Automaty a gramatiky

Roman Barták, KTIML

bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Organizační záležitosti

Přednáška:

- na webu (http://ktiml.mff.cuni.cz/~bartak/automaty)
- Proč chodit na přednášku?
  - dozvíte se více než při pouhém čtení slajdů
  - bude zábava (někdy)

Cvičení:

- Proč chodit na cvičení?
  - vyzkoušíte si, zda látce rozumíte
  - rozšíříte si znalosti z přednášky

Zkouška:

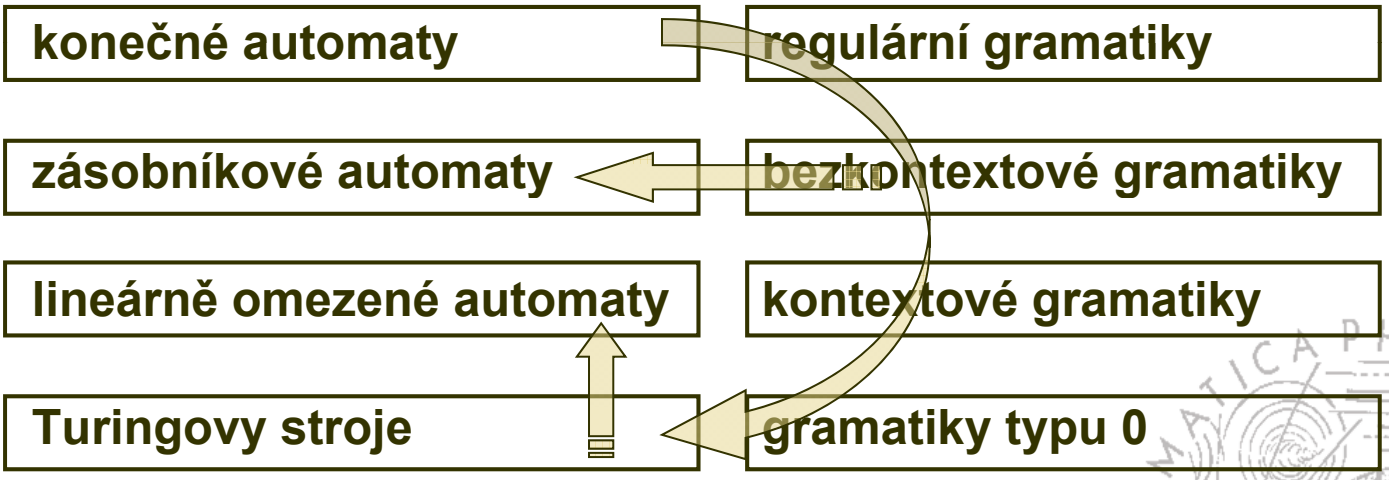
- písemná a ústní část
- porozumění látce + schopnost formalizace

O čem bude přednáška?

studium konečného popisu nekonečných objektů

studium abstraktních výpočetních zařízení

dvě větve: automaty a gramatiky



Zdroje a literatura

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

R. Barták: *Automaty a gramatiky: on-line* http://ktiml.mff.cuni.cz/~bartak/automaty

V. Koubek: *Automaty a gramatiky*, elektronický text, 1996

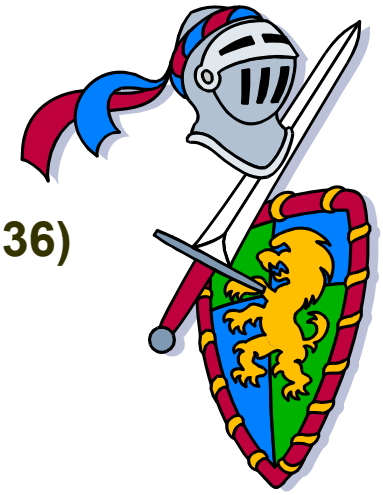
M. Chytil: *Teorie automatů a formálních jazyků*, skripta

M. Chytil: *Sbírka řešených příkladů z teorie automatů a formálních jazyků*, skripta

M. Demlová, V. Koubek: *Algebraická teorie automatů*, SNTL Praha, 1990

J.E. Hopcroft, R. Motwani, J.D. Ullman: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley

Pohled do historie



Počátky ve druhé čtvrtině 20. století
- první formalizace pojmu algoritmus (1936)
- co stroje umí a co ne?
- Church, Turing, Kleene, Post, Markov

Polovina 20. století
- neuronové sítě (1943)
- konečné automaty (Kleene 1956 neuronové sítě ≈ KA)

60. léta 20. století
- gramatiky (Chomsky)
- zásobníkové automaty
- formální teorie konečných automatů

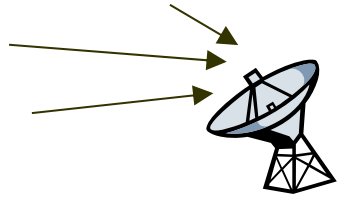
Praktické využití

- ✓ zpracování přirozeného jazyka
- ✓ překladače
- ✓ návrh, popis a verifikace hardware
  - integrované obvody, stroje, automaty
- ✓ realizace pomocí software
  - formální popis → program
  - hledání výskytu slova v textu, verifikace systémů s konečně stavy (protokoly,…), …
- ✓ aplikace v biologii
  - simulace růstu
  - celulární automaty
  - sebe-reprodukce automatů

Úvod do konečných automatů

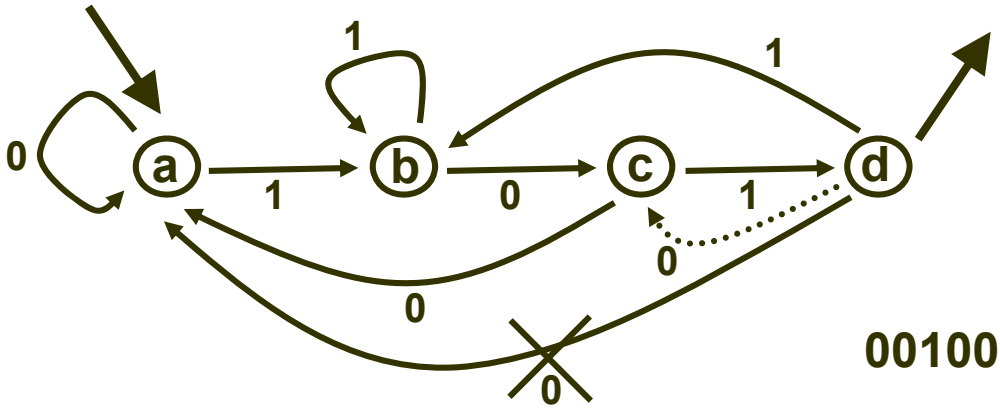
Projekt SETI (Search Extra-Terrestrial Intelligence)

analýza signálů - hledání vzorku



0101000101010010101001010101001010101001010100010001100011111001001000101011111100010

Hledání vzorku „101“

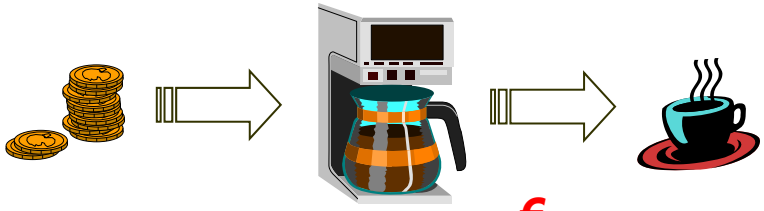


001001010101101

Úvod do konečných automatů 2

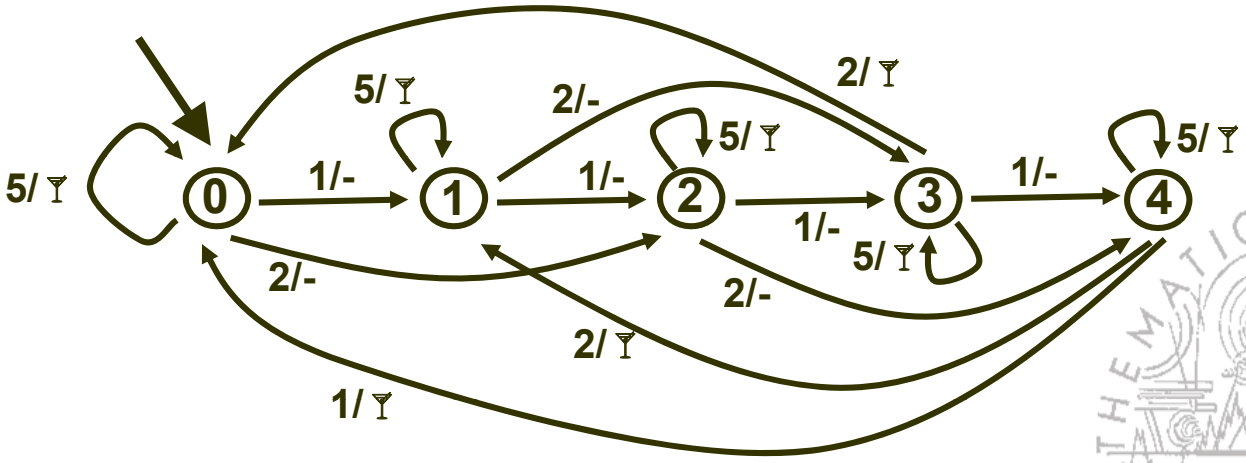
Stroj na kávu

stroj signalizuje vydání kávy po vhození potřebného obnosu



€ €

Vstupem stroje jsou mince 1,2,5 Kč, káva stojí 5 Kč



Realizace pomocí Mealyho stroje s výstupem při přechodu.

