Teoretická informatika

TIN 2021/2022

doc. RNDr. Milan Češka, Ph.D.

ceskam@fit.vutbr.cz

Upravené přednášky prof. Češky a prof. Vojnara

Forma kurzu a novinky

- přednášky (primárně doc. Milan Češka)
- demo cvičení (primárně prof. Tomáš Vojnar)
- numerická cvičení (prof. Vojnar, doc. Češka, doc. Holík, doc. Rogalewicz, dr. Lengál)
- demo cvičení (pátek) se nepravidelně střídají s numerickými cvičeními (po a pa)
- přednášky a dema se zaznamenávají a budou průběžně zveřejňovány na video serveru
- hvězdičkové notace pro doplňující témata/důkazy, které se nebudou zkoušet
- 2 vnitrosemestrální testy (10+15 bodů) + 3 domácí úkoly (3x 5 bodů, příprava na testy/zkoušky)
- zápočet min 15 bodů z písemek a prvních dvou úkolů
- NOVINKA: sbírka příkladů s ukázkou řešení
- HLAVNÍ INFORMAČNÍ KANÁL: stránky předmětu `https://www.fit.vutbr.cz/study/courses/TIN/public/.en`

Referenční literatura

- Opora – dostupná na stránkách předmětu.

- Předmět vychází zejména z následujících zdrojů:
  - Češka, M.: Teoretická informatika, učební text FIT VUT v Brně, 2002. `http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/ti.pdf`
  - Kozen, D.C.: Automata and Computability, Springer-Verlag, New York, Inc, 1997. ISBN 0-387-94907-0
  - Černá, I., Křetínský, M., Kučera, A.: Automaty a formální jazyky I, učební text FI MU, Brno, 1999.
  - Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, Addison Wesley, 2. vydání, 2000. ISBN 0-201-44124-1
  - Gruska, J.: Foundations of Computing, International Thomson Computer Press, 1997. ISBN 1-85032-243-0
  - Bovet, D.P., Crescenzi, P.: Introduction to the Theory of Complexity, Prentice Hall Europe, Pearson Education Limited, 1994. ISBN 0-13-915380-2

Motivace: Proč se učíme TIN?

- prohloubit znalosti základních informatických konceptů (automaty, gramatiky, regulární výrazy) včetně algoritmů pro práci s nimi, které se přímo používají v mnohých reálných aplikacích (překladače, bezpečnost, analýza systémů atd.)

- získat základní orientaci v oblasti *vyčíslitelnosti a složitosti* (tj. jaké problémy umíme algoritmicky řešit a jaké výpočetní prostředky jsou nutné k jejich řešení)

- osvojit si schopnost *abstraktního* a *systematického* myšlení

- osvojit si schopnost *formálního* (tj. přesného) vyjadřovaní

Reálný problém řešený v rámci jedné BP

- Vytvořte mobilní aplikaci pro následující problém plánování aktivit na dětském táboře. Je dáno m disciplín a n dětí – každé dítě má vybrané dvě oblíbené disciplíny. Vaším úkolem je najít (pokud existuje) bezkonfliktní rozdělení disciplín do 3 skupin (rund). Konflikt nastává pokud obě disciplíny vybrané jedním účastníkem přísluší stejné skupině.

- Je tento problém algoritmicky řešitelný?

- Je tento problém efektivně řešitelný?

- Jaká je časová složitost tohoto problému? Pro jak velké hodnoty m a n můžeme očekávat existenci algoritmu, který tento problém vyřeší v řádech sekund/minut.

- Dovednosti získané v TINu nám pomohou takové otázky zodpovědět a tudíž lépe pochopit daný problém

Formulace problému

❖ Vytvořte mobilní aplikaci pro následující problém plánování aktivit na dětském táboře. Je dáno m disciplín a n dětí – každé dítě má vybrané dvě oblíbené disciplíny. Vaším úkolem je najít (pokud existuje) bezkonfliktní rozdělení disciplín do 3 skupin (rund). Konflikt nastává pokud obě disciplíny vybrané jedním účastníkem přísluší stejné skupině.

1. množina disciplín $D = \{d_1, d_2, \dots d_m\}$
2. výběry dětí: $V = \{v_1, v_2, \dots, v_n\}$, kde $v_i = \{d_{i1}, d_{i2}\}$ pro $d_{i1}, d_{i2} \in D$.
3. rozdělení do skupin: funkce $p : D \to \{1, 2, 3\}$, kde $\forall v_i \in V : p(d_{i1}) \neq p(d_{i2})$

❖ Je tento problém algoritmicky řešitelný? Ano, existuje pouze konečný počet funkcí p mezi dvěma konečnými množinami

❖ Je tento problém efektivně řešitelný? Naivní řešení musí vyzkoušet až 3^m možných funkcí p a pro každou ověřit existenci konfliktu – exponenciální složitost.

❖ Umím to dělat lépe? Určitě ano, ale jak moc?

Redukce na známý problém

1. množina disciplín $D = \{d_1, d_2, \dots d_m\}$
2. výběry dětí: $V = \{v_1, v_2, \dots, v_n\}$, kde $v_i = \{d_{i1}, d_{i2}\}$ pro $d_{i1}, d_{i2} \in D$.
3. rozdělení do skupin: funkce $p : D \to \{1, 2, 3\}$, kde $\forall v_i \in V : p(d_{i1}) \neq p(d_{i2})$

- Barvení grafů – 3 obarvitelnost

1. D je množina vrcholů
2. $V = \{v_1, v_2, \dots, v_n\}$ je množina hran, kde $v_i = \{d_{i1}, d_{i2}\}$ je hrana mezi vrcholy d_{i1} a d_{i2}.
3. $p : D \to \{1, 2, 3\}$ je barvení grafu pomocí 3 barev.
4. existuje obarvení p, kde $\forall v_i \in V : p(d_{i1}) \neq p(d_{i2})$

Redukce na známý problém

- 3 obarvitelnost je **NP**-úplný problém: a) není znám žádný efektivní (tj. polynomiální algoritmus), b) existence takového algoritmu by znamenala zásadní průlom v computer science.

- Nemá (asi) cenu hledat obecné efektivní řešení našeho problému, ale spíše se soustředit na heuristiky, které mohou fungovat v prakticky zajímavých instancích: inspirace z řešení problému 3 obarvitelnost – branch-bound algoritmy

- Netriviální instance 3 obarvitelnosti pro $m > 32$ už nejsou řešitelné v rozumném čase.

Jazyky a jejich reprezentace

Formální jazyky

- Prvotní pojmy: symbol, abeceda (neprázdná konečná množina symbolů).

Definice 1.1 Nechť Σ je abeceda. Označme Σ^* množinu všech konečných posloupností w tvaru:

$$w = a_1 a_2 \dots a_n, a_i \in \Sigma \text{ pro } i = 1, \dots, n.$$

Posloupnosti w nazýváme řetězce nad abecedou Σ. Dále definujeme délku $|w|$ řetězce w jako $|w| = n$. Množina Σ^* obsahuje také speciální řetězec ε, pro který platí $|\varepsilon| = 0$. ε se nazývá prázdný řetězec.

Definice 1.2 Množinu L, pro kterou platí $L \subseteq \Sigma^*$ nazýváme formálním jazykem nad abecedou Σ.

- Příklady jazyků nad abecedou $\Sigma = \{0, 1\}$:
  - $L_1 = \{01, 0011\}$
  - $L_2 = \{0^n 1^n \mid n \geq 0\}$
  - $L_3 = \{ww^R \mid w \in \{0, 1\}^+\}$

❖ Na množině Σ^* zavedeme operaci $\cdot$ takto:
Jsou-li dány dva řetězce $w, w' \in \Sigma^*$:

$$w = a_1 a_2 \dots a_n,$$
$$w' = a'_1 a'_2 \dots a'_m \; n, m \geq 0,$$

pak $w \cdot w' = a_1 a_2 \dots a_n a'_1 a'_2 \dots a'_m \; (= ww')$.

Operace $\cdot$ se nazývá zřetězení nebo konkatenace.
Pro w, w', w'' platí:

1. $|ww'| = |w| + |w'|$,
2. $w(w'w'') = (ww')w''$ tj. asociativnost konkatenace,
3. $w\varepsilon = \varepsilon w = w$ tj. ε je jednotkový prvek vzhledem k operaci $\cdot$.

❖ Dále zavádíme pojmy:

- prefix, sufix, podřetězec,
- $a^i = a_1 a_2 \; \dots a_i$ kde $\forall 1 \leq j \leq i : a_j = a$
- $\#_a(w)$ je počet znaků $a \in \Sigma$ v řetězci $w \in \Sigma^*$
- w^R (tj. revers řetězce): pro $w = a_1 a_2 \dots a_n$ je $w^R = a_n a_{n-1} \dots a_1$

Operace nad jazyky

Jazyk je množina $\rightarrow$ jsou definovány všechny množinové operace nad jazyky.
Připomeňme operaci komplement jazyka L nad abecedou Σ (často značený jako $co{-}L$), který je definován jako $co{-}L = \Sigma^* \setminus L$.

Definice 1.3 Nechť L_1 je jazyk nad abecedou Σ_1, L_2 je jazyk nad abecedou Σ_2.
Součinem (konkatenací) jazyků L_1 a L_2 nad abecedou $\Sigma_1 \cup \Sigma_2$ rozumíme jazyk

$$L_1 \cdot L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$$

Příklad 1.1 Nechť $P = \{A, B, \ldots, Z, a, b, \ldots, z\}$, $C = \{0, 1, \ldots, 9\}$ jsou abecedy, $L_1 = P$ a $L_2 = (P \cup C)^*$ jazyky nad P resp. $P \cup C$.
Jaký jazyk určuje součin $L_1 L_2$?

Iterace a pozitivní iterace

Definice 1.4 Nechť L je jazyk. Iterací L^* jazyka L a pozitivní iterací L^+ jazyka L definujeme takto:

1. $L^0 = \{\varepsilon\}$
2. $L^n = L \cdot L^{n-1}$ pro $n \geq 1$
3. $L^* = \bigcup\limits_{n \geq 0} L^n$
4. $L^+ = \bigcup\limits_{n \geq 1} L^n$

Je-li L jazyk, pak platí:

1. $L^* = L^+ \cup \{\varepsilon\}$
2. $L^+ = L \cdot L^* = L^* \cdot L$

Příklad 1.2 $L_1 = \{(p\}, L_2 = \{,p\}, L_3 = \{)\}$

$$L_1 L_2^* L_3 = \{(p),\ (p,p),\ (p,p,p),\ \ldots\}$$

Algebra jazyků

Definice 1.5 Algebraická struktura $< A,\ +,\ \cdot,\ 0,\ 1 >$ se nazývá polookruh, jestliže:

1. $< A,\ +,\ 0 >$ je komutativní monoid (tj. $+$ je asociativní a 0 je neutrální prvek) ,
2. $< A,\ \cdot,\ 1 >$ je monoid,
3. pro operaci $\cdot$ platí distributivní zákon vzhledem k $+$:
   $\forall a, b, c \in A : a(b + c) = ab + ac, (a + b)c = ac + bc$.

Věta 1.1 Algebra jazyků $< 2^{\Sigma^*},\ \cup,\ \cdot,\ \emptyset,\ \{\varepsilon\} >$, kde $\cup$ je sjednocení a $\cdot$ konkatenace jazyků tvoří polookruh.

Důkaz.

1. $< 2^{\Sigma^*},\ \cup,\ \emptyset >$ je komutativní monoid ($\cup$ je komutativní a asociativní operace a $L \cup \emptyset = \emptyset \cup L = L$ pro všechna $L \in 2^{\Sigma^*}$).
2. $< 2^{\Sigma^*},\ \cdot,\ \{\varepsilon\} >$ je monoid:
   $L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L$ pro všechna $L \in 2^{\Sigma^*}$.
3. Pro všechny $L_1, L_2, L_3 \in 2^{\Sigma^*}$:
   $L_1(L_2 \cup L_3) = \{xy \mid (x \in L_1) \wedge (y \in L_2 \vee y \in L_3)\} = \ldots = L_1L_2 \cup L_1L_3$

Důkazy identity jazyků

Příklad:

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
2. $\exists L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
3. $\forall L \subseteq \Sigma^* : L^* = (L^*)^*$

Důkazy identity jazyků

Příklad:

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
2. $\exists L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
3. $\forall L \subseteq \Sigma^* : L^* = (L^*)^*$

Řešení 1: Tvrzení neplatí to jest platí jeho negace $\exists L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* \neq (L_1 \cup L_2)^*$
Zvolme například $L_1 = \{a\}$ a $L_2 = \{b\}$. Pak $L_1^* \cup L_2^* = \{a\}^* \cup \{b\}^*$ ale $(L_1 \cup L_2)^* = \{a, b\}^*$. Zřejmě $\{a\}^* \cup \{b\}^* \neq \{a, b\}^*$.

Důkazy identity jazyků

Příklad:

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
2. $\exists L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
3. $\forall L \subseteq \Sigma^* : L^* = (L^*)^*$

Řešení 1: Tvrzení neplatí to jest platí jeho negace $\exists L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* \neq (L_1 \cup L_2)^*$
Zvolme například $L_1 = \{a\}$ a $L_2 = \{b\}$. Pak $L_1^* \cup L_2^* = \{a\}^* \cup \{b\}^*$ ale
$(L_1 \cup L_2)^* = \{a, b\}^*$. Zřejmě $\{a\}^* \cup \{b\}^* \neq \{a, b\}^*$.

Řešení 2: Tvrzení platí. Zvolme například $L_1 = L_2 = \{a\}$. Pak
$L_1^* \cup L_2^* = \{a\}^* = (L_1 \cup L_2)^*$.

Důkazy identity jazyků

Příklad:

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
2. $\exists L_1, L_2 \subseteq \Sigma^* : L_1^* \cup L_2^* = (L_1 \cup L_2)^*$
3. $\forall L \subseteq \Sigma^* : L^* = (L^*)^*$

Řešení 3: Tvrzení platí. Notace: w_0 i w_i^0 (pro $i \geq 1$) označuje ϵ

$$L^* = \bigcup_{n \geq 0} L^n = \{w \mid w \in L^i \wedge i \geq 0\} = \{w_1 w_2 \dots w_n \mid n \geq 0 \wedge \forall 1 \leq i \leq n : w_i \in L\}$$

$$\begin{aligned}
&(L^*)^* = \{w_1 w_2 \dots w_n \mid n \geq 0 \wedge \forall 1 \leq i \leq n : w_i \in L^{m_i} \wedge m_i \geq 0\} = \\
&\{w_1^1 w_1^2 \dots w_1^{m_1} w_2^1 w_2^2 \dots w_2^{m_2} \dots w_n^1 w_n^2 \dots w_n^{m_n} \mid n \geq 0 \wedge \\
&\qquad \forall 1 \leq i \leq n : m_i \geq 0 \wedge \forall 1 \leq j \leq \max\{m_i \mid 1 \leq i \leq n\} : w_i^j \in L\} = \\
&\{w_1 w_2 \dots w_k \mid k = m_1 + m_2 + \dots m_n \wedge \forall 1 \leq i \leq k : w_k \in L\} = \\
&\{w_1 w_2 \dots w_k \mid k \geq 0 \wedge \forall 1 \leq i \leq k : w_i \in L\} = L^*
\end{aligned}$$

Gramatiky

❖ Pozn. Reprezentace jazyků – problém reprezentace, způsoby reprezentace.

Definice 1.6 Gramatika G je čtveřice $G = (N,\ \Sigma,\ P,\ S)$, kde

1. N je konečná množina nonterminálních symbolů.
2. Σ je abeceda (tj. konečná množina terminálních symbolů), kde $N \cap \Sigma = \emptyset$.
3. P je konečná podmnožina kartézského součinu

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

   nazývaná množina přepisovacích pravidel
4. $S \in N$ je výchozí (startovací) symbol gramatiky G.

❖ Prvek $(\alpha, \beta) \in P$ je přepisovací pravidlo a zapisuje se ve tvaru

$$\alpha \rightarrow \beta,$$

kde α je levá strana, β je pravá strana pravidla $\alpha \rightarrow \beta$.

Příklady:

- $G_1 = (\{A, S\}, \{0, 1\}, P_1, S)$

$$P_1: \quad \begin{array}{rcl} S & \rightarrow & 0A1 \\ 0A & \rightarrow & 00A1 \\ A & \rightarrow & \varepsilon \end{array}$$

- $G_2 = (N_2, \Sigma_2, P_2, I)$
  $N_2 = \{I, P, C\}$
  $\Sigma_2 = \{a, b, \ldots, z, 0, 1, \ldots, 9\}$

$$P_2: \quad \begin{array}{rclcrcl} I & \rightarrow & P & & & & \\ I & \rightarrow & IP & & & & \\ I & \rightarrow & IC & & & & \\ P & \rightarrow & a & & C & \rightarrow & 0 \\ P & \rightarrow & b & & C & \rightarrow & 1 \\ & \vdots & & & & \vdots & \\ P & \rightarrow & z & & C & \rightarrow & 9 \end{array}$$

❖ Konvence 1: Obsahuje-li množina P přepisovací pravidla tvaru

$$\alpha \to \beta_1, \alpha \to \beta_2, \ldots, \alpha \to \beta_n$$

pak pro zkrácení budeme používat zápisu

$$\alpha \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

❖ Konvence 2: Pro zápis symbolů a řetězců budeme užívat této úmluvy:

1. a, b, c, d reprezentují terminální symboly.
2. A, B, C, D, S reprezentují nonterminální symboly, S výchozí symbol.
3. $U, V, \ldots, Z$ reprezentují terminální nebo nonterminální symboly.
4. $\alpha, \beta, \ldots, \omega$ reprezentují řetězce z množiny $(N \cup \Sigma)^*$.
5. $u, v, w, \ldots, z$ reprezentují řetězce z Σ^*.

Definice 1.7 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť λ, μ jsou řetězce z $(N \cup \Sigma)^*$. Mezi λ a μ platí binární relace $\underset{G}{\Rightarrow}$, zvaná přímá derivace, můžeme-li řetězce λ a μ vyjádřit ve tvaru

$$\lambda = \gamma\alpha\delta$$
$$\mu = \gamma\beta\delta$$

$\gamma, \delta \in (N \cup \Sigma)^*$ a $\alpha \rightarrow \beta$ je nějaké přepisovací pravidlo z P. Pak píšeme

$$\lambda \underset{G}{\Rightarrow} \mu \text{ nebo}$$
$$\lambda \Rightarrow \mu.$$

Definice 1.8 Nechť $G = (N, \Sigma, P, S)$ je gramatika a $\Rightarrow$ relace přímé derivace na $(N \cup \Sigma)^*$. Relace $\overset{+}{\Rightarrow}$ označuje tranzitivní uzávěr relace $\Rightarrow$ a nazývá se relací derivace. Platí-li $\lambda \overset{+}{\Rightarrow} \mu$, pak existuje posloupnost

$$\lambda = \nu_0 \Rightarrow \nu_1 \Rightarrow \ldots \Rightarrow \nu_n = \mu, \; n \geq 1,$$

která se nazývá derivací délky n.

- Relace $\overset{*}{\Rightarrow}$ označuje reflexivní a tranzitivní uzávěr relace $\Rightarrow$:

$$\lambda \overset{*}{\Rightarrow} \mu \qquad \Rightarrow \qquad \lambda \overset{+}{\Rightarrow} \mu \text{ nebo } \lambda = \mu$$

Příklad 1.3 Derivace v gramatice G_1, resp. G_2, ze strany 11:

- V gramatice G_1:
  - Pravidlo $0A \to 00A1$ implikuje $0^n A1^n \Rightarrow 0^{n+1} A1^{n+1}$,
  - tedy $0A1 \overset{*}{\Rightarrow} 0^n A1^n$ pro libovolné $n > 0$.

- V gramatice G_2:
  - $I \Rightarrow IP \Rightarrow IPP \Rightarrow ICPP \Rightarrow PCPP \Rightarrow aCPP \Rightarrow a1PP \Rightarrow a1xP \Rightarrow a1xy$,
  - tj. $I \overset{+}{\Rightarrow} a1xy$.

Definice 1.9 Nechť $G = (N, \Sigma, P, S)$ je gramatika. Řetězec $\alpha \in (N \cup \Sigma)^*$ nazýváme větnou formou, platí-li $S \stackrel{*}{\Rightarrow} \alpha$. Větná forma, která obsahuje pouze terminální symboly se nazývá věta.

Jazyk $L(G)$ generovaný gramatikou G je množina:

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

Příklad 1.4

$$L(G_1) = \{0^n 1^n \mid n > 0\}$$

protože

$$\begin{array}{ll} S \Rightarrow 0A1 & \\ S \stackrel{*}{\Rightarrow} 0^n A 1^n & \text{(viz předchozí příklad)} \\ S \stackrel{*}{\Rightarrow} 0^n 1^n & \text{(pravidlo } A \to \varepsilon\text{)} \end{array}$$

Chomského hierarchie

Je definována na základě tvaru přepisovacích pravidel:

- Typ 0 – obecné (neomezené) gramatiky:

$$\alpha \to \beta \qquad \alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*, \beta \in (N \cup \Sigma)^*$$

- Typ 1 – kontextové gramatiky:

$$\alpha A \beta \to \alpha \gamma \beta \qquad A \in N,\ \alpha, \beta \in (N \cup \Sigma)^*, \gamma \in (N \cup \Sigma)^+$$

nebo $S \to \varepsilon$, pakliže se S neobjevuje na pravé straně žádného pravidla

(Alternativní definice definující stejnou třídu jazyků: $\alpha \to \beta,\ |\alpha| \leq |\beta|$ nebo $S \to \varepsilon$ omezené jako výše.)

- Typ 2 – bezkontextové gramatiky:

$$A \to \alpha \qquad A \in N,\ \alpha \in (N \cup \Sigma)^*$$

- Typ 3 – regulární gramatiky:

$$A \to xB \qquad \text{nebo}$$
$$A \to x \mid \varepsilon \qquad A, B \in N,\ x \in \Sigma$$

(Alternativní tvary gramatik, které mají stejnou vyjadřovací sílu, jsou uvedeny v opoře.)

Definice 1.10 Jazyk generovaný gram. typu $i \in \{0, 1, 2, 3\}$, se nazývá jazykem typu i.

Existuje synonymní označení jazyků:

- Jazyk typu 0 — rekurzivně vyčíslitelný jazyk.
- Jazyk typu 1 — kontextový jazyk.
- Jazyk typu 2 — bezkontextový jazyk.
- Jazyk typu 3 — regulární jazyk.

Věta 1.2 Nechť $\mathcal{L}_i$ značí třídu všech jazyků typu i nad abecedou Σ.
Pak platí:

$$\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_0$$

Důkaz.
Důkaz plyne z definice tříd Chomského hierarchie jazyků.

$\square$

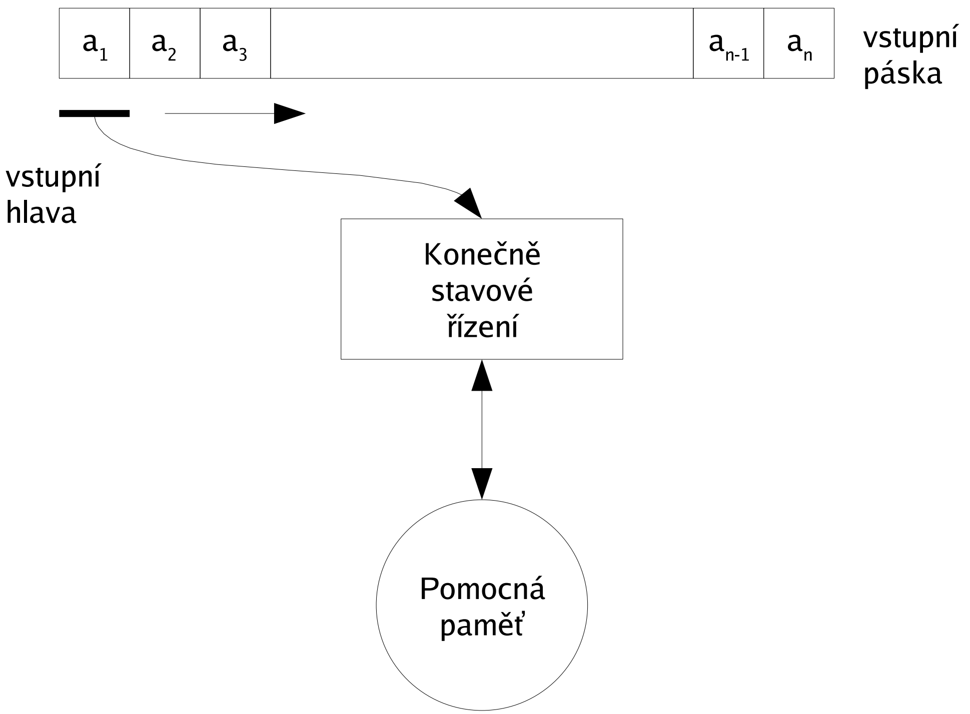
Věta 1.3 Platí:

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$

Důkaz jednotlivých inkluzí v průběhu semestru.

Regulární jazyky a Konečné automaty

Automaty



❖ Klasifikace:
- podle mechanismu a funkce čtecí hlavy,
- pomocné paměti,

Konečný automat

Definice 1.11 Konečným nedeterministický automatem (NKA) rozumíme automat M specifikovaný 5-ticí

$$M = (Q, \Sigma, \delta, q_0, F), \qquad \text{kde:}$$

1. Q je konečná množina stavů,
2. Σ je konečná vstupní abeceda,
3. δ je funkce přechodů (přechodová funkce) tvaru $\delta : Q \times \Sigma \to 2^Q$,
4. $q_0 \in Q$ je počáteční stav,
5. $F \subseteq Q$ je množina koncových stavů.

Je-li $\forall q \in Q\ \forall a \in \Sigma : |\delta(q, a)| \leq 1$, pak M nazýváme deterministickým konečným automatem (zkráceně DKA).

Deterministický konečným automat je také často specifikován 5-ticí

$$M = (Q, \Sigma, \delta, q_0, F), \qquad \text{kde:}$$

- δ je parciální funkce tvaru $\delta : Q \times \Sigma \to Q$,
- a význam ostatních složek zůstává zachován.

Je-li přechodová funkce δ totální, pak M nazýváme úplně definovaným deterministickým konečným automatem.

Dále budeme obvykle pracovat s touto specifikací DKA.

Lemma 1.1 Ke každému DKA M existuje „ekvivalentní“ úplně definovaný DKA M'.

Důkaz. (idea) Množinu stavů automatu M' rozšíříme o nový, nekoncový stav (anglicky označovaný jako *SINK* stav) a s využitím tohoto stavu doplníme prvky přechodové funkce δ' automatu M' tak, aby byla totální. $\square$

Příklad 1.5 Zápis NKA.

$$\text{NKA } M_1 = (\{q_0, q_1, q_2, q_F\},\ \{0,1\},\ \delta,\ q_0,\ \{q_F\})$$

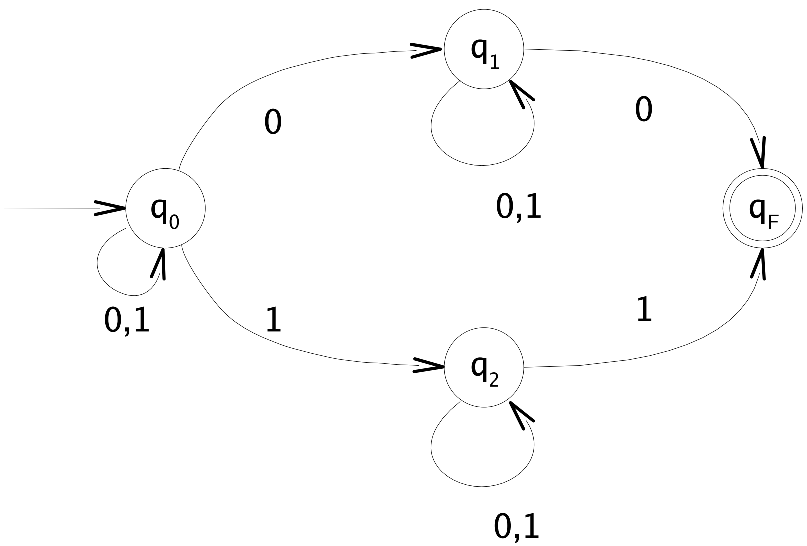
$$\begin{array}{lll}
\delta: & \delta(q_0,0) = \{q_0,q_1\} & \delta(q_0,1) = \{q_0,q_2\} \\
 & \delta(q_1,0) = \{q_1,q_F\} & \delta(q_1,1) = \{q_1\} \\
 & \delta(q_2,0) = \{q_2\} & \delta(q_2,1) = \{q_2,q_F\} \\
 & \delta(q_F,0) = \emptyset & \delta(q_F,1) = \emptyset
\end{array}$$

❖ Alternativní způsoby reprezentace funkce δ:

1. maticí (přechodů)

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_1, q_F\}$	$\{q_1\}$
q_2	$\{q_2\}$	$\{q_2, q_F\}$
q_F	$\emptyset$	$\emptyset$

2. diagramem přechodů



Definice 1.12 Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat (tj. NKA).

❖ Konfigurace C konečného automatu M je uspořádaná dvojice

$$C = (q, w), \quad (q, w) \in Q \times \Sigma^*$$

kde q je aktuální stav, w je dosud nezpracovaná část vstupního řetězce.

❖ Počáteční konfigurace je konfigurace $(q_0, a_1 a_2 \ldots a_n)$.

❖ Koncová konfigurace je konfigurace $(q_F, \varepsilon), \quad q_F \in F$.

❖ Přechodem automatu M rozumíme binární relaci $\underset{M}{\vdash}$ v množině konfigurací C

$$\underset{M}{\vdash} \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$$

která je definována takto:

$$(q, w) \underset{M}{\vdash} (q', w') \stackrel{def.}{\Longleftrightarrow} w = aw' \ \wedge \ q' \in \delta(q, a) \ \ pro \ q, q' \in Q, a \in \Sigma, w, w' \in \Sigma^*$$

Relace $\overset{k}{\underset{M}{\vdash}}, \overset{+}{\underset{M}{\vdash}}, \overset{*}{\underset{M}{\vdash}}$ mají obvyklý význam, tj. $k-$tá mocnina relace $\vdash$, tranzitivní a tranzitivní a reflexivní uzávěr relace $\vdash$.

❖ Řetězec w přijímaný NKA M je definován takto: $(q_0, w) \overset{*}{\underset{M}{\vdash}} (q, \varepsilon), \quad q \in F$.

❖ Jazyk $L(M)$ přijímaný NKA M je definován takto:

$$L(M) = \{w \mid (q_0, w) \overset{*}{\underset{M}{\vdash}} (q, \varepsilon) \wedge q \in F\}.$$

Příklad 1.6 Uvažujme NKA M_1 z příkladu 1.5. Platí:

$$(q_0, 1010) \vdash (q_0, 010) \vdash (q_1, 10) \vdash (q_1, 0) \vdash (q_f, \varepsilon)$$

a tedy: $(q_0, 1010) \overset{*}{\vdash} (q_f, \varepsilon)$

Neplatí například $(q_0, \varepsilon) \overset{*}{\vdash} (q_f, \varepsilon)$

Vyjádření jazyka $L(M_1)$:

$$L(M_1) = \{w \mid w \in \{0, 1\}^* \wedge w \text{ končí symbolem, který je již v řetězci } w \text{ obsažen}\}$$

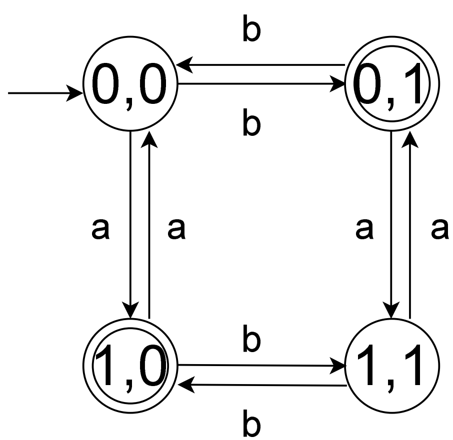
$\square$

Konstrukce konečných automatů

❖ Interpretace stavů: Stav reprezentuje informaci o průběhu výpočtu

Příklad: $L = \{w \in \{a,b\}^* \mid \#_a(w) \ \mathtt{mod}\ 2 \neq \#_b(w) \ \mathtt{mod}\ 2\}$

Stav kóduje dvoji (p,q) kde $p = \#_a(u) \ \mathtt{mod}\ 2$ a $q = \#_b(u) \ \mathtt{mod}\ 2$ a u je zatím přečtený vstup.



Konstrukce konečných automatů

❖ Interpretace stavů: Stav reprezentuje informaci o průběhu výpočtu

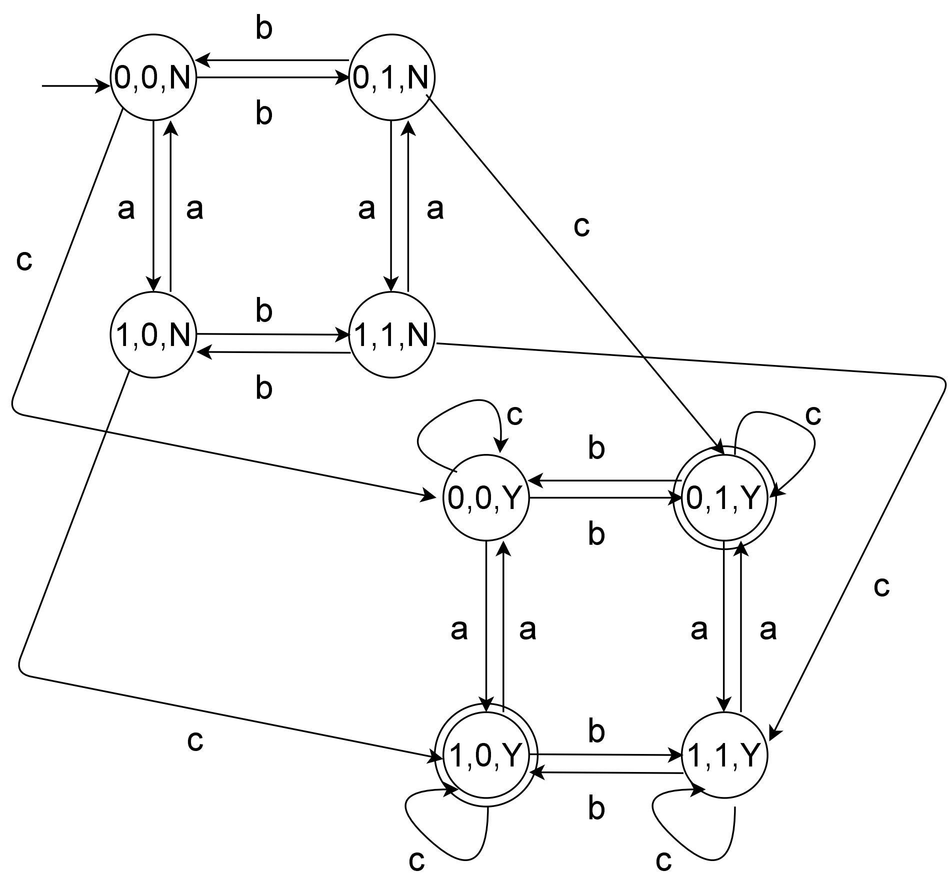
$$L = \{w \in \{a,b,c\}^* \mid \#_a(w) \texttt{ mod } 2 \neq \#_b(w) \texttt{ mod } 2 \wedge \#_c(w) > 0\}$$

Konstrukce konečných automatů

❖ Interpretace stavů: Stav reprezentuje informaci o průběhu výpočtu

$L = \{w \in \{a,b,c\}^* \mid \#_a(w) \texttt{ mod } 2 \neq \#_b(w) \texttt{ mod } 2 \wedge \#_c(w) > 0\}$

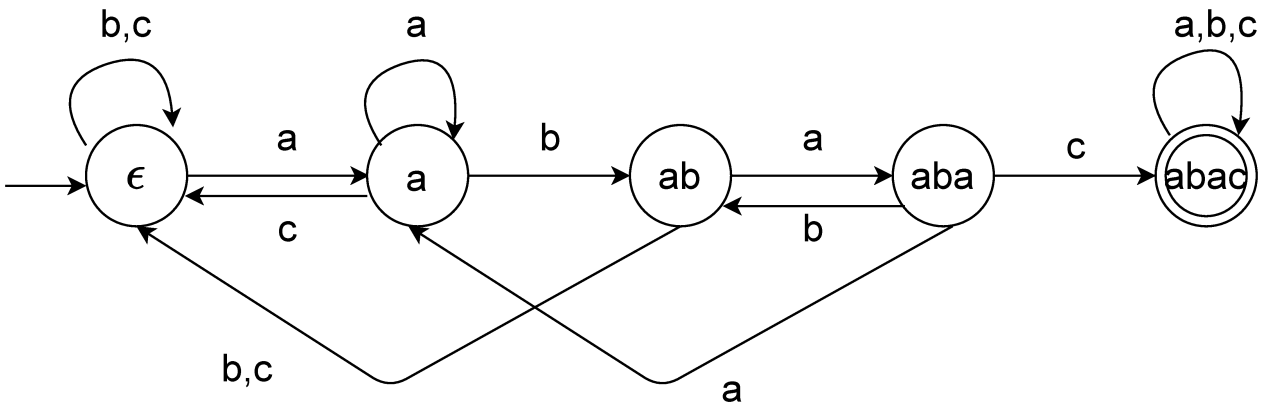
Stav kóduje trojici (p,q,r) kde $p = \#_a(u) \texttt{ mod } 2$ a $q = \#_b(u) \texttt{ mod } 2$ a $r = N$ pokud $\#_c(u) = 0$ a $r = Y$ pokud $\#_c(u) > 0$



Konstrukce konečných automatů

❖ Využití nedeterminismu: Stroj "uhádne" jistý aspekt výpočtu vedoucí k přijetí slova z jazyka.

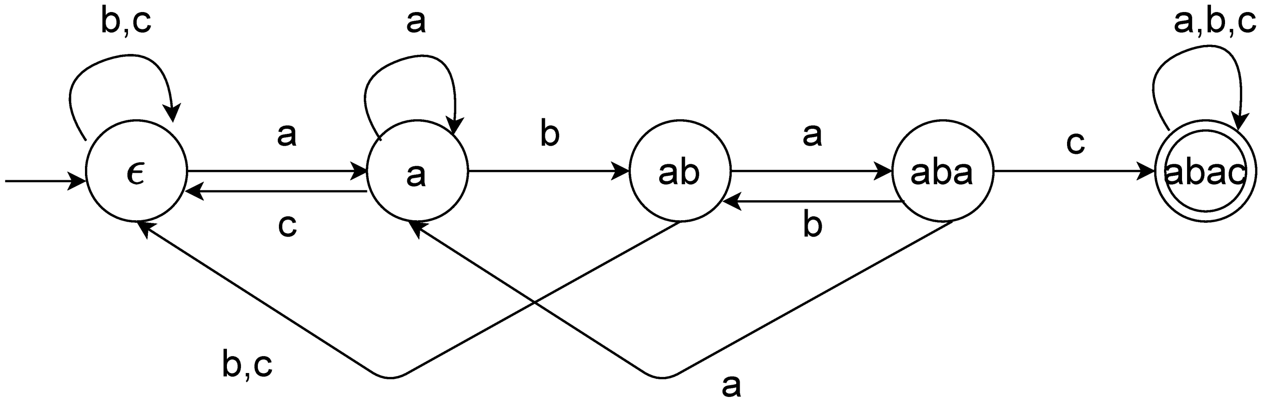
$L = \{w \in \{a, b, c\}^* \mid w$ obsahuje podslovo $abac\}$



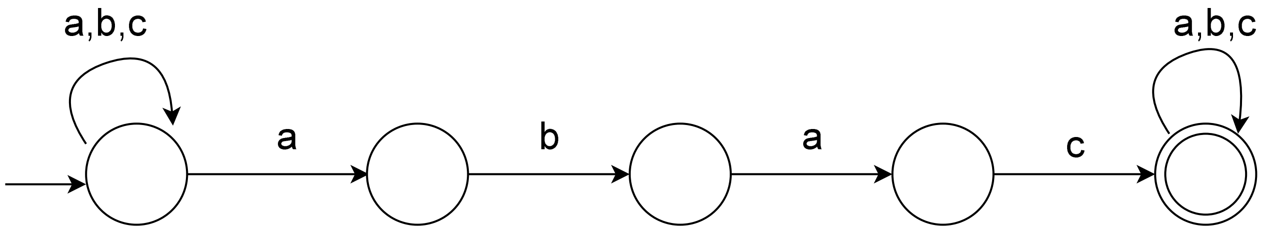
Konstrukce konečných automatů

❖ Využití nedeterminismu: Stroj "uhádne" jistý aspekt výpočtu vedoucí k přijetí slova z jazyka.

$L = \{w \in \{a,b,c\}^* \mid w$ obsahuje podslovo $abac\}$



❖ NKA "uhádne" kde začíná hledané podslovo a pouze ověří jeho tvar.



Ekvivalence NKA a DKA

Věta 1.4 Každý NKA M lze převést na DKA M' tak, že

$$L(M) = L(M').$$

Důkaz.

1. Nalezneme algoritmus převodu $M \to M'$ (níže).
2. Ukážeme, že $L(M) = L(M')$ tj. ukážeme, že platí:
   (a) $L(M) \subseteq L(M')$ a současně,
   (b) $L(M') \subseteq L(M)$.

$\square$

Převod NKA na ekvivalentní DKA

Algoritmus 1.1

- Vstup: NKA $M = (Q, \Sigma, \delta, q_0, F)$
- Výstup: DKA $M' = (Q', \Sigma, \delta', q_0', F')$, $L(M) = L(M')$
- Metoda:
  1. Polož $Q' = 2^Q$.
  2. Polož $q_0' = \{q_0\}$.
  3. Polož $F' = \{S \mid S \in 2^Q \wedge S \cap F \neq \emptyset\}$.
  4. Pro všechna $S \in 2^Q$ a pro všechna $a \in \Sigma$ polož:
     - $\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$.

Důkaz. $L(M) = L(M')$

Na deterministické automaty lze pohlížet jako na speciální případ nedeterministických automatů (tj. $\delta : Q \times \Sigma \to 2^Q$), kdy pro každý $q \in Q$ a $a \in \Sigma$ je množina $\delta(q, a)$ nanejvýš jednoprvková.

Zaveďme tedy rozšířenou přechodovou funkci $\hat{\delta} : Q \times \Sigma^* \to 2^Q$, kde

- $\hat{\delta}(q, \varepsilon) = \{q\}$
- $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a)$

Nyní ukážeme, že pro každé $w \in \Sigma^*$ platí $\hat{\delta}(q_0, w) = \hat{\delta}'(\{q_0\}, w)$. Indukcí k délce w dostáváme.

- Pro $|w| = 0$ platí $\hat{\delta}(q_0, \varepsilon) = \{q_0\} = \hat{\delta}'(\{q_0\}, \varepsilon)$.
- Indukční krok: Nechť $w = va$, kde $v \in \Sigma^*$ a $a \in \Sigma$. Pak platí $\hat{\delta}(q_0, va) = \bigcup_{p \in \hat{\delta}(q_0,v)} \delta(p, a) = \delta'(\hat{\delta}(q_0, v), a)$ (dle definice δ') $=$ $\delta'(\hat{\delta}'(\{q_0\}, v), a)$ (dle indukčního předpokladu) $= \hat{\delta}'(\{q_0\}, va)$.

Pak platí: $w \in L(M) \Leftrightarrow \hat{\delta}(q_0, w) \cap F \neq \emptyset \Leftrightarrow \hat{\delta}'(\{q_0\}, w) \cap F \neq \emptyset \Leftrightarrow w \in L(M')$.

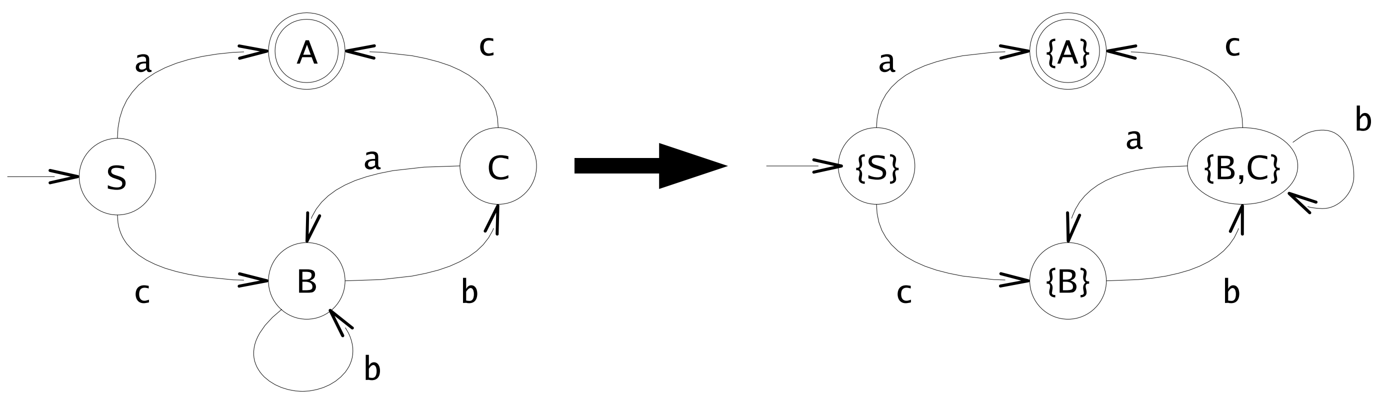
$\square$

Příklad 1.7 Uvažujme NKA $M_2 = (\{S, A, B, C\},\ \{a, b, c\},\ \delta,\ S,\ \{A\})$

$\delta: \quad \delta(S, a) = \{A\} \quad \delta(S, c) = \{B\} \quad \delta(B, b) = \{B, C\} \quad \delta(C, a) = \{B\} \quad \delta(C, c) = \{A\}$

K nalezení funkce δ' příslušného DKA aplikujeme zkrácený postup, využívající skutečnosti, že řada stavů z 2^Q může být nedostupných:

1. Počáteční stav: $\{S\}$
2. $\delta'(\{S\}, a) = \{A\}$ — koncový stav
   $\delta'(\{S\}, c) = \{B\}$
3. $\delta'(\{B\}, b) = \{B, C\}$
4. $\delta'(\{B, C\}, a) = \delta(B, a) \cup \delta(C, a) = \{B\}$
   $\delta'(\{B, C\}, b) = \{B, C\}$ $\delta'(\{B, C\}, c) = \{A\}$



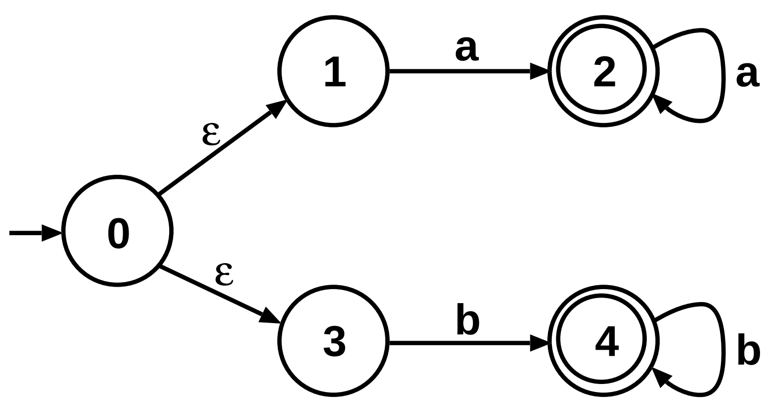
Rozšířené konečné automaty

❖ Dovolují jednodušší návrh a konstrukci automatů.

Definice 1.13 Rozšířený konečný automat (RKA) je pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde

- Q je konečná množina stavů,
- Σ je konečná vstupní abeceda,
- δ je zobrazení $Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$,
- $q_0 \in Q$ je počáteční stav,
- $F \subseteq Q$ je množina koncových stavů.

