{E \to (\ F \mid i \mid \varepsilon \\
 & & \ F \to iG \\
 & & \ G \to) \mid iFG \ \}
\end{array}
$$

Gramatika G_2 obsahuje ε-pravidlo, které splňuje kritéria stanovené pro bezkontextovou gramatiku (pro startovací symbol a tento symbol se pak nevyskytuje na pravé straně žádného z pravidel). Protože pro všechna ostatní pravidla platí to, co bylo napsáno ke gramatice G_2, je i gramatika G_2 v GNF.

$$
\begin{array}{rcll}
G_3 & = & (N_3, T_3, P_3, K) & \text{Gramatika } G_3 \text{ je regulární gra-} \\
N_3 & = & \{K, L\} & \text{matika. Reulární pravidla splňují} \\
T_3 & = & \{0, 1, 2, 3\} & \text{všechny požadavky na to, aby gra-} \\
P_3 & = & \{K \rightarrow 0K \mid 1L & \text{matiky mohla být v GNF} \\
 & & L \rightarrow 3K \mid 2\ \} &
\end{array}
$$

6.4 Pumping lemma pro bezkontextové jazyky

Pumping lemma pro bezkontextové jazyky bude podobné obdobnému lemmatu pro regulární jazyky (viz sekce 4.4). Toto lemma udává nutnou a nikoli postačující podmínku toho, aby jazyk byl bezkontextový. Proto dokazujeme, že když daný jazyk nemá tuto vlastnost, pak nemůže být bezkontextový).

Věta 6.5

V

Lemma o vkládání. Nechť L je bezkontextový jazyk. Pak existují $p, q \in N$ taková, že libovolné slovo $w \in L$, jehož délka je alespoň p, lze psát ve tvaru $w = uvwxy$, kde

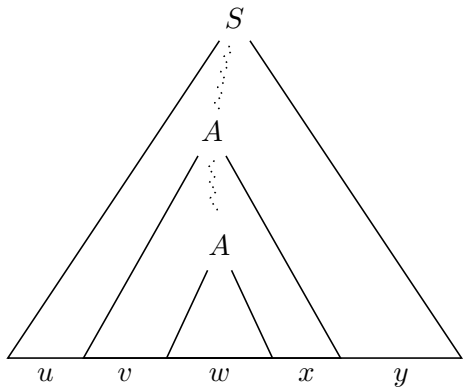
1. $|vwx| \leq q$,
2. $|vx| \geq 0$ a
3. $uv^i wx^i y \in L$ pro každé $i \in N_0$.

Jinými slovy: Každé dostatečně dlouhé slovo bezkontextového jazyka lze rozdělit na pět částí. První podmínka omezuje délku prostředních částí slova. Podle druhé podmínky musí mít alespoň jedna „pumpovaná“ část nenulovou délku. Třetí podmínka se vztahuje opět k pumpování – jedná se o to, že pokud části slova v a x zopakujeme (i vícekrát), dostaneme nové slovo, které opět bude patřit do jazyka. Protože číslo i může mít i hodnotu nula, bude do jazyka L patřit i slovo, které vznikne vypuštěním těchto částí.

Uvedeme ideu důkazu lemmatu o vkládání.

Idea důkazu: Jelikož L je bezkontextový, existuje bezkontextová gramatika v Chomského normálním tvaru $G = (N, T, P, S)$ generující jazyk L. Položme

$n = |P|$, $p = 2^{n-1}, q = 2^n$. Pokud vezmeme libovolné slovo jazyka, které má délku větší než p, Pak v libovolném derivačním stromu pro odvození tohoto slova existuje cesta ze startovacího symbolu délky větší než n. Na této cestě musel být jeden neterminál přepsán alespoň dvakrát (protože v gramatice je jen n neterminálů, ale na cestě délky n je jich použito $n+1$). Na obrázku 6.2 je zobrazen strom odvození slova ω . Zvýrazněny jsou opakující se neterminály A a jejich souvislost s jednotlivými podslovy.



Obrázek 6.2: Strom odvození slova $uvwxy$

Posloupnost aplikací pravidel, která vede k tomu, že z neterminálu A odvodíme větnou formu, která obsahuje ten stejný neterminál, lze znovu zopakovat nebo vypustit (viz obrázek 6.3).