Formalizace konečného automatu

Konečným automatem nazýváme pětici

$$A = (Q, X, \delta, q_0, F),$$

kde:

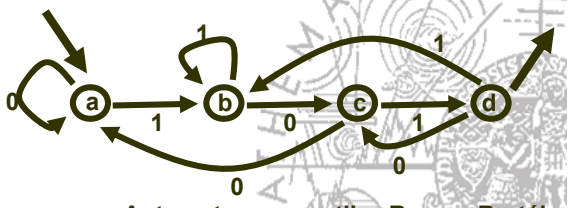
Q - konečná neprázdná množina stavů (*stavový prostor*)

X - konečná neprázdná množina symbolů (*vstupní abeceda*)

δ - zobrazení $Q \times X \rightarrow Q$ (*přechodová funkce*)

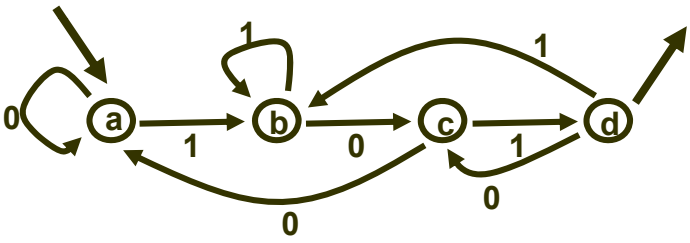
$q_0 \in Q$ (*počáteční stav*)

$F \subseteq Q$ (*množina přijímacích stavů*)



Popis konečného automatu

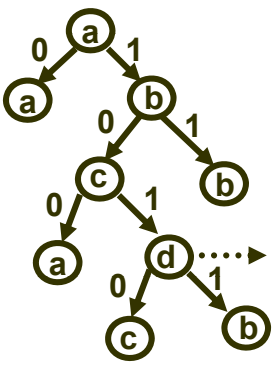
Stavový diagram (graf)
vrcholy = stavy
hrany = přechody



Tabulka
řádky = stavy+přechody
sloupce = písmena

	0	1
→ a	a	b
b	c	b
c	a	d
← d	c	b

Stavový strom
vrcholy = stavy
hrany = přechody
pouze dosažitelné stavy!



Abeceda, slova, jazyky

abeceda X = konečná neprázdná množina symbolů

slovo = konečná posloupnost symbolů (i prázdná)

prázdné slovo λ (e, ε, ...)

X^* = množina všech slov v abecedě X

X^+ = množina všech neprázdných slov v abecedě X

$X^* = X^+ \cup \{\lambda\}$

jazyk $L \subseteq X^*$ (množina slov v abecedě X)

Základní operace se slovy:

zřetězení slov u.v, uv

mocnina u^n ($u^0 = \lambda$, $u^1 = u$, $u^{n+1} = u^n.u$)

délka slova |u| ($|\lambda| = 0$)

Rozšířená přechodová funkce

přechodová funkce $\delta: Q \times X \rightarrow Q$

rozšířená přechodová funkce $\delta^* : Q \times X^* \rightarrow Q$

tranzitivní uzávěr δ

induktivní definice

$\delta^*(q,\lambda) = q$

$\delta^*(q,wx) = \delta(\delta^*(q,w),x)$, $x \in X, w \in X^*$

úmluva:

δ^* budeme někdy označovat také jako δ

Jazyky rozpoznatelné konečnými automaty

Jazykem rozpoznávaným (akceptovaným, přijímaným) konečným automatem $A = (Q,X,\delta,q_0,F)$ nazveme jazyk:

$$L(A) = \{w \mid w \in X^* \wedge \delta^*(q_0,w) \in F\}.$$

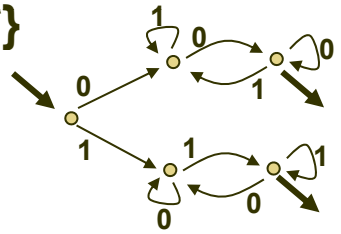
Slovo w je *přijímáno* automatem A, právě když $w \in L(A)$.

Jazyk L je *rozpoznatelný* konečným automatem, jestliže existuje konečný automat A takový, že L=L(A).

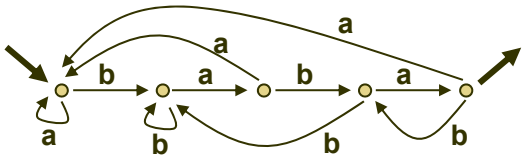
Třídu jazyků rozpoznatelných konečnými automaty značíme $\mathcal{F}$, tzv. *regulární jazyky*.

Příklady regulárních jazyků

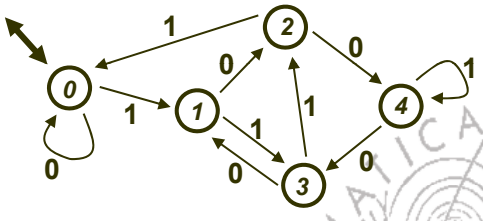
$L = \{ w \mid w \in \{0,1\}^*, w=xux, x \in \{0,1\}, u \in \{0,1\}^* \}$



$L = \{ w \mid w \in \{a,b\}^*, w=ubaba, u \in \{a,b\}^* \}$

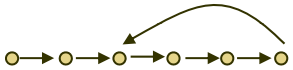


$L = \{ w \mid w \in \{0,1\}^* \wedge w$ je binární zápis čísla dělitelného 5$\}$



$L = \{0^n1^n \mid n \geq 1\}$

není regulární jazyk!



Kongruence

Jak zjistit, že jazyk není rozpoznatelný konečným automatem?

Jak charakterizovat regulární jazyky?

Kongruence

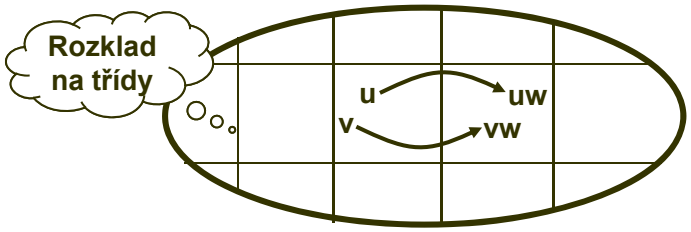
Nechť X je konečná abeceda, ~ je relace ekvivalence (reflexivní, symetrická, transitivní) na X*. Potom:

a) ~ je *pravá kongruence*, jestliže

$$\forall u,v,w \in X^* \quad u \sim v \Rightarrow uw \sim vw$$

b) je *konečného indexu*, jestliže

rozklad X*/~ má konečný počet tříd

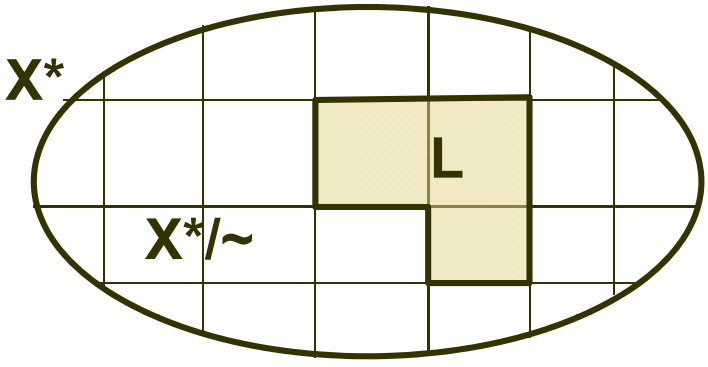


Nerodova věta

Nechť L je jazyk nad konečnou abecedou X. Potom následující tvrzení jsou ekvivalentní:

a) L je rozpoznatelný konečným automatem,

b) existuje pravá kongruence ~ konečného indexu na X* tak, že L je sjednocením jistých tříd rozkladu X*/~.