Příklad 1.8 $M = (\{0, 1, 2, 3, 4\}, \{a, b\}, \delta, 0, \{2, 4\})$



$$L(M) = aa^* + bb^* = a^+ + b^+$$

ε-uzávěr

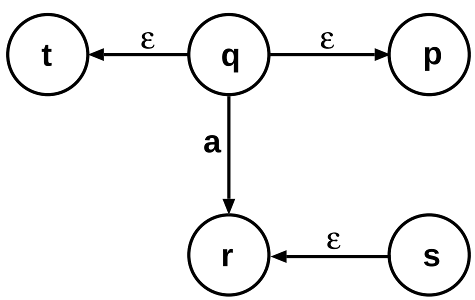
❖ Klíčovou funkci v algoritmu převodu RKA na DKA má výpočet funkce, která k danému stavu určí množinu všech stavů, jež jsou dostupné po ε hranách diagramu přechodů funkce δ. Označme tuto funkci jako ε-uzávěr:

$$\varepsilon\text{-uzávěr}(q) = \{p \mid \exists w \in \Sigma^* : (q, w) \overset{*}{\vdash} (p, w)\}$$

❖ Funkci ε-uzávěr zobecníme tak, aby argumentem mohla být množina $T \subseteq Q$:

$$\varepsilon\text{-uzávěr}(T) = \bigcup_{s \in T} \varepsilon\text{-uzávěr}(s)$$

Příklad 1.9



ε-uzávěr$(\{q, r, s\}) = \{p, q, r, s, t\}$

Výpočet ε-uzávěru

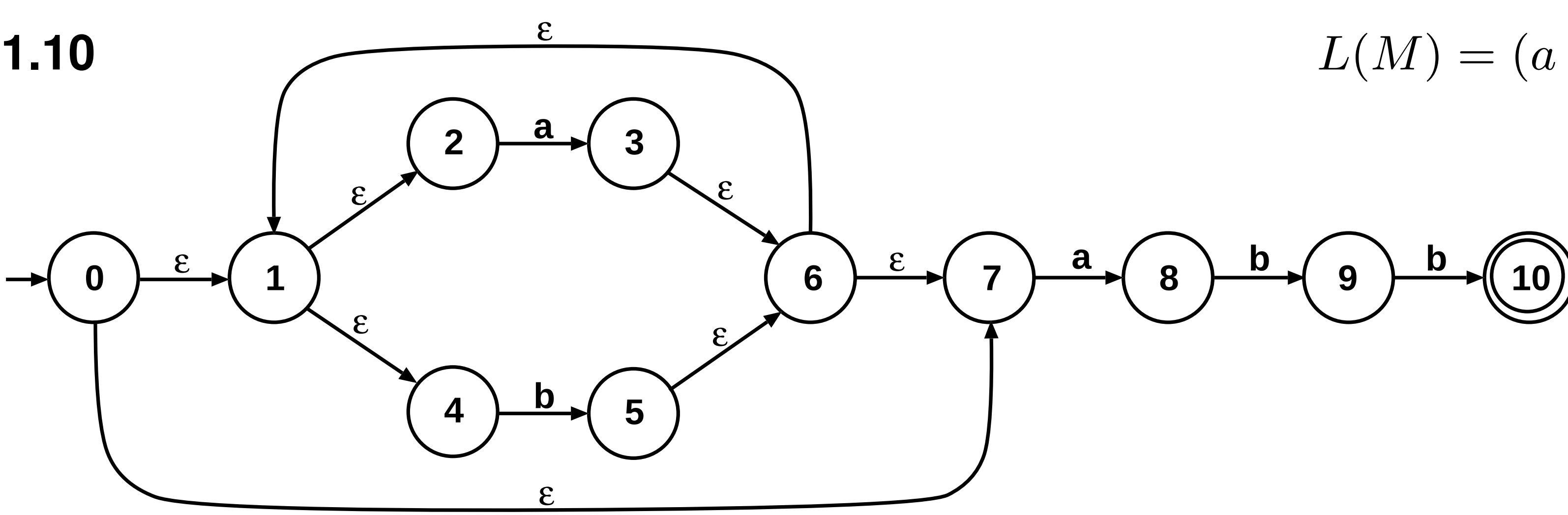
❖ Zavedeme relaci $\xrightarrow{\varepsilon}$ na množině Q takto:

$$\forall q_1, q_2 \in Q : q_1 \xrightarrow{\varepsilon} q_2 \overset{def}{\Longleftrightarrow} q_2 \in \delta(q_1, \varepsilon)$$

Pak ε-uzávěr$(p) = \{q \in Q \mid p \xrightarrow{\varepsilon}{}^* q\}$ je reflexivní a tranzitivní uzávěr relace $\xrightarrow{\varepsilon}{}^*$.

❖ K výpočtu tranzitivního uzávěr použijeme Warshallův algoritmus.

Příklad 1.10 $L(M) = (a+b)^* abb$



ε-uzávěr$(3) = \{3, 6, 7, 1, 2, 4\}$
ε-uzávěr$(\{1, 0\}) = \{0, 1, 2, 4, 7\}$

Převod RKA na ekvivalentní DKA

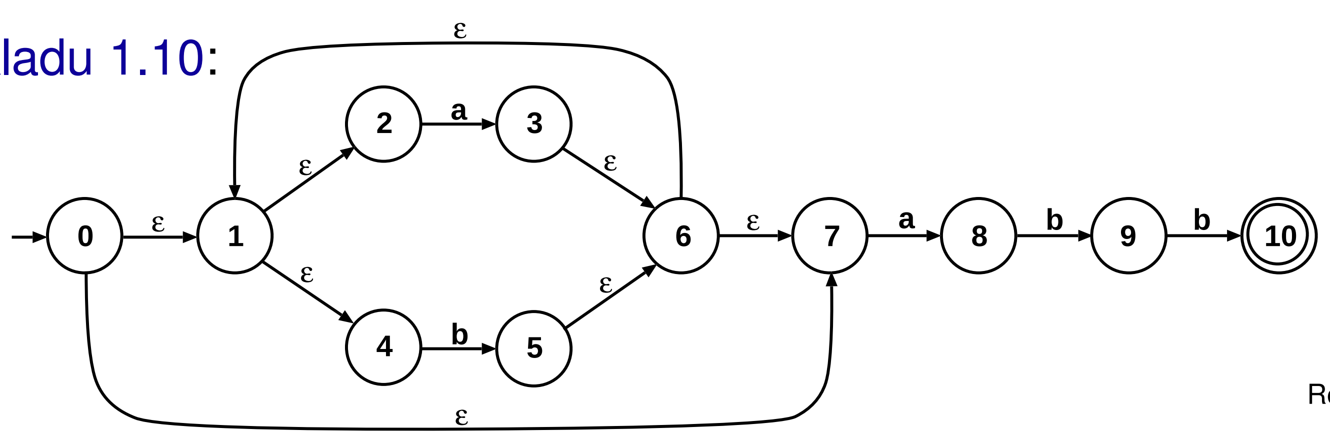
Algoritmus 1.2 Převod RKA na DKA
Vstup: RKA $M = (Q, \Sigma, \delta, q_0, F)$.
Výstup: DKA $M' = (Q', \Sigma, \delta', q_0', F')$, $L(M) = L(M')$.
Metoda:

1. $Q' := 2^Q$.
2. $q_0' := \varepsilon$-uzávěr(q_0).
3. $\delta' : Q' \times \Sigma \to Q'$ je vypočtena takto:
   - Nechť $\forall T \in Q', a \in \Sigma : \overline{\delta}(T, a) = \bigcup_{q \in T} \delta(q, a)$.
   - Pak pro každé $T \in Q'$, $a \in \Sigma$: $\delta'(T, a) = \varepsilon$-uzávěr$(\overline{\delta}(T, a))$,
4. $F' := \{S \mid S \in Q' \wedge S \cap F \neq \emptyset\}$.

Příklad 1.11 Aplikujeme algoritmus na automat z příkladu 1.10:

1. Počáteční stav, označíme ho A, je $A = \varepsilon$-uzávěr$(0) = \{0, 1, 2, 4, 7\}$.
2. $\delta'(A, a) = \varepsilon$-uzávěr$(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$.
3. $\delta'(A, b) = \varepsilon$-uzávěr$(\{5\}) = \{1, 2, 4, 5, 6, 7\} = C$.
4. $\delta'(B, a) = \varepsilon$-uzávěr$(\{3, 8\}) = B$.
5. $\delta'(B, b) = \varepsilon$-uzávěr$(\{5, 9\} = \{1, 2, 4, 5, 6, 7, 9\} = D$.
6. $\delta'(C, a) = \varepsilon$-uzávěr$(\{3, 8\}) = B$.
7. $\delta'(C, b) = \varepsilon$-uzávěr$(\{5\}) = C$.
8. $\delta'(D, a) = \varepsilon$-uzávěr$(\{3, 8\}) = B$.
9. $\delta'(D, b) = \varepsilon$-uzávěr$(\{5, 10\} = \{1, 2, 4, 5, 6, 7, 10\} = E$.
10. $\delta'(E, a) = \varepsilon$-uzávěr$(\{3, 8\}) = B$.
11. $\delta'(E, b) = \varepsilon$-uzávěr$(\{5\}) = C$.
12. Množina koncových stavů $F = \{E\}$.

Automat z příkladu 1.10:



Převod gramatiky typu 3 na NKA

Věta 1.5 Nechť $\mathcal{L}_M$ je množina (třída) všech jazyků přijímaných konečnými automaty a nechť L je libovolný jazyk typu 3 ($L \in \mathcal{L}_3$). Pak existuje konečný automat M takový, že:

$$L = L(M), \text{ tj. } \mathcal{L}_3 \subseteq \mathcal{L}_M.$$

Důkaz.

1. Ke gramatice $G = (N, \Sigma, P, S)$ sestrojíme NKA $M = (Q, \Sigma, \delta, q_0, F)$ takto:
   (a) $Q = N \cup \{q_F\}$
   (b) $\Sigma = \Sigma$
   (c) $\delta : \delta(A, a)$ obsahuje B, právě když $A \rightarrow aB$ je v P
   (d) $\delta : \delta(A, a)$ obsahuje q_F, právě když $A \rightarrow a$ je v P
   (e) $q_0 = S$
   (f) $F = \{A \mid A \rightarrow \varepsilon \text{ je v } P\} \cup \{q_F\}$

Důkaz pokračuje dále.

Pokračování důkazu.

2. Matematickou indukcí ukážeme, že $L(G) = L(M)$. Indukční hypotézu formulujeme obecněji ve tvaru:

$$\forall A \in N : A \overset{i+1}{\underset{G}{\Rightarrow}} w \quad \Longleftrightarrow \quad (A, w) \overset{i}{\underset{M}{\vdash}} (C, \varepsilon) \text{ pro } C \in F, w \in \Sigma^*$$

Pro $i = 0$ dostáváme

$$A \Rightarrow \varepsilon \quad \Longleftrightarrow \quad (A, \varepsilon) \overset{0}{\vdash} (A, \varepsilon) \text{ pro } A \in F$$

a tvrzení tedy platí.
Pro $i = 1$ dostáváme

$$A \Rightarrow a \quad \Longleftrightarrow \quad (A, a) \overset{1}{\vdash} (q_F, \varepsilon) \text{ a } q_F \in F$$

a tvrzení tedy platí.

Nyní předpokládejme, že dokazovaná hypotéza platí pro $i > 0$ a položme $w = ax$, kde $a \in \Sigma$ a $|x| = i - 1$.

Důkaz pokračuje dále.

Pokračování důkazu.

3. pokračování.

   Dále předpokládejme $A \Rightarrow aB \overset{i}{\Rightarrow} ax$,

   z indukční hypotézy plyne $B \overset{i}{\Rightarrow} x \iff (B, x) \overset{i-1}{\vdash} (C, \varepsilon), C \in F$

   a z definice funkce δ: $A \Rightarrow aB \iff B \in \delta(A, a)$

   Dohromady tedy

$$A \Rightarrow aB \overset{i}{\Rightarrow} ax = w' \iff (A, ax) \vdash (B, x) \overset{i-1}{\vdash} (C, \varepsilon), C \in F$$

   tedy $$A \overset{i+1}{\Rightarrow} w' \iff (A, w') \overset{i}{\vdash} (C, \varepsilon), C \in F$$

   tj. tvrzení platí i pro $i + 1$.

   Pro případ $A = S$ je dokázaná hypotéza tvrzením věty, tj.

$$\forall w' \in \Sigma^* : S \overset{*}{\Rightarrow} w' \iff (S, w') \overset{*}{\vdash} (C, \varepsilon), C \in F, \text{ tj. } L(G) = L(M)$$

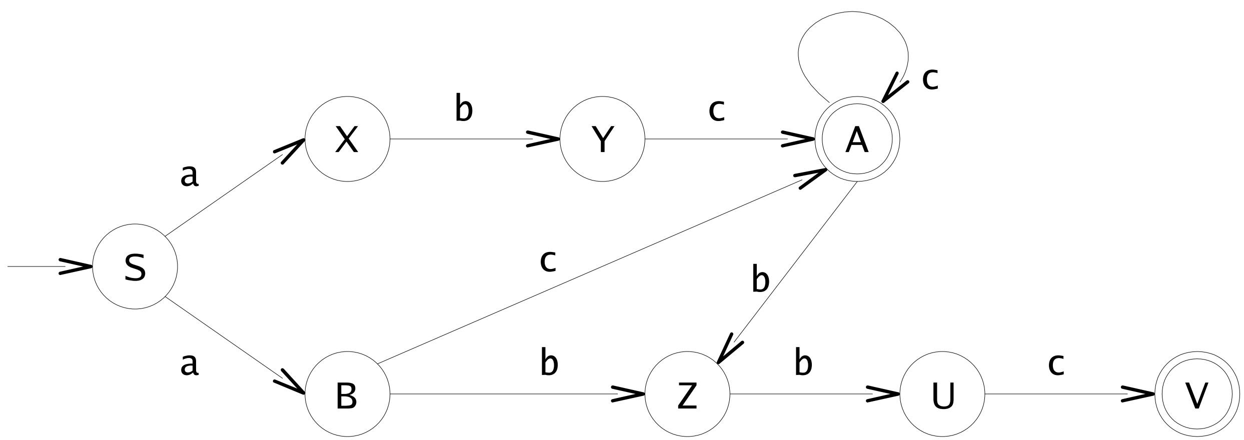
□

Příklad 1.12

Uvažme gramatiku $G = (\{S, A, B, U, V, X, Y, Z\},\ \{a, b, c\},\ P,\ S)$ s pravidly P:

$$\begin{array}{ll} S \to aX \mid aB & \quad A \to cA \mid bZ \mid \varepsilon \\ X \to bY & \quad B \to cA \mid bZ \\ Y \to cA & \quad U \to cV \\ Z \to bU & \quad V \to \varepsilon \end{array}$$

Takové gramatice odpovídá konečný automat:



Převod NKA na gramatiku typu 3

Věta 1.6 Nechť M je NKA. Pak existuje gramatika G typu 3 taková, že:

$$L(M) = L(G), \text{ tj. } \mathcal{L}_M \subseteq \mathcal{L}_3.$$

Důkaz. Nechť $M = (Q, \Sigma, \delta, q_0, F)$. Předpokládejme, že M je NKA. Nechť $G = (Q, \Sigma, P, q_0)$ je gramatika, jejíž pravidla jsou definována takto:

1. pokud $\delta(q, a)$ obsahuje r, pak P obsahuje pravidlo $q \to ar$
2. je-li $p \in F$, pak P obsahuje pravidlo $p \to \varepsilon$
3. jiná pravidla množina P neobsahuje.

G je zřejmě typu 3 a indukcí lze dokázat, že platí $L(G) = L(M)$.

$\square$

Příklad 1.13 Uvažujme KA $M_3 = (\{A, B, C, D\},\ \{a, b, c\},\ \delta,\ A,\ \{C, D\})$, kde

$$\begin{array}{lll} \delta : & \delta(A, a) = B & \delta(C, c) = D \\ & \delta(B, b) = A & \delta(D, a) = A \\ & \delta(B, c) = B & \delta(D, b) = D \\ & \delta(B, a) = C & \end{array}$$

Gramatika G typu 3, která generuje jazyk $L(M_3)$, má tvar:

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

$$\begin{array}{lll} P : & A \to aB & C \to cD \mid \varepsilon \\ & B \to bA \mid cB \mid aC & D \to aA \mid bD \mid \varepsilon \end{array}$$

Regulární množiny a výrazy

Regulární množiny

Definice 1.14 Nechť Σ je konečná abeceda. Regulární množinu nad Σ definujeme rekurzívně takto:

1. $\emptyset$ (tj. prázdná množina) je regulární množina nad Σ,
2. $\{\varepsilon\}$ je regulární množina nad Σ,
3. $\{a\}$ je regulární množina nad Σ pro všechny $a \in \Sigma$,
4. jsou-li P a Q regulární množiny nad Σ, pak také
   (a) $P \cup Q$,
   (b) $P.Q$,
   (c) P^*

   jsou regulární množiny nad Σ.
5. Žádné jiné množiny, než ty, které lze získat pomocí výše uvedených pravidel, nejsou regulárními množinami.

Příklad 1.14 $L = (\{a\} \cup \{d\}).(\{b\}^*).\{c\}$ je regulární množina nad $\Sigma = \{a, b, c, d\}$.

Regulární výrazy

Definice 1.15 Regulární výrazy nad Σ a regulární množiny, které označují, jsou rekurzívně definovány takto:

1. $\emptyset$ je regulární výraz označující regulární množinu $\emptyset$,
2. ε je regulární výraz označující regulární množinu $\{\varepsilon\}$,
3. a je regulární výraz označující regulární množinu $\{a\}$ pro všechny $a \in \Sigma$,
4. jsou-li p, q regulární výrazy označující regulární množiny P a Q, pak
   (a) $(p + q)$ je regulární výraz označující regulární množinu $P \cup Q$,
   (b) (pq) je regulární výraz označující regulární množinu $P.Q$,
   (c) (p^*) je regulární výraz označující regulární množinu P^*.
5. Žádné jiné regulární výrazy nad Σ neexistují.

❖ Konvence:

1. Regulární výraz p^+ značí regulární výraz pp^*.
2. Abychom minimalizovali počet používaných závorek, stanovujeme priority operátorů:
   1. $*$, $+$ (iterace – nejvyšší priorita),
   2. . (konkatenace),
   3. $+$ (alternativa).

Příklad 1.15

1. 01 odpovídá $\{01\}$.
2. 0^* odpovídá $\{0\}^*$.
3. $(0+1)^*$ odpovídá $\{0,1\}^*$.
4. $(0+1)^*011$ značí množinu řetězců nad $\{0,1\}$ končících 011.
5. $(a+b)(a+b+0+1)^*(0+1)$ značí množinu řetězců nad $\{a,b,0,1\}$, které začínají symbolem a nebo b a končí symbolem 0 nebo 1.

Kleeneho algebra

Definice 1.16 Kleeneho algebra sestává z neprázdné množiny se dvěma význačnými konstantami 0 a 1, dvěma binárními operacemi $+$ a . a unární operací $*$, které splňují následující axiomy:

$a + (b + c) = (a + b) + c$	asociativita $+$	[A.1]
$a + b = b + a$	komutativita $+$	[A.2]
$a + a = a$	idempotence $+$	[A.3]
$a + 0 = a$	0 je identitou pro $+$	[A.4]
$a(bc) = (ab)c$	asociativita .	[A.5]
$a1 = 1a = a$	1 je identitou pro .	[A.6]
$a0 = 0a = 0$	0 je anihilátorem pro .	[A.7]
$a(b + c) = ab + ac$	distributivita zleva	[A.8]
$(a + b)c = ac + bc$	distributivita zprava	[A.9]
$1 + aa^* = a^*$		[A.10]
$1 + a^*a = a^*$		[A.11]
$b + ac \leq c \Rightarrow a^*b \leq c$		[A.12]
$b + ca \leq c \Rightarrow ba^* \leq c$		[A.13]

V A.12 a A13 reprezentuje $\leq$ uspořádání definované takto: $a \leq b \overset{def}{\iff} a + b = b$.

- Příklady Kleeneho algeber:
  - Třída 2^{Σ^*} všech podmnožin Σ^* s konstantami $\emptyset$ a $\{\varepsilon\}$ a operacemi $\cup$, . a $*$.
  - Třída všech regulárních podmnožin Σ^* s konstantami $\emptyset$ a $\{\varepsilon\}$ a operacemi $\cup$, . a $*$.
  - Třída všech binárních relací nad množinou X s konstantami v podobě prázdné relace a identity a $\cup$, kompozicí (součinem) binárních relací a reflexivním tranzitivním uzávěrem binární relace jako operacemi.
  - Matice nad Kleeneho algebrami.

- Vlastnosti Kleeneho algeber umožňují snadno řešit systémy lineárních rovnic nad těmito algebrami.

- Kleeneho algebra nad regulárními výrazy je klíčová pro úpravy a zjednodušování RV.

Rovnice nad regulárními výrazy

Definice 1.17 Rovnice, jejímiž složkami jsou koeficienty a neznámé, které reprezentují (dané a hledané) regulární výrazy, nazýváme rovnicemi nad regulárními výrazy.

Příklad 1.16 Uvažujme rovnici nad regulárními výrazy nad abecedou $\{a, b\}$

$$X = aX + b$$

Jejím řešením je regulární výraz $X = a^*b$.

Důkaz.

- $LS = a^*b$
- $PS = a(a^*b) + b = a^+b + b = (a^+ + \varepsilon)b = a^*b$.

$\square$

❖ Ne vždy existuje jediné řešení rovnice nad reg. výrazy.

Věta 1.7 Nejmenším pevným bodem („nejmenším řešením“) rovnice $X = pX + q$ je:

$$X = p^*q$$

Důkaz.

- $PS = p^*q$
- $LS = pp^*q + q = (pp^* + \varepsilon)q = p^*q$
- Minimalita plyne přímo z A.12.

$\square$

Soustavy rovnic nad regulárními výrazy

Definice 1.18 Soustava rovnic nad reg. výrazy je ve standardním tvaru vzhledem k neznámým $\Delta = \{X_1, X_2, ..., X_n\}$, má-li soustava tvar

$$\bigwedge_{i \in \{1,...,n\}} X_i = \alpha_{i0} + \alpha_{i1}X_1 + \alpha_{i2}X_2 + ... + \alpha_{in}X_n$$

kde α_{ij} jsou reg. výrazy nad nějakou abecedou Σ, $\Sigma \cap \Delta = \emptyset$.

Věta 1.8 Je-li soustava rovnic nad reg. výrazy ve std. tvaru, pak existuje její minimální pevný bod a algoritmus jeho nalezení.

Důkaz. Vyjadřujeme hodnotu jednotlivých proměnných pomocí řešení rovnice $X = pX + q$ jako regulární výraz s proměnnými, jejichž počet se postupně snižuje: Z rovnice pro X_n vyjádříme např. X_n jako regulární výraz nad Σ a $X_1, ..., X_{n-1}$. Dosadíme za X_n do rovnice pro X_{n-1} a postup opakujeme. Jsou přitom možné (ale ne nutné) různé optimalizace tohoto pořadí. □

Příklad 1.17 Řešme soustavu rovnic nad reg. výrazy:

(1) $X_1 \;=\; (01^* + 1)X_1 + X_2$

(2) $X_2 \;=\; 11 + 1X_1 + 00X_3$

(3) $X_3 \;=\; \varepsilon + X_1 + X_2$

- Výraz pro X_3 dosadíme z (3) do (2). Dostaneme soustavu:

  (4) $X_1 \;=\; (01^* + 1)X_1 + X_2$

  (5) $X_2 \;=\; 11 + 1X_1 + 00(\varepsilon + X_1 + X_2) = 00 + 11 + (1 + 00)X_1 + 00X_2$

- Ze (4) vyjádříme X_1 s využitím řešení rovnice $X = pX + q$:

  (6) $X_1 \;=\; (01^* + 1)^* X_2 = (0 + 1)^* X_2$

- Dosazením do (5):

  (7) $X_2 \;=\; 00 + 11 + (1 + 00)(0 + 1)^* X_2 + 00X_2 = 00 + 11 + (1 + 00)(0 + 1)^* X_2$

- Vypočtením X_2 jako řešení rovnice $X = pX + q$ dostaneme:

  (8) $X_2 \;=\; ((1 + 00)(0 + 1)^*)^*(00 + 11)$

- Dosazením do (6) dostaneme:

  (9) $X_1 = \; (0 + 1)^*((1 + 00)(0 + 1)^*)^*(00 + 11) = (0 + 1)^*(00 + 11)$

- Dosazením do (3) dostaneme:

  (10) $X_3 = \; \varepsilon + (0 + 1)^*(00 + 11) + ((1 + 00)(0 + 1)^*)^*(00 + 11) =$

  $= \; \varepsilon + ((0 + 1)^* + ((1 + 00)(0 + 1)^*)^*)(00 + 11) =$

  $= \; \varepsilon + (0 + 1)^*(00 + 11)$

Regulární množiny a jazyky typu 3

Věta 1.9 Jazyk L je regulární množinou právě tehdy, je-li L jazykem typu 3. Označíme-li $\mathcal{L}_R$ třídu všech regulárních množin, pak:

$$\mathcal{L}_R = \mathcal{L}_3$$

Důkaz. I. $\mathcal{L}_R \subseteq \mathcal{L}_3$, tj. každou regulární množinu lze generovat gramatikou typu 3.

	regulární množina	*gramatika typu 3*
(1)	$\emptyset$	$G_\emptyset = (\{S\}, \Sigma, \emptyset, S)$
(2)	$\{\varepsilon\}$	$G_\varepsilon = (\{S\}, \Sigma, \{S \to \varepsilon\}, S)$
(3)	$\{a\}$ pro každé $a \in \Sigma$	$G_a = (\{S\}, \Sigma, \{S \to a\}, S)$

Nyní ukážeme, že sjednocení, konkatenaci a iteraci reg. množin lze generovat rovněž gramatikou typu 3. Nechť tedy

- $L_1 = L(G_1)$, kde $G_1 = (N_1, \Sigma_1, P_1, S_1)$,
- $L_2 = L(G_2)$, kde $G_2 = (N_2, \Sigma_2, P_2, S_2)$

a G_1, G_2 jsou gramatiky typu 3, $N_1 \cap N_2 = \emptyset$ (nonterminály je vždy možno takto odlišit).

Důkaz pokračuje dále.

Pokračování důkazu.

(4) $L_1 \cup L_2$

$G_4 = (N_4, \Sigma_1 \cup \Sigma_2, P_4, S_4)$, kde

- $N_4 = N_1 \cup N_2 \cup \{S_4\}$, $S_4 \notin N_1 \cup N_2$,
- $P_4 = \{S_4 \to \alpha \mid S_1 \to \alpha \in P_1 \vee S_2 \to \alpha \in P_2\} \cup P_1 \cup P_2$

(5) $L_1 . L_2$

$G_5 = (N_1 \cup N_2, \Sigma_1 \cup \Sigma_2, P_5, S_1)$ a P_5 je nejmenší množina splňující:

- je-li $(A \to xB) \in P_1$, pak $(A \to xB) \in P_5$,
- je-li $(A \to x) \in P_1$, pak $(A \to xS_2) \in P_5$,
- je-li $(A \to \varepsilon) \in P_1$, pak $(A \to \alpha) \in P_5$ pro všechna pravidla $(S_2 \to \alpha) \in P_2$,
- $\forall (A \to \alpha) \in P_2 : (A \to \alpha) \in P_5$.

(6) L_1^*

$G_6 = (N_1 \cup \{S_6\}, \Sigma_1, P_6, S_6)$, $S_6 \notin N_1$ a P_6 je nejmenší množina splňující:

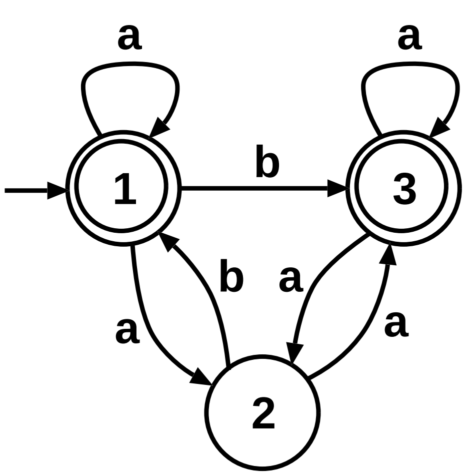
- je-li $(A \to xB) \in P_1$, pak $(A \to xB) \in P_6$,
- je-li $(A \to x) \in P_1$, pak $(A \to xS_6) \in P_6$,
- je-li $(A \to \varepsilon) \in P_1$, pak $(A \to \alpha) \in P_6$ pro všechna pravidla $(S_1 \to \alpha) \in P_1$,
- je-li $(S_1 \to xB) \in P_1$, pak $(S_6 \to xB) \in P_6$
- je-li $(S_1 \to x) \in P_1$, pak $(S_6 \to xS_6) \in P_6$
- $(S_6 \to \varepsilon) \in P_6$.

Pokračování důkazu. II. $\mathcal{L}_3 \subseteq \mathcal{L}_R$, tj. každý jazyk generovaný gramatikou typu 3 je regulární množinou.

- Nechť $L \in \mathcal{L}_3$ je libovolný jazyk typu 3. Již víme, že ho můžeme popsat KA $M = (Q, \Sigma, \delta, q_0, F)$. Nechť $Q = \{q_0, q_1, ..., q_n\}$.
- Vytvoříme soustavu rovnic na reg. výrazy s proměnnými $X_0, X_1, ..., X_n$ ve standardním tvaru. Rovnice pro X_i popisuje množinu řetězců přijímaných ze stavu Q_i.
- Řešením této soustavy získáme reg. výraz pro proměnnou X_0, který reprezentuje jazyk L.

$\square$

Příklad 1.18



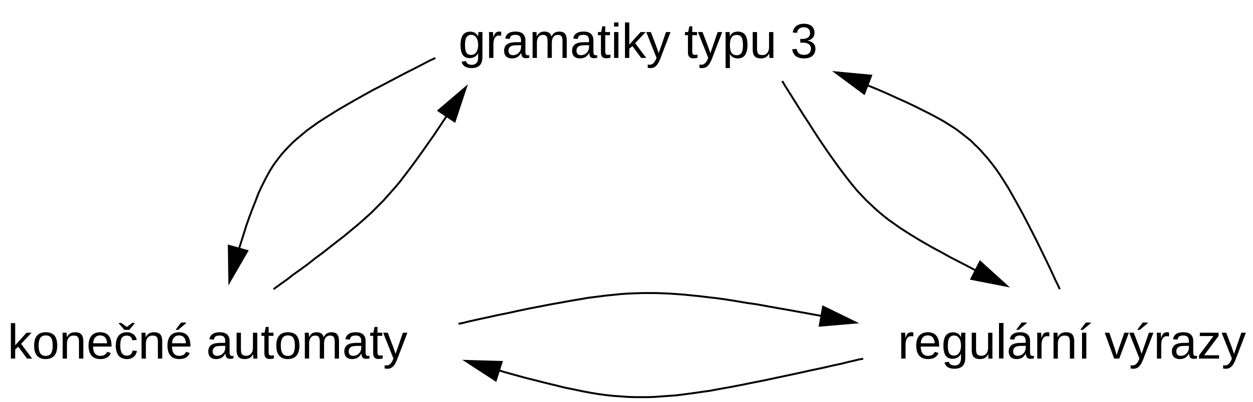
$$
\begin{array}{llllll}
X_1 & = & \varepsilon + & aX_1 + & aX_2 + & bX_3 \\
X_2 & = & & bX_1 + & & aX_3 \\
X_3 & = & \varepsilon + & & aX_2 + & aX_3
\end{array}
$$

Jazyk L popisuje reg. výraz, který je řešením této soustavy pro proměnnou X_1.

Vztahy regulárních gramatik, KA a RV

- Můžeme tedy shrnout, že
  - gramatiky typu 3
  - (rozšířené/nedeterministické/deterministické) konečné automaty a
  - regulární výrazy

mají ekvivalentní vyjadřovací sílu.

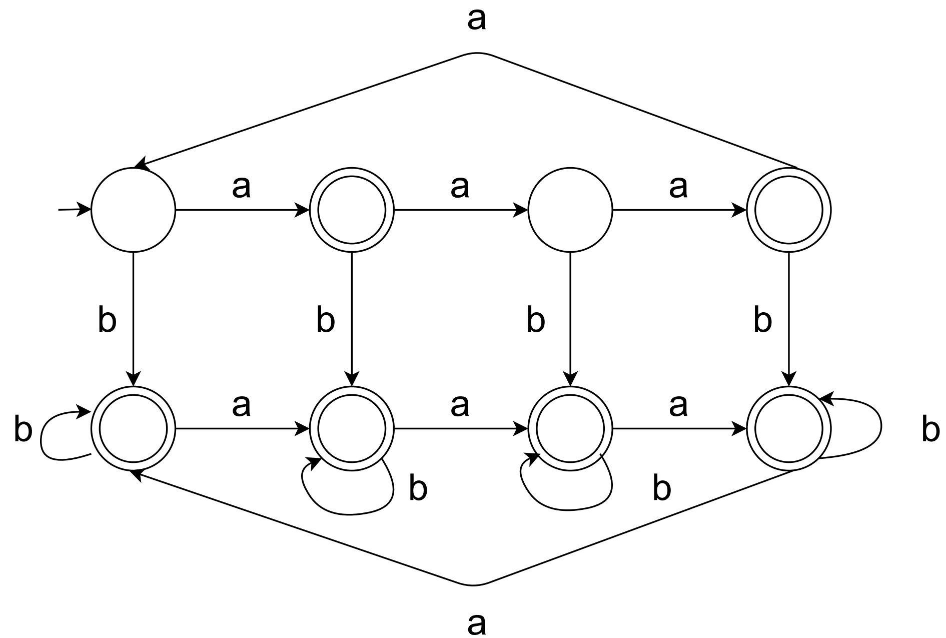


- *Alternativní algoritmy pro převod viz opora*

Minimalizace Konečných Automatů

Motivační příklad

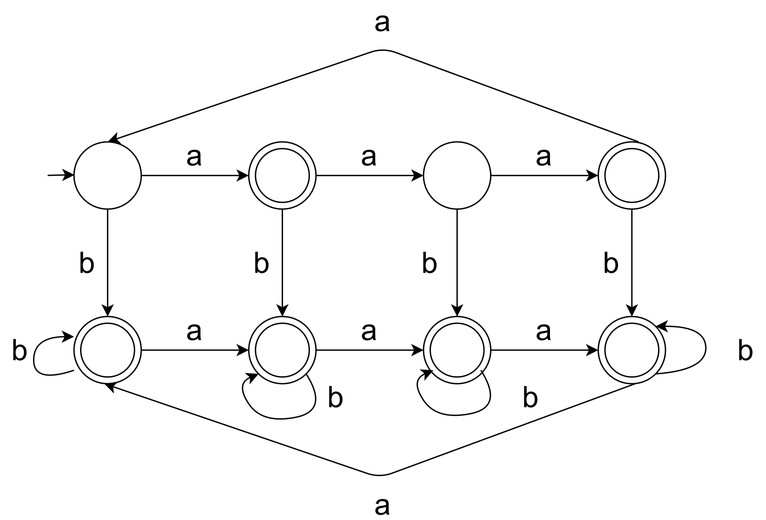
- Uvažme jazyk $L = \{w \in \{a,b\}^* \mid \#_a(w) \texttt{ mod } 2 = 1 \vee \#_b(w) > 0\}$



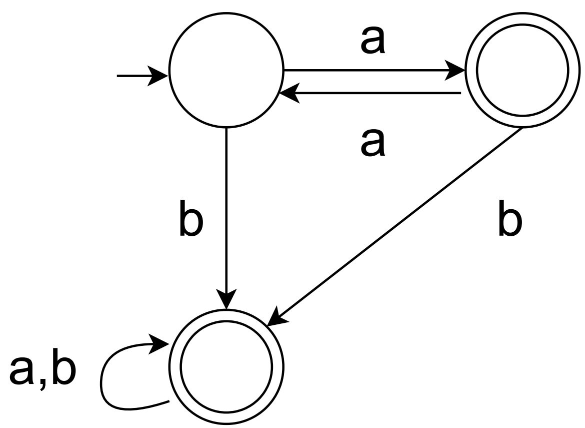
- Existuje menší automat akceptující jazyk L?

Motivační příklad

❖ Uvažme jazyk $L = \{w \in \{a, b\}^* \mid \#_a(w) \texttt{ mod } 2 = 1 \vee \#_b(w) > 0\}$



❖ Existuje menší automat akceptující jazyk L?



Eliminace nedosažitelných stavů

Definice 2.1 Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Stav $q \in Q$ nazveme dosažitelný, pokud existuje $w \in \Sigma^*$ takové, že $(q_0, w) \overset{*}{\underset{M}{\vdash}} (q, \varepsilon)$. Stav je nedosažitelný, pokud není dosažitelný.

Algoritmus 2.1 Eliminace nedosažitelných stavů
Vstup: DKA $M = (Q, \Sigma, \delta, q_0, F)$.
Výstup: DKA M' bez nedosažitelných stavů, $L(M) = L(M')$.
Metoda:

1. i := 0
2. S_i := $\{q_0\}$
3. `repeat`
4. $\quad S_{i+1}$:= $S_i \cup \{q \mid \exists p \in S_i\ \exists a \in \Sigma : \delta(p, a) = q\}$
5. $\quad i$:= i + 1
6. `until` $S_i = S_{i-1}$
7. M' := $(S_i, \Sigma, \delta_{|S_i}, q_0, F \cap S_i)$

Jazykově nerozlišitelné stavy

Definice 2.2

- Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je úplně definovaný DKA. Říkáme, že řetězec $w \in \Sigma^*$ rozlišuje q_1, q_2, jestliže $(q_1, w) \overset{*}{\underset{M}{\vdash}} (q_3, \varepsilon) \wedge (q_2, w) \overset{*}{\underset{M}{\vdash}} (q_4, \varepsilon)$ pro nějaké q_3, q_4 a *právě jeden* ze stavů q_3, q_4 je v F.
- Říkáme, že stavy $q_1, q_2 \in Q$ jsou k-nerozlišitelné a píšeme $q_1 \overset{k}{\equiv} q_2$, právě když neexistuje $w \in \Sigma^*$, $|w| \leq k$, který rozlišuje q_1 a q_2.
- Stavy q_1, q_2 jsou nerozlišitelné, značíme $q_1 \equiv q_2$, jsou-li pro každé $k \geq 0$ k-nerozlišitelné.

❖ Poznámka: Dá se snadno dokázat, že $\equiv$ je relací ekvivalence na Q, tj. relací, která je reflexivní, symetrickou a tranzitivní.

Definice 2.3 Úplně definovaný DKA M nazýváme redukovaný, jestliže žádný stav z Q není nedostupný a žádné dva stavy nerozlišitelné.

Věta 2.1 Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je úplně definovaný DKA a $|Q| = n$, $n \geq 2$. Platí $\forall q_1, q_2 \in Q : q_1 \equiv q_2 \Leftrightarrow q_1 \overset{n-2}{\equiv} q_2$.

Důkaz. „⇒“ triviální, ukážeme „⇐“:

1. Jestliže $|F| = 0$ nebo $|F| = n$, pak platí $q_1 \overset{n-2}{\equiv} q_2 \;\Rightarrow\; q_1 \equiv q_2$.
2. Nechť $|F| > 0 \wedge |F| < n$. Ukážeme, že platí $\equiv\, = \overset{n-2}{\equiv} \subseteq \overset{n-3}{\equiv} \subseteq \ldots \subseteq \overset{1}{\equiv} \subseteq \overset{0}{\equiv}$:
   - Zřejmě platí:
     (a) $\forall q_1, q_2 \in Q : q_1 \overset{0}{\equiv} q_2 \Leftrightarrow (q_1 \in F \wedge q_2 \in F) \vee (q_1 \notin F \wedge q_2 \notin F)$, tj. $q_1 \overset{0}{\equiv} q_2 \Leftrightarrow (q_1 \in F \Leftrightarrow q_2 \in F)$.
     (b) $\forall q_1, q_2 \in Q\ \forall k \geq 1 : q_1 \overset{k}{\equiv} q_2 \Leftrightarrow (q_1 \overset{k-1}{\equiv} q_2 \wedge \forall a \in \Sigma : \delta(q_1, a) \overset{k-1}{\equiv} \delta(q_2, a))$.
   - Relace $\overset{0}{\equiv}$ je ekvivalencí určující rozklad $\{F, Q \setminus F\}$.
   - Je-li $\overset{k+1}{\equiv} \neq \overset{k}{\equiv}$, pak $\overset{k+1}{\equiv}$ je vlastním zjemněním $\overset{k}{\equiv}$, tj. obsahuje alespoň o jednu třídu více než rozklad $\overset{k}{\equiv}$.
   - Jestliže pro nějaké k platí $\overset{k+1}{\equiv} = \overset{k}{\equiv}$, pak také $\overset{k+1}{\equiv} = \overset{k+2}{\equiv} = \overset{k+3}{\equiv} = \ldots$ podle (b) a tedy $\overset{k}{\equiv}$ je hledaná ekvivalence.
   - Protože F nebo $Q \setminus F$ obsahuje nejvýše $n-1$ prvků, získáme relaci $\equiv$ po nejvýše $n-2$ zjemněních $\overset{0}{\equiv}$.

□

Převod na redukovaný DKA

Algoritmus 2.2 Převod na redukovaný DKA

Vstup: Úplně definovaný DKA $M = (Q, \Sigma, \delta, q_0, F)$.

Výstup: Redukovaný DKA $M' = (Q', \Sigma, \delta', q_0', F')$, $L(M) = L(M')$.

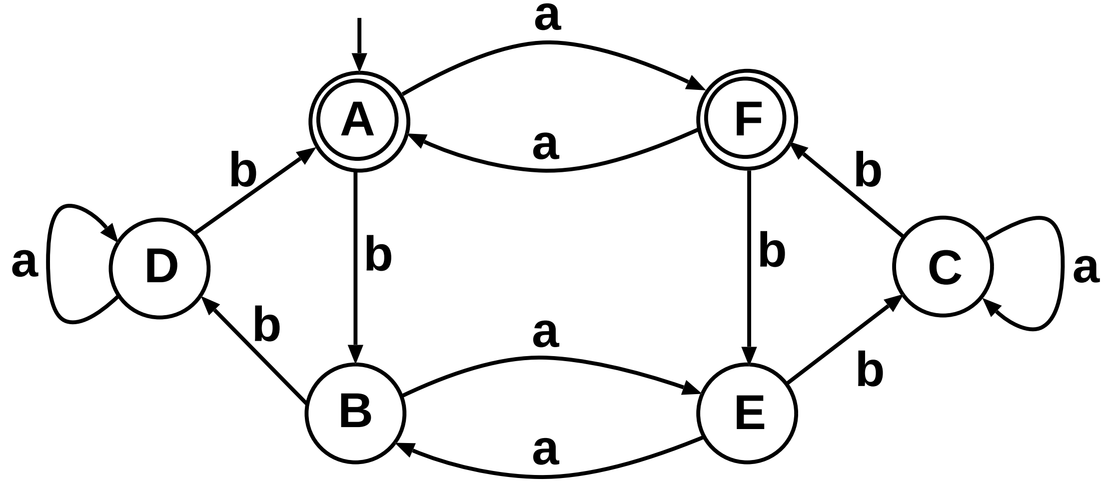
Metoda:

1. Odstraň nedostupné stavy s využitím alg. 2.1.
2. i := 0
3. $\stackrel{0}{\equiv}$:= $\{(p, q) \mid p \in F \iff q \in F\}$
4. `repeat`
5. $\quad \stackrel{i+1}{\equiv}$:= $\{(p, q) \mid p \stackrel{i}{\equiv} q \wedge \forall a \in \Sigma : \delta(p, a) \stackrel{i}{\equiv} \delta(q, a)\}$
6. $\quad i$:= i + 1
7. `until` $\stackrel{i}{\equiv} = \stackrel{i-1}{\equiv}$
8. Q' := $Q/ \stackrel{i}{\equiv}$
9. $\forall p, q \in Q \; \forall a \in \Sigma : \delta'([p], a) = [q] \Leftrightarrow \delta(p, a) = q$
10. $q_0' = [q_0]$
11. $F' = \{[q] \mid q \in F\}$

❖ *Poznámka:* Výraz $[x]$ značí ekvivalenční třídu určenou prvkem x.

Příklad minimalizace DKA

Příklad 2.1 Převeďte níže uvedený DKA (zadaný diagram přechodů) na odpovídající redukovaný DKA.



1. Neobsahuje nedostupné stavy.

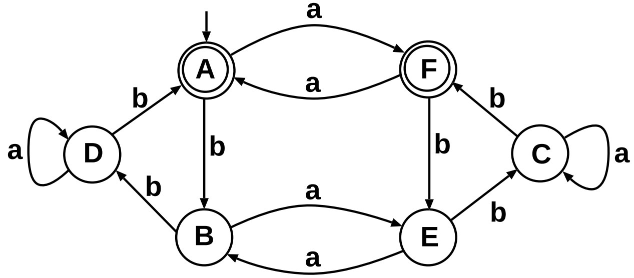
3. $\stackrel{0}{\equiv} = \{\{A, F\}, \{B, C, D, E\}\}$

5.1. $\stackrel{1}{\equiv} = \{\{A, F\}, \{B, E\}, \{C, D\}\}$

$\stackrel{0}{\equiv}$	δ	a	b
I:	A	F_I	B_{II}
	F	A_I	E_{II}
II:	B	E_{II}	D_{II}
	C	C_{II}	F_I
	D	D_{II}	A_I
	E	B_{II}	C_{II}

Pokračuje na druhé straně...

Pro zopakování automat z předchozího slajdu, v jehož minimalizaci níže pokračujeme:

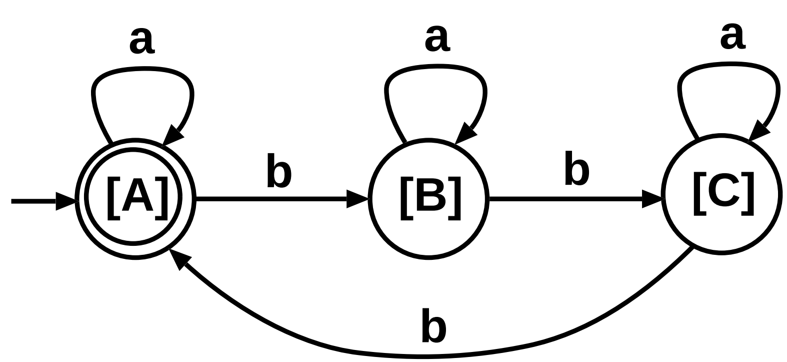


$\overset{1}{\equiv}$	δ	a	b
I:	A	F_I	B_{II}
	F	A_I	E_{II}
II:	B	E_{II}	D_{III}
	E	B_{II}	C_{III}
III:	C	C_{III}	F_I
	D	D_{III}	A_I

5.2. $\overset{2}{\equiv} = \{\{A,F\},\{B,E\},\{C,D\}\} = \overset{1}{\equiv} = \equiv$

8. $Q' = \{[A],[B],[C]\}$, kde $[A] = \{A,F\}$, $[B] = \{B,E\}$, $[C] = \{C,D\}$

Výsledný automat:



Strukturální vlastnosti regulárních jazyků

Konečné jazyky

Věta 2.2 Každý konečný jazyk je regulární.

Důkaz. Nechť $L = \{w_1, w_2, \ldots, w_n\},\ w_i \in \Sigma$.

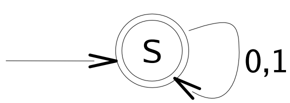
Pak $L = L(G)$, kde $G = (\{S\}, \Sigma, \{S \to w_1, S \to w_2, \ldots, S \to w_n\}, S)$. G je zřejmě gramatika typu 3.

$\square$

Opak věty 3.1 zjevně neplatí:

Příklad 2.2 Sestrojte gramatiku typu 3 generující jazyk $\{0, 1\}^*$.

Řešení:



$\Rightarrow \quad G = (\{S\},\ \{0, 1\},\ \{S \to \varepsilon, S \to 0S, S \to 1S\},\ S)$

Pumping lemma

Věta 2.3 Nechť L je regulární jazyk. Pak existuje celočíselná konstanta $p > 0$ taková, že platí:

$$
\begin{aligned}
w \in L \ \wedge \ |w| \geq p \quad \Rightarrow \quad & w = xyz \ \wedge \\
& y \neq \varepsilon \ \wedge |xy| \leq p \ \wedge \\
& xy^i z \in L \text{ pro } i \geq 0.
\end{aligned}
$$

❖ Ekvivalentní formulace Pumping lemmatu (použití explicitní alternace kvantifikátorů) :

$$
\begin{aligned}
L \in \mathcal{L}_3 \Rightarrow \quad & \exists p > 0 : \\
& \forall w \in \Sigma^* : w \in L \wedge \ |w| \geq p \Rightarrow \\
& (\exists x, y, z \in \Sigma^* : w = xyz \wedge \ y \neq \epsilon \ \wedge \ |xy| \leq p \wedge \forall i \geq 0 : xy^i z \in L)
\end{aligned}
$$

❖ Poznámka: Neformálně řečeno Pumping lemma tvrdí, že v každé dostatečně dlouhé větě každého regulárního jazyka jsme schopni poblíž jejího začátku najít poměrně krátkou sekvenci, kterou je možné vypustit, resp. zopakovat libovolný počet krát, přičemž zůstáváme stále v rámci daného jazyka.

Důkaz. Pumping lemmatu
Nechť $L = L(M)$, $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat, kde $|Q| = n > 0$. Položme $p = n$. Je-li $w \in L$ a $|w| \geq n$, pak M přijme větu w „průchodem“ alespoň $n + 1$ konfiguracemi a tudíž alespoň dvě z nich obsahují stejný stav, tedy:

$$(q_0, w) = (q_0, xyz) \overset{*}{\vdash} (r, yz) \overset{k}{\vdash} (r, z) \overset{*}{\vdash} (q_F, \varepsilon),\ q_F \in F$$

pro nějaký stav $r \in Q$ a k takové, že $0 < k \leq n$. Dále je zřejmé, že k „zopakování“ stavu r dojde nejpozději po přečtení prvních n znaků vstupního řetězce a tudíž $|xy| \leq p$.

Pak ale existuje posloupnost konfigurací:

$$\begin{array}{ll} (q_0, xy^iz) & \overset{*}{\vdash} (r, y^iz) \\ & \overset{+}{\vdash} (r, y^{i-1}z) \\ & \vdots \\ & \overset{+}{\vdash} (r, z) \\ & \overset{*}{\vdash} (q_F, \varepsilon) \end{array}$$

z které plyne $xy^iz \in L(M)$, a to nejen pro $i > 0$, ale i pro případ $i = 0$:

$$(q_0, xz) \overset{*}{\vdash} (r, z) \overset{*}{\vdash} (q_F, \varepsilon),\ q_F \in F$$

$\square$

Význam Pumping lemmatu

❖ Jak můžeme dokázat, že daný problém je/není řešitelný pomocí uvažovaných výpočetních prostředků (např. jestli pro daný jazyk existuje KA)?