Opakováním posloupnosti aplikovaných pravidel mezi prvním a druhým výskytem neterminálu A dostáváme poslední tvrzení věty:

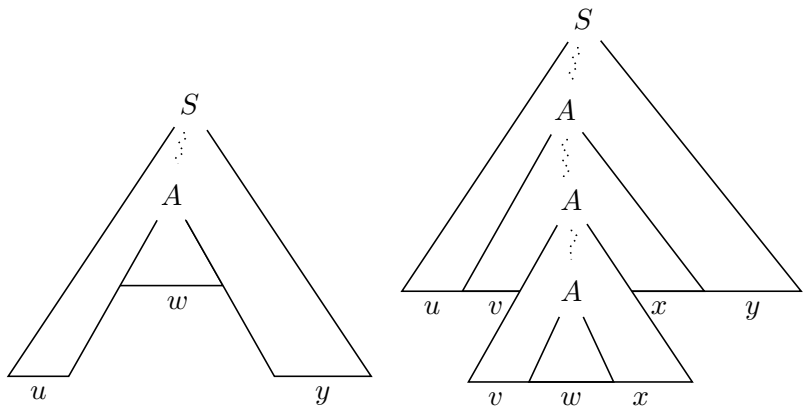
$$uv^i wx^i y \in L, i \geq 0$$

□

Pumping lemma pro bezkontextové jazyky se používá při důkazech, že zadaný jazyk není bezkontextový. Uvedeme příklad použití pumping lemmatu.

Příklad 6.11 P

Dokážeme, že jazyk $L = \{a^{n^2} \mid n \geq 0\}$ není bezkontextový. Zvolíme slovo $w = a^{p^2} \in L$ s „dostatečně“ velkým indexem p. Toto slovo rozdělíme násle-



Obrázek 6.3: Stromy odvození slov uwy a $uvvwxxy$

dovně: $a^{p^2} = a^{x_1}a^{x_2}a^{x_3}a^{x_4}a^{x_5}$ a platí $x_2 + x_4 > 0,\ x_2 + x_3 + x_4 \leq q$. Dále platí $p^2 = x_1 + x_2 + x_3 + x_4 + x_5$. Pokud vytvoříme iterované slovo $\omega = uv^iwx^iy$ dostáváme $\omega = a^{x_1}a^{i\cdot x_2}a^{x_3}a^{i\cdot x_4}a^{x_5}$ a pro horní indexy rovnici:

$$(p+j)^2 = x_1 + i\cdot x_2 + x_3 + i\cdot x_4 + x_5 = i(x_2 + x_4) + x_1 + x_3 + x_5,\ i,j \geq 0$$

Získali jsme rovnici, kde na levé straně je kvadratická funkce, kdežto na pravé straně je funkce lineární. Ať stanovíme indexy x_2 a x_4 jakkoliv, vždy najdeme číslo i, pro které součet indexů $i(x_2 + x_4) + x_1 + x_3 + x_5$ není druhou mocninou žádného přirozeného čísla. Z toho tedy vyplývá, že L není bezkontextový jazyk.

6.5 Bezkontextové gramatiky a zásobníkové automaty

V této části uvedeme věty, které hovoří o vztahu bezkontextových jazyků a jazyků akceptovaných zásobníkovými automaty.

Věta 6.6 V

Nechť G je libovolná bezkontextová gramatika. Pak lze sestrojit zásobníkový automat M takový, že $L(G) = L(M)$.

Idea důkazu: Ke gramatice G zkonstruujeme (jednostavový) zásobníkový automat tak, aby ve svém zásobníku simuloval levé derivace v G.

Zásobníkový automat začne svůj výpočet se vstupním slovem ω a startovacím symbolem S jako počátečním obsahem zásobníku.

Stejně jako je v gramatice G v jednom kroku levého odvození nahrazen neterminál A pravou stranou pravidla $A \to \alpha$, tak zásobníkový automat nahradí symbol (korespondující s neterminálem gramatiky) na vrcholu zásobníku řetězcem α.

