Teoretická informatika

Ladislav Lhotka
⟨*lhotka@cesnet.cz*⟩

2011-12

Zdroje

❶ Linz, P. *Formal Languages and Automata, Fourth Edition*. Sudbury: Jones and Bartlett, 2006, 415+xiii s. ISBN 07-63-73798-4.

❷ Chytil, M. *Automaty a gramatiky*. Praha: SNTL, 1984, 336 s.

❸ Melichar, B. *Jazyky a překlady*. Skriptum FEL ČVUT, 2. vydání. Praha: Nakladatelství ČVUT, 2003. 263 s. ISBN 80-01-02776-7.

❹ Melichar, B. et al. *Jazyky a překlady: Cvičení*. Skriptum FEL ČVUT. Praha: Nakladatelství ČVUT, 2004. 214 s. ISBN 80-01-03031-8.

❺ `http://moodle.prf.jcu.cz`

Použité partie z matematiky

Konečná matematika, i když často pracujeme s nekonečnými množinami. Výsledky budeme často formulovat formou matematických tvrzení (vět), důkazy ale budou téměř ve všech případech konstruktivní. Často bude použita *matematická indukce*.

- množiny
- relace
- funkce
- orientované grafy

Množiny

prázdná množina $\emptyset$, podmnožina $A \subseteq B$, *vlastní* podmnožina $A \subset B$, průnik $A \cap B$, sjednocení $A \cup B$, rozdíl $A - B$, doplněk $\overline{A}$.

Potenční množina (množina všech podmnožin) množiny A:

$$2^A = \{ B \mid B \subseteq A \}.$$

$|A|$ – počet prvků množiny A

Kartézský součin

$$A \times B = \{ (x, y) \mid x \in X, y \in Y \}.$$

De Morganovy vzorce:

$$\overline{A \cup B} = \overline{A} \cap \overline{B},$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}.$$

Relace

Relace mezi množinami A a B je každá podmnožina kartézského součinu $A \times B$. Pokud je $A = B$ – relace *na* množině A.

Důležité vlastnosti relací (R je relace na množině A):

1. reflexivita – $\forall x \in A : (x, x) \in R$.
2. symetrie – $\forall x, y \in A : (x, y) \in R \Rightarrow (y, x) \in R$.
3. tranzitivita –

$$\forall x, y, z \in A : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R.$$

Reflexivní, symetrická a tranzitivní relace – *ekvivalence* ($x \equiv y$).

Funkce

Funkce je speciálním případem relace.

$f\colon A \to B$, $y = f(x)$ – y je *obrazem* prvku x, x je *vzorem* prvku y.

Parciální funkce: není definováno pro všechna $x \in A$

Totální funkce: je definováno pro všechna $x \in A$

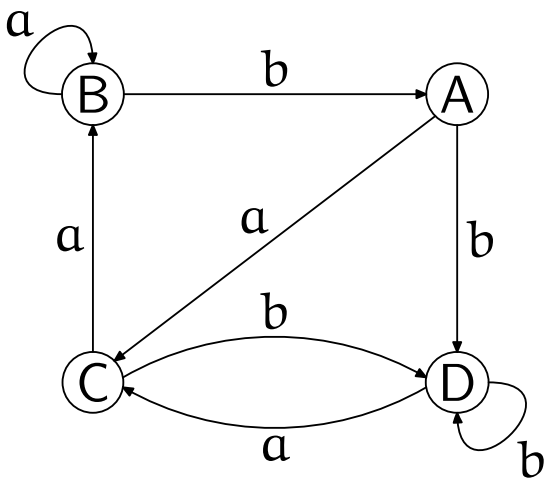
Prostá (injektivní) funkce: $x \neq x' \Rightarrow f(x) \neq f(x')$

Funkce *na* množinu B (*surjektivní*): Ke každému $y \in B$ najdeme jeho vzor $x \in A$, tj. $y = f(x)$ pro nějaké $x \in A$.

Bijektivní funkce (bijekce): prostá a na

Orientovaný graf

Orientovaný graf sestává z neprázdné množiny *vrcholů* či *uzlů* libovolně pospojovaných *orientovanými hranami*.



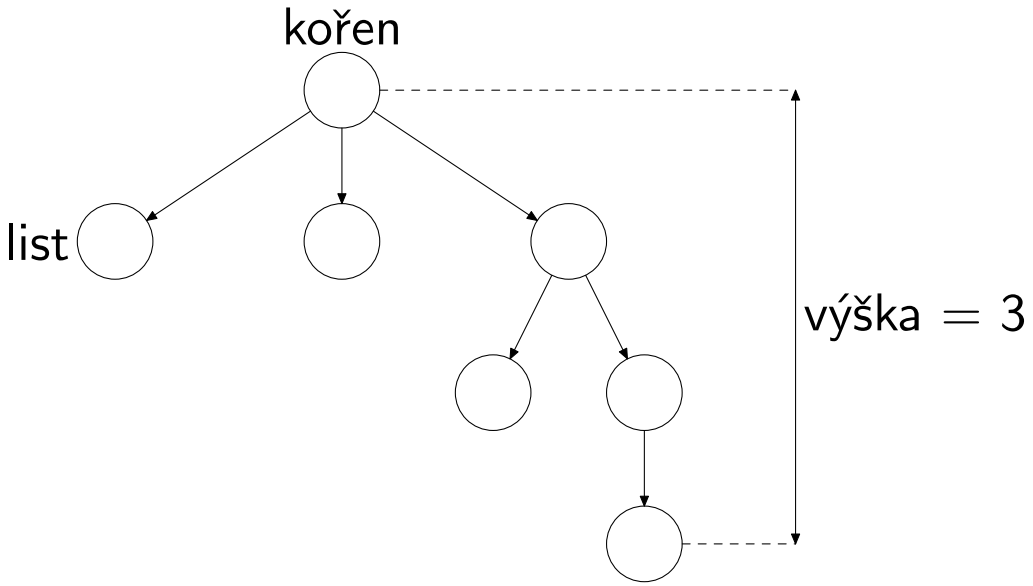
Orientovanou hranu chápeme jako uspořádanou dvojici vrcholů.

Cesta délky n (z v_1 do v_{n+1}): konečná posloupnost hran (v_1, v_2), (v_2, v_3), . . . , (v_n, v_{n+1}).

Cesta se nazývá *jednoduchá*, když se na ní žádný uzel neopakuje.

Kružnice (cyklus) – cesta, která se vrací do stejného uzlu.

Strom



Tři hlavní pojmy

- Jazyk
- Gramatika
- Automat

Abeceda, znaky, řetězy

Σ – *abeceda*, neprázdná konečná množina *znaků (symbolů)*.

Ze znaků sestavujeme konečné posloupnosti – *řetězy*.

Příklad: Je-li $\Sigma = \{a, b\}$, pak $u = abba$ a $v = aaabba$ jsou řetězy nad abecedou Σ.

$|u|$ – délka řetězu u (počet znaků).

Řeckým písmenem λ se označuje *prázdný řetěz*; $|\lambda| = 0$.

Σ^* ... množina *všech* řetězů nad abecedou Σ.

$\Sigma^+ = \Sigma^* - \{\lambda\}$.

Σ^* i Σ^+ jsou nekonečné množiny, protože délka řetězu není omezena.

Podřetěz je řetěz, který je celý obsažen v jiném řetězu. Speciálně, pokud $w = vu$, označujeme v jako *prefix* a u jako *sufix*.

Operace s řetězy

$u = a_1 a_2 \dots a_n$, $v = b_1 b_2 \dots b_m$ jsou řetězy.

Zřetězení: $uv = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$.

Obrácení (reverze): $u^R = a_n a_{n-1} \dots a_1$.

Zřejmě platí:

- $\lambda w = w\lambda = w$,
- $|uv| = |u| + |v|$,
- $|u^R| = |u|$.

Dále značíme: $w^2 = ww$, $w^3 = www$, obecně $w^n = ww \cdots w$ (n-krát), a definujeme $w^0 = \lambda$.

Jazyk

Velmi obecná definice:

Jazykem nad abecedou Σ nazveme libovolnou podmnožinu $L \subseteq \Sigma^*$.

Příklady: pro $\Sigma = \{a, b\}$

- $L_1 = \{\lambda, a, aa, b\}$ (konečný)
- $L_2 = \{\, a^n b^n \mid n \geq 0 \,\}$ (nekonečný)

Řetěz $w \in L$ nazýváme *větou* jazyka L.

Operace s jazyky

Jazyky jsou množiny, můžeme tedy použít množinové operace: $L_1 \cap L_2$, $L_1 \cup L_2$, $\overline{L} = \Sigma^* - L$.

Zobecnění operací s řetězy:

1. Zřetězení: $L_1 L_2 = \{ uv \mid u \in L_1, v \in L_2 \}$
2. Reverze: $L^R = \{ u^R \mid u \in L \}$
3. Opakování: $L^n = LL \cdots L$ (n-krát), $L^0 = \{\lambda\}$.

(Kleeneův) uzávěr jazyka L:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$$

Pozitivní uzávěr jazyka L:

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$$

Příklad 1

Pro $L = \{ a^n b^n \mid n \geq 0 \}$ snadno odvodíme, že

$$L^2 = \{ a^n b^n a^m b^m \mid n \geq 0, m \geq 0 \},$$
$$L^R = \{ b^n a^n \mid n \geq 0 \}.$$

Podobně jednoduchý zápis např. pro doplněk $\overline{L}$ ale nedostaneme. Potřebujeme proto lepší formální nástroje pro reprezentaci jazyků než je množinový zápis.

Gramatika

Gramatiky podobně jako v přirozených jazycích určují které věty do jazyka patří a které ne.

$$\begin{aligned} \langle \text{věta holá} \rangle &\to \langle \text{podmět} \rangle \langle \text{přísudek} \rangle \\ \langle \text{podmět} \rangle &\to \lambda \mid \langle \text{zájmeno} \rangle \mid \langle \text{podstatné jméno} \rangle \\ \langle \text{přísudek} \rangle &\to \langle \text{sloveso} \rangle \end{aligned}$$

Definice 1

Gramatika G je definována jako uspořádaná čtveřice

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

kde

- V je neprázdná konečná množina *proměnných*,
- T je konečná množina *koncových (terminálních) symbolů*,
- $S \in V$ je speciální proměnná zvaná *počáteční (startovní) proměnná*,
- P je konečná množina *přepisovacích pravidel (produkcí)*.

Přepisovací pravidla

Přepisovací pravidla se zapisují stylem $x \to y$, kde $x \in (V \cup T)^+$ a $y \in (V \cup T)^*$.

Používají se takto: Máme-li řetěz w ve tvaru

$$w = uxv,$$

říkáme, že produkce $x \to y$ je aplikovatelná na w a w můžeme přepsat na nový řetěz

$$z = uyv.$$

V takovém případě říkáme, že řetěz z lze *přímo odvodit* z w. Zápis: $w \Rightarrow z$.

Pokud má gramatika více produkcí, můžeme pro přepis v každém kroku použít kteroukoli z nich (nedeterminismus!). Posloupnost odvození

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

znamená, že w_n lze *odvodit* z w_1. Zápis:

$$w_1 \overset{*}{\Rightarrow} w_n.$$

Jazyk generovaný gramatikou

Množina všech řetězů, které lze odvodit z počáteční proměnné.

Definice 2

Nechť $G = (V, T, S, P)$ je gramatika. Množinu

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

nazýváme jazykem generovaným gramatikou G.

Je-li $w \in L(G)$, pak

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w$$

nazýváme *odvozením (derivací)* věty w pomocí gramatiky G. Mezistupně S, w_1, w_2 atd. se nazývají *větné formy* tohoto odvození.

Pro mnohé jazyky lze najít více gramatik, které je generují. Pokud tedy máme dvě gramatiky G_1 a G_2 takové, že $L(G_1) = L(G_2)$, říkáme, že tyto gramatiky jsou *ekvivalentní*.

Příklad 2

Uvažujme gramatiku $G = (\{S\}, \{a, b\}, S, P)$ s těmito přepisovacími pravidly:

$$S \to aSb,$$
$$S \to \lambda.$$

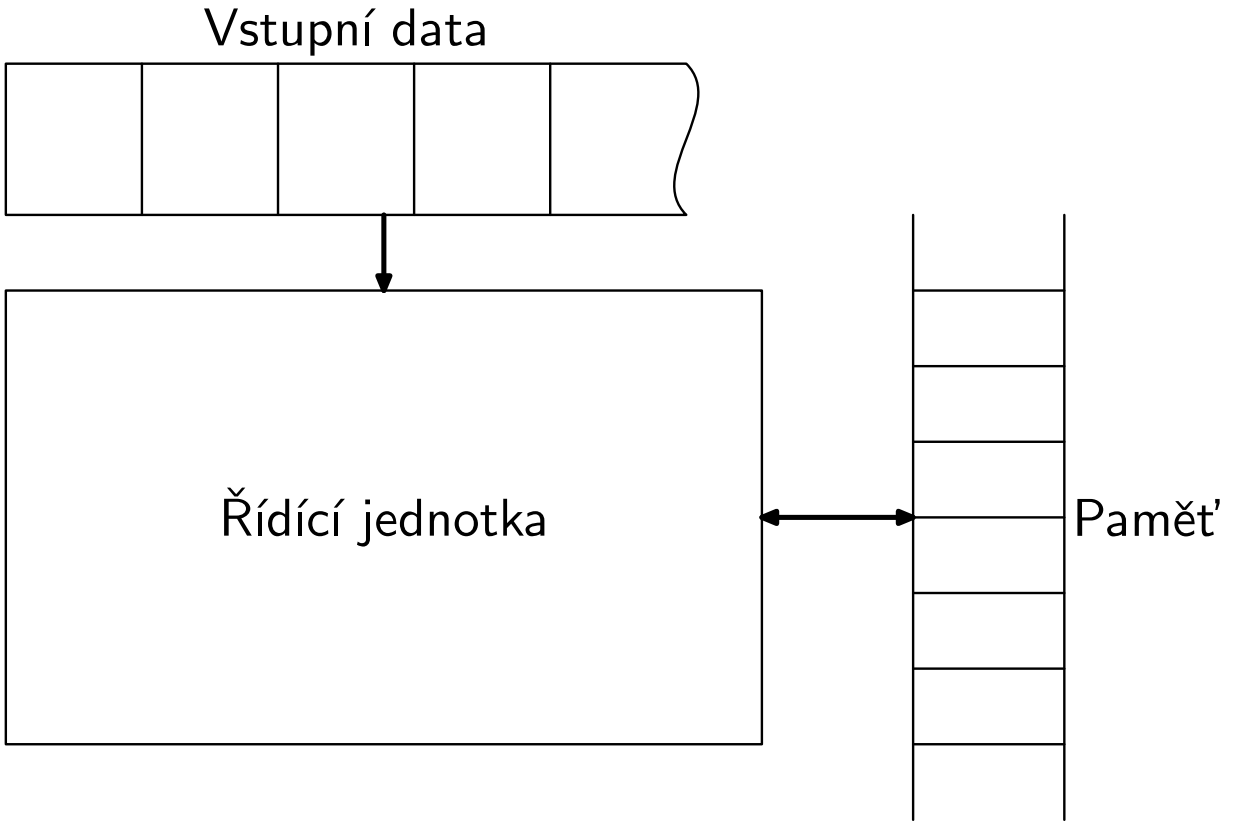
Jaký jazyk generuje?

Zkrácený zápis: Pravidla se shodnou levou stranou se zapisují dohromady, pravé strany se oddělují svislou čarou. Produkce z příkladu 2 tedy můžeme zapsat takto:

$$S \to aSb \mid \lambda.$$

Automat

Abstraktní model konkrétního algoritmu nebo celého digitálního počítače.



Vlastnosti automatu

Vstupní soubor na začátku obsahuje řetěz znaků z nějaké abecedy, automat jej může pouze sekvenčně číst od prvního znaku k poslednímu (a rozpozná konec vstupu). *Paměť* může mít různé vlastnosti a nemusí být vůbec přítomna. Automat z ní může číst a do ní zapisovat (měnit obsah). Paměťové buňky (obvykle bez omezení počtu) obsahují znaky z nějaké abecedy, která nemusí být stejná jako vstupní abeceda. *Řídící jednotka* má konečný počet vnitřních stavů a zařízení pro vstup a výstup.

Automat pracuje v diskrétních krocích. V každém kroku je automat v nějakém vnitřním stavu a čte znak ze vstupního souboru. Postup výpočtu je určen *přechodovou funkcí*, která na základě aktuálního stavu, čteného vstupního symbolu a obsahu paměti určí nový stav řídící jednotky.

Konfigurace automatu je dána vnitřním stavem řídící jednotky, a obsahy dosud nepřečtené části vstupního souboru a paměti. Přechod od jedné konfigurace k jiné se někdy nazývá *tah* automatu.

Deterministický a nedeterministický automat

Deterministický – následující tah je vždy jednoznačně určen aktuální konfigurací.

Nedeterministický – některé konfigurace umožňují více různých tahů.

Vztah mezi deterministickými a nedeterministickými verzemi automatů tvoří důležitou součást teorie automatů.

Příklad 3

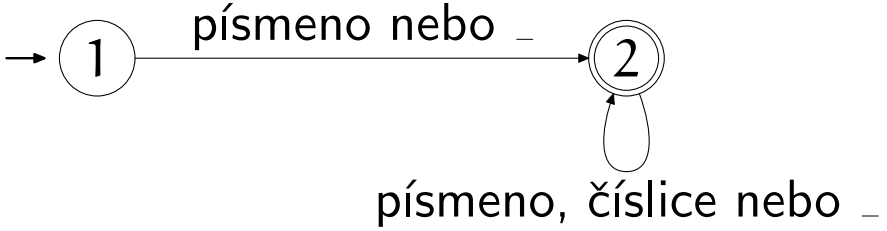
Identifikátor v jazyku C se tvoří podle těchto dvou pravidel:

1. Identifikátor je posloupnost písmen, číslic a podtržítka.
2. Na začátku musí být písmeno nebo podtržítko.

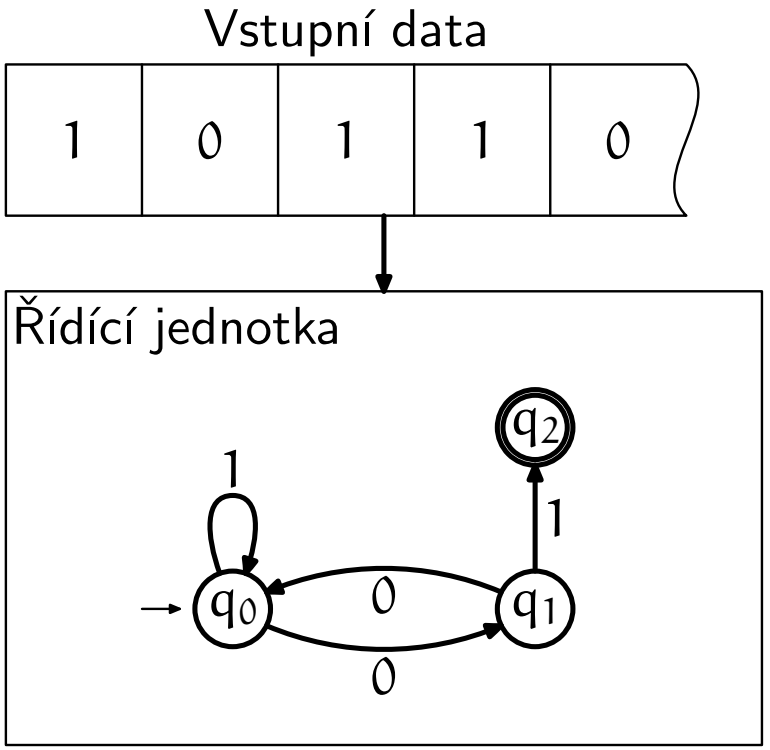
Gramatika:

$$\langle \text{identifikátor} \rangle \to \langle \text{písmeno} \rangle \langle \text{zbytek} \rangle \mid \_ \ \langle \text{zbytek} \rangle$$
$$\langle \text{zbytek} \rangle \to \langle \text{písmeno} \rangle \langle \text{zbytek} \rangle \mid \langle \text{číslice} \rangle \langle \text{zbytek} \rangle \mid \_ \ \langle \text{zbytek} \rangle \mid \lambda$$
$$\langle \text{písmeno} \rangle \to \text{a} \mid \text{b} \mid \ldots \mid \text{z} \mid \text{A} \mid \text{B} \mid \ldots \mid \text{Z}$$
$$\langle \text{číslice} \rangle \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid$$

Automat:



Konečný automat



Konečný automat čte sekvenčně znaky vstupního řetězu, podle přečteného znaku a aktuálního stavu řídící jednotky přechází do nového stavu.

Konečný automat nemá žádnou paměť.

Deterministický konečný automat

Definice 3

Deterministickým konečným automatem (DFA) nazýváme uspořádanou pětici $M = (Q, \Sigma, \delta, q_0, F)$, kde

- Q je neprázdná konečná množina zvaná *množina vnitřních stavů*,
- Σ je konečná množina zvaná *vstupní abeceda*,
- $\delta\colon Q \times \Sigma \to Q$ je totální funkce zvaná *přechodová funkce*,
- $q_0 \in Q$ je *počáteční stav*,
- $F \subseteq Q$ je *množina koncových stavů (cílová množina)*

Činnost konečného automatu

1. Na počátku je automat v počátečním stavu q_0 a na začátku vstupního souboru (čte první symbol vstupního řetězu).
2. V každém kroku se přečte jeden symbol v pořadí ze vstupního souboru a podle tohoto symbolu a aktuálního vnitřního stavu přejde automat do nového vnitřního stavu.
3. Pokud je automat na konci vstupního souboru a nachází se v koncovém stavu, je vstupní řetěz *přijat*, v opačném případě (tj. když automat je v jiném než koncovém stavu) je vstupní řetěz *odmítnut*.

Automat, který pouze přijímá anebo odmítá vstupní řetězec, se někdy také nazývá *akceptér*.

Přechodový graf

Každý stav DFA je reprezentován jedním vrcholem grafu, pro každý vstupní symbol vychází z každého vrcholu právě jedna hrana označená příslušným vstupním symbolem.

Hrana (q_i, q_j) označená symbolem $a \in \Sigma$ se v přechodovém grafu nachází právě tehdy, když

$$\delta(q_i, a) = q_j.$$

počáteční stav je označen šipkou,

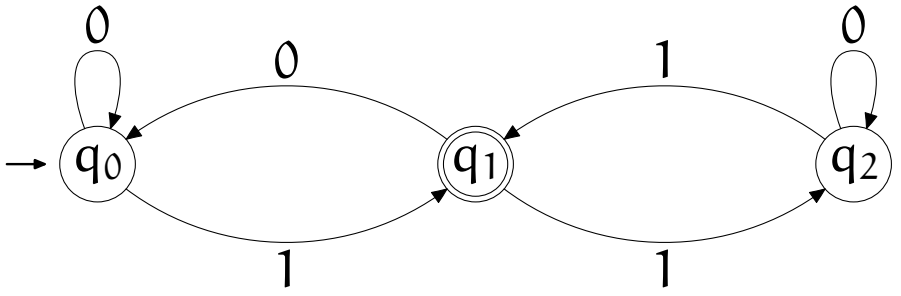
koncový stav je označen dvojitým kroužkem.

Příklad 4

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\}),$$

přechodová funkce δ je definována tímto výčtem:

$$\begin{aligned}
\delta(q_0, 0) &= q_0, \ \delta(q_0, 1) = q_1, \\
\delta(q_1, 0) &= q_0, \ \delta(q_1, 1) = q_2, \\
\delta(q_2, 0) &= q_2, \ \delta(q_2, 1) = q_1.
\end{aligned}$$



Rozšířená přechodová funkce δ^*

$$\delta^*\colon Q \times \Sigma^* \to Q$$

Druhým argumentem je řetěz, a hodnotou $\delta^*(q, w)$ je stav, do něhož se automat dostane (ve více krocích) ze stavu q přečtením řetězu w.

Formální definice – pro všechna $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$ je funkce δ^* definována tímto rekurzivním předpisem:

$$\delta^*(q, \lambda) = q,$$
$$\delta^*(q, wa) = \delta(\delta^*(q, w), a).$$

Speciálně pro $|v| = 1$ platí $\delta^*(q, v) = \delta(q, v)$ pro všechna q.

Jazyk přijímaný konečným automatem

Množina všech řetězů, které daný konečný automat přijímá.