- ukázat existenci řešení je jednoduché: poskytneme řešení (např. KA)
- ukázat neexistenci je principiálně náročnější: nemůžeme vyzkoušet všechny možná řešení (všech KA je nekonečně mnoho)

❖ Pumping lemma nám dovoluje dokazovat neexistenci řešení (tj. neexistenci KA pro daný jazyk)

❖ V rámci TIN si ukážeme i další techniky (diagonalizace, redukce), které lze použít i pro jiné výpočetní třídy

Použití Pumping lemmatu

$$(L \in \mathcal{L}_3 \Rightarrow A) \Leftrightarrow (\neg A \Rightarrow L \notin \mathcal{L}_3)$$ Obměna implikace

$$\begin{aligned}
A \equiv \quad & \exists p > 0 : \\
& \forall w \in \Sigma^* : w \in L \wedge\ |w| \geq p \Rightarrow \\
& (\exists x, y, z \in \Sigma^* : w = xyz \wedge\ y \neq \epsilon\ \wedge\ |xy| \leq p \wedge \forall i \geq 0 : xy^i z \in L)
\end{aligned}$$

$$\begin{aligned}
\neg A \equiv \quad & \forall p > 0 : \\
& \exists w \in \Sigma^* : w \in L \wedge\ |w| \geq p \wedge \\
& (\forall x, y, z \in \Sigma^* : w = xyz \wedge\ y \neq \epsilon\ \wedge\ |xy| \leq p \Rightarrow \exists i \geq 0 : xy^i z \notin L)
\end{aligned}$$

- K důkazu, že jazyk L není regulární stačí dokázat tvrzení $\neg A$.

Příklad 2.3 Dokažte, že jazyk $L = \{0^n 1^n \mid n \geq 1\}$ není regulární.

Důkaz:

- Pro libovolné $p > 0$ zvolíme slovo $w = 0^p 1^p$ ($w \in L \wedge |w| \geq p$).

- Dále uvažme všechny rozdělení $w = xyz$, kde $y \neq \varepsilon \ \wedge |xy| \leq p$. Je zřejmé, že $y \in \{0\}^+$.

$$0\,\underbrace{0\,0\,\ldots}_{y}\,.\,0\,1\,1\,1\ldots 1$$

- Pak ale pro libovolné $y \in \{0\}^+$ (libovolné rozdělení), $\exists i \geq 0$, pro které $xy^iz \notin L$ — nesouhlasí počet 0 a 1 (zde to platí pro všechna $i \neq 1$).

- Ukázali jsme, že pro L platí tvrzení $\neg A$ (viz. předchozí slajd) a tudíž $L \notin \mathcal{L}_3$.

$\square$

Příklad 2.4 Dokažte, že jazyk $L = \{a^q \mid q \text{ je prvočíslo } \}$ není regulární.

Důkaz:

❖ Pro libovolné $p > 0$ zvolíme slovo $w = a^r$, kde r prvočíslo větší než p.

❖ Dále uvažme všechny rozdělení $w = xyz$, kde $y \neq \varepsilon \ \wedge |xy| \leq p$. Je zřejmé, že $y = a^k$, kde $0 < k \leq p$.

❖ Pak ale pro libovolné k (libovolné rozdělení), zvolme $i = r + 1$. Dostáváme, že $|xy^i z| = |xy^{r+1}z| = |xyz| + |y^r| = r + r.k = r.(k+1)$, což však není prvočíslo (pro žádné k), a tedy $xy^{r+1}z \notin L$

❖ Ukázali jsme, že pro L platí tvrzení A a tudíž $L \notin \mathcal{L}_3$.

$\square$

Myhill-Nerodova věta

Motivace

- Myhill-Nerodova věta
  - charakterizuje některé zásadní vztahy mezi konečnými automaty nad abecedou Σ a jistými ekvivalenčními relacemi nad řetězci ze Σ^*,
  - popisuje některé z nutných a postačujících podmínek pro to, aby daný jazyk byl jazykem regulárním (používá se často k důkazu neregularity jazyka),
  - poskytuje formální bázi pro elegantní důkaz existence unikátního (až na isomorfismus) minimálního DKA k danému regulárnímu jazyku.

Pravá kongruence a prefixová ekvivalence

❖ Zopakování: ekvivalence $\sim$ je binární relace, která je *reflexivní, symetrická a tranzitivní*. Index ekvivalence $\sim$ je počet tříd rozkladu $\Sigma^*/\sim$. Je-li těchto tříd nekonečně mnoho, definujeme index jako ∞.

Definice 2.4 Nechť Σ je abeceda a $\sim$ je ekvivalence na Σ^*. Ekvivalence $\sim$ je pravou kongruencí (je zprava invariantní), pokud pro každé $u, v, w \in \Sigma^*$ platí

$$u \sim v \Longrightarrow uw \sim vw$$

Věta 2.4 Ekvivalence $\sim$ na Σ^* je pravá kongruence právě tehdy, když pro každé $u, v \in \Sigma^*$, $a \in \Sigma$ platí $u \sim v \Longrightarrow ua \sim va$.

Důkaz. „$\Rightarrow$“ je triviální, „$\Leftarrow$“ lze snadno ukázat indukcí nad délkou w. $\square$

Definice 2.5 Nechť L je libovolný (ne nutně regulární) jazyk nad abecedou Σ. Na množině Σ^* definujeme relaci $\sim_L$ zvanou prefixová ekvivalence pro L takto:

$$u \sim_L v \overset{def}{\Longleftrightarrow} \forall w \in \Sigma^* : uw \in L \Longleftrightarrow vw \in L$$

Myhill-Nerodova věta

Věta 2.5 Nechť L je jazyk nad Σ. Pak následující tvrzení jsou ekvivalentní:

1. L je jazyk přijímaný deterministickým konečným automatem.
2. L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.
3. Relace $\sim_L$ má konečný index.

Důkaz. Dokážeme následující implikace:

- **1** $\Rightarrow$ **2**
- **2** $\Rightarrow$ **3**
- **3** $\Rightarrow$ **1**

Z definice ekvivalence $(a \Leftrightarrow b \stackrel{def}{\Longleftrightarrow} a \Rightarrow b \ \wedge \ b \Rightarrow a)$ a ze základní tautologie výrokové logiky $(a \Rightarrow b \ \wedge \ b \Rightarrow c) \Rightarrow (a \Rightarrow c)$ plyne tvrzení věty.

$\square$

Důkaz implikace 1 $\Rightarrow$ 2

❖ Je-li L přijímán DKA, pak L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.

❖ Pro DKA $M = (Q, \Sigma, \delta, q_0, F)$ zaveďme zobecněnou přechodovou funkci $\hat{\delta} : Q \times \Sigma^* \to Q$ tak, že $\forall q_1, q_2 \in Q, w \in \Sigma^* : \hat{\delta}(q_1, w) = q_2 \Leftrightarrow (q_1, w) \overset{*}{\underset{M}{\vdash}} (q_2, \varepsilon)$.

Důkaz. Pro daný L přijímaný konečným automatem M zkonstruujeme $\sim$ s potřebnými vlastnostmi:

- Nechť $M = (Q, \Sigma, \delta, q_0, F)$ a δ je totální.
- Zvolíme $\sim$ jako binární relaci na Σ^* takovou, že $u \sim v \iff \hat{\delta}(q_0, u) = \hat{\delta}(q_0, v)$.
- Ukážeme, že $\sim$ má potřebné vlastnosti:
  - $\sim$ je *ekvivalence*: je reflexivní, symetrická a tranzitivní.
  - $\sim$ má *konečný index*: třídy rozkladu odpovídají stavům automatu.
  - $\sim$ je *pravá kongruence*: Nechť $u \sim v$ a $a \in \Sigma$. Pak $\hat{\delta}(q_0, ua) = \delta(\hat{\delta}(q_0, u), a) = \delta(\hat{\delta}(q_0, v), a) = \hat{\delta}(q_0, va)$ a tedy $ua \sim va$.
  - L je *sjednocením některých tříd* $\Sigma^* \backslash \sim$: těch, které odpovídají F.

$\square$

Důkaz implikace 2 $\Rightarrow$ 3

❖ Existuje-li relace $\sim$ splňující podmínku 2, pak $\sim_L$ má konečný index.

Důkaz.

- Pro všechny $u, v \in \Sigma^*$ takové, že $u \sim v$, platí $u \sim_L v$:
  - Nechť $u \sim v$. Ukážeme, že také $u \sim_L v$, tj. $\forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L$.
  - Víme, že $uw \sim vw$ a protože L je sjednocením některých tříd rozkladu $\Sigma^* \backslash \sim$, platí též $uw \in L \Leftrightarrow vw \in L$.
- Víme tedy, že $\sim \subseteq \sim_L$ (tj. $\sim_L$ je největší pravá kongruence s danými vlastnostmi).
- Každá třída $\sim$ je obsažena v nějaké třídě $\sim_L$.
- Index $\sim_L$ nemůže být větší než index $\sim$.
- $\sim$ má konečný index a tedy i $\sim_L$ má konečný index.

$\square$

Důkaz implikace **3** $\Rightarrow$ **1**

❖ Má-li $\sim_L$ konečný index, pak L je přijímán nějakým konečným automatem.

Důkaz. Zkonstruujeme $M = (Q, \Sigma, \delta, q_0, F)$ přijímající L:

- $Q = \Sigma^* \backslash \sim_L$ (stavy jsou třídy rozkladu Σ^* relací $\sim_L$),
- $\forall u \in \Sigma^*, a \in \Sigma : \delta([u], a) = [ua]$,
- $q_0 = [\varepsilon]$,
- $F = \{[x] \mid x \in L\}$.

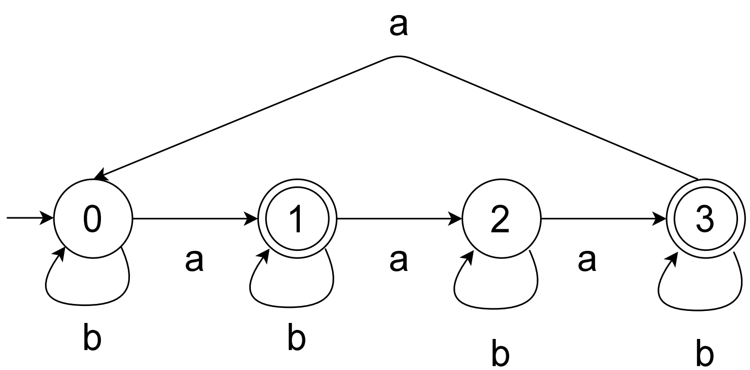
Uvedená konstrukce *je korektní*, tj. $L = L(M)$:

- Indukcí nad délkou slova v ukážeme, že $\forall v \in \Sigma^* : \hat{\delta}([\varepsilon], v) = [v]$.
- $v \in L \iff [v] \in F \iff \hat{\delta}([\varepsilon], v) \in F$.

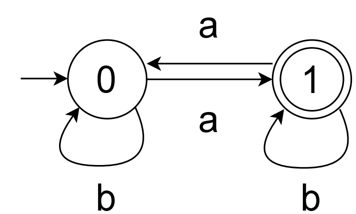
$\square$

Příklad: Interpretace M.-N. věty

- Uvažme jazyk $L = \{w \in \{a,b\}^* \mid \#_a(w) \mod 2 = 1\}$.



DKA A_1



DKA A_2

Pravá kongruence $\sim_1$ odpovídající DKA A_1:

$$u \sim_1 v \iff \#_a(u) \equiv \#_a(v) (\bmod\ 4)$$

$$\{a,b\}^* \backslash \sim_1 = \{[0]_4, [1]_4, [2]_4, [3]_4\}$$

kde $[i]_4 = \{w \in \{a,b\}^* \mid \#_a(w) \mod 4 = i\}$

$$L = [1]_4 \cup [3]_4$$

$\sim_2$ odpovídající DKA A_2:

$$u \sim_2 v \iff \#_a(u) \equiv \#_a(v) (\bmod\ 2)$$

$$\{a,b\}^* \backslash \sim_2 = \{[0]_2, [1]_2\}$$

$$[i]_2 = \{w \in \{a,b\}^* \mid \#_a(w) \mod 2 = i\}$$

$$L = [1]_2$$

$\sim_1 \subseteq \sim_2 = \sim_L$ (A_2 je minimální automat pro L)

Důkaz neregularity pomocí M.-N. věty

Příklad 2.5 Dokažte, že jazyk $L = \{a^n b^n \mid n \geq 0\}$ není regulární.

Důkaz.

- Žádné řetězce $\varepsilon, a, a^2, a^3, \ldots$ nejsou $\sim_L$-ekvivalentní, protože $a^i b^i \in L$, ale $a^j b^i \notin L$ pro $i \neq j$.
- $\sim_L$ má tedy nekonečně mnoho tříd (neboli nekonečný index).
- Dle Myhill-Nerodovy věty tudíž nemůže být L přijímán žádným konečným automatem.

$\square$

Důkaz regularity pomocí M.-N. věty

Příklad 2.6 Dokažte, že jazyk $L = \{w \in \{a,b\}^* \mid 2010 \leq \#_a(w) \leq 2020\}$ je regulární.

Důkaz.

- Uvažme relaci pravé kongruence $\sim$ definovanou následovně:

$$u \sim v \Leftrightarrow (\#_a(u) = \#_a(v)) \vee (\#_a(u) > 2020 \wedge \#_a(v) > 2020)$$

  – $\sim$ je *ekvivalence*: je reflexivní, symetrická a tranzitivní.
  – $\sim$ je *pravá kongruence*: Nechť $u \sim v$, pak $ua \sim va$, jelikož $\#_a(ua) = \#_a(va) = \#_a(u) + 1$ nebo $\#_a(ua) > 2020 \wedge \#_a(va) > 2020$
  Rovněž $ub \sim vb$, jelikož $\#_a(ub) = \#_a(u) \wedge \#_a(v) = \#_a(vb)$.

- $\sim$ má konečný index (rozklad Σ^* má 2021 tříd).
- L je sjednocením tříd rozkladu $[x_i]$ pro $2010 \leq i \leq 2020$, kde

$$[x_i] = \{w \in \{a,b\}^* \mid \#_a(w) = i\}$$

- Dle Myhill-Nerodovy věty je L přijímán konečným automatem.

$\square$

M.-N. věta a minimalita DKA

Věta 2.6 (2. varianta Myhill-Nerodovy věty) Počet stavů libovolného minimálního DKA přijímajícího L je roven indexu $\sim_L$. (Takový DKA existuje právě tehdy, když je index $\sim_L$ konečný.)

Důkaz.

- Každý DKA (můžeme uvažovat DKA bez nedosažitelných stavů) určuje jistou pravou kongruenci s konečným indexem a naopak.
- Je-li L regulární, je $\sim_L$ největší pravou kongruencí s konečným indexem takovou, že L je sjednocením některých tříd příslušného rozkladu.
- Konečný automat, který odpovídá $\sim_L$ (viz důkaz 3 $\Rightarrow$ 1 Myhill-Nerodovy věty), je tedy minimální konečný automat přijímající L.

$\square$

Uzávěrové vlastnosti regulárních jazyků

Uzávěrové vlastnosti regulárních jazyků

Věta 2.7 Třída regulárních jazyků **je uzavřena** (mimo jiné) vzhledem k operacím:

$\cup$ (sjednocení),

$\cdot$ (konkatenace) a

* (iterace).

$\cap$ (průnik)

$co-$ (doplněk/komplement)

Důkaz. Uzavřenost na operace $\cup$, $\cdot$ a * plyne z definice regulárních množin a ekvivalence regulárních množin a regulárních jazyků.

Důkaz pokračuje dále.

Důkaz.

1. Dokážeme uzavřenost vzhledem ke komplementu nad abecedou Σ. K jazyku L sestrojíme *úplně definovaný* KA M.

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

   takový, že $L = L(M)$. Pak KA M'

   $$M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$$

   zřejmě přijímá jazyk $co{-}L = \Sigma^* \setminus L$ (tj. komplement jazyk L).

2. Uzavřenost vzhledem k průniku plyne z de Morganových zákonů:

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

   a tedy $L_1, L_2 \in \mathcal{L}_3 \Rightarrow L_1 \cap L2 \in \mathcal{L}_3$.

$\square$

❖ Alternativní důkazy uzávěrových vlastností (konstrukce příslušných gramatik a automatů) ukážeme na cvičení

Příklady na uzávěrové vlastnosti

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1 \in \mathcal{L}_3 \wedge L_2 \in \mathcal{L}_3 \iff (L_1 \cup L_2) \in \mathcal{L}_3$
2. $\forall L_1, L_2 \subseteq \Sigma^* : L_1 \in \mathcal{L}_3 \wedge L_2 \in \mathcal{L}_3 \Rightarrow (L_1 \diamond L_2) \in \mathcal{L}_3$, kde $L_1 \diamond L_2 = \{uv \mid u, v \in L_1 \cup L_2\}$

Příklady na uzávěrové vlastnosti

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1 \in \mathcal{L}_3 \wedge L_2 \in \mathcal{L}_3 \iff (L_1 \cup L_2) \in \mathcal{L}_3$
2. $\forall L_1, L_2 \subseteq \Sigma^* : L_1 \in \mathcal{L}_3 \wedge L_2 \in \mathcal{L}_3 \Rightarrow (L_1 \diamond L_2) \in \mathcal{L}_3$, kde
   $L_1 \diamond L_2 = \{uv \mid u, v \in L_1 \cup L_2\}$

Řešení 1: Tvrzení neplatí. Ukážeme, že neplatí implikace $\Leftarrow$. Zvolme $L_1 = \{a^n b^n \mid n > 0\}$ a $L_2 = \Sigma^*$. Pak $L_1 \cup L_2 = \Sigma^* \in \mathcal{L}_3$, ale $L_1 \notin \mathcal{L}_3$. Opačná implikace platí přímo z uzávěrových vlastností regulárních jazyků.

Příklady na uzávěrové vlastnosti

Rozhodněte a dokažte, zda platí následující tvrzení:

1. $\forall L_1, L_2 \subseteq \Sigma^* : L_1 \in \mathcal{L}_3 \wedge L_2 \in \mathcal{L}_3 \iff (L_1 \cup L_2) \in \mathcal{L}_3$
2. $\forall L_1, L_2 \subseteq \Sigma^* : L_1 \in \mathcal{L}_3 \wedge L_2 \in \mathcal{L}_3 \Rightarrow (L_1 \diamond L_2) \in \mathcal{L}_3$, kde $L_1 \diamond L_2 = \{uv \mid u, v \in L_1 \cup L_2\}$

Řešení 1: Tvrzení neplatí. Ukážeme, že neplatí implikace $\Leftarrow$. Zvolme $L_1 = \{a^n b^n \mid n > 0\}$ a $L_2 = \Sigma^*$. Pak $L_1 \cup L_2 = \Sigma^* \in \mathcal{L}_3$, ale $L_1 \notin \mathcal{L}_3$. Opačná implikace platí přímo z uzávěrových vlastností regulárních jazyků.

Řešení 2: Tvrzení platí. Uvědomme si, že $L_1 \diamond L_2 = (L_1 \cup L_2) \cdot (L_1 \cup L_2)$. Z uzavřenosti regulárních jazyků vzhledem k operacím $\cup$ a $\cdot$ tudíž dostáváme požadované tvrzení.

Rozhodnutelné problémy regulárních jazyků

Rozhodnutelné problémy v $\mathcal{L}_3$

Základní problémy:

- problém neprázdnosti: $L \neq \emptyset$?
- problém universality: $L = \Sigma^*$?
- problém náležitosti: $w \in L$?
- problém ekvivalence: $L(G_1) = L(G_2)$?

Věta 2.8 Ve třídě $\mathcal{L}_3$ je rozhodnutelný problémy **neprázdnosti** a **universality** jazyka i problém **náležitosti** řetězce (do jazyka).

Důkaz.
K jazyku $L \in \mathcal{L}_3$ sestrojíme úplně definovaný DKA M, $L = L(M)$:

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

neprázdnost: $L(M) \neq \emptyset \iff \exists q \in Q : (q \in F \wedge q$ je dostupný z $q_0)$
universalita: $L(M) = \Sigma^* \iff \forall q \in Q : (q \in F \vee q$ není dostupný z $q_0)$
náležitost: $w \in L \iff (q_0, w) \overset{*}{\vdash} (q, \varepsilon) \wedge q \in F$

$\square$

Věta 2.9 Nechť $L_1 = L(G_1)$ a $L_2 = L(G_2)$ jsou dva jazyky generované regulárními gramatikami G_1 a G_2. Pak je rozhodnutelný problém **ekvivalence**, tj. $L(G_1) = L(G_2)$.

Důkaz.
Nechť $M_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$, resp. $M_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2)$ jsou KA přijímající jazyky L_1, resp. L_2 takové, že $Q_1 \cap Q_2 = \emptyset$.

Vytvoříme konečný automat M takto:

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2, q_0^1, F_1 \cup F_2)$$

a vypočítáme relaci $\equiv$ nerozlišitelnosti stavů z $Q_1 \cup Q_2$ pro automat M.
Pak

$$L(G_1) = L(G_2) \iff q_0^1 \equiv q_0^2$$

□

Bezkontextové jazyky

Jazyky typu 2

Definice 4.1 Gramatika $G = (N, \Sigma, P, S)$ si nazývá bezkontextovou gramatikou, jestliže všechna pravidla z P mají tvar

$$A \to \alpha, \quad A \in N, \quad \alpha \in (N \cup \Sigma)^*$$

Lemma 4.1 Každý regulární jazyk je jazykem bezkontextovým.

❖ Proč studujeme bezkontextové jazyky?

Příklad 4.1 Jazyk $L = \{a^n b^n \mid n \geq 0\}$, jak víme, není jazykem regulárním, je však jazykem bezkontextovým:
$L = L(G)$ kde
$G = (\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S)$

Proč mohou BG „počítat“

- Sebevkládání pomocí pravidel $A \to \alpha A \beta$ kde $\alpha, \beta \in (N \cup \Sigma)^*$

Zkonstruujte bezkontextovou gramatiku pro jazyk $L = \{a^{3n}b^{2n} \mid n \geq 0\}$

Proč mohou BG „počítat“

- Sebevkládání pomocí pravidel $A \rightarrow \alpha A \beta$ kde $\alpha, \beta \in (N \cup \Sigma)^*$

Zkonstruujte bezkontextovou gramatiku pro jazyk $L = \{a^{3n}b^{2n} \mid n \geq 0\}$

$G = (\{S\}, \{a, b\}, \{S \rightarrow aaaSbb, S \rightarrow \varepsilon\}, S)$

Proč mohou BG „počítat“

- Sebevkládání pomocí pravidel $A \to \alpha A \beta$ kde $\alpha, \beta \in (N \cup \Sigma)^*$

Zkonstruujte bezkontextovou gramatiku pro jazyk $L = \{a^{3n}b^{2n} \mid n \geq 0\}$

$G = (\{S\}, \{a, b\}, \{S \to aaaSbb, S \to \varepsilon\}, S)$

- Jak docílit libovolné pořadí symbolů?

Zkonstruujte bezkontextovou gramatiku pro jazyk $L = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$

Proč mohou BG „počítat“

- Sebevkládání pomocí pravidel $A \to \alpha A \beta$ kde $\alpha, \beta \in (N \cup \Sigma)^*$

Zkonstruujte bezkontextovou gramatiku pro jazyk $L = \{a^{3n} b^{2n} \mid n \geq 0\}$

$G = (\{S\}, \{a, b\}, \{S \to aaaSbb, S \to \varepsilon\}, S)$

- Jak docílit libovolné pořadí symbolů?

Zkonstruujte bezkontextovou gramatiku pro jazyk $L = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$

$G = (\{S\}, \{a, b\}, \{S \to aSb, S \to bSa, S \to SS, S \to \varepsilon\}, S)$

Příklad bezkontextové gramatiky

- Pro účely demonstrace vysvětlovaných pojmů budeme v následujících příkladech používat následující gramatiku.

Příklad 4.2 $G = (\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla

$$
\begin{aligned}
S &\to AB \\
A &\to aAb \mid ab \\
B &\to bBc \mid bc
\end{aligned}
$$

Gramatika G generuje bezkontextový jazyk $L(G) = \{a^m b^{m+n} c^n \mid n \geq 1, m \geq 1\}$

Derivační strom

❖ Důležitým prostředkem pro grafické vyjádření struktury věty (její derivace) je strom, který se nazývá derivačním nebo syntaktickým stromem.

Definice 4.2 Nechť δ je věta nebo větná forma generovaná v gramatice $G = (N, \Sigma, P, S)$ a nechť $S = v_0 \Rightarrow v_1 \Rightarrow \ldots \Rightarrow v_k = \delta$ její derivace v G. Derivační strom příslušející této derivaci je vrcholově ohodnocený strom s těmito vlastnostmi:

1. Vrcholy derivačního stromu jsou ohodnoceny symboly z množiny $N \cup \Sigma \cup \{\varepsilon\}$; kořen stromu je označen výchozím symbolem S.
2. Přímé derivaci $v_{i-1} \Rightarrow v_i, i = 1, 2, \ldots, k$ kde
   - $v_{i-1} = \mu A \lambda,\ \mu, \lambda \in (N \cup \Sigma)^*,\ A \in N$
   - $v_i = \mu \alpha \lambda$
   - $A \rightarrow \alpha,\ \alpha = X_1 \ldots X_n$ je pravidlo z P,

   odpovídá právě n hran $(A, X_j), j = 1, \ldots, n$ vycházejících z uzlu A, jež jsou uspořádány zleva doprava v pořadí $(A, X_1), (A, X_2), \ldots, (A, X_n)$.
3. Ohodnocení koncových uzlů derivačního stromu vytváří zleva doprava větnou formu nebo větu δ (plyne z 1. a 2.).

Příklad derivačního stromu

Příklad 4.3 V gramatice z příkladu 4.2 můžeme generovat řetězec $aabbbbcc$ např. derivací:

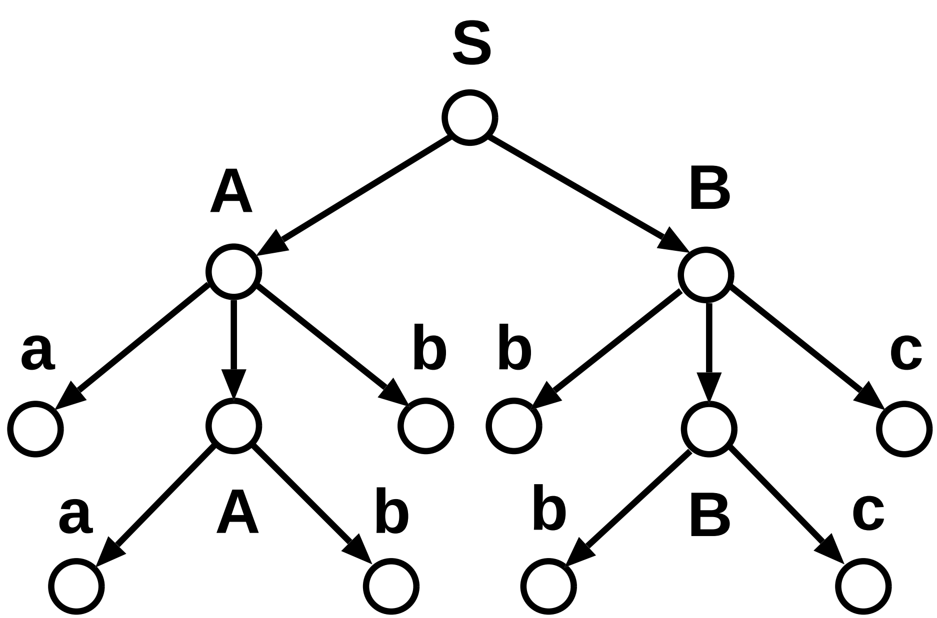
$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aAbbBc \Rightarrow aAbbbcc \Rightarrow aabbbbcc$$

Derivační strom odpovídající této derivaci vypadá takto (po stranách jsou uvedena použitá pravidla):

$S \rightarrow AB$

$A \rightarrow aAb$

$A \rightarrow ab$



$B \rightarrow bBc$

$B \rightarrow bc$

Levá a pravá derivace

❖ Ukažme si i jiné derivace věty $aabbbbcc$, které se liší v pořadí, v němž byly vybírány nonterminály pro přímé derivace.

1. $S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbbBc \Rightarrow aabbbbcc$
2. $S \Rightarrow AB \Rightarrow AbBc \Rightarrow Abbcc \Rightarrow aAbbbc \Rightarrow aabbbbcc$

Definice 4.3 Nechť $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = \alpha$ je derivace větné formy α. Jestliže byl v každém řetězci $\alpha_i, i = 1, \ldots, n-1$ přepsán nejlevější (nejpravější) nonterminál, pak tuto derivaci nazýváme levou (pravou) derivací větné formy α.

Výše uvedené příklady derivací představují levou (1.) a pravou (2.) derivaci.

Lemma 4.2 Je-li $S \equiv \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n \equiv w$ levá, resp. pravá derivace věty w, pak každá z větných forem α_i, $i = 1, 2, \ldots, n-1$ má tvar:

$$x_i A_i \beta_i \text{ kde } x_i \in \Sigma^*, A_i \in N, \beta_i \in (N \cup \Sigma)^*$$

resp.

$$\gamma_i B_i y_i \text{ kde } y_i \in \Sigma^*, B_i \in N, \gamma_i \in (N \cup \Sigma)^*$$

t.j. větné formy levé, resp. pravé derivace mají terminální prefixy, resp. sufixy.

Víceznačnost gramatik

Definice 4.4 Nechť G je gramatika. Říkáme, že věta w generovaná gramatikou G je víceznačná, existují-li alespoň dva různé derivační stromy s koncovými uzly tvořícími větu w. Gramatika G je víceznačná, pokud generuje alespoň jednu víceznačnou větu. V opačném případě mluvíme o jednoznačné gramatice.
Jazyky, které lze generovat víceznačnou gramatikou, ale které nelze generovat jednoznačnou gramatikou, se nazývají jazyky s inherentní víceznačností.

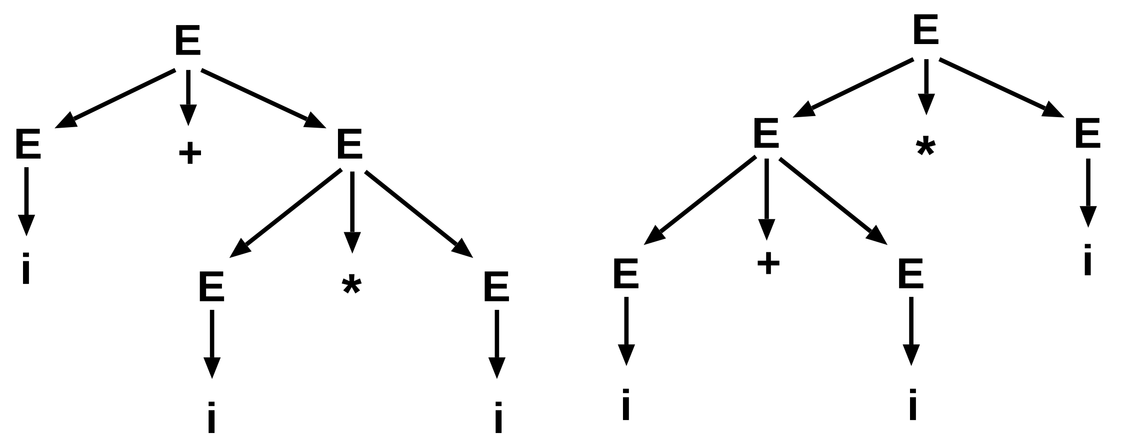
- Problém víceznačnosti gramatik je nerozhodnutelný, tj. neexistuje algoritmus, který by byl schopen v konečném čase rozhodnout, zda daná gramatika je nebo není víceznačná.
- Víceznačnost gramatiky je pokládána za negativní rys (vede k větám, které mají několik interpretací). Na druhé straně může být víceznačná gramatika jednodušší než odpovídající jednoznačná gramatika.

Víceznačnost gramatik

Příklad 4.4 Uvažujme gramatiku $G = (\{E\}, \{+, -, *, /, (,)\}, P, E)$, kde P je množina pravidel

$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (\ E\ ) \mid i$$

Jazyk $L(G)$ je tvořen aritmetickými výrazy s binárními operacemi. Gramatika G je na rozdíl od gramatiky z příkladu 4.2 víceznačná. Vezměme například větu $i + i * i$ a uvažujme všechny možné derivační stromy.



Není jasné, zda první operací bude násobení (derivační strom vlevo), nebo sčítání (derivační strom vpravo).

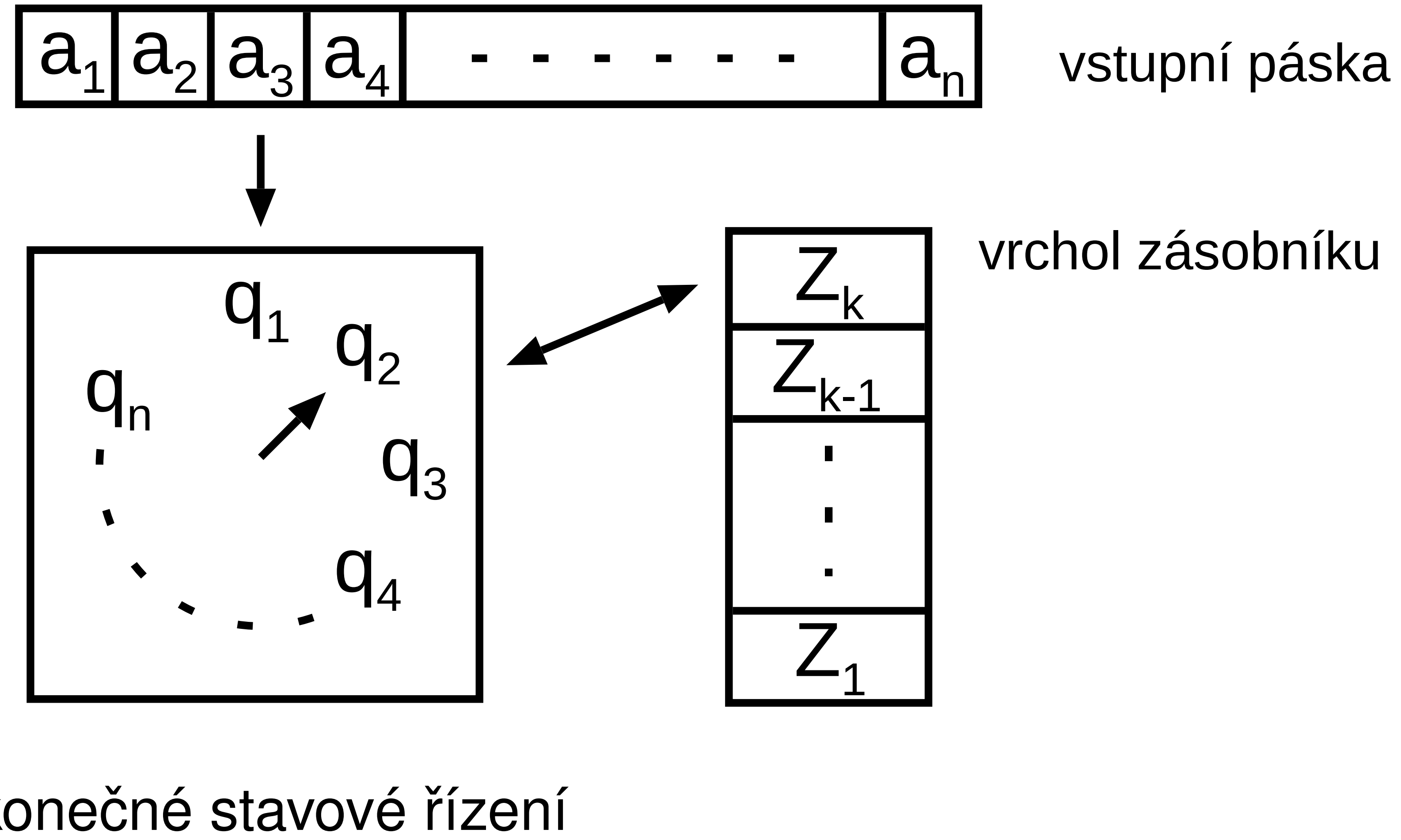
Příklad 4.5 Jednoznačnou gramatikou generující tentýž jazyk je gramatika $G = (\{E, T, F\}, \{+, -, *, /, (,), i\}, P, E)$ s množinou přepisovacích pravidel P definovanou následujícím způsobem:

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

Zásobníkové automaty

Základní schéma

Schéma zásobníkového automatu:



Základní definice

Definice 4.5 Zásobníkový automat P je n-tice $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

1. Q je konečná množina vnitřních stavů
2. Σ je konečná vstupní abeceda
3. Γ je konečná zásobníková abeceda
4. δ je přechodová funkce ve tvaru $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$
5. $q_0 \in Q$ je počáteční stav
6. $Z_0 \in \Gamma$ je startovací symbol zásobníku
7. $F \subseteq Q$ je množina koncových stavů

Konfigurace a přechod ZA

Definice 4.6 Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Konfigurací automatu P nazveme trojici $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$, kde

1. q je přítomný stav vnitřního řízení
2. w je dosud nezpracovaná část vstupního řetězce
3. α je obsah zásobníku ($\alpha = Z_{i_1} Z_{i_2} \ldots Z_{i_k}$, Z_{i_1} je vrchol)

Přechod ZA P je binární relace $\vdash_P$ definovaná na množině konfigurací:

$$(q, w, \beta) \vdash_P (q', w', \beta') \overset{def}{\Longleftrightarrow} w = aw' \wedge \beta = Z\alpha \wedge \beta' = \gamma\alpha \wedge (q', \gamma) \in \delta(q, a, Z),$$

kde $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w, w' \in \Sigma^*$, $Z \in \Gamma$ a $\alpha, \beta, \beta', \gamma \in \Gamma^*$.

- Je-li $a = \varepsilon$, pak odpovídající přechod nazýváme ε-přechodem.
- Relace $\vdash_P^i$, $\vdash_P^*$, $\vdash_P^+$ jsou definovány obvyklým způsobem.
- Platí-li pro řetězec $w \in \Sigma^*$ relace $(q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \gamma)$, kde $q \in F$ a $\gamma \in \Gamma^*$, pak říkáme, že w je přijímán zásobníkovým automatem P (q_0, w, Z_0), resp. (q, ε, γ) je počáteční, resp. koncová konfigurace.
- Definujeme jazyk přijímaný zásobníkovým automatem P: $L(P) = \{w | (q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \gamma) \wedge q \in F\}$.

Příklad zásobníkového automatu

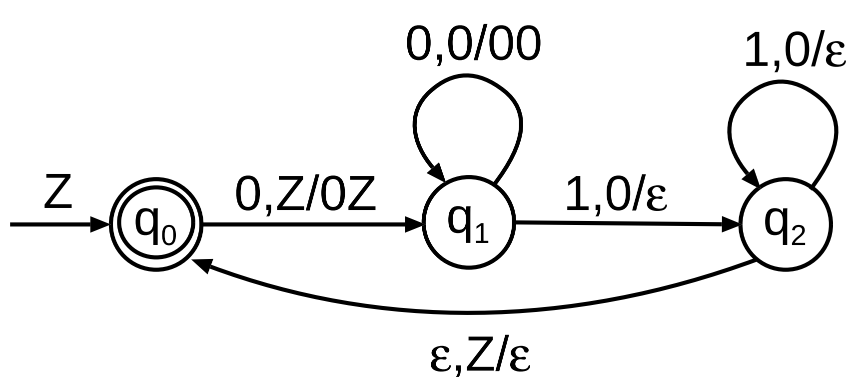
Příklad 4.6 Sestrojme zásobníkový automat, který přijímá jazyk $L = \{0^n 1^n \mid n \geq 0\}$.
– Řešením je $P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{Z, 0\}, \delta, q_0, Z, \{q_0\})$, kde

$\delta(q_0, 0, Z) = \{(q_1, 0Z)\}$
$\delta(q_1, 0, 0) = \{(q_1, 00)\}$
$\delta(q_1, 1, 0) = \{(q_2, \varepsilon)\}$
$\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$
$\delta(q_2, \varepsilon, Z) = \{(q_0, \varepsilon)\}$

– Při přijetí řetězce 0011 projde P těmito konfiguracemi:

$(q_0, 0011, Z) \vdash (q_1, 011, 0Z) \vdash (q_1, 11, 00Z) \vdash (q_2, 1, 0Z) \vdash (q_2, \varepsilon, Z) \vdash (q_0, \varepsilon, \varepsilon)$

– Zásobníkové automaty lze také popsat přechodovým diagramem, jak je ilustrováno níže na právě sestrojeném automatu P:



Návrh složitějších automatů

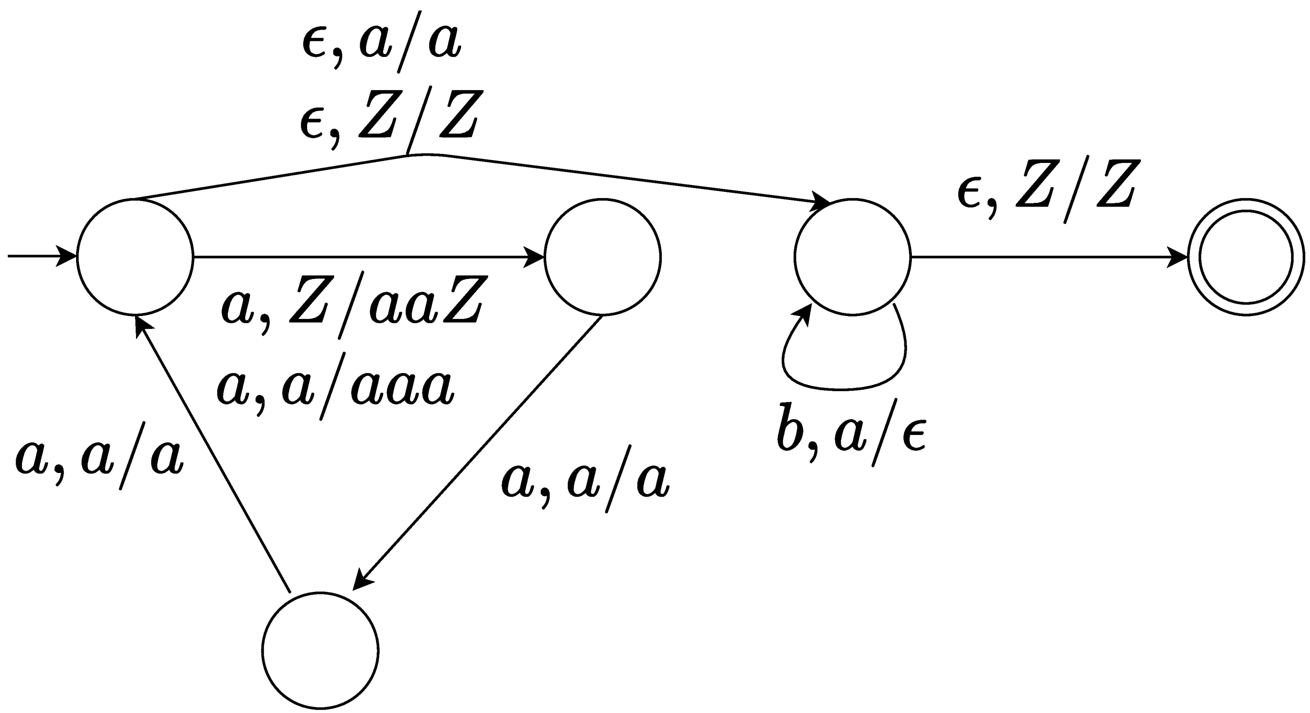
❖ Pokročilejší práce se zásobníkem.

Zkonstruujte zásobníkový automat pro jazyk $L = \{a^{3n}b^{2n} \mid n \geq 0\}$

Návrh složitějších automatů

❖ Pokročilejší práce se zásobníkem.

Zkonstruujte zásobníkový automat pro jazyk $L = \{a^{3n}b^{2n} \mid n \geq 0\}$



Varianty zásobníkových automatů

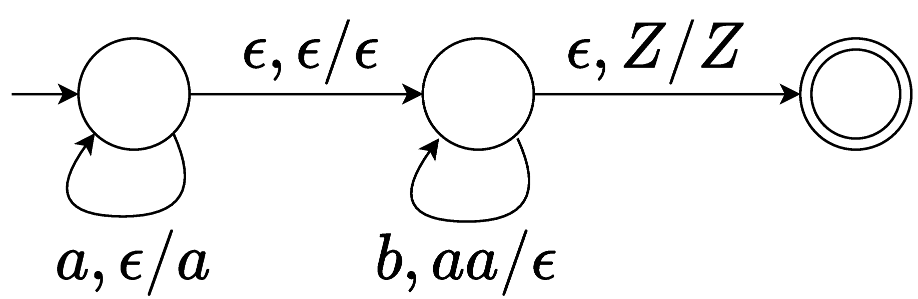
Rozšířený zásobníkový automat

Definice 4.7 Rozšířený zásobníkový automat P je sedmice $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde δ je přechodová funkce definovaná takto:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \to 2^{Q \times \Gamma^*}$$

Ostatní složky mají stejný význam jako v definici 4.5.

Příklad 4.7 Zkonstruujte RZA pro jazyk $L = \{a^{2n}b^n \mid n \geq 0\}$



Ekvivalence RZA a ZA

Věta 4.1 Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je rozšířený zásobníkový automat. Pak existuje zásobníkový automat P_1 takový, že $L(P_1) = L(P)$.

Důkaz. Položme $m = max\{|\alpha| \mid \delta(q, a, \alpha) \neq \emptyset$ pro nějaké $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ a $\alpha \in \Gamma^*\}$.

Zásobníkový automat P_1 budeme konstruovat tak, aby simuloval automat P.

Protože automat P neurčuje přechody podle vrcholu zásobníku, ale podle vrcholového řetězce zásobníku, bude automat P_1 ukládat m vrcholových symbolů v jakési vyrovnávací paměti řídící jednotky tak, aby na počátku každého přechodu věděl, jakých m vrcholových symbolů je v zásobníku automatu P.

Nahrazuje-li automat P k vrcholových symbolů řetězcem délky l, pak se totéž provede ve vyrovnávací paměti automatu P_1.

Jestliže $l < k$, pak P_1 realizuje $k - l$ ε-přechodů, které přesouvají $k - l$ symbolů z vrcholu zásobníku do vyrovnávací paměti. Automat P_1 pak může simulovat další přechod automatu P.

Je-li $l \geq k$ pak se symboly přesouvají z vyrovnávací paměti do zásobníku.

Formálně můžeme konstrukci zásobníkového automatu P_1 popsat takto:
$P_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, Z_1, F_1)$, kde

1. $Q_1 = \{[q, \alpha] | q \in Q, \alpha \in \Gamma_1^* \wedge 0 \leq |\alpha| \leq m\}$
2. $\Gamma_1 = \Gamma \cup \{Z_1\}$
3. Zobrazení δ_1 je definováno takto:
   (a) Předpokládejme, že $\delta(q, a, X_1 \dots X_k)$ obsahuje $(r, Y_1 \dots Y_l)$.
      i. Jestliže $l \geq k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$, pak $\delta_1([q, X_1 \dots X_k \alpha], a, Z)$ obsahuje $([r, \beta], \gamma Z)$, kde $\beta\gamma = Y_1 \dots Y_l \alpha$ a $|\beta| = m$.
      ii. Je-li $l < k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$, pak $\delta_1([q, X_1 \dots X_k \alpha], a, Z)$ obsahuje $([r, Y_1 \dots Y_l \alpha Z], \varepsilon)$.
   (b) Pro všechna $q \in Q, Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| < m$, platí $\delta_1([q, \alpha], \varepsilon, Z) = \{([q, \alpha Z], \varepsilon)\}$. Tato pravidla vedou k naplnění vyrovnávací paměti.

4. $q_1 = [q_0, Z_0, Z_1^{m-1}]$. Vyrovnávací paměť obsahuje na počátku symbol Z_0 na vrcholu a $m-1$ symbolů Z_1 na dalších místech. Symboly Z_1 jsou speciální znaky pro označení dna zásobníku.
5. $F_1 = \{[q, \alpha] \mid q \in F, \alpha \in \Gamma_1^*\}$
   Lze ukázat, že $(a, aw, X_1 \dots X_k X_{k+1} \dots X_n) \vdash_P (r, w, Y_1 \dots Y_l X_{k+1} \dots X_n)$ platí, právě když $([q, \alpha], aw, \beta) \vdash_{P_1}^{+} ([r, \alpha'], w, \beta')$ kde
   $\alpha\beta = X_1 \dots X_n Z_1^m$
   $\alpha'\beta' = Y_1 \dots Y_l X_{k+1} \dots X_n Z_1^m$
   $|\alpha| = |\alpha'| = m$
   a mezi těmito dvěma konfiguracemi automatu P_1 není žádná konfigurace, ve které by druhý člen stavu (vyrovnávací paměť) měl délku m.

Tedy relace $(q_0, w, Z_0) \vdash_P (q, \varepsilon, \alpha)$ pro $q \in F, \alpha \in \Gamma^*$ platí, právě když $([q_0, Z_0, Z_1^{m-1}], w, Z_1) \vdash_{P_1}^{*} ([q, \beta], \varepsilon, \gamma)$, kde $|\beta| = m$ a $\beta\gamma = \alpha Z_1^m$. Tedy $L(P) = L(P_1)$. $\square$

ZA přijímající s vyprázdněním zás.