Důkaz Nerodovy věty

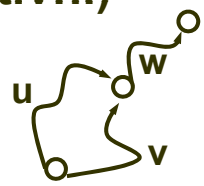
a) ⇒ b)

automat ⇒ pravá kongruence konečného indexu

definujme $u \sim v \equiv \delta^*(q_0,u) = \delta^*(q_0,v)$

je to ekvivalence (reflexivní, symetrická, transitivní)

je to pravá kongruence (z definice δ^*)



má konečný index (konečně mnoho stavů)

$L = \{ w \mid \delta^*(q_0,w) \in F\} = \cup_{q \in F} \{ w \mid \delta^*(q_0,w) = q\}$

Pozorování:

stavy odpovídají třídám ekvivalence

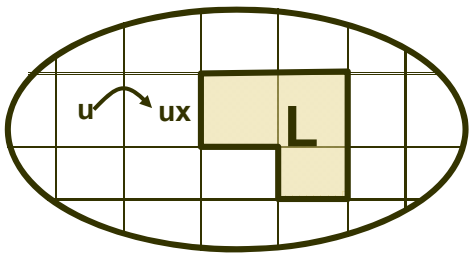
Důkaz Nerodovy věty - pokračování

b) ⇒ a)

pravá kongruence konečného indexu ⇒ automat

označme [u] třídu rozkladu obsahující slovo u

Jak sestrojíme konečný automat A?



abeceda X dána

stavy Q - třídy rozkladu X*/~

stav $q_0 = [\lambda]$

koncové stavy $F = \{c_1,..,c_n\}$, kde $L = \cup_{i=1..n} c_i$

přechodová funkce $\delta([u],x) = [ux]$

přechodová funkce je korektní (z definice pravé kongruence)

Ještě L(A) = L?

$w \in L \Leftrightarrow w \in \cup_{i=1..n} c_i \Leftrightarrow w \in c_1 \vee \ldots \vee w \in c_n \Leftrightarrow [w] = c_1 \vee \ldots \vee [w] = c_n \Leftrightarrow [w] \in F \Leftrightarrow w \in L(A)$

$\delta^*([\lambda],w) = [w]$

Použití Nerodovy věty

Konstrukce automatů

Příklad:

Sestrojte automat přijímající jazyk

$L = \{w \mid w \in \{a,b\}^* \ \& \ w \text{ obsahuje } 3k+2 \text{ symbolů } a\}$

označme $|u|_x$ počet symbolů x ve slově u

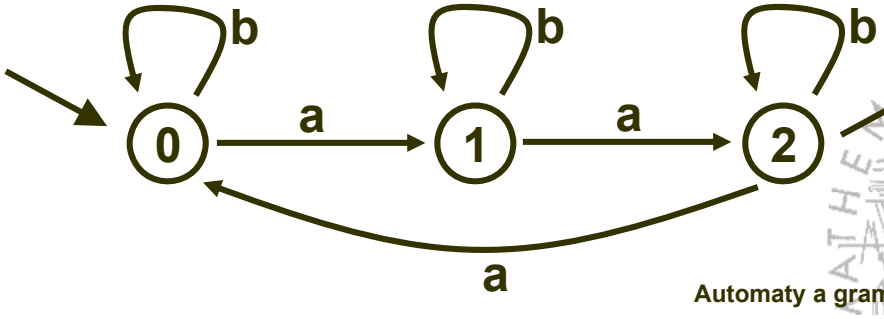
definujme $u \sim v \equiv (|u|_a \bmod 3 = |v|_a \bmod 3)$

tři třídy ekvivalence 0,1,2 (zbytky po dělení 3)

L odpovídá třídě 2

a-přechody přesouvají do následující třídy (mod 3)

b-přechody zachovávají třídu



Použití Nerodovy věty - pokračování

Důkaz neregulárnosti jazyka!

Příklad:

Rozhodněte zda následující jazyk je regulární

$L = \{0^n1^n \mid n \geq 1\}$.

Předpokládejme, že jazyk je regulární

$\Rightarrow$ existuje pravá kongruence konečného indexu m,

L je sjednocením tříd

vezmeme slova 0, 00, …, 0^{m+1}

dvě slova padnou do stejné třídy (krabičkový princip)

$i \neq j$	$0^i \sim 0^j$	
přidejme 1^i	$0^i1^i \sim 0^j1^i$	(pravá kongruence)
spor	$0^i1^i \in L \ \& \ 0^j1^i \notin L$	

„Pravidelnost“ regulárních jazyků

Konečný automat kóduje pouze konečnou informaci.

Přesto můžeme rozpoznávat nekonečné jazyky!

Můžeme přijímat libovolně (ale konečně) dlouhá slova!

Pozorování:

$L = \{w \mid w \in \{0,1\}^* \;\&\; |w|_1 = 3k+1\}$

$0100111010 \in L$ potom také:

$0100111011101010 \in L$ řetězec 01101 jsme zdvojili

$0101110111011101010 \in L$ řetězec 01101 jsme ztrojili

$0100 \in L$ řetězec 01101 jsme vypustili

Lze takovouto operaci provést s každým slovem libovolného regulárního jazyka?

ANO (pokud je slovo dostatečně dlouhé)

Iterační (pumping) lemma

Nechť L je regulární jazyk. Potom existuje přirozené číslo n takové, že libovolné slovo $z \in L$, $|z| \geq n$ lze psát ve tvaru uvw, kde: $|uv| \leq n$, $1 \leq |v|$ a pro všechna $i \geq 0$ $uv^iw \in L$.

Důkaz:

za n vezměme počet stavů příslušného automatu

při zpracování slova délky ≥n se automat nutně musí dostat do jednoho stavu dvakrát (krabičkový princip)

vezměme takový stav p, který se opakuje jako první

potom $\delta(q_0,u)=p$ poprvé jsme se dostali do p

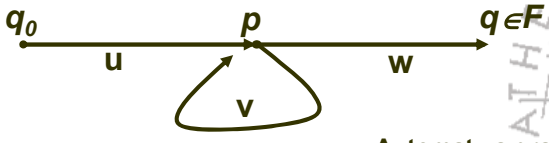
$\delta(q_0,uv)=p$ podruhé jsme se dostali do p

zřejmě $|uv| \leq n$ (pouze p se opakuje tj. maximálně n+1 stavů)

$|v| \geq 1$ (smyčka obsahuje alespoň jedno písmeno)

smyčku mezi prvním a druhým průchodem p (odpovídá jí slovo v) nyní můžeme vypustit nebo projít vícekrát

tj. $\forall i \geq 0$ $uv^iw \in L$



Použití iteračního lemmatu

Důkaz neregulárnosti jazyka

$L = \{0^i 1^i \mid i \geq 0\}$ není regulární jazyk

sporem:

nechť L je regulární, potom lze pumpovat ($\exists n$ tž. …)

vezměme slovo $0^n 1^n$ (to je určitě delší než n)

pumpovat můžeme pouze v části 0^n

potom, ale $0^{n+i}1^n \notin L$ (i>0 je délka pumpované části), což je spor

POZOR! Iterační lemma představuje nutnou podmínku regulárnosti jazyka, nedává podmínku postačující.

Existuje jazyk, který není regulární a lze pumpovat.

$L = \{u \mid u=a^+b^ic^i \vee u=b^ic^j\}$

vždy lze pumpovat první písmeno

L není regulární (Nerodova věta)

Iterační lemma a nekonečnost jazyků

Umíme algoritmicky rozhodnout, zda je regulární jazyk L nekonečný?

ANO!

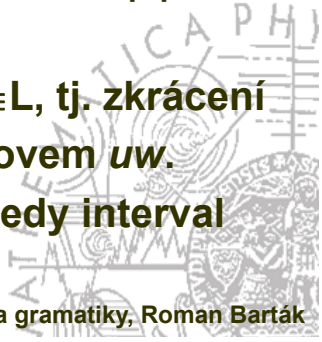
jazyk L je nekonečný $\Leftrightarrow \exists u \in L$ tž. $n \leq |u| < 2n$
n je číslo z iteračního lemmatu

stačí prozkoumat všechna slova u taková, že $n \leq |u| < 2n$ (to je konečně mnoho slov)

PROČ?

- Pokud $\exists u \in L$ tž. $n \leq |u|$ (< 2n), potom lze slovo *u* pumpovat, čímž dostaneme nekonečně mnoho slov z jazyka L.
- Je-li jazyk L nekonečný, obsahuje slovo *z* takové, že $n \leq |z|$.
  - Pokud $|z| < 2n$ máme hledané slovo.
  - Jinak, podle iteračního lemmatu $z = uvw$ a $uw \in L$, tj. zkrácení
  - Platí-li stále $2n \leq |uw|$, opakuj zkracování se slovem *uw*.

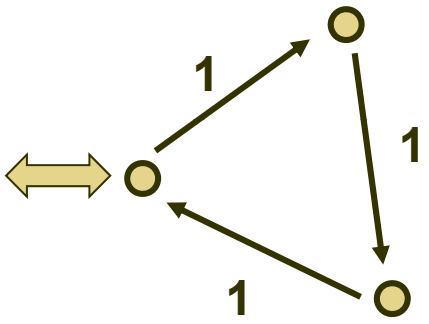
  Poznámka: zkracujeme maximálně o *n* písmen, tedy interval [n,2n) nelze přeskočit!