Definice 4

Jazykem přijímaným (rozpoznávaným) deterministickým konečným automatem $M = (Q, \Sigma, \delta, q_0, F)$ je množina

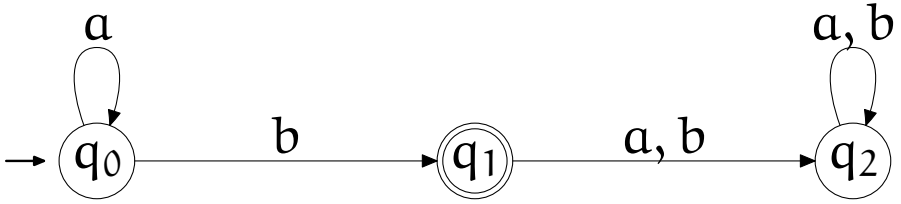
$$L(M) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \in F \}.$$

Funkce δ i δ^* jsou totální, to znamená, že každý řetěz $w \in \Sigma^*$ patří buď do $L(M)$ anebo do

$$\overline{L(M)} = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \notin F \}.$$

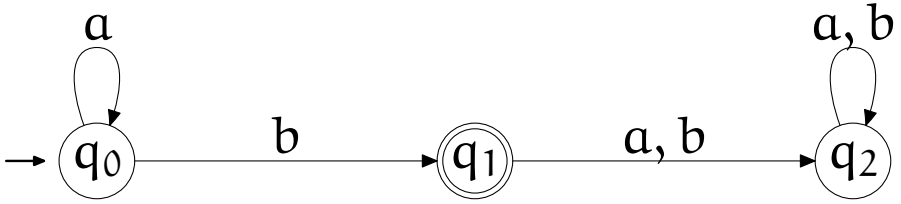
Příklad 5

Deterministický konečný automat M:



Příklad 5

Deterministický konečný automat M:



$$L(M) = \{ a^n b \mid n \geq 0 \}.$$

Přechodová tabulka:

	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

Vztah mezi DFA a přechodovým grafem

Věta 1

Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat a G_M jemu odpovídající přechodový graf. Potom pro každé $q_1, q_2 \in Q$ a $w \in \Sigma^*$ je $\delta^*(q_1, w) = q_2$ právě tehdy, když v grafu G_M existuje orientovaná cesta z vrcholu q_1 do q_2, jejíž hrany jsou postupně označeny všemi symboly řetězu w.

Regulární jazyk

Uvažujeme-li všechny možné DFA, tvoří jimi přijímané jazyky určitou třídu jazyků se specifickými vlastnostmi.

Definice 5

Jazyk L se nazývá *regulární*, pokud existuje takový deterministický konečný automat M, že platí

$$\mathrm{L} = \mathrm{L}(\mathrm{M}).$$

Nedeterministický konečný automat

Definice 6

Nedeterministickým konečným automatem (NFA) nazýváme uspořádanou pětici $M = (Q, \Sigma, \delta, q_0, F)$, kde Q, Σ, q_0 a F jsou definovány stejně jako u DFA, liší se pouze předpis pro přechodovou funkci:

$$\delta: Q \times (\Sigma \cup \{\lambda\}) \to 2^Q.$$

Rozdíly:

1. NFA může mít pro daný stav a vstupní symbol na výběr několik možných tahů.
2. Druhým argumentem přechodové funkce může být λ, tj. $\delta(q_i, \lambda) = q_j$. NFA tedy může přejít do nového stavu, aniž by zkonzumoval vstupní symbol.
3. Hodnotou přechodové funkce může být prázdná množina, tj. $\delta(q_i, a) = \emptyset$. Pro takovou dvojici q_i a a tedy není definován žádný přechod. V přechodovém grafu takové hrany vynecháváme.

Jak NFA rozpoznává řetězy?

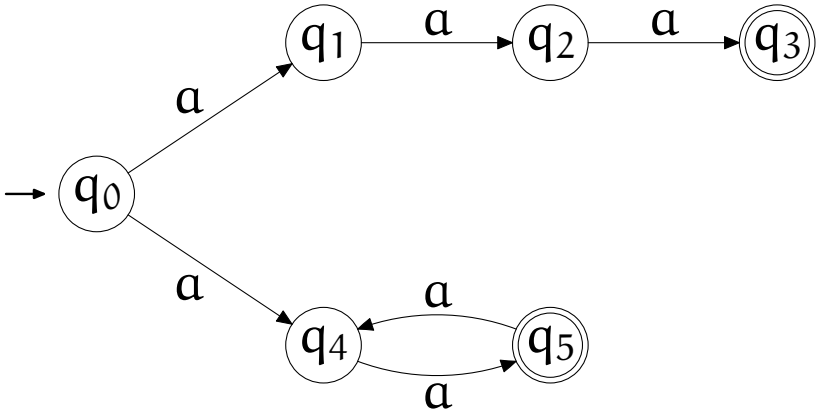
Vstupní řetěz je přijat nedeterministickým konečným automatem právě tehdy, když existuje nějaká posloupnost možných tahů, které

- načtou celý vstupní řetěz,
- přivedou NFA do některého koncového stavu.

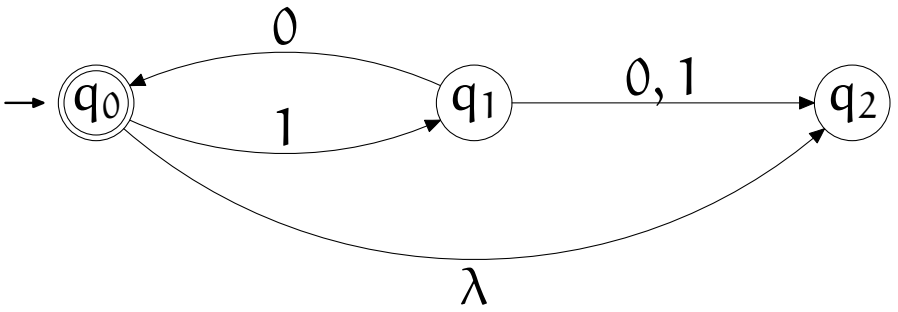
Pokud žádná taková posloupnost neexistuje, vstupní řetěz je odmítnut.

Příklad 6

(a)



(b)



Rozšířená přechodová funkce

Hodnotami jsou množiny stavů: $\delta^*(q_i, w) = Q_j \subseteq Q$.

Definice 7

Pro daný nedeterministický konečný automat M definujeme rozšířenou přechodovou funkci δ^*: $Q \times \Sigma^* \to 2^Q$ takto:

Pro každé dva stavy q_i a q_j a řetěz $w \in \Sigma^*$ je $q_j \in \delta^*(q_i, w)$ právě tehdy, když v přechodovém grafu existuje cesta z q_i do q_j označkovaná symboly řetězu w (popřípadě s vloženým λ-přechody).

Existenci cesty můžeme zjistit v konečném počtu kroků, jen musíme dát pozor na cykly označkované samými λ.

Jazyk přijímaný NFA

Definice 8

Jazyk L přijímaný nedeterministickým konečným automatem $M = (Q, \Sigma, \delta, q_0, F)$ je množina všech řetězů přijímaných tímto automatem, tedy

$$L(M) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset \}.$$

Je to tedy množina všech řetězů w takových, že pro ně existuje v přechodovém grafu cesta označkovaná symboly řetězu w a vedoucí z počátečního vrcholu do některého koncového vrcholu.

K čemu je nedeterminismus?

1. Pro některé jazyky (viz třeba příklad 6 (a)) se snáze najde NFA než DFA.
2. V gramatikách se nedeterminismus (alternativy) vyskytuje často, například

$$\langle \text{podmět} \rangle \to \langle \text{podstatné jméno} \rangle | \langle \text{zájmeno} \rangle.$$

3. Mnohé vlastnosti jazyků a teoretické výsledky se snáze ukazují pro NFA než pro DFA.
4. V některých algoritmech (např. hry) je mnohdy více možností postupu, přitom ale není dopředu jasné, která je správná. Při praktické realizaci se využívá *backtracking*, NFA ale modeluje procházení všech variant zároveň.

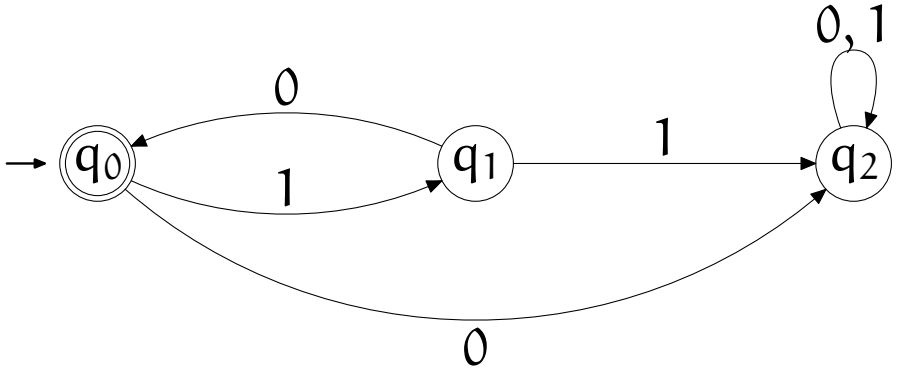
Ekvivalence konečných automatů

Definice 9

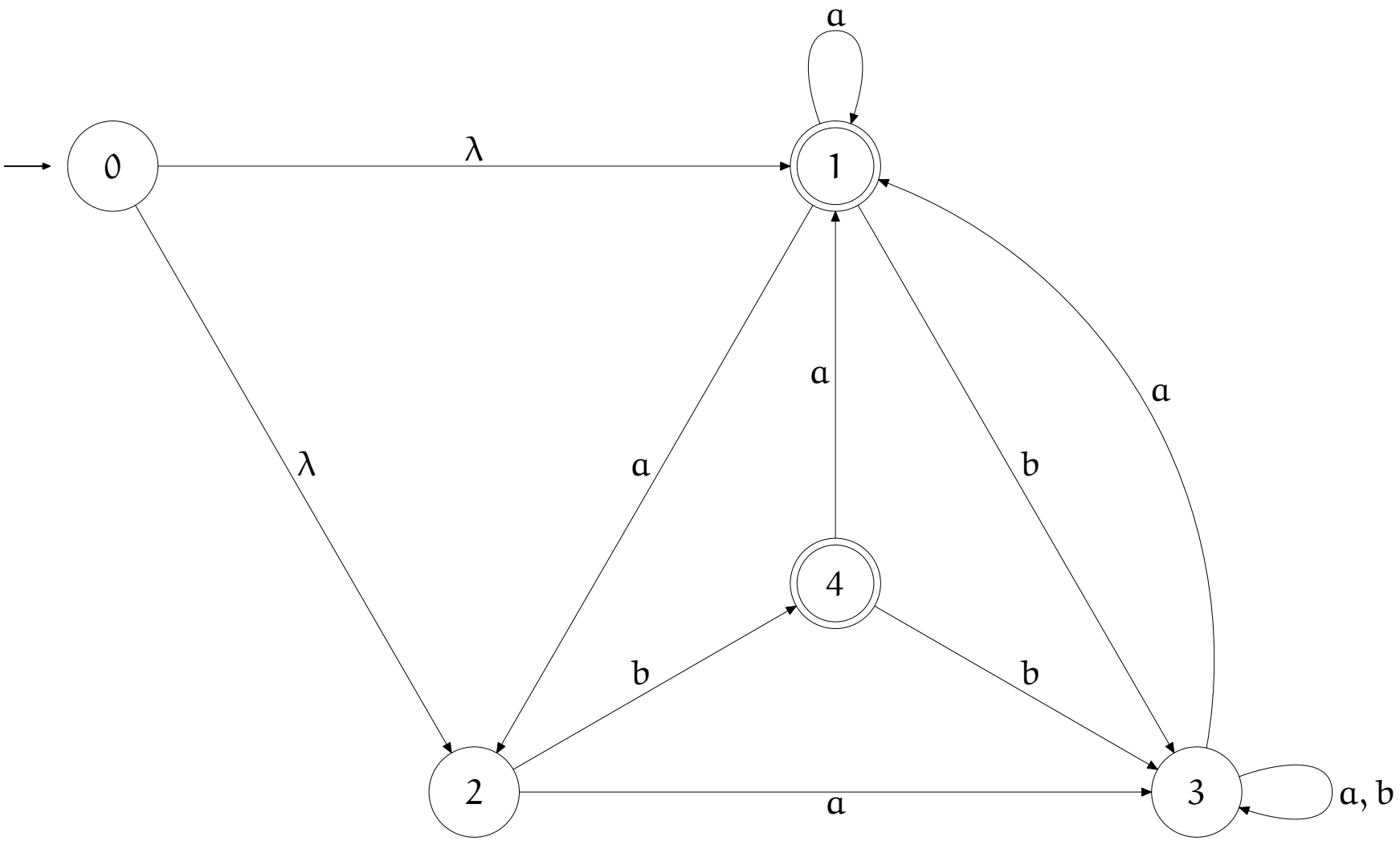
Řekneme, že dva konečné automaty M_1 a M_2 jsou *ekvivalentní*, pokud oba přijímají stejný jazyk, tj.

$$L(M_1) = L(M_2).$$

Příklad 7: Následující automat je ekvivalentní automatu z příkladu 2.3 (b). Oba přijímají jazyk $\{ (10)^n \mid n \geq 0 \}$.

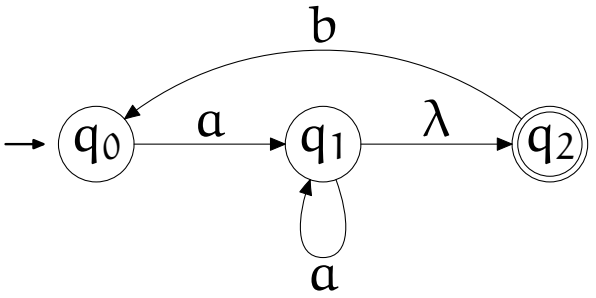


Příklad 8

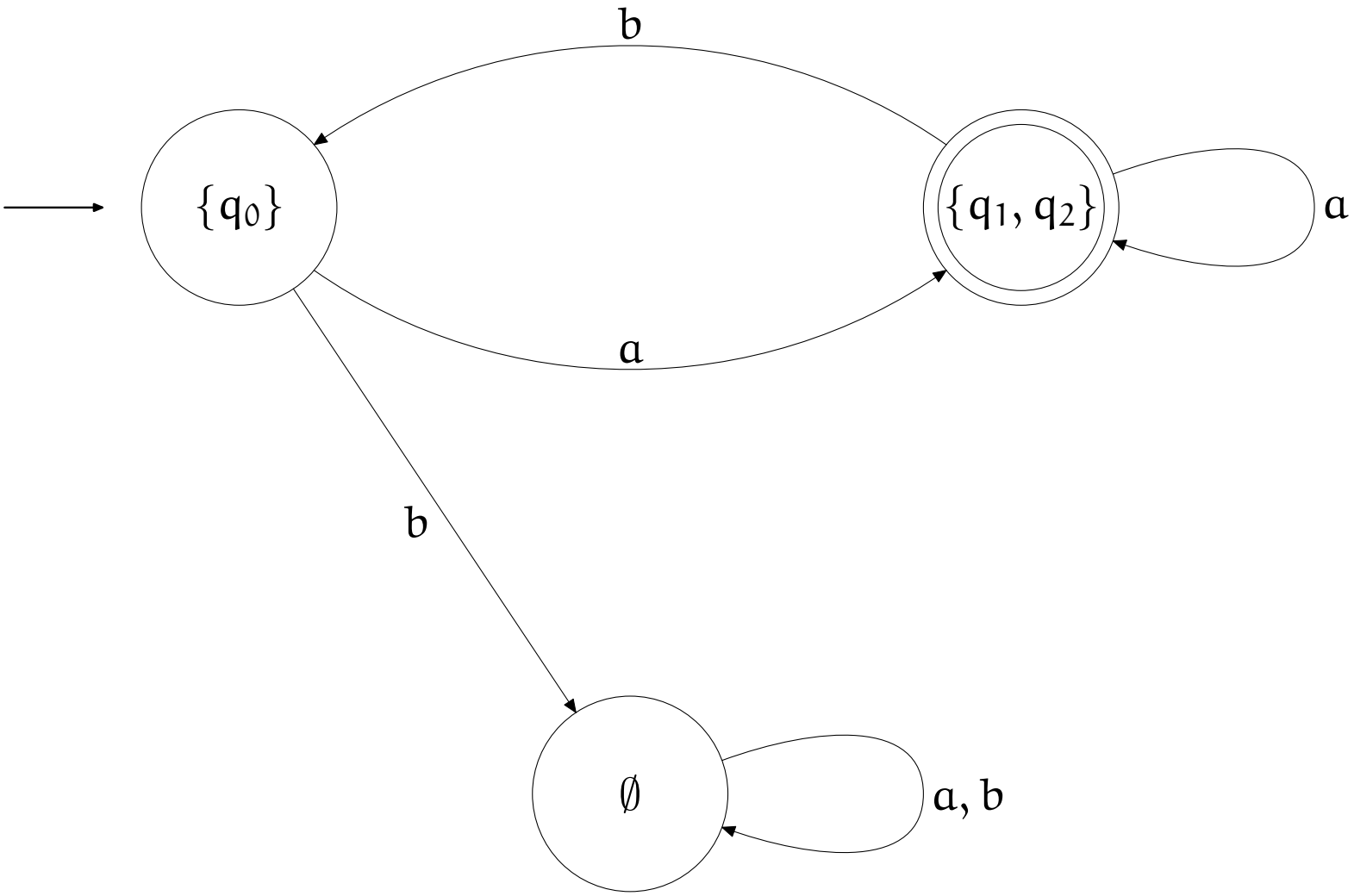


Příklad 9

Najděte ekvivalentní DFA k tomuto NFA:



Výsledek:



Ekvivalence tříd NFA a DFA

Věta 2

Nechť L je jazyk přijímaný nedeterministickým konečným automatem

$$M_N = (Q_N, \Sigma, \delta_N, q_0, F_N).$$

Potom existuje deterministický konečný automat

$$M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$$

takový, že

$$L = L(M_D).$$

Přímý důsledek: Každý jazyk rozpoznávaný NFA je *regulární*.

Procedura NFA $\longrightarrow$ DFA

❶ Vytvoř graf G_D s jediným (počátečním) vrcholem označeným $\{q_0\}$. Pokud M_N přijímá λ, označ $\{q_0\}$ také jako koncový vrchol.

❷ Opakuj následující kroky i–iv tak dlouho, dokud chybí některá z hran:

(i) Vezmi libovolný vrchol $\{q_i, q_j, \ldots, q_k\}$ jemuž chybí výstupní hrana pro nějaké $a \in \Sigma$.

(ii) Urči $\delta_N^*(q_i, a)$, $\delta_N^*(q_j, a)$, …, $\delta_N^*(q_k, a)$. Nechť

$$\delta_N^*(q_i, a) \cup \delta_N^*(q_j, a) \cup \delta_N^*(q_k, a) = \{q_l, q_m, \ldots, q_n\}.$$

(iii) Vytvoř nový vrchol grafu G_D označený $\{q_l, q_m, \ldots, q_n\}$, pokud ještě neexistuje.

(iv) Přidej do grafu G_D hranu z $\{q_i, q_j, \ldots, q_k\}$ do $\{q_l, q_m, \ldots, q_n\}$ a označ ji symbolem a.

❸ Každý vrchol grafu G_D jehož označení obsahuje aspoň jeden $q_f \in F_N$ označ jako koncový vrchol.

Regulární výrazy

Pomocí symbolů abecedy Σ, závorek a operátorů + (sjednocení), $\cdot$ (zřetězení) a * (uzávěr) můžeme definovat složitější jazyky jako kombinace jednodušších.

Regulární výraz	**Jazyk**
a	$\{a\}$
$a + b + c$	$\{a, b, c\}$
$(a + (b \cdot c))^*$	$\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \ldots\}$

Definice 10

Nechť Σ je abeceda. Potom

1. $\emptyset$, λ, a a (pro každé $a \in \Sigma$) jsou regulární výrazy – říkáme jim *primitivní regulární výrazy*.
2. Jsou-li r_1 a r_2 regulární výrazy, pak jsou jimi také $r_1 + r_2$, $r_1 \cdot r_2$, r_1^* a (r_1).
3. Řetěz je regulárním výrazem právě tehdy, když je možno jej odvodit z primitivních regulárních výrazů konečným počtem použití pravidel uvedených v bodu 2.

Příklad: $(a + b \cdot c)^* \cdot (c + \emptyset)$ je regulární výraz nad abecedou $\Sigma = \{a, b, c\}$.

Jazyk reprezentovaný regulárním výrazem

Definice 11

Jazyk $L(r)$ reprezentovaný regulárním výrazem r je definován následujícími pravidly:

1. Regulární výraz $\emptyset$ označuje prázdný jazyk.
2. Regulární výraz λ označuje jazyk $\{\lambda\}$.
3. Pro každé $a \in \Sigma$ označuje regulární výraz a jednoprvkový jazyk $\{a\}$.
4. Jsou-li r_1 a r_2 regulární výrazy, pak

$$\begin{aligned} L(r_1 + r_2) &= L(r_1) \cup L(r_2), \\ L(r_1 \cdot r_2) &= L(r_1)L(r_2), \\ L((r_1)) &= L(r_1), \\ L(r_1^*) &= (L(r_1))^*. \end{aligned}$$

Priorita operací, zjednodušení

Abychom nemuseli používat příliš závorek, definujeme prioritu operací:

* má nejvyšší prioritu, $\cdot$ má vyšší prioritu než +.

Například $a + b \cdot c^*$ tedy znamená totéž jako $a + (b \cdot (c)^*)$.

Další zjednodušující konvence: operátor $\cdot$ vynecháváme, takže

$$ab = a \cdot b.$$

Příklad 10: Je-li $\Sigma = \{0, 1\}$, najděte regulární výraz pro jazyk, v němž každý řetěz má v sobě alespoň jeden pár po sobě jdoucích nul.

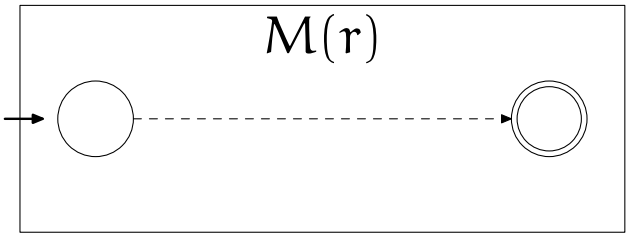
[Výsledek: $(0 + 1)^*00(0 + 1)^*$]

Regulární výrazy reprezentují regulární jazyky

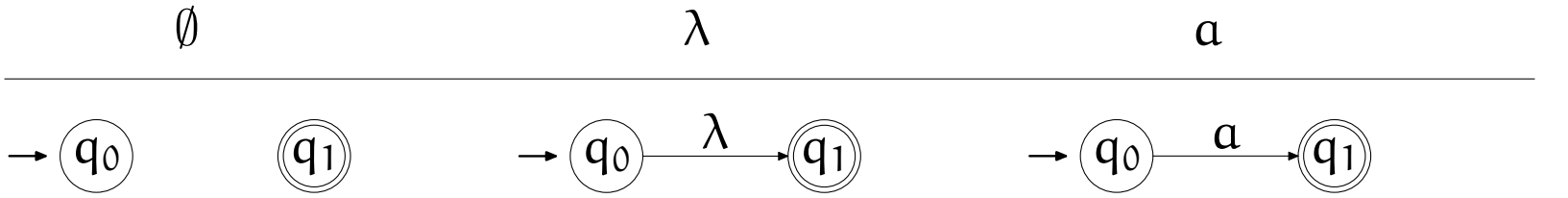
Věta 3

Pro každý regulární výraz r existuje nedeterministický konečný automat, který přijímá jazyk $L(r)$. Jinými slovy, $L(r)$ je regulární jazyk.