Definice 4.8 Zásobníkový automat nebo rozšířený zásobníkový automat $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, \emptyset)$ přijímá s vyprázdněním zásobníku, pokud

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$$

Věta 4.2 Ke každému ZA (resp. RZA) P existuje ZA (resp. RZA) P', který přijímá s vyprázdněním zásobníku, takový, že $L(P) = L(P')$.

Důkaz. (Hlavní myšlenka) Opět budeme konstruovat automat P' tak, aby simuloval automat P. Kdykoli automat P dospěje do koncového stavu, přejde automat P' do speciálního stavu q_ε, který způsobí vyprázdnění zásobníku. Musíme však uvážit situaci, kdy automat P je v konfiguraci s prázdným zásobníkem, nikoli však v koncovém stavu. Abychom zabránili případům, že automat P' přijímá řetězec, který nemá být přijat, přidáme k zásobníkové abecedě automatu P' znak, jenž bude označovat dno zásobníku a může být vybrán pouze tehdy, je-li automat P' ve stavu q_ε. $\square$

Ekvivalence bezkontextových jazyků a jazyků přijímaných zásobníkovým automatem

Označme třídu všech jazyků přijímaných zásobníkovými automaty symbolem $\mathcal{L}_P$. Dokážeme, že $\mathcal{L}_2 = \mathcal{L}_P$ postupem analogickým s důkazem tvrzení $\mathcal{L}_3 = \mathcal{L}_M$. Ukážeme tedy, že

- ke každé bezkontextové gramatice existuje ekvivalentní zásobníkový automat, tj. $\mathcal{L}_2 \subseteq \mathcal{L}_P$
- a ke každému zásobníkovému automatu existuje ekvivalentní gramatika typu 2, tj. $\mathcal{L}_P \subseteq \mathcal{L}_2$

Pro důkaz inkluze $\mathcal{L}_2 \subseteq \mathcal{L}_P$ zkonstruujeme (redundantně) automaty modelující oba typy syntaktické analýzy příslušného bezkontextového jazyka.

$\mathcal{L}_2 \subseteq \mathcal{L}_P$ – Analýza shora dolů

Věta 4.3 Nechť $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Pak existuje zásobníkový automat P, který přijímá s vyprázdněním zásobníku takový, že $L(G) = L(P)$.

Důkaz. Zásobníkový automat P vytvoříme tak, aby vytvářel levou derivaci vstupního řetězce v gramatice G (modeloval syntaktickou analýzu shora dolů). Nechť P je ZA:

$$P = (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \emptyset), \text{ kde } \delta \text{ je určena takto:}$$

- Je-li $A \to \alpha$ pravidlo z P, pak $(q, \alpha) \in \delta(q, \varepsilon, A)$
- $\delta(q, a, a) = \{(q, \varepsilon)\}$ pro všechna $a \in \Sigma$

Indukcí lze dokázat ekvivalenci

$$A \Rightarrow^m w \;\Leftrightarrow\; (q, w, A) \vdash^n (q, \varepsilon, \varepsilon), m, n \geq 1, w \in \Sigma^*$$

což pro případ $A = S$ znamená $L(G) = L(P)$.

$\square$

Příklad 4.8 Ke gramatice

$$G = (\{S\}, \{0,1\}, \{S \to 0S1, S \to 01\}, S),$$

sestrojíme zásobníkový automat P, který modeluje syntaktickou analýzu shora dolů:

$$P = (\{q\}, \{0,1\}, \{S,0,1\}, \delta, q, S, 0), \text{ kde}$$

$$\begin{aligned}
&\delta(q,\varepsilon,S) = \{(q,0S1),(q,01)\}\\
&\delta(q,0,0) = \{(q,\varepsilon)\}\\
&\delta(q,1,1) = \{(q,\varepsilon)\}
\end{aligned}$$

Skutečně, např. derivaci

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

odpovídá posloupnost přechodů automatu P:

$(q,000111,S) \vdash (q,000111,0S1) \vdash (q,00111,S1) \vdash (q,00111,0S11) \vdash (q,0111,S11) \vdash$
$(q,0111,0111) \vdash (q,111,111) \vdash (q,11,11) \vdash (q,1,1) \vdash (q,\varepsilon,\varepsilon)$

$\mathcal{L}_2 \subseteq \mathcal{L}_P$ – Analýza zdola nahoru

Věta 4.4 Nechť $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Pak lze ke gramatice G sestrojit RZA P takový, že $L(G) = L(P)$.

Důkaz. RZA P sestrojme tak, aby modeloval syntaktickou analýzu zdola nahoru. Nechť P je RZA

$$P = (\{q, r\}, \Sigma, N \cup \Sigma \cup \{\#\}, \delta, q, \#, \{r\})$$

kde δ je určena takto:

1. Je-li $A \to \alpha$ pravidlo z P, pak $\delta(q, \varepsilon, \alpha)$ obsahuje (q, A). - redukce
2. $\delta(q, a, \varepsilon) = \{(q, a)\}$ pro všechna $a \in \Sigma$ - shift
3. $\delta(q, \varepsilon, S\#) = \{(r, \varepsilon)\}$

Indukcí lze opět dokázat $L(G) = L(P)$.

$\square$

Příklad 4.9 Na gramatiku

$$G = (\{S\}, \{0,1\}, \{S \to 0S1, S \to 01\}, S)$$

aplikujeme nyní větu 5.4. Výsledný RZA bude mít tvar:

$$P = (\{q,r\}, \{0,1\}, \{0,1,S,\#\}, \delta, q, \#, \{r\})$$

kde δ je definována takto

$$
\begin{array}{ll}
\delta(q,\varepsilon,0S1) = \{(q,S)\} & \text{redukce} \\
\delta(q,\varepsilon,01) = \{(q,S)\} & \text{redukce} \\
\delta(q,0,\varepsilon) = \{(q,0)\} & \text{shift} \\
\delta(q,1,\varepsilon) = \{(q,1)\} & \text{shift} \\
\delta(q,\varepsilon,\#S) = \{(r,\varepsilon)\} &
\end{array}
$$

Derivaci $S \Rightarrow 0S1 \Rightarrow 0011$ odpovídá posloupnosti konfigurací $(q,0011,\#) \vdash (q,011,\#0) \vdash (q,11,\#00) \vdash (q,1,\#001) \vdash (q,1,\#0S) \vdash (q,\varepsilon,\#0S1) \vdash (q,\varepsilon,\#S) \vdash (r,\varepsilon,\varepsilon)$

Poznámka 4.1 Vrchol zásobníku uvádíme, pro lepší čitelnost, vpravo

$*\mathcal{L}_P \subseteq \mathcal{L}_2*$

Věta 4.5 Nechť $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ je zásobníkový automat přijímající s vyprázdněním zásobníku. Pak existuje gramatika $G = (N, \Sigma, P, S)$ taková, že

$$L(P) = L(G).$$

Důkaz. Gramatiku G budeme definovat formálně takto:

- $N = \{[qZr] \mid q, r \in Q, Z \in \Gamma\} \cup \{S\}$
- Jestliže $(r, X_1X_2 \dots X_k) \in \delta(q, a, Z)$, $k \geq 1$, pak k P přidej pravidla tvaru
  $$[qZs_k] \to a[rX_1s_1][s_1X_2s_2] \dots [s_{k-1}X_ks_k]$$
  pro každou posloupnost stavů $s_1, s_2, \dots, s_k$ z množiny Q
- Jestliže $(r, \varepsilon) \in \delta(q, a, Z)$, pak k P přidej pravidlo $[qZr] \to a$ (pro $a \in \Sigma \cup \{\varepsilon\}$)
- Pro každý stav $q \in Q$ přidej k P pravidlo $S \to [q_0Z_0q]$

Indukcí lze dokázat $S \Rightarrow [q_0Z_0q] \Rightarrow^+ w$ právě když $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$

$\square$

Ekvivalence $\mathcal{L}_2 = \mathcal{L}_P$

Věta 4.6 **Třída bezkontextových jazyků a třída jazyků přijímaných zásobníkovými automaty jsou totožné.**

Důkaz. Přímý důsledek vět 4.4 a 4.5. $\square$

Transformace bezkontextových gramatik

Ekvivalentní gramatiky

Definice 4.9 Nechť G_1 a G_2 jsou gramatiky libovolného typu Chomského klasifikace. G_1 a G_2 jsou ekvivalentní, pokud $L(G_1) = L(G_2)$.

Věta 4.7 Nechť $G = (N, \Sigma, P, S)$ je gramatika, $A \to \alpha B \beta$, $B \in N$, $\alpha, \beta \in (N \cup \Sigma)^*$ je pravidlo z P a nechť $B \to \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n$ jsou všechna B-pravidla z P. Pak gramatika $G' = (N, \Sigma, P', S)$ kde

$$P' = P \setminus \{A \to \alpha B \beta\} \cup \{A \to \alpha \gamma_1 \beta, A \to \alpha \gamma_2 \beta, \dots, A \to \alpha \gamma_n \beta\}$$

je ekvivalentní s gramatikou G.

Důkaz. Na cvičení. □

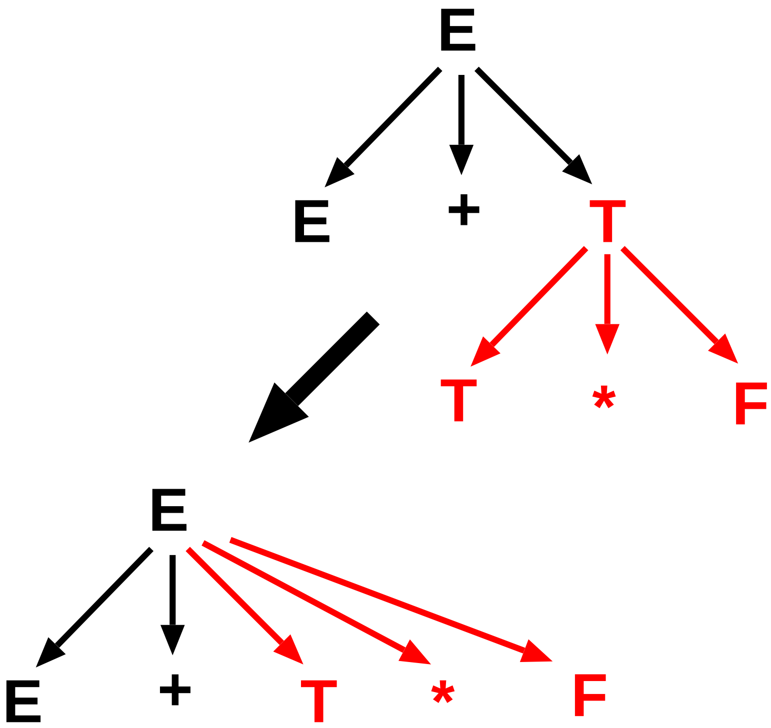
Příklad 4.10
Gramatiky s pravidly

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

resp.

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

jsou ekvivalentní.



Nedostupné a zbytečné symboly

Definice 4.10 Nechť $G = (N, \Sigma, P, S)$ je gramatika a $X \in N \cup \Sigma$ symbol. Říkáme, že symbol X je nedostupný v G, jestliže v G neexistuje derivace $S \Rightarrow^* \alpha X \beta$ pro nějaké $\alpha, \beta \in (N \cup \Sigma)^*$. Symbol X nazýváme zbytečný v G, jestliže v G neexistuje derivace tvaru $S \Rightarrow^* \alpha X \beta \Rightarrow^* zxy$ pro nějaké $\alpha, \beta \in (N \cup \Sigma)^*$ a $zxy \in \Sigma^*$.

Příklad 4.11 Uvažujme gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$ s pravidly:

$S \rightarrow SB \mid a$
$A \rightarrow b$
$B \rightarrow Ba$

Symboly A, B, b jsou zbytečné. Symboly A, b jsou nedostupné.

Poznámka 4.2 $G' = (\{S\}, \{a\}, \{S \rightarrow a\}, S)$ je ekvivalentní s G.

Nonterminály generující terminální řetězce

Algoritmus 4.1 Výpočet množiny nonterminálů generujících terminální řetězce
Vstup: Gramatika $G = (N, \Sigma, P, S)$.
Výstup: Množina $N_t = \{A \mid A \Rightarrow^+ w, w \in \Sigma^*\}$.
Metoda: Počítáme množiny $N_0, N_1, N_2, \ldots$ rekurentně takto:

1. N_0 := $\emptyset, i = 1$
2. N_i := $\{A \mid A \rightarrow \alpha$ je v P a $\alpha \in (N_{i-1} \cup \Sigma)^*\}$
3. Je-li $N_i \neq N_{i-1}$, i := $i + 1$ a vrať se k (2). Je-li $N_i = N_{i-1}$, polož $N_t = N_i$ a skonči.

Dostupné symboly

Algoritmus 4.2 Výpočet množiny dostupných symbolů
Vstup: Gramatika $G = (N, \Sigma, P, S)$.
Výstup: Množina $V = \{X \mid S \Rightarrow^* \alpha X \beta, \alpha, \beta \in (N \cup \Sigma)^*\}$.
Metoda:

1. V_0 := $\{S\}, i = 1$
2. V_i := $\{X \mid A \rightarrow \alpha X \beta$ je v P a $A \in V_{i-1}\} \cup V_{i-1}$
3. Je-li $V_i \neq V_{i-1}$, i := $i + 1$ a vrať se k (2). Je-li $V_i = V_{i-1}$, polož $V = V_i$ a skonči.

Odstranění zbytečných symbolů

Algoritmus 4.3 Odstranění zbytečných symbolů
Vstup: Gramatika $G = (N, \Sigma, P, S)$.
Výstup: Gramatika $G' = (N', \Sigma', P', S)$ bez zbytečných symbolů, $L(G) = L(G')$.
Metoda:

1. Aplikací algoritmu 4.1 na G vypočti množinu N_t.
2. Polož $\overline{G} = (N_t \cup \{S\}, \Sigma, \overline{P}, S)$, kde
   $\overline{P} = \{A \to \alpha | (A \to \alpha) \in P \wedge A \in N_t \wedge \alpha \in (N_t \cup \Sigma)^*\}$.
3. Aplikací algoritmu 4.2 na $\overline{G}$ vypočti množinu V.
4. Výslednou gramatiku G' sestroj takto:
   (a) $N' = N_t \cap V$
   (b) $\Sigma' = \Sigma \cap V$
   (c) $P' = \{A \to \alpha | (A \to \alpha) \in P \wedge A \in N' \wedge \alpha \in V^*\}$

Poznámka: Sjednocení $N_t \cup \{S\}$ v bodě 2 řeší případ, kdy $L(G) = \emptyset$ a $N_t = \emptyset$, ovšem gramatika musí mít svůj startovací symbol.

Příklad 4.12 Uvažujme gramatiku
$G = (\{S, A, B\}, \{a, b\}, \{S \to a, S \to A, A \to AB, B \to b\}, S)$.

1. $N_0 = \emptyset$, $N_1 = \{S, B\}$, $N_2 = N_1 = N_t = \{S, B\}$
2. $\overline{G} = (\{S, B\}, \{a, b\}, \{S \to a, B \to b\}, S)$
3. $V_0 = \{S\}$, $V_1 = \{S, a\}$, $V_2 = V_1 = V = \{S, a\}$
4. $G' = (\{S\}, \{a\}, \{S \to a\}, S)$.

Poznámka 4.3 Pořadí kroků 2. a 4. je významné.

Odstranění ε-pravidel

Definice 4.11 G je gramatikou bez ε-pravidel, jestliže neobsahuje žádné ε-pravidlo (pravidlo tvaru $A \to \varepsilon$), nebo, pokud $\varepsilon \in L(G)$, potom obsahuje jediné ε-pravidlo $S \to \varepsilon$ a S se pak nevyskytuje na pravé straně žádného přepisovacího pravidla.

Algoritmus 4.4 Transformace na gramatiku bez ε-pravidel
Vstup: Gramatika $G = (N, \Sigma, P, S)$.
Výstup: Gramatika $G' = (N', \Sigma, P', S')$ bez ε-pravidel ekvivalentní s G.
Metoda:

1. Vypočítej množinu $N_\varepsilon = \{A \mid A \Rightarrow^+ \varepsilon\}$
2. Každé pravidlo z P, které není ε-pravidlem, uvažuj ve tvaru $A \to \alpha_0 B_1 \alpha_1 B_2 \dots B_k \alpha_k$, kde $B_i \in N_\varepsilon, \alpha_i \in (N \setminus N_\varepsilon \cup \Sigma)^*$ pro $i = 1, \dots, k$
   Toto pravidlo nahraď množinou pravidel, které vzniknou všemi možnými substitucemi $B_i \rightsquigarrow B_i$ a $B_i \rightsquigarrow \varepsilon$ pro $i = 1, \dots, k$ (to jest substitucemi, kdy nonterminály z N_ε jsou alternativně ponechávány a vypouštěny). Počet těchto substitucí (nových pravidel) je zřejmě 2^k.
3. Všechna ε-pravidla vypusť.
4. Pokud $S \in N_\varepsilon$, pak $N' = N \cup \{S'\}$, kde S' je nový nonterminální symbol, a přidej pravidla $S' \to \varepsilon \mid S$, v opačném případě $N' = N$, $S' = S$

Příklad 4.13 Uvažujme gramatiku $G = (\{A, B, C\}, \{a, b, c\}, P, A)$ s pravidly:
$A \rightarrow AbAcBC \mid \varepsilon \mid a$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow c \mid \varepsilon$

1. $N_\varepsilon = \{A, B, C\}$
2. $A \rightarrow AbAcBC$
   $A \rightarrow \ bAcBC$
   $A \rightarrow Ab\ cBC$
   $A \rightarrow \ b\ cBC$
   $\vdots$
   $A \rightarrow \ bc$
   $A \rightarrow a$
   $B \rightarrow b$
   $C \rightarrow c$
3. $A' \rightarrow \varepsilon$
   $A' \rightarrow A$

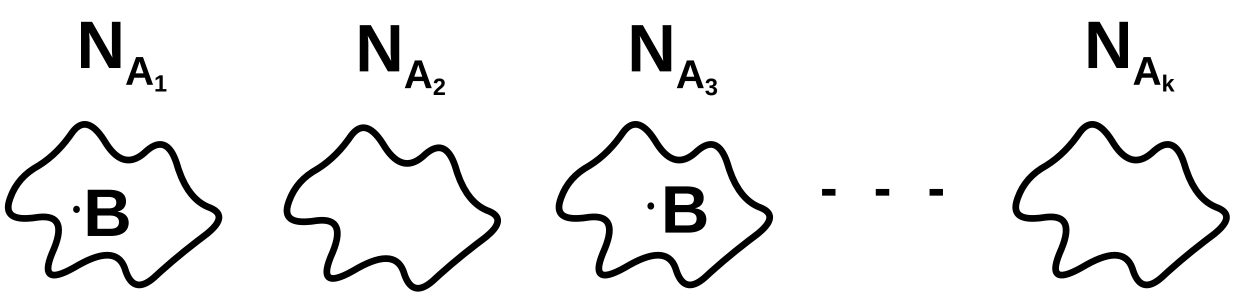
A' je nový startovací symbol.

Odstranění jednoduchých pravidel

Definice 4.12 Pravidlo tvaru $A \to B$, kde $A, B \in N$ nazýváme jednoduché pravidlo.

Algoritmus 4.5 Transformace na gramatiku bez jednoduchých pravidel
Vstup: Gramatika $G = (N, \Sigma, P, S)$ bez ε-pravidel.
Výstup: Gramatika $G' = (N, \Sigma, P', S)$ bez jednoduchých pravidel ekvivalentní s G.
Metoda:

1. Pro všechny $A \in N$ vypočítej množinu $N_A = \{B \mid A \Rightarrow^* B\}$, polož $P' := \emptyset$.
2. Nechť $B \to \alpha$, $\alpha \notin N$ je pravidlo z P. Potom k P' přidej nová pravidla $A_i \to \alpha$ pro všechny A_i, kde $B \in N_{A_i}$.
3. Výsledná množina pravidel P' tvoří všechna pravidla gramatiky G' (neobsahuje jednoduchá pravidla).



Nová pravidla: $A_1 \to \alpha$ $\qquad$ $A_3 \to \alpha$

Příklad 4.14 Uvažujme gramatiku $G = (\{E, T, F\}, \{i, +, *, (,)\}, P, E)$ s pravidly:
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid i$

1. Nalezneme množiny N_A pro všechny $A \in N$:
   $N_E = \{E, T, F\} \quad N_T = \{T, F\} \quad N_F = \{F\}$
2. Doplňujeme nová pravidla a vypouštíme jednoduchá pravidla:
   $E \rightarrow E + T \mid T * F \mid (E) \mid i$
   $T \rightarrow T * F \mid (E) \mid i$
   $F \rightarrow (E) \mid i$

Cyklus

Definice 4.13 Nechť $G = (N, \Sigma, P, S)$ je gramatika, $A \in N$. Gramatika G obsahuje cyklus, jestliže $A \Rightarrow^+ A$.

Věta 4.8 Jestliže gramatika $G = (N, \Sigma, P, S)$ obsahuje cyklus v nonterminálu A, $A \in N$ a jestliže existuje derivace

$$S \Rightarrow^* \alpha A \beta \Rightarrow^+ w, w \in \Sigma^*, \alpha, \beta \in (N \cup \Sigma)^*$$

pak G je víceznačná.

Důkaz. Existuje-li derivace

$$S \Rightarrow^* \alpha A \beta \Rightarrow^+ w$$

pak vzhledem k existenci cyklu $A \Rightarrow^+$ existuje i derivace

$$S \Rightarrow^* \alpha A \beta \Rightarrow \alpha \gamma_1 \beta \Rightarrow \alpha \gamma_2 \beta \Rightarrow \ldots \Rightarrow \alpha A \beta \Rightarrow^+ w$$

Těmto derivacím přísluší různé derivační stromy. $\square$

Zdroje cyklu

- Jednoduchá pravidla (tvaru $A \to B$), např.

$$A \Rightarrow B \Rightarrow C \Rightarrow A$$

  v důsledku pravidel $A \to B$, $B \to C$, $C \to A$
- ε-pravidla, např.

$$A \Rightarrow AB \Rightarrow A$$

  v důsledku pravidla $B \to \varepsilon$

Vlastní gramatika

Definice 4.14 Gramatika bez zbytečných symbolů, ε-pravidel a bez cyklů se nazývá vlastní gramatikou.

Věta 4.9 Každá bezkontextová gramatika má ekvivalentní vlastní gramatiku.

Důkaz. Aplikací algoritmů 4.3 a 4.4 odstraníme zbytečné symboly a ε-pravidla. Jestliže po této transformaci existuje v G derivace $A \Rightarrow^+ A$, tj. cyklus, pak jeho příčinou mohou být pouze jednoduchá pravidla a ty lze odstranit aplikací algoritmu 4.5. Na závěr je nutné znovu aplikovat algoritmus 4.3, jelikož po aplikaci algoritmu 4.5 mohou znovu vzniknout nedostupné symboly. $\square$

Odstranění levé rekurze

❖ Základem algoritmu je odstranění přímé levé rekurze, tj. levě rekurzivních pravidel, podle následující transformace:

Věta 4.10 Nechť gramatika G má levě rekurzivní pravidla v nonterminálu A a nechť

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

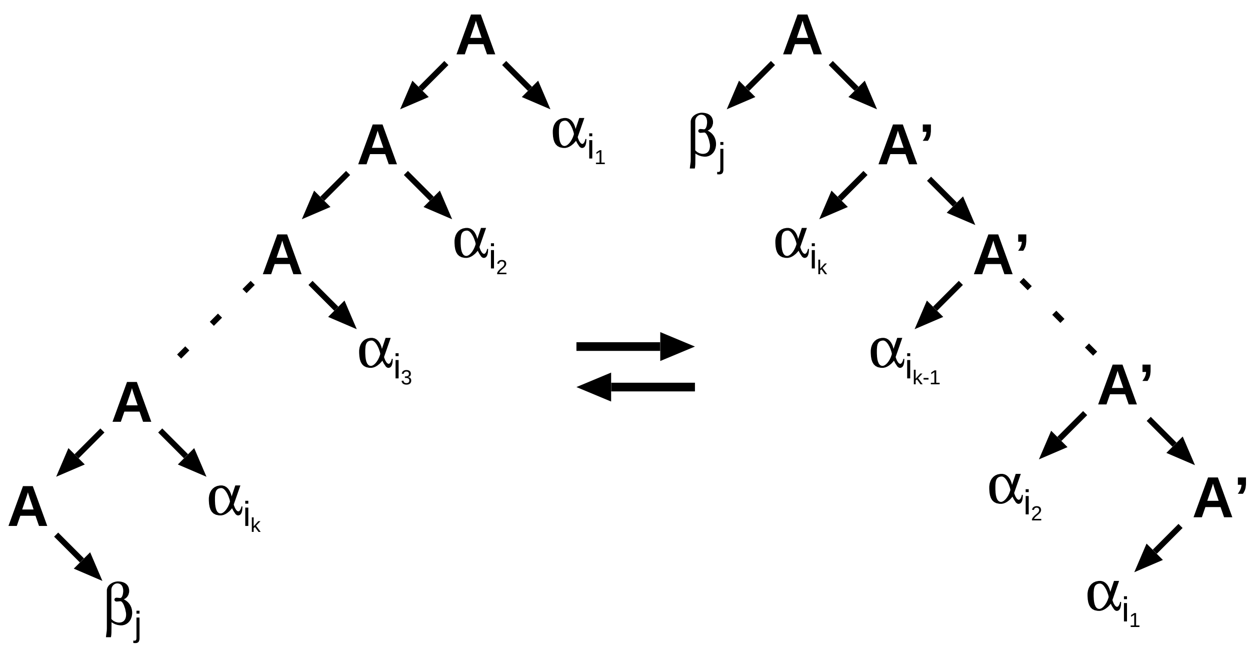
jsou všechna její A-pravidla, přičemž řetězce β_i nezačínají symbolem A. Pak gramatika G', ve které budou tato pravidla nahrazena pravidly:

$$A \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \mid \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$

$$A' \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A'$$

kde A' je nový nonterminál, je ekvivalentní s G.

Důkaz. Uvedená transformace nahrazuje pravidla rekurzivní zleva pravidly, které jsou rekurzivní zprava. Označíme-li jazyky $L_1 = \{\beta_1, \beta_2, \ldots, \beta_n\}$ a $L_2 = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, vidíme, že v G lze z nonterminálu A derivovat řetězce tvořící jazyk $L_1 L_2^*$. Právě tyto řetězce můžeme však derivovat z A také v gramatice G'. Efekt popisované transformace ilustruje následující obrázek. □



Příklad 4.15 Uvažujme gramatiku $G = (\{E, T, F\}, \{i, +, *, (,)\}, P, E)$ s pravidly:

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

$$\begin{array}{rcl}
E \to E + T \mid T & \longrightarrow & E \to T \mid TE' \\
\alpha_1 = +T, \beta_1 = T & & E' \to +T \mid\ +TE' \\
T \to T * F \mid F & \longrightarrow & T \to F \mid FT' \\
\alpha_1 = *F, \beta_1 = F & & T' \to *F \mid\ *FT' \\
 & & F \to (\ E\ ) \mid i
\end{array}$$

Odstranění nepřímé levé rekurze

- Odstranění nepřímé levé rekurze spočívá v opakovaném aplikování transformace podle věty 4.7 (rozgenerování pravidla) a transformačních vzorců pro odstranění přímé levé rekurze (věta 4.10).

Příklad 4.16 Uvažujme gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$ s pravidly:

$$S \rightarrow AB$$
$$A \rightarrow BS \mid b$$
$$B \rightarrow SA \mid a$$

Na pravidlo $B \rightarrow SA$ aplikujeme dvakrát větu 4.7:

$$B \rightarrow ABA$$
$$B \rightarrow BSBA \mid bBA$$

Na všechna B-pravidla

$$B \rightarrow BSBA \mid bBA \mid a$$

aplikujme transformaci věty 4.10:

$$B \rightarrow bBA \mid a \mid bBAB' \mid aB'$$
$$B' \rightarrow SBA \mid SBAB'$$

Normální formy bezkontextových gramatik

Chomského normální forma (CNF)

Definice 4.15 Bezkontextová gramatika $G = (N, \Sigma, P, S)$ je v Chomského normální formě, má-li každé pravidlo z P jeden z těchto tvarů:

1. $A \to BC$, kde $A, B, C \in N$
2. $A \to a$, kde $a \in \Sigma$
3. je-li $\varepsilon \in L(G)$, pak $S \to \varepsilon$ je jediné ε-pravidlo a S se nevyskytuje na pravé straně žádného přepisovacího pravidla.

Problém: Nechť $G = (N, \Sigma, P, S)$ je bezkontextová gramatika v CNF a nechť $w \in L(G)$ a $S \Rightarrow_G^p w$. Jaká je délka řetězce w?

Řešení: Označme $|w| = n$. Zřejmě platí

$$p = n + (n - 1) = 2n - 1$$

$$|w| = \frac{p + 1}{2}$$

Věta 4.11 Nechť G je bezkontextová gramatika. Pak existuje gramatika G' v Chomského normální formě taková, že $L(G') = L(G)$.

Důkaz. (Hlavní myšlenka) Gramatiku G převedeme na ekvivalentní vlastní gramatiku bez jednoduchých pravidel.

1. Pravidla tvaru (1), (2) a (3) ponecháme.
2. Pravidla tvaru $A \to X_1 X_2 \dots X_n$, kde $X_i \in (N \cup \Sigma)$ pro $i = 1, \dots, n$, $n > 2$, transformujeme na $A \to X_1' \langle X_2 X_3 \dots X_n \rangle$, kde $\langle X_2 X_3 \dots X_n \rangle$ je nový nonterminál a X_1' je nový nonterminál pokud $X_1 \in \Sigma$, nebo $X_1' = X_1$ v opačném případě.
3. Pravidla tvaru $A \to X_1 X_2$ transformujeme na pravidla $A \to X_1' X_2'$, kde X_i' je nový nonterminál pokud $X_i \in \Sigma$, nebo $X_i' = X_i$ v opačném případě pro $i \in \{1, 2\}$
4. Pro nové nonterminály tvaru $\langle X_1 X_2 \dots X_n \rangle$, $n \geq 2$, zavedeme pravidla $\langle X_1 X_2 \dots X_n \rangle \to X_1' \langle X_2 \dots X_n \rangle$ pro $n > 2$ a $\langle X_1 X_2 \rangle \to X_1' X_2'$ pro $n = 2$, kde $\langle X_2 \dots X_n \rangle$ je nový nonterminál a X_i' je nový nonterminál pokud $X_i \in \Sigma$, nebo $X_i' = X_i$ v opačném případě pro $i \in \{1, 2\}$.
5. Pro nové nonterminály tvaru X_i', kde $X_i \in \Sigma$ přidáme pravidla tvaru $X_i' \to X_i$.

$\square$

Příklad 4.17 Uvažujme gramatiku $G = (\{A, B\}, \{a, b, c\}, P, A)$ s pravidly:

$A \rightarrow BAB \mid Ba \mid bc$
$B \rightarrow AB \mid a \mid BBB$

Po aplikaci transformací (1.)-(4.) získáme CNF ve tvaru:

$A \rightarrow B\langle AB\rangle|Ba'|b'c'$
$B \rightarrow AB \mid a \mid B\langle BB\rangle$
$\langle AB\rangle \rightarrow AB$
$\langle BB\rangle \rightarrow BB$
$a' \rightarrow a$
$b' \rightarrow b$
$c' \rightarrow c$

Greibachové normální forma (GNF)

Definice 4.16 Bezkontextová gramatika $G = (N, \Sigma, P, S)$ je v Greibachové normální formě, je-li G gramatikou bez ε-pravidel a každé pravidlo z P (vyjma případného pravidla $S \to \varepsilon$) má tvar:
$A \to a\alpha$, kde $a \in \Sigma, \alpha \in N^*$

Vlastnosti bezkontextových jazyků

Pumping teorém pro BJ

Věta 6.1 Nechť L je bezkontextový jazyk. Pak existuje konstanta $k > 0$ taková, že je-li $z \in L$ a $|z| \geq k$, pak lze z napsat ve tvaru:

$$z = uvwxy, vx \neq \varepsilon, |vwx| \leq k$$

a pro všechna $i \geq 0$ je $uv^iwx^iy \in L$.

❖ Ekvivalentní formulace Pumping lemmatu (použití explicitní alternace kvantifikátorů) :

$$\begin{aligned} L \in \mathcal{L}_2 \Rightarrow \quad & \exists k > 0 : \\ & \forall z \in \Sigma^* : z \in L \wedge \ |z| \geq k \Rightarrow \\ & (\exists uvwxy \in \Sigma^* : z = uvwxy \wedge \ vx \neq \epsilon \ \wedge \ |vwx| \leq k \wedge \forall i \geq 0 : uv^iwx^iy \in L) \end{aligned}$$

Důkaz. Nechť $L = L(G)$ a nechť $G = (N, \Sigma, P, S)$ je gramatika v CNF.

1. Nejprve dokážeme implikaci:
   Jestliže $A \Rightarrow^+ w$ pro nějaké $A \in N$, $w \in \Sigma^*$, pak $|w| \leq 2^{m-2}$, kde m je počet vrcholů nejdelší cesty v odpovídajícím derivačním stromu.

   Tato implikace platí, protože $|w|$ je rovno počtu přímých předchůdců listů příslušného derivačního stromu, který je maximálně roven počtu listů plného binárního stromu, jehož všechny větve obsahují $m-1$ uzlů, což je právě 2^{m-2}.

   Skutečně:

   - Plný binární strom s větvemi o n uzlech, má 2^{n-1} listů, což se snadno ukáže indukcí:
     – Plný binární strom s (jedinou) větví o $n = 1$ uzlu, má $1 = 2^0 = 2^{n-1}$ listů.
     – Plný binární strom s větvemi délky $n = n' + 1$ uzlů, $n' \geq 1$, má $2^{n'-1} + 2^{n'-1} = 2.2^{n'-1} = 2^{1+n'-1} = 2^{n'} = 2^{n-1}$ listů.
   - Postačí tedy volit $n = m - 1$, přičemž případ neplných binárních stromů není třeba uvažovat, neboť se zajímáme o stromy s maximálním počtem listů při dané maximální délce větví.

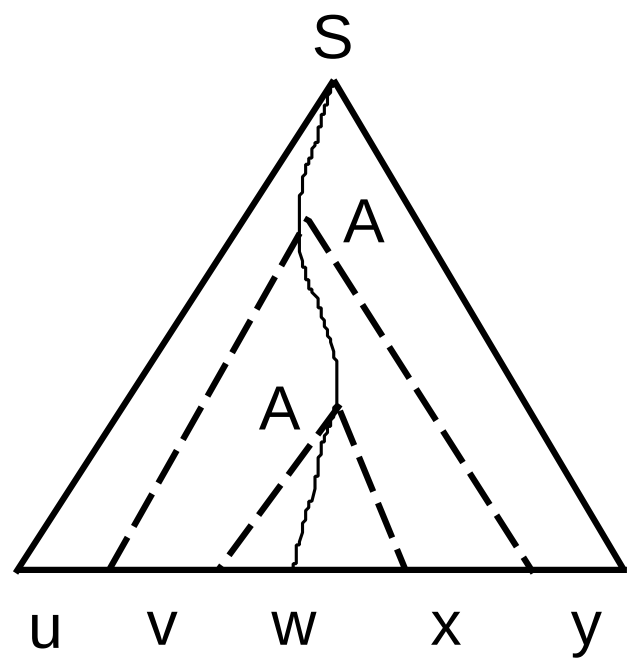
Důkaz pokračuje dále.

2. Položme $k = 2^{|N|} > 0$ a uvažujme libovolnou větu z takovou, že $|z| \geq k$.

   Označíme-li m počet vrcholů nejdelší cesty v odpovídajícím derivačním stromu, pak $2^{|N|} \leq 2^{m-2}$ a taková cesta pak obsahuje alespoň $|N| + 2$ vrcholů ($|N| + 2 \leq m$).

   Z těchto $|N| + 2$ vrcholů je jeden terminál a nutně alespoň dva jsou označeny stejným nonterminálem, řekněme A.

   Viz obrázek vpravo.



Řetězce v, x nemohou být prázdné, protože aplikované pravidlo mělo tvar $A \to BC$. Nyní uvažujme derivaci řetězce z tvaru:

$S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvwxy = z$

To pak ovšem znamená, že v G existuje rovněž derivace:

$S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvvAxxy \Rightarrow^+ uv^2wx^2y$,

protože $A \Rightarrow^+ w$, a tedy derivace $S \Rightarrow^* uv^iwx^iy$ pro libovolné $i > 0$, což je dokazované tvrzení. $\square$

Použití Pumping lemmatu

$$(L \in \mathcal{L}_2 \Rightarrow A) \Leftrightarrow (\neg A \Rightarrow L \notin \mathcal{L}_2) \quad \text{Obměna implikace}$$

$$\begin{aligned} A \equiv \quad & \exists k > 0 : \\ & \forall z \in \Sigma^* : z \in L \wedge \ |z| \geq k \Rightarrow \\ & (\exists uvwxy \in \Sigma^* : z = uvwxy \wedge \ vx \neq \epsilon \ \wedge \ |vwx| \leq k \wedge \forall i \geq 0 : uv^i wx^i y \in L) \end{aligned}$$

$$\begin{aligned} \neg A \equiv \quad & \forall k > 0 : \\ & \exists z \in \Sigma^* : z \in L \wedge \ |z| \geq k \ \wedge \\ & (\forall u, v, w, x, y \in \Sigma^* : z = uvwxy \wedge \ vx \neq \epsilon \ \wedge \ |vwx| \leq k \Rightarrow \exists i \geq 0 : uv^i wx^i y \notin L) \end{aligned}$$

- K důkazu, že jazyk L není bezkontextový stačí dokázat tvrzení $\neg A$.

Aplikace pumping teorému

Lemma 6.1 Jazyk $L = \{ww \mid w \in \{a, b, c\}^*\}$ není bezkontextovým jazykem.

Důkaz.

- Pro libovolné $k > 0$ zvolíme slovo $z = a^k b^k a^k b^k$ ($z \in L \wedge |z| \geq k$).

Poznámka: Uvažte, proč je volba slov typu $z = a^{2k}$ či $z = a^k b^{10} a^k b^{10}$ špatná (tj. důkaz pro tyto slova nelze provést).

- Dále uvažme všechny rozdělení $z = uvwxy$ kde $vx \neq \epsilon \ \wedge \ |vwx| \leq k$.

1. $vwx = a^m$: Při volbě $i \neq 1$ ve slově $uv^i wx^i y$ porušíme počty znaků a v první a druhé části slova.
2. $vwx = b^m$: Podobně jako v (1) akorát porušíme počty znaků b.
3. $vwx = a^m b^n$: Při volbě $i \neq 1$ ve slově $uv^i wx^i y$ porušíme shodu první a druhé části slova.
4. $vwx = b^m a^n$: Stejné jako v (3).

Uvědomme si, že volby $vwx = a^m b^n a^o$ a $vwx = b^m a^n b^o$ porušují podmínku $|vwx| \leq k$.

- Ukázali jsme, že pro L platí tvrzení $\neg A$ (viz. předchozí slajd) a tudíž $L \notin \mathcal{L}_2$.

Substituce jazyků

Definice 6.1 Nechť $\mathcal{L}$ je třída jazyků a nechť $L \subseteq \Sigma^*$ je jazykem třídy $\mathcal{L}$. Dále nechť $\Sigma = \{a_1, a_2, ..., a_n\}$ pro nějaké $n \in \mathbb{N}$ a nechť jazyky označené $L_{a_1}, L_{a_2}, ..., L_{a_n}$ jsou rovněž jazyky třídy $\mathcal{L}$. Říkáme, že třída $\mathcal{L}$ je uzavřena vzhledem k substituci, jestliže pro každý výběr jazyků $L, L_{a_1}, L_{a_2}, ..., L_{a_n}$ je také jazyk $\sigma_{L_{a_1}, L_{a_2}, ..., L_{a_n}}(L)$

$$\sigma_{L_{a_1}, L_{a_2}, ..., L_{a_n}}(L) = \{x_1 x_2 ... x_m \mid b_1 b_2 ... b_m \in L \land \forall i \in \{1, ..., m\} : x_i \in L_{b_i}\}$$

ve třídě $\mathcal{L}$.

Příklad 6.1 Nechť $L = \{0^n 1^n \mid n \geq 1\}$, $L_0 = \{a\}$, $L_1 = \{b^m c^m \mid m \geq 1\}$. Substitucí jazyků L_0 a L_1 do L dostaneme jazyk

$$L' = \{a^n b^{m_1} c^{m_1} b^{m_2} c^{m_2} ... b^{m_n} c^{m_n} \mid n \geq 1 \land \forall i \in \{1, ..., n\} : m_i \geq 1\}$$

Morfismus jazyků

Definice 6.2 Nechť Σ a Δ jsou abecedy a $L \subseteq \Sigma^*$ je jazyk nad abecedou Σ.

- Zobrazení $h : \Sigma^* \to \Delta^*$ nazveme morfismem nad slovy, platí-li $\forall w = a_1 a_2 ... a_n \in \Sigma^* : h(w) = h(a_1)h(a_2)...h(a_n)$.
- Morfismus jazyka $h(L)$ pak definujeme jako $h(L) = \{h(w) \mid w \in L\}$.

❖ Morfismus jazyků je zvláštní případ substituce, kde každý substituovaný jazyk má právě jednu větu.

Uzavřenost vůči substituci

Věta 6.2 Třída bezkontextových jazyků je uzavřena vůči substituci.

Důkaz.

- Ve shodě s definicí substituce nechť $\Sigma = \{a_1, a_2, ..., a_n\}$ je abeceda bezkontextového jazyka L a L_a pro $a \in \Sigma$ libovolné bezkontextové jazyky. Nechť $G = (N, \Sigma, P, S)$ a $G_a = (N_a, \Sigma_a, P_a, S_a)$ pro $a \in \Sigma$ jsou gramatiky, pro které $L = L(G)$ a $L_a = L(G_a)$ pro $a \in \Sigma$.
- Předpokládejme, že $N \cap N_a = \emptyset$ a $N_a \cap N_b = \emptyset$ pro každé $a, b \in \Sigma$, $a \neq b$. Sestrojme gramatiku $G' = (N', \Sigma', P', S)$ takto:
  1. $N' = N \cup \bigcup_{a\in\Sigma} N_a$.
  2. $\Sigma' = \bigcup_{a\in\Sigma} \Sigma_a$.
  3. Nechť h je morfismus na $N \cup \Sigma$ takový, že
     - $h(A) = A$ pro $A \in N$ a
     - $h(a) = S_a$ pro $a \in \Sigma$

     a nechť $P' = \{A \rightarrow h(\alpha) \mid (A \rightarrow \alpha) \in P\} \cup \bigcup_{a\in\Sigma} P_a$.
- Uvažujme libovolnou větu $a_{i_1} a_{i_2} ... a_{i_m} \in L$ a věty $x_j \in L_{a_j}$, $1 \leq j \leq m$. Pak $S \overset{*}{\underset{G'}{\Rightarrow}} S_{a_{i_1}} S_{a_{i_2}} .... S_{a_{i_m}} \overset{*}{\underset{G'}{\Rightarrow}} x_1 S_{a_{i_2}} .... S_{a_{i_m}} \overset{*}{\underset{G'}{\Rightarrow}} ... \overset{*}{\underset{G'}{\Rightarrow}} x_1 x_2 ... x_m$ a tedy $L' \subseteq L(G')$. Podobně $L(G') \subseteq L'$. $\square$

Důkaz uzavřenosti $\mathcal{L}_2$ jazyků

Nechť L_a a L_b jsou bezkontextové jazyky.

1. Uzavřenost vůči $\cup$ plyne ze substituce L_a, L_b do jazyka $\{a, b\}$.
2. Uzavřenost vůči $.$ plyne ze substituce L_a, L_b do jazyka $\{ab\}$.
3. Uzavřenost vůči $*$ plyne ze substituce L_a do jazyka $\{a\}^*$.
4. Uzavřenost vůči $+$ plyne ze substituce L_a do jazyka $\{a\}^+$.
5. Nechť h je daný morfismus a $L'_a = \{h(a)\}$ pro $a \in \Sigma$. Substitucí jazyků L'_a do jazyka L získáme jazyk $h(L)$.

Věta 6.3 Bezkontextové jazyky jsou uzavřeny vzhledem k průniku s regulárními jazyky.

Důkaz. Snadno zkonstruujeme ZA přijímající příslušný průnik – konstruujeme průnik na konečném řízení, zásobníkové operace zůstávají. □

Neuzavřenost $\mathcal{L}_2$ vůči průniku a doplňku

Věta 6.4 Bezkontextové jazyky *nejsou* uzavřeny vůči průniku a doplňku.

Důkaz.

1. Neuzavřenost vůči $\cap$:
   Uvažujme jazyky $L_1 = \{a^m b^m c^n \mid n, m \geq 1\}$ a $L_2 = \{a^m b^n c^n \mid m, n \geq 1\}$, které jsou oba bezkontextové. Ovšem $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$, což není bezkontextový jazyk (lze ukázat např. pomocí Pumping lemmatu).
2. Neuzavřenost vůči doplňku: Předpokládejme, že bezk. jazyky jsou uzavřeny vůči doplňku. Z De Morganových zákonů (a z uzavřenosti vůči sjednocení) pak ovšem plyne uzavřenost vůči průniku $L_1 \cap L_2 = \overline{\overline{L_1 \cap L_2}} = \overline{\overline{L_1} \cup \overline{L_2}}$, což je spor.

$\square$

Rozhodnutelné problémy pro $\mathcal{L}_2$

Věta 6.5 Následující problémy jsou rozhodnutelné, tj. jsou algoritmicky řešitelné:

1. problém neprázdnosti jazyka $L(G)$ pro libovolnou bezkontextovou gramatiku G,
2. problém příslušnosti řetězce $w \in \Sigma^*$ do jazyka $L(G)$ pro libovolnou bezkontextovou gramatiku G,
3. problém konečnosti jazyka $L(G)$ pro libovolnou bezkontextovou gramatiku G.

Důkaz.

1. K rozhodování neprázdnosti lze využít algoritmus iterativně určující množinu N_t nonterminálů generujících terminální řetězce uvedený v předchozí přednášce. Pak $L(G) \neq \emptyset \Leftrightarrow S \in N_t$.
2. U problému příslušnosti řetězce můžeme např. určit průnik NZA s KA přijímajícím právě řetězec w a pak ověřit neprázdnost.

Důkaz pokračuje dále.

Pokračování důkazu.

3. Problém konečnosti můžeme rozhodovat na základě platnosti Pumping lemma pro CFL:
   - Dle Pumping lemma pro bezkontextové jazyky existuje pro každý bezkontextový jazyk L konstanta $k \in \mathbb{N}$ taková, že každou větu $w \in L$, $|w| \geq k$, můžeme rozepsat jako $uvwxy$, kde $vx \neq \varepsilon$ a $|vwx| \leq k$, a $\forall i \in \mathbb{N} : uv^iwx^iy \in L$.
   - Pro testování konečnosti tedy postačí ověřit, že žádný řetězec ze Σ^* o délce mezi k a $2k-1$ nepatří do daného jazyka:
     - Pokud takový řetězec existuje, může být „napumpován“ a dostáváme nekonečně mnoho řetězců patřících do daného jazyka.
     - Jestliže takový řetězec neexistuje, $k-1$ je horní limit délky řetězců L.
     - Pokud by existoval řetězec délky $2k$ nebo větší patřící do L, můžeme v něm podle Pumping lemma najít vwx a vypustit vx. Vzhledem k tomu, že $0 < |vx| \leq k$, postupným opakováním vypouštění bychom se dostali k nutné existenci řetězce z L o délce mezi k a $2k-1$.
   - K určení konstanty k postačí reprezentovat L pomocí bezkontextové gramatiky v CNF s n nonterminály a zvolit $k = 2^n$ (viz důkaz Pumping lemma).

$\square$

Nerozhodnutelné problémy pro $\mathcal{L}_2$

Věta 6.6 Následující problémy jsou nerozhodnutelné, tj. nejsou algoritmicky řešitelné:

1. problém ekvivalence jazyků bezkontextových gramatik, tj. otázka, zda $L(G_1) = L(G_2)$ pro dvě bezkontextové gramatiky G_1, G_2,
2. problém inkluze jazyků bezkontextových gramatik, tj. otázka, zda $L(G_1) \subseteq L(G_2)$ pro dvě bezkontextové gramatiky G_1, G_2.

Důkaz. Důkaz lze vést pomocí techniky redukce. Více v pozdějších přednáškách o nerozhodnutelnosti. $\square$

Regularita

Definice 6.3 Bezkontextová gramatika $G = (N, \Sigma, P, S)$ má vlastnost sebevložení, jestliže existují $A \in N$ a $u, v \in \Sigma^+$ takové, že $A \Rightarrow^+ uAv$ a A není zbytečný nonterminál. Bezkontextový jazyk má vlastnost sebevložení, jestliže každá gramatika, která jej generuje, má vlastnost sebevložení.

Věta 6.7 Bezkontextový jazyk má vlastnost sebevložení právě tehdy, když není regulární.

Důkaz. Můžeme využít GNF – blíže viz doporučená literatura. □

❖ Problém, zda daná bezkontextová gramatika generuje regulární jazyk, není algoritmicky rozhodnutelný.

Deterministické zásobníkové automaty

Deterministický zásobníkový automat

Definice 6.4 Zásobníkový automat $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ nazýváme deterministický zásobníkový automat (DZA), jestliže pro každé $q \in Q$ a $z \in \Gamma$ platí buď

- $\forall a \in \Sigma : |\delta(q, a, z)| \leq 1 \wedge \delta(q, \varepsilon, z) = \emptyset$, nebo
- $\forall a \in \Sigma : \delta(q, a, z) = \emptyset \wedge |\delta(q, \varepsilon, z)| \leq 1$.