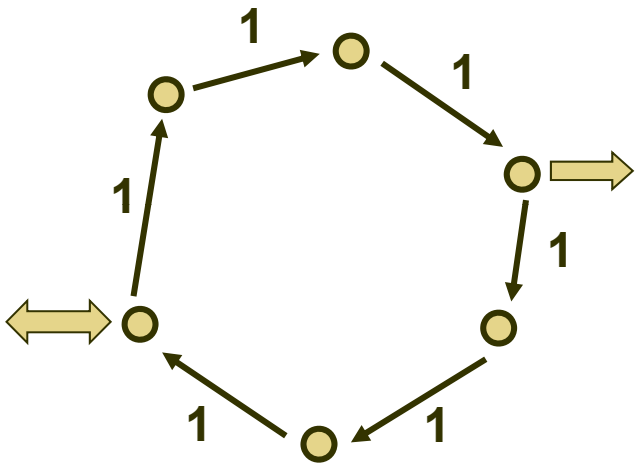


Jazyk a přijímající automaty

Je automat pro daný jazyk určen jednoznačně?

NENÍ!





$$L = \{w \mid w \in \{1\}^* \wedge |w| = 3k\}$$

Ekvivalence automatů a homomorfismus

Jak zjistit, zda dva automaty přijímají stejný jazyk?

Říkáme, že konečné automaty A a B jsou *ekvivalentní*, jestliže rozpoznávají stejný jazyk, tj. L(A)=L(B).

Nechť A_1 a A_2 jsou konečné automaty. Řekneme, že zobrazení $h: Q_1 \rightarrow Q_2$ je (automatovým) *homomorfismem*, jestliže:

1) $h(q_1) = q_2$ „stejné“ počáteční stavy
2) $h(\delta_1(q,x)) = \delta_2(h(q),x)$ „stejné“ přechodové funkce
3) $q \in F_1 \Leftrightarrow h(q) \in F_2$ „stejné“ koncové stavy

Homomorfismus prostý a na nazýváme *isomorfismus*.

Věta o ekvivalenci automatů

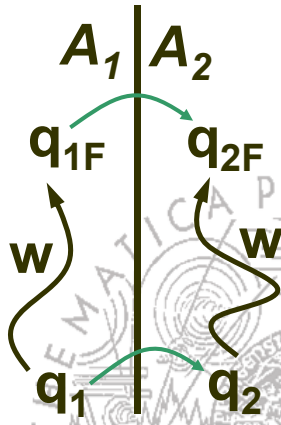
Existuje-li homomorfismus konečných automatů A_1 do A_2, pak A_1 a A_2 jsou ekvivalentní.

Důkaz:

konečnou iterací

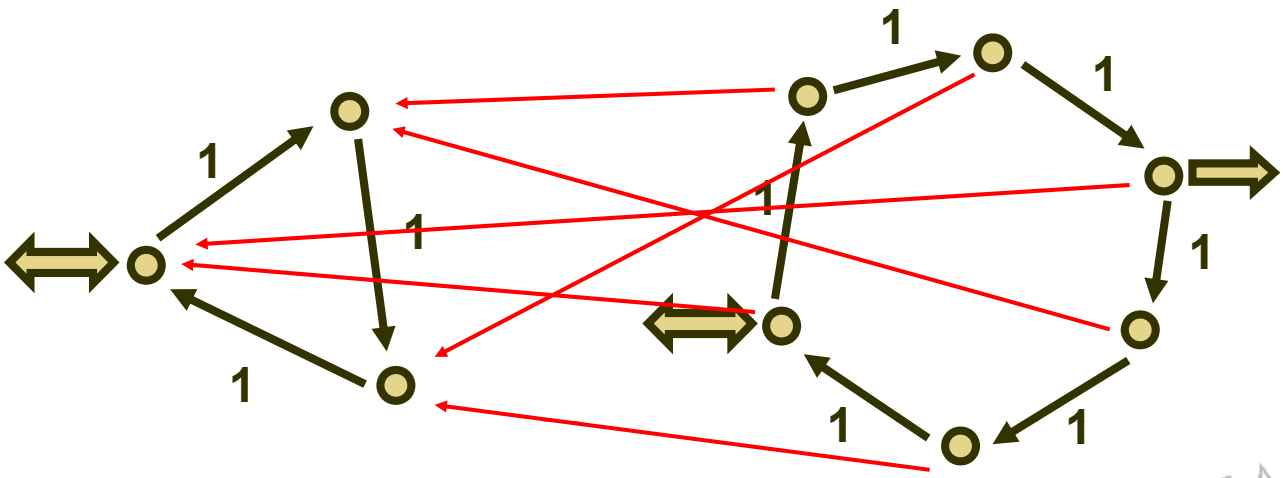
$$h(\delta_1(q,w)) = \delta_2(h(q),w) \quad w \in X^*$$

$$\begin{aligned} w \in L(A_1) &\Leftrightarrow \delta_1(q_1,w) \in F_1 \\ &\Leftrightarrow h(\delta_1(q_1,w)) \in F_2 \\ &\Leftrightarrow \delta_2(h(q_1),w) \in F_2 \\ &\Leftrightarrow \delta_2(q_2,w) \in F_2 \\ &\Leftrightarrow w \in L(A_2) \end{aligned}$$



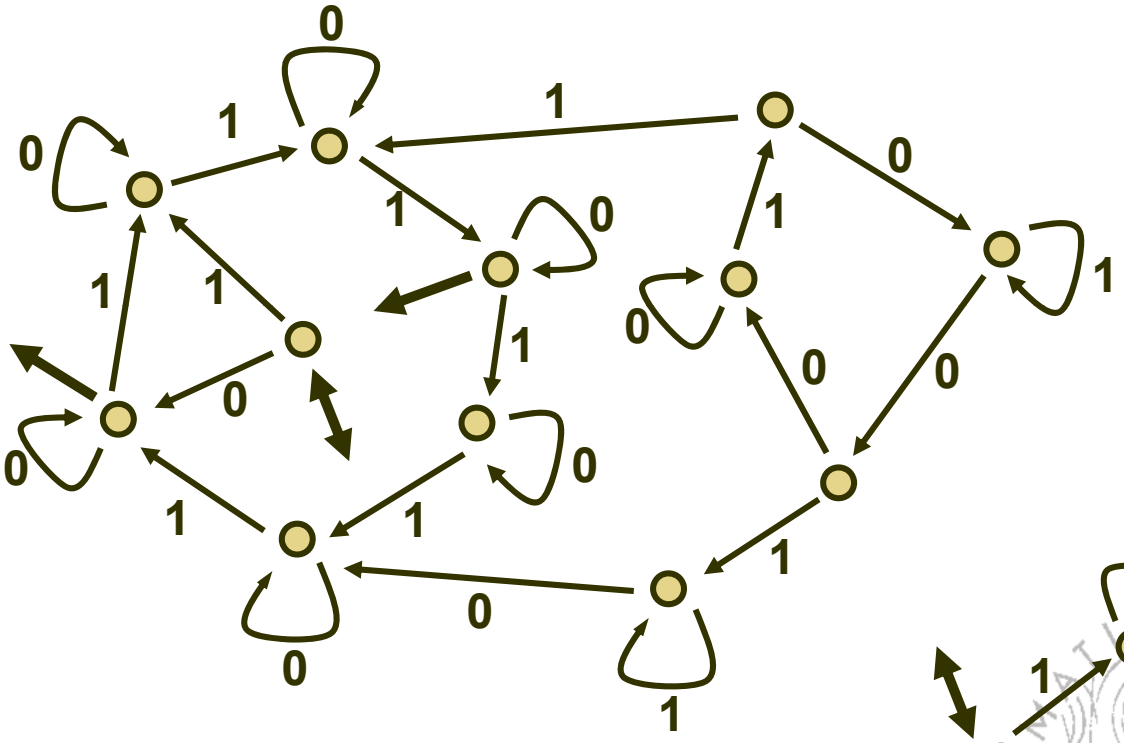
Homomorfismus automatů

$L = \{w \mid w=1^* \wedge |w|=3k\}$

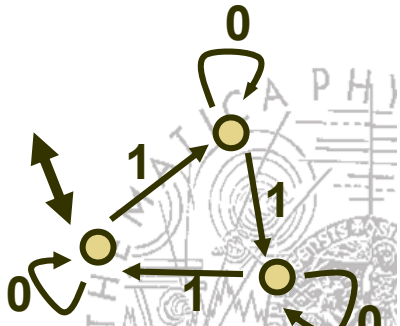


Trochu motivace

Co dělá tento automat?