Pokud se na vrcholu zásobníku nachází symbol odpovídající terminálnímu symbolu, provede automat kontrolu, zda se odpovídající terminál nachází i na vstupní pásce. Protože zásobníkový automat simuluje levé odvození, objevují se terminální symboly gramatiky na vrcholu zásobníku ve stejném pořadí, v jakém by měly být v akceptovaném vstupním slově.

Výpočet zásobníkového automatu M končí v akceptujícím případě vyprázdněným zásobníkem a dočteným vstupním slovem – simulace odvození skončila vygenerováním terminálních symbolů a všechny terminální symboly zapsané do zásobníku byly vymazány.

Přejděme nyní ke konstrukci takového zásobníkového automatu. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Sestrojíme zásobníkový automat $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$, kde:

- $Q = \{q\}$; $\Sigma = T$; $\Gamma = N \cup T$;
- $q_0 = q$; $Z_0 = S$;
- a přechodové zobrazení definujeme takto:

$$\delta(q, \varepsilon, A) = \{(q, \alpha) \mid A \to \alpha \in P\}$$
$$\delta(q, a, a) = \{(q, \varepsilon)\}; \forall a \in T$$

□

Uvedeme nyní příklad transformace bezkontextové gramatiky G na zásobníkový automat M, takový, že $L(M) = L(G)$.

Příklad 6.12 P

K bezkontextové gramatice $G = (\{S\}, \{0,1\}, P, S)$, kde: $P = \{S \to 0S0 \mid 1S1 \mid 1\}$ sestavte zásobníkový automat, který bude akceptovat jazyk $L(G)$.

Sestrojíme zásobníkový automat $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varnothing)$, kde:

- $Q = \{q\}$;
- $\Sigma = \{0,1\}$;
- $\Gamma = \{S,0,1\}$;
- $q_0 = q$;
- $Z_0 = S$;
- a přechodové zobrazení vypadá takto:

pravidlo $S \to 0S0$
$\delta(q,\varepsilon,S) = \{(q,0S0)\}$

pravidlo $S \to 1S1$
$\delta(q,\varepsilon,S) = \{(q,1S1)\}$

pravidlo $S \to 1$
$\delta(q,\varepsilon,S) = \{(q,1)\}$

pro terminály
$\delta(q,0,0) = \{(q,\varepsilon)\}$
$\delta(q,1,1) = \{(q,\varepsilon)\}$

Uvedeme zde příklad odvození slova 0101010 v gramatice G a odpovídající výpočet zásobníkového automatu M.

G: $S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 010S010 \Rightarrow 0101010$

Barevně odlišíme znak $\vdash$ pro takový krok odvození, kdy proběhlo čtení ze vstupní pásky.

M: $(q,0101010,S) \vdash (q,0101010,0S0) \vdash (q,101010,S0) \vdash$
$\vdash (q,101010,1S10) \vdash (q,01010,S10) \vdash (q,01010,0S010) \vdash$
$\vdash (q,1010,S010) \vdash (q,1010,1010) \vdash (q,010,010) \vdash$
$\vdash (q,10,10) \vdash (q,0,0) \vdash (q,\varepsilon,\varepsilon)$

Obrácenou větu, to znamená, že ke každému zásobníkovému automatu M existuje bezkontextová gramatika taková, že generuje jazyk $L(M)$ zde uvedeme bez dalšího komentáře. Důkaz věty i konstrukce takové gramatiky přesahuje rozsah tohoto textu. Čtenář může nalézt více například v [1].

Věta 6.7 V

Ke každému zásobníkovému automatu M existuje bezkontextová gramatika G taková, že $L(M) = L(G)$.