Definice 6.5 Nechť $L = L(P)$, kde P je deterministický zásobníkový automat. Jazyk L se pak nazývá deterministickým bezkontextovým jazykem.

Příklad 6.2 Uvažujme gramatiku $G = (\{X, Y\}, \{a, b, c\}, P, X)$ s pravidly:

$$\begin{array}{lcl} X & \longrightarrow & aXa \mid cYc \mid b \\ Y & \longrightarrow & aYbX \mid c \end{array}$$

Jedná se o LL(1) gramatiku a tudíž můžeme sestrojit DZA
$P = (\{q\}, \{a, b, c\}, \{X, Y, a, b, c\}, \delta, q, X, \emptyset)$ takový, že $L(G) = L(P)$ a provádí $LL(1)$ analýzu :

$$\begin{array}{lll} \delta: & \delta(q, a, X) = (q, Xa) & \delta(q, c, Y) = (q, \varepsilon) \\ & \delta(q, b, X) = (q, \varepsilon) & \delta(q, a, a) = (q, \varepsilon) \\ & \delta(q, c, X) = (q, Yc) & \delta(q, b, b) = (q, \varepsilon) \\ & \delta(q, a, Y) = (q, YbX) & \delta(q, c, c) = (q, \varepsilon) \end{array}$$

Skutečně, např. derivaci $X \Rightarrow aXa \Rightarrow aba$ odpovídá přijímající posloupnost konfigurací $(a, aba, X) \vdash (q, ba, Xa) \vdash (q, a, a) \vdash (q, \varepsilon, \varepsilon)$.

Neekvivalence NZA a DZA

Věta 6.8 DZA mají striktně menší vyjadřovací sílu než NZA.

Důkaz. (idea) Bezkontextový jazyk $L = \{ww^R \mid w \in \Sigma^+\}$ nelze přijímat žádným DZA. Neformálně řečeno, DZA nemá možnost uhádnout, kdy končí w a začíná w^R. $\square$

❖ *Poznámka:* Jiná možnost důkazu věty je přes následně uvedenou uzavřenost jazyků DZA vůči doplňku a přes uvážení, že $\overline{\{a^n b^n c^n | n \geq 1\}}$ je bezkontextový jazyk.

❖ Problém, zda daný bezkontextový jazyk je jazykem nějakého DZA, není obecně rozhodnutelný (podobně jako není rozhodnutelná víceznačnost).

Vlastnosti jazyků DZA

Věta 6.9 Jazyky DZA jsou uzavřeny vůči:

1. průniku s regulárními jazyky,
2. doplňku.

Důkaz. (idea) Bod 1 dokážeme podobně jako u NZA. U bodu 2 postupujeme podobně jako u DKA – použijeme záměnu koncových a nekoncových stavů, musíme ale navíc řešit dva okruhy problémů: (a) DZA nemusí vždy dočíst vstupní slovo až do konce (buď se dostane do konfigurace, z níž nemůže pokračovat, nebo cyklí přes ε-kroky) a (b) DZA slovo dočte do konce, ale pak ještě provede posloupnost ε-kroků jdoucích přes koncové i nekoncové stavy. Popis řešení těchto problémů je možno nalézt v doporučené literatuře.□

Věta 6.10 Jazyky DZA *nejsou* uzavřeny vůči:

1. průniku,
2. sjednocení.

Důkaz. U bodu 1 použijeme stejný postup jako u NZA (uvědomíme si, že $\{a^m b^m c^n \mid n, m \geq 1\}$ a $\{a^m b^n c^n \mid n, m \geq 1\}$ lze přijímat DZA). Bod 2 plyne z De Morganových zákonů. □

Věta 6.11 Jazyky DZA *nejsou* uzavřeny vůči:

1. konkatenaci,
2. iteraci.

* *Důkaz.* (idea) Vyjdeme z toho, že zatímco jazyky $L_1 = \{a^m b^m c^n \mid m, n \geq 1\}$ a $L_2 = \{a^m b^n c^n \mid m, n \geq 1\}$ jsou deterministické bezkontextové, jazyk $L_1 \cup L_2$ ne. (Intuitivně DZA nemůže odhadnout, zda má kontrolovat první nebo druhou rovnost, a tedy, zda na zásobník ukládat symboly a nebo b.)

1. Neuzavřenost vůči konkatenaci. Jazyk $L_3 = 0L_1 \cup L_2$ je zřejmě deterministický bezkontextový. Jazyk 0^* je také deterministický bezkontextový (dokonce regulární), ovšem není těžké nahlédnout, že $0^* L_3$ není deterministický bezkontextový. Stačí uvážit, že $0a^*b^*c^*$ je deterministický bezkontextový (dokonce regulární) jazyk a $0^* L_3 \cap 0a^*b^*c^* = 0L_1 \cup 0L_2 = 0(L_1 \cup L_2)$.
2. Neuzavřenost vůči iteraci. Uvážíme $(\{0\} \cup L_3)^* \cap 0a^+b^+c^+ = 0(L_1 \cup L_2)$.

$\square$ *

Některé další zajímavé vlastnosti bezkontextových jazyků

Teorém Chomského a Schützenbergera

❖ Tento teorém postihuje úzkou vazbu bezkontextových jazyků na závorkování.

Definice 6.6 Označme ZAV_n pro $n \geq 0$ jazyky sestávající ze všech vyvážených řetězců závorek n typů. Tyto jazyky – označované též jako Dyckovy jazyky – jsou generovány gramatikami s pravidly tvaru:

$$S \rightarrow [^1 \; S \;]^1 \mid [^2 \; S \;]^2 \mid \ldots \mid [^n \; S \;]^n \mid SS \mid \varepsilon$$

Věta 6.12 (Chomsky-Schützenberger) Každý bezkontextový jazyk je morfismem průniku nějakého jazyka závorek a nějaké regulární množiny. Jinými slovy, pro každý $L \in \mathcal{L}_2$ existují $n \geq 0$, regulární množina R a morfismus h takový, že

$$L = h(ZAV_n \cap R)$$

Důkaz. Viz doporučená literatura. □

Parikhův teorém

❖ Tento teorém opět postihuje strukturu bezkontextových jazyků – zabývá se tím, co dostaneme, pokud ve větách odhlédneme od pořadí jednotlivých symbolů a zkoumáme pouze počet jejich opakování (tj. zahrneme vlastně libovolné přeházení znaků v řetězci).

Definice 6.7 Mějme abecedu $\Sigma = \{a_1, ..., a_k\}$. Parikhova funkce je funkce $\psi : \Sigma^* \to \mathbb{N}^k$ definovaná pro $w \in \Sigma^*$ jako $\psi(w) = (\#a_1(w), ..., \#a_k(w))$, kde $\#a_i(w)$ udává počet výskytů symbolu a_i ve w.

Definice 6.8 Podmnožinu množiny vektorů $\mathbb{N}^k$ nazveme lineární množinou, je-li dána bází $u_0 \in \mathbb{N}^k$ a periodami $u_1, ..., u_m \in \mathbb{N}^k$ jako $\{u_0 + a_1 u_1 + ... + a_m u_m \mid a_1, ..., a_m \in \mathbb{N}\}$. Podmnožinu $\mathbb{N}^k$ nazveme semilineární množinou, je-li sjednocením konečného počtu lineárních množin.

Věta 6.13 (Parikh) Pro libovolný bezkontextový jazyk L, $\psi(L)$ je semilineární množina.

Důkaz. Viz doporučená literatura. $\square$

- Ke každé semilineární množině S můžeme najít regulární množinu $R \subseteq \Sigma^*$ takovou, že $\psi(R) = S$.

- Proto bývá Parikhův teorém někdy formulován takto: Komutativní obraz každého bezkontextového jazyka odpovídá nějakému regulárnímu jazyku.

- Semilineární množiny se navíc dají reprezentovat konečnými automaty přímo jako množiny číselných vektorů v binárním kódování (tzv. *NDDs – number decision diagrams*).

Turingovy stroje

Churchova teze

❖ ***Churchova (Church-Turingova) teze**: Turingovy stroje (a jim ekvivalentní systémy) definují svou výpočetní silou to, co intuitivně považujeme za efektivně vyčíslitelné.*

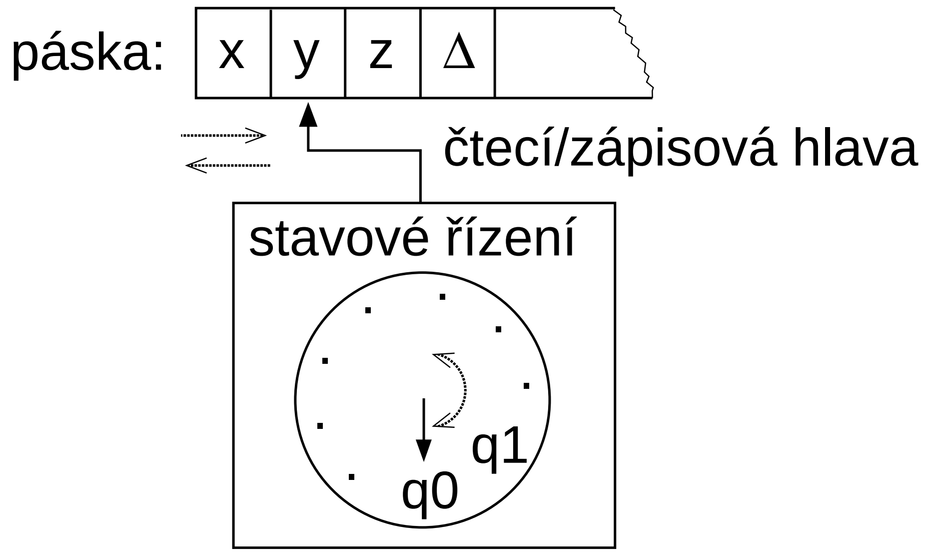
❖ Churchova teze není teorém, nemůžeme formálně dokazovat, že něco odpovídá našim intuitivním představám, nicméně je podpořena řadou argumentů:

- Turingovy stroje jsou velmi robustní – uvidíme, že jejich různé úpravy nemění jejich výpočetní sílu (determinismus x nedeterminismus, počet pásek, ...).
- Byla navržena řada zcela odlišných výpočetních modelů (λ-kalkulus, parciálně rekurzívní funkce, Minského stroje, ...), jejichž síla odpovídá Turingovým strojům.
- Není znám žádný výpočetní proces, který bychom označili za efektivně vyčíslitelný a který by nebylo možné realizovat na Turingově stroji.[a] □

---

[a] Existují formalizované výpočetní procesy, realizovatelné např. na TS s orákulem (nápovědou) rozhodujícím atomicky nějaký Turingovsky nerozhodnutelný problém (např. problém zastavení), které ale nepovažujeme za efektivní výpočetní procesy.

Turingův stroj



Definice 7.1 Turingův stroj (TS) je šestice tvaru $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde:

- Q je konečná množina vnitřních (řídicích) stavů,
- Σ je konečná množina symbolů nazývaná vstupní abeceda, $\Delta \notin \Sigma$,
- Γ je konečná množina symbolů, $\Sigma \subset \Gamma$, $\Delta \in \Gamma$, nazývaná pásková abeceda,
- parciální funkce $\delta : (Q \setminus \{q_F\}) \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$, kde $L, R \notin \Gamma$, je přechodová funkce,
- q_0 je počáteční stav, $q_0 \in Q$ a
- q_F je koncový stav, $q_F \in Q$.

Konfigurace Turingova stroje

❖ Symbol Δ značí tzv. blank (prázdný symbol), který se vyskytuje na místech pásky, která nebyla ještě použita (může ale být na pásku zapsán i později).

❖ Konfigurace pásky je dvojice sestávající z nekonečného řetězce reprezentujícího obsah pásky a pozice hlavy na tomto řetězci – přesněji jde o prvek množiny $\{\gamma\Delta^\omega \mid \gamma \in \Gamma^*\} \times \mathbb{N}$. [a]

❖ Konfiguraci pásky zapisujeme jako $\Delta xyz\underline{z}\Delta x\Delta\Delta\ldots$ (podtržení značí pozici hlavy).

❖ Konfigurace stroje je pak dána stavem řízení a konfigurací pásky – formálně se jedná o prvek množiny $Q \times \{\gamma\Delta^\omega \mid \gamma \in \Gamma^*\} \times \mathbb{N}$.

❖ I když je páska nekonečná, dokážeme ji jednoznačně specifikovat konečným řetězcem popisující obsah neprázdných políček (těch je vždy konečně mnoho). Proto můžeme zjednodušit zápis konfigurace a specifikovat pouze neprázdné políčka pásky $\gamma \in \Gamma^*$.

---

[a] Pro libovolnou abecedu Σ je Σ^ω množina všech *nekonečných* řetězců nad Σ, tj. nekonečných posloupností symbolů ze Σ. Pro připomenutí: Σ^* je množina všech *konečných* řetězců nad Σ.

Přechodová relace TS

- Pro libovolný řetězec $\gamma \in \Gamma^\omega$ a číslo $n \in \mathbb{N}$ označme γ_n n-tý symbol daného řetězce a označme $s_b^n(\gamma)$ řetězec, který vznikne z γ záměnou γ_n za b.

- Krok výpočtu TS M definujeme jako *nejmenší* binární relaci $\underset{M}{\vdash}$ takovou, že $\forall q_1, q_2 \in Q \; \forall \gamma \in \Gamma^\omega \; \forall n \in \mathbb{N} \; \forall b \in \Gamma$:
  - $(q_1, \gamma, n) \underset{M}{\vdash} (q_2, \gamma, n+1)$ pro $\delta(q_1, \gamma_n) = (q_2, R)$ – operace posuvu doprava při γ_n pod hlavou,
  - $(q_1, \gamma, n) \underset{M}{\vdash} (q_2, \gamma, n-1)$ pro $\delta(q_1, \gamma_n) = (q_2, L)$ a $n > 0$ – operace posuvu doleva při γ_n pod hlavou a
  - $(q_1, \gamma, n) \underset{M}{\vdash} (q_2, s_b^n(\gamma), n)$ pro $\delta(q_1, \gamma_n) = (q_2, b)$ – operace zápisu b při γ_n pod hlavou.

Výpočet TS

❖ Výpočet TS M začínající z konfigurace K_0 je posloupnost konfigurací K_0, K_1, K_2, ...,

- ve které $K_i \underset{M}{\vdash} K_{i+1}$ pro všechna $i \geq 0$ taková, že K_{i+1} je v dané posloupnosti, a
- která je buď
  - nekonečná, a nebo
  - konečná s koncovou konfigurací (q, γ, n), přičemž rozlišujeme následující typy zastavení TS:
    1. normální – přechodem do koncového stavu, tj. $q = q_F$, a
    2. abnormální, kdy $q \neq q_F$ a:
       (a) pro (q, γ_n) není δ definována, nebo
       (b) hlava je na nejlevější pozici pásky a dojde k posunu doleva, tj. $n = 0$ a $\delta(q, \gamma_n) = (q', L)$ pro nějaké $q' \in Q$.

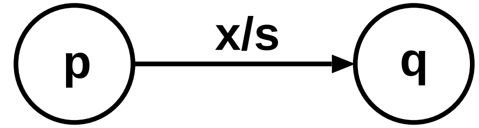
Poznámka – alternativní definice TS

❖ Používají se i některé alternativní definice TS, u kterých se dá snadno ukázat vzájemná převoditelnost:

- namísto jediného q_F je povolena množina koncových stavů,
- namísto q_F je zavedena dvojice q_{accept} a q_{reject},
- na prvním políčku pásky je „napevno“ zapsán symbol konce pásky, z něhož není možný posun doleva,
- při zavedení obou předchozích bodů je δ obvykle definovaná jako totální funkce,
- přepis a posuv hlavy jsou spojeny do jedné operace
- apod.

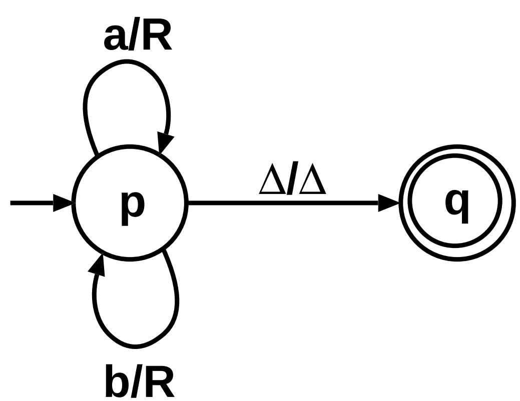
Grafická reprezentace TS

- Grafická reprezentace přechodu (x – co se čte, s – zápis/L/R):



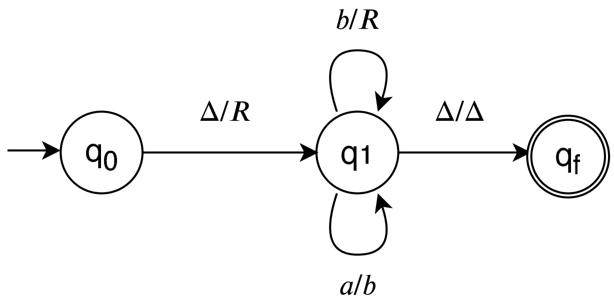
- Grafická reprezentace počátečního a koncového stavu:

Příklad 7.1 TS, který posouvá hlavu doprava na první Δ počínaje aktuální pozicí (např. $\Delta\underline{a}ab\Delta... \longrightarrow \Delta aab\underline{\Delta}...$):

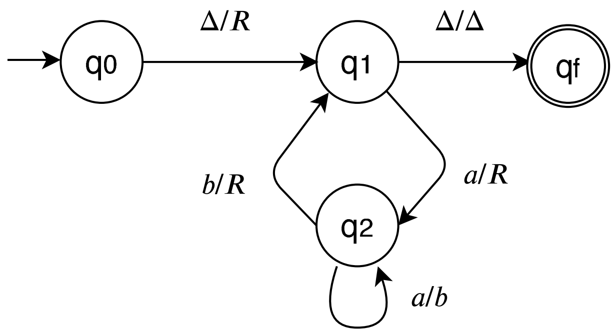


Příklady TS

Příklad 7.2 TS, který modifikuje pásku ve tvaru $\underline{\Delta}a^n\Delta$ pro $n > 0$ na $\Delta b^n\underline{\Delta}$

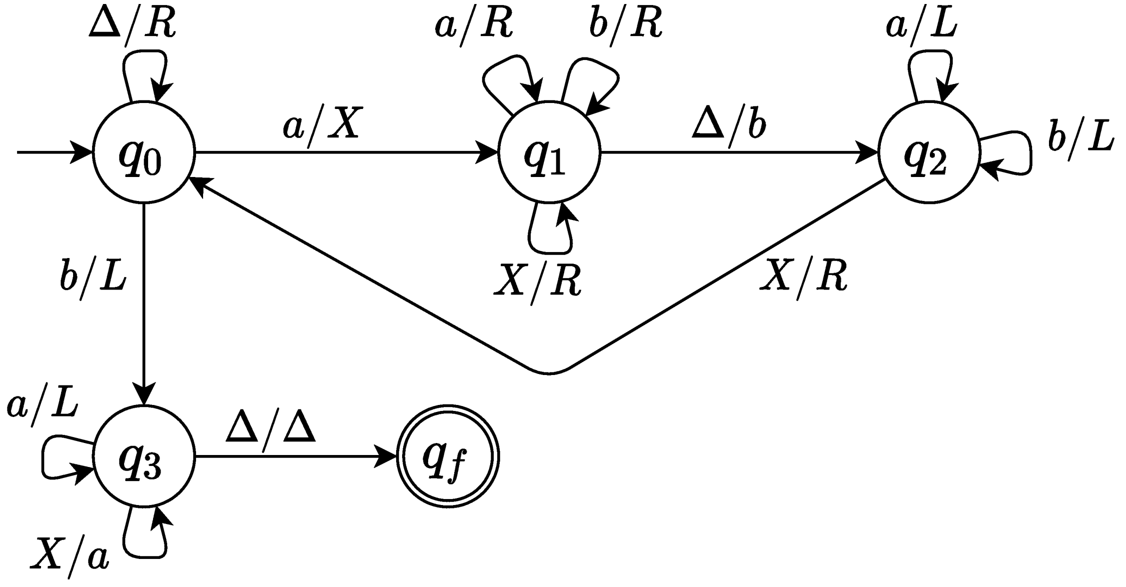


Příklad 7.3 TS, který modifikuje pásku ve tvaru $\underline{\Delta}a^{2n}\Delta$ pro $n > 0$ na $\Delta(ab)^n\underline{\Delta}$



Příklady TS

Příklad 7.4 TS, který modifikuje pásku ve tvaru $\underline{\Delta}a^n\Delta$ pro $n > 0$ na $\underline{\Delta}a^nb^n\Delta$



Poznámka 7.1

- Uvědomme si, že na TS lze také nahlížet jako na výpočetní mechanismy implementující funkce $\Sigma^* \rightarrow \Gamma^*$ tím, že transformují počáteční neblankový prefix své pásky na jiný neblankový prefix při přechodu do koncového stavu.
- Vzhledem k tomu, že TS nemusí každý svůj vstup přijmout, jsou funkce jimi implementované obecně parciální.
- Blíže se budeme vyčíslováním funkcí TS zabývat dále.

Turingovy stroje jako akceptory jazyků

Jazyk přijímaný TS

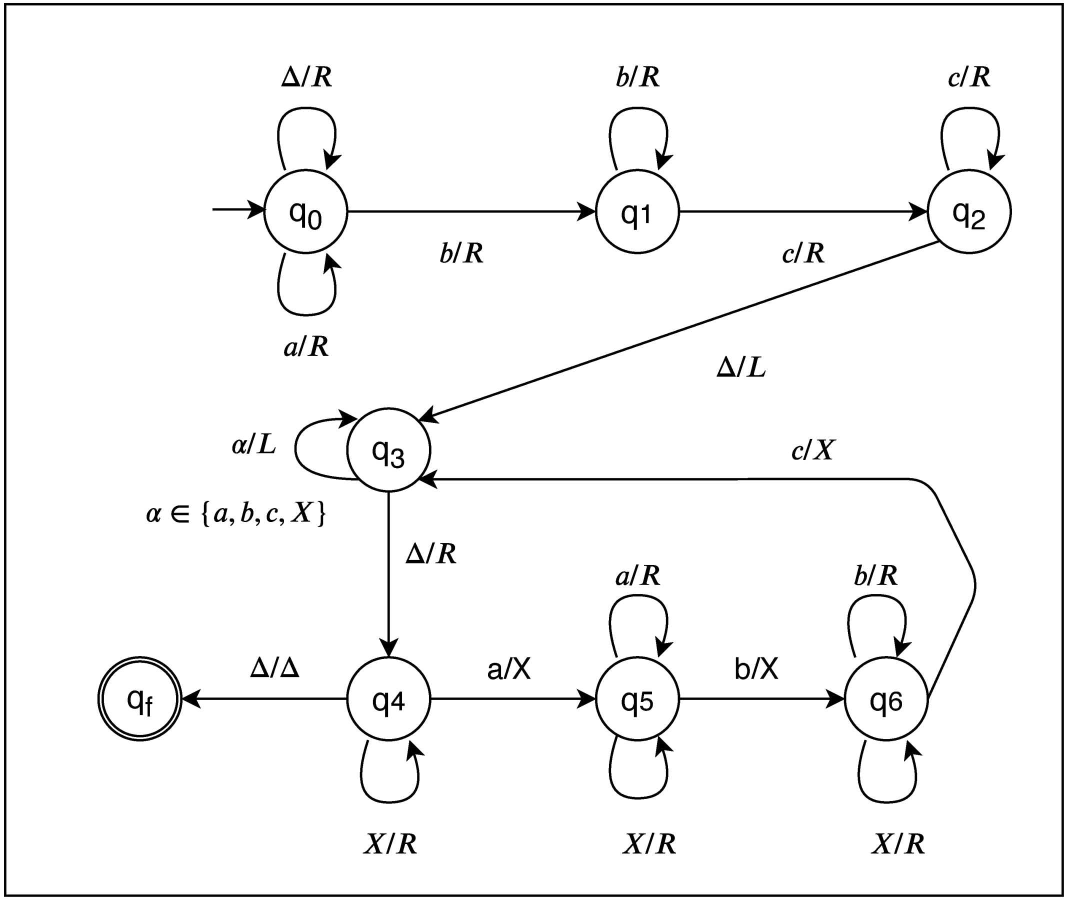
Definice 7.2

1. Řetězec $w \in \Sigma^*$ je přijat TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, jestliže M při aktivaci z počáteční konfigurace pásky $\underline{\Delta} w \Delta ...$ a počátečního stavu q_0 zastaví přechodem do koncového stavu q_F, tj. $(q_0, \Delta w \Delta^\omega, 0) \overset{*}{\underset{M}{\vdash}} (q_F, \gamma, n)$ pro nějaké $\gamma \in \Gamma^*$ a $n \in \mathbb{N}$.
2. Množinu $L(M) = \{w \mid w$ je přijat TS $M\} \subseteq \Sigma^*$ nazýváme jazyk přijímaný TS M.

❖ Alternativně můžeme přijetí řetězce TS definovat tak, že TS začíná s konfigurací pásky $\underline{\Delta} w \Delta ...$ a zastaví s konfigurací pásky $\underline{\Delta} Y \Delta ...$, $Y \in \Gamma \setminus \Sigma$, ($Y$ *značí Yes*).

TS pro nebezkontextový jazyk

Příklad 7.5 TS, který přijímá jazyk $L = \{a^n b^n c^n \mid n > 0\}$



Vícepáskové Turingovy stroje

Vícepáskové TS

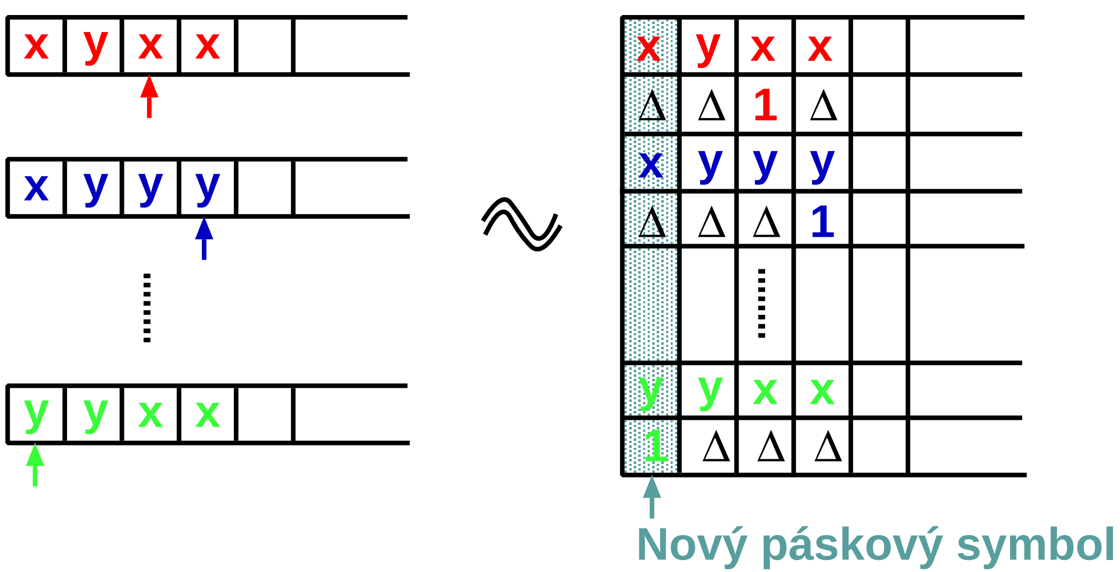
❖ Uvažujme TS, který má k pásek s páskovými abecedami $\Gamma_1, \Gamma_2, ..., \Gamma_k$ a k *odpovídajících hlav* s přechodovou funkcí tvaru

$$\delta : (Q \setminus \{q_F\}) \times \Gamma_1 \times \Gamma_2 \times ... \times \Gamma_k \longrightarrow Q \times \Gamma'_1 \times \Gamma'_2 \times ... \times \Gamma'_k$$

kde $\Gamma'_i = \Gamma_i \cup \{L, R\}$.

Věta 7.1 Pro každý k-páskový TS M existuje jednopáskový TS M' takový, že $L(M) = L(M')$.

Důkaz. (idea)



Důkaz pokračuje dále.

Pokračování důkazu. Důkaz provedeme tak, že ukážeme algoritmus převodu na ekvivalentní jednopáskový TS:

- Předpokládáme, že přijímaný řetězec je u k-páskového stroje na počátku zapsán na první pásce, všechny ostatní pásky jsou prázdné a všechny hlavy jsou na nejlevější pozici.
- Původních k pásek simulujeme rozšířením páskové abecedy o $2k$-tice, v nichž vždy i-tá složka pro liché i reprezentuje obsah $(\frac{i+1}{2})$-ní pásky a na pozici $i+1$ je Δ nebo 1 podle toho, zda se na ní nachází příslušná hlava či nikoliv.
- Počet načítaných kombinací symbolů v původním automatu je konečný a tudíž si výše uvedené rozšíření můžeme skutečně dovolit.
- Při simulaci k-páskového TS pak nejprve převedeme původní obsah první pásky na ekvivalentní obsah zakódovaný v $2k$-ticích a pak každý krok simulujeme několika kroky.
- Při rozhodování o dalším kroku stroj M' projde celou pásku a zapamatuje si ve svém stavu uspořádanou k-tici aktuálně čtených symbolů. Stavy jsou tedy $(k+1)$-tice, kde první složka je stav původního TS, a dovolují tedy stroji M' korektně simulovat původní stroj.
- Po přečtení pásky a aktualizace stavu M' přemístí hlavu na speciální pozici nalevo od užitečného obsahu pásky.

Důkaz pokračuje dále.

Pokračování důkazu.

- Po rozhodnutí o dalším kroku se posunujeme postupně doprava na pozice, která se mají modifikovat, provedeme příslušnou změnu a vrátíme se zpět doleva.
- Za nový aktuální stav považujeme $(k+1)$-tici danou novým stavem simulovaného TS a novou k-tici reprezentující modifikace na simulovaných páskách.
- Uvědomme si, že jeden krok simulace vícepáskového stroje vyžaduje 2 průchody páskou – více viz přednáška o složitosti.
- Navíc je nutné korektně simulovat „přepadnutí" hlavy na kterékoliv pásce a převod dosud nevyužitých míst pásky s Δ na odpovídající $2k$-tici blank symbolů.
- Při řádné formalizaci popsaného algoritmu pak není těžké ukázat, že výsledný TS skutečně simuluje původní TS.

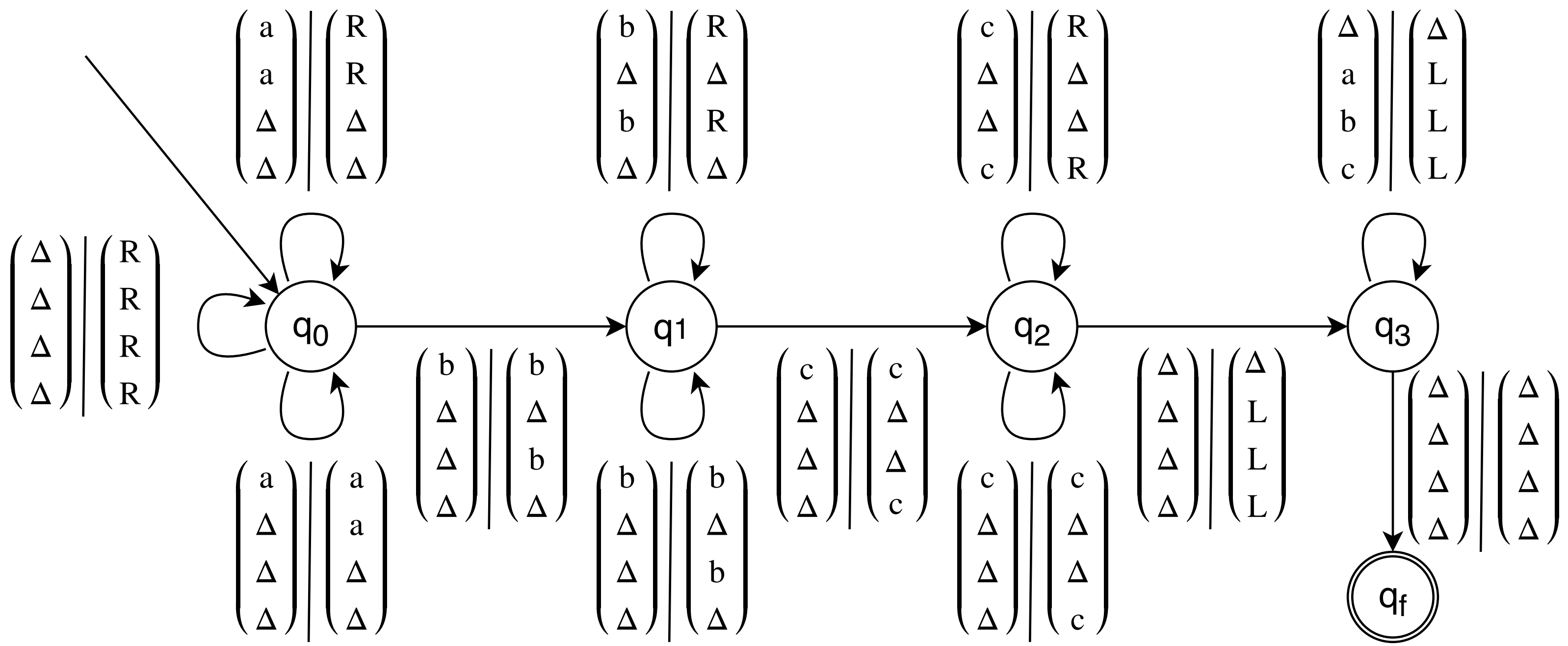
$\square$

❖ Závěr:

Zvětšení paměťových možností TS nerozšiřuje jejich schopnosti přijímat jazyky!

Příklad vícepáskového TS

Příklad 7.6 4-páskový TS, který přijímá jazyk $L = \{a^n b^n c^n \mid n > 0\}$



Nedeterministické Turingovy stroje

Nedeterministické TS

Definice 7.3 Nedeterministický TS je šestice $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde význam jednotlivých složek je shodný s deterministickým TS až na δ, jež má tvar:

$$\delta : (Q \setminus \{q_F\}) \times \Gamma \longrightarrow 2^{Q \times (\Gamma \cup \{L,R\})}$$

Definice 7.4 Jazyk $L(M)$ NTS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je množina řetězců $w \in \Sigma^*$ takových, že M při aktivaci z q_0 při počátečním obsahu pásky $\underline{\Delta} w \Delta \ldots$ **může** zastavit přechodem do q_F.

Věta 7.2 Pro každý NTS M existuje DTS M' takový, že $L(M) = L(M')$.

Důkaz. (idea)

- NTS M budeme simulovat třípáskovým DTS. Význam jednotlivých pásek tohoto stroje je následující:
  - Páska 1 obsahuje vstupní řetězec.
  - Páska 2 je pracovní páska. Obsahuje kopii pásky 1 ohraničenou vhodnými speciálními značkami. Po neúspěšném pokusu o přijetí je její obsah smazán a obnoven z první pásky.
  - Páska 3 obsahuje kódovanou volbu posloupností přechodů; při neúspěchu bude její obsah nahrazen jinou posloupností.

Důkaz pokračuje dále.

Pokračování důkazu.

- Zvolená posloupnost přechodů je kódována posloupností čísel přiřazených přechodům simulovaného stoje.
- Jednotlivé posloupnosti přechodů na pásce 3 generujme pomocí BFS: nejprve všechny výpočty délky 1, potom všechny výpočty délky 2 atd.
- Vlastní simulace probíhá takto:
  1. Okopíruj obsah pásky 1 na pásku 2.
  2. Generuj příští posloupnost přechodů na pásce 3.
  3. Simuluj provedení posloupnosti z pásky 3 na obsahu pásky 2.
  4. Vede-li zkoumaná posloupnost do q_F simulovaného stroje, zastav – vstupní řetězec je přijat. V opačném případě smaž pásku 2 a vrať se k bodu 1.
- Není obtížné nahlédnout, že jazyk takto vytvořeného stroje odpovídá jazyku původního NTS.

$\square$

❖ Závěr:

Zavedením nedeterminismu do TS se nezvyšují jejich schopnosti přijímat jazyky!

Univerzální Turingovy stroje

Kódování TS

❖ Kódovací systém pro TS zahrnuje (1) kódování stavů (tak, aby byly odlišeny všechny stavy včetně q_0 a q_F), (2) symbolů z Γ a (3) přechodové funkce δ.

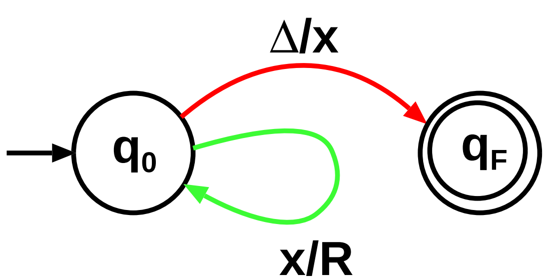
❖ Kódování stavů: Množinu stavů Q uspořádáme do posloupnosti q_0, q_F, q, p, ..., t. Stav q_j zakódujeme jako 0^j, přičemž indexujeme (např.) od nuly.

❖ Kódování symbolů a příkazů L/R: Předpokládejme, že $\Gamma = \Sigma \cup \{\Delta\}$. Uspořádáme Σ do posloupnosti a_1, a_2, ..., a_n a zvolíme tyto kódy: $\Delta \mapsto \varepsilon$, $L \mapsto 0$, $R \mapsto 00$, $a_i \mapsto 0^{i+2}$.

❖ Přechod $\delta(p, x) = (q, y)$, kde $y \in \Gamma \cup \{L, R\}$, reprezentujeme čtveřicí (p, x, q, y) a kódujeme zřetězením kódů p, x, q, y při použití 1 jako oddělovače, tj. jako $\langle p\rangle 1\langle x\rangle 1\langle q\rangle 1\langle y\rangle$, kde $\langle \_ \rangle$ značí kód _.

❖ Celý TS kódujeme jako posloupnost kódů přechodů oddělených a ohraničených 1.

Příklad 7.7



kód: $1110100011000110 01$

Univerzální TS

❖ Zavádí koncept „programovatelného“ stroje, který umožňuje ve vstupním řetězci specifikovat konkrétní TS (tj. program) i data, nad nimiž má tento stroj pracovat.

❖ TS, který má být simulován, budeme kódovat, jak bylo uvedeno na předchozí straně, vstupní řetězec budeme kódovat jako posloupnost příslušných kódů symbolů oddělených a ohraničených 1. Kód stroje a vstupního řetězce oddělíme např. #.

Příklad 7.8 TS z předchozí strany mající na vstupu xxx:

$$111010001100011001\#1000100010001$$

❖ Univerzální TS, který zpracuje toto zadání můžeme navrhnout jako třípáskový stroj, který

- má na 1. pásce zadání (a později výstup),
- 2. pásku používá k simulaci pracovní pásky původního stroje a
- na 3. pásce má zaznamenán řídicí stav simulovaného stroje a aktuální pozici hlavy (pozice hlavy i je kódována jako 0^i).

❖ Univerzální stroj pracuje takto:

1. Stroj zkontroluje, zda vstup odpovídá nějakému $M\#w$ a pokud ne, abnormálně zastaví.
2. Přepíše w na 2. pásku, na 3. pásku umístí kód q_0 a za něj poznačí, že hlava se nachází na levém okraji pásky.
3. Na 2. pásce vyhledá aktuální symbol pod hlavou simulovaného stroje a na 1. pásce vyhledá přechod proveditelný ze stavu zapsaného na začátku 3. pásky pro tento vstupní symbol. Pokud žádný přechod možný není, stroj abnormálně zastaví.
4. Stroj provede na 2. a 3. pásce změny odpovídající simulovanému přechodu (přepis aktuálního symbolu, změna pozice hlavy, změna řídicího stavu).
5. Pokud nebyl dosažen stav q_F simulovaného stroje, přejdeme na bod 3. Jinak stroj vymaže 1. pásku, umístí na ní obsah 2. pásky a zastaví přechodem do svého koncového stavu.

❖ Víme, že výše uvedený stroj můžeme převést na jednopáskový univerzální TS, který budeme v dalším značit jako T_U.

Jazyky rekurzívně vyčíslitelné a jazyky rekurzívní

Rekurzívní vyčíslitelnost a rekurzívnost

❖ Turingův stroj se nazývá úplný (*total*), právě když se pro každý vstup zastaví.

❖ *Poznámka*: Nedeterministický Turingův stroj je úplný, právě když pro každý vstup je každá výpočetní větev konečná (tj. pro každý vstup vždy zastaví).

Definice 8.1 Jazyk $L \subseteq \Sigma^*$ se nazývá

- rekurzívně vyčíslitelný, jestliže $L = L(M)$ pro nějaký TS M,
- rekurzívní, jestliže $L = L(M)$ pro nějaký *úplný* TS M.

❖ Je-li M úplný Turingův stroj, pak říkáme, že M rozhoduje jazyk $L(M)$.

❖ Ke každému rekurzívnímu jazyku existuje TS, který ho rozhoduje, tj. zastaví pro každé vstupní slovo – tento TS lze samozřejmě upravit tak, aby pro každý řetězec z daného jazyka zastavil s páskou $\Delta Y \Delta\Delta$... a jinak zastavil s páskou $\Delta N \Delta\Delta$....

❖ TS přijímající rekurzívně vyčíslitelný jazyk L zastaví pro každé $w \in L$, ovšem pro $w \notin L$ může zastavit, ale také může donekonečna cyklit.

Rozhodovací problémy

❖ Rozhodovací problém (*decision problem*) P může být chápán jako funkce f_P s oborem hodnot $\{true, false\}$.

❖ Rozhodovací problém je obvykle specifikován:

- definičním oborem A_P reprezentujícím množinu možných instancí problému (vstupů) a
- podmnožinou $B_P \subseteq A_P$, $B_P = \{p \mid f_P(p) = true\}$ instancí, pro které je hodnota f_P rovna $true$.

❖ V teorii formálních jazyků používáme ke kódování jednotlivých instancí problémů řetězce nad vhodnou abecedou Σ. Pak je rozhodovací problém P přirozeně specifikován jazykem $L_p = \{w \in \Sigma^* \mid w = code(p), p \in B_P\}$, kde $code : A_P \to \Sigma^*$ je injektivní funkce, která přiřazuje instancím problému příslušný řetězec (nezávisle na f_P).

Příklad 8.1 Příklady rozhodovacích problémů:

- P_1 – orientovaný graf je silně souvislý.
- P_2 – dvě bezkontextové gramatiky jsou ekvivalentní,
- P_3 – n je prvočíslo.

❖ *Poznámka*: Dále budeme o rozhodovacích problémech hovořit jednoduše jako o problémech.

Rozhodování problémů TS

Definice 8.2 Nechť P je problém specifikovaný jazykem L_P nad Σ. Problém P nazveme:

- rozhodnutelný, pokud L_P je rekurzívní jazyk, tj. existuje TS, který L_P rozhoduje (přijme každý řetězec $w \in L_P$, a zamítne každý řetězec $w \in \Sigma^* \setminus L_P$),
- nerozhodnutelný, když není rozhodnutelný, a
- částečně rozhodnutelný, jestliže L_P je rekurzívně vyčíslitelný jazyk.

❖ ***Poznámka:*** Z definice 8.2 plyne, že každý rozhodnutelný problém je současně částečně rozhodnutelný, ale některé nerozhodnutelné problémy nejsou ani částečně rozhodnutelné.

TS a jazyky typu 0

Jazyky přijímané TS jsou typu 0

❖ Pro zápis konfigurace TS v řídícím stavu q a s konfigurací pásky $\Delta x \underline{y} z \Delta ...$ zavedeme konvenci $[\Delta x q y z \Delta ...]$.

Věta 8.1 Každý rekurzívně vyčíslitelný jazyk je jazykem typu 0.

Důkaz. * Nechť $L = L(M)$ pro nějaký TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$. Sestrojíme gramatiku $G = (N, \Sigma, P, S)$ typu 0 takovou, že $L(G) = L(M)$. Gramatika G dovoluje vytvářet derivace odpovídající reverzi posloupnosti konfigurací TS M při přijetí $w \in L(M)$:

1. $N = \{S\} \cup Q \cup (\Gamma \setminus \Sigma) \cup \{[,]\}$ (množiny jsou po dvou disjunktní).
2. P je nejmenší množina obsahující následující pravidla:
   (a) $S \rightarrow [q_f \Delta Y \Delta]$,
   (b) $\Delta] \rightarrow \Delta\Delta]$ – *doplnění* Δ,
   (c) $qy \rightarrow px$, jestliže $\delta(p, x) = (q, y)$,
   (d) $xq \rightarrow px$, jestliže $\delta(p, x) = (q, R)$,
   (e) $qyx \rightarrow ypx$ pro každé $y \in \Gamma$, jestliže $\delta(p, x) = (q, L)$,
   (f) $[q_0\Delta \rightarrow \varepsilon, \quad \Delta\Delta] \rightarrow \Delta], \quad \Delta] \rightarrow \varepsilon$ – *zajištění* $[q_0 \Delta w \Delta ... \Delta] \overset{+}{\underset{G}{\Rightarrow}} w$.

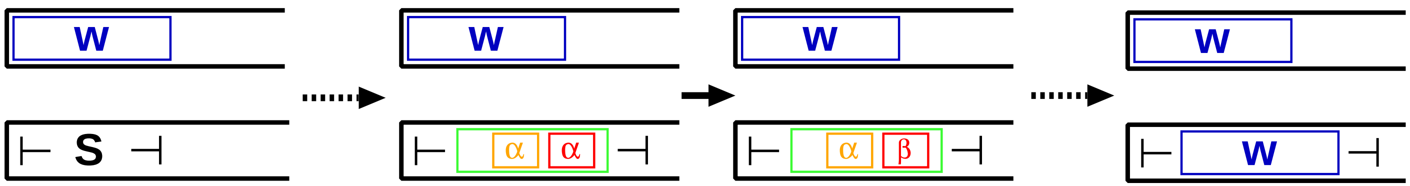
Snadno se nyní nahlédne, že $w \in L(M)$ právě tehdy, když existuje derivace $S \Rightarrow_G [q_F \Delta Y \Delta] \Rightarrow_G ... \Rightarrow_G [q_0 \Delta w \Delta ...] \Rightarrow_G ... \Rightarrow_G w$, a že $L(G) = L(M)$. □ *

Jazyky typu 0 jsou přijímány TS

Věta 8.2 Každý jazyk typu 0 je přijímán nějakým TS (tj. je rekurzívně vyčíslitelný).

Důkaz. Nechť $L = L(G)$ pro $G = (N, \Sigma, P, S)$ je jazykem typu 0. Sestrojíme nedeterministický dvoupáskový TS M takový, že $L(G) = L(M)$:

- 1. páska obsahuje přijímaný vstupní řetězec w.
- Na 2. pásce se M pokouší pomocí simulace použití přepisovacích pravidel $(\alpha \rightarrow \beta) \in P$ vytvořit derivaci w:



1. Stroj nejprve umístí na 2. pásku symbol S.
2. Stroj opakovaně simuluje na 2. pásce provádění pravidel $(\alpha \rightarrow \beta) \in P$. Nedeterministicky zvolí pravidlo a také výskyt α na pásce. Při přepisu α na β, $|\alpha| \neq |\beta|$, může využít posuv části užitečného obsahu pásky vlevo či vpravo.
3. Stroj srovná finální obsah 2. pásky s 1. páskou. Shodují-li se, zastaví přechodem do q_F. Jinak posouvá hlavu doleva až do abnormálního zastavení.

Snadno se nyní nahlédne, že skutečně $L(G) = L(M)$. Navíc lze M podobně jako u vícepáskových DTS převést na jednopáskový NTS a ten dále na jednopáskový DTS. $\square$

Jazyky typu 0 = jazyky přijímané TS

Věta 8.3 ***Třída jazyků přijímaných TS (neboli jazyků rekurzívně vyčíslitelných) je shodná se třídou jazyků typu 0.***

Důkaz. Důsledek dvou předchozích vět. □

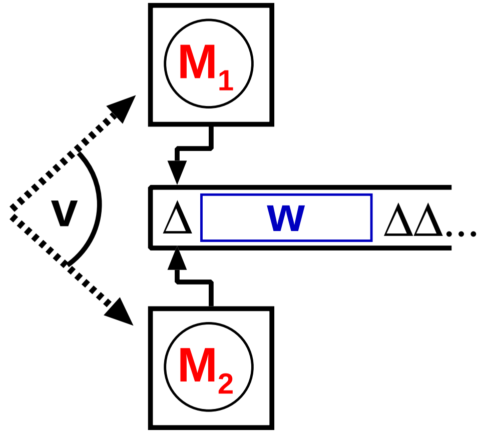
Vlastnosti jazyků rekurzívních a rekurzívně vyčíslitelných

Uzavřenost vůči $\cup$, $\cap$, . a $*$

Věta 8.4 Třídy rekurzívních a rekurzívně vyčíslitelných jazyků jsou uzavřeny vůči operacím $\cup$, $\cap$, . a $*$.

Důkaz. Nechť L_1, L_2 jsou jazyky přijímané TS M_1, M_2. Zřejmě můžeme předpokládat, že množiny stavů TS M_1, M_2 jsou disjunktní.