$L = \{w \mid w \in \{0,1\}^* \wedge |w|_1 = 3k\}$

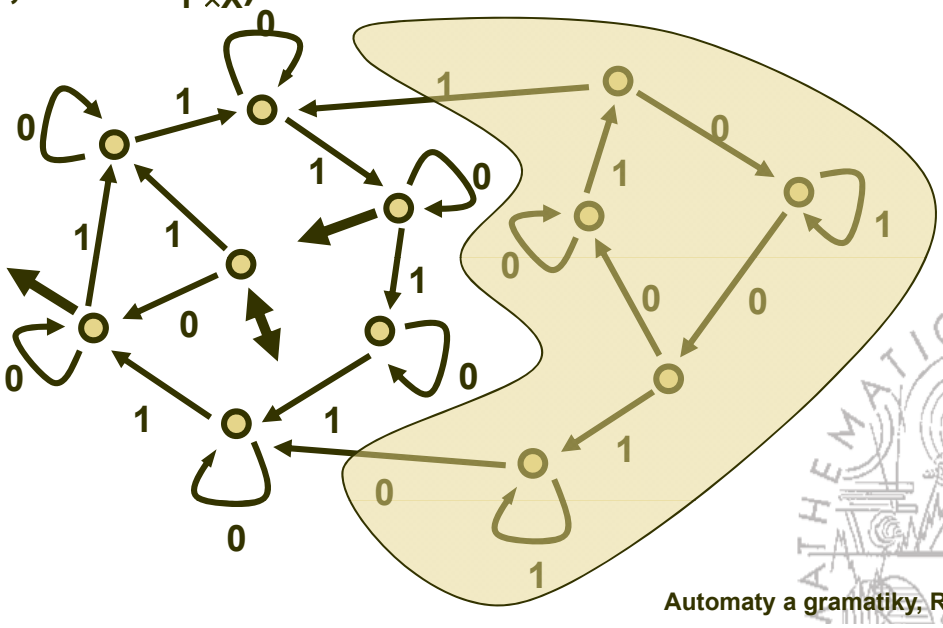


Dosažitelné stavy

$A = (Q,X,\delta,q_0,F)$ je konečný automat a $q \in Q$.

Řekneme, že stav q je *dosažitelný*, jestliže $\exists w \in X^* \; \delta^*(q_0,w)=q$

Věta: Nechť P jsou dosažitelné stavy automatu A. Potom $B = (P,X,\delta',q_0,F')$ je konečný automat ekvivalentní s A ($F' = F \cap P$, $\delta' = \delta\restriction_{P\times X}$).



Hledání dosažitelných stavů

Iterační algoritmus (dosažitelnost po i krocích):

$M_0 = \{q_0\}$

repeat

$M_{i+1} = M_i \cup \{q \mid q \in Q, \exists p \in M_i \; \exists x \in X \; \delta(p,x)=q\}$

until $M_{i+1} = M_i$

Korektnost

$M_0 \subset M_1 \subset \ldots \subset Q$ (algoritmus je konečný)

každé M_i obsahuje pouze dosažitelné stavy

Úplnost

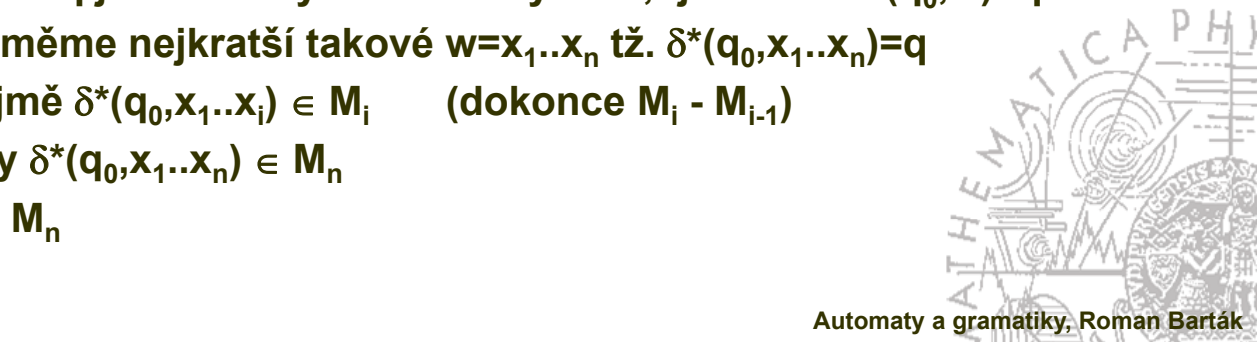
nechť q je libovolný dosažitelný stav, tj. $\exists w \in X^* \; \delta^*(q_0,w)=q$

vezměme nejkratší takové $w=x_1..x_n$ tž. $\delta^*(q_0,x_1..x_n)=q$

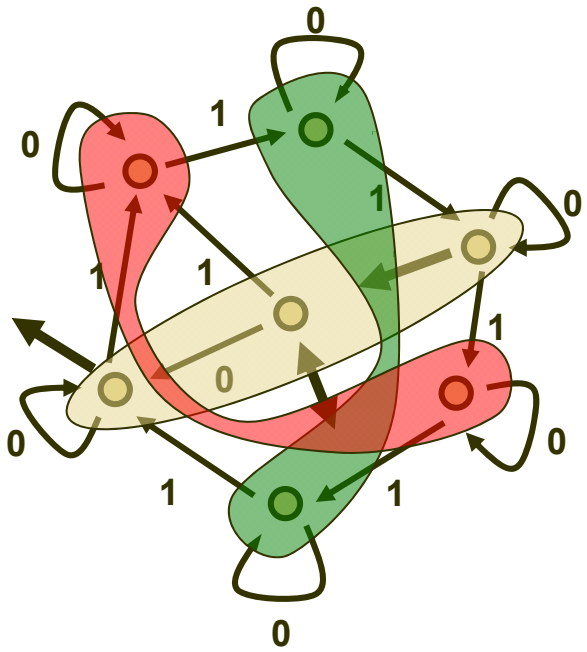
zřejmě $\delta^*(q_0,x_1..x_i) \in M_i$ (dokonce $M_i - M_{i-1}$)

tedy $\delta^*(q_0,x_1..x_n) \in M_n$

$q \in M_n$



Ekvivalentní stavy



Výpočty startující z ekvivalentních stavů nelze rozlišit!

Nechť $A = (Q,X,\delta,q_0,F)$ je konečný automat.

Stavy p,q jsou *ekvivalentní*, značíme p~q, jestliže:

$\forall w \in X^* \; \delta^*(p,w) \in F \Leftrightarrow \delta^*(q,w) \in F$

Ekvivalence po krocích

Ekvivalence po i krocích

$p \sim^i q$ $\quad \forall w \in X^* \text{ tž. } |w| \leq i \;\; \delta^*(p,w) \in F \Leftrightarrow \delta^*(q,w) \in F$

$p \sim q \quad \Leftrightarrow \; \forall i \;\; p \sim^i q$

Iterativní konstrukce $\sim^i$

$p \sim^0 q$ ... $p \in F \Leftrightarrow q \in F$

$p \sim^{i+1} q$ ... $p \sim^i q \;\wedge\; \forall x \in X \; \delta(p,x) \sim^i \delta(q,x)$

Je to v pořádku?

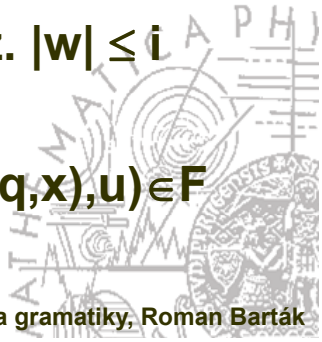
$p \sim^0 q$ … $\delta^*(p,\lambda)=p \in F \;\Leftrightarrow\; \delta^*(q,\lambda)= q \in F$

$p \sim^{i+1} q$ ... $p \sim^i q$ tj. platí pro slova w tž. $|w| \leq i$

slova délky i+1, w = xu, |u|=i

$\delta(p,x) \sim^i \delta(q,x)$ tj. $\delta^*(\delta(p,x),u) \in F \Leftrightarrow \delta^*(\delta(q,x),u) \in F$

dohromady $\delta^*(p,xu) \in F \Leftrightarrow \delta^*(q,xu) \in F$



Vlastnosti ekvivalence po krocích

1) $\forall i \geq 0$ je $\sim^i$ ekvivalence na Q, označme $\mathcal{R}_i = Q/\sim^i$

2) $\mathcal{R}_{i+1}$ zjemňuje $\mathcal{R}_i$

3) $\mathcal{R}_{i+1} = \mathcal{R}_i \Rightarrow \forall t>0 \; \mathcal{R}_{i+t} = \mathcal{R}_i$

4) nechť |Q|=n, potom $\exists k \leq n-1 \; \mathcal{R}_{k+1} = \mathcal{R}_k$

5) $\mathcal{R}_{k+1} = \mathcal{R}_k \Rightarrow (p \sim q \Leftrightarrow p \sim^k q)$

Důkaz:

1) a 2) přímo z definice

3) $p \sim^{i+1} q \;\Leftrightarrow\; p \sim^i q \wedge \forall x \in X \; \delta(p,x) \sim^i \delta(q,x)$

$p \sim^{i+2} q \;\Leftrightarrow\; p \sim^{i+1} q \wedge \forall x \in X \; \delta(p,x) \sim^{i+1} \delta(q,x)$

4) přímo z max. počtu tříd rozkladu (n)

5) $p \sim q \quad \Leftrightarrow \forall i \geq 0 \; p \sim^i q$

$\Leftrightarrow \forall 0 \leq i \leq k \; p \sim^i q \wedge \forall i > k \; p \sim^i q$