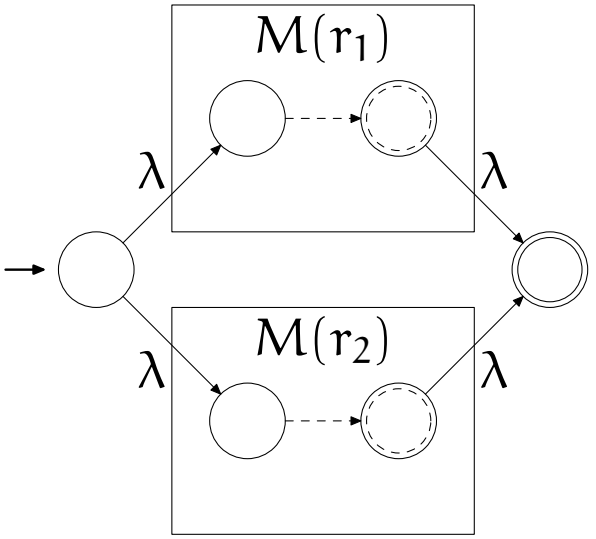
Znázornění NFA pro libovolný regulární výraz r:



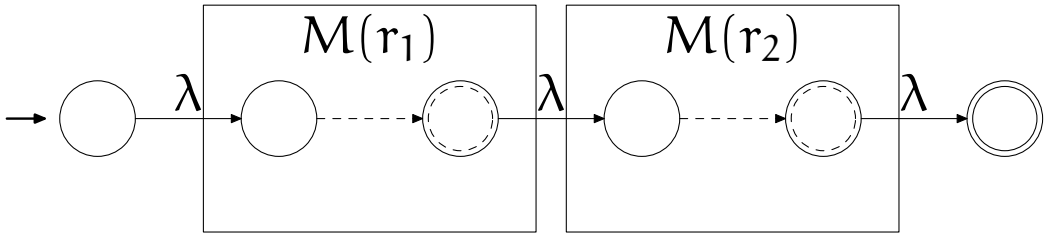
Automaty pro primitivní regulární výrazy



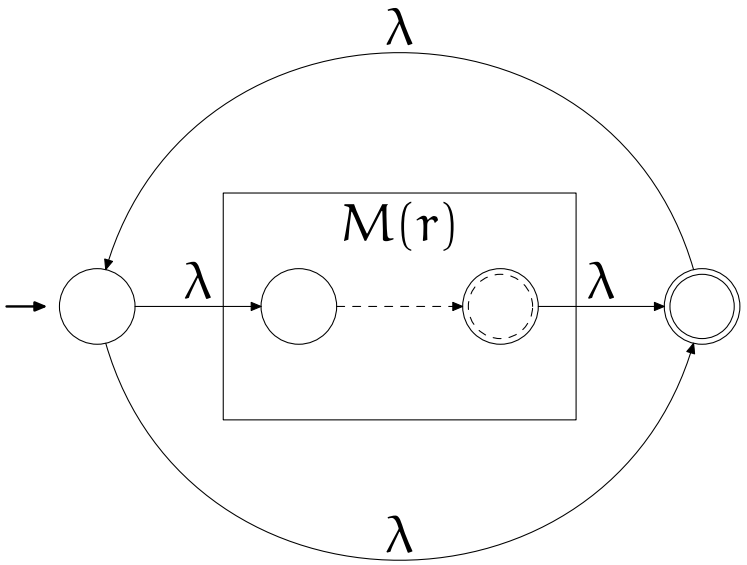
Automat pro alternativu $r_1 + r_2$



Automat pro zřetězení $r_1 r_2$



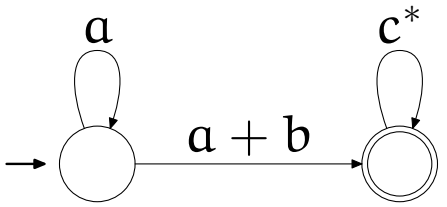
Automat pro uzávěr r^*



Zobecněný přechodový graf

Dále budeme chtít k danému NFA M zkonstruovat regulární výraz r takový, že $L(M) = L(r)$. Půjde v podstatě o obrácení postupu konstrukce v důkazu věty 3: automaty, které reprezentují primitivní regulární výrazy, budeme postupně agregovat.

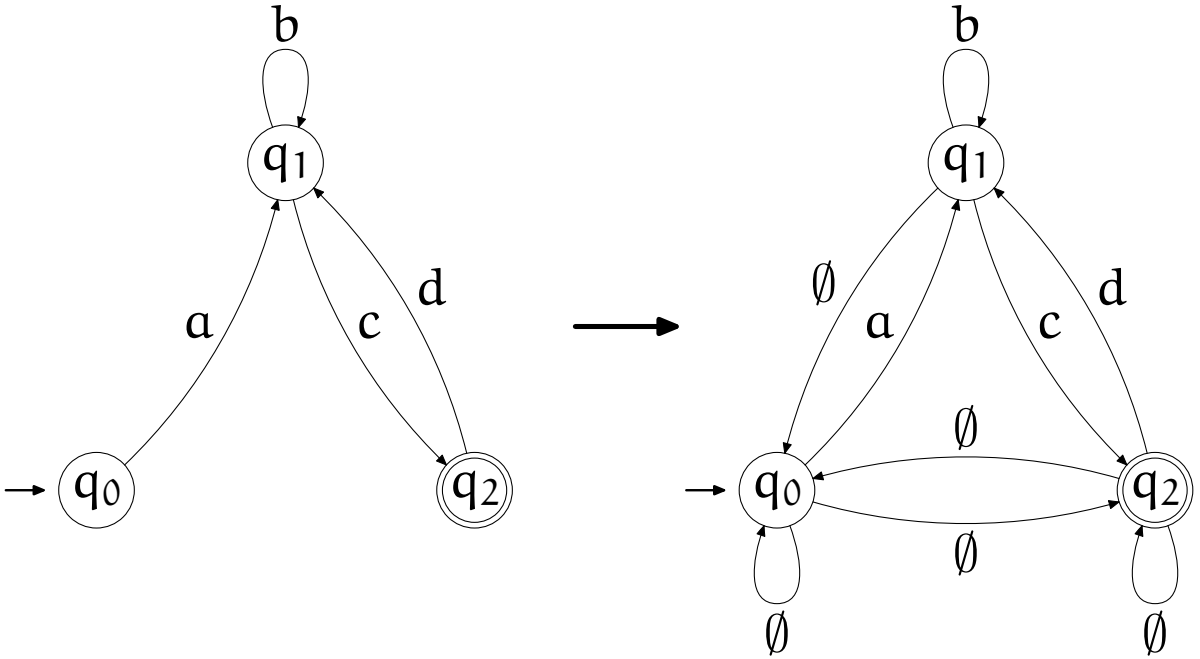
Zobecněný přechodový graf (GTG, generalized transition graph) má hrany označené regulárními výrazy. Například:



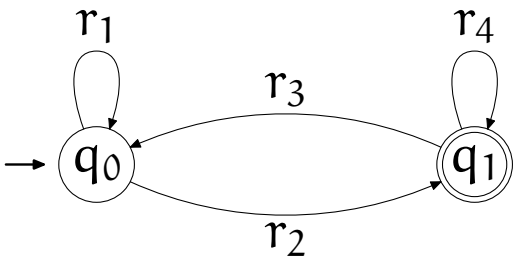
Přechodový graf NFA můžeme považovat za GTG.

Pro zjednodušení teoretických úvah budeme vždy pracovat s *plnými* GTG, které mají pro každou uspořádanou dvojici vrcholů právě jednu hranu. (Počet hran v úplném grafu s n vrcholy je n^2.)

Příklad 11



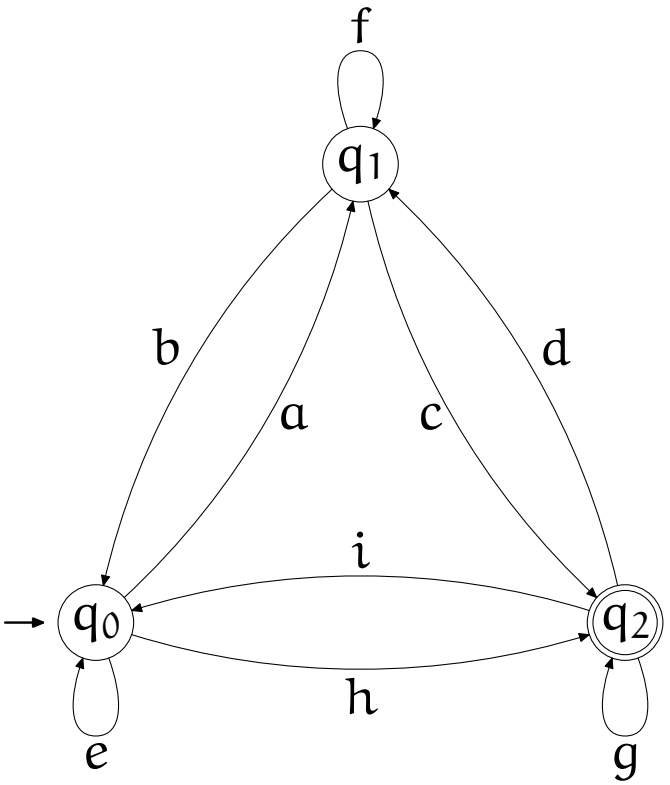
Obecný dvoustavový automat



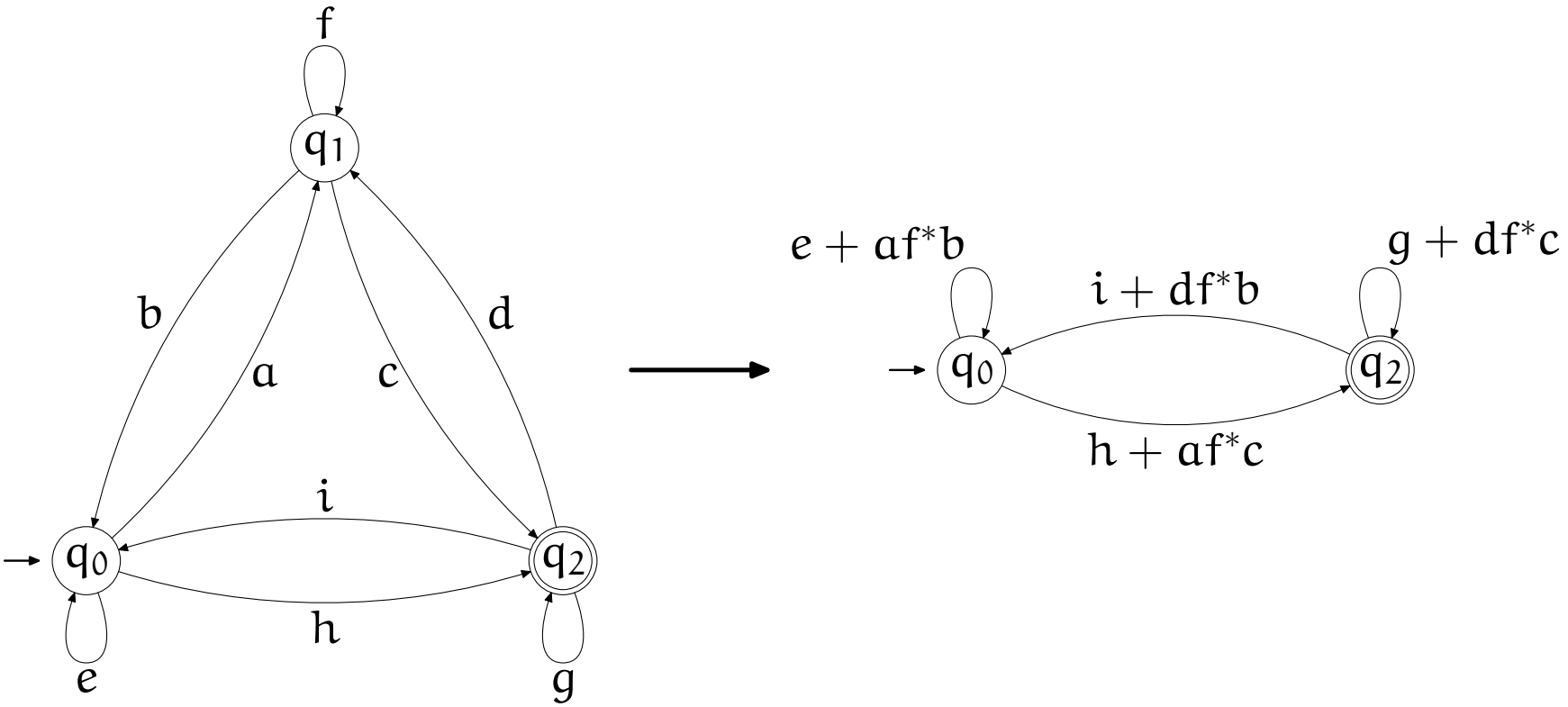
odpovídá regulárnímu výrazu

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

Příklad 12



Příklad 12



Procedura NFA$\rightarrow$RE

❶ Začni s NFA se stavy $q_0, q_1, \ldots, q_n$ a jediným koncovým stavem q_n.

❷ Převeď NFA na *úplný* zobecněný přechodový graf, v němž je mezi každými dvěma vrcholy q_i a q_j právě jedna hrana označená r_{ij}.

❸ Má-li graf pouze dva vrcholy (tj. $n = 1$), výstupem algoritmu je regulární výraz

$$r = r_{00}^* r_{01} (r_{11} + r_{10} r_{00}^* r_{01})^*.$$

❹ Má-li graf pouze tři vrcholy (tj. $n = 2$), zaveď nové označení všech hran mezi vrcholy q_i a q_i, kde $i = 0, 2$ a $j = 0, 2$:

$$r_{ij} + r_{i1} r_{11}^* r_{1j}.$$

Potom zruš vrchol q_1 se všemi jeho hranami a jdi na ❸.

❺ Má-li graf čtyři nebo více vrcholů (tj. $n \geq 3$), zvol k likvidaci libovolný vrchol q_k, kde $0 < k < n$: Pro všechny dvojice stavů (q_i, q_j), kde $i \neq k$ a $j \neq k$, aplikuj postup z kroku ❹ a podle nového počtu vrcholů pak buď přejdi na ❹ nebo opakuj ❺.

Zjednodušování regulárních výrazů

V průběhu celého algoritmu je možné a vhodné při každé agregaci regulárních výrazů v krocích ❸–❺ použít zjednodušující vzorce

$$\begin{aligned} r + \emptyset &= r, \\ r\emptyset &= \emptyset, \\ r\lambda &= r, \\ \emptyset^* &= \lambda. \end{aligned}$$

Formulace výsledku

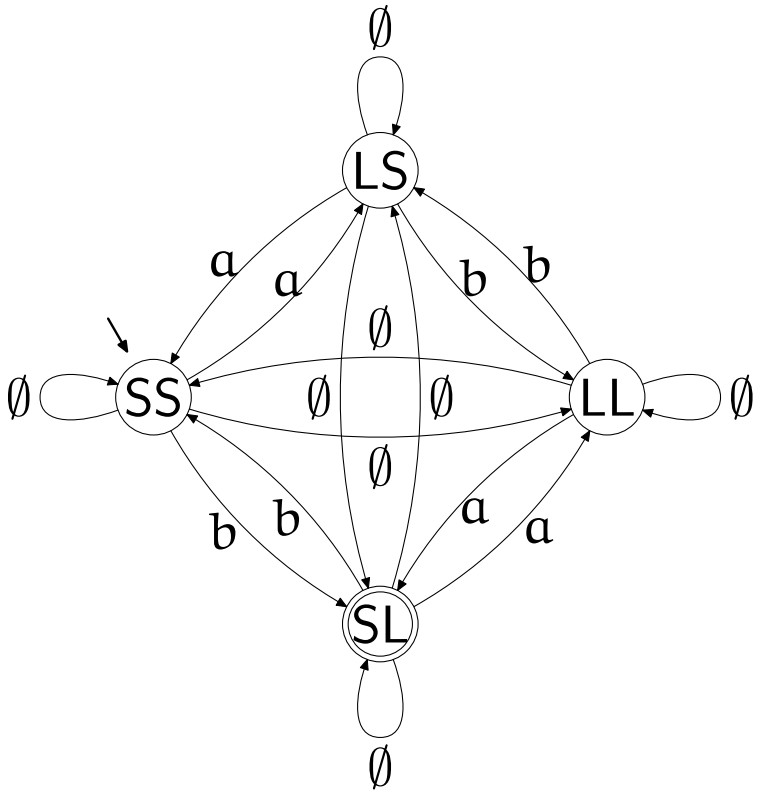
Věta 4

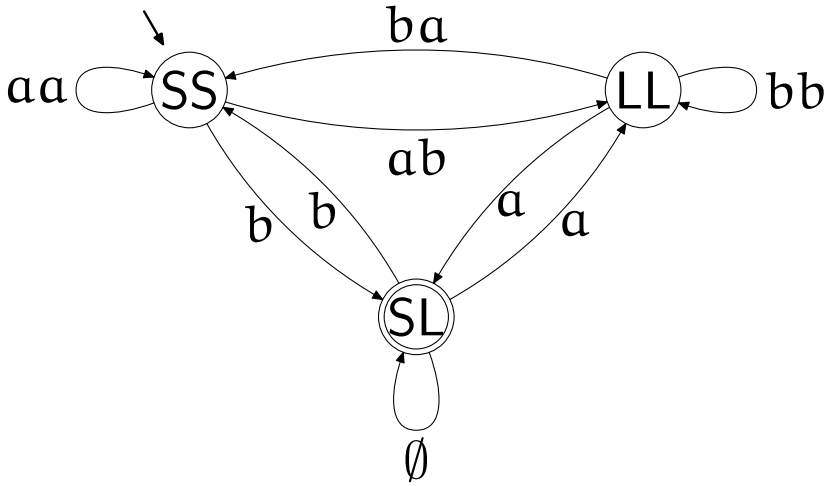
Je-li L regulární jazyk, pak existuje regulární výraz r takový, že $\mathrm{L} = \mathrm{L}(\mathrm{r})$.

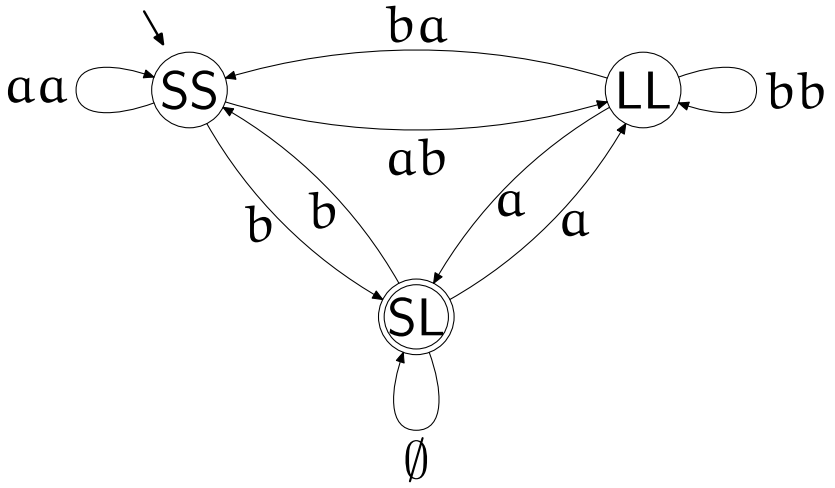
Věty 3 a 4 tedy říkají, že třída jazyků definovaných regulárním výrazem je shodná s třídou regulárních jazyků.

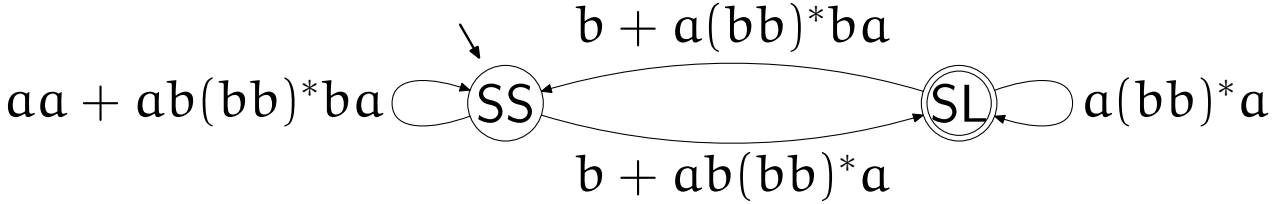
Příklad 13

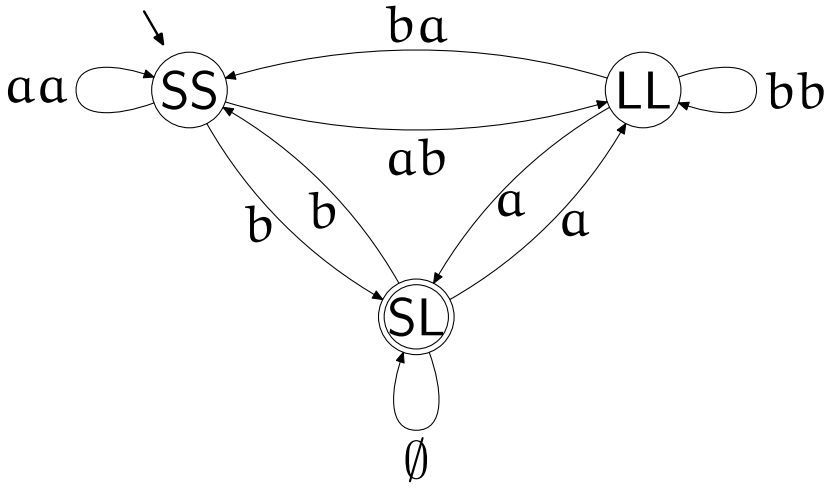
Najděte regulární výraz pro jazyk L nad abecedou $\Sigma = \{a, b\}$ obsahující slova se sudým počtem symbolů a a lichým počtem symbolů b.

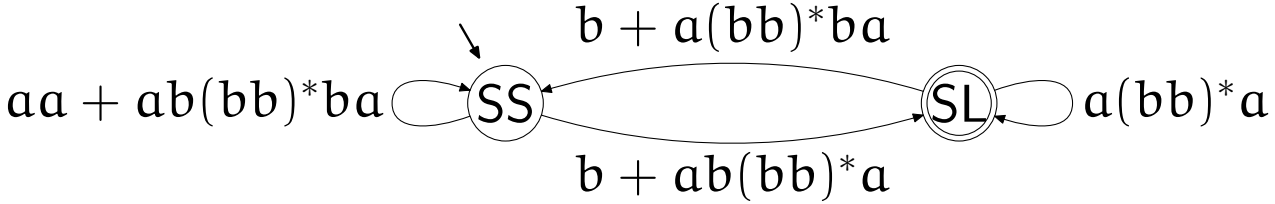



ba
aa
SS
LL
bb
ab
b
b
a
a
SL
∅


aa
SS
ba
LL
bb
ab
b
b
a
a
SL
∅


aa + ab(bb)*ba
SS
b + a(bb)*ba
SL
a(bb)*a
b + ab(bb)*a





$$(aa + ab(bb)^*ba)^*(b + ab(bb)^*a)$$
$$(a(bb)^*a + (b + a(bb)^*ba)(aa + ab(bb)^*ba)^*(b + ab(bb)^*a))^*$$

Shrnutí

Věta 5

Pro každý regulární jazyk L existuje regulární výraz r takový, že $\mathrm{L} = \mathrm{L}(\mathrm{r})$.

Gramatika

Definice 12

Gramatika G je definována jako uspořádaná čtveřice

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

kde

- V je neprázdná konečná množina *proměnných*,
- T je konečná množina *koncových (terminálních) symbolů*,
- $S \in V$ je speciální symbol zvaný *počáteční (startovní) proměnná*,
- P je konečná množina *přepisovacích pravidel (produkcí)*.

Přepisovací pravidla

Přepisovací pravidla se zapisují stylem $x \to y$, kde $x \in (V \cup T)^+$ a $y \in (V \cup T)^*$.

Používají se takto: Máme-li řetěz w ve tvaru

$$w = uxv,$$

říkáme, že produkce $x \to y$ je aplikovatelná na w a w můžeme přepsat na nový řetěz

$$z = uyv.$$

V takovém případě říkáme, že řetěz z lze *přímo odvodit* z w. Zápis: $w \Rightarrow z$.

Pokud má gramatika více produkcí, můžeme pro přepis v každém kroku použít kteroukoli z nich (nedeterminismus!). Posloupnost odvození

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

znamená, že w_n lze *odvodit* z w_1. Zápis:

$$w_1 \overset{*}{\Rightarrow} w_n.$$

Jazyk generovaný gramatikou

Množina všech řetězů, které lze odvodit z počáteční proměnné.

Definice 13

Nechť $G = (V, T, S, P)$ je gramatika. Množinu

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

nazýváme jazykem generovaným gramatikou G.

Je-li $w \in L(G)$, pak

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w$$

nazýváme *odvozením (derivací)* věty w pomocí gramatiky G. Mezistupně S, w_1, w_2 atd. se nazývají *větné formy* tohoto odvození.

Pro mnohé jazyky lze najít více gramatik, které je generují. Pokud tedy máme dvě gramatiky G_1 a G_2 takové, že $L(G_1) = L(G_2)$, říkáme, že tyto gramatiky jsou *ekvivalentní*.

Regulární gramatiky

Třetí způsob reprezentace regulárních jazyků.

Definice 14

1. Gramatika $G = (V, T, S, P)$ se nazývá *pravá lineární*, pokud jsou všechny její produkce v jednom z následujících dvou tvarů:

   $$A \to xB,$$
   $$A \to x.$$

   kde $A, B \in V$ a $x \in T^*$.

2. Gramatika se nazývá *levá lineární*, pokud jsou všechny její produkce v jednom z následujících dvou tvarů:

   $$A \to Bx,$$
   $$A \to x.$$

3. Gramatika se nazývá *regulární*, pokud je buď pravá lineární nebo levá lineární.

Příklad 14

Gramatika $G_1 = (\{S\}, \{a, b\}, S, P_1)$, kde P_1 obsahuje jedinou produkci

$$S \to abS \mid a,$$

je pravá lineární.

Gramatika $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$ s produkcemi

$$\begin{aligned} S &\to S_1 ab, \\ S_1 &\to S_1 ab \mid S_2, \\ S_2 &\to a, \end{aligned}$$

je levá lineární.

Příklad 15

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

$$S \to A,$$
$$A \to aB \mid \lambda,$$
$$B \to Ab,$$

není regulární.

Takováto gramatika se nazývá *lineární* – na pravé straně všech produkcí se může vyskytovat nejvýše jedna proměnná, ale může být kdekoli.

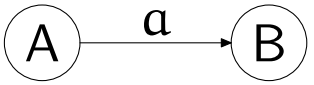
Regulární gramatiky a regulární jazyky

Věta 6

Nechť $G = (V, T, S, P)$ je pravá lineární gramatika. Potom $L(G)$ je regulární jazyk.