Úkoly

1. Vytvořte zásobníkový automat, který akceptuje jazyk generovaný gramatikou $G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$ s množinou pravidel

$$\begin{aligned} P = \{S &\rightarrow aSb \mid aAbb \mid \varepsilon \\ A &\rightarrow aAB \mid bB \\ B &\rightarrow aAb \mid BDB \\ C &\rightarrow CC \mid DcA \\ D &\rightarrow aba \mid cD\} \end{aligned}$$

2. Vytvořte zásobníkový automat, který akceptuje jazyk generovaný gramatikou $G = (\{S, A, B, C\}, \{a, b, c, d\}, P, S)$ s množinou pravidel

$$\begin{aligned} P = \{S &\rightarrow ABc \mid ASA \mid BB \\ A &\rightarrow aB \mid aA \mid aa \\ B &\rightarrow bB \\ C &\rightarrow dB \mid dd\} \end{aligned}$$

3. Vytvořte zásobníkový automat, který akceptuje jazyk generovaný gramatikou $G = (\{S, A, B, C\}, \{a, b\}, P, S)$ s množinou pravidel

$$\begin{aligned} P = \{S &\rightarrow AB \mid CA \mid \varepsilon \\ A &\rightarrow a \\ B &\rightarrow BC \mid AB \\ C &\rightarrow aB \mid b\} \end{aligned}$$

Kapitola 7

Úvod do teorie vyčíslitelnosti

Vyčíslitelnost je oblast teoretické informatiky zabývající se zkoumáním, které problémy lze řešit pomocí algoritmů a které ne.

V předchozí větě se objevili dva pojmy, algoritmus a problém. Podívejme se nejdříve na to, co vlastně znamenají.

7.1 Algoritmus, problém

Na otázku, co je to algoritmus jsme již odpovídali na začátku tohoto textu. Připomeňme si, že pokud chceme tento pojem nějak podrobněji popsat, používáme slova a slovní spojení jako „postup“, „návod“, „posloupnost kroků“ a podobně. Požadujeme také, aby takový postup bylo možné provádět opakovaně a aby jeho jednotlivé kroky byly konečné. Posledním kritériem je, aby stejné vstupní údaje vedly opakovaně ke stejným výsledkům.

Algoritmus

Pojem algoritmus je základním pojmem, který není definován pomocí jednodušších pojmů, je jen objasňován svými vlastnostmi na konkrétních příkladech.

K čemu takové algoritmy slouží? Obecně se dá říci, že algoritmy se používají k řešení problémů.

Slovo problém má v přirozeném jazyce více významů. Na příkladech si přiblížíme ten, o který se zajímáme v teoretické informatice. Příkladem je problém nalézt součet dvou čísel, zjistit, zda je dané číslo prvočíslem, vynásobit dvě matice, setřídit seznam slov podle zvoleného klíče apod.

Konkrétní problém definujeme názvem, vstupem a požadovaným výstupem.

Definice 7.1 **Def**

Problém je určen trojicí (IN, OUT, p), kde IN je množina (přípustných) vstupů, OUT je množina výstupů a $p : IN \to OUT$ je funkce přiřazující každému vstupu odpovídající výstup. Problém

Algoritmy jsou pak navrženy tak, aby řešily takovéto problémy. Co znamená, pokud řekneme, že algoritmus A řeší problém P? Stručně to vyjádříme takto:

Algoritmus A pro každý vstup z množiny IN problému P vypočte a vydá výstup z množiny OUT. Jak vstup tak výstup jsou většinou vhodně zapsány tak, aby jim „stroj“, který bude algoritmus realizovat, „rozumněl“.

I když to tak na první pohled nevypadá (a záplava detektivních seriálů s dokonalou výpočetní technikou nás chce přesvědčit o opaku) existuje mnoho problémů, které nejsou algoritmicky řešitelné.

Můžeme zde zmínit dva druhy problémů. Prvním typem je problém rozhodovací. Jedná se o takový problém jehož výstupem je ANO nebo NE. V zadání takového problému pak většinou figuruje otázka – například *Náleží slovo ω do jazyka L?* Druhým typem, neméně významným, je problém optimalizační. Ke každému vstupu existuje množina výstupů, které lze ohodnotit reálným číslem (na základě kriteriální funkce). Úkolem optimalizačního algoritmu je nalézt takové řešení z množiny výstupů, aby ohodnocení bylo maximální (nebo minimální, záleží na zadání). Příkladem takového problému je *nalezení nejkratší cesty* pro dané dva vrcholy orientovaného grafu.