$\Leftrightarrow p \sim^k q$

Hledání ekvivalence stavů

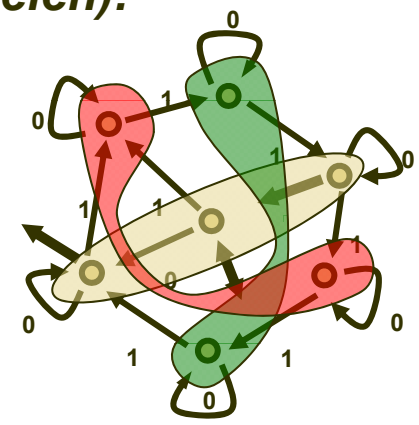
Iterační algoritmus (ekvivalence po i krocích):

sestroj $\mathcal{R}_0$

repeat

sestroj $\mathcal{R}_{i+1}$ z $\mathcal{R}_i$

until $\mathcal{R}_{i+1} = \mathcal{R}_i$



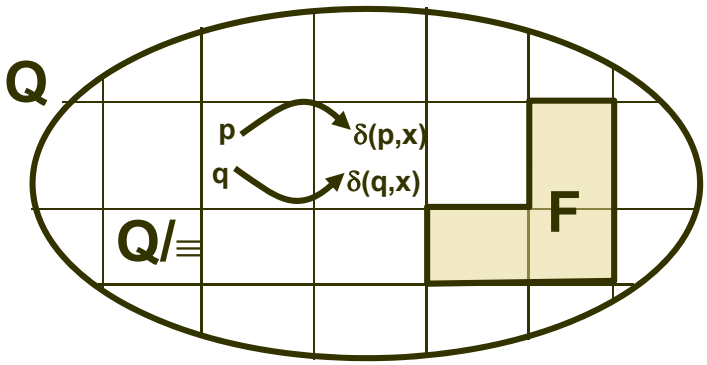
Příklad:

	0	1	$\mathcal{R}_0$	0	1	$\mathcal{R}_1$	0	1	$\mathcal{R}_2$
↔ a	b	c	A	A	C	A	A	C	A
← b	b	c	A	A	C	A	A	C	A
c	c	d	C	C	C	C	C	D	C
d	d	e	C	C	A	D	D	A	D
← e	e	f	A	A	C	A	A	C	A
f	f	g	C	C	C	C	C	D	C
g	g	b	C	C	A	D	D	A	D

Automatová kongruence

Nechť ≡ je relace ekvivalence na Q. Říkáme, že ≡ je *automatovou kongruencí*, jestliže:

$\forall p,q \in Q\ p \equiv q \Rightarrow (p \in F \Leftrightarrow q \in F) \wedge \forall x \in X\ (\delta(p,x) \equiv \delta(q,x))$



Tvrzení: Ekvivalence stavů ~ je automatovou kongruencí.

Důkaz: nechť p~q, potom: $p \in F \Leftrightarrow q \in F$ (položme $w=\lambda$, $\delta^*(p,\lambda)=p$)

nechť p~q, potom: $\forall x \in X\ \forall u \in X^*\ (\delta^*(p,xu) \in F \Leftrightarrow \delta^*(q,xu) \in F)$

$\Leftrightarrow \forall x \in X\ \forall u \in X^*\ (\delta^*(\delta(p,x),u) \in F \Leftrightarrow \delta^*(\delta(q,x),u) \in F)$

$\Leftrightarrow \forall x \in X\ \delta(p,x) \sim \delta(q,x)$

Podílový automat

$A = (Q,X,\delta,q_0,F)$ je konečný automat a $\equiv$ je automatová kongruence.

Potom $A/\equiv = (Q/\equiv, X, \delta_\equiv, [q_0]_\equiv, \{[q]_\equiv \mid q \in F\})$, kde $\delta_\equiv([q],x) = [\delta(q,x)]_\equiv$ je konečný automat (nazvěme ho *podílovým automatem*) ekvivalentní s A.

1) Je $A/\equiv$ skutečně konečný automat?

Množiny $Q/\equiv$ a X jsou neprázdné a konečné.

$\delta_\equiv$ je definována korektně (vlastnosti automatové kongruence)

2) Jsou oba automaty ekvivalentní?

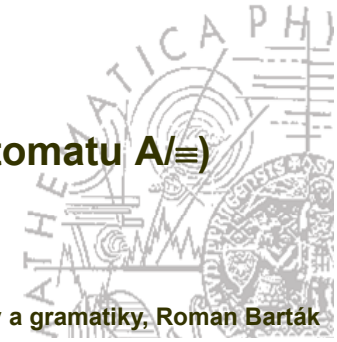
Stačí najít homomorfismus A do $A/\equiv$ (věta o ekvivalenci automatů)!

Definujme $h: Q \to Q/\equiv$ takto $h(q) = [q]_\equiv$

$h(q_0) = [q_0]_\equiv$

$h(\delta(q,x)) = [\delta(q,x)]_\equiv = \delta_\equiv([q]_\equiv,x) = \delta_\equiv(h(q),x)$

$q \in F \Leftrightarrow h(q)=[q]_\equiv \in F_\equiv$ ($F_\equiv$ jsou koncové stavy automatu $A/\equiv$)



Podílový automat a ekvivalence stavů

A je konečný automat a ~ je ekvivalence stavů.

Potom A/~ je konečný automat ekvivalentní s A a žádné stavy A/~ nejsou ekvivalentní.

1) Ekvivalence stavů ~ je automatová kongruence, a tedy víme, že A/~ je konečný automat ekvivalentní s A.

2) V A/~ nejsou ekvivalentní stavy.

Sporem: nechť $[p]_\sim$ a $[q]_\sim$ jsou různé ekvivalentní stavy (tj. $\neg\, p\sim q$)

vezměme libovolné $w \in X^*$:

$\delta_\sim([p]_\sim,w) \in F_\sim \Leftrightarrow \delta_\sim([q]_\sim,w) \in F_\sim$	($[p]_\sim$ a $[q]_\sim$ jsou ekvivalentní)
$\delta_\sim(h(p),w) \in F_\sim \Leftrightarrow \delta_\sim(h(q),w) \in F_\sim$	($h(p)= [p]_\sim$)
$h(\delta(p,w)) \in F_\sim \Leftrightarrow h(\delta(q,w)) \in F_\sim$	(h je homomorfismus)
$\delta(p,w) \in F \Leftrightarrow \delta(q,w) \in F$	
$p\sim q$	spor

Redukce automatu

Konečný automat je *redukovaný*, jestliže:

- nemá nedosažitelné stavy,
- žádné dva stavy nejsou ekvivalentní.

Konečný automat B je *reduktem* automatu A, jestliže:

- B je redukovaný,
- A a B jsou ekvivalentní.

Věta: Ke každému konečnému automatu existuje nějaký jeho redukt.

Konstruktivní důkaz (dvě možné metody):

1) vyřazení nedosažitelných stavů
   faktorizace podle ekvivalence stavů (nezpůsobí nedosažitelnost)

nebo

2) faktorizace podle ekvivalence stavů
   vyřazení nedosažitelných stavů (nezpůsobí změnu ekvivalence)

Příklad redukce automatů

Co dělá tento automat (jaký jazyk přijímá)?

	a	b	$\mathcal{R}_0$	a	b	$\mathcal{R}_1$	a	b	$\mathcal{R}_2$
→ 1	2	3	I	I	III	I	II	III	I
2	2	4	I	I	I	II	II	II	II
← 3	3	5	III	III	III	III	III	III	III
4	2	7	I	I	I	II	II	II	II
← 5	6	3	III	III	III	III	III	III	III
← 6	6	6	III	III	III	III	III	III	III
7	7	4	I	I	I	II	II	II	II
~~8~~	~~2~~	~~3~~	~~I~~	~~I~~	~~III~~	~~I~~	~~II~~	~~III~~	~~I~~
~~9~~	~~9~~	~~4~~	~~I~~	~~I~~	~~I~~	~~II~~	~~II~~	~~II~~	~~II~~