Zkonstruujeme NFA, který bude simulovat odvozování věty v gramatice.

$A \to aB$



Příklad 16

Zkonstruujte konečný automat přijímající jazyk generovaný gramatikou

$$V_0 \to \mathrm{a}V_1,$$
$$V_1 \to \mathrm{a}\mathrm{b}V_0 \mid \mathrm{b},$$

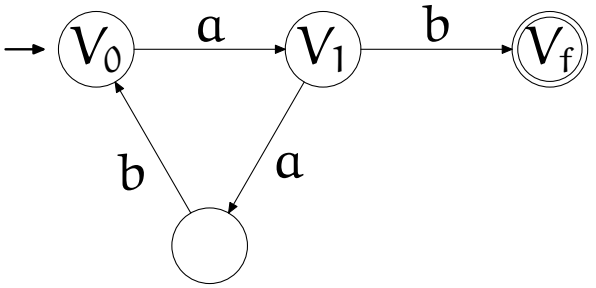
kde V_0 je počáteční proměnná.

Příklad 16

Zkonstruujte konečný automat přijímající jazyk generovaný gramatikou

$$V_0 \to aV_1,$$
$$V_1 \to abV_0 \mid b,$$

kde V_0 je počáteční proměnná.



Věta 7

Je-li L regulární jazyk nad abecedou Σ, pak existuje pravá lineární gramatika $\mathrm{G} = (V, \Sigma, S, \mathrm{P})$ taková, že $\mathrm{L} = \mathrm{L}(\mathrm{G})$.

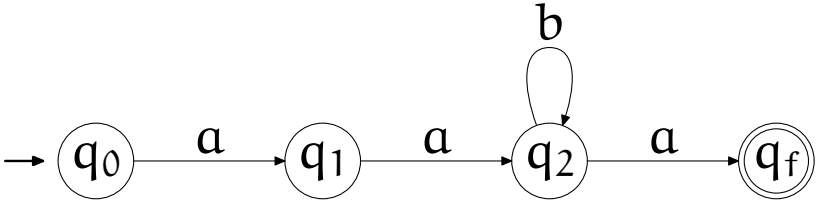
Obrácení postupu konstrukce použité u věty 4.2.

Příklad 17

Zkonstruujte pravou lineární gramatiku pro $\mathrm{L}(\mathrm{aab}^*\mathrm{a})$.

Příklad 17

Zkonstruujte pravou lineární gramatiku pro $L(aab^*a)$.



$$q_0 \to aq_1$$
$$q_1 \to aq_2$$
$$q_2 \to a \mid bq_2$$

Místo pravé lineární gramatiky lze použít i levou lineární.

Věta 8

Jazyk L je regulární právě tehdy, když existuje levá lineární gramatika G taková, že $\mathrm{L} = \mathrm{L}(\mathrm{G})$.

Souhrnné tvrzení:

Věta 9

Jazyk L je regulární právě tehdy, když existuje regulární gramatika G taková, že $\mathrm{L} = \mathrm{L}(\mathrm{G})$.

Vlastnosti regulárních jazyků

1. Které operace s regulárními jazyky dají jako výsledek regulární jazyk?
2. Můžeme rozhodnout, zda má daný jazyk určité vlastnosti?
3. Můžeme rozhodnout, jestli je jazyk regulární nebo ne?

Uzavřenost vůči základním operacím

Věta 10

Jsou-li L_1 a L_2 regulární jazyky, pak také $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 L_2$, $\overline{L_1}$, L_1^* a L_1^R jsou regulární jazyky.

Práce s regulárními jazyky

Tři základní typy otázek (pro jakýkoli jazyk):

příslušnost:	Patří daný řetěz do jazyka?
počet prvků:	Je jazyk prázdný, konečný nebo nekonečný?
rovnost jazyků:	Jsou dva zadané jazyky stejné nebo ne?

Příslušnost

Věta 11

Je-li regulární jazyk L nad abecedou Σ zadán jednou ze standardních reprezentací, potom existuje algoritmus, který pro každý řetěz $w \in \Sigma^*$ určí, zda je $w \in L$ anebo není.

Standardní reprezentace regulárního jazyka:

- konečný automat (DFA nebo NFA),
- regulární výraz
- regulární gramatika

Počet prvků

Věta 12

Existuje algoritmus, který určí, zda je daný regulární jazyk prázdný, konečný anebo nekonečný.

1. Neprázdný: V grafu DFA existuje aspoň jedna cesta z počátečního vrcholu do nějakého koncového vrcholu.
2. Nekonečný: V grafu DFA najdeme všechny cykly; pokud některý cyklus leží na cestě z počátečního do koncového vrcholu, je jazyk nekonečný, jinak je konečný.

Rovnost jazyků

Věta 13

Pro každou dvojici regulárních jazyků L_1 a L_2 existuje algoritmus, který rozhodne, zda $L_1 = L_2$ anebo ne.

$$L = (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1})$$

$L_1 = L_2$ právě když L je neprázdný.

Jak poznat **ne**regulární jazyk?

Dirichletův princip (pigeonhole principle): Máme-li rozdělit m předmětů do n přihrádek, přičemž je $m > n$, musí být alespoň v jedné přihrádce více než jeden předmět.

Příklad 18: Rozhodněte, zda je jazyk $L = \{ a^n b^n \mid n \geq 0 \}$ regulární.

Pumpovací lemma

Věta 14

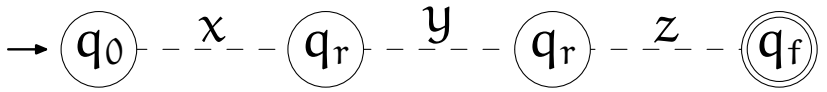
Nechť L je nekonečný regulární jazyk. Potom existuje přirozené číslo $m \in \mathbb{N}$ takové, že všechny řetězy w o délce alespoň m ($|w| \geq m$) lze napsat ve tvaru

$$w = xyz,$$

kde $|xy| \leq m$, $|y| \geq 1$ a navíc pro všechna $i \geq 0$ platí

$$w_i = xy^i z \in L.$$

Je-li počet stavů konečného automatu n, volíme $m = n + 1$.



Opakování musí nastat nejpozději v $(n + 1)$. vrcholu, takže $|xy| \leq m$.

Příklad 19

Pomocí pumpovacího lemmatu ukážeme, že následující jazyky nejsou regulární:

- $L_1 = \{ a^n b^n \mid n \geq 0 \}$
- $L_2 = \{ ww^R \mid w \in \{a, b\}^* \}$
- $L_3 = \{ w \in \{a, b\}^* \mid n_a(w) < n_b(w) \}$

  ($n_x(w)$ je počet výskytů symbolu x v řetězu w).

Použití pumpovacího lemmatu

Pumpovací lemma je jen jednosměrná implikace: „Je-li L regulární jazyk, pak …“.

Důkaz neregularity jazyka děláme sporem: předpokládáme, že zadaný jazyk je regulární a najdeme vhodný řetěz w, který po přiměřeném napumpování vypadne z jazyka.

Postup důkazu si lze představit jako hru:

1. Protivník nám zadá m.
2. Já musím najít vhodný řetěz w délky aspoň m (stačí jeden).
3. Protivník zvolí rozklad $w = xyz$.
4. Já vyberu $i \geq 0$ tak, aby $xy^iz \notin L$.

Bezkontextové jazyky

Přepisovací pravidla jsou pouze tvaru $X \to \gamma$, kde X je proměnná a γ řetěz terminálních symbolů a proměnných.

Důležité vlastnosti:

- Když se během posloupnosti přepisů někde objeví terminální symbol, už tam zůstane.
- Přepisy různých proměnných jsou navzájem nezávislé:

  Máme-li řetěz $\zeta X \eta Y \xi$, můžeme postupně přepisovat proměnné X a Y v libovolném pořadí.

Příklad 20

Nechť $\Sigma = \{a, b\}$. Následující jazyky jsou bezkontextové:

$\{ a^n b^n \mid n \geq 0 \}$ $\{ a^m b^n \mid m \neq n \}$ $\{ ww^R \mid w \in \Sigma^* \}$

Příklad 20

Nechť $\Sigma = \{a, b\}$. Následující jazyky jsou bezkontextové:

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

$S \to aSb \mid \lambda$

$\{ a^m b^n \mid m \neq n \}$

$S \to AS_1 \mid S_1B$
$S_1 \to aS_1b \mid \lambda$
$A \to aA \mid a$
$B \to bB \mid b$

$\{ ww^R \mid w \in \Sigma^* \}$

$S \to aSa \mid bSb \mid \lambda$

Příklad 21

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

1. $S \to AB$
2. $A \to aaA$
3. $A \to \lambda$
4. $B \to Bb$
5. $B \to \lambda$

Dvě různá odvození stejného řetězu aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

$$S \overset{1}{\Rightarrow} AB \overset{4}{\Rightarrow} ABb \overset{2}{\Rightarrow} aaABb \overset{5}{\Rightarrow} aaAb \overset{3}{\Rightarrow} aab$$

Levé a pravé odvození

Definice 15

Odvození nazveme *levým*, pokud se v každém kroku přepisuje vždy první (levou) proměnnou.

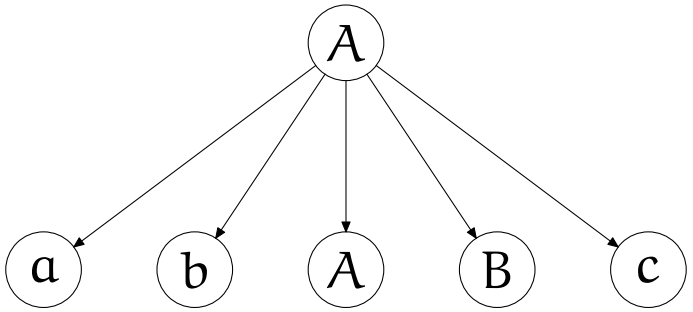
Odvození nazveme *pravým*, pokud se v každém kroku přepisuje vždy poslední (pravou) proměnnou.

Tímto způsobem lze efektivně zmenšit počet stupňů volnosti při odvozování.

Derivační strom

Odvození (derivaci) slova lze graficky znázornit pomocí *derivačního stromu* (derivation tree, parse tree). Každému použitému přepisovacímu pravidlu v něm odpovídá právě jeden vrchol označený proměnnou na levé straně s dětmi označenými symboly, které jsou na pravé straně.

Například pravidlu $A \to abABc$ odpovídá fragment stromu



Definice 16

Nechť $G = (V, T, S, P)$ je bezkontextová gramatika. *Derivačním stromem* gramatiky G nazveme orientovaný strom s těmito vlastnostmi:

1. Každý list je označen symbolem z množiny $T \cup \{\lambda\}$.
2. Každý vnitřní vrchol (tj. kromě listů) je označen proměnnou.
3. Kořen je označen počátečním proměnnou S.
4. Je-li některý vrchol označen proměnnou $A \in V$ a jeho potomci symboly (zprava doleva) $\alpha_1, \alpha_2, \ldots, \alpha_n$, musí množina P obsahovat přepisovací pravidlo
$$A \to \alpha_1 \alpha_2 \cdots \alpha_n.$$
5. List označený symbolem prázdného řetězu λ nemá sourozence.

Parciální derivační strom: nemusí platit 3 (kořen je pak označen libovolnou proměnnou) a místo 1 je splněna podmínka

1a. Každý list je označen symbolem z množiny $V \cup T \cup \{\lambda\}$.

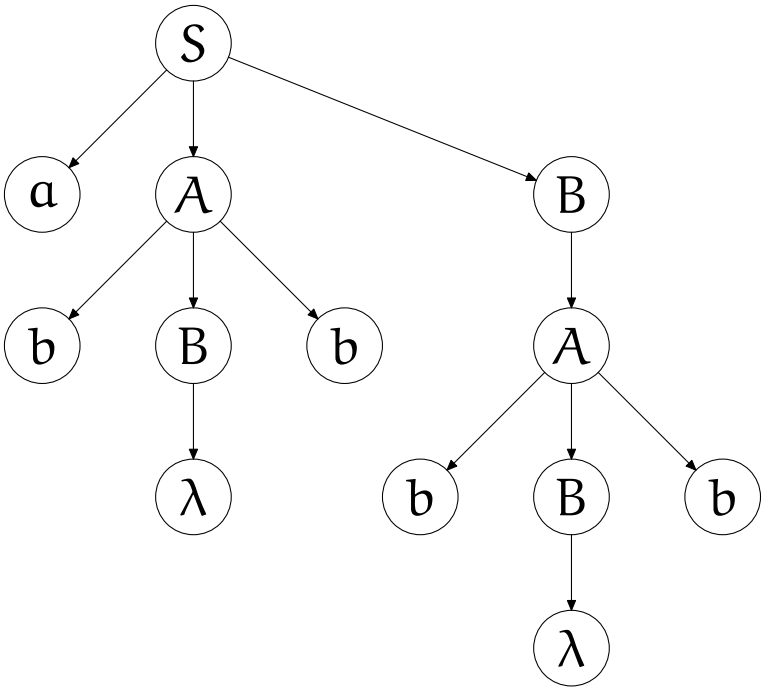
Řetěz, který dostaneme čtením všech listů derivačního stromu „zleva doprava" – listy označené λ přitom vynecháváme – se nazývá *výsledek* derivačního stromu.

„zleva doprava" = prohledávání do hloubky (depth-first), přičemž ze všech dosud neprozkoumaných sourozenců vždy bereme toho nejvíc vlevo.

Příklad 22: Zkonstruujeme derivační strom pro gramatiku s pravidly

$$S \to aAB$$
$$A \to bBb$$
$$B \to A \mid \lambda$$

a řetěz $abbbb$.


S
a
A
B
b
B
b
A
λ
b
B
b
λ

Vztah gramatiky a derivačního stromu

Věta 15

Nechť $G = (V, T, S, P)$ je bezkontextová gramatika. Potom pro každé $w \in L(G)$ existuje derivační strom gramatiky G, jehož výsledkem je w.

Obráceně, výsledkem libovolného derivačního stromu je věta patřící do $L(G)$.

Navíc, je-li T_G nějaký parciální derivační strom gramatiky G, jehož kořenem je S, pak výsledkem stromu T_G je větnou formou gramatiky G.

Syntaktická analýza (parsing)

Teorie formálních jazyků má největší uplatnění při návrhu programovacích jazyků. Rozlišují se *syntaktická* a *sémantická* pravidla.

```
ckar c;
c = 3.14;
```

Pro syntaktická pravidla se využívají (bezkontextové) gramatiky (Backus-Naurova Forma), pro sémantická zatím žádný obecně přijatý formalismus neexistuje.

Pro syntaktickou analýzu je třeba kromě určení, zda $w \in \mathrm{L}(\mathrm{G})$, také najít odvození $S \overset{*}{\Rightarrow} w$, tj. určit, z jakých syntaktických celků se w skládá (klíčové slovo, operátor, číslo, ...).

Analýza všech možností

1. Vezmeme všechna pravidla tvaru $S \to x_1$, kde $x_1 \in (V \cup T)^*$.
2. Pro všechna taková x_1 vezmeme všechna pravidla tvaru $A_1 \to x_2$, kde A_1 je první proměnná v x_1 (nejvíc vlevo).
3. ...

Pokud se některou cestou dostaneme až ke slovu w, máme hledané (levé) odvození. Dospějeme-li k řetězu, z něhož určitě nelze w odvodit, větev ukončíme.

Tento přístup se též nazývá analýza hrubou silou. Jde o typ analýzy *shora dolů*.

Příklad 23:

Najdeme odvození slova $aabb$ v gramatice

$$S \to SS \mid aSb \mid bSa \mid \lambda.$$

Nevýhoda #1: hledání nemusí skončit

Pokud $w \notin L(G)$, můžeme vytvářet stále delší cesty a nikdy nedospět k cíli – v příkladu 23 se to stane s řetězcem abb.

Tento problém nenastane, pokud gramatiku upravíme tak, aby neobsahovala žádná vypouštějící a jednotková pravidla. V příkladu 23 dostaneme:

$$S \to SS \mid aSb \mid bSa \mid ab \mid ba$$

Věta 16

Nechť $G = (V, T, S, P)$ je bezkontextová gramatika, která neobsahuje žádná pravidla tvaru $A \to \lambda$ ani $A \to B$, kde $A, B \in V$. Algoritmus analýzy všech možností je pak schopen pro každý řetěz $w \in T^*$ rozhodnout, zda w patří anebo nepatří do $L(G)$.

Délka libovolné cesty je maximálně $2|w|$.

Nevýhoda #2: exponenciální složitost

V každém kroku algoritmu můžeme v nejhorším případě použít všechna přepisovací pravidla gramatiky. V prvním kroku tak můžeme mít až $|\mathrm{P}|$ větví, ve druhém $|\mathrm{P}|^2$, ve druhém $|\mathrm{P}|^3$, atd.

Celý algoritmus tedy může posuzovat až

$$|\mathrm{P}| + |\mathrm{P}|^2 + |\mathrm{P}|^3 + \cdots + |\mathrm{P}|^{2|w|} = |\mathrm{P}|\frac{|\mathrm{P}|^{2|w|} - 1}{|\mathrm{P}| - 1}.$$

řetězů. S rostoucím $|w|$ tedy narůstá složitost algoritmu až exponenciálně. Obvykle to není tak zlé, ale algoritmus v žádném případě není efektivní.

Věta 17

Pro každou bezkontextovou gramatiku existuje algoritmus, který libovolné slovo $|w|$ syntakticky zanalyzuje v počtu kroků úměrném $|w|^3$.

To už je lepší, chtěli bychom ale pokud možno *lineární* složitost.

s-gramatika

Definice 17

Bezkontextová gramatika $G = (V, T, S, P)$ se nazývá *jednoduchá* neboli *s-gramatika*, pokud jsou všechna přepisovací pravidla tvaru

$$A \to ax$$

kde $A \in V$, $a \in T$ a $x \in V^*$, a navíc se každá taková dvojice (A, a) vyskytuje v P nejvýše jednou.

Přepisujeme-li odleva, máme vždy jen jednu možnost a v každém kroku přibude v levé části jeden terminál – složitost je tedy lineární.

Klíčová slova v programovacích jazycích proto významně zjednodušují syntaktickou analýzu, např.

$$\langle \text{příkaz while} \rangle ::= \textbf{while} \langle \text{podmínka} \rangle \langle \text{příkaz} \rangle$$

Nejednoznačnost gramatik

Může se stát, že pro nějaký řetěz existuje několik (mnoho) různých odvození. To někdy nevadí, někdy to ale může změnit význam řetězu.

Definice 18

Bezkontextová gramatika G se nazývá *nejednoznačná (ambiguous)*, pokud existuje aspoň jeden řetěz $w \in \mathrm{L}(\mathrm{G})$, který má dva nebo více různých derivačních stromů.

Příklad 24

Uvažujme gramatiku

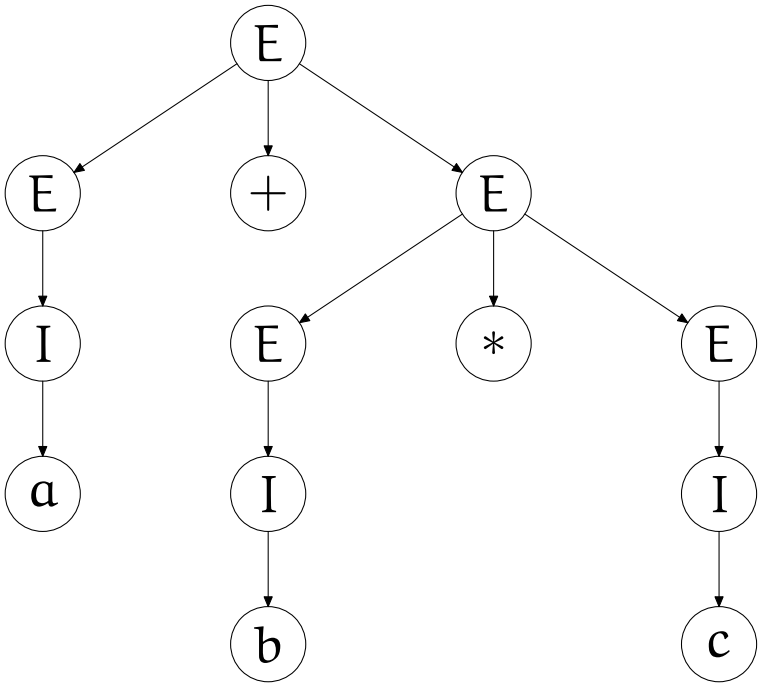
$$G = (\{E, I\}, \{a, b, c, +, *, (,)\}, E, P)$$

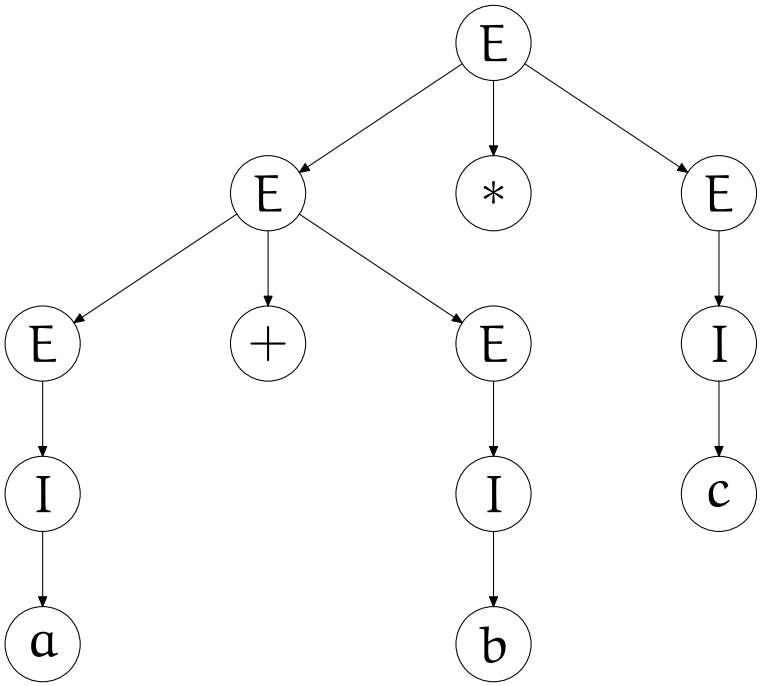
s těmito přepisovacími pravidly:

$$\begin{aligned} E &\to I, \\ E &\to E + E, \\ E &\to E * E, \\ E &\to (E), \\ I &\to a \mid b \mid c. \end{aligned}$$

Tato gramatika je nejednoznačná.

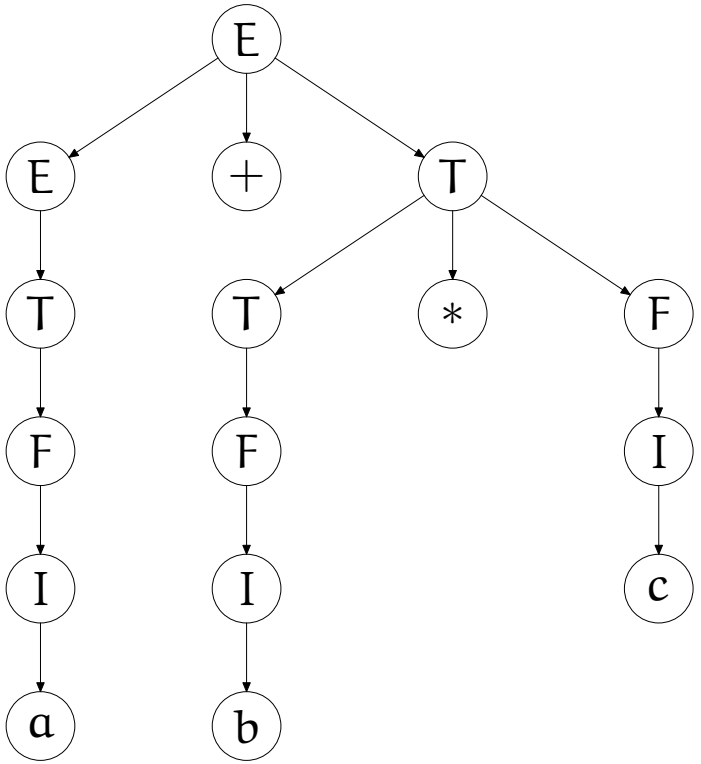
Dvě odvození výrazu a + b ∗ c





Jednoznačná verze

$$
\begin{aligned}
E &\to T,\\
T &\to F,\\
F &\to I,\\
E &\to E + T,\\
T &\to T * F,\\
F &\to (E),\\
I &\to a \mid b \mid c.
\end{aligned}
$$



Definice 19

Bezkontextový jazyk L se nazývá *jednoznačný*, pokud existuje alespoň jedna jednoznačná gramatika G generující jazyk L.

Pokud žádná taková gramatika neexistuje, nazývá se jazyk L *vnitřně nejednoznačný (inherently ambiguous)*.

Příklad 25: Tento jazyk je vnitřně nejednoznačný:

$$L = \{ a^n b^n c^m \} \cup \{ a^n b^m c^m \}$$

Všechny řetězy tvaru $a^n b^n c^n$ mají dvojí odvození.