Rozhodovací problém

Optimalizační problém

7.2 Turingův stroj a Church-Turingova teze

Vraťme se zpět k algoritmům. Definice pojmu „algoritmus“ nijak nespecifikuje, jak má vypadá jeho zápis, jaké instrukce můžeme použít. K popisu algoritmu se tak používá přirozený jazyk, rozhodovací tabulky, obrázky, náčrtky atd. Se vznikem počítačů vyvstala potřeba definovat instrukce takovým

způsobem, aby tyto algoritmy mohly provádět počítače. Vznikly proto programovací jazyky.

Již ve třicátých letech dvacátého století se objevil při snaze prokázat algoritmickou nerozhodnutelnost některých problémů návrh stroje, který používal velmi jednoduché instrukce. Jednalo se o tzv. Turingův stroj. Jedná se o abstraktní stroj, podobně jako tomu bylo u konečného nebo zásobníkového automatu. Základní odlišností Turingova stroje od konečného a zásobníkového automatu je využívání vstupní pásky coby paměťového média. Páska je nekonečně dlouhá a kromě vstupního slova zapsaného před zahájením výpočtu stroje na konečně mnoha políčcích budou všechna ostatní políčka pásky obsahovat tzv. prázdný znak. Čtecí hlava je rovněž i zapisovací a bude moci čtené symboly přepisovat na jiné a poté se posunout na sousední políčko buď vlevo nebo vpravo — podle příkazu přechodové funkce. Další změnou, která vyplývá z možnosti obousměrného pohybu hlavy po pásce, je způsob akceptování slova. Aby Turingův stroj slovo akceptoval, nemusí ho nutně celé přečíst, „stačí“ když přejde (v libovolném okamžiku výpočtu nad daným slovem) do koncového stavu.

Tringův stroj

Než uvedeme formální definici Turingova stroje poznamenejme ještě, že v literatuře je možné nalézt různé varianty Turingových strojů. Liší se nejen strukturou, ale i definicí funkce. Všechny varianty jsou vzájemně ekvivalentní.

Definice 7.2

Def

Turingův stroj

Turingovým strojem (zkráceně TS) nazýváme uspořádanou šestici $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde

- Q je konečná neprázdná množina stavů,
- Σ je vstupní abeceda,
- Γ je konečná neprázdná množina páskových symbolů, pásková abeceda, $\Sigma \subseteq \Gamma$, do $\Gamma - \Sigma$ patří minimálně symbol prázdného políčka $\sqcup$,
- δ je přechodová funkce; $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$,
- $q_0 \in Q$ je počáteční stav,
- $F \subseteq Q$ je množina koncových stavů.

Přechodovou funkci $\delta(p, a) = (q, b, z)$ interpretujeme takto: Pokud je stroj ve stavu q a na pásce čte symbol a, pak přejde do stavu q, na pásku zapíše (na místo, kde četl symbol a) symbol b a pokud je z různé od nuly, posune hlavu – pro $z = -1$ doleva, pro $z = 1$ doprava.

Nadefinujeme dále konfiguraci Turingova stroje.

Definice 7.3

Def
Konfigurace TS

Mějme libovolný Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Každou uspořádanou trojici $(u, q, v) \in \Gamma^* \times Q \times \Gamma^*$ nazveme konfigurací Turingova stroje M.

Konfigurace (u, q, v) popisuje takový stav Turingova stroje, kdy řetězec u leží na pásce nalevo od hlavy a v pak napravo od hlavy. V aktuálně čteném políčku je první znak řetězce v. Políčka obsahující prázdný znak $\sqcup$ nebereme v úvahu.

Definice 7.4

Def
Krok výpočtu TS

Krok výpočtu M definujeme jako binární relaci $\vdash$ na množině všech konfigurací takto: Pro všechna $p, q \in Q$, $a, b, c, d \in \Gamma$, $\alpha, \beta \in \Gamma^*$ je

$$(\alpha c, p, a\beta) \vdash (\alpha cb, q, \beta),$$

právě když $\delta(p, a) = (q, b, 1)$;

$$(\alpha c, p, a\beta) \vdash (\alpha c, q, b\beta),$$

právě když $\delta(p, a) = (q, b, 0)$;

$$(\alpha c, p, a\beta) \vdash (\alpha, q, cb\beta),$$

právě když $\delta(p, a) = (q, b, -1)$.
Symbolem $\vdash^*$ budeme označovat reflexivní a tranzitivní uzávěr relace $\vdash$, symbolem $\vdash^+$ tranzitivní uzávěr relace $\vdash$.