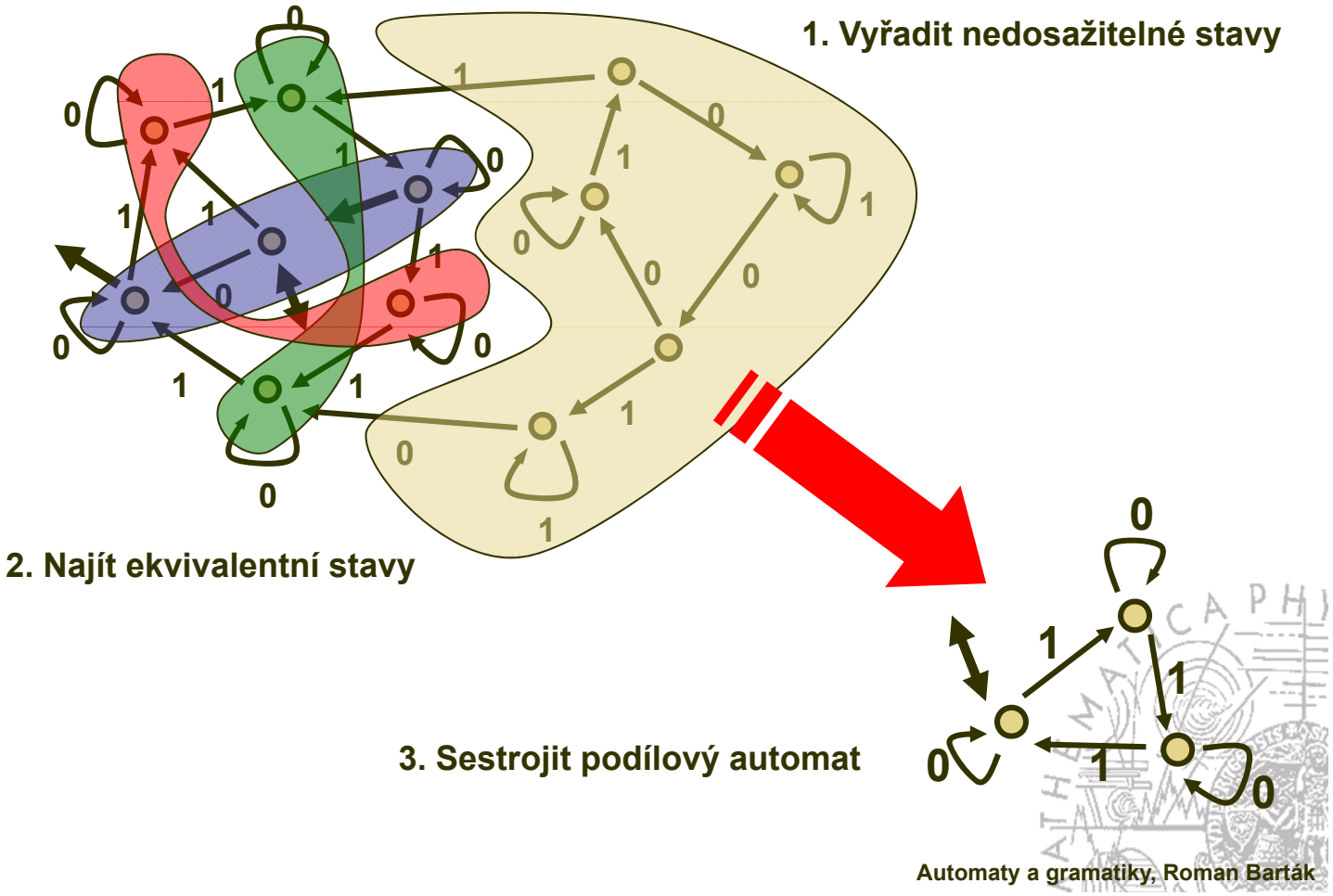
Nedosažitelné stavy

Redukovaný automat

	a	b
→ I	II	III
II	II	II
← III	III	III

$L = \{w \mid w=bu,\ u \in \{a,b\}^*\}$

Redukce automatu v kostce



Věta o isomorfismu reduktů

Pro libovolné dva redukované konečné automaty jsou následující dvě tvrzení ekvivalentní:

a) automaty jsou ekvivalentní,

b) automaty jsou isomorfní.

Důsledky:

Dva redukty libovolných dvou ekvivalentních konečných automatů se shodují až na isomorfismus.

Pro každý konečný automat je jeho redukt určen až na isomorfismus jednoznačně.

Ve třídě navzájem ekvivalentních konečných automatů existuje „minimální“ automat.

Důkaz věty o isomorfismu reduktů

a) isomorfismus ⇒ ekvivalence (víme)

b) ekvivalence reduktů ⇒ isomorfismus

hledáme homomorfismus $h:Q_1 \to Q_2$, který je „prostý a na“ tj. pro každé $q \in Q_1$ hledáme právě jedno $p \in Q_2$

q je dosažitelný stav, tudíž $\exists u \in X^* \ \delta_1(q_1,u)=q$

položme $h(q) = \delta_2(q_2,u)$

je to skutečně funkce?

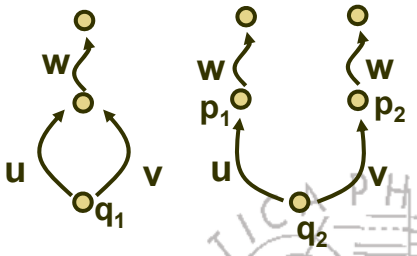
$\delta_1(q_1,u)=\delta_1(q_1,v) \Leftrightarrow \delta_2(q_2,u)=\delta_2(q_2,v)$ (*)

sporem, nechť $\delta_1(q_1,u)=\delta_1(q_1,v)$ & $\delta_2(q_2,u)\neq\delta_2(q_2,v)$

z A_1 víme $\forall w \in X^*$ $uw \in L \Leftrightarrow vw \in L$

automaty jsou ekvivalentní, tedy $p_1 \sim p_2$

spor - automat A_2 je redukovaný



funkce h je „prostá a na“ (vlastnost (*))

$h(q_1)=q_2$ (pro $u=\lambda$)

$h(\delta_1(q,x)) = \delta_2(h(q),x)$ ($\delta_1(q_1,v)=q$, $u=vx$)

$q \in F_1 \Leftrightarrow h(q) \in F_2$ (pro $u \in L$ + ekvivalentní automaty)

Normalizace automatu

Jak najít isomorfismus automatů?

Normovaný tvar automatu

1) fixujme pořadí písmen v abecedě
2) počáteční stav označme 1
3) tabulku (automatu) vyplňujme po řádcích zleva doprava a pokud narazíme na nový stav, přiřadíme mu první volné číslo

Příklad:

	a	b
A	B	A
⟷ B	D	C
C	A	D
⟵ D	A	B

	a	b
⟷ 1(B)	2	3
⟵ 2(D)	4	1
3(C)	4	2
4(A)	1	4

Poznámky k redukci a ekvivalenci

Algoritmicky umíme řešit:

- **zjištění ekvivalence automatů**
  - zredukujeme, znormalizujeme a porovnáme
- **zjištění zda L(A)=∅**
  - žádný koncový stav není dosažitelný
- **zjištění zda L(A)=X***
  - po redukci dostaneme jednostavový automat (s koncovým stavem)

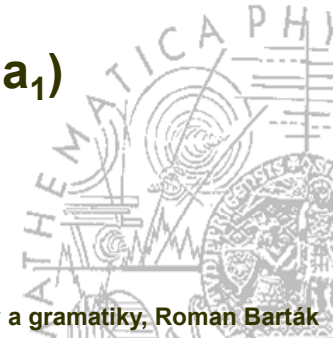
Umíme najít nejkratší slovo rozlišující dva stavy

$p \sim^i q \;\&\; \neg\, p \sim^{i+1} q$

$\exists a_1 \in X \quad \delta(p,a_1) \sim^{i-1} \delta(q,a_1) \;\&\; \neg\, \delta(p,a_1) \sim^{i} \delta(q,a_1)$

…

iterací najdeme slovo $a_1 \ldots a_{i+1}$



Slovo odlišující dva stavy

		0	1	$\Re_0$	0	1	$\Re_1$	0	1	$\Re_2$
	a	a	b	A	A	A	A	A	B	A
	b	c	a	A	C	A	B	C	A	B
←	c	c	e	C	C	C	C	C	E	C
	d	e	d	A	C	A	B	E	B	D
←	e	e	d	C	C	A	E	E	B	E

Jak je dlouhé nejkratší slovo rozlišující stavy „b“ a „d“?

2 znaky

A jaké je to slovo?

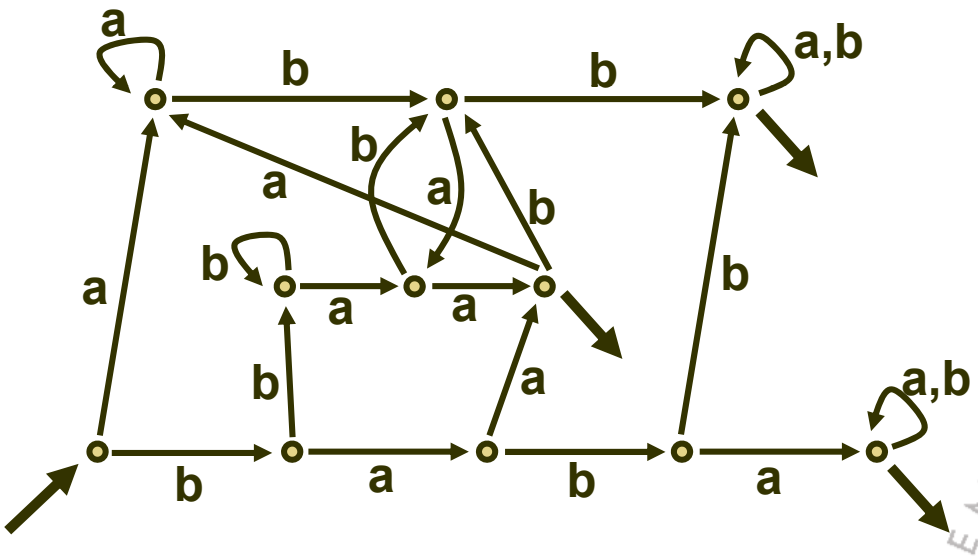
01 nebo 10

Trochu motivace

Dosud: Stav a písmeno jednoznačně určuje další stav!