Zjednodušování bezkontextových gramatik

Bezkontextové gramatiky mohou mít na pravé straně obecně jakoukoli kombinaci proměnných a terminálů. Některé typy pravidel ale někdy působí problémy (viz pravidla $A \to \lambda$ a $A \to B$ u parsingu hrubou silou) – jak se ukazuje, některých takových problematických pravidel se lze zbavit a zkonstruovat ekvivalentní gramatiku, která je neobsahuje.

Dvě tzv. *normální formy* gramatik:

- Chomskyho normální forma (CNF)
- Normální forma Greibachové (GNF)

λ-free jazyky

Pokud jazyk L obsahuje λ a $G = (V, T, S, P)$ je gramatika pro $L - \{\lambda\}$, pak gramatiku pro L dostaneme jednoduše přidáním pravidla

$$S_0 \to S \mid \lambda.$$

Všechny podstatné vlastnosti jazyka $L - \{\lambda\}$ lze proto přenést i na L. Navíc si ukážeme, že z libovolné bezkontextové gramatiky G lze odvodit gramatiku $\hat{G}$ takovou, že

$$L(\hat{G}) = L(G) - \{\lambda\}.$$

Proto budeme nadále pracovat s λ-free jazyky (nebude-li řečeno jinak).

Jednoduché substituční pravidlo

Věta 18

Nechť $G = (V, T, S, P)$ je bezkontextová gramatika. Předpokládejme, že P obsahuje produkci ve tvaru

$$A \to x_1 B x_2, \tag{1}$$

přičemž A a B jsou *různé* proměnné. Nechť dále

$$B \to y_1 \mid y_2 \mid \cdots \mid y_n$$

jsou všechny produkce s proměnnou B na levé straně.

Potom gramatika $\hat{G} = (V, T, S, \hat{P})$ vzniklá z G vyjmutím produkce (1) a přidáním produkcí

$$A \to x_1 y_1 x_2 \mid x_1 y_2 x_2 \mid \cdots x_1 y_n x_2$$

je ekvivalentní s G, tj. $L(\hat{G}) = L(G)$.

Příklad 26

V gramatice $G = (\{A, B\}, \{a, b, c\}, A, P)$ s produkcemi

$$A \to a \mid aaA \mid abBc,$$
$$B \to abbA \mid b$$

substituujeme proměnnou B:

$$A \to a \mid aaA \mid ababbAc \mid abbc,$$
$$B \to abbA \mid b$$

Odstranění zbytečných produkcí

Definice 20

Nechť $G = (V, T, S, P)$ je bezkontextová gramatika. Řekneme, že proměnná $A \in V$ je *užitečná*, pokud existuje aspoň jedna věta $w \in L(G)$ taková, že

$$S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w,$$

kde $x, y \in (V \cup T)^*$.

Proměnnou, která není užitečná, nazýváme *zbytečná*.

Druhý důvod zbytečnosti proměnné: nelze z ní odvodit žádnou větu:

$$S \to aSb \mid \lambda \mid A,$$
$$A \to aA.$$

Odstranění zbytečných produkcí a proměnných

Věta 19

Ke každé bezkontextové gramatice $G = (V, T, S, P)$ existuje ekvivalentní bezkontextová gramatika $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$, která neobsahuje žádné zbytečné proměnné a produkce.

Konstrukce $\hat{G}$ ve dvou krocích:

1. Nejprve zkonstruujeme gramatiku $G_1 = (V_1, T_1, S, P_1)$, která bude obsahovat jen takové symboly A, z nichž je možné odvodit nějakou větu, tj.
$$A \overset{*}{\Rightarrow} w \in T^*.$$
2. Výslednou gramatiku $\hat{G}$ dostaneme tak, že z G_1 vyřadíme všechny proměnné, které nejsou dosažitelné z počáteční proměnné S.

Algoritmus pro 1. krok

❶ $V_1 := \emptyset$

❷ Opakuj tento krok tak dlouho, dokud lze do V_1 přidat nějakou proměnnou A: Má-li $A \in V$ produkci ve tvaru

$$A \to x_1 x_2 \cdots x_n,$$

přičemž $x_i \in V_1 \cup T$ pro všechna $i = 1, 2, \ldots, n$, pak přidej A do V_1.

❸ Novou množinu produkcí P_1 vytvoř z P ponecháním jen těch pravidel, v nichž jsou všechny symboly ve $V_1 \cup T$.

2. krok

Vytvoříme graf závislostí proměnných v množině V_1 na počátečním symbolu S a vyhodíme všechny proměnné, které v tomto grafu nebudou. Tím dostaneme výslednou množinu proměnných $\hat{V}$.

V množině $\hat{P}$ pak ponecháme jen produkce s proměnnými z $\hat{V}$.

Příklad 27

Odstraňte zbytečné proměnné a produkce z gramatiky $G = (V, T, S, P)$, kde $V = \{S, A, B, C\}$, $T = \{a, b\}$ a P sestává z těchto produkcí:

$$\begin{aligned} S &\to aS \mid A \mid C, \\ A &\to a, \\ B &\to aa, \\ C &\to aCb, \end{aligned}$$

Příklad 27

Odstraňte zbytečné proměnné a produkce z gramatiky $G = (V, T, S, P)$, kde $V = \{S, A, B, C\}$, $T = \{a, b\}$ a P sestává z těchto produkcí:

$$\begin{aligned} S &\to aS \mid A \mid C, \\ A &\to a, \\ B &\to aa, \\ C &\to aCb, \end{aligned}$$

Výsledek:

$$\begin{aligned} S &\to aS \mid A, \\ A &\to a. \end{aligned}$$

Odstranění λ-produkcí

Definice 21

Produkční pravidlo ve tvaru $A \to \lambda$ nazýváme λ-*produkce*.

Proměnná A se nazývá *vynulovatelná (nullable)*, pokud pro ni existuje odvození $A \overset{*}{\Rightarrow} \lambda$.

Příklad 28: Odstraňte λ-produkci z gramatiky

$$S \to aS_1b,$$
$$S_1 \to aS_1b \mid \lambda.$$

Věta 20

Nechť G je bezkontextová gramatika taková, že $\lambda \notin L(G)$. Potom existuje ekvivalentní gramatika $\hat{G}$, která nemá λ-produkce.

Nejprve najdeme množinu V_N všech vynulovatelných proměnných:

1. $V_N := \{ A \in V \mid A \Rightarrow \lambda \}$
2. Opakuj tento krok tak dlouho, dokud lze nějakou proměnnou přidat do V_N: Pro všechny produkce tvaru

   $$B \to A_1 A_2 \cdots A_n,$$

   kde $A_i \in V_N$ pro $i = 1, 2, \ldots, n$ přidej B do V_N.

Dále každé produkční pravidlo nahradíme sadou pravidel, v nichž budou postupně vyjmuty všechny možné kombinace vynulovatelných proměnných (nesmí ale na pravé straně zbýt λ).

λ-produkce vyhodíme.

Příklad 29

Odstraňte λ-produkce z gramatiky

$$
\begin{aligned}
S &\to ABaC,\\
A &\to BC,\\
B &\to b \mid \lambda,\\
C &\to D \mid \lambda,\\
D &\to d.
\end{aligned}
$$

Výsledek:

$$
\begin{aligned}
S &\to ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a,\\
A &\to B \mid C \mid BC,\\
B &\to b,\\
C &\to D,\\
D &\to d.
\end{aligned}
$$

Jednotkové produkce

Definice 22

Produkční pravidlo bezkontextové gramatiky ve tvaru

$$A \to B,$$

kde $A, B \in V$ se nazývá *jednotkové pravidlo*.

Věta 21

Nechť $G = (V, T, S, P)$ je bezkontextová gramatika bez λ-produkcí. Potom existuje bezkontextová gramatika $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$, která je ekvivalentní s G a neobsahuje žádné jednotkové produkce.

Důkaz využívá větu 6.3.

Příklad 30

Eliminujte jednotkové produkce z gramatiky

$$\begin{aligned} S &\to Aa \mid B, \\ B &\to A \mid bb, \\ A &\to a \mid bc \mid B. \end{aligned}$$

Výsledek:

$$\begin{aligned} S &\to a \mid bc \mid bb \mid Aa, \\ B &\to a \mid bb \mid bc, \\ A &\to a \mid bb \mid bc. \end{aligned}$$

Souhrnný výsledek

Věta 22

Nechť L je bezkontextový jazyk neobsahující λ. Potom existuje bezkontextová gramatika, která generuje L a přitom neobsahuje žádné zbytečné proměnné a produkce, λ-produkce ani jednotkové produkce.

Postup:

1. Odstraníme λ-produkce.
2. Odstraníme jednotkové produkce.
3. Odstraníme zbytečné proměnné a produkce.

Chomskyho normální forma

Definice 23

Řekneme, že bezkontextová gramatika je v Chomskyho normální formě, pokud jsou všechna její pravidla v jednom z následujících tvarů:

$$A \to BC$$

nebo

$$A \to a,$$

kde $A, B, C \in V$ a $a \in T$.

CNF redukuje a přesně určuje počet symbolů na pravé straně (dvě proměnné anebo jeden terminál).

Věta 23

Ke každé bezkontextové gramatice $G = (V, T, S, P)$ takové, že $\lambda \notin L(G)$, existuje s ní ekvivalentní gramatika $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$, která je v Chomskyho normální formě.

Postup ve dvou krocích:

1. Pouze pravidla tvaru $A \to C_1 C_2 \cdots C_n$ anebo $A \to a$.
2. Pravidla v CNF.

Příklad 31

Převeďte následující gramatiku do CNF:

$$
\begin{aligned}
S &\to ABa,\\
A &\to aab,\\
B &\to Ac.
\end{aligned}
$$

Výsledek:

$$
\begin{aligned}
S &\to AD_1,\\
D_1 &\to BB_a,\\
A &\to B_aD_2,\\
D_2 &\to B_aB_b,\\
B &\to AB_c,\\
B_a &\to a,\\
B_b &\to b,\\
B_c &\to c.
\end{aligned}
$$

Normální forma Greibachové

Definice 24

Řekneme, že bezkontextová gramatika je v normální formě Greibachové, pokud jsou všechna její pravidla ve tvaru

$$A \to ax,$$

kde $a \in T$ a $x \in V^*$.

Každé pravidlo má na pravé straně jen jeden terminál, a to nejvíce vlevo. To je stejné jako u s-gramatiky, u GNF ovšem obecně nemusíme mít pouze jedno pravidlo pro každou dvojici (A, a).

Věta 24

Ke každé bezkontextové gramatice G takové, že $\lambda \notin L(G)$, existuje s ní ekvivalentní gramatika $\hat{G}$, která je v normální formě Greibachové.

Příklad 32

Převeďte následující gramatiku do GNF:

$$S \to abSb \mid aa$$

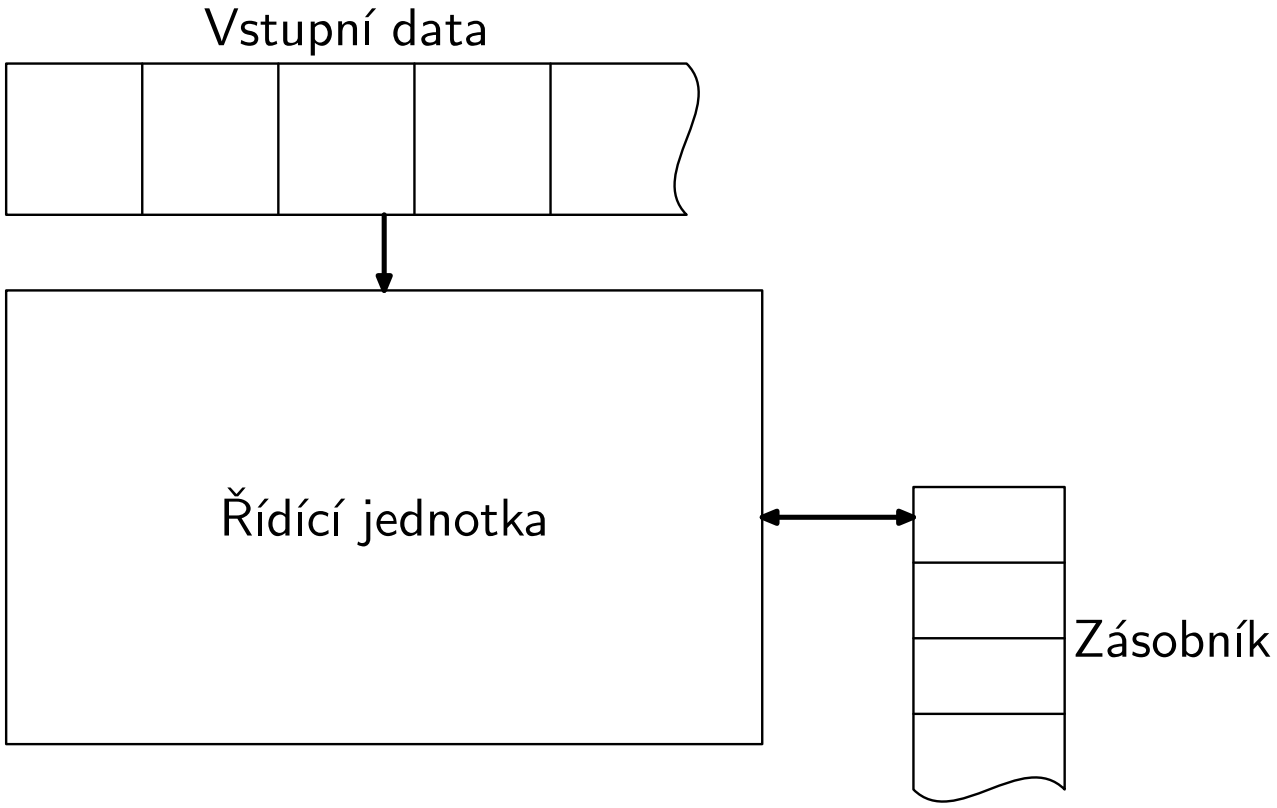
Výsledek:

$$\begin{aligned} S &\to aBSB \mid aA, \\ A &\to a, \\ B &\to b. \end{aligned}$$

Zásobníkový automat

Hledáme automat, který bude schopen rozeznávat bezkontextový jazyk.

Zásobníkový automat (push-down automaton, PDA)



Definice 25

Nedeterministický zásobníkový automat je uspořádaná sedmice

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$

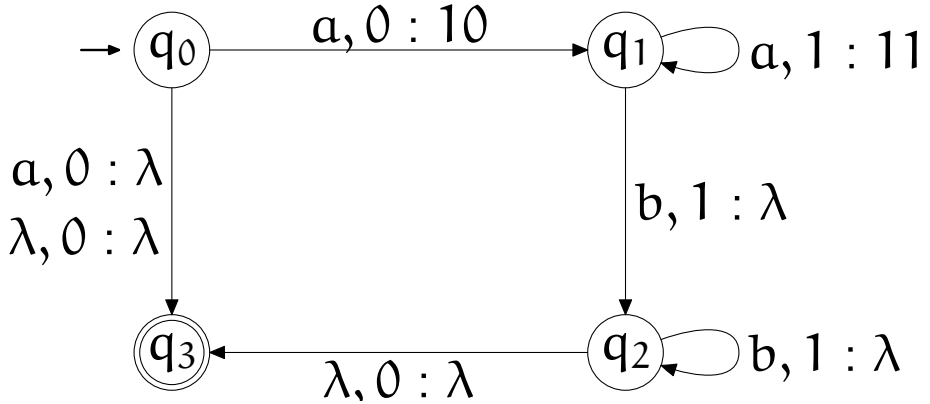
kde

- Q je konečná *množina stavů* řídící jednotky;
- Σ je konečná *vstupní abeceda*;
- Γ je konečná množina symbolů – *abeceda zásobníku*;
- δ je *přechodová funkce* – funkce z $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ do množiny všech konečných podmnožin kartézského součinu $Q \times \Gamma^*$;
- $q_0 \in Q$ je *počáteční stav* řídící jednotky;
- $z \in \Gamma$ je *počáteční symbol* zásobníku;
- $F \subset Q$ je *množina koncových stavů* řídící jednotky.

Příklad 33

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{0, 1\}$$
$$z = 0$$
$$F = \{q_3\}$$

$$\delta(q_0, a, 0) = \{(q_1, 10), (q_3, \lambda)\}$$
$$\delta(q_0, \lambda, 0) = \{(q_3, \lambda)\}$$
$$\delta(q_1, a, 1) = \{(q_1, 11)\}$$
$$\delta(q_1, b, 1) = \{(q_2, \lambda)\}$$
$$\delta(q_2, b, 1) = \{(q_2, \lambda)\}$$
$$\delta(q_2, \lambda, 0) = \{(q_3, \lambda)\}$$



$$L = \{ a^n b^n \mid n \in N_0 \} \cup \{a\}$$

Zápis přechodů zásobníkového automatu

Přechodový diagram je názorný, ale pro teoretické úvahy (důkazy) je přehlednější zápis, který vychází z toho, že uspořádaná trojice

$$(q, w, u)$$

(*konfigurace* NPDA) plně určuje stav PDA, kde w je zbývající část vstupních dat a u je obsah zásobníku. Přechod mezi dvěma stavy daný přechodovou funkcí δ pak zapisujeme pomocí relace $\vdash$, např.

$$(q_1, aw, bx) \vdash (q_2, w, yx)$$

platí pouze v případě, že

$$(q_2, y) \in \delta(q_1, a, b).$$

Přechody zahrnující libovolný počet kroků se označují pomocí $\overset{*}{\vdash}$. Tedy

$$(q_1, w_1, x_1) \overset{*}{\vdash} (q_2, w_2, x_2)$$

znamená, že automat *může* přejít z prvního stavu do druhého v libovolném (konečném) počtu kroků.

Jazyk rozpoznávaný zásobníkovým automatem

Definice 26

Nechť $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ je nedeterministický zásobníkový automat. *Jazyk rozpoznávaný automatem* M je množina

$$L(M) = \{ w \in \Sigma^* \mid (q_0, w, z) \overset{*}{\vdash} (p, \lambda, u), p \in F, u \in \Gamma^* \}$$

Příklad 34

Sestrojíme NPDA pro jazyk $L = \{ w \in \{a, b\}^* \mid n_a(w) = n_b(w) \}$

Příklad 34

Sestrojíme NPDA pro jazyk $L = \{ w \in \{a, b\}^* \mid n_a(w) = n_b(w) \}$

$$Q = \{q_0, q_1\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{z, 0, 1\}$$
$$F = \{q_1\}$$

$$\delta(q_0, a, z) = \{q_0, 0z\}$$
$$\delta(q_0, b, z) = \{q_0, 1z\}$$
$$\delta(q_0, a, 0) = \{q_0, 00\}$$
$$\delta(q_0, b, 0) = \{q_0, \lambda\}$$
$$\delta(q_0, a, 1) = \{q_0, \lambda\}$$
$$\delta(q_0, b, 1) = \{q_0, 11\}$$
$$\delta(q_0, \lambda, z) = \{q_1, \lambda\}$$

NPDA pro bezkontextový jazyk

Problém: Jak zkonstruovat k danému bezkontextovému jazyku L zásobníkový automat, který jazyk L rozpoznává? Tedy jak ze zadané bezkontextové gramatiky vytvořit „program" pro zásobníkový automat?

Usnadníme si úlohu předpokladem, že gramatika je v GNF – víme, že do ní lze libovolnou bezkontextovou gramatiku λ-free jazyka převést (pravidlo $S \to \lambda$ vyřešíme zvlášť). Máme tedy pouze pravidla tvaru $A \to ax$, kde $a \in T$ je terminál a $x \in V^*$ řetěz proměnných (může být prázdný).

Napřed vložíme S do zásobníku a dále simulujeme přepisovací pravidla aplikovaná vždy na *nejlevější* proměnnou v odvozovaném slově (= *levé odvození*):

Máme-li na vrcholu zásobníku proměnnou A a na vstupu znak a, vybereme nějaké pravidlo tvaru

$$A \to ax$$

a proměnnou A v zásobníku nahradíme řetězem proměnných x.

Příklad 35

Zkonstruujeme NPDA rozpoznávající jazyk s gramatikou

$$S \to aSbb \mid a.$$

$$S \to aSA \mid a$$
$$A \to bB$$
$$B \to b$$

$$Q = q_0, q_1, q_2$$
$$\Gamma = V \cup \{z\}$$
$$F = \{q_2\}$$

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$$
$$\delta(q_1, a, S) = \{(q_1, SA), (q_1, \lambda)\}$$
$$\delta(q_1, b, A) = \{(q_1, B)\}$$
$$\delta(q_0, b, B) = \{q_1, \lambda\}$$
$$\delta(q_1, \lambda, z) = \{q_2, z\}$$

Věta 25

Pro libovolný bezkontextový jazyk L existuje nedeterministický zásobníkový automat M takový, že

$$L = L(M).$$

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

$$M = (\{q_0, q_1, q_f\}, T, V \cup \{z\}, \delta, q_0, z, \{q_f\})$$

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$$

$$A \to \alpha x \iff (q_1, x) \in \delta(q_1, a, A)$$

$$\delta(q_1, \lambda, z) = \{(q_f, z)\}$$

Je-li $\lambda \in L$, přidáme přechod

$$\delta(q_0, \lambda, z) = \{(q_f, z)\}.$$

Příklad 36

Zkonstruujeme NPDA pro gramatiku

$$
\begin{aligned}
S &\to aA \\
A &\to aABC \mid bB \mid a \\
B &\to b \\
C &\to c
\end{aligned}
$$

$$
\begin{aligned}
\delta(q_0, \lambda, z) &= \{(q_1, Sz)\} \\
\delta(q_1, \lambda, z) &= \{(q_f, z)\} \\
\delta(q_1, a, S) &= \{(q_1, A)\} \\
\delta(q_1, a, A) &= \{(q_1, ABC), (q_1, \lambda)\} \\
\delta(q_1, b, A) &= \{(q_1, B)\} \\
\delta(q_1, b, B) &= \{(q_1, \lambda)\} \\
\delta(q_1, c, C) &= \{(q_1, \lambda)\}
\end{aligned}
$$

Postup bez GNF

Zásobníkový automat lze zkonstruovat přímo, i když nemáme gramatiku v GNF:

$$\Gamma = V \cup T \cup \{z\}$$

1. Pro produkce typu $A \to Bx$ odebereme A ze zásobníku a vložíme tam Bx.
2. Pro produkce typu $A \to abCx$ musíme napřed porovnat prefix složený z terminálů (ab) se stejným obsahem zásobníku a potom A nahradíme v zásobníku řetězem Cx.

Bezkontextové gramatiky pro NPDA

Věta 26

Je-li M nedeterministický zásobníkový automat, pak $L(M)$ je bezkontextový jazyk.

Konstrukce odpovídající bezkontextové gramatiky je poměrně komplikovaná, protože kromě terminálů odpovídajících symbolům na vstupu a zbytku větné formy v zásobníku je činnost NPDA řízena také vnitřními stavy.

Deterministický zásobníkový automat (DPDA)

DPDA nesmí mít nikdy možnost volby dvou nebo více různých tahů.

Definice 27

Řekneme, že zásobníkový automat $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ je *deterministický*, pokud platí všechny podmínky pro NPDA (Definice 7.4) a navíc tato dvě omezení:

1. $\delta(q, a, b)$ obsahuje nejvýše jeden prvek pro všechna $q \in Q$, $a \in \Sigma$ a $b \in \Gamma$;
2. Je-li $\delta(q, \lambda, c)$ neprázdné, musí být $\delta(q, a, b) = \emptyset$ pro všechna $a \in \Sigma$.

Na rozdíl od definice DFA jsou zde hodnoty přechodové funkce nadále množinové, protože chceme zachovat λ-přechody – tím žádný nedeterminismus nevzniká, protože vždy odebíráme symbol ze zásobníku.

Deterministický bezkontextový jazyk

Definice 28

Bezkontextový jazyk L se nazývá *deterministický*, pokud existuje DPDA M takový, že $L = L(M)$.

Příklad 37:

- Jazyk $\{ a^n b^n \mid n \geq 0 \}$ je deterministický.
- Jazyk $\{ ww^R \mid w \in \Sigma^+ \}$ *není* deterministický.

Existují tedy nedeterministické bezkontextové jazyky, takže třídy automatů NPDA a DPDA nejsou stejné (NPDA toho umí víc).

LL gramatiky

Deterministické bezkontextové jazyky lze velmi dobře syntakticky analyzovat pomocí DPDA. Vzhledem k možnosti λ-přechodů (bez načtení vstupního symbolu) nelze sice tvrdit, že složitost analýzy je lineární, ale je výrazně efektivnější než pro obecné bezkontextové jazyky.

Požadavek efektivní syntaktické analýzy vede k požadavku, abychom v každém kroku odvozování (např. levého) věděli přesně, jaké přepisovací pravidlo použít. Splňují ho s-gramatiky, ty jsou ale příliš omezující.

LL gramatiky: následující přepisovací pravidlo můžeme přesně určit, pokud se podíváme na omezenou část vstupního řetězu (příští symbol + pevný počet dalších). První L: vstup čteme odleva, druhé L: děláme levé odvození.