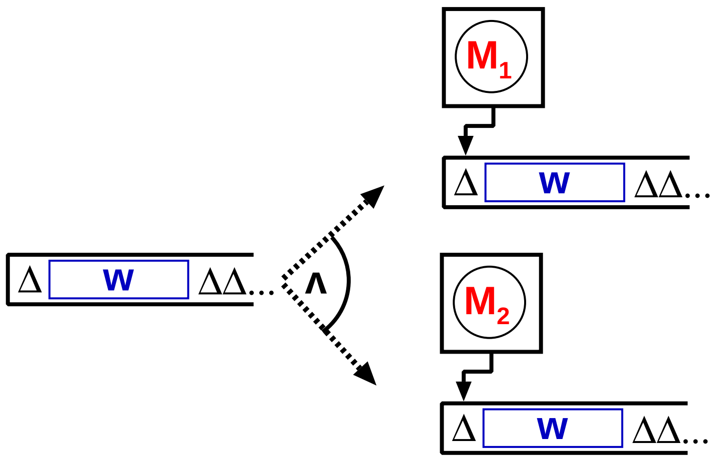
- NTS $M_{L_1 \cup L_2}$, $L(M_{L_1 \cup L_2}) = L_1 \cup L_2$, sestrojíme tak, že sjednotíme po složkách stroje M_1 a M_2, zavedeme nový počáteční stav, z něj nedeterministické přechody přes Δ/Δ do obou původních počátečních stavů a sloučíme původní koncové stavy do jediného nového koncového stavu.



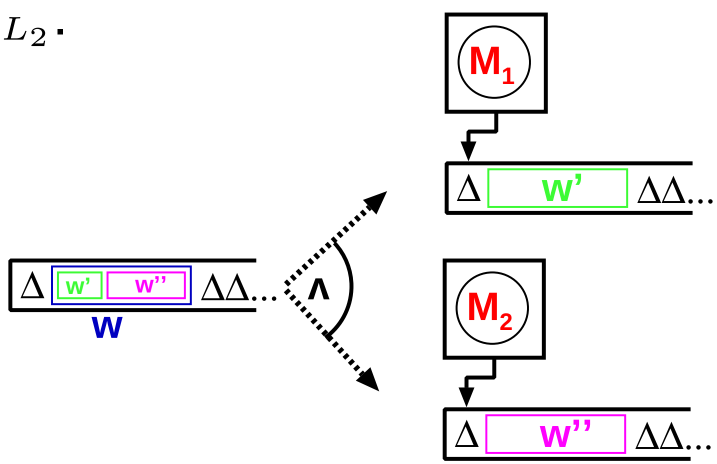
Důkaz pokračuje dále.

Pokračování důkazu.

- Třípáskový TS $M_{L_1 \cap L_2}$, $L(M_{L_1 \cap L_2}) = L_1 \cap L_2$, okopíruje vstup z první pásky na druhou, na ní simuluje stroj M_1, pokud ten přijme, okopíruje vstup z první pásky na třetí, na ní simuluje stroj M_2, pokud i ten přijme, přijme i stroj $M_{L_1 \cap L_2}$.



- Třípáskový NTS $M_{L_1 . L_2}$, $L(M_{L_1 . L_2}) = L_1 . L_2$, okopíruje nedeterministicky zvolený prefix vstupu z první pásky na druhou, na ní simuluje stroj M_1, pokud ten přijme, okopíruje zbytek vstupu z první pásky na třetí, na ní simuluje stroj M_2, pokud i ten přijme, přijme i stroj $M_{L_1 . L_2}$.



Důkaz pokračuje dále.

Pokračování důkazu.

- Dvoupáskový NTS $M_{L_1^*}$, $L(M_{L_1^*}) = L_1^*$, je zobecněním předchozího stroje: po částech kopíruje vstup z první pásky na druhou a na ní simuluje opakovaně stroj M_1. Obsah druhé pásky má ohraničený speciálními značkami a po každé simulaci stroje M_1 ho smaže. Umožňuje samozřejmě posuv pravé značky dále doprava při nedostatku místa.

Jsou-li stroje M_1 a M_2 úplné, je možné vybudovat stroje podle výše uvedených pravidel také jako úplné (u $M_{L_1 \cup L_2}$, $M_{L_1 \cap L_2}$, $M_{L_1 . L_2}$ je to okamžité, u $M_{L_1^*}$ nepřipustíme načítání prázdného podřetězce vstupu z 1. na 2. pásku – pouze umožníme jednorázově přijmout prázdný vstup). To dokazuje uzavřenost vůči uvedeným operacím také u rekurzívních jazyků.

$\square$

(Ne)uzavřenost vůči komplementu

Věta 8.5 Třída rekurzívních jazyků je uzavřena vůči komplementu.

Důkaz. TS M přijímající rekurzívní jazyk L vždy zastaví. Snadno upravíme M na M', který při nepřijetí řetězce vždy přejde do unikátního stavu q_{reject}. TS $\overline{M}$, $L(\overline{M}) = \overline{L}$, snadno dostaneme z M' záměnou q_F a q_{reject}. □

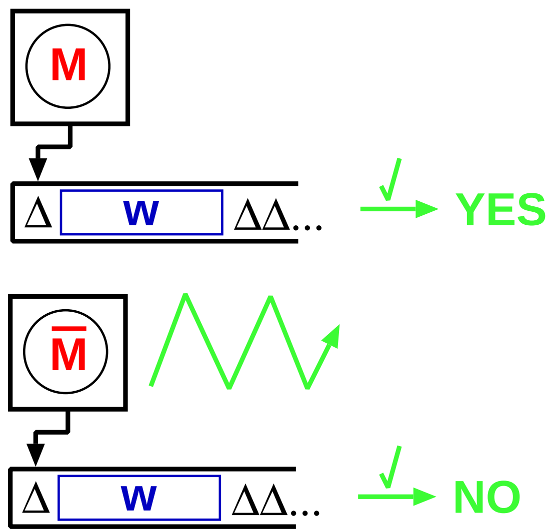
❖ Třída rekurzívně vyčíslitelných jazyků není uzavřena vůči komplementu!

- Výše uvedené konstrukce nelze užít – cyklení zůstane zachováno.
- Důkaz neuzavřenosti bude uveden v dalších přednáškách.

Věta 8.6 Jsou-li L i $\overline{L}$ rekurzívně vyčíslitelné, pak jsou oba rekurzívní.

Důkaz.

Mějme M, $L(M) = L$, a $\overline{M}$, $L(\overline{M}) = \overline{L}$. Úplný TS přijímající L sestrojíme takto:



- Použijeme dvě pásky. Na jedné budeme simulovat M, na druhé $\overline{M}$. Simulace se bude provádět proloženě krok po kroku: krok M, krok $\overline{M}$, krok M, ...
- Přijmeme, právě když by přijal M, zamítneme abnormálním zastavením, právě když by přijal $\overline{M}$. Jedna z těchto situací určitě nastane v konečném počtu kroků.

Existence úplného TS pro $\overline{L}$ plyne z uzavřenosti rekurzívních jazyků vůči komplementu.

□

❖ Důsledkem výše uvedených vět je mj. to, že pro L a $\overline{L}$ musí vždy nastat jedna z následujících situací:

- L i $\overline{L}$ jsou rekurzívní,
- L ani $\overline{L}$ nejsou rekurzívně vyčíslitelné,
- jeden z těchto jazyků je rekurzívně vyčíslitelný, ale ne rekurzívní, druhý není rekurzívně vyčíslitelný.

Lineárně omezené automaty

Lineárně omezené automaty

❖ Lineárně omezený automat (LOA) je nedeterministický TS, který nikdy neopustí tu část pásky, na níž je zapsán jeho vstup.

❖ Formálně můžeme LOA definovat jako NTS, který má v Γ speciální symbol, kterým unikátně označujeme pravý konec vstupu na pásce, přičemž tento symbol není možné přepsat, ani z něj provést posun doprava.

❖ Deterministický LOA můžeme přirozeně definovat jako (deterministický) TS, který nikdy neopustí část pásky se zapsaným vstupem.

❖ Není známo, zda deterministický LOA je či není striktně slabší než LOA.

LOA a kontextové jazyky

Věta 8.7 Třída jazyků, kterou lze generovat kontextovými gramatikami, odpovídá třídě jazyků, které lze přijímat LOA.

Důkaz.

- Uvážíme definici kontextových gramatik jako gramatik s pravidly v podobě $\alpha \to \beta$, kde $|\alpha| \leq |\beta|$, nebo $S \to \varepsilon$.
- *LOA $\longrightarrow$ G1:
  - Použijeme podobnou konstrukci jako u TS $\longrightarrow$ G0.
  - Na počátku vygenerujeme příslušný pracovní prostor, který se pak již nebude měnit: odpadá nekontextové pravidlo $\Delta\Delta] \to \Delta]$.
  - Užití nekontextových pravidel $[q_0\Delta \to \varepsilon$ a $\Delta] \to \varepsilon$ obejdeme (1) zavedením zvláštních koncových nonterminálů integrujících původní informaci a příznak, že se jedná o první/poslední symbol a (2) integrací symbolu reprezentujícího řídící stav a pozici hlavy s následujícím páskovým symbolem.*
- G1 $\longrightarrow$ LOA:
  - Použijeme podobnou konstrukci jako u G0 $\longrightarrow$ TS s tím, že nepovolíme, aby rozsah druhé pásky někdy překročil rozsah první pásky.

$\square$

Kontextové a rekurzívní jazyky

Věta 8.8 Každý kontextový jazyk je rekurzívní.

Důkaz. (Idea)

- Počet konfigurací, které se mohou objevit při přijímání w příslušným LOA M je vzhledem k nemožnosti zvětšovat pracovní prostor pásky konečný: lze shora ohraničit funkcí c^n pro vhodnou konstantu c – exponenciála plyne z nutnosti uvažovat výskyt všechny možných symbolů na všech místech pásky.
- Pro zápis libovolného čísla z intervalu $0, ..., c^n - 1$ nikdy nebude třeba více než n symbolů, užijeme-li c-ární soustavu.
- Můžeme zkonstruovat úplný LOA ekvivalentní s M, který bude mít každý symbol na pásce strukturovaný jako dvojici:
  - S využitím 1. složek těchto dvojic simulujeme M.
  - V 2. složkách počítáme počet kroků; dojde-li k přetečení, odmítneme vstup.

$\square$

Věta 8.9 *Ne* každý rekurzívní jazyk je kontextový.

Důkaz. (Idea) Lze užít techniku diagonalizace prezentovanou dále . $\square$

Vlastnosti kontextových jazyků

Věta 8.10 Třída kontextových jazyků je uzavřena vůči operacím $\cup$, $\cap$, ., $*$ a komplementu.

Důkaz.

- Uzavřenost vůči $\cup$, $\cap$, . a $*$ lze ukázat stejně jako u rekurzívně spočetných jazyků.
- Důkaz uzavřenosti vůči komplementu je značně komplikovaný (všimněme si, že LOA je *nedeterministický* a nelze tudíž užít konstrukce použité u rekurzívních jazyků) – zájemci naleznou důkaz v doporučené literatuře.

□

❖ Poznamenejme, že již víme, že u kontextových jazyků

- lze rozhodovat členství věty do jazyka (rekurzívnost) a
- nelze rozhodovat inkluzi jazyků (neplatí ani pro bezkontextové jazyky).

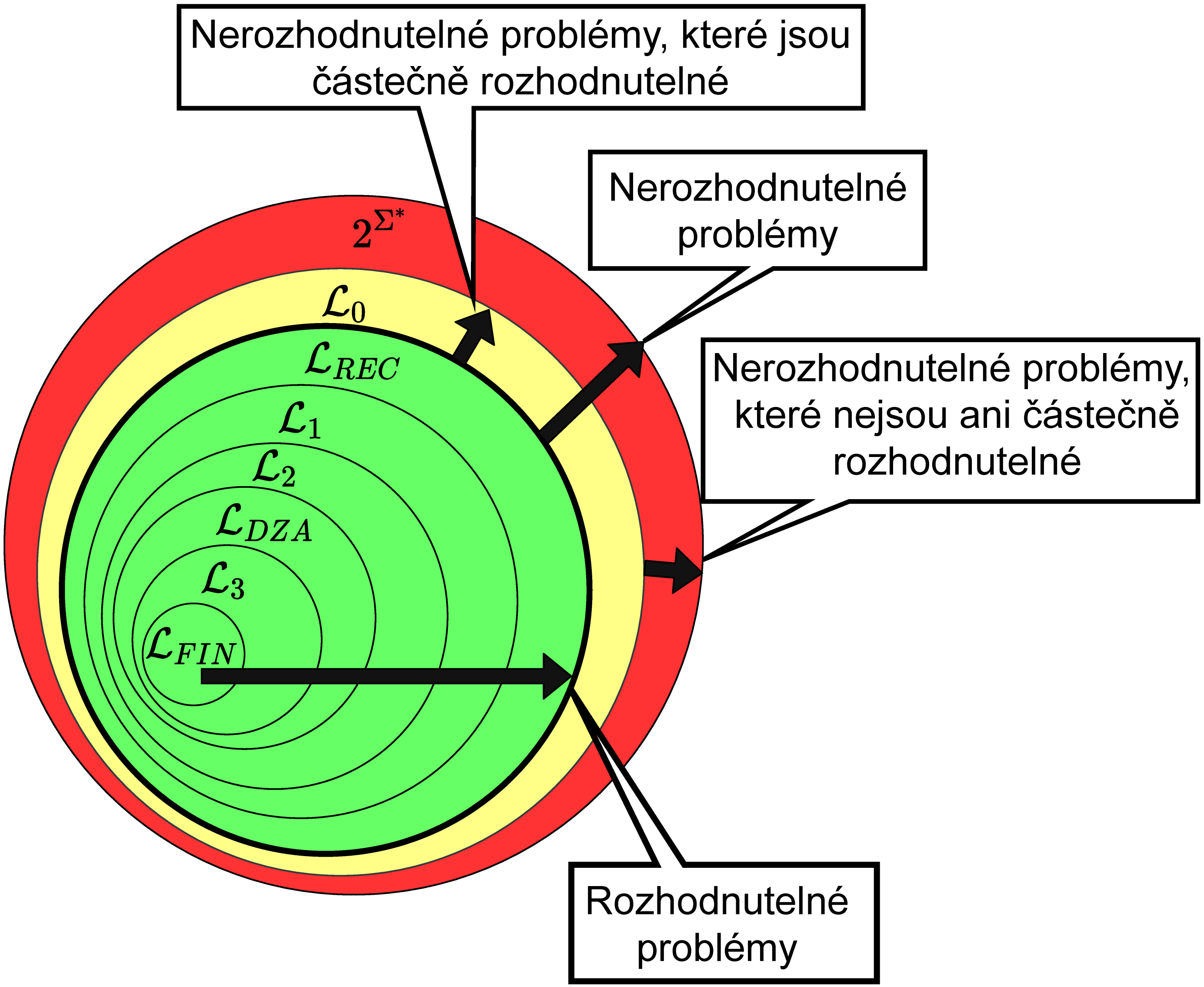
❖ Dále lze ukázat, že pro kontextové jazyky nelze rozhodovat prázdnost jazyka (užije se redukce z Postova problému přiřazení – viz další přednášky).

Meze rozhodnutelnosti

Existují jazyky (problémy), jež nejsou rekurzivně vyčíslitelné (částečně rozhodnutelné)?

Které jazyky, resp. problémy, nejsou rekurzívní (rozhodnutelné)?

Hierarchie jazyků/problémů



- **Ekvivalence tříd:** $\mathcal{L}_3 = \mathcal{L}_{KA}$ $\quad \mathcal{L}_2 = \mathcal{L}_{ZA}$ $\quad \mathcal{L}_1 = \mathcal{L}_{LOA}$ $\quad \mathcal{L}_0 = \mathcal{L}_{TS} = \mathcal{L}_{RE}$

Jazyky mimo třídu 0

Existence jazyků mimo třídu 0

Věta 9.1 Pro každou abecedu Σ existuje jazyk nad Σ, který není typu 0 (tj. rekurzívně vyčíslitelný).

Důkaz.

1. Libovolný jazyk typu 0 nad Σ může být přijat TS s $\Gamma = \Sigma \cup \{\Delta\}$: Pokud M používá více symbolů, můžeme je zakódovat jako jisté posloupnosti symbolů ze $\Sigma \cup \{\Delta\}$ a sestrojit TS M', který simuluje M.
2. Nyní můžeme snadno systematicky vypisovat všechny TS s $\Gamma = \Sigma \cup \{\Delta\}$. Začneme stroji se dvěma stavy, pak se třemi stavy, ...
   Závěr: Množina všech takových strojů a tedy i jazyků typu 0 je spočetná.
3. Množina Σ^* ale obsahuje nekonečně mnoho řetězců a proto je množina 2^{Σ^*} zahrnující všechny jazyky nespočetná – důkaz viz další strana.
4. Z rozdílnosti mohutností spočetných a nespočetných množin plyne platnost uvedené věty.

$\square$

Lemma 9.1 Pro neprázdné, konečné Σ je množina 2^{Σ^*} nespočetná.

Důkaz. Důkaz provedeme tzv. diagonalizací (poprvé použitou Cantorem při důkazu rozdílné mohutnosti $\mathbb{N}$ a $\mathbb{R}$).

- Předpokládejme, že 2^{Σ^*} je spočetná. Pak dle definice spočetnosti existuje bijekce $f : \mathbb{N} \longleftrightarrow 2^{\Sigma^*}$.
- Uspořádejme Σ^* do nějaké posloupnosti $w_1, w_2, w_3, \ldots$, např. ε, x, y, xx, xy, yx, yy, xxx, ... pro $\Sigma = \{x, y\}$. Nyní můžeme f zobrazit nekonečnou maticí:

$$\begin{array}{lcccccc}
 & w_0 & w_1 & w_2 & \ldots & w_i & \ldots \\
L_0 = f(0) & a_{00} & a_{01} & a_{02} & \ldots & a_{0i} & \ldots \\
L_1 = f(1) & a_{10} & a_{11} & a_{12} & \ldots & a_{1i} & \ldots \\
L_2 = f(2) & a_{20} & a_{21} & a_{22} & \ldots & a_{2i} & \ldots \\
\ldots & & & & & &
\end{array}
\text{, kde } a_{ij} = \begin{cases} 0, \text{ jestliže } w_j \notin L_i, \\ 1, \text{ jestliže } w_j \in L_i. \end{cases}$$

- Uvažujme jazyk $\overline{L} = \{w_i \mid a_{ii} = 0\}$. $\overline{L}$ se liší od každého jazyka $L_i = f(i)$, $i \in \mathbb{N}$:
  - je-li $a_{ii} = 0$, pak w_i patří do jazyka,
  - je-li $a_{ii} = 1$, pak w_i nepatří do jazyka.
- Současně ale $\overline{L} \in 2^{\Sigma^*}$, f tudíž není surjektivní, což je spor.

$\square$

Problém zastavení

Problém zastavení TS

Věta 9.2 **Problém zastavení TS (Halting Problem)**, kdy nás zajímá, zda daný TS M pro danou vstupní větu w zastaví, **není rozhodnutelný**, ale je **částečně rozhodnutelný**.

Důkaz.

- Problému zastavení odpovídá rozhodování jazyka $HP = \{\langle M \rangle \# \langle w \rangle \mid M \text{ zastaví při } w\}$, kde $\langle M \rangle$ je kód TS M a $\langle w \rangle$ je kód w.
- Částečnou rozhodnutelnost ukážeme snadno použitím modifikovaného univerzálního TS T_U, který zastaví přijetím vstupu $\langle M \rangle \# \langle w \rangle$ právě tehdy, když M zastaví při w – modifikace spočívá v převedení abnormálního zastavení při simulaci na zastavení přechodem do q_F.
- Nerozhodnutelnost ukážeme pomocí diagonalizace:
  1. Pro $x \in \{0,1\}^*$, nechť M_x je TS s kódem x, je-li x legální kód TS. Jinak ztotožníme M_x s pevně zvoleným TS, např. TS, který pro libovolný vstup okamžitě zastaví.
  2. Můžeme nyní sestavit posloupnost $M_\varepsilon, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{11}, M_{000}, \ldots$ zahrnující všechny TS nad $\Sigma = \{0,1\}$ indexované řetězci z $\{0,1\}^*$.

Důkaz pokračuje dále.

Pokračování důkazu.

3. Uvažme nekonečnou matici

$$
\begin{array}{lllllll}
 & \varepsilon & 0 & 1 & 00 & 01 & 10 \quad \ldots \\
M_\varepsilon & H_{M_\varepsilon,\varepsilon} & H_{M_\varepsilon,0} & H_{M_\varepsilon,1} & H_{M_\varepsilon,00} & H_{M_\varepsilon,01} & \ldots \\
M_0 & H_{M_0,\varepsilon} & H_{M_0,0} & H_{M_0,1} & H_{M_0,00} & H_{M_0,01} & \ldots \\
M_1 & H_{M_1,\varepsilon} & H_{M_1,0} & H_{M_1,1} & H_{M_1,00} & H_{M_1,01} & \ldots \\
M_{00} & H_{M_{00},\varepsilon} & H_{M_{00},0} & H_{M_{00},1} & H_{M_{00},00} & H_{M_{00},01} & \ldots \\
M_{01} & H_{M_{01},\varepsilon} & H_{M_{01},0} & H_{M_{01},1} & H_{M_{01},00} & H_{M_{01},01} & \ldots \\
\ldots
\end{array}
$$

   kde $H_{M_x,y} = \begin{cases} \textbf{C}\text{, jestliže } M_x \text{ cyklí na } y, \\ \textbf{Z}\text{, jestliže } M_x \text{ zastaví na } y. \end{cases}$

4. Předpokládejme, že existuje *úplný* TS K přijímající jazyk HP, tj. K pro vstup $\langle M \rangle \# \langle w \rangle$
   - zastaví normálně (přijme) právě tehdy, když M zastaví na w,
   - zastaví abnormálně (odmítne) právě tehdy, když M cyklí na w.
5. Sestavíme TS N, který pro vstup $x \in \{0,1\}^*$:
   - Sestaví M_x z x a zapíše $\langle M_x \rangle \# x$ na svou pásku.
   - Simuluje K na $\langle M_x \rangle \# x$, přijme, pokud K odmítne, a přejde do nekonečného cyklu, pokud K přijme.

Důkaz pokračuje dále.

Pokračování důkazu.

Všimněme si, že N v podstatě komplementuje diagonálu uvedené matice:

	ε	0	1	00	01	10 ...
M_ε	$H_{M_\varepsilon,\varepsilon}$	$H_{M_\varepsilon,0}$	$H_{M_\varepsilon,1}$	$H_{M_\varepsilon,00}$	$H_{M_\varepsilon,01}$	...
M_0	$H_{M_0,\varepsilon}$	$H_{M_0,0}$	$H_{M_0,1}$	$H_{M_0,00}$	$H_{M_0,01}$	...
M_1	$H_{M_1,\varepsilon}$	$H_{M_1,0}$	$H_{M_1,1}$	$H_{M_1,00}$	$H_{M_1,01}$	...
M_{00}	$H_{M_{00},\varepsilon}$	$H_{M_{00},0}$	$H_{M_{00},1}$	$H_{M_{00},00}$	$H_{M_{00},01}$	...
M_{01}	$H_{M_{01},\varepsilon}$	$H_{M_{01},0}$	$H_{M_{01},1}$	$H_{M_{01},00}$	$H_{M_{01},01}$	...
...						

6. Dostáváme, že

   N zastaví na x $\quad\Leftrightarrow\quad$ K odmítne $\langle M_x\rangle\#\langle x\rangle$ $\quad$ (definice N)

   $\quad\Leftrightarrow\quad$ M_x cyklí na x $\quad$ (předpoklad o K).

7. To ale znamená, že N se liší od každého M_x alespoň na jednom řetězci – konkrétně x. Což je ovšem spor s tím, že posloupnost $M_\varepsilon, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{11}, M_{000}, \ldots$ zahrnuje všechny TS nad $\Sigma = \{0,1\}$. Tento spor plyne z předpokladu, že existuje TS K, který pro daný TS M a daný vstup x určí (rozhodne), zda M zastaví na x, či nikoliv.

$\square$

❖ Ukázali jsme, že problém zastavení TS je částečně rozhodnutelný a tedy jazyk HP rekurzívně vyčíslitelný. Z věty 8.6 pak plyne, že **komplement problému zastavení** není ani částečně rozhodnutelný a jazyk co-$HP = \{\langle M\rangle\#\langle w\rangle \mid M$ nezastaví při $w\}$ je příkladem jazyka, jenž není ani rekurzívně vyčíslitelný.

S dalším příkladem takového jazyka se seznámíme v následujícím problému.

Redukce

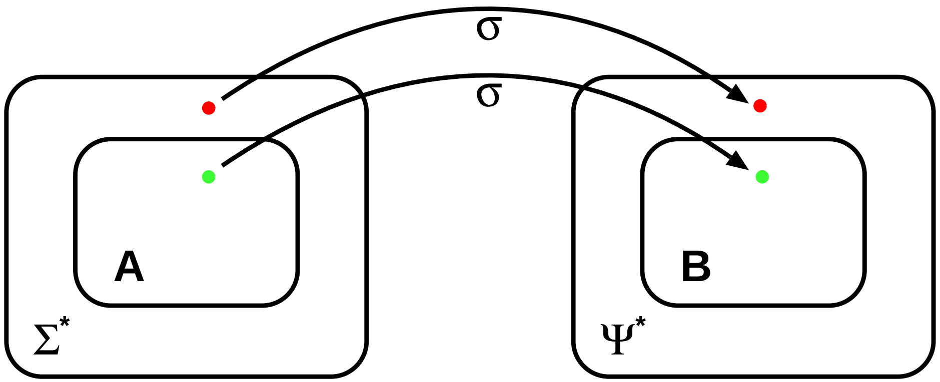
Důkaz nerozhodnutelnosti redukcí

❖ Technika redukce patří spolu s diagonalizací k nejpoužívanějším technikám důkazu, že nějaký problém není rozhodnutelný (částečně rozhodnutelný) – neboli, že určitý jazyk není rekurzívní (rekurzívně vyčíslitelný):

- víme, že jazyk A není rekurzívní (rekurzívně vyčíslitelný),
- zkoumáme jazyk B,
- ukážeme, že A lze úplným TS převést (redukovat) na B,
- to ale znamená, že B rovněž není rekurzívní (rekurzívně vyčíslitelný) – jinak by šlo použít úplný TS (ne-úplný TS) přijímající B a příslušné redukce k sestavení úplného TS (ne-úplného TS) přijímajícího A, což by byl spor.

❖ Argumentace výše samozřejmě ukazuje, že redukci lze použít i při dokazování, že určitý problém je rekurzívní (částečně rekurzívní).

Definice 9.1 Nechť A, B jsou jazyky, $A \subseteq \Sigma^*$, $B \subseteq \Psi^*$. Redukce jazyka A na jazyk B je totální, rekurzívně vyčíslitelná funkce $\sigma : \Sigma^* \rightarrow \Psi^*$ taková, že $\forall w \in \Sigma^*. w \in A \Leftrightarrow \sigma(w) \in B$.



- Existuje-li redukce jazyka A na B, říkáme, že A je redukovatelný na B, což značíme $A \leq B$.

Věta 9.3 Nechť $A \leq B$.

1. Není-li jazyk A rekurzívně vyčíslitelný, pak ani jazyk B není rekurzívně vyčíslitelný.
2. Není-li jazyk A rekurzívní, pak ani jazyk B není rekurzívní.

$\overline{1}$. Je-li jazyk B rekurzívně vyčíslitelný, pak i jazyk A je rekurzívně vyčíslitelný.

$\overline{2}$. Je-li jazyk B rekurzívní, pak i jazyk A je rekurzívní.

Důkaz. Dokážeme, že pokud $A \leq B$, pak ($\overline{1}$) je-li jazyk B rekurzívně vyčíslitelný, pak i jazyk A je rekurzívně vyčíslitelný:

- Nechť M_R je úplný TS počítající redukci σ z A na B a M_B je TS přijímající B.
- Sestrojíme M_A přijímající A:
  1. M_A simuluje M_R na vstupu w, což transformuje obsah pásky na $\sigma(w)$.
  2. M_A simuluje výpočet M_B na $\sigma(w)$.
  3. Pokud M_B zastaví a přijme, M_A rovněž zastaví a přijme, jinak M_A zastaví abnormálně nebo cyklí.
- Zřejmě platí:

$$\begin{array}{lcll} M_A \text{ přijme } w & \Leftrightarrow & M_B \text{ přijme } \sigma(w) & \\ & \Leftrightarrow & \sigma(w) \in B & \\ & \Leftrightarrow & w \in A & \text{(definice redukce).} \end{array}$$

Tvrzení (1) je kontrapozicí ($\overline{1}$); tvrzení ($\overline{2}$) dokážeme podobně jako ($\overline{1}$) při použití úplného TS M_B; tvrzení (2) je kontrapozicí ($\overline{2}$).

Kontrapozice: $p \to q \Leftrightarrow \neg q \to \neg p$ (Modus tollens)

$\square$

Problém náležitosti a další problémy

Problém náležitosti pro $\mathcal{L}_0$

Věta 9.4 **Problém náležitosti (Membership Problem (MP))** řetězce w do jazyka L typu 0 **není rozhodnutelný**, ale **je částečně rozhodnutelný**.

$$MP = \{ <M> \# <w> \mid M \text{ je TS, který akceptuje } w\}$$

Důkaz. Částečná rozhodnutelnost je zřejmá (podobně jako u HP využijeme T_U). Nerozhodnutelnost ukážeme redukcí z problému zastavení: $HP \leq MP$.

- Požadovaná redukce je funkce $\sigma : \{0, 1, \#\} \rightarrow \{0, 1, \#\}$ definovaná jako $\sigma(<M> \# <w>) = <M'> \# <w>$.
- Pokud vstup $<M> \# <w>$ není korektní instance HP, tak funkce σ vrací kód TS M' takového, že $L(M') = \emptyset$ (tj. $<M'> \# <w> \notin MP$). Jinak σ vrací kód TS M', který pracuje následovně.
-
  - M' nejdříve spustí simulaci stroje M na vstupu w. Poznamenejme, že kódy M a w jsou přímo uloženy v kódu M'.
  - Pokud simulace cyklí, tak M' cyklí pro svůj vstup w.
  - Pokud simulace skončí, tak M' akceptuje svůj vstup w.

Důkaz pokračuje dále.

Pokračování důkazu.

- Tudíž pro M' platí:

$$<M> \# <w> \in HP \;\Rightarrow w \in L(M') \;\Rightarrow <M'> \# <w> \in MP \text{ a}$$

$$<M> \# <w> \notin HP \;\Rightarrow w \notin L(M') \;\Rightarrow <M'> \# <w> \notin MP \;\Longleftrightarrow$$

$$<M'> \# <w> \in MP \;\Rightarrow <M> \# <w> \in HP \text{ (použijeme kontrapozici)}$$

$$\text{neboli } <M> \# <w> \in HP \iff \sigma(<M> \# <w>) \in MP.$$

- Je vidět, že výše popsanou konstrukci stroje M' lze implementovat pomocí úplného TS a tudíž funkce σ je totální, rekurzivně vyčíslitelná funkce.

$\square$

❖ Podobně jako u problému zastavení nyní z věty 8.6 plyne, že

- **komplement problému náležitosti** není ani částečně rozhodnutelný a
- jazyk co-$MP = \{\langle M''\rangle\#\langle w''\rangle \mid w'' \notin L(M'')\}$ je dalším příkladem jazyka, jenž není ani rekurzívně vyčíslitelný.

Příklady dalších problémů pro TS

❖ Konstrukcí příslušného úplného TS (a v případě složitější konstrukce důkazem její korektnosti) lze ukázat, že např. následující problémy jsou rozhodnutelné:

- Daný TS má alespoň 2005 stavů.
- Daný TS učiní více než 2005 kroků na vstupu ε.
- Daný TS učiní více než 2005 kroků na *nějakém* vstupu.

❖ Konstrukcí příslušného (ne-úplného) TS a důkazem nerekurzívnosti redukcí lze ukázat, že např. následující problémy jsou částečně rozhodnutelné:

- Jazyk daného TS je neprázdný.
- Jazyk daného TS obsahuje alespoň 2005 slov.

❖ Důkazem redukcí, že jazyky odpovídající následujícím problémům nejsou ani parciálně rekurzívní lze ukázat, že např. následující problémy nejsou ani částečně rozhodnutelné:

- Jazyk daného TS je prázdný.
- Jazyk daného TS obsahuje nanejvýš 2005 slov.
- Jazyk daného TS je konečný (regulární, bezkontextový, kontextový, rekurzívní).

Jak poznat, že jazyk je/není REC/RE?

❖ **Tato charakterizace nelze efektivně použít k důkazu dané vlastnosti.**

❖ Jazyk je L rekurzivní pokud existuje pro všechna $w \in L$ konečný certifikát příslušnosti do jazyka a pro všechna $w \notin L$ konečný certifikát nepříslušnosti.

- $L_1 = \{ <M> \# <w> \mid M$ je úplný TS, který akceptuje $w\}$
- $L_2 = \{ <A> \mid A$ je KA, t.ž. $L(A) = \emptyset\}$
- $L_3 = \{ <M> \mid M$ je TS, který učiní více než 2005 kroků na všech vstupech$\}$

❖ Jazyk je L rekurzivně vyčíslitelný pokud existuje pro všechna $w \in L$ konečný certifikát příslušnosti do jazyka (certifikát nepříslušnosti může být nekonečný) .

- $MP = \{ <M> \# <w> \mid M$ je TS, který akceptuje $w\}$
- $L_4 = \{ <M> \mid M$ je TS, t.ž. $L(M) \neq \emptyset\}$
- $L_5 = \{ <M> \mid M$ je úplný TS, t.ž. $L(M) \neq \Sigma^*\}$

❖ Jazyk je L není ani rekurzivně vyčíslitelný pokud pro nějaké $w \in L$ neexistuje konečný certifikát příslušnosti do jazyka (certifikát nepříslušnosti může být konečný) .

- co-$MP = \{ <M> \# <w> \mid M$ je TS, který neakceptuje $w\}$
- $L_4 = \{ <M> \mid M$ je TS, t.ž. $L(M) = \emptyset\}$
- $L_5 = \{ <M> \mid M$ je úplný TS, t.ž. $L(M) = \Sigma^*\}$

Příklad: L ani $\overline{L}$ není RE

$$L = \{<M_1> \# <M_2> | \ M_1 \text{ a } M_2 \text{ jsou TS, t.ž. } L(M_1) \subseteq L(M_2)\}$$

❖ Napřed dokážeme, že $L \notin \mathcal{L}_0$ pomocí redukce co-$HP \leq L$.

Důkaz.

- Požadovaná redukce je funkce $\sigma : \{0,1,\#\}^* \to \{0,1,\#\}^*$ definovaná takto:

$$\sigma(<M> \# <w>) = <M_1> \# <M_2>$$

- Funkce σ vrací kód TS M_2, pro který platí $L(M_2) = \{a\}$.
- Pokud vstup $<M> \# <w>$ není korektní instance co-HP, tak funkce σ vrací kód TS M_1 takový, že $L(M_1) = \Sigma^*$ (tj. $<M_1> \# <M_2> \notin L$). Jinak σ vrací kód TS M_1, který pracuje následovně.
-
  - M_1 nejdříve spustí simulaci stroje M na vstupu w. Poznamenejme, že kódy M a w jsou přímo uloženy v kódu M_1.
  - Pokud simulace cyklí, tak M_1 cyklí pro svůj každý vstup
  - Pokud simulace skončí, tak M_1 akceptuje každý svůj vstup.

Důkaz pokračuje dále.

Pokračování důkazu.

- Tudíž pro M' platí:

$$<M> \# <w> \in \text{co-}HP \;\Rightarrow\; L(M_1) = \emptyset \;\Rightarrow <M_1> \# <M_2> \in L \text{ a}$$

$$<M> \# <w> \notin \text{co-}HP \;\Rightarrow\; L(M_1) = \Sigma^* \;\Rightarrow <M_1> \# <M_2> \notin L$$

$$\text{neboli } <M> \# <w> \in \text{co-}HP \iff \sigma(<M> \# <w>) \in L.$$

- Je vidět, že výše popsanou konstrukci stroje M_1 a M_2 lze implementovat pomocí úplného TS a tudíž funkce σ je totální, rekurzivně vyčíslitelná funkce.

$\square$

Příklad: L ani $\overline{L}$ není RE (pokračování)

$$L = \{<M_1> \# <M_2>|\ M_1 \text{ a } M_2 \text{ jsou TS, t.ž. } L(M_1) \subseteq L(M_2)\}$$

❖ Nyní dokážeme, že $\overline{L} \notin \mathcal{L}_0$ pomocí redukce co-$HP \leq \overline{L}$.

Důkaz.

- Požadovaná redukce je funkce $\sigma : \{0, 1, \#\}^* \rightarrow \{0, 1, \#\}^*$ definovaná takto:

$$\sigma(<M> \# <w>) =<M_1> \# <M_2>$$

- Funkce σ vrací kód TS M_1, pro který platí $L(M_1) = \{a\}$.
- Pokud vstup $<M> \# <w>$ není korektní instance co-HP, tak funkce σ vrací kód TS M_2 takový, že $L(M_2) = \Sigma^*$ (tj. $<M_1> \# <M_2> \notin \overline{L}$). Jinak σ vrací kód TS M_2, který pracuje následovně.
- – M_2 nejdříve spustí simulaci stroje M na vstupu w. Poznamenejme, že kódy M a w jsou přímo uloženy v kódu M_2.
  – Pokud simulace cyklí, tak M_2 cyklí pro svůj každý vstup
  – Pokud simulace skončí, tak M_2 akceptuje každý svůj vstup.

Důkaz pokračuje dále.

Pokračování důkazu.

- Tudíž pro M' platí:

$$<\!M\!> \# <\!w\!> \in \text{co-}HP \;\Rightarrow\; L(M_2) = \emptyset \;\Rightarrow <\!M_1\!> \# <\!M_2\!> \in \overline{L} \text{ a}$$

$$<\!M\!> \# <\!w\!> \notin \text{co-}HP \;\Rightarrow\; L(M_2) = \Sigma^* \;\Rightarrow <\!M_1\!> \# <\!M_2\!> \notin \overline{L}$$

$$\text{neboli } <\!M\!> \# <\!w\!> \in \text{co-}HP \iff \sigma(<\!M\!> \# <\!w\!>) \in \overline{L}.$$

- Je vidět, že výše popsanou konstrukci stroje M_1 a M_2 lze implementovat pomocí úplného TS a tudíž funkce σ je totální, rekurzivně vyčíslitelná funkce.

$\square$

Příklad komplikovanější redukce

❖ Dokažte, že následující jazyk není RE.

$$L_{NB} = \{<M> \mid M \text{ je TS t.ž. } L(M) \notin \mathcal{L}_2\}$$

❖ Opět použijeme redukci co-$HP \leq L_{NB}$.

Důkaz.

- Požadovaná redukce je funkce $\sigma : \{0, 1, \#\}^* \to \{0, 1\}^*$ definovaná takto:

$$\sigma(<M> \# <w>) = <M'>$$

- Pokud vstup $<M> \# <w>$ není korektní instance co-HP, tak funkce σ vrací kód TS M' takový, že $L(M') = \Sigma^*$ (tj. $<M'> \notin L_{NB}$). Jinak σ vrací kód TS M', který pracuje následovně.

Důkaz pokračuje dále.

Pokračování důkazu.

- – M' nejdříve ověří, zda jeho vstup $w' \in \{a^n b^n c^n \mid n > 0\}$. Pokud ano, M' akceptuje w', jinak pokračuje.
  – M' spustí simulaci stroje M na vstupu w. Poznamenejme, že kódy M a w jsou přímo uloženy v kódu M'.
  – Pokud simulace cyklí, tak M' cyklí pro svůj vstup w'.
  – Pokud simulace skončí, tak M' akceptuje svůj vstup w'.

- Tudíž pro M' platí:

$$<M> \# <w> \in \text{co-}HP \;\Rightarrow\; L(M') = \{a^n b^n c^n \mid n > 0\} \;\Rightarrow <M'> \in L_{NB} \text{ a}$$

$$<M> \# <w> \notin \text{co-}HP \;\Rightarrow\; L(M') = \Sigma^* \;\Rightarrow <M'> \notin L_{NB}$$

$$\text{neboli } <M> \# <w> \in \text{co-}HP \iff \sigma(<M> \# <w>) \in L_{NB}.$$

- Je vidět, že výše popsanou konstrukci stroje M' lze implementovat pomocí úplného TS a tudíž funkce σ je totální, rekurzivně vyčíslitelná funkce.

□

Postův korespondenční problém

Postův korespondenční problém

Definice 9.2

- Postův systém nad abecedou Σ je dán neprázdným seznamem S dvojic neprázdných řetězců nad Σ, $S = \langle(\alpha_1, \beta_1), ..., (\alpha_k, \beta_k)\rangle$, $\alpha_i, \beta_i \in \Sigma^+$, $k \geq 1$.
- Řešením Postova systému je každá neprázdná posloupnost přirozených čísel $I = \langle i_1, i_2, ..., i_m\rangle$, $1 \leq i_j \leq k$, $m \geq 1$, taková, že:

$$\alpha_{i_1}\alpha_{i_2}...\alpha_{i_m} = \beta_{i_1}\beta_{i_2}...\beta_{i_m}$$

(Pozor: m není omezené a indexy se mohou opakovat!)

- Postův problém (PCP) zní: Existuje pro daný Postův systém řešení?

Příklad 9.1

- Uvažujme Postův systém $S_1 = \{(b, bbb), (babbb, ba), (ba, a)\}$ nad $\Sigma = \{a, b\}$. Tento systém má řešení $I = \langle 2, 1, 1, 3\rangle$: $babbb\ b\ b\ ba = ba\ bbb\ bbb\ a$.
- Naopak Postův systém $S_2 = \{(ab, abb), (a, ba), (b, bb)\}$ nad $\Sigma = \{a, b\}$ nemá řešení, protože $|\alpha_i| < |\beta_i|$ pro $i = 1, 2, 3$.

Nerozhodnutelnost PCP

Věta 9.5 **Postův korespondenční problém je nerozhodnutelný.**

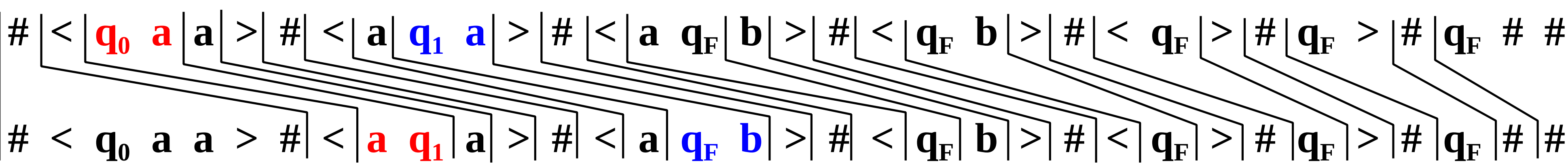
Důkaz. *(Idea) Dá se ukázat, že nerozhodnutelnost PCP plyne z nerozhodnutelnosti tzv. iniciálního PCP, u kterého požadujeme, aby řešení začínalo vždy jedničkou. Nerozhodnutelnost iniciálního PCP se dá ukázat redukcí z problému náležitosti pro TS:

- Konfiguraci výpočtu TS lze zakódovat do řetězce: použitý obsah pásky ohraničíme speciálními značkami (a jen ten si pamatujeme), řídicí stav vložíme na aktuální pozici hlavy na pásce.
- Posloupnost konfigurací TS při přijetí řetězce budeme reprezentovat jako konkatenaci řetězců, která vzniká řešením PCP.
- Jedna z uvažovaných konkatenací bude celou dobu (až na poslední fázi) delší: na začátku bude obsahovat počáteční konfiguraci a pak bude vždy o krok napřed. V posledním fázi výpočtu konkatenace „zarovnáme“ (bude-li možné výpočet simulovaného TS ukončit přijetím).
- Výpočet TS budeme modelovat tak, že vždy jednu konkatenaci postupně prodloužíme o aktuální konfiguraci simulovaného TS a současně v druhé konkatenaci vygenerujeme novou konfiguraci TS.

Důkaz pokračuje dále.

Pokračování důkazu.

- Jednotlivé dvojice PCP budou modelovat následující kroky:
  - Vložení počáteční konfigurace simulovaného TS do jedné z konkatenací např. pravostranné $(\#, \#$počáteční_konfigurace$)$, $\# \notin \Gamma$ používáme jako oddělovač konfigurací.
  - Kopírování symbolů na pásce před a po aktuální pozici hlavy (z, z) pro každé $z \in \Gamma \cup \{\#, <, >\}$, kde $<, >$ ohraničují použitou část pásky.
  - Základní změna konfigurace: přepis $\delta(q_1, a) = (q_2, b)$: $(q_1 a, q_2 b)$, posuv doprava $\delta(q_1, a) = (q_2, R)$: $(q_1 a, a q_2)$, posuv doleva $\delta(q_1, b) = (q_2, L)$: $(a q_1 b, q_2 a b)$ pro každé $a \in \Gamma \cup \{<\}$. Navíc je zapotřebí ošetřit nájezd na $>$: čtení Δ, rozšiřování použité části pásky.
  - Pravidla pro „zarovnání“ obou konkatenací při přijetí: na levé straně umožníme přidat symbol v okolí q_F, aniž bychom ho přidali na pravé straně.
- Simulace výpočtu TS, který načte a, posune hlavu doprava, přepíše a na b a zastaví, na vstupu aa by pak vypadala takto:



- Obecná korektnost konstrukce se dá dokázat indukcí nad délkou výpočtu.

□ *

Nerozhodnutelnost redukcí z PCP

❖ Redukce z PCP (resp. jeho doplňku) se velmi často používají k důkazům, že určitý problém není rozhodnutelný (resp. není ani částečně rozhodnutelný).

❖ Jako příklad uvedeme důkaz faktu, že problém prázdnosti jazyka dané kontextové gramatiky není ani částečně rozhodnutelný:

- Použijeme redukci z komplementu PCP. Redukce přiřadí seznamu $S = (\alpha_1, \beta_1), ..., (\alpha_k, \beta_k)$, definujícímu instanci PCP, kontextovou gramatiku G takovou, že PCP založený na S nemá řešení právě tehdy, když $L(G) = \emptyset$.
- Uvažme jazyky L_α, L_β nad $\Sigma \cup \{\#, 1, ..., k\}$ (předp. $\Sigma \cap \{\#, 1, ..., k\} = \emptyset$):
  - $L_\alpha = \{\alpha_{i_1} ... \alpha_{i_m} \# i_m ... i_1 \mid 1 \leq i_j \leq k, j = 1, ..., m, m \geq 1\}$,
  - $L_\beta = \{\beta_{i_1} ... \beta_{i_m} \# i_m ... i_1 \mid 1 \leq i_j \leq k, j = 1, ..., m, m \geq 1\}$.
- Je zřejmé, že L_α, L_β jsou kontextové (dokonce deterministické bezkontextové) a tudíž $L_\alpha \cap L_\beta$ je také kontextový jazyk (věta 8.10) a můžeme tedy efektivně sestavit gramatiku G, která tento jazyk generuje (např. konstrukcí přes LOA).
- $L_\alpha \cap L_\beta$ zřejmě obsahuje právě řetězce $u\#v$, kde v odpovídá reverzi řešení dané instance PCP.
- Hledaná redukce tedy přiřadí dané instanci PCP gramatiku G. $\square$

Souhrn některých vlastností jazyků

❖ Uvedeme nyní souhrn některých důležitých vlastností různých tříd jazyků; některé jsme dokázali, důkazy jiných lze nalézt v literatuře[a] (u otázek nerozhodnutelnosti se často užívá redukce z PCP) – R = rozhodnutelný, N = nerozhodnutelný, A = vždy splněno:

	Reg	DCF	CF	CS	Rec	RE
$w \in L(G)$?	R	R	R	R	R	N
$L(G)$ prázdný? konečný?	R	R	R	N	N	N
$L(G) = \Sigma^*$?	R	R	N	N	N	N
$L(G) = R$, $R \in \mathcal{L}_3$?	R	R	N	N	N	N
$L(G_1) = L(G_2)$?	R	R	N	N	N	N
$L(G_1) \subseteq L(G_2)$?	R	N	N	N	N	N
$L(G_1) \in \mathcal{L}_3$?	A	R	N	N	N	N
$L(G_1) \cap L(G_2)$ je stejného typu?	A	N	N	A	A	A
$L(G_1) \cup L(G_2)$ je stejného typu?	A	N	A	A	A	A
Komplement $L(G)$ je stejného typu?	A	A	N	A	A	N
$L(G_1).L(G_2)$ je stejného typu?	A	N	A	A	A	A
$L(G)^*$ je stejného typu?	A	N	A	A	A	A
Je G víceznačná?	R	N	N	N	N	N

[a] Např. I. Černá, M. Křetínský, A. Kučera. Automaty a formální jazyky I. FI MU, 1999.

Riceova věta

Nerozhodnutelnost je pravidlo, ne výjimka.

Riceova věta – první část

Věta 9.6 ***Každá*** **netriviální vlastnost rekurzívně vyčíslitelných jazyků je nerozhodnutelná.**

Definice 9.3 Budiž dána abeceda Σ. Vlastnost rekurzívně vyčíslitelných množin je zobrazení P : { rekurzívně vyčíslitelné podmnožiny množiny Σ^* } $\rightarrow \{\perp, \top\}$, kde $\top$, resp. $\perp$ reprezentují pravdu, resp. nepravdu.