Pro jistotu ještě zopakujme, že každý výpočet Turingova stroje začíná v počáteční konfiguraci, tj. konfiguraci $(\varepsilon, q_0, \omega)$, kde q_0 je počáteční stav a ω je slovo nad vstupní abecedou.

Výpočet je buď nekonečný nebo skončí v koncové konfiguraci, tj. konfiguraci uqv, kde q je jeden z koncových stavů stroje a slovo uv reprezentuje obsah pásky. Obsah pásky uv v koncové konfiguraci uqv chápeme jako výstup, který stroj vydá pro zadané vstupní slovo, tedy pro zadaný vstup.

Definice 7.5

Def

Jazyk přijímaný TS

Nechť je dán TS $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Jazyk přijímaný TS je definován takto:

$$L(M) = \{\omega \in \Sigma^* \mid (\varepsilon, q_0, \omega) \vdash^* (u, q_f, v), \text{ kde } u, v \in \Gamma^*, q_f \in F\}$$

Pokud se navíc pro každý vstup Turingův stroj zastaví (ať v koncovém nebo jiném stavu), říkáme, že TS **rozhoduje** jazyk $L(M)$. Pokud Turingův stroj rozhoduje nějaký jazyk, říkáme, že řeší problém, neboli je algoritmem. Následující tvrzení je známé pod názvem Church-Turingova teze.

Ke každému algoritmu je možné zkonstruovat s ním ekvivalentní Turingův stroj (s rozumným kódováním vstupů a výstupů řetězci v určité abecedě).

Tohoto tvrzení se využívá při důkazech, že daný problém je algoritmicky rozhodnutelný (existuje TS, který ho rozhodne) nebo algoritmicky řešitelný (existuje TS, který akceptuje vstup, pro nekorektní vstup se nemusí zatavit). S trochou nadsázky můžeme tvrdit, že pokud problém není řešitelný pomocí Turingova stroje, pak není mechanicky řešitelný.

Seznam doporučené literatury

[1] ČERNÁ, I., KŘETÍNSKÝ, M., KUČERA, A. Formální jazyky a automaty I [online]. Elportál, Brno: Masarykova univerzita, 2006. ISSN 1802-128X. [cit. 2014-10-20]
Dostupné z: https://is.muni.cz/publication/703389/cs?lang=cs

[2] HOPCROFT, J., ULLMAN, J. Formal languages and their relation to automata - slov. překlad Formalne jazyky a automaty, Alfa, Bratislava 1978

[3] CHYTIL, M. Automaty a gramatiky. SNTL, Praha 1984.

[4] MELICHAR, B. Jazyky a překlady, Praha: Vydavatelství ČVUT 2003

[5] MOLNÁR, L., MELICHAR, B., ČEŠKA, M. Gramatiky a jazyky, Bratislava: Alfa 1987

Teoretické základy informatiky
pro Informační studia a knihovnictví

RNDr. Lucie Ciencialová Ph.D.

Tato inovace předmětu Teoretické základy informatiky je spolufinancována Evropským sociálním fondem a Státním rozpočtem ČR, projekt číslo CZ.1.07/2.2.00/28.0014 „Interdisciplinární vzdělávání v ICT s jazykovou kompetencí“. Skripta Teoretické základy informatiky slouží primárně jako vzdělávací materiál pro cílovou skupinu projektu.

Recenzenti: RNDr. Šárka Vavrečková, Ph.D.
ing. Richard Pečonka

Vydavatel: Ústav informatiky
Filozoficko-přírodovědecká fakulta
Slezská univerzita v Opavě
Bezručovo náměstí 13
746 01 Opava

Typesetted with LaTeX

Počet stran: 143

Vydání: první, 2014

Tisk: Z+M Partner, spol. s. r. o., Ostrava

Vydáno v Opavě v prosinci 2014

ISBN: 978 – 80 –7510 – 130 – 3