LL(k): díváme se na k vstupních symbolů, speciálně u LL(1) jen na příští.

Příklad 38:

$S \to aSb \mid ab$ je LL(2) gramatika.

Příklad 39

$$S \to SS \mid aSb \mid ab$$

Toto není LL(k) gramatika pro žádné k. Můžeme ale zkonstruovat ekvivalentní LL(1) gramatiku.

Gramatika

$$S \to aSbS \mid \lambda$$

už téměř vyhovuje: a vybírá první pravidlo, b nebo konec vstupního řetězu druhé. Pouze musíme odstranit prázdné slovo:

$$S_0 \to aSbS$$
$$S \to aSbS \mid \lambda$$

Definice 29

Bezkontextovou gramatiku $\mathcal{G} = (V, \Sigma, S, P)$ nazýváme gramatikou LL(k), pro $k \in \mathbb{N}$, pokud pro každou dvojici odvození

$$S \overset{*}{\Rightarrow} w_1 A x_1 \Rightarrow w_1 y_1 x_1 \overset{*}{\Rightarrow} w_1 w_2,$$
$$S \overset{*}{\Rightarrow} w_1 A x_2 \Rightarrow w_1 y_2 x_2 \overset{*}{\Rightarrow} w_1 w_3,$$

kde $w_1, w_2, w_3 \in \Sigma^*$, $A \in V$ a $x_1, x_2, y_1, y_2 \in (V \cup \Sigma)^*$, platí: Je-li k nejlevějších symbolů ve slovech w_2 a w_3 shodných, musí být $y_1 = y_2$.

Řada programovacích jazyků je definována pomocí LL gramatik a jejich kompilátory využívají LL analyzátory.

Jiným užitečným typem jsou LR gramatiky, které realizují analýzu *zdola nahoru*.

Pumpovací lemma pro bezkontextové jazyky

Věta 27

Nechť L je nekonečný bezkontextový jazyk. Potom existuje kladné číslo $m \in \mathbb{N}$ takové, že každý řetěz $w \in L$ s délkou $|w| \geq m$ lze rozložit do tvaru

$$w = uvxyz,$$

kde

$$|vxy| \leq m$$

a

$$|vy| \geq 1$$

a pro všechna $i = 0, 1, 2, \ldots$ platí

$$uv^i xy^i z \in L.$$

Idea důkazu

Budeme se zabývat jazykem $L - \{\lambda\}$.

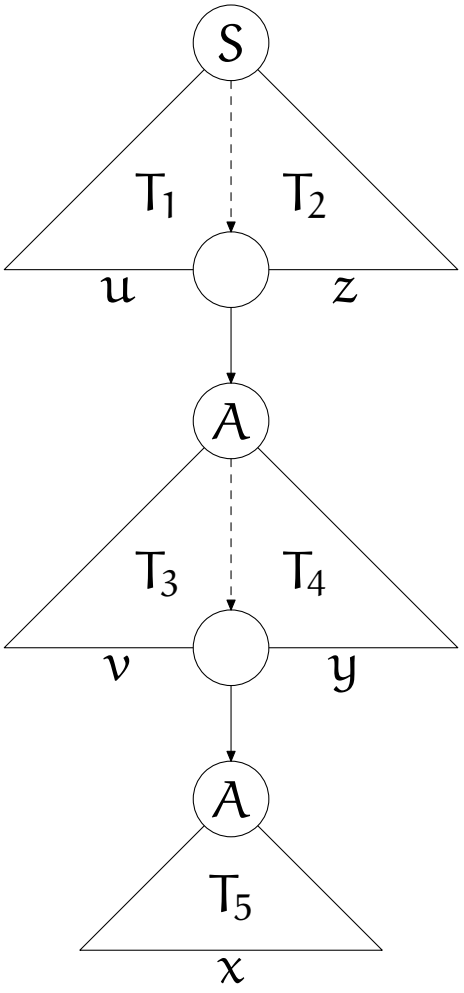
Jazyk je nekonečný, takže v něm musíme mít řetězy libovolné délky.

Počet přepisovacích pravidel je konečný, takže délka pravých stran je omezena nějakým číslem k. Pro odvození slova w tedy potřebujeme aspoň $|w|/k$ odvození. Délka odvození musí tedy být také neomezená.

Počet neterminálů je konečný, takže v dostatečně dlouhém odvození se musí nějaký terminál opakovat – označme ho A. Odvození pak vypadá takto:

$$S \overset{*}{\Rightarrow} uAz \overset{*}{\Rightarrow} uvAyz \overset{*}{\Rightarrow} uvxyz,$$

kde u, v, x, y a z jsou řetězy terminálů.



Příklad 40

Ukážeme, že jazyk $L = \{ ww \mid w \in \{a, b\}^+ \}$ není bezkontextový.

$$w = a^m b^m a^m b^m = uvxyz$$

Omezení délek: $|vxy| \leq m$, $|vy| \geq 1$.

Pro $\{ww^R\}$ tento postup (samozřejmě) nefunguje:

$$w = a^m b^m b^m a^m = uvxyz$$

Vsuvka: rozšířené regulární výrazy

Moderní programovací jazyky umožňují pracovat s *rozšířenými* regulární výrazy. Např. Java:

`(\d\d)\1`

Pozor na to, že tímto způsobem můžeme definovat jazyky, které *nejsou* regulární.

Některá rozšíření můžeme nahradit regulárními výrazy podle naší definice, například výrazu `\d` odpovídá `(0+1+2+3+4+5+6+7+8+9)`.

Se zpětnými referencemi se už ale můžeme dostat dokonce mimo třídu bezkontextových jazyků:

$$(\,.^{+})\backslash 1 = \{\, ww \mid w \in \Sigma^{+} \,\}$$

Viz heslo Wikipedie „Regular expression”.

Příklad 41

Ukážeme, že jazyk $L = \{ a^n b^n c^n \mid n \geq 0 \}$ není bezkontextový.

$$w = a^m b^m c^m = uvxyz$$

Omezení délek: $|vxy| \leq m$, $|vy| \geq 1$.

Pro $\{a^n b^n\}$ tento postup (samozřejmě) nefunguje:

$$w = a^m b^m = uvxyz$$

Pumpovací lemma pro lineární jazyky

Definice 30

Řekneme, že bezkontextový jazyk L je *lineární*, pokud existuje lineární bezkontextová gramatika G taková, že $L = L(G)$.

Věta 28

Nechť L je nekonečný lineární jazyk. Potom existuje přirozené číslo m takové, že libovolná věta $w \in L$ může být rozložena do podoby $w = uvxyz$, přičemž jsou splněny podmínky

$$|uvyz| \leq m,$$
$$|vy| \geq 1,$$

a navíc pro libovolné $i = 0, 1, 2, \ldots$ platí

$$uv^i xy^i z \in L.$$

Ve větných formách lineární gramatiky máme stále nejvýše jednu proměnnou, která se proto musí opakovat nejpozději po n odvozeních, kde n je celkový počet proměnných.

Příklad 42

Ukažte, že jazyk

$$L = \{ w \in \{a, b\}^* \mid n_a(w) = n_b(w) \}$$

není lineární.

Volíme $w = a^m b^{2m} a^m$.

Uzavřenost třídy bezkontextových jazyků

Věta 29

Jsou-li L_1 a L_2 bezkontextové jazyky, pak také $L_1 \cup L_2$, $L_1 L_2$ a L_1^* jsou bezkontextové jazyky.

$$G_1 = (V_1, T_1, S_1, P_1),\ G_2 = (V_2, T_2, S_2, P_2)$$

$$G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_3)$$
$$P_3 = P_1 \cup P_2 \cup \{\, S_3 \to S_1 \mid S_2 \,\}$$

$$G_4 = (V_1 \cup V_2 \cup \{S_4\}, T_1 \cup T_2, S_4, P_4)$$
$$P_4 = P_1 \cup P_2 \cup \{\, S_4 \to S_1 S_2 \,\}$$

$$G_5 = (V_1 \cup \{S_5\}, T_1, S_5, P_5)$$
$$P_5 = P_1 \cup \{\, S_5 \to S_1 S_5 \mid \lambda \,\}$$

Věta 30

Jsou-li L_1 a L_2 bezkontextové jazyky, pak $L_1 \cap L_2$ a $\overline{L_1}$ *nemusí* být bezkontextové jazyky.

$$L_1 = \{ a^n b^n c^m \mid n \geq 0, m \geq 0 \}$$
$$L_2 = \{ a^n b^m c^m \mid n \geq 0, m \geq 0 \}$$

Oba jsou bezkontextové – zřetězení dvou bezkontextových jazyků, např. $L_1 = \{ a^n b^n \mid n \in N_0 \}\{c\}^*$. Ovšem

$$L_1 \cap L_2 = \{ a^n b^n c^n \mid n \in N_0 \}$$

není bezkontextový.

Tvrzení pro doplněk plyne z de Morganova vzorce

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

Průnik bezkontextového a regulárního jazyka

Věta 31

Nechť L_1 je bezkontextový jazyk a L_2 regulární jazyk. Potom $L_1 \cap L_2$ (tzv. *regulární průnik*) je bezkontextový jazyk.

$$M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_{10}, z, F_1)$$
$$A_2 = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$$

Zkonstruujeme PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$, který simuluje paralelní zpracování vstupního řetězu oběma automaty M_1 a A_2.

$$Q = Q_1 \times Q_2$$
$$q_0 = (q_{10}, q_{20})$$
$$F = F_1 \times F_2$$

$((q_{1k}, q_{2l}), x) \in \delta((q_{1i}, q_{2j}), a, b)$ právě tehdy, když $(q_{1k}, x) \in \delta_1(q_{1i}, a, b)$ a zároveň $\delta_2(q_{2j}, a) = q_{2l}$.

Navíc pro $a = \lambda$ bereme $q_{2j} = q_{2l}$.

Příklad 43

1. Ukážeme, že jazyk $L_1 = \{ a^n b^n \mid n \geq 0, n \neq 42 \}$ je bezkontextový.

$$L_2 = \{ a^n b^n \mid n \in N_0 \} \cap \overline{\{a^{42} b^{42}\}}$$

2. Ukážeme, že jazyk $L_2 = \{ w \in \{a, b, c\}^* \mid n_a(w) = n_b(w) = n_c(w) \}$ není bezkontextový.

$$L_2 \cap L(a^* b^* c^*) = \{ a^n b^n c^n \mid n \in N_0 \}$$

Rozhodnutelné vlastnosti bezkontextových jazyků

Umíme rozhodnout, jestli daný řetěz patří do bezkontextového jazyka (NPDA).

Věta 32

Je-li G bezkontextová gramatika, pak existuje algoritmus, který rozhodne, zda je $L(G)$ prázdný.

Věta 33

Je-li G bezkontextová gramatika, existuje algoritmus, který rozhodne, zda je $L(G)$ nekonečný.

Neexistuje však žádný algoritmus, který by určil, zda jsou dvě bezkontextové gramatiky ekvivalentní, tj. generují stejný jazyk.

Jak na obecnější jazyky?

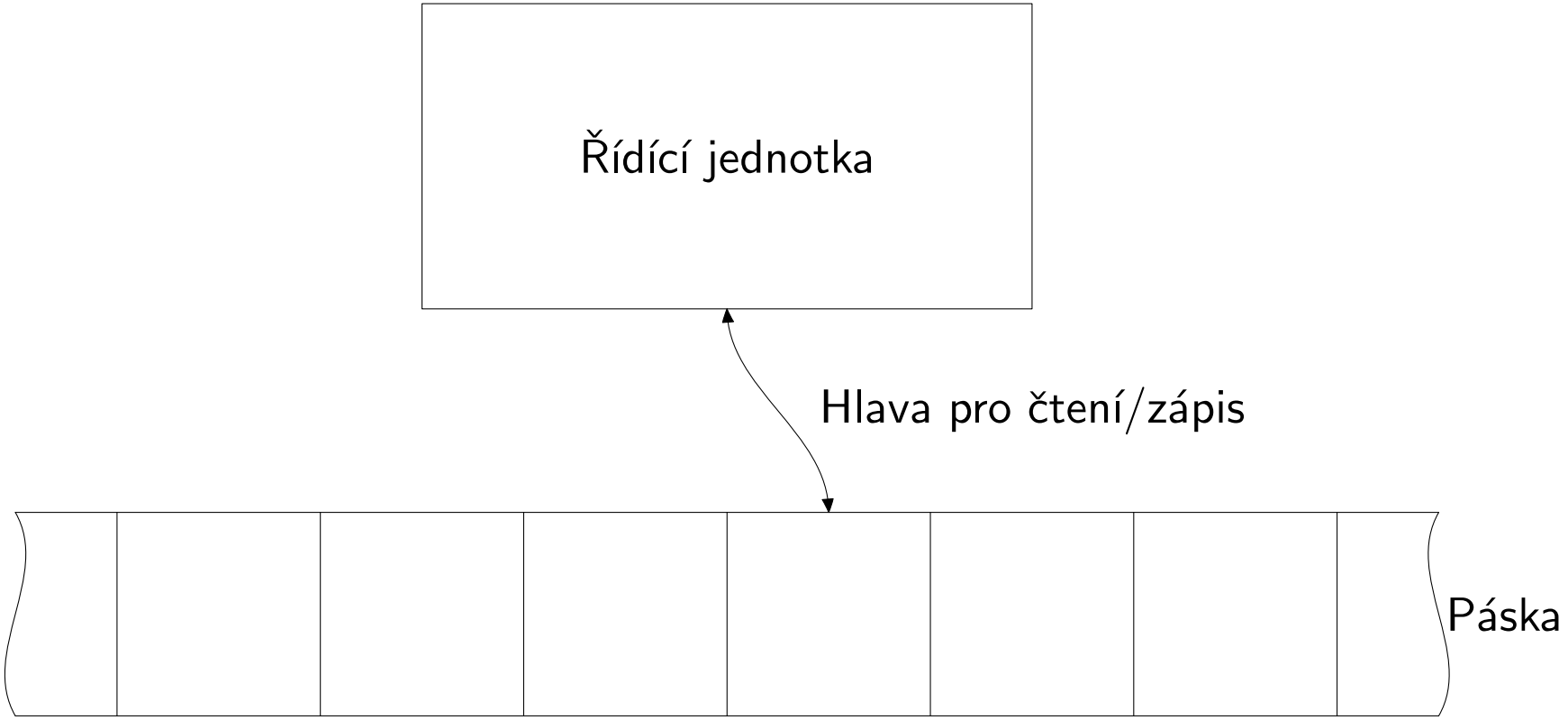
- regulární jazyky – konečný automat
- bezkontextové jazyky - zásobníkový automat

Víme, že existují jazyky, které nejsou bezkontextové. Jaké automaty je mohou rozpoznávat?

Hlavní rozdíl mezi NFA a NPDA je v „externí paměti”.

Turingův stroj

Alan M. Turing: On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* **2**, 42, s. 230-265, 1936.



Definice 31

Turingův stroj M je definován jako uspořádaná sedmice

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F),$$

kde

- Q je konečná množina stavů řídící jednotky,
- Σ je vstupní abeceda,
- Γ je konečná množina symbolů – *abeceda pásky*
- $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ je přechodová funkce,
- $q_0 \in Q$ je počáteční stav,
- $\square \in \Gamma$ je speciální *prázdný symbol (blank)*
- $F \subset Q$ je množina koncových stavů.

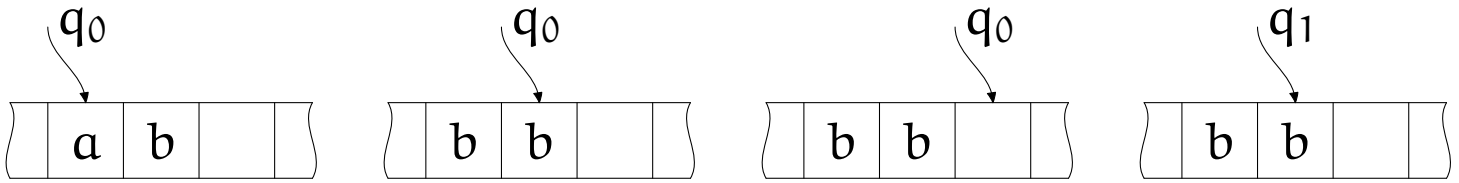
Turingův stroj nemá žádný „vstupní soubor", veškerá vstupní data musí být napsána na pásku. Proto předpokládáme, že $\Sigma \subset \Gamma - \{\square\}$.

Přechodová funkce na základě aktuálního stavu a přečteného symbolu změní stav, přepíše symbol a posune hlavu na pásce o políčko vlevo nebo vpravo.

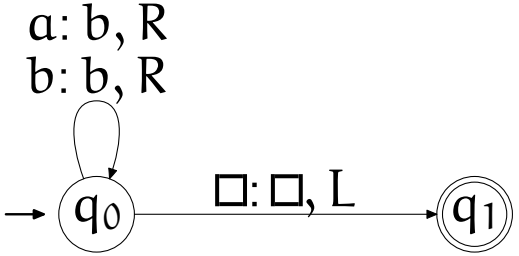
Příklad 44

$$Q = \{q_0, q_1\},$$
$$\Sigma = \{a, b\},$$
$$\Gamma = \{a, b, \square\},$$
$$F = \{q_1\},$$

$$\delta(q_0, a) = (q_0, b, R),$$
$$\delta(q_0, b) = (q_0, b, R),$$
$$\delta(q_0, \square) = (q_1, \square, L).$$



Přechodový graf

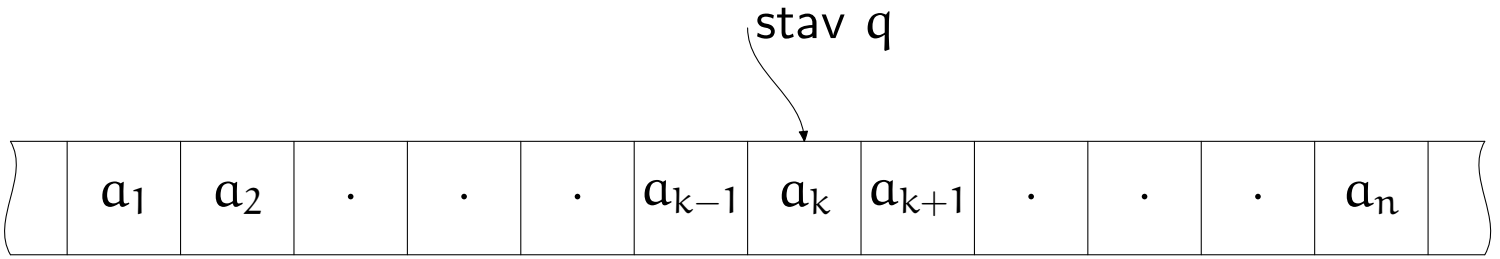


Zápis stavu Turingova stroje

Okamžitý stav daného Turingova stroje je plně určen stavem řídící jednotky, obsahem pásky a pozicí hlavy. Můžeme ho kompaktně zapsat např. takto:

$$a_1 a_2 \ldots a_{k-1} q a_k a_{k+1} \ldots a_n$$

To v grafickém znázornění znamená:



Nalevo a napravo od zapsaného bloku jsou už jen samé blanky.

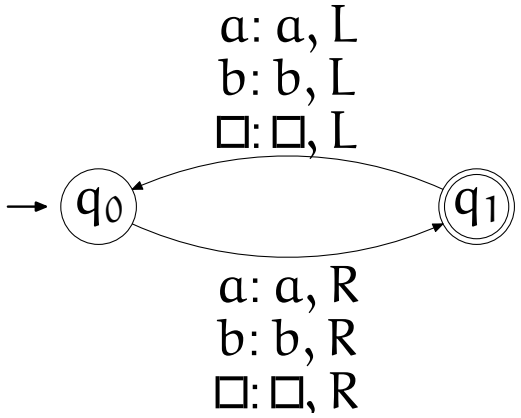
Přechod mezi po sobě následujícími stavy budeme opět zapisovat pomocí symbolu $\vdash$ a přechody s libovolným počtem kroků pomocí $\overset{*}{\vdash}$. V příkladu 44 jsme měli přechody

$$q_0ab \vdash bq_0b \vdash bbq_0\square \vdash bq_1b \quad \textbf{neboli} \quad q_0ab \overset{*}{\vdash} bq_1b.$$

Zastavení Turingova stroje

Pokud pro aktuální stav řídící jednotky a čtený symbol není definována hodnota přechodové funkce δ, Turingův stroj se zastaví – **halt**. U koncových stavů $q \in F$ je to často zařízeno tak, že $\delta(q, a)$ není definováno pro žádné $a \in \Gamma$.

Může se ale stát, že se Turingův stroj „zacyklí", tzn. nezastaví nikdy:



Tuto skutečnost zapisujeme takto:

$$x_1 q_0 x_2 \overset{*}{\vdash} \infty.$$

Jazyk rozpoznávaný Turingovým strojem

Definice 32

Jazyk $L(M)$ rozpoznávaný Turingovým strojem $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ je množina všech slov $w \in \Sigma^*$ takových, že

$$q_0 w \overset{*}{\vdash} x_1 q_f x_2$$

pro nějaké $q_f \in F$ a $x_1, x_2 \in \Gamma^*$.

To znamená, že vstupní slovo w je zapsáno na úseku pásky ohraničeném z obou stran blanky, řídící jednotka je v počátečním stavu q_0 a hlava je na prvním symbolu slova w. Rozpoznání slova je indikováno *zastavením* v nějakém koncovém stavu, na konečném obsahu pásky přitom nezáleží.

Nic se neříká o tom, co se stane pro slova $w \notin L(M)$: stroj se může buď zacyklit anebo zastavit ve stavu, který není koncový.

Příklad 45

Navrhneme Turingův stroj pro jazyk $L = \{ a^n b^n \mid n \in N \}$.

Příklad 45

Navrhneme Turingův stroj pro jazyk $L = \{ a^n b^n \mid n \in N \}$.

$Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_4\}$, $\Gamma = \{a, b, x, y, \square\}$.

$$\begin{aligned}
\delta(q_0, a) &= (q_1, x, R) \\
\delta(q_1, a) &= (q_1, a, R) \\
\delta(q_1, y) &= (q_1, y, R) \\
\delta(q_1, b) &= (q_2, y, L) \\
\delta(q_2, y) &= (q_2, y, L) \\
\delta(q_2, a) &= (q_2, a, L) \\
\delta(q_2, x) &= (q_0, x, R) \\
\delta(q_0, y) &= (q_3, y, R) \\
\delta(q_3, y) &= (q_3, y, R) \\
\delta(q_3, \square) &= (q_4, \square, R)
\end{aligned}$$

Příklad 46

Navrhneme Turingův stroj pro jazyk $L = \{ a^n b^n c^n \mid n \in N \}$.

Postup je velmi podobný jako u předchozího příkladu (rozmyslete!).

Turingův stroj je tedy schopen rozeznávat i jazyky, které nejsou bezkontextové.

Výpočty pomocí Turingova stroje

Implementace funkce ϕ pomocí Turingova stroje:

$$w' = \phi(w) \iff q_0 w \overset{*}{\vdash} q_f w'$$

pro nějaký koncový stav $q_f \in F$.

Definice 33

Nechť Σ je konečná abeceda a $D \subset \Sigma^*$. Řekneme, že funkce ϕ s definičním oborem D je vypočitatelná (v Turingově smyslu), pokud existuje nějaký Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ takový, že pro všechna $w \in D$ dostaneme

$$q_0 w \overset{*}{\vdash} q_f \phi(w)$$

pro nějaké $q_f \in F$.

Příklad 47

Navrhneme Turingův stroj, který sečte dvě libovolná přirozená čísla x a y.

Použijeme unární kódování čísel (např. $w(3) = 111$) a oba zakódované sčítance $w(x)$ a $w(y)$ umístíme na pásku oddělené nulou, takže $\Sigma = \{0, 1\}$. Naším cílem je výpočet

$$q_0 w(x) 0 w(y) \overset{*}{\vdash} q_f w(x + y) 0.$$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_4\}$, $\Gamma = \{0, 1, \square\}$.

$$\begin{aligned}
\delta(q_0, 1) &= (q_0, 1, R) \\
\delta(q_0, 0) &= (q_1, 1, R) \\
\delta(q_1, 1) &= (q_1, 1, R) \\
\delta(q_1, \square) &= (q_2, \square, L) \\
\delta(q_2, 1) &= (q_3, 0, L) \\
\delta(q_3, 1) &= (q_3, 1, L) \\
\delta(q_3, \square) &= (q_4, \square, R)
\end{aligned}$$

Příklad 48

Navrhneme Turingův stroj, který zkopíruje libovolný blok jedniček, tedy pro každé $w \in \{1\}^*$ provede výpočet

$$q_0 w \overset{*}{\vdash} q_f ww.$$

Možný postup:

1. Všechny jedničky nahradíme symbolem x.
2. Najdeme nejpravější x a nahradíme jej jedničkou.
3. Přesuneme se doprava až na první blank a ten přepíšeme jedničkou.
4. Kroky 2 a 3 opakujeme, dokud na pásce zbývají nějaká x.