Příklad 9.2 Vlastnost prázdnosti můžeme reprezentovat jako zobrazení

$$P(A) = \begin{cases} \top, \text{ jestliže } A = \emptyset, \\ \perp, \text{ jestliže } A \neq \emptyset. \end{cases}$$

❖ Zdůrazněme, že nyní mluvíme o vlastnostech rekurzívně vyčíslitelných množin, *nikoliv* TS, které je přijímají – následující vlastnosti tedy nejsou vlastnostmi r. v. množin:

- TS M má alespoň 2005 stavů.
- TS M zastaví na všech vstupech.

Definice 9.4 Vlastnost rekurzívně vyčíslitelných množin je netriviální, pokud není vždy pravdivá ani vždy nepravdivá.

Důkaz 1. části Riceovy věty

Důkaz.

- Nechť P je netriviální vlastnost r.v. množin. Předpokládejme beze ztráty obecnosti, že $P(\emptyset) = \bot$, pro $P(\emptyset) = \top$ můžeme postupovat analogicky.
- Jelikož P je netriviální vlastnost, existuje r.v. množina A taková, že $P(A) = \top$. Nechť K je TS přijímající A.
- Redukujeme HP na $\{\langle M \rangle \mid P(L(M)) = \top\}$. Z $\langle M \rangle \# \langle w \rangle$ sestrojíme $\sigma(\langle M \rangle \# \langle w \rangle) = \langle M' \rangle$, kde M' je 2-páskový TS, který na vstupu x:
  1. Uloží x na 2. pásku.
  2. Zapíše na 1. pásku w – w je „uložen“ v řízení M'.
  3. Odsimuluje na 1. pásce M – M je rovněž „uložen“ v řízení M'.
  4. Pokud M zastaví na w, odsimuluje K na x a přijme, pokud K přijme.
- Dostáváme:
  - M zastaví na $w \Rightarrow L(M') = A \Rightarrow P(L(M')) = P(A) = \top$,
  - M cyklí na $w \Rightarrow L(M') = \emptyset \Rightarrow P(L(M')) = P(\emptyset) = \bot$,

  A máme tedy skutečně redukci HP na $\{\langle M \rangle \mid P(L(M)) = \top\}$. Protože HP není rekurzivní, není rekurzivní ani $P(L(M))$ a tudíž není rozhodnutelné, zda $L(M)$ splňuje vlastnost P.

$\square$

Riceova věta – druhá část

Definice 9.5 Vlastnost P r.v. množin nazveme monotónní, pokud pro každé dvě r.v. množiny A, B takové, že $A \subseteq B$, $P(A) \Rightarrow P(B)$.

Příklad 9.3 Mezi monotónní vlastnosti patří např.:

- A je nekonečné.
- $A = \Sigma^*$.

Naopak mezi nemonotónní vlastnosti patří např.:

- A je konečné.
- $A = \emptyset$.

Věta 9.7 ***Každá* netriviální nemonotónní vlastnost rekurzívně vyčíslitelných jazyků není ani částečně rozhodnutelná.**

Důkaz. Redukcí z co-HP – viz např. D. Kozen. Automata and Computability. □

Alternativy k TS

Některé alternativy k TS

❖ Mezi výpočetní mechanismy mající ekvivalentní výpočetní sílu jako TS patří např. automaty s (jednou) frontou:

- Uvažme stroj s konečným řízením, (neomezenou) FIFO frontou a přechody na nichž je možno načíst ze začátku fronty a zapsat na konec fronty symboly z frontové abecedy Γ.
- Pomocí „rotace“ fronty je zřejmě možné simulovat pásku TS.

❖ Ekvivalentní výpočetní sílu jako TS mají také zásobníkové automaty se dvěma (a více) zásobníky:

- Intuitivně: obsah pásky simulovaného TS máme v jednom zásobníku; chceme-li ho změnit (obecně nejen na vrcholu), přesuneme část do druhého zásobníku, abychom se dostali na potřebné místo, provedeme patřičnou změnu a vrátíme zpět odloženou část zásobníku.
- Poznámka: rovněž víme, že pomocí dvou zásobníků můžeme implementovat frontu.

❖ Jiným výpočetním modelem s plnou Turingovskou silou jsou automaty s čítači (pro dva a více čítačů) a operacemi $+1$, -1 a test na 0:

- Zmíněné automaty mají konečné řízení a k čítačů, přičemž v každém kroku je možné tyto čítače nezávisle inkrementovat, dekrementovat a testovat na nulu (přechod je podmíněn tím, že jistý čítač obsahuje nulu).
- Pomocí čtyř čítačů je snadné simulovat dva zásobníky:
  - U ZA postačí mít $\Gamma = \{0, 1\}$: různé symboly můžeme kódovat určitým počtem 0 oddělených 1. Obsah zásobníku má pak charakter binárně zapsaného čísla. Vložení 0 odpovídá vynásobení 2, odebrání 0 vydělení 2. Podobně je tomu s vložením/odebráním 1.
  - Binární zásobník můžeme simulovat dvěma čítači: při násobení/dělení 2 odečítáme 1 (resp. 2) z jednoho čítače a přičítáme 2 (resp. 1) k druhému.
- Postačí ovšem i čítače dva:
  - Obsah čtyř čítačů i, j, k, l je možné zakódovat jako $2^i 3^j 5^k 7^l$.
  - Přičtení/odečtení je pak možné realizovat násobením/dělením 2, 3, 5, či 7.

❖ Mezi další Turingovsky úplné výpočetní mechanismy pak patří např. λ-kalkul či parciálně-rekurzívní funkce (viz další přednáška).

Vyčíslitelné funkce

Základy teorie rekurzivních funkcí

Budeme se snažit identifikovat takové funkce, které jsou „spočitatelné“, tj. vyčíslitelné v obecném smyslu (bez ohledu na konkrétní výpočetní systém). Abychom snížili extrémní velikost třídy těchto funkcí, která je dána také varietou definičních oborů a oborů hodnot, omezíme se, uvažujíce možnost kódování, na funkce tvaru:

$$\boxed{f : \mathbb{N}^m \to \mathbb{N}^n}$$

kde $\mathbb{N} = \{0, 1, 2, \ldots\}, \quad m, n \in \mathbb{N}$

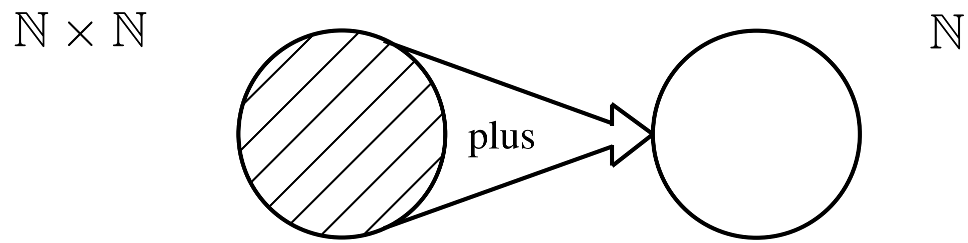
❖ Konvence: n-tici $(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$ budeme označovat jako $\overline{x}$

❖ Klasifikace parciálních funkcí:

- *Totální funkce*
- *Striktně parciální funkce*

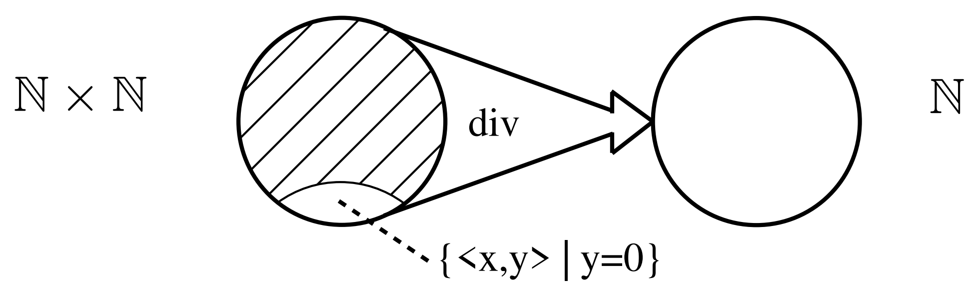
Příklad 11.1 Totální funkce *plus*

$$plus : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$
$$plus(x, y) = x + y$$



Příklad 11.2 Striktně parciální funkce *div*

$$div : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$
$div(x, y) =$ celá část x/y, je-li $y \neq 0$



Počáteční funkce

Hierarchie vyčíslitelných funkcí je založena na dostatečně elementárních tzv. *počátečních funkcích*, které tvoří „stavební kameny“ vyšších funkcí.

- Jsou to tyto funkce:

1. *Nulová funkce* (zero function): $\xi() = 0$
   zobrazuje „prázdnou n-tici“ $\mapsto 0$

2. *Funkce následníka* (successor function): $\sigma : \mathbb{N} \to \mathbb{N}$
   $\sigma(x) = x + 1$

3. *Projekce* (projection): $\pi_k^n : \mathbb{N}^n \to \mathbb{N}$
   Vybírá z n-tice k-tou složku, např.: $\pi_2^3(7, 6, 4) = 6$ a $\pi_1^2(5, 17) = 5$
   Speciální případ: $\pi_0^n : \mathbb{N}^n \to \mathbb{N}^0$, tj. např. $\pi_0^3(1, 2, 3) = ()$

Primitivně rekurzivní funkce

Nyní definujme tři způsoby vytváření nových, složitějších funkcí:

1. *Kombinace*:
   Kombinací dvou funkcí $f : \mathbb{N}^k \to \mathbb{N}^m$ a $g : \mathbb{N}^k \to \mathbb{N}^n$ získáme funkci, pro kterou:

$$\boxed{\begin{array}{l} f \times g : \mathbb{N}^k \to \mathbb{N}^{m+n} \\ f \times g(\overline{x}) = (f(\overline{x}), g(\overline{x})),\ \overline{x} \in \mathbb{N}^k \end{array}}$$

   Např.: $\pi_1^3 \times \pi_3^3(4, 12, 8) = (4, 8)$

2. *Kompozice*:
   Kompozice dvou funkcí $f : \mathbb{N}^k \to \mathbb{N}^m$ a $g : \mathbb{N}^m \to \mathbb{N}^n$ je funkce, pro kterou:

$$\boxed{\begin{array}{l} g \circ f : \mathbb{N}^k \to \mathbb{N}^n \\ g \circ f(\overline{x}) = g(f(\overline{x})),\ \overline{x} \in \mathbb{N}^k \end{array}}$$

   Např.:
   $\sigma \circ \xi() = 1$
   $\sigma \circ \sigma \circ \xi() = 2$

3. *Primitivní rekurze*:

Příklad 11.3 Předpokládejme, že chceme definovat funkci násobení $mult : \mathbb{N}^2 \to \mathbb{N}$.

$$mult(x, y) = \underbrace{x + \cdots + x}_{y}$$

Zřejmě:

(a) Pro $y = 0$ platí $x * 0 = 0$

(b) Pro $y > 0$ je výsledek $x + mult(x, y - 1)$

Takže funkci $mult$ můžeme definovat následujícím předpisem:

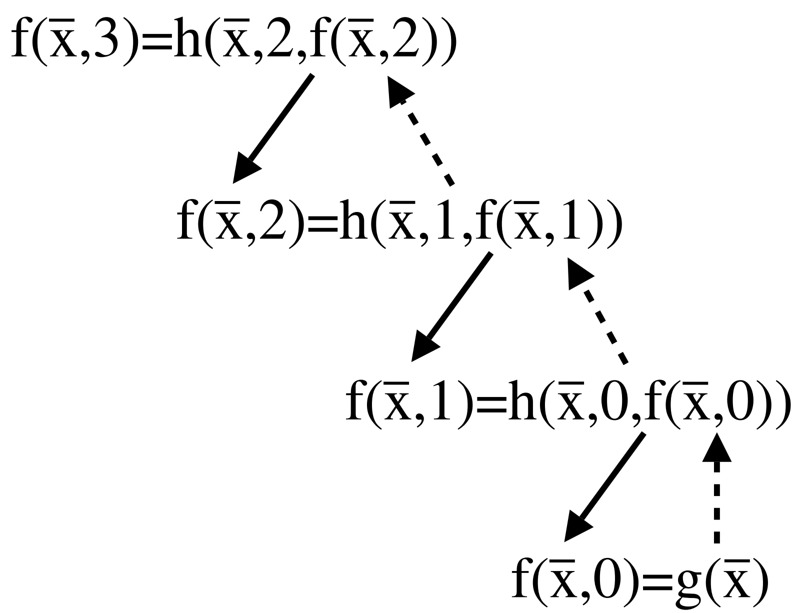
$mult(x, 0) = 0$

$mult(x, y + 1) = x + mult(x, y)$

Primitivní rekurze je technika, která umožňuje vytvořit funkci $f : \mathbb{N}^{k+1} \to \mathbb{N}^m$ na základě jiných dvou funkcí $g : \mathbb{N}^k \to \mathbb{N}^m$ a $h : \mathbb{N}^{k+m+1} \to \mathbb{N}^m$ rovnicemi:

$$\boxed{\begin{array}{l} f(\overline{x}, 0) = g(\overline{x}) \\ f(\overline{x}, y+1) = h(\overline{x}, y, f(\overline{x}, y)), \overline{x} \in \mathbb{N}^k \end{array}}$$

Ilustrace schématu vyčíslení (pro $y = 3$):



Příklad 11.4 Uvažujme funkci $plus : \mathbb{N}^2 \to \mathbb{N}$. Může být definována pomocí primitivní rekurze takto:

$$plus(x, 0) = \pi_1^1(x)$$
$$plus(x, y + 1) = \sigma \circ \pi_3^3(x, y, plus(x, y))$$

což vyjadřuje:

1. $x + 0 = x$
2. $x + (y + 1) = (x + y) + 1 = \sigma(x + y)$

Definice 11.1 *Třída primitivních rekurzivních funkcí* obsahuje všechny funkce, které mohou být z počátečních funkcí vytvořeny:

(a) kombinací

(b) kompozicí

(c) primitivní rekurzí

Věta 11.1 Každá primitivní rekurzivní funkce je totální funkcí.

Důkaz. Počáteční funkce jsou totální. Aplikací kombinace, kompozice a primitivní rekurze na totální funkce dostaneme totální funkce. □

Příklady primitivně rekurzivních funkcí

Třída primitivně rekurzivních funkcí zahrnuje většinu funkcí typických v aplikacích počítačů.

- Konvence: Namísto funkcionálních zápisů typu $h \equiv plus \circ (\pi_1^3 \times \pi_3^3)$ budeme někdy používat zápis $h(x, y, z) = plus(x, z)$ nebo $h(x, y, z) = x + z$

- *Konstantní funkce*: Zavedeme funkci κ_m^n, která libovolné n-tici $\overline{x} \in \mathbb{N}^n$ přiřadí konstantní hodnotu $m \in \mathbb{N}$

$$\kappa_m^0 \equiv \underbrace{\sigma \circ \sigma \circ \ldots \circ \sigma}_{m-\text{krát}} \circ \xi$$

Pro pro $n > 0$ je κ_m^n definována následovně:

$$\kappa_m^n = \kappa_m^0 \circ \pi_0^n$$

Např.: $\kappa_3^2(1,1) = \pi_3^3(1, 0, \kappa_3^2(1,0)) = \kappa_3^2(1,0) = \kappa_3^1(1) = \kappa_3^1(0) = \kappa_3^0() = 3$
Kombinací funkcí κ_m^n dostáváme konstanty z $\mathbb{N}^n$, $n > 1$
Např.: $\kappa_2^3 \times \kappa_5^3(x, y, z) = (2, 5)$

❖ *Funkce násobení*:

$$mult(x,0) = \kappa_0^1(x)$$
$$mult(x,y+1) = plus(x, mult(x,y))$$

❖ *Funkce umocňování*: $\exp : \mathbb{N}^2 \to \mathbb{N}$ - analogicky - viz. cvičení

❖ *Funkce předchůdce*:

$$pred(0) = \xi()$$
$$pred(y+1) = \pi_1^2(y, pred(y))$$

Poznámka: $pred$ je totální funkcí: $pred(0) = 0$

❖ *Funkce monus*:

$$monus(x,0) = \pi_1^1(x)$$
$$monus(x,y+1) = pred(monus(x,y))$$

Význam: $monus(x,y) = \begin{cases} x - y & \text{je-li } x \geq y \\ 0 & \text{jinak} \end{cases}$

Notace: $monus(x,y) \equiv x \dot{-} y$

❖ *Funkce eq* (equal): $eg(x,y) = \begin{cases} 1 & \text{je-li } x = y \\ 0 & \text{je-li } x \neq y \end{cases}$

Definice 11.2 $eq(x,y) = 1\dot{-}((y\dot{-}x) + (x\dot{-}y))$ nebo formálněji
$eq \equiv monus \circ (\kappa_1^2 \times (plus \circ ((monus \circ (\pi_2^2 \times \pi_1^2)) \times monus \circ (\pi_1^2 \times \pi_2^2))))$

Příklad 11.5 $eq(5,3) = 1\dot{-}((3\dot{-}5) + (5\dot{-}3)) = 1\dot{-}(0+2) = 1\dot{-}2 = 0$

❖ *Funkce $\neg$ eq*: $\neg eq \equiv monus \circ (\kappa_1^2 \times eq)$ $\quad (\equiv 1\dot{-}eq)$

❖ *Funkce quo* (quotient): $quo(x,y) = \begin{cases} \text{celá část podílu } x/y \text{ je-li } y \neq 0 \\ 0 \text{ je-li } y = 0 \end{cases}$

Tato funkce může být definována primitivní rekurzí:

$$quo(0,y) = 0$$
$$quo(x+1,y) = quo(x,y) + eq(x+1, mult(quo(x,y),y) + y)$$

Funkce mimo primitivně rekurzivní funkce

Existují funkce, které jsou vyčíslitelné a nejsou primitivně rekurzivními funkcemi. Jsou to všechny striktně parciální funkce (jako div), ale i totální funkce. Taková totální funkce byla prezentována W. Ackermannem (1928) a nazývá se *Ackermannova funkce*. Je dána rovnicemi:

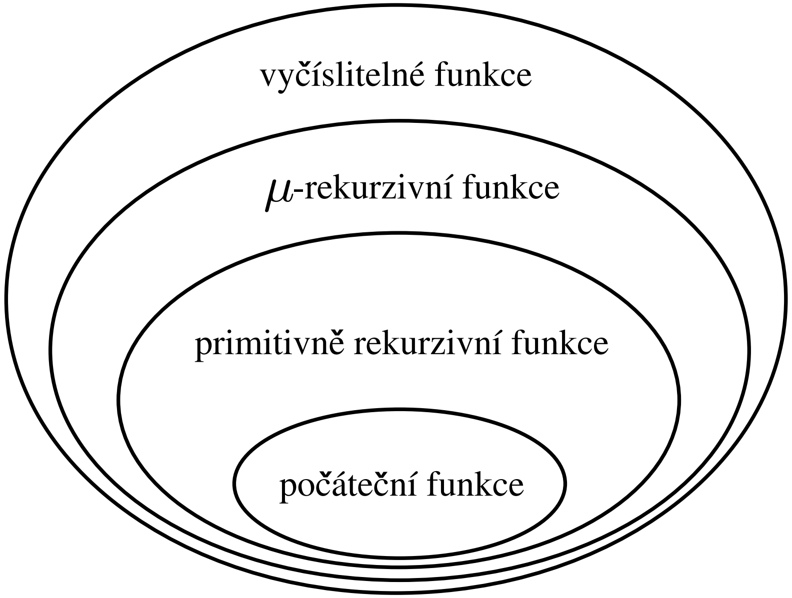
$$\boxed{\begin{aligned} &A(0,y) = y+1 \\ &A(x+1,0) = A(x,1) \\ &A(x+1,y+1) = A(x,A(x+1,y)) \end{aligned}}$$

Věta 11.2 Existuje totální funkce z $\mathbb{N}$ do $\mathbb{N}$, která není primitivně rekurzivní.

Důkaz.
Definice funkcí, které jsou primitivně rekurzivní budeme chápat jako řetězce a můžeme je uspořádat v lexikografickém pořadí s označením $f_1, f_2, \ldots, f_n, \ldots$
Definujeme nyní funkci $f : \mathbb{N} \to \mathbb{N}$ tak, že $f(n) = f_n(n) + 1$ pro $\forall n \in \mathbb{N} \setminus \{0\}$. f je jasně totální a vyčíslitelná. f však není primitivně rekurzivní (kdyby byla, pak $f \equiv f_m$ pro nějaké $m \in \mathbb{N}$. Pak ale $f(m) = f_m(m)$ a ne $f_m(m) + 1$, jak vyžaduje definice funkce f).

$\square$

Definice 11.3 Třída totálních vyčíslitelných funkcí se nazývá *μ-rekurzivní funkce*.



Parciálně rekurzivní funkce

K rozšíření třídy vyčíslitelných funkcí za totální vyčíslitelné funkce zavedeme techniku známou pod názvem *minimalizace*. Tato technika umožňuje vytvořit funkci $f : \mathbb{N}^n \to \mathbb{N}$ z jiné funkce $g : \mathbb{N}^{n+1} \to \mathbb{N}$ předpisem, v němž $f(\overline{x})$ je nejmenší y takové, že:

1. $g(\overline{x}, y) = 0$
2. $g(\overline{x}, z)$ je definována pro $\forall z < y$, $z \in \mathbb{N}$

Tuto konstrukci zapisujeme notací:

$$\boxed{f(\overline{x}) = \mu y[g(\overline{x}, y) = 0]}$$

Příklad 11.6 $f(x) = \mu y[plus(x, y) = 0]$ tj. $f(x) = \begin{cases} 0 \text{ pro } x = 0 \\ \text{nedef. jinak} \end{cases}$

Příklad 11.7 $div(x, y) = \mu t[((x + 1) \dot{-} (mult(t, y) + y)) = 0]$

Příklad 11.8 $i(x) = \mu y[monus(x, y) = 0]$ tj. identická funkce

❖ Funkce definovaná minimalizací je skutečně vyčíslitelná. Výpočet hodnoty $f(\overline{x})$ zahrnuje výpočet $g(\overline{x}, 0), g(\overline{x}, 1), \ldots$ tak dlouho, pokud nedostaneme:

(a) $g(\overline{x}, y) = 0$ ($f(\overline{x}) = y$)

(b) $g(\overline{x}, z)$ je nedefinována ($f(\overline{x})$ je nedefinována)

Definice 11.4 *Třída parciálně rekurzivních funkcí* je třída parciálních funkcí, které mohou být vytvořeny z počátečních funkcí aplikací:

(a) kombinace

(b) kompozice

(c) primitivní rekurze

(d) minimalizace

Vztah vyčíslitelných funkcí a Turingových strojů

Turingovsky vyčíslitelné funkce

Definice 11.5 Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ *vyčísluje (počítá)* parciální funkci $f : \Sigma^{*m} \rightarrow \Sigma_1^{*n}$, $\Sigma_1 \subseteq \Gamma, \Delta \notin \Sigma_1$, jestliže pro každé $(w_1, w_2, \ldots, w_m) \in \Sigma^{*m}$ a odpovídající počáteční konfiguraci $\underline{\Delta} w_1 \Delta w_2 \Delta \ldots \Delta w_m \Delta^\omega$ stroj M:

1. v případě, že $f(w_1, \ldots, w_m)$ je definována, pak M zastaví a páska obsahuje $\underline{\Delta} v_1 \Delta v_2 \Delta \ldots \Delta v_n \Delta\Delta\Delta$, kde $(v_1, v_2, \ldots, v_n) = f(w_1, \ldots, w_m)$
2. v případě, že $f(w_1, \ldots, w_m)$ není definována, M cyklí nebo zastaví abnormálně.

Parciální funkce, kterou může počítat nějaký Turingův stroj se nazývá funkcí *Turingovsky vyčíslitelnou*.

Příklad 11.9
Funkci $f(w) = w^R$ je Turingovsky vyčíslitelná.

Příklad 11.10

Nechť L je libovolný jazyk.

Funkce $f(w) = \begin{cases} |w| & \text{jestliže } w \in L \\ 0 & \text{jestliže } w \notin L \end{cases}$ není Turingovsky vyčíslitelná.

Turingovská vyčíslitelnost a parciálně rekurzivní funkce

Věta 11.3 Každá parciálně rekurzivní funkce je Turingovsky vyčíslitelná.

Idea důkazu: počáteční funkce, kombinaci, kompozici, projekci, primitivní rekurzi i minimalizaci lze implementovat pomocí Turingova stroje.

Věta 11.4 Každý výpočetní proces prováděný Turingovým strojem je procesem vyčíslení nějaké parciálně rekurzivní funkce.

Idea důkazu: viz další slide.

Idea důkazu

Konfiguraci Turingova stroje lze zakódovat například jako čtveřici následujícím způsobem:

- Pro páskovou abecedu definujme funkci $c : \Gamma \rightarrow < 0, |\Gamma|)$, $c(\Delta) = 0)$.
- *Konfiguraci TS* $\Delta a_1 \dots a_{n-1} a_n \underline{a} b_1 b_2 \dots b_m \Delta^\omega$ ve stavu q_i zakódujeme jako *čtveřici čísel* (v soustavě o základu Γ): $(c(a_1) \dots c(a_n), c(a), c(b_m) \dots c(b_2)c(b_1), i)$
- *Provedení přechodu* $\delta(q_i, a) = (q_j, X)$ na konfiguraci (l, c, r, i) vypočteme následovně:
  - $X = L : (l\ quo\ |\Gamma|, l\ mod\ |\Gamma|, plus(mult(r, |\Gamma|), c), j)$
  - $X = R : (plus(mult(l, |\Gamma|), c), r\ mod\ |\Gamma|, r\ quo\ |\Gamma|, j)$
  - $X = a : (l, c(a), r, j)$
- plus, mult, quo a mod jsou *primitivne rekursivní*.
- Pro přechodovou funkci δ je možné definovat primitivně rekursivní funkci $step_\delta : \mathbb{N}^4 \rightarrow \mathbb{N}^4$ provádějící *jeden krok výpočtu* TS. Pro koncovou konfiguraci x je $step_\delta(x) = x$.
- Dále je možné definovat funkci $final : \mathbb{N}^4 \rightarrow \{0, 1\}$, která vrací 1 pro nekoncovou konfiguraci a 0 pro koncovou.

Idea důkazu

- Nyní definujme následující primitivně rekursivní funkci provádějící k kroků TS:

$$step_k(a, b, c, d, 0) = (a, b, c, d)$$

$$step_k(a, b, c, d, k + 1) = step_\delta(step_k(a, b, c, d, k))$$

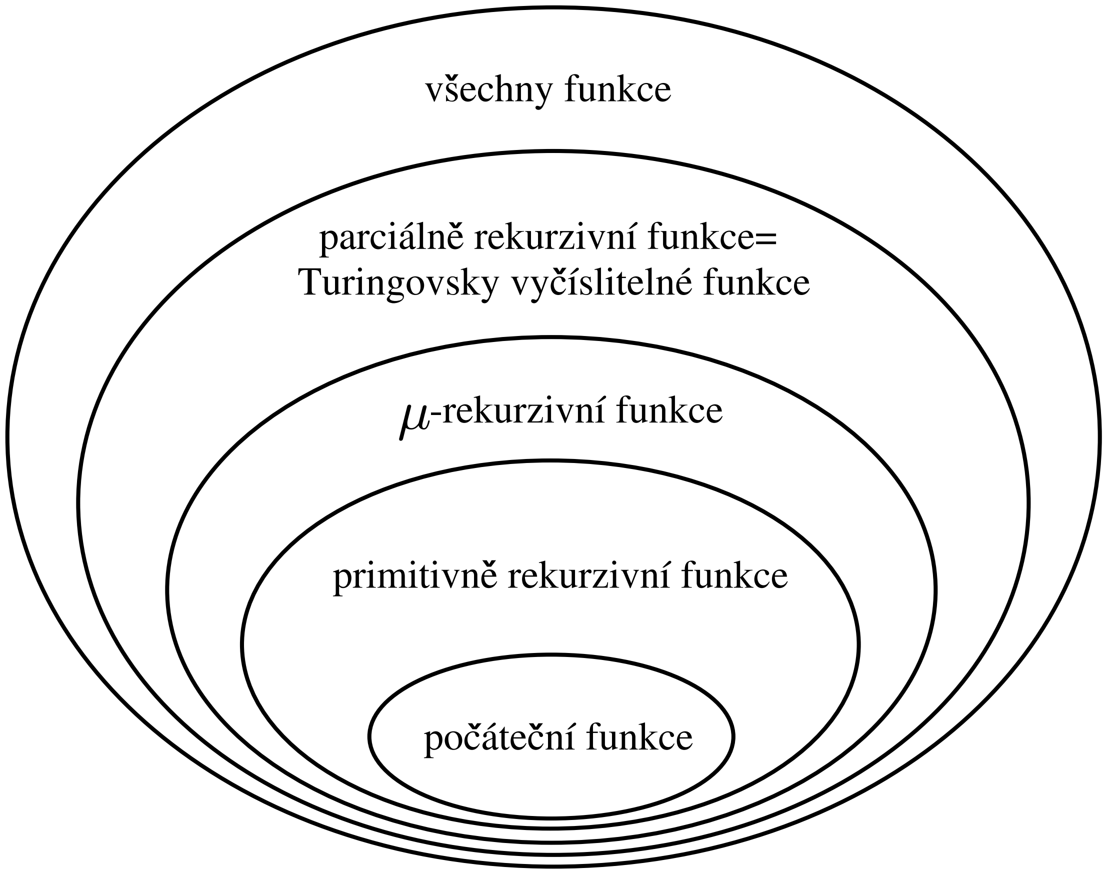
- Výpočet počtu kroků nutných k zastavení TS je pak definován jako:

$$nsteps(a, b, c, d) = \mu k[final(steps_k(a, b, c, d, k)) = 0]$$

- Konfigurace v čase zastavení je pak definována pak:

$$result(a, b, c, d) = step_k(a, b, c, d, nsteps(a, b, c, d))$$

Shrňme v obrázku získané informace:



Souvislost počítání a dokazování

Část I

Opakování

Predikátová logika (PL)

$\models \varphi$, ***φ je logicky platná***

φ platí ve všech mat strukturách, interpretacích symbolů jazyka

např. $\models (\forall x : x > x) \rightarrow (1 > 1)$

ale $\not\models 1 + 1 = 2$

$\vdash \varphi$, ***φ je dokazatelná***

důkaz = sekvence formulí, které jsou axiomy nebo odvozené odvozovacími pravidly z předchozích, končí φ.

V podstatě syntaktická manipulace s formulemi.

Vlastnosti PL

Platné je právě to, co je dokazatelné.

PL je korektní $\vdash \varphi \Rightarrow \models \varphi$
PL je úplná $\models \varphi \Rightarrow \vdash \varphi$

PL je efektivní.
Je možné čistě mechanicky ověřit, co je správný důkaz.

$\{w \mid w$ je důkazem v PL$\}$ je rekurzivní.

Efektivnost $\Rightarrow$ generování důkazů

Pro efektivní log. systém existuje program ***Generátor důkazů***, který vypisuje (nekonečný) seznam důkazů, a každý důkaz časem vypíše.

foreach *řetězec* **do**
  if *řetězec* je důkazem **then**
    vypiš řetězec

* Důkaz je řetězec symbolů z konečné abecedy symbolů.
* Řetězce je možné generovat v abecedním pořadí, na každý jednou dojde.
* Efektivnost: existuje program, který rozhodne, zda je řetězec důkazem.

V efektivním systému je $\{\varphi \mid \varphi$ je dokazatelná$\}$ částečně rozhodnutelná.

Příklad běhu Generátoru důkazů

a	$xyp = \rightarrow$
b	$\ldots$
c	$x = y \rightarrow p$
$\ldots$	$\ldots$
$\wedge$	$xy = \rightarrow (((,((\wedge\neg\neg x,,\exists$
$\neg$	$\ldots$
$\exists$	$p{\rightarrow}((p{\rightarrow}p){\rightarrow}p),\ \ (p{\rightarrow}(p{\rightarrow}p){\rightarrow}(p{\rightarrow}p)),\ \ \forall x\forall y\, x{=}y$
aa	$\ldots$
ab	Proletěl mi zubr zdí, myslel, že to ubrzdí. (Plíhal)
ac	$\ldots$
$\ldots$	$\boldsymbol{p{\rightarrow}((p{\rightarrow}p){\rightarrow}p),\ \ (p{\rightarrow}((p{\rightarrow}p){\rightarrow}p){\rightarrow}((p{\rightarrow}(p{\rightarrow}p)){\rightarrow}(p{\rightarrow}p)),}\ \ \underline{\boldsymbol{(p{\rightarrow}(p{\rightarrow}p){\rightarrow}(p{\rightarrow}p))}}$
aab	$\ldots$
$\ldots$	$\ldots$
$ab\exists$	$\ldots$
$\ldots$	**...důkaz...,** $\underline{\boldsymbol{(\forall x\, x{>}x){\rightarrow}(1{>}1)}}$
$a \vee b\neg($	$\ldots$
$\ldots$	

Efektivnost + korektnost + úplnost $\Rightarrow$ rozhodování platnosti

Tedy umíme řešit „Entscheidungsproblem“.

Pro PL tedy existuje program ***Rozhodovač platnosti formulí***. Pro danou φ:

> Spusť Generátor důkazů.
> **if** *Generátor vypsal důkaz* φ **then** φ je platná
> **if** *Generátor vypsal důkaz* $\neg\varphi$ **then** φ není platná

- Úplnost zaručuje, že program skončí, protože:
  - Pro každou větu máme $\models \varphi$, nebo $\models \neg\varphi$ (z def. sémantiky).
  - Z úplnosti proto $\vdash \varphi$, nebo $\vdash \neg\varphi$.
- Korektnost zaručuje, že odpověď je správná.

Prvořádové teorie

* Rozhodování platnosti v čisté PL ještě není výhra.
  Umíme jen dokazovat logickou platnost formulí jako $\forall x\ x > x \rightarrow 1 > 1$.
* $1 + 1 = 2$ ale není logicky platná formule, neplatí ve všech interpretacích.
* Chceme dokazovat a rozhodovat, jestli $1 + 1 = 2$ je platná v přirozených číslech (t.j. když 1, 2, a + jsou interpretovány, jak jsme zvyklí).
* Zajímají nás pravdy o konkrétních matematických strukturách.

Teorie, množina speciálních axiomů, definuje vlastnosti matematických struktur.

* $\boldsymbol{T} \models \boldsymbol{\varphi}$, φ platí ve všech strukturách, které splňují axiomy T.
* $\boldsymbol{T} \vdash \boldsymbol{\varphi}$, φ je dokazatelná z axiomů T.

Peanova aritmetika T_{PA}

1. $\forall x \neg(S(x) = 0)$ (nula je první)
2. $\forall xy(S(x) = S(y) \to x = y)$ (každý má jiného následníka)
3. pro formule φ jazyka T_{PA} s jednou volnou proměnnou:

$$\left[\varphi(0) \wedge (\forall x(\varphi(x) \to \varphi(S(x))))\right] \to \forall x\,(\varphi(x)) \qquad \text{(axiom indukce)}$$

4. $\forall x(x + 0 = x)$ (0 je neutrální k $+$)
5. $\forall xy(x + S(y) = S(x + y))$ (def. sčítání)
6. $\forall x(x \cdot 0 = 0)$ (0 je nulová k $\cdot$)
7. $\forall xy(x \cdot S(y) = x \cdot y + x)$ (def. násobení)

Hurá, umíme říct, že 1 + 1 = 2 platí.

$$T_{PA} \models \overset{1}{S(0)} + \overset{1}{S(0)} = \overset{2}{S(S(0))}$$

Vlastnosti teorií

Rozumná teorie T je

* ***efektivní***: T (množina axiomů) je rekurzivní.
* ***bezesporná***: nikdy $T \vdash \varphi$ a zároveň $T \vdash \neg\varphi$.

V efektivní teorii je $\{\varphi \mid T \vdash \varphi\}$ částečně rozhodnutelná.

Když T definuje strukturu přesně (jediný model až na izomorfismus), nebo v ní není možné vyjádřit rozdíl mezi různými modely, pak je

* ***úplná***: $T \vdash \varphi$ nebo $T \vdash \neg\varphi$.

Gödelova úplnost pro prvořádové teorie:
$T \models \varphi$ právě když $T \vdash \varphi$

Rozhodnutelná teorie

Teorie je rozhodnutelná, pokud $\{\varphi \mid T \models \varphi\}$ je rozhodnutelná (rekurzivní).

Pro efektivní, bezespornou a úplnou teorii je $\{\varphi \mid T \vdash \varphi\} = \{\varphi \mid T \models \varphi\}$ rozhodnutelná (rekurzivní) monžina. (generátor časem vygeneruje důkaz φ nebo $\neg\varphi$)

Efektivní, bezesporná a úplná teorie je rozhodnutelná.
Efektivní a bezesporná teorie je částečně rozhodnutelná.

Sporná teorie je rozhodnutelná triviálně, protože každá formule je jejím důsledkem.

Část II

Souvislost neúplnosti a nerozhodnutelnosti

Neúplnost a nerozhodnutelnost

Gödel: Teorie PL, která zahrnuje Peanovu aritmetiku, nemůže být úplná.
V PL není možné přesně definovat, co jsou přirozená čísla
(ani v žádném jiném korektním a efektivním axiomatickém systému).

Turing: Existují problémy, které nejsou obecně algoritmicky řešitelné.
(jako problém zastavení)

Tyto dva výsledky těsně souvisejí.

Důkaz a výpočet

* Důkaz a výpočet jsou velmi podobné věci:
    - Je to sekvence ***formulí*** / ***konfigurací***,
    - které jsou buď ***axiomy*** / ***iniciální konfigurace***
    - nebo jsou odvozeny z předchozích pomocí jednoduchých mechanických ***odvozovacích pravidel*** / ***pravidel daných přechodovou funkcí***.
* Dá se ukázat, že:
    - Dokazatelnost aritm. formule můžeme redukovat na zastavení Turingova stroje (generátor důkazů).
    - Zastavení Turingova stroje můžeme redukovat na platnost aritm. formule (ukážeme na dalším slajdu).

Redukce problému zastavení na problém platnosti aritm. formule

Mějme DTS M který, pro jednoduchost, nikdy neskončí abnormálně.
Sestrojíme aritm. formuli φ_w^M platnou právě tehdy, když M zastaví na slově $w = a_1 \cdots a_n$.
Bude definovat vztah mezi stavem $S(k)$ v kroce k výpočtu, pozicí hlavy $H(k)$ v kroce k, a znakem $Z(k,p)$ v kroce k na poli pásky p:

$$\varphi_w^M \equiv \varphi_{init} \wedge \varphi_\Delta \wedge \varphi_{stop}, \text{ kde}$$

$$\varphi_{init} \equiv S(0) = q_0 \wedge H(0) = 1 \wedge Z(0,1) = \Delta \wedge \left(\bigwedge_{p=2}^{n+1} Z(0,p) = a_p\right) \wedge (\forall p > n+1 : Z(0,p) = \Delta)$$

$$\varphi_\Delta \equiv \forall k \forall p \bigwedge_{q \in Q, a \in \Gamma} \varphi_{(q,a)}, \quad \text{kde pokud } \delta(q,a) = (q', X), X \in \{L, R\} \cup \Sigma, \text{ potom}$$

$$\begin{aligned} \varphi_{(q,a)} \equiv (S(k) = q \wedge H(k) = p \wedge Z(k,p) = a) \rightarrow & \\ (s(k+1) = q') \wedge H(k+1) = p' \wedge Z(k+1,p) = a' \wedge & \\ \forall p' \neq p : Z(k+1,p') = Z(k,p')) & \end{aligned}$$

$$\text{kde } p' = \begin{cases} p & \text{pokud } X \in \Sigma \\ p+1 & \text{pokud } X = R \\ p-1 & \text{pokud } X = L \end{cases}, \qquad a' = \begin{cases} X & \text{pokud } X \in \Sigma \\ a & \text{pokud } X \in \{L, R\} \end{cases}.$$

$$\varphi_{stop} \equiv \exists k : S(k) = q_f$$

Redukce problému zastavení na problém platnosti aritm. formule

φ_w^M je platná právě tehdy, když T zastaví na w.

Tedy, platnost aritmetických formulí nemůže být rozhodnutelná, protože potom by byl rozhodnutelný problém zastavení.

T_{PA} nemůže být úplná, protože pak by platnost aritm. formulí byla rozhodnutelná.

Stejně ani žádné rozšíření T_{PA} (efektivní a bezesporné), ani jakýkoliv jiný efektivní a korektní systém charakterizující aritmetiku přirozených čísel přesněji, nemůže být úplný.

Kostra Gödelova důkazu

Gödel ještě neměl Turingovy stroje. Podařilo se mu ale „programovat“ v aritmetice.
Hlavní myšlenkou je konstrukce formule, která, velmi neformálně, říká „Nejsem dokazatelná“.
Na to je potřeba do sčítání a násobení zakódovat formule a důkazy.

* Každý symbol c je kódován číslem. Formule φ je tedy zapsána jako slovo $w_\varphi = a_1 \cdots a_n \in \mathbb{N}^*$, a je kódována ***Gödelovým číslem***

$$G(\varphi) = 2^{a_1} \cdot 3^{a_2} \cdot 5^{a_3} \cdot p_4^{a_4} \cdots p_n^{a_n}, \qquad \text{kde } p_i \text{ je } i\text{-té prvočíslo.}$$

* Kódování φ do $G(\varphi)$ a dekódování je popsatelné aritmetickými formulemi.
* Důkaz je sekvence formulí, a má tedy také své Gödelovo číslo.
* Aplikace odvozovacích pravidel je popsatelná aritmetickými formulemi.
* V aritmetice je možné definovat predikát D tak, že $D(m,n)$ platí, právě když m je kódem důkazu formule $\varphi(G(\varphi))$, kde n je číslo formule $\varphi(x)$ ($\varphi(x)$ je formule s jednou volnou proměnnou, za kterou je v $\varphi(G(\varphi))$ dosazeno $G(\varphi)$).
* Nechť $\psi(x)$ je formule $\neg\exists y\ D(y,x)$.
* Formule $\psi(G(\psi)) : \neg\exists y\ D(y, G(\psi))$ komplikovaně říká „Nejsem dokazatelná“.
    * $\psi(G(\psi))$ je dokazatelná. Pak podle definice D neexistuje důkaz $\psi(G(\psi))$.
    * $\neg\psi(G(\psi))$ je dokazatelná. Pak podle definice D existuje důkaz $\psi(G(\psi))$. Spor.

Podobnost Gödelova a Turingova důkazu

Gödel: Nemůžeme dokázat nebo vyvrátit každou formuli.
Sporem. Formule $\psi(x) : \neg\exists y\ D(y, x)$ říká, že formule s G. číslem x není dokazatelná, pokud je x dosazeno za její volnou proměnnou.

* $\psi(G(\psi))$ je dokazatelná. Pak, podle definice ψ, neexistuje důkaz $\psi(G(\psi))$.
* $\neg\psi(G(\psi))$ je dokazatelná. Pak, podle definice ψ, existuje důkaz $\psi(G(\psi))$.

Turing: Nemůžeme rozhodovat problém zastavení.
Sporem. TS M, který zastaví, právě když jeho vstup je kódem TS, který nezastaví na vlastním kódu.

* $M(\langle M\rangle)$ zastaví. Pak, podle definice M, M nezastaví s vlastním kódem na vstupu.
* $M(\langle M\rangle)$ nezastaví. Pak, podle definice M, M zastaví s vlastním kódem na vstupu.

Za všechno může sebereference?

* ***Kréťan***: Teď lžu.
* ***Russel***: Je množina množin, které nejsou prvkem sama sebe, prvkem sama sebe?
* ***Gödel***: Je formule „Nejsem dokazatelná.“ dokazatelná?
* ***Turing***: Stroj M zastaví na kódu stroje M' právě tehdy, když M' nezastaví na vlastním kódu. Zastaví M na vlastním kódu?

O čem přemýšlejí vrány na elektrickém vedení

Je možné logicky zdůvodnit (dokázat) všechno, co je pravda?
(např. o přirozených číslech?)
Už víme, že to nejde v žádném jednom rozumném axiomatickém systému.
Můžeme ale dokazovcí systémy dál vyvíjet, např. objevovat nové axiomy, které nám umožní dokázat více a více formulí.
Můžeme tak časem dokázat nebo vyvrátit cokoliv?
Dvě možnosti:

1. Proces vymýšlení nových axiomů a systémů je také výpočtem stroje s Turingovskou silou.
   Pak je to jen komplikovaný TS generující teorémy.
   Aplikují se tedy věty o neúplnosti a nerozhodnutelnosti:
   Nemůžeme dokázat každou platnou formuli. Existují nepoznatelné pravdy.
2. Pokud můžeme každou formuli někdy nějak logicky zdůvodnit, dokázat, pak je lidské přemýšlení a vývoj procesem s větší než Turingovskou silou.

Poznámky a cvičení pojmů

- ***Úplnost teorie implikuje rozhodnutelnost*** (generátorem teorémů a důkazů). ***Ne naopak***. Teorie může být rozhodnutelné, tj., existuje algoritmus rozhodující platnost v teorii, a stále neúplná.
- ***Čistá prvořádová logiky s rovností*** (axiomy rovnosti: reflexivita, funkční a predikátová kongruence) ***je neúplná*** (např. jednoprvkový vs. nekonečný model a formule $\forall x \forall y(x = y)$, je možné ji zúplnit přidáním axiomu specifikujícího velikost domény). ***Je ale rozhodnutelná*** (Leopold Löwenheim, 1915).
- Úplnost a „přesnost definice" není přesně to stejné. Peanova aritmetika je neúplná, ale Presburgerovat aritmetika (Peanova aritmetika bez násobení) je úplná. Protože T_{PA} je neúplná, má více modelů, a Protože presburgerova aritmetika je podmnožinou speciálních axiomů T_{PA}, musí mít také více modelů. Presburgerova aritmetika je ale uplná, protože ***rozdíl mezi různými modely se v ní nedá vyjádřit***.
- Úplná teorie je rozhodnutelné generátorem teorémů/důkazů, ale většinou existuje i mnohem efektivnější algoritmus. Například pro Presburgerovu aritmetiku.

Složitost

Složitost algoritmů

❖ Základní teoretický přístup vychází z Church-Turingovy teze:

Každý algoritmus je implementovatelný jistým TS.

❖ Zavedení TS nám umožňuje klasifikovat problémy (resp. funkce) do dvou tříd:

1. problémy, jež nejsou algoritmicky ani částečně rozhodnutelné (resp. funkce algoritmicky nevyčíslitelné) a
2. problémy algoritmicky alespoň částečně rozhodnutelné (resp. funkce algoritmicky vyčíslitelné).

❖ Nyní se budeme zabývat třídou algoritmicky (částečně) rozhodnutelných problémů (vyčíslitelných funkcí) v souvislosti s otázkou složitosti jejich rozhodování (vyčíslování).

❖ Analýzu složitosti algoritmu budeme chápat jako analýzu složitosti výpočtů příslušného TS, jejímž cílem je vyjádřit (kvantifikovat) požadované zdroje (čas, prostor) jako funkci závisející na délce vstupního řetězce.

Různé případy při analýze složitosti

❖ Můžeme rozlišit:

1. analýzu složitosti nejhoršího případu,
2. analýzu složitosti nejlepšího případu,
3. analýzu složitosti průměrného případu,
4. amortizovanou analýzu

❖ Průměrná složitost algoritmu je definována následovně:

- Jestliže algoritmus (TS) vede k m různým výpočtům (případům) se složitostí c_1, c_2, ..., c_m, jež nastávají s pravděpodobností p_1, p_2, ..., p_m, pak průměrná složitost algoritmu je dána jako $\Sigma_{i=1}^{n} p_i c_i$.

❖ Amortizovaná analýza studuje posloupnost operací jako celek. Tato technika umožňuje, na rozdíl od klasického přístupu mnohem přesnější určení časové složitosti algoritmu.

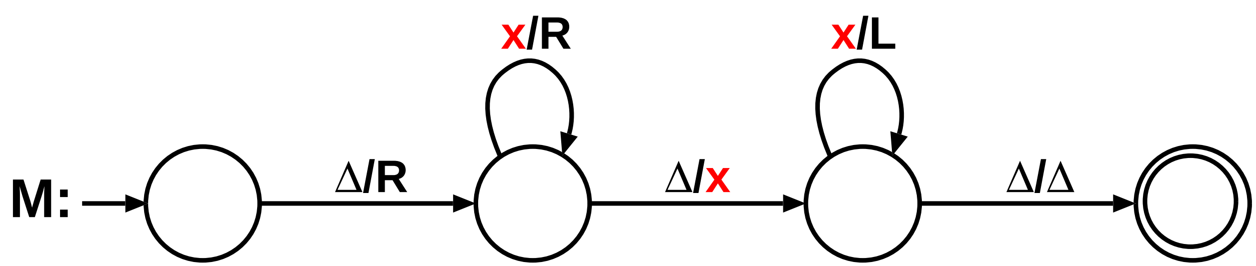
❖ Obvykle (alespoň na teoretické úrovni) se věnuje největší pozornost složitosti nejhoršího případu.

Složitost výpočtů TS

❖ Časová složitost – počet kroků (přechodů) TS provedený od počátku do konce výpočtu.

❖ Prostorová (paměťová) složitost – počet „buněk“ pásky TS požadovaný pro daný výpočet.

Příklad 12.1 Uvažme následující TS M:



Pro vstup $\Delta xxx\Delta\Delta$... je:

- časová složitost výpočtu M rovna 10,
- prostorová složitost výpočtu M rovna 5.