Příklad 49

Navrhneme Turingův stroj, který pro libovolná přirozená čísla x a y rozhodne, zda $x \geq y$, tj. takovýto výpočet:

$$q_0 w(x) 0 w(y) \vdash^* \begin{cases} q_y w(x) 0 w(y), & \text{pro } x \geq y; \\ q_n w(x) 0 w(y), & \text{pro } x < y. \end{cases}$$

Kombinace Turingových strojů

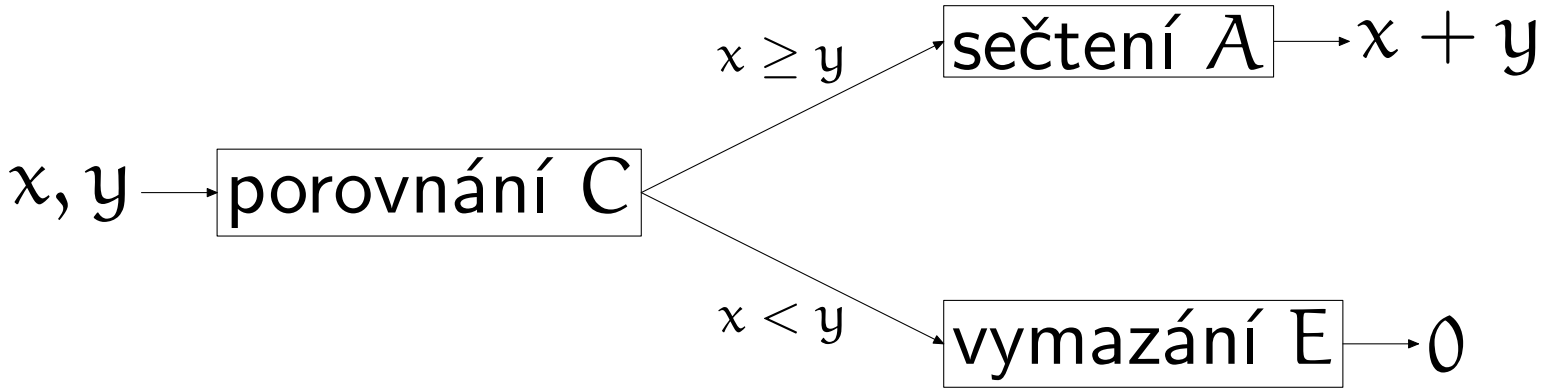
Pro realizaci složitějších výpočtů bývá užitečné kombinovat Turingovy stroje, které realizují jednodušší operace. Používají se dvě metody:

- blokový diagram
- pseudokód

Příklad 50 – blokový diagram

Navrhneme Turingův stroj počítající funkci

$$\phi(x,y) = \begin{cases} x + y, & \text{pro } x \geq y; \\ 0, & \text{jinak.} \end{cases}$$



Napojení bloků: Stavy jednotlivých bloků rozlišíme přidáním symbolu daného bloku, např. počáteční stav bloku C bude q_{C0}. Použijeme blok C z příkladu 11.6, přičemž stav q_{Cy} ztotožníme s q_{A0} a podobně q_{Cn} ztotožníme s q_{E0}.

Příklad 51 – pseudokód

Implementujeme makroinstrukci

if a **then** q_j **else** q_k

s touto interpretací: Pokud je symbol na pásce a, přejdi bez ohledu na stav řídící jednotky do stavu q_j, jinak přejdi do stavu q_k. V obou případech se symbol na pásce nemění.

$$\delta(q_i, a) = (q'_j, a, R) \quad \text{pro všechna } q_i \in Q$$
$$\delta(q_i, b) = (q'_k, b, R) \quad \text{pro všechna } q_i \in Q \text{ a } b \in \Gamma - \{a\}$$
$$\delta(q'_j, c) = (q_j, c, L) \quad \text{pro všechna } c \in \Gamma$$
$$\delta(q'_k, c) = (q_j, c, L) \quad \text{pro všechna } c \in \Gamma$$

Volání podprogramu

Turingův stroj (hlavní program) A volá stroj B jako svůj podprogram.

Chceme, aby:

- bylo možné podprogramu předat parametry,
- podprogram může vrátit nějakou hodnotu jako svůj výsledek,
- po ukončení podprogramu pokračoval hlavní program přesně v té konfiguraci, v níž byl při volání podprogramu.

Všechny potřebné informace musíme zapsat na pásku.

Turingova hypotéza

Otázka: Jsou výpočetní možnosti Turingova stroje stejné jako možnosti standardního počítače?

Turingova hypotéza: Každý výpočet provedený jakýmikoli mechanickými prostředky může být také proveden nějakým Turingovým strojem.

Turingova hypotéza se nedá formálně dokázat, protože nevíme přesně, co jsou to „mechanické prostředky". Můžeme ji proto spíše považovat za informatický axiom.

Nalezením protipříkladu by ji bylo možné vyvrátit, to se však zatím nikomu nepovedlo.

Algoritmus

Definice 34

Algoritmem pro výpočet funkce $\phi\colon D \to H$ rozumíme Turingův stroj M, který se pro libovolný vstup $d \in D$ umístěný na pásce zastaví se správným výsledkem $\phi(d) \in H$ na pásce. Konkrétně požadujeme, aby pro všechna $d \in D$ proběhl tento výpočet:

$$q_0 d \overset{*}{\vdash} q_f \phi(d), \quad \text{pro } q_f \in F.$$

Z praktických důvodů se ovšem obvykle spoléháme na Turingovu hypotézu a místo Turingova stroje raději použijeme program v nějakém vyšším programovacím jazyce.

Modifikace Turingova stroje

Turingova hypotéza naznačuje, že žádná přímočará „vylepšení" Turingova stroje nemohou přinést žádné principiálně nové možnosti. Probereme následující modifikace Turingova stroje a ukážeme, že všechny lze simulovat základním Turingovým strojem:

- možnost setrvání hlavy na místě;
- vstupní (read-only) soubor;
- více pásek;
- dvojrozměrná „páska"

Ekvivalence tříd automatů

Definice 35

Uvažujme dvě třídy automatů C_1 a C_2. Pokud pro každý automat $M_1 \in C_1$ existuje automat $M_2 \in C_2$ takový, že $L(M_1) = L(M_2)$, řekneme, že třída C_2 je alespoň tak mocná jako třída C_2. Pokud platí i obrácený vztah obou tříd, tj. pro každé $M_2 \in C_2$ existuje $M_1 \in C_1$ tak, že $L(M_1) = L(M_2)$, řekneme, že třídy C_1 a C_2 jsou ekvivalentní.

Simulace činnosti automatu M automatem M':

Pro každý výpočet automatu M

$$d_0 \vdash_M d_1 \vdash_M \cdots \vdash_M d_n$$

můžeme automatem M' realizovat simulační výpočet

$$d'_0 \overset{*}{\vdash}_{M'} d'_1 \overset{*}{\vdash}_{M'} \cdots \overset{*}{\vdash}_{M'} d'_n$$

přičemž „čárkované" konfigurace d'_i simulačního stroje M' odpovídají konfiguracím d_i původního stroje M.

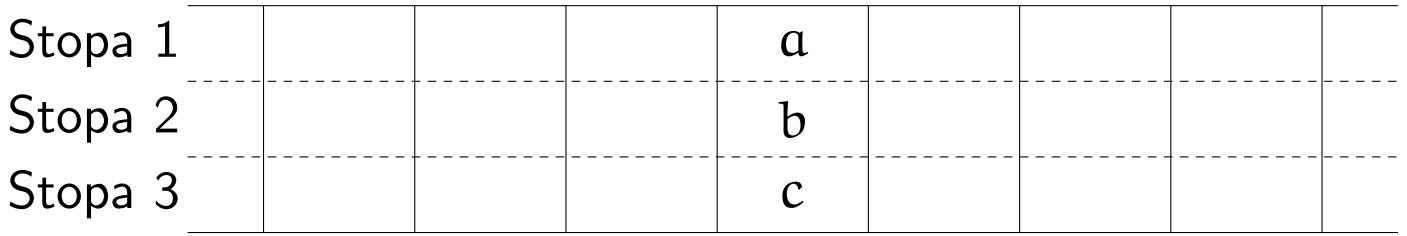
Hlava může zůstat na místě

Oproti definici Turingova stroje (9.2) se liší jen přechodová funkce:

$$\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}.$$

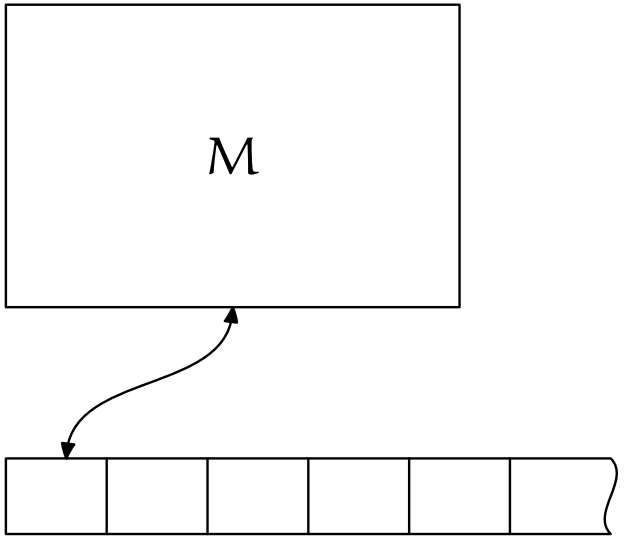
Simulace standardním Turingovým strojem: Každý přechod bez pohybu hlavy $\delta(q_i, a) = (q_j, b, S)$ simulujeme dvojicí přechodů standardního Turingova stroje: $\delta'(q'_i, a) = (q'_{jS}, b, R)$ a $\delta'(q'_{jS}, c) = (q'_j, c, L)$ pro všechna $c \in \Gamma$.

Turingův stroj s vícestopou páskou



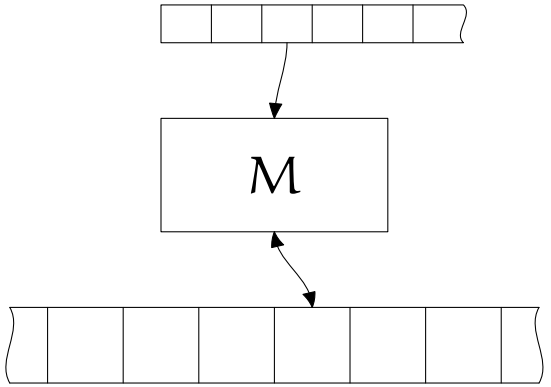
Toto je stále standardní Turingův stroj, abecedu pásky můžeme brát jako uspořádané trojice znaků.

Turingův stroj s „polonekonečnou” páskou



Standardní Turingův stroj můžeme tímto strojem M simulovat tak, že polonekonečnou pásku rozdělíme na dvě stopy, z nichž každá bude reprezentovat „polovinu” standardní pásky.

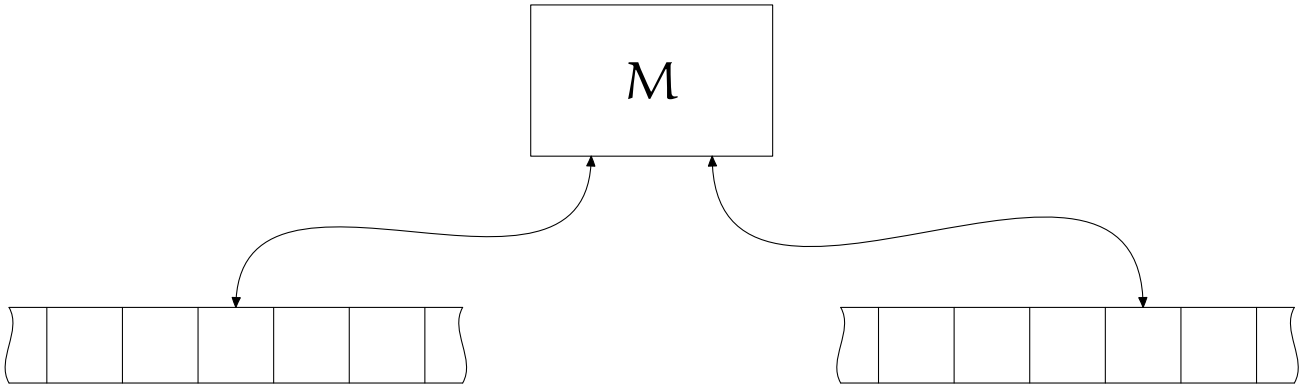
Turingův stroj se vstupním souborem



Tento stroj můžeme standardním Turingovým strojem simulovat tak, že použijeme pásku se 4 stopami:

1. obsah vstupního souboru
2. pozice čtecí hlavy vstupního souboru
3. obsah pásky stroje M
4. pozice hlavy na pásce stroje M

Turingův stroj s více páskami



Stroj s více páskami můžeme standardním Turingovým strojem simulovat podobně jako v předchozím případě: Každé pásce bude odpovídat dvojice stop – v jedné bude zapsán obsah pásky a druhá bude zachycovat pozici hlavy na této pásce.

Turingův stroj s dvourozměrnou páskou

$$\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D\}$$

			$(0,1)$			
		$(-1,0)$	$(0,0)$	$(1,0)$		
			$(0,-1)$			

Použijeme standardní pásku se dvěma stopami. Bude-li např. na 2D pásce a v pozici $(1,2)$ a b v pozici $(10,-3)$, použijeme toto kódování:

	a				b						
	1	$\#$	2	$\#$	1	0	$\#$	$-$	3	$\#$	

Nedeterministický Turingův stroj

Definice 36

Nedeterministický Turingův stroj se liší od deterministického (Definice 9.2) pouze v tom, že přechodová funkce δ může mít více hodnot:

$$\delta\colon Q \times \Gamma \to 2^{Q\times\Gamma\times\{L,R\}}$$

Příklad 52: Turingův stroj s přechodovou funkcí

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

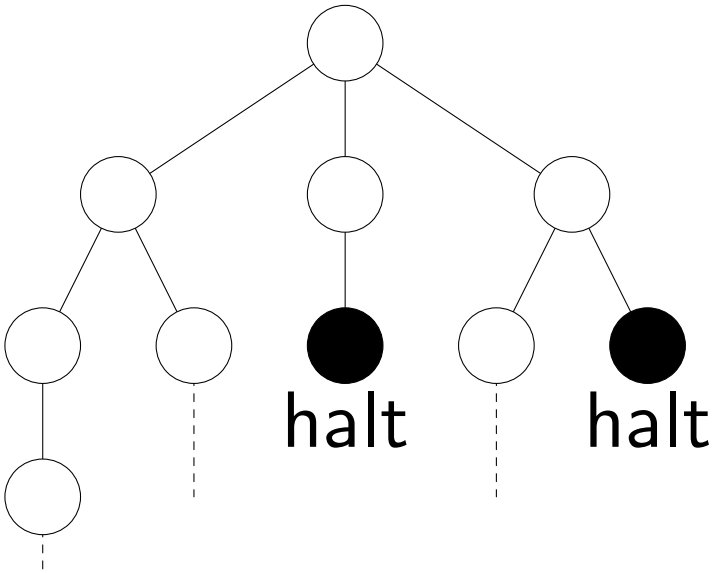
je nedeterministický, protože jsou možné alternativní přechody ze stejného stavu, např. $q_0aaa \vdash bq_1aa$ i $q_0aaa \vdash q_2\Box caa$.

U nedeterministického Turingova stroje ale není zřejmé, zda můžeme aplikovat Turingovu hypotézu, protože nedeterministický výpočet nemá zcela mechanickou povahu.

Nedeterministický Turingův stroj ale můžeme simulovat deterministickým algoritmem procházení stromu všech aktuálních možností.

Věta 34

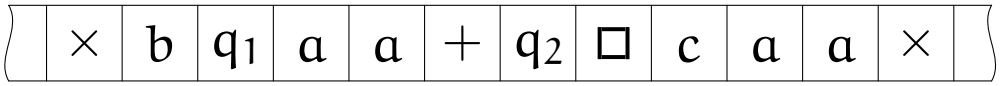
Třída deterministických Turingových strojů je ekvivalentní s třídou nedeterministických Turingových strojů.



počáteční stav

stav po prvním kroku

stav po druhém kroku



Univerzální Turingův stroj

Turingův stroj, který simuluje činnost libovolného zadaného Turingova stroje.

Kódování Turingova stroje:

$$Q = \{q_1, q_2, \ldots, q_n\}$$

q_1 je počáteční stav, kódujeme jako 1
q_2 je koncový stav, kódujeme jako 11, atd.

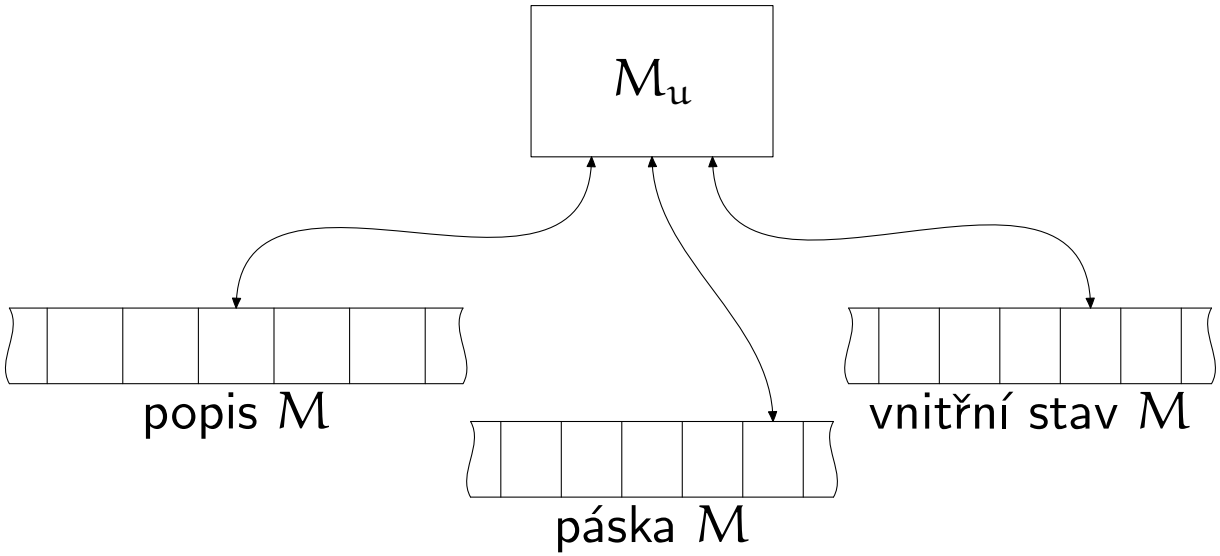
$$\Gamma = \{a_1, a_2, \ldots, a_m\}$$

a_1 kódujeme jako 1, a_2 jako 11 atd.

Přechodovou funkci kódujeme jako soubor pětic (2 argumenty, 3 hodnoty) oddělených nulou, tedy například $\delta(q_1, a_2) = (q_2, a_3, L)$ zakódujeme jako

$$1\,0\,1\,1\,0\,1\,1\,0\,1\,1\,1\,0\,1\,0$$

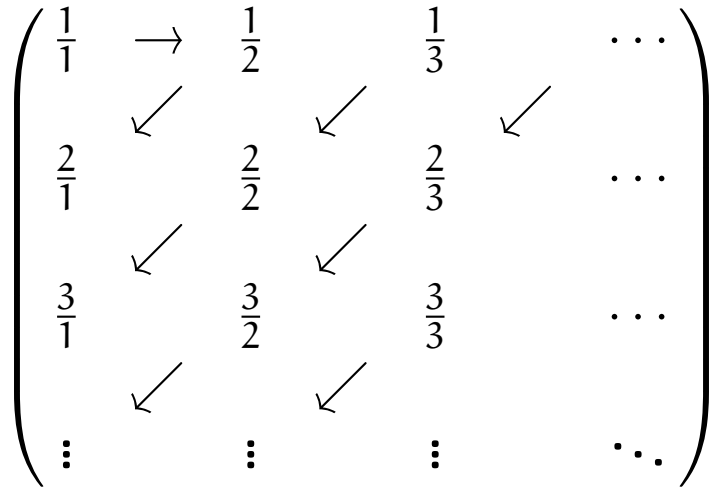
Realizace univerzálního Turingova stroje



Výčtová procedura pro nekonečné množiny

Nekonečné spočetné množiny – prvkům můžeme vzájemně jednoznačně přiřadit přirozená čísla.

Příklad 53: Zlomky



Po řádcích nemůžeme jít, protože by se na $\frac{2}{1}$ nikdy nedostalo.

Výčtovou proceduru můžeme formálně definovat pomocí Turingova stroje

Definice 37

Nechť S je množina řetězů nad abecedou Σ. *Výčtovou procedurou* pro množinu S nazveme výpočet Turingova stroje sestávající z posloupnosti kroků

$$q_0\square \overset{*}{\vdash} q_s x_1 \# s_1 \overset{*}{\vdash} q_s x_2 \# s_2 \ldots,$$

kde $x_i \in \Gamma^* - \{\#\}$ a $s_i \in S$, přičemž každý řetěz $s \in S$ je tímto postupem vygenerován po konečném počtu kroků výpočtu. Jakmile se stroj dostane do stavu q_s musí být za symbolem # vždy řetěz z množiny S.

Množina, pro niž existuje výčtové procedura, je spočetná.

Příklad 54

Ukážeme, že $\mathrm{L} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}^+$ je spočetná množina.

Lexikografické uspořádání (jako ve slovníku) nemůžeme použít, protože slov začínajících na a je nekonečně mnoho, takže žádné slovo začínající na b by nebylo nikdy vygenerováno.

Proto volíme jako primární kritérium délku řetězu a teprve řetězy stejné délky uspořádáme lexikograficky:

$$\mathrm{a} \prec \mathrm{b} \prec \mathrm{c} \prec \mathrm{aa} \prec \mathrm{ab} \prec \mathrm{ac} \prec \mathrm{ba} \prec \mathrm{bb} \prec \mathrm{bc} \prec \mathrm{ca} \prec \mathrm{cb} \prec \mathrm{cc} \prec \mathrm{aaa} \prec \ldots$$

Toto je takzvané *kanonické uspořádání*.

Kolik je Turingových strojů?

Věta 35

Množina všech Turingových strojů je nekonečná, ale spočetná.

Víme, že každý Turingův stroj můžeme zakódovat jako jistý řetěz nul a jedniček. Můžeme tedy použít následující výčtovou proceduru pro Turingovy stroje:

❶ Vygenerujeme další řetěz $w \in \{0,1\}^+$ v pořadí podle kanonického uspořádání.

❷ Zkontrolujeme, zda w může popisovat Turingův stroj. Pokud ne, řetěz w ignorujeme. Pokud ano, zapíšeme w na pásku ve formě požadované definicí 37.

❸ Vrátíme se ke kroku ❶.

Lineárně omezený automat

Předpoklad nekonečné pásky Turingova stroje je nerealistický. Pokud ale délku pásky a priori omezíme, dostaneme konečný automat (rozmyslete!). Zajímavým případem je *lineárně omezený automat (LBA)*, u nějž vstupní řetěz zapsaný na pásce určuje použitelnou část pásky.

Definice 38

Lineárně omezený automat je *nedeterministický* Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ podle definice 11.2, pro který navíc platí tato omezení:

1. Σ obsahuje dva speciální symboly $[$ a $]$,
2. $\delta(q_i, [)$ může obsahovat pouze uspořádané trojice tvaru $(q_j, [, R)$,
3. $\delta(q_i,])$ může obsahovat pouze uspořádané trojice tvaru $(q_j,], L)$.

Definice 39

Řetěz w je přijat (rozpoznán) lineárně omezeným automatem pokud existuje posloupnost tahů

$$q_0[w] \overset{*}{\vdash} [x_1 q_f x_2]$$

pro nějaké $q_f \in F$ a $x_1, x_2 \in \Gamma^*$. Jazyk přijímaný (rozpoznávaný) lineárně omezeným automatem je množina všech takových řetězů.

Příklad 55: Jazyk $L = \{ a^n b^n c^n \mid n \geq 0 \}$ je rozpoznatelný lineárně omezeným automatem.

Rekurzivně spočetné a rekurzivní jazyky

Definice 40

Jazyk L se nazývá rekurzivně spočetný, pokud existuje Turingův stroj, který tento jazyk rozpoznává.

Pro řetězy, které do L nepatří, se může rozpoznávající Turingův stroj zacyklit.

Definice 41

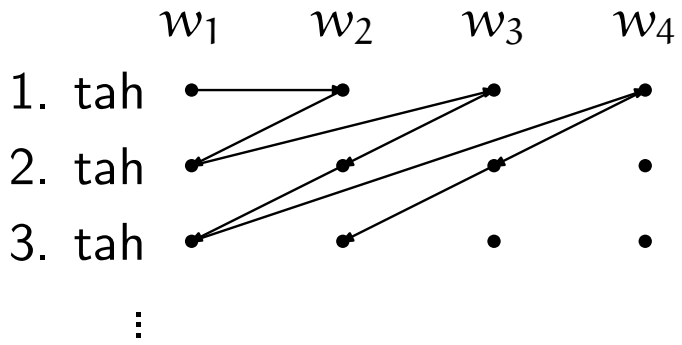
Jazyk L se nazývá rekurzivní, pokud existuje Turingův stroj M, který tento jazyk rozpoznává a navíc se pro každý řetěz $w \in \Sigma^*$ zastaví.

Pro rekurzivní jazyk L tedy máme algoritmus, který pro každý řetěz w rozhodne, zda $w \in \mathrm{L}$ anebo $w \notin \mathrm{L}$.

Výčtové procedury

Pro rekurzivní jazyk L: Použijeme Turingův stroj, který rozhoduje, zda $w \in \mathrm{L}$ nebo $w \notin \mathrm{L}$, a spouštíme jej postupně na řetězy v kanonickém uspořádání – řetězy patřící do L se zapíší na pásku.

Pro rekurzivně spočetné jazyky musíme dát pozor na zacyklení. Postup:



Libovolný řetěz $w \in \mathrm{L}$ takto rozpoznáme (a můžeme zapsat na pásku) po konečném počtu kroků.

Každý jazyk L, pro nějž existuje výčtová procedura je rekurzivně spočetný: každé $w \in \mathrm{L}$ najdeme v řetězech generovaných výčtovou procedurou po konečném počtu kroků.

Všechny jazyky nejsou rekurzivně spočetné

Věta 36

Je-li S nekonečná spočetná množina, pak její potenční množina 2^S je nespočetná.

Dokáže se diagonalizační procedurou.

Věta 37

Pro libovolnou abecedu Σ existují jazyky, které nejsou rekurzivně spočetné.

Σ^* je nekonečná spočetná, tedy 2^{Σ^*} (= množina všech jazyků) je nespočetná. Množina Turingových strojů (a tedy i rekurzivně spočetných jazyků) je ale spočetná, proto musí existovat jazyky, které nejsou rekurzivně spočetné.

Konstrukce jazyka, který není rekurzivně spočetný

Věta 38

Existuje rekurzivně spočetný jazyk jehož doplněk není rekurzivně spočetný.

Množina všech Turingových strojů se vstupní abecedou $\Sigma = \{a\}$ je spočetná, proto ji můžeme uspořádat do posloupnosti $\{M_i\}_{i \geq 1}$.

$$L = \{\, a^i \mid a^i \in L(M_i) \,\}$$

Tento jazyk je rekurzivně spočetný, doplněk $\overline{L}$ ale není.

Rekurzivně spočetný jazyk, který není rekurzivní

Věta 39

1. Pokud je jazyk L i jeho doplněk $\overline{L}$ rekuzivně spočetný, potom jsou oba tyto jazyky rekurzivní.
2. Je-li L rekurzivní, potom je i $\overline{L}$ rekurzivní (a tím pádem jsou oba jazyky rekurzivně spočetné).

Věta 40

Existuje rekurzivně spočetný jazyk, který není rekurzivní. To znamená, že třída rekurzivních jazyků je vlastní podmnožinou rekurzivně spočetných jazyků.

Jazyk L v důkazu věty 12.3, je rekurzivně spočetný, jeho doplněk ale není. Proto podle věty 12.4 nemůže být L rekurzivní.

Neomezené gramatiky

Definice 42

Gramatika $G = (V, T, S, P)$ se nazývá *neomezená*, pokud jdou všechny její produkce tvaru

$$u \to v,$$

kde $u \in (V \cup T)^+$ a $v \in (V \cup T)^*$.

Jediným omezením je to, že na levé straně produkcí nesmí být λ.

Věta 41

Každý jazyk generovaný neomezenou gramatikou je rekurzivně spočetný.

Výčtová procedura: nejprve vygenerujeme všechny věty přímo odvoditelné z S (tj. $S \Rightarrow w$), potom všechny věty odvoditelné ve dvou krocích (tj. $S \Rightarrow x \Rightarrow w$) atd.

Gramatika pro rekurzivně spočetný jazyk

K danému Turingovu stroji $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ rozpoznávajícímu jazyk $L(M)$ chceme najít gramatiku G takovou, že $L(G) = L(M)$.

Postup pro libovolné $w \in \Sigma^+$:

1. Z počáteční proměnné S odvodíme $q_0 w$.
2. Odvození $q_0 w \overset{*}{\Rightarrow} x q_f y$ bude možné právě tehdy, když $q_0 w \overset{*}{\vdash} x q_f y$.
3. Pokud se vygeneruje řetěz $x q_f y$, kde $q_f \in F$, přepíšeme tento řetěz na původní řetěz w.

Kvůli kroku 3 musíme zařídit, abychom si v průběhu celého odvození pamatovali původní řetěz w. Kromě toho musíme nějak kódovat aktuální vnitřní stav Turingova stroje a pozici hlavy na pásce.

Zavedeme proměnné V_{ab} a V_{aib} pro všechna $a \in \Sigma \cup \{\square\}$, $b \in \Gamma$ a všechna i taková, že $q_i \in Q$.

Krok ❶: Pro všechna $a \in \Sigma$ přidáme produkce

$$S \to V_{\square\square}S \mid SV_{\square\square} \mid T,$$
$$T \to TV_{aa} \mid V_{a0a}$$

Krok ❷: Pro každý přechod Turingova stroje

$$\delta(q_i, c) = (q_j, d, R)$$

přidáme produkce

$$V_{aic}V_{ef} \to V_{ad}V_{ejf}$$

pro všechna $a, e \in \Sigma \cup \{\square\}$ a $f \in \Gamma$. Podobně pro

$$\delta(q_i, c) = (q_j, d, L)$$

přidáme sadu produkcí

$$V_{ef}V_{aic} \to V_{ejf}V_{ad}.$$

Krok ❸: Pokud se výpočet zastaví v koncovém stavu, musíme znovu vygenerovat původní řetěz w a všechno ostatní vymazat. Pro všechny koncové stavy $q_j \in F$ proto přidáme produkce

$$V_{ajb} \to a,$$

pro všechna $a \in \Sigma \cup \{\square\}$ a $b \in \Gamma$. Tím změníme symbol b, na němž je po zastavení nastavena hlava Turingova stroje, na a, což je symbol, který byl na tomto místě na začátku. Od něj pak vracíme oběma směry všechny původní symboly:

$$cV_{ab} \to ca,$$
$$V_{ab}c \to ac,$$

pro všechna $a, c \in \Sigma \cup \{\square\}$ a $b \in \Gamma$.

Nakonec potřebujeme ještě vymazat blanky:

$$\square \to \lambda.$$

Příklad 56

Zkonstruujte gramatiku pro Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ rozpoznávající jazyk $L(aa^*)$, kde

$$\begin{aligned} Q &= \{q_0, q_1\}, \\ \Sigma &= \{a, b\}, \\ \Gamma &= \Sigma \cup \{\square\}, \\ F &= \{q_1\} \end{aligned}$$

a

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

Výsledné tvrzení:

Věta 42

Pro každý rekurzivně spočetný jazyk L existuje neomezená gramatika G taková, že $L(G) = L$.

Kontextová gramatika

Definice 43

Gramatika $G = (V, T, S, P)$ se nazývá *kontextová*, pokud jdou všechny její produkce tvaru

$$u \to v,$$

kde $u, v \in (V \cup T)^+$ a zároveň platí

$$|u| \leq |v|. \qquad (*)$$

Vlastnost $(*)$ znamená, že gramatika je *nekontrahující*, to znamená, že postupně odvozované větné formy se nemohou zkracovat.

Lze ukázat, že takové gramatiky lze převést do normální formy, v níž jsou všechny produkce tvaru

$$xAy \to xvy,$$

tj. přepis proměnné A na řetěz v je možný jen v určitém *kontextu*, který je dán řetězy x a y.

Kontextový jazyk

Základní vlastnost $(*)$ kontextových gramatik vylučuje produkce mající na pravé straně λ. V kontextových jazycích je prázdný řetěz ponechán jako speciální případ.

Definice 44

Řekneme, že jazyk L je *kontextový*, pokud existuje kontextová gramatika G taková, že buď $\mathrm{L} = \mathrm{L}(\mathrm{G})$ anebo $\mathrm{L} = \mathrm{L}(\mathrm{G}) \cup \{\lambda\}$.

Příklad 57

Jazyk $\{ a^n b^n c^n \mid n \geq 1 \}$ je kontextový.

$$
\begin{aligned}
S &\to abc \mid aAbc, \\
Ab &\to bA, \\
Ac &\to Bbcc, \\
bB &\to Bb, \\
aB &\to aa \mid aaA.
\end{aligned}
$$

Automat pro kontextové jazyky = LBA

Věta 43

Pro každý kontextový jazyk L neobsahující prázdný řetěz existuje lineárně omezený automat M takový, že $\mathrm{L} = \mathrm{L}(\mathrm{M})$.

Odvozování pomocí produkcí simulujeme pomocí LBA se dvěma stopami, na jedné je vstupní řetěz w a na druhou zapisujeme postupně odvozované větné formy – vzhledem k podmínce $(*)$ v definici 12.4 ale žádná z nich nemůže být delší než w. Podstatné je také to, že LBA je definován jako *nedeterministický*.

Věta 44

Je-li L jazyk přijímaný nějakým lineárně omezeným automatem M, potom existuje kontextová gramatika G taková, že $\mathrm{L}(\mathrm{G}) = \mathrm{L}(\mathrm{M})$.

Stejná konstrukce gramatiky jako u věty 12.7, všechny produkce jdou nekontrahující kromě $\square \to \lambda$, tu ale zde nepotřebujeme.

Vztah mezi rekurzivními a kontextovými jazyky

Věta 45

Každý kontextový jazyk je rekurzivní.

Podmínka $(*)$ garantuje, že počet odvození libovolného řetězu $w \in L$ je shora omezen číslem, které je funkcí $|w|$.

Věta 46

Existuje rekurzivní jazyk, který není kontextový.

Konstrukce pro $T = \{0, 1\}$: Libovolnou gramatiku zakódujeme jako řetěz nul a jedniček.

Proměnné: $V = \{V_0, V_1, V_2, \ldots\}$, V_0 je počáteční proměnná. Produkce zapíšeme jako jeden řetěz

$$x_1 \to y_1; x_2 \to y_2; \ldots x_m \to y_m.$$

Kódování produkcí:

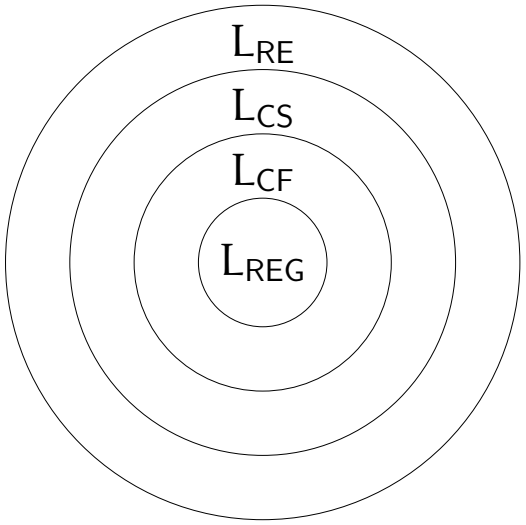
$$\begin{aligned} h(0) &= 010, \\ h(1) &= 0110, \\ h(\to) &= 01110, \\ h(;) &= 011110, \\ h(V_i) &= 01^{i+5}0. \end{aligned}$$

Každá kontextová gramatika je tedy vzájemně jednoznačně zakódována jako konečný řetěz nul a jedniček. Procházíme nyní řetězy $w_i \in \{0,1\}^*$ v *kanonickém uspořádání* a definujeme jazyk L tímto způsobem: w_i patří do L pokud zároveň platí:

1. w_i kóduje nějakou kontextovou gramatiku G_i,
2. $w_i \notin L(G_i)$.

Jazyk L je rekurzivní, protože $L(G_i)$ je rekurzivní, není ale kontextový – ukáže se variací na diagonální schéma.

Chomskyho hierarchie jazyků



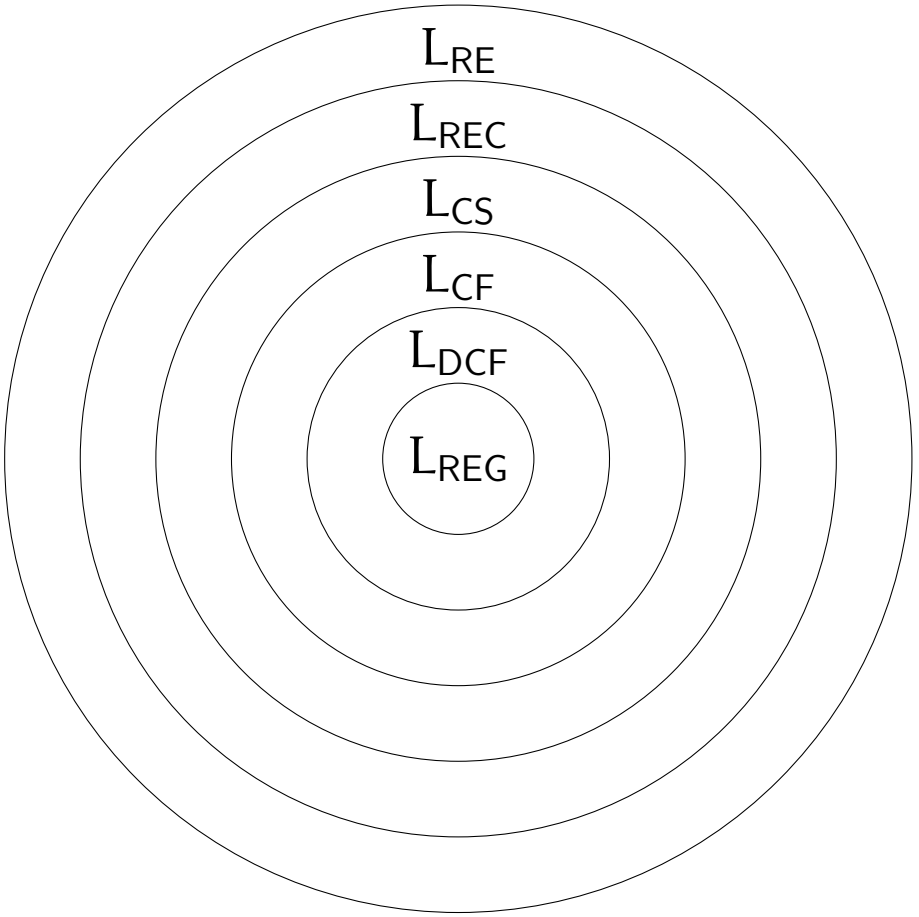
L_{RE}: rekurzivně spočetné jazyky (typ 0),

L_{CS}: kontextové jazyky (typ 1),

L_{CF}: bezkontextové jazyky (typ 2),

L_{REG}: regulární jazyky (typ 3).

Podrobnější klasifikace



L_{DCF}: deterministické bezkontextové, L_{REC}: rekurzivní.

Meze možností algoritmických výpočtů

Existují úlohy, pro něž žádný algoritmus nemůže existovat.

Rozhodovací problém: zodpovězení určité otázky pomocí „ano" nebo „ne".

Například: „Je bezkontextová gramatika G jednoznačná?"

Řešením rozhodovacího problému se rozumí algoritmus, který dá odpověď pro libovolný prvek třídy objektů, o něž se jedná, tedy např. bezkontextové gramatiky. Takový problém se nazývá *rozhodnutelný*.

Problém zastavení Turingova stroje

Existuje algoritmus, který pro libovolný Turingův stroj a libovolný počáteční řetěz zapsaný na pásce rozhodne, zda výpočet skončí nebo ne?

Definice 45

Označme symbolem $w_M \in \{0,1\}^*$ řetěz popisující Turingův stroj

$$M = (Q, \{0,1\}, \Gamma, \delta, q_0, \square, F).$$

Řešením problému zastavení je Turingův stroj H, který pro libovolný popis w_M a libovolný řetěz $w \in \{0,1\}^*$ provede tento výpočet:

1. $q_0 w_M w \overset{*}{\vdash} x_1 q_y x_2$, pokud se výpočet stroje M aplikovaný na w zastaví;
2. $q_0 w_M w \overset{*}{\vdash} y_1 q_n y_2$, pokud se výpočet stroje M aplikovaný na w nezastaví.

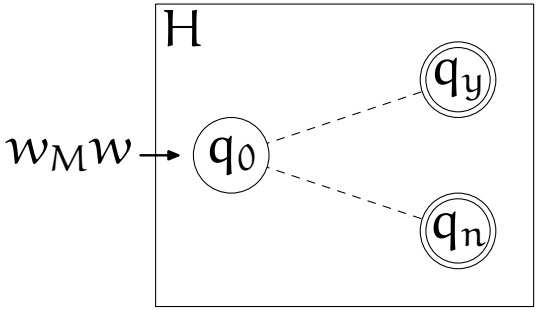
Oba stavy q_y a q_n jsou přitom koncovými stavy stroje M.

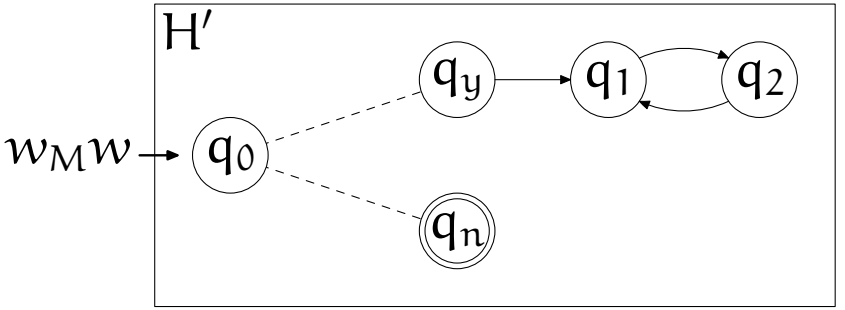
Problém zastavení nemá řešení

Věta 47

Neexistuje Turingův stroj, který by byl řešením problému zastavení podle definice 45.

Důkaz sporem: předpokládejme, že by takový Turingův stroj H existoval.





Zkonstruujeme Turingův stroj $\hat{\mathrm{H}}$, který zkopíruje kódovaný popis Turingova stroje w_M na pásce, přejde do stavu q_0 a dále postupuje přesně stejně jako H'.

Aplikujeme-li $\hat{H}$ na popis libovolného Turingova stroje M, dostaneme

1. $q_0 w_M \overset{*}{\vdash}_{\hat{H}} q_0 w_M w_M \overset{*}{\vdash}_{\hat{H}} \infty$, pokud se M aplikovaný na w_M zastaví;
2. $q_0 w_M \overset{*}{\vdash}_{\hat{H}} q_0 w_M w_M \overset{*}{\vdash}_{\hat{H}} y_1 q_n y_2$, pokud se M aplikovaný na w_M nezastaví.

V roli testovaného Turingova stroje M teď použijeme samotný $\hat{H}$:

1. $q_0 w_{\hat{H}} \overset{*}{\vdash}_{\hat{H}} q_0 w_{\hat{H}} w_{\hat{H}} \overset{*}{\vdash}_{\hat{H}} \infty$, pokud se $\hat{H}$ aplikovaný na $w_{\hat{H}}$ zastaví;
2. $q_0 w_{\hat{H}} \overset{*}{\vdash}_{\hat{H}} q_0 w_{\hat{H}} w_{\hat{H}} \overset{*}{\vdash}_{\hat{H}} y_1 q_n y_2$, pokud se $\hat{H}$ aplikovaný na $w_{\hat{H}}$ nezastaví.

Toto tvrzení je nesmyslné, tím dostáváme spor.

Jednoduchý důkaz

Věta 48

Pokud by problém zastavení Turingova stroje byl rozhodnutelný, musel by každý rekurzivně spočetný jazyk být rekurzivní. (Z toho tedy plyne, že problém zastavení Turingova stroje rozhodnutelný není.)

Opět dokazujeme sporem: předpokládáme, že existuje T. s. H, který rozhoduje problém zastavení libovolného Turingova stroje.

Rekurzivně spočetný jazyk $\mathrm{L} \longrightarrow$ Turingův stroj M, který jazyk L přijímá.

Pro každý řetěz w bychom tedy napřed nechali stroj H rozhodnout, zda se M zastaví pro vstupní řetěz w – pokud se dozvíme, že ne, pak w nepatří do L. Když ano, necháme rozhodnout stroj M.

Odvozené nerozhodnutelné problémy

1. Vstoupí Turingův stroj během výpočtu do daného stavu q?
2. Zastaví se Turingův stroj, pokud zahájí výpočet s prázdnou páskou?

Oba tyto problémy lze *redukovat* na problém zastavení Turingova stroje.

Nerozhodnutelné problémy RE jazyků

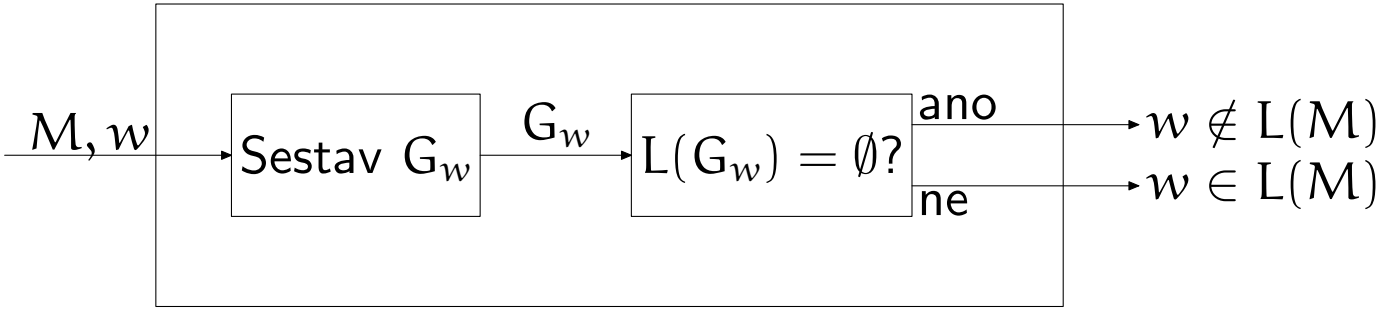
Věta 49

Otázka, zda pro libovolnou neomezenou gramatiku G je jazyk $L(G)$ prázdný, není rozhodnutelný problém.

Pro zadaný Turingův stroj M a řetěz w sestrojíme T. s. M_w:

$$L(M_w) = L(M) \cap w$$

K němu najdeme gramatiku G_w postupem podle věty 12.7.

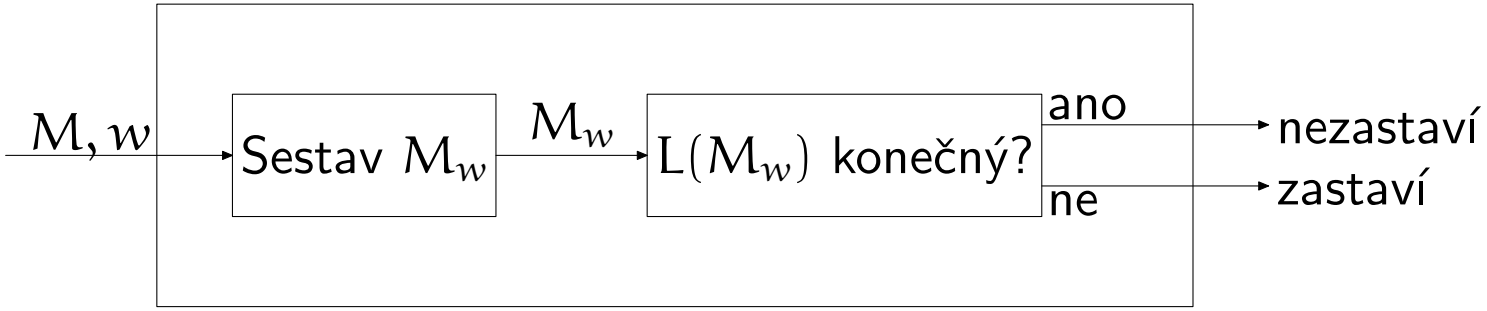


Věta 50

Otázka, zda pro libovolný Turingův stroj M je jazyk $L(M)$ konečný, není rozhodnutelný problém.

Pro zadaný Turingův stroj M a řetěz w sestrojíme T. s. M_w:

$$L(M_w) = \begin{cases} \Sigma^+, & \text{pokud } M \text{ přijme } w; \\ \emptyset, & \text{jinak.} \end{cases}$$



Riceova věta

Stejným způsobem jako u věty 50 bychom mohli ukázat, že pro třídu Turingových strojů M nad vstupní abecedou $\Sigma = \{0, 1\}$ jsou například následující problémy nerozhodnutelné:

- Obsahuje $\mathrm{L}(M)$ aspoň dva různé řetězy stejné délky?
- Obsahuje $\mathrm{L}(M)$ řetěz délky 5?
- Je $\mathrm{L}(M)$ regulární jazyk?

Věta 51

Každá netriviální vlastnost rekurzivně spočetného jazyka je nerozhodnutelná.

Netriviální vlastnost – platí pro některé, ale ne pro všechny RE jazyky.