Lemma 12.1 Je-li časová složitost výpočtu prováděného TS rovna n, pak prostorová složitost tohoto výpočtu není větší než $n+1$.

Důkaz. Tvrzení je jednoduchou implikací plynoucí z definice časové a prostorové složitosti. $\square$

Definice 12.1 Řekneme, že k-páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v čase $T_M : \mathbb{N} \to \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. *může* přijmout) každé $w \in L$ v nanejvýš $T_M(|w|)$ krocích.

Definice 12.2 Řekneme, že k-páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v prostoru $S_M : \mathbb{N} \to \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. *může* přijmout) každé $w \in L$ při použití nanejvýš $S_M(|w|)$ buněk pásky – *nepočítáme zde buňky pásky, na nichž je zapsán vstup, ale nikdy na nich nespočine hlava stroje*.

- Zcela analogicky můžeme definovat vyčíslování určité funkce daným TS v určitém čase, resp. prostoru.

Analýza složitosti mimo prostředí TS

- V jiném výpočetním prostředí, než jsou TS, nemusí mít každá primitivní operace stejnou cenu.

- Velmi často se užívá tzv. uniformní cenové kritérium, kdy každé operaci přiřadíme stejnou cenu.

- Používá se ale např. také tzv. logaritmické cenové kritérium, kdy operaci manipulující operand o velikosti i, $i > 0$, přiřadíme cenu $\lfloor lg\ i \rfloor + 1$:
  - Zohledňujeme to, že s rostoucí velikostí operandů roste cena operací – logaritmus odráží růst velikosti s ohledem na binární kódování (v n bitech zakódujeme 2^n hodnot).

- Analýza složitosti za takových předpokladů není zcela přesná, důležité ale obvykle je to, aby se zachovala informace o tom, jak rychle roste čas/prostor potřebný k výpočtu v závislosti na velikosti vstupu.[a]

[a] Algoritmus A_1, jehož nároky rostou pomaleji než u jiného algoritmu A_2, nemusí být výhodnější než A_2 pro řešení malých instancí.

❖ Význam srovnávání rychlosti růstu složitosti výpočtů si snad nejlépe ilustrujeme na příkladu:

Příklad 12.2 Srovnání polynomiální (přesněji kvadratické – n^2) a exponenciální (2^n) časové složitosti:

délka vstupu n	časová složitost $c_1.n^2$, $c_1 = 10^{-6}$	časová složitost $c_2.2^n$, $c_2 \doteq 9.766.10^{-8}$
10	0.0001 s	0.0001 s
20	0.0004 s	0.1024 s
30	0.0009 s	1.75 min
40	0.0016 s	1.24 dne
50	0.0025 s	3.48 roku
60	0.0036 s	35.68 století
70	0.0049 s	3.65 mil. roků

Složitost výpočtů na TS a v jiných prostředích

❖ Pro rozumná cenová kritéria se ukazuje, že výpočetní složitost je pro různé modely výpočtu blízké běžným počítačům (RAM, RASP stroje) polynomiálně vázaná se složitostí výpočtu na TS (viz dále) a tudíž složitost výpočtu na TS není „příliš“ rozdílná oproti výpočtům na běžných počítačích.

- RAM stroje mají paměť s náhodným přístupem (jedna buňka obsahuje libovolné přirozené číslo), instrukce typu LOAD, STORE, ADD, SUB, MULT, DIV (akumulátor a konstanta/přímá adresa/nepřímá adresa), vstup/výstup, nepodmíněný skok a podmíněný skok (test akumulátoru na nulu), HALT. U RAM stroje je program součástí řízení stroje (okamžitě dostupný), u RASP je uložen v paměti stejně jako operandy.
- Funkce $f_1(n), f_2(n) : \mathbb{N} \to \mathbb{N}$ jsou polynomiálně vázané, existují-li polynomy $p_1(x)$ a $p_2(x)$ takové, že pro všechny hodnoty n je $f_1(n) \leq p_1(f_2(n))$ a $f_2(n) \leq p_2(f_1(n))$.
- Logaritmické cenové kritérium se uplatní, uvažujeme-li možnost násobení dvou čísel. Jednoduchým cyklem typu $A_{i+1} = A_i * A_i$ ($A_0 = 2$) jsme schopni počítat 2^{2^n}, což TS s omezenou abecedou není schopen provést v polynomiálním čase (pro uložení/načtení potřebuje projít 2^n buněk při použití binárního kódování).

❖ Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 12.3 Uvažme následující implementaci porovnávání řetězců.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i<n) {
      if (a[i] != b[i]) break;
      i++;
    }

    return (i==n);
}
```

- Pro určení složitosti aplikujme uniformní cenové kritérium např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- Složitost nejhoršího případu

❖ Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 12.4 Uvažme následující implementaci porovnávání řetězců.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i<n) {
      if (a[i] != b[i]) break;
      i++;
    }

    return (i==n);
}
```

- Pro určení složitosti aplikujme uniformní cenové kritérium např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- Složitost nejhoršího případu lze pak snadno určit jako $3n + 3$:
  - cyklus má 3 kroky, provede se n krát, tj. $3n$ kroků,
  - tělo funkce má 3 kroky (včetně testu ukončujícího cyklus).

Příklad 12.5 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i<n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
                a[j+1] = a[j];
                j = j - 1;
        }
        a[j+1] = value;
    }
}
```

- Pro určení složitosti aplikujme uniformní cenové kritérium např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- Složitost nejhoršího případu

Příklad 12.6 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i<n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
                a[j+1] = a[j];
                j = j - 1;
        }
        a[j+1] = value;
    }
}
```

- Pro určení složitosti aplikujme uniformní cenové kritérium např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- Složitost lze pak určit jako $1.5n^2 + 3.5n + 1$:
  - vnitřní cyklus má 3 kroky, provede se 0, 1, ..., $n-1$ krát, tj. $3(0+1+...+n-1) = 3\frac{n}{2}(0+n-1) = 1.5n^2 - 1.5n$ kroků,
  - vnější cyklus má mimo vnitřní cyklus 5 kroků (včetně testu ukončujícího vnitřní cyklus) a provede se n krát, tj. $5n$ kroků,
  - jeden krok připadá na ukončení vnějšího cyklu.

Asymptotická omezení složitosti

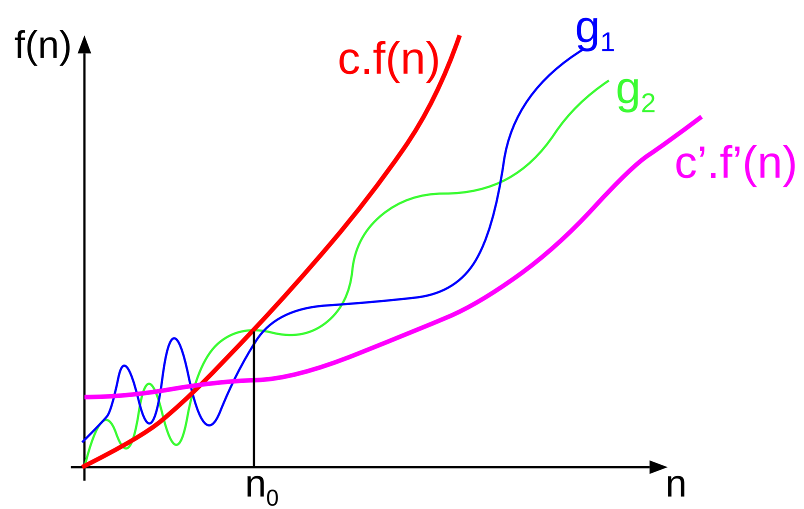
❖ Při popisu složitosti algoritmů (výpočtů TS), chceme často vyloučit vliv aditivních a multiplikativních konstant:

- různé aditivní a multiplikativní konstanty vzniknou velmi snadno „drobnými“ úpravami uvažovaných algoritmů,
- např. srovnávání dvou řetězců je možné donekonečna zrychlovat tak, že budeme porovnávat současně 2, 3, 4, ... za sebou jdoucích znaků,
- lineární komprese prostoru (rozšíření abecedy) a zrychlení výpočtu (před-vypočítání výsledků)
- tyto úpravy ovšem nemají zásadní vliv na rychlost nárůstu časové složitosti (ve výše uvedeném případě nárůst zůstane kvadratický),
- navíc při analýze složitosti mimo prostředí TS se dopouštíme jisté nepřesnosti již zavedením různých cenových kritérií.

❖ Proto se často složitost popisuje pomocí tzv. asymptotických odhadů složitosti – viz další stránka.

Definice 12.3 Nechť $\mathcal{F}$ je množina funkcí $f : \mathbb{N} \to \mathbb{N}$. Pro danou funkci $f \in \mathcal{F}$ definujeme množiny funkcí $O(f(n))$, $\Omega(f(n))$ a $\Theta(f(n))$ takto:

- Asymptotické horní omezení funkce $f(n)$ je množina **PISOMKA**
  $O(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c.f(n)\}$.
- Asymptotické dolní omezení funkce $f(n)$ je množina
  $\Omega(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c.f(n) \leq g(n)\}$.
- Asymptotické oboustranné omezení funkce $f(n)$ je množina $\Theta(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c_1, c_2 \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c_1.f(n) \leq g(n) \leq c_2.f(n)\}$.



Příklad 12.7 S využitím asymptotických odhadů složitosti můžeme říci, že složitost našeho srovnání řetězců patří do $O(n)$ a složitost insert-sort do $O(n^2)$.

Amortizovaná složitost – ukázka

- Příklad: dynamicky alokovaná tabulka T (odebraní záznamu zneplatní řádek)
  - neznámá velikost – dynamická realokace mění kapacitu
  - při naplnění tabulky alokujeme větší tabulku a složky do ní přesuneme
  - při vyprázdnění na určitou mez alokujeme menší tabulku a složky do ní přesuneme
  - $\alpha(T)$ je poměr platných položek (velikost) ku kapacitě tabulky
  - pokud je $\alpha(T) = 1$ vložení prvku vede k alokaci 2-krát větší tabulky a přesunu
  - pokud je $\alpha(T) \leq 0.5$ odebrání prvku vede k alokaci 2-krát menší tabulky a přesunu
- Složitost nejhoršího případu
  - složitost jedné operace vložení i odebrání je rovna n (velikost tabulky)
  - složitost n operací je v nejhorším případě $O(n^2)$
- Amortizovaná složitost
  - n operací vložení:
    - c_i – cena i-té operace vložení
    - $c_i = i$ pokud $i - 1$ je mocninou 2 , jinak $c_i = 1$
    - $\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n < 3n$

Amortizovaná složitost – ukázka 2

- existuje posloupnost n operací (vložení, odebrání) se složitostí $\Theta(n^2)$
  - střídavě přidáváme odebíráme prvky kolem faktoru $\alpha(T) = 0.5$
  - každá třetí operace má cenu n

❖ Existuje implementace s lineární amortizovanou složitostí?

- zabráníme tomu, že kapacita může oscilovat mezi dvěma hodnotami při malém počtu operací (viz výše)
- pokud je $\alpha(T) \leq 0.25$ odebrání prvku vede k alokaci 2-krát menšího pole a přesunu
- určete amortizovanou složitost libovolných n operací (hlavní myšlenka)
  - $2^k + 1$ operací vložení, kapacita 2^{k+1}, cena $< 3(2^k + 1)$
  - realokaci vyvolá $2^{k-1} + 1$ operací odebrání, kapacita je 2^k, cena $2^{k-1} + 2^k + 1$
  - další realokaci vyvolá $2^{k-1} + 1$ operací vložení, kapacita 2^{k+1}, cena $2^{k-1} + 2^k + 1$
  - celkově operací: $2^k + 1 + 2 * 2^{k-1} + 2 = 2^{k+1} + 3$
  - celkově cena: $2 * 2^{k-1} + 5 * 2^k + 5 = 6 * 2^k + 5 = 3 * 2^{k+1} + 5$
  - ukázali jsem, že n operací má cenu $O(n)$

Třídy složitosti

Složitost problémů

❖ Přejdeme nyní od složitosti konkrétních algoritmů (Turingových strojů) ke složitosti problémů.

❖ Třídy složitosti zavádíme jako prostředek ke kategorizaci (vytvoření hierarchie) problémů dle jejich složitosti, tedy dle toho, jak dalece efektivní algoritmy můžeme navrhnout pro jejich rozhodování (u funkcí můžeme analogicky mluvit o složitosti jejich vyčíslování).

❖ Podobně jako u určování typu jazyka se budeme snažit zařadit problém do co nejnižší třídy složitosti – tedy určit složitost problému jako složitost jeho nejlepšího možného řešení.

- Omezením této snahy je to, že (jak uvidíme) existují problémy, jejichž řešení je možné do nekonečna *významně* zrychlovat.

❖ Zařazení různých problémů do téže třídy složitosti může odhalit různé vnitřní podobnosti těchto problémů a může umožnit řešení jednoho převodem na druhý (a pragmatické využití např. různých již vyvinutých nástrojů).

Třídy složitosti

Definice 12.4 Mějme dány funkce $t, s : \mathbb{N} \to \mathbb{N}$ a nechť T_M, resp. S_M, značí časovou, resp. prostorovou, složitost TS M. Definujeme následující časové a prostorové třídy složitosti deterministických a nedeterministických TS:

- $DTime[t(n)] = \{L \mid \exists\ k$-páskový DTS $M : L = L(M)$ a $T_M \in O(t(n))\}$.
- $NTime[t(n)] = \{L \mid \exists\ k$-páskový NTS $M : L = L(M)$ a $T_M \in O(t(n))\}$.
- $DSpace[s(n)] = \{L \mid \exists\ k$-páskový DTS $M : L = L(M)$ a $S_M \in O(s(n))\}$.
- $NSpace[s(n)] = \{L \mid \exists\ k$-páskový NTS $M : L = L(M)$ a $S_M \in O(s(n))\}$.

❖ Definici tříd složitosti pak přímočaře zobecňujeme tak, aby mohly být založeny na množině funkcí, nejen na jedné konkrétní funkci.

❖ *Poznámka*: Dále ukážeme, že použití více pásek přináší jen polynomiální zrychlení. Na druhou stranu ukážeme, že zatímco nedeterminismus nepřináší nic podstatného z hlediska vyčíslitelnosti, může přinášet mnoho z hlediska složitosti.

Časově/prostorově zkonstruovatelné funkce

❖ Třídy složitosti obvykle budujeme nad tzv. časově/prostorově zkonstruovatelnými funkcemi:

- Důvodem zavedení časově/prostorově zkonstruovatelných funkcí je dosáhnout intuitivní hierarchické struktury tříd složitosti – např. odlišení tříd $f(n)$ a $2^{f(n)}$, což, jak uvidíme, pro třídy založené na obecných rekurzívních funkcích nelze.

Definice 12.5 Funkci $t : \mathbb{N} \to \mathbb{N}$ nazveme časově zkonstruovatelnou (time constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví po přesně $t(|w|)$ krocích.

Definice 12.6 Funkci $s : \mathbb{N} \to \mathbb{N}$ nazveme prostorově zkonstruovatelnou (space constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví s využitím přesně $s(|w|)$ buněk pásky.

❖ *Poznámka*: Pokud je jazyk L nad Σ přijímán strojem v čase/prostoru omezeném časově/prostorově zkonstruovatelnou funkcí, pak je také přijímán strojem, který pro každé $w \in \Sigma^*$ vždy zastaví:

- U časového omezení $t(n)$ si stačí předem spočíst, jaký je potřebný počet kroků a zastavit po jeho vyčerpání.
- U prostorového omezení spočteme z $s(n)$, $|Q|$, $|\Delta|$ maximální počet konfigurací, které můžeme vidět a z toho také plyne maximální počet kroků, které můžeme udělat, aniž bychom cyklili.

Nejběžněji užívané třídy složitosti

- Deterministický/nedeterministický polynomiální čas:

$$\mathbf{P} = \bigcup_{k=0}^{\infty} DTime(n^k) \qquad \mathbf{NP} = \bigcup_{k=0}^{\infty} NTime(n^k)$$

- Deterministický/nedeterministický polynomiální prostor:

$$\mathbf{PSPACE} = \bigcup_{k=0}^{\infty} DSpace(n^k) \qquad \equiv \qquad \mathbf{NPSPACE} = \bigcup_{k=0}^{\infty} NSpace(n^k)$$

- *Poznámka*: Problémy ze třídy $\mathbf{PSPACE}$ se často reálně neřeší v polynomiálním prostoru – zvyšují se prostorové nároky výměnou za alespoň částečné snížení časových nároků (např. z $O(2^{n^2})$ na $O(2^n)$ u LTL model checkingu apod.). Intuitivně: je možno si pamatovat více mezivýsledků a není třeba je znovu spočíst.

Třídy pod a nad polynomiální složitostí

- Deterministický/nedeterministický logaritmický prostor:

$$\mathbf{LOGSPACE} = \bigcup_{k=0}^{\infty} DSpace(k\, lg\, n) \qquad \mathbf{NLOGSPACE} = \bigcup_{k=0}^{\infty} NSpace(k\, lg\, n)$$

- Deterministický/nedeterministický exponenciální čas:

$$\mathbf{EXP} = \bigcup_{k=0}^{\infty} DTime(2^{n^k}) \qquad \mathbf{NEXP} = \bigcup_{k=0}^{\infty} NTime(2^{n^k})$$

- Deterministický/nedeterministický exponenciální prostor:

$$\mathbf{EXPSPACE} = \bigcup_{k=0}^{\infty} DSpace(2^{n^k}) \quad \equiv \quad \mathbf{NEXPSPACE} = \bigcup_{k=0}^{\infty} NSpace(2^{n^k})$$

Třídy nad exponenciální složitostí

- Det./nedet. k-exponenciální čas/prostor založený na věži exponenciál $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ o výšce k:

$$\textbf{k-EXP} = \bigcup_{l=0}^{\infty} DTime(2^{2^{\cdot^{\cdot^{\cdot^{2^{n^l}}}}}}) \qquad \textbf{k-NEXP} = \bigcup_{l=0}^{\infty} NTime(2^{2^{\cdot^{\cdot^{\cdot^{2^{n^l}}}}}})$$

$$\textbf{k-EXPSPACE} = \bigcup_{l=0}^{\infty} DSpace(2^{2^{\cdot^{\cdot^{\cdot^{2^{n^l}}}}}}) \equiv \textbf{k-NEXPSPACE} = \bigcup_{l=0}^{\infty} NSpace(2^{2^{\cdot^{\cdot^{\cdot^{2^{n^l}}}}}})$$

$$\textbf{ELEMENTARY} = \bigcup_{k=0}^{\infty} \textbf{k-EXP}$$

Vrchol hierarchie tříd složitosti

❖ Na vrcholu hierarchie tříd složitosti se pak hovoří o obecných třídách jazyků (funkcí), se kterými jsme se již setkali:

- třída primitivně-rekurzívních funkcí **PR** (implementovatelných pomocí zanořených cyklů s pevným počtem opakování – `for i=... to ...`),
- třída $\mu-$rekurzívních funkcí (rekurzívních jazyků) **R** (implementovatelných pomocí cyklů s předem neurčeným počtem opakování – `while ...`) a
- třída rekurzívně vyčíslitelných funkcí (rekurzívně vyčíslitelných jazyků) **RE**.

Vlastnosti tříd složitosti

k-páskové Turingovy stroje

Věta 12.1 Je-li jazyk L přijímán nějakým k-páskovým DTS M_k v čase $t(n)$, pak je také přijímán nějakým 1-páskovým DTS M_1 v čase $O(t(n)^2)$.

Důkaz. (idea)

- Obsah k pásek stroje M_k můžeme zapsat na jednu pásku stroje M_1 za sebe a oddělit je vhodným oddělovačem. Pozici hlav M_k na pásce můžeme zakódovat vhodným označením symbolů, pod kterými se hlavy aktuálně nacházejí.
- Při simulaci kroku stroje M_k stroj M_1 dvakrát projde páskou – při jednom průchodu zjistí znaky pod hlavami M_k, při druhém je patřičně upraví.
- Jestliže je na některé simulované pásce stroje M_k nedostatek prostoru, je zapotřebí provést posun zbytku pásky stroje M_1, což si může opět vyžádat dva průchody touto páskou.
- Užitečná délka pásek stroje M_k je ovšem omezena na $t(n)$ a tudíž užitečná délka pásky stroje M_1 je omezena na $k.t(n) + k$, tj. $O(t(n))$.
- Simulace jednoho kroku M_k strojem M_1 je proto v $O(t(n))$.
- Takových kroků se provede $t(n)$ a tudíž M_1 přijímá L v čase $O(t(n)^2)$.

$\square$

Determinismus a nedeterminismus

Věta 12.2 Je-li jazyk L přijímán nějakým NTS M_n v čase $t(n)$, pak je také přijímán nějakým DTS M_d v čase $2^{O(t(n))}$.

Důkaz. (idea) Ukážeme, jak může M_d simulovat M_n v uvedeném čase:

- Očíslujeme si přechody M_n jako $1, 2, ..., k$.
- M_d bude postupně simulovat veškeré možné posloupnosti přechodů M_n (obsah vstupní pásky si uloží na pomocnou pásku, aby ho mohl vždy obnovit; na jinou pásku si vygeneruje posloupnost přechodů z $\{1, 2, ..., k\}^*$ a tu simuluje).
- Vzhledem k možnosti nekonečných výpočtů M_n nelze procházet jeho možné výpočty do hloubky – budeme-li je ale procházet do šířky (tj. nejdřív všechny řetězce z $\{1, 2, ..., k\}^*$ délky 1, pak 2, pak 3, ...), určitě nalezneme nejkratší přijímající posloupnost přechodů pro M_n, existuje-li.
- Takto projdeme nanejvýš $O(k^{t(n)})$ cest, simulace každé z nich je v $O(t(n))$ a tudíž celkem využijeme nanejvýš čas $O(k^{t(n)})O(t(n)) = 2^{O(t(n))}$.

$\square$

❖ Doposud nikdo nebyl schopen přijít s polynomiální simulací NTS pomocí DTS. **Zdá se tedy, že nedeterminismus přináší značnou výhodu z hlediska časové složitosti výpočtů.**

❖ Zatímco *se zdá*, že nedeterminismus přináší značnou výhodu z hlediska časové složitosti výpočtů, v případě prostorové složitosti je situace jiná:

Věta 12.3 (Savitchův teorém) $NSpace[s(n)] \subseteq DSpace[s^2(n)]$ pro každou prostorově zkonstruovatelnou funkci $s(n) \geq lg\ n$.

Důkaz. Uvažme NTS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ rozhodující $L(M)$ v prostoru $s(n)$:

- Existuje $k \in \mathbb{N}$ závislé jen na $|Q|$ a $|\Gamma|$ takové, že pro libovolný vstup w, M projde nanejvýš $k^{s(n)}$ konfigurací o délce max. $s(n)$.
- To implikuje, že M provede pro w nanejvýš $k^{s(n)} = 2^{s(n) lg\ k}$ kroků.
- Pomocí DTS můžeme snadno implementovat proceduru $test(c, c', i)$, která otestuje, zda je možné v M dojít z konfigurace c do c' v 2^i krocích:

  procedure $test(c, c', i)$

  if $(i = 0)$ **then return** $((c = c') \vee (c \underset{M}{\vdash} c'))$

  else for all configurations c'' such that $|c''| \leq s(n)$ **do**

      if $(test(c, c'', i-1) \wedge test(c'', c', i-1))$ **then return** $true$

  return $false$

Důkaz pokračuje dále.

Pokračování důkazu.

- Všimněme si, že rekurzivním vyvoláváním $test$ vzniká strom o výšce i simulující posloupností svých listů posloupnost 2^i výpočetních kroků.
- Nyní k deterministické simulaci M postačí projít všechny akceptující konfigurace c_F takové, že $|c_F| \leq s(n)$, a ověřit, zda $test(c_o, c_f, \lceil s(n)lg\ k \rceil)$, kde c_0 je počáteční konfigurace.
- Každé vyvolání $test$ zabere prostor $O(s(n))$, hloubka rekurze je $\lceil s(n)lg\ k \rceil = O(s(n))$ a tedy celkově deterministicky simulujeme M v prostoru $O(s^2(n))$.
- Dodejme, že $s(n)$ může být zkonstruováno v prostoru $O(s(n))$ (jedná se o prostorově zkonstruovatelnou funkci) a tudíž neovlivňuje výše uvedené úvahy.

$\square$

❖ Důsledkem Savitchova teorému jsou již dříve uvedené rovnosti:

- **PSPACE** $\equiv$ **NPSPACE**,
- **k-EXPSPACE** $\equiv$ **k-NEXPSPACE**.

Prostor kontra čas

❖ Intuitivně můžeme říci, že zatímco prostor může růste relativně pomalu, čas může růst výrazně rychleji, neboť můžeme opakovaně procházet týmiž buňkami pásky – opačně tomu být zřejmě nemůže (nemá smysl mít nevyužitý prostor).

Věta 12.4 *$NSpace[t(n)] \subseteq DTime[O(1)^{t(n)}]$ pro každou časově zkonstruovatelnou funkci $t(n) \geq lg\ n$.

Důkaz. Dá se použít do jisté míry podobná konstrukce jako u Savitchova teorému – blíže viz literatura. $\square$ *

Věty o kompresi prostoru a zrychlení

Věta 12.5 Nechť M je DTS. Pak existuje ekvivalentní TS N takový, že

$$S_N(n) \leq \frac{S_M(n)}{2} + 2$$

Důkaz. (Idea) Abeceda stroje N obsahuje dvojice znaků abecedy stroje M. Obsah pásky $a_1, a_2, \ldots, a_n$ stroje M pak bude reprezentován jako

$$\binom{a_1}{a_2}, \binom{a_2}{a_3}, \ldots$$

$\square$

Věta 12.6 Nechť M je DTS. Pak existuje ekvivalentní TS N takový, že

$$T_N(n) \leq \frac{T_M(n)}{2} + 2n$$

Důkaz. (Idea) Stroj N si předpočítá výsledky všech možných výpočtů délky 2 stroje M.

$\square$

Uzavřenost vůči doplňku

❖ Doplňkem třídy rozumíme třídu jazyků, které jsou doplňkem jazyků dané třídy. Tedy označíme-li doplněk třídy $\mathcal{C}$ jako co-$\mathcal{C}$, pak $L \in \mathcal{C} \Leftrightarrow \overline{L} \in co\text{-}\mathcal{C}$. U rozhodování problémů toto znamená rozhodování komplementárního problému (prázdnost x neprázdnost apod.).

❖ Prostorové třídy jsou obvykle uzavřeny vůči doplňku:

Věta 12.7 *Jestliže $s(n) \geq lg\ n$, pak $NSpace(s(n)) = co\text{-}NSpace(s(n))$.

Důkaz. Jedná se o teorém Immermana a Szelepcsényiho – důkaz viz literatura. □ *

❖ Pro časové třídy je situace jiná:

- Některé třídy jako **P** či **EXP** jsou uzavřeny vůči doplňku.
- U jiných významných tříd zůstává otázka uzavřenosti vůči doplňku otevřená. Proto má smysl hovořit např. i o třídách jako:
  - **co-NP** či
  - **co-NEXP**.

Věta 12.8 Třída **P** je uzavřena vůči doplňku.

Důkaz. (idea) Základem je to, že ukážeme, že jestliže jazyk L nad Σ může být přijat DTS M v polynomiálním čase, pak také existuje DTS M', který L rozhoduje v polynomiálním čase, tj. $L = L(M')$ a existuje $k \in \mathbb{N}$ takové, že pro každé $w \in \Sigma^*$ M' zastaví v čase $O(|w|^k)$:

- M' na začátku výpočtu určí délku vstupu w a vypočte $p(|w|)$, kde $p(n)$ je polynom určující složitost přijímání strojem M. Na speciální dodatečnou pásku uloží $p(|w|)$ symbolů.
- Následně M' simuluje M, přičemž za každý krok umaže jeden symbol z dodatečné pásky. Pokud odebere z této pásky všechny symboly a M by mezitím nepřijal, M' abnormálně ukončí výpočet (odmítne).
- M' evidentně přijme všechny řetězce, které přijme M na to mu stačí $p(n)$ simulovaných kroků a nepřijme všechny řetězce, které by M nepřijal – pokud M nepřijme v $p(n)$ krocích, nepřijme vůbec. M' však vždy zastaví v $O(p(n))$ krocích.

□

Ostrost hierarchie tříd

❖ Z dosavadního můžeme shrnout, že platí následující:

- **LOGSPACE** $\subseteq$ **NLOGSPACE** $\subseteq$ **P** $\subseteq$ **NP**
- **NP** $\subseteq$ **PSPACE** $=$ **NPSPACE** $\subseteq$ **EXP** $\subseteq$ **NEXP**
- **NEXP** $\subseteq$ **EXPSPACE** $=$ **NEXPSPACE** $\subseteq$ **2-EXP** $\subseteq$ **2-NEXP** $\subseteq$ ...
- ... $\subset$ **ELEMENTARY** $\subset$ **PR** $\subset$ **R** $\subset$ **RE** [a]

❖ Řada otázek ostrosti uvedených vztahů pak zůstává otevřená, nicméně z tzv. teorému hierarchie (nebudeme ho zde přesně uvádět, neboť je velmi technický – zájemci ho naleznou v literatuře) plyne, že exponenciální „mezery“ mezi třídami jsou „ostré“:

- **LOGSPACE**, **NLOGSPACE** $\subset$ **PSPACE**,
- **P** $\subset$ **EXP**,
- **NP** $\subset$ **NEXP**,
- **PSPACE** $\subset$ **NEXPSPACE**,
- **EXP** $\subset$ **2-EXP**, ...

[a] Bez důkazu jsme doplnili, že **ELEMENTARY** $\subset$ **PR**.

Jazyky $\mathcal{C}$-těžké a $\mathcal{C}$-úplné

❖ Až doposud jsme třídy používali jako horní omezení složitosti problémů. Všimněme si nyní omezení dolního – to zavedeme pomocí redukovatelnosti třídy problémů na daný problém.

Definice 12.7 Nechť $\mathcal{R}$ je třída funkcí. Jazyk $L_1 \subseteq \Sigma_1^*$ je $\mathcal{R}$ redukovatelný (přesněji $\mathcal{R}$ many-to-one reducible) na jazyk $L_2 \subseteq \Sigma_2^*$, což zapisujeme $L_1 \leq_{\mathcal{R}}^m L_2$, jestliže existuje funkce f z $\mathcal{R}$ taková, že $\forall w \in \Sigma_1^* : w \in L_1 \Leftrightarrow f(w) \in L_2$.

Definice 12.8 Nechť $\mathcal{R}$ je třída funkcí a $\mathcal{C}$ třída jazyků. Jazyk L_0 je $\mathcal{C}$-těžký ($\mathcal{C}$-hard) vzhledem k $\mathcal{R}$ redukovatelnosti, jestliže $\forall L \in \mathcal{C} : L \leq_{\mathcal{R}}^m L_0$.

Definice 12.9 Nechť $\mathcal{R}$ je třída funkcí a $\mathcal{C}$ třída jazyků. Jazyk L_0 nazveme $\mathcal{C}$-úplný ($\mathcal{C}$-complete) vzhledem k $\mathcal{R}$ redukovatelnosti, jestliže $L_0 \in \mathcal{C}$ a L_0 je $\mathcal{C}$-těžký ($\mathcal{C}$-hard) vzhledem k $\mathcal{R}$ redukovatelnosti.

❖ Pro třídy $\mathcal{C}_1 \subseteq \mathcal{C}_2$ a jazyk L, jenž je $\mathcal{C}_2$-úplný vůči $\mathcal{R}$ redukovatelnosti, platí, že buď $\mathcal{C}_2$ je celá $\mathcal{R}$ redukovatelná na $\mathcal{C}_1$, nebo $L \in \mathcal{C}_2 \setminus \mathcal{C}_1$.

Nejběžnější typy úplnosti

❖ Uveďme nyní nejběžněji používané typy úplnosti – všimněme si, že je použita redukovatelnost dostatečně silná na to, aby bylo možné najít úplné problémy vůči ní a na druhou stranu nebyly příslušné třídy triviálně redukovány na jejich (možné) podtřídy:

- **NP**, **PSPACE**, **EXP** úplnost je definována vůči polynomiální redukovatelnosti (tj. redukovatelnosti pomocí DTS pracujících v polynomiálním čase),
- **P**, **NLOGSPACE** úplnost definujeme vůči redukovatelnosti v deterministickém logaritmickém prostoru,
- **NEXP** úplnost definujeme vůči exponenciální redukovatelnosti (tj. redukovatelnosti pomocí DTS pracujících v exponenciálním čase).

Polynomiální redukce

Definice 12.10 *Polynomiální redukce* jazyka L_1 nad abecedou Σ_1 na jazyk L_2 nad abecedou Σ_2 je funkce $f : \Sigma_1^* \to \Sigma_2^*$, pro kterou platí:

1. $\forall w \in \Sigma_1^* : w \in L_1 \Leftrightarrow f(w) \in L_2$
2. f je vyčíslitelná DTS v polynomiálním čase.

Existuje-li polynomiální redukce jazyka L_1 na L_2, říkáme, že L_1 se *(polynomiálně) redukuje na* L_2 a píšeme $L_1 \leq_P^m L_2$.

Věta 12.9 Je-li $L_1 \leq_P^m L_2$ a L_2 je ve třídě P, pak L_1 je ve třídě P.

Důkaz. Nechť M_f je Turingův stroj, který provádí redukci f jazyka L_1 na L_2 a nechť $p(x)$ je jeho časová složitost. Pro libovolné $w \in L_1$ výpočet $f(w)$ vyžaduje nanejvýš $p(|w|)$ kroků a produkuje výstup maximální délky $p(|w|) + |w|$.
Nechť M_2 přijímá jazyk L_2 v polynomiálním čase daném polynomem $q(x)$.
Uvažujme Turingův stroj, který vznikne kompozicí $M_f M_2$. Tento stroj přijímá jazyk L_1 tak, že pro každé $w \in L_1$ udělá stroj $M_f M_2$ maximálně $p(|w|) + q(p(|w|) + |w|)$ kroků, což je polynomem ve $|w|$ a tedy L_1 leží ve třídě P. $\square$

Příklady složitosti problémů

- Příklady **LOGSPACE** problémů:
  - existence cesty mezi dvěma uzly v neorientovaném grafu.

- Příklady **NLOGSPACE**-úplných problémů:
  - existence cesty mezi dvěma uzly v orientovaném grafu,
  - 2-SAT (*SATisfiability*), tj. splnitelnost výrokových formulí tvaru konjunkce disjunkcí dvou literálů (literál je výroková proměnná nebo její negace), např. $(x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$.

- Příklady **P**-úplných problémů:
  - splnitelnost konjunkce Hornových klauzulí $(p \wedge q \wedge ... \wedge t) \Rightarrow u$, kde p, q, ... jsou atomické formule výrokové logiky (výrokové proměnné),
    - nerozhodnutelné v predikátové podobě, částečně rozhodnutelná nesplnitelnost,
  - náležitost řetězce do jazyka bezkontextové gramatiky: algoritmus „CYK“ založený na dynamickém programování (Cocke, Younger, Kasami) – $O(n^3)$,
  - topologické uspořádání – pořadí uzlů při průchodu grafem do hloubky (pro dané řazení přímých následníků).

- Příklady $\mathbf{NP}$-úplných problémů:
  - 3-SAT a obecný SAT – viz dále,
  - řada grafových problémů, např.:
    - existence kliky dané velikosti,
    - existence Hamiltonovské kružnice v neorientovaném grafu,
    - existence orientované Hamiltonovské kružnice v orientovaném grafu,
    - barvitelnost – lze daný neorientovaný graf obarvit určitým počtem barev?,
    - uzlové pokrytí neorientovaného grafu množinou uzlů o určité velikosti (tj. množinou uzlů, se kterou souvisí všechny hrany),
    - ...
  - problém obchodního cestujícího,
  - „knapsack" – máme položky s váhou a hodnotou, maximalizujeme hodnotu tak, aby váha nepřekročila určitou mez.

- Příklady co-$\mathbf{NP}$-úplných problémů:
  - ekvivalence regulárních výrazů bez iterace.

❖ Příklady **PSPACE**-úplných problémů:

- ekvivalence regulárních výrazů,
- náležitost řetězce do jazyka kontextové gramatiky,
- inkluze jazyků nedeterministických konečných automatů,
- model checking formulí lineární temporální logiky (LTL – výroková logika doplněná o temporální operátory *until*, *always*, *eventually*, *next-time*) vůči velikosti formule.

❖ Příklady **EXP**-úplných problémů:

- inkluze jazyka bezkontextové gramatiky v jazyce NKA (opačná $\subseteq$ nerozh.),
- inkluze nedeterministických konečných *stromových automatů*, zobecňujících přechody KA do podoby $p \xrightarrow{a} (q_1, \ldots, q_n)$,
- inkluze pro tzv. *visibly push-down* jazyky (operace push/pop, které provádí přijímající automat, jsou součástí vstupního řetězce),
- nejlepší tah v šachu (zobecněno na šachovnici $n \times n$),
- model checking procesů s neomezeným zásobníkem (rekurzí) vůči zafixované formuli logiky větvícího se času (CTL) – tj. EXP ve velikosti procesu.

❖ Příklady **EXPSPACE**-úplných problémů:

- ekvivalence regulárních výrazů doplněných o operaci kvadrát (tj. r^2).

- **k-EXP** / **k-EXPSPACE**:
  - rozhodování splnitelnosti formulí Presburgerovy aritmetiky – tj. celočíselné aritmetiky se sčítáním, porovnáváním (ne násobením – to vede na tzv. Peanovu aritmetiku, která je již nerozhodnutelná) a kvantifikací prvního řádu (např. $\forall x, y : x \leq x + y$) je problém, který je v **3-EXP** (**2-EXPSPACE**-úplný).

- Problémy mimo **ELEMENTARY**:
  - ekvivalence regulárních výrazů doplněných o negaci,
  - rozhodování splnitelnosti formulí logiky WS1S – celočíselná aritmetika s operací $+1$ a kvantifikací prvního a druhého řádu (tj. pro každou/existuje hodnota, resp. množina hodnot, taková, že ... – např. $\exists A : \forall x \in A : x + 1 \notin A$),
  - verifikace dosažitelnosti v tzv. *Lossy Channel Systems* – procesy komunikující přes neomezenou, ale ztrátovou frontu (cokoliv se může kdykoliv ztratit).

Prvočíselný rozklad

- Problém rozkladu daného celého čísla na součin prvočísel.
- Velmi důležitý problém pro kryptografii.
- Ví se, že je v **NP** $\cap$ **co-NP**.
- Neví se, zda je v **P**, ani zda je **NP**-úplný.
- Existuje polynomiální algoritmus pro kvantové počítače.

SAT-problém

Problém splnitelnosti – SAT problém

❖ Nechť $V = \{v_1, v_2, \ldots, v_m\}$ je konečná množina Booleovských proměnných (prvotních formulí výrokového počtu). *Literálem* nazveme každou proměnnou v_i nebo její negaci $\overline{v_i}$. *Klausulí* nazveme výrokovou formuli obsahující pouze literály spojené výrokovou spojkou $\vee$ (nebo).

Příklady klausulí: $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$.

❖ *SAT-problém* lze formulovat takto: Je dána množina proměnných V a množina klausulí nad V. Je tato množina klausulí splnitelná?

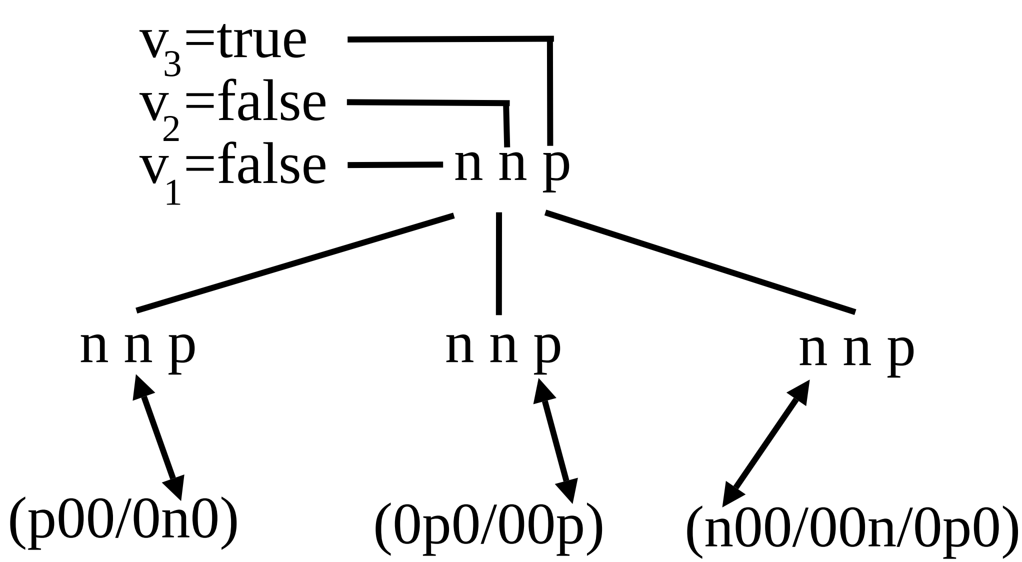
❖ Každý konkrétní SAT-problém můžeme zakódovat jediným řetězcem takto: Nechť $V = \{v_1, v_2, \ldots, v_m\}$, každý literál v_i zakódujeme řetězcem délky m, který obsahuje samé 0 s výjimkou i-té pozice, která obsahuje symbol p, jde-li o literál v_i, nebo n, jde-li o literál $\overline{v_i}$. Klausuli reprezentujeme seznamem zakódovaných literálů oddělených symbolem $/$. SAT-problém bude seznam klausulí uzavřených v aritmetických závorkách.

Příklad 12.8 SAT-problém obsahuje proměnné v_1, v_2, v_3 a klausule $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$ bude reprezentována řetězcem: $(p00/0n0)(0p0/00p)(n00/00n/0p0)$.

❖ Označme L_{SAT} jazyk obsahující řetězce tohoto typu, které reprezentují splnitelné množiny klausulí.
Řetězec $(p00/0n0)(0p0/00p)(n00/00n/0p0)$ je prvkem L_{SAT} $(v_1 = F, v_2 = F, v_3 = T)$, na rozdíl od řetězce $(p00/0p0)(n00/0p0)(p00/0n0)(n00/0n0)$, který je kódem nesplnitelné množiny klausulí $v_1 \vee v_2$, $\overline{v_1} \vee v_2$, $v_1 \vee \overline{v_2}$, $\overline{v_1} \vee \overline{v_2}$.

❖ Přiřazení pravdivostních hodnot budeme reprezentovat řetězcem z $\{p, n\}^+$, kde p v i-té pozici představuje přiřazení $v_i \approx$ true a n v i-té pozici představuje přiřazení $v_i \approx$ false.

❖ Pak test, zda určité hodnocení je modelem množiny klausulí (množina klausulí je pro toto hodnocení splněna), je velmi jednoduchý a ilustruje ho obrázek:



❖ Na uvedeném principu můžeme zkonstruovat nedeterministický Turingův stroj, který přijímá jazyk L_{SAT} v polynomiálním čase. Zvolíme 2-páskový Turingův stroj, který:

1. začíná kontrolou, zda vstup reprezentuje množinu klausulí,
2. na $2.$ pásku nageneruje řetězec z $\{n, p\}^m$ nedeterministickým způsobem,
3. posouvá hlavu na $1.$ pásce a testuje, zda pro dané ohodnocení (na $2.$ pásce) je množina klausulí splnitelná.

Tento proces může být snadno implementován s polynomiální složitostí přijetí v závislosti na délce vstupního řetězce a tedy $L_{SAT} \in NP$.

❖ Princip „guess & check“ použitý výše se často užívá při ukázání členství v **NP**.

Věta 12.10 *Cookův teorém:* Je-li L libovolný jazyk z NP, pak $L \leq_P^m L_{SAT}$.

Hlavní myšlenka důkazu: Protože $L \in NP$, existuje nedeterministický Turingův stroj M a polynom $p(x)$ tak, že pro každé $w \in L$ stroj M přijímá w v maximálně $p(|w|)$ krocích. Jádro důkazu tvoří konstrukce polynomiální redukce f z L na L_{SAT}: Pro každý řetězec $w \in L$ bude $f(w)$ množina klausulí, které jsou splnitelné, právě když M přijímá w.

***NP**-úplné jazyky*

❖ Po objevení Cookova teorému se ukázalo, že mnoho dalších **NP** jazyků má vlastnost podobnou jako L_{SAT}, t.j. jsou polynomiálními redukcemi ostatních **NP** jazyků.

❖ Tyto jazyky se – jak již víme – nazývají ***NP**-úplné* (**NP**-complete) jazyky.

❖ Kdyby se ukázalo, že libovolný z těchto jazyků je v **P**, pak by muselo platit **P** $=$ **NP**; naopak důkaz, že některý z nich leží mimo **P** by znamenalo **P** $\subset$ **NP**.

Význačné NP-úplné problémy

- *Satisfiability*: Je boolovský výraz splnitelný?
- *Clique*: Obsahuje neorientovaný graf kliku velikosti k?
- *Vertex cover*: Má neorientovaný graf dominantní množinu mohutnosti k?
- *Hamilton circuit*: Má neorientovaný graf Hamiltonovskou kružnici?
- *Colorability*: Má neorientovaný graf chromatické číslo k?
- *Directed Hamilton circuit*: Má neorientovaný graf Hamiltonovský cyklus?
- *Set cover*: Je dána třída množin $S_1, S_2, \ldots, S_n$. Existuje podtřída k množin $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ taková, že $\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j$?
- *Exact cover*: Je dána třída množin $S_1, S_2, \ldots, S_n$. Existuje množinové pokrytí (set cover) tvořené podtřídou po dvojicích disjunktních množin?

Příklad na analýzu složitosti I

Uvažme jazyk

$$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě,}$$
$$\text{pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$$

Poznámka: $\Phi_i(\bar{v}) \in \{\mathrm{true}, \mathrm{false}\}$ označuje, zda je formule Φ_i pravdivá při valuaci proměnných $\bar{v}$.

Nejdříve ukážeme, že $L_{NE} \in \mathbf{EXP}$.

- Sestrojíme DTS M, t.ž. $L(M) = L_{NE}$ a M pracuje v exponenciálním čase.
- M postupně generuje (např. v lexikografickém pořadí) jednotlivé valuace $\bar{v}$ proměnných z formule Φ_1 a Φ_2.
- Pro každou valuaci $\bar{v}$ M vyhodnotí $\Phi_1(\bar{v})$ a $\Phi_2(\bar{v})$ (lineární průchod oběma formulemi).
- Pokud $\Phi_1(\bar{v}) \neq \Phi_2(\bar{v})$, tak M akceptuje. Pokud prošel všechny valuace a neakceptoval, tak zamítne.
- Pokud Φ_1 a Φ_2 obsahuje n proměnných, pak existuje 2^n různých valuací. Složitost M je tedy v $O(2^n \cdot n) \subseteq O\left(2^{n^2}\right)$.

Příklad na analýzu složitosti II

Uvažme jazyk

$$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$$

Nyní ukážeme, že $L_{NE} \in \mathbf{NP}$.

- Sestrojíme NTS M, t.ž. $L(M) = L_{NE}$ a M pracuje v polynomiálním čase.
- M nedeterministicky zvolí (uhádne) valuaci $\bar{v}$ proměnných z formule Φ_1 a Φ_2.
- M vyhodnotí $\Phi_1(\bar{v})$ a $\Phi_2(\bar{v})$ (lineární průchod oběma formulemi).
- Pokud $\Phi_1(\bar{v}) \neq \Phi_2(\bar{v})$, tak M akceptuje. V opačném případě zamítne.
- Složitost M je tedy v $O(n)$. Ukázali jsme, že pro L_{NE} existuje polynomiální verifikátor.

Připomeňme, že $L_{NE} \in \mathbf{EXP}$ plyne taktéž přímo z $L_{NE} \in \mathbf{NP}$.

Příklad na polynomiální redukci

Uvažme jazyk

$$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě,}$$
$$\text{pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$$

Nyní ukážeme, že L_{NE} je $\mathbf{NP}$-těžký což nám s předchozím důkazem dá $\mathbf{NP}$-úplnost.

- Ukážeme, že $L_{SAT} \leq_P^m L_{NE}$. Bez ztráty na obecnosti uvažme, že

$$L_{SAT} = \{\Phi \mid \Phi \text{ je výroková formule v konjunktivní normální formě,}$$
$$\text{pro kterou existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi(\bar{v}) = \mathrm{true}\}$$

- Sestrojíme funkci f (vyčíslitelnou DTS v polynomiálním čase), pro kterou platí $f(\Phi) = (\Phi_1, \Phi_2)$ a $\Phi \in L_{SAT} \iff (\Phi_1, \Phi_2) \in L_{NE}$.
- Stačí položit $\Phi_1 \equiv \Phi$ a $\Phi_2 \equiv x_1 \wedge \overline{x_1}$ (x_1 je proměnná a tudíž Φ_2 je kontradikce).

$$\Phi \in L_{SAT} \Rightarrow \exists \bar{v} : \Phi_1(\bar{v}) = \mathrm{true} \wedge \Phi_2(\bar{v}) = \mathrm{false} \Rightarrow (\Phi_1, \Phi_2) \in L_{NE}$$

$$\Phi \notin L_{SAT} \Rightarrow \forall \bar{v} : \Phi_1(\bar{v}) = \mathrm{false} = \Phi_2(\bar{v}) \Rightarrow (\Phi_1, \Phi_2) \notin L_{NE}$$