Příklad:

L = { w | w=babau ∨ w=uabbv ∨ w=ubaa, u,v∈{a,b}* }

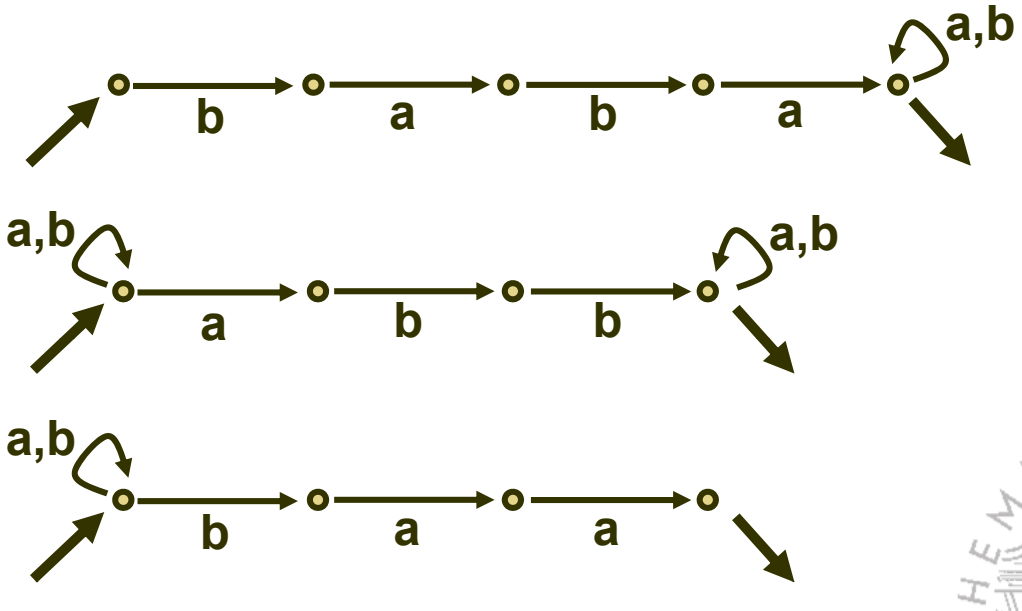


Úvod do nedeterminismu

Stav a písmeno určuje množinu možných dalších stavů!

Příklad:

L = { w | w=babau ∨ w=uabbv ∨ w=ubaa, u,v∈{a,b}* }



Nedeterministický konečný automat

Nedeterministickým konečným automatem nazýváme pětici $A = (Q,X,\delta,S,F)$, kde:

Q - konečná neprázdná množina stavů (*stavový prostor*)

X - konečná neprázdná množina symbolů (*vstupní abeceda*)

δ - zobrazení $Q \times X \to P(Q)$ (*přechodová funkce*)

$S \subseteq Q$ (*množina počátečních stavů*)

$F \subseteq Q$ (*množina přijímacích stavů*)

Reprezentace:

stavový diagram, tabulka, stavový strom

Jak se počítá s nedeterminismem?

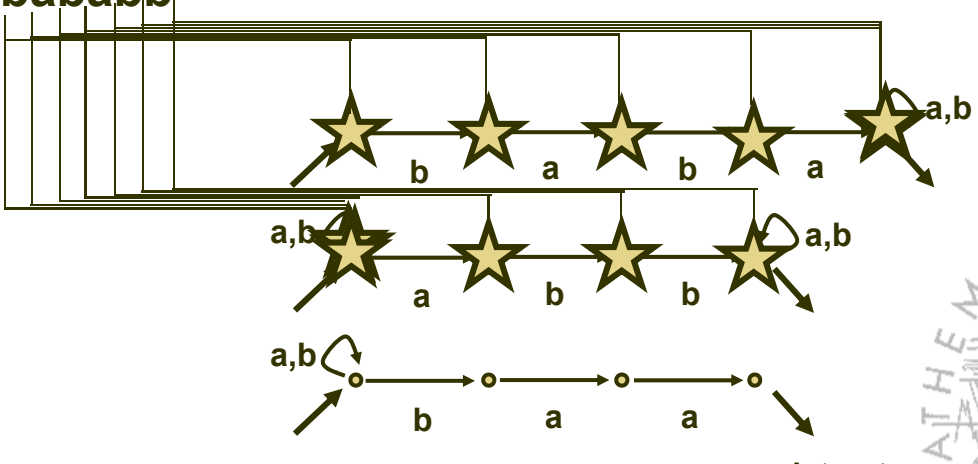
Slovo $w = x_1...x_n$ je *přijímáno* nedeterministickým konečným automatem, jestliže existuje posloupnost stavů $q_1,\ldots,q_{n+1}$ taková, že:

$q_1 \in S$, $q_{i+1} \in \delta(q_i, x_i)$ pro $i=1\ldots n$, $q_{n+1} \in F$.

Prázdné slovo je přijímáno právě když $S \cap F \neq \emptyset$

Přijímajících výpočtů pro dané slovo může být více!

Př. bababb



Determinismus vs. nedeterminismus

Konečný automat je speciálním případem nedeterministického konečného automatu!

Důsledek: Jazyky rozpoznávané konečnými automaty jsou rozpoznávané nedeterministickými konečnými automaty.

Platí i obrácené tvrzení?

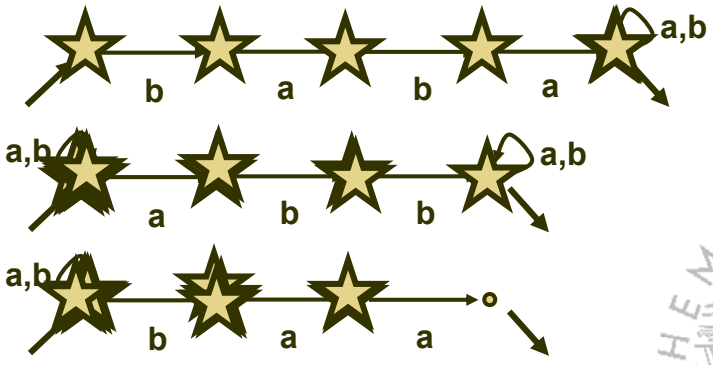
Zkusme to!

potřebujeme postupovat systematicky a s konečnou pamětí

pomocí značek na stavech simulujeme všechny možné výpočty

tzv. podmnožinová konstrukce

Př. bababb



Převod nedeterminismu na determinismus

Věta: Je-li A nedeterministický konečný automat, potom lze sestrojit konečný automat B takový, že L(A)=L(B).

Důkaz: (podmnožinová konstrukce)

nechť $A = (Q,X,\delta,S,F)$

potom definujme $B = (P(Q),X,\delta',S,F')$, kde

$F' = \{ K \mid K \in P(Q), K \cap F \neq \emptyset \}$

$\delta'(K,x) = \cup_{q \in K} \delta(q,x)$

1) B je definován korektně

2) L(A)=L(B)?

$\lambda \in L(A) \Leftrightarrow S \cap F \neq \emptyset \Leftrightarrow S \in F' \Leftrightarrow \lambda \in L(B)$

$L(A) \subseteq L(B)$

$w = x_1 \ldots x_n \in L(A) \Leftrightarrow \exists q_1,\ldots,q_{n+1} \in Q \;\; q_1 \in S, q_{i+1} \in \delta(q_i, x_i), q_{n+1} \in F$

položme $K_1 = S \;\; (q_1 \in K_1)$, $K_{i+1} = \delta'(K_i,x_i) \;\; (q_{i+1} \in K_{i+1})$, potom $K_{n+1} \in F'$

$L(B) \subseteq L(A)$

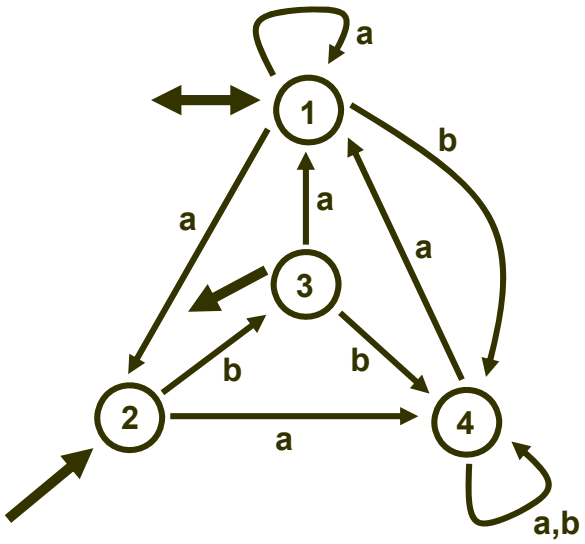
$w = x_1 \ldots x_n \in L(B) \Leftrightarrow \exists K_1,\ldots,K_{n+1} \in P(Q) \;\; K_1 = S, K_{i+1} = \delta'(K_i, x_i), K_{n+1} \in F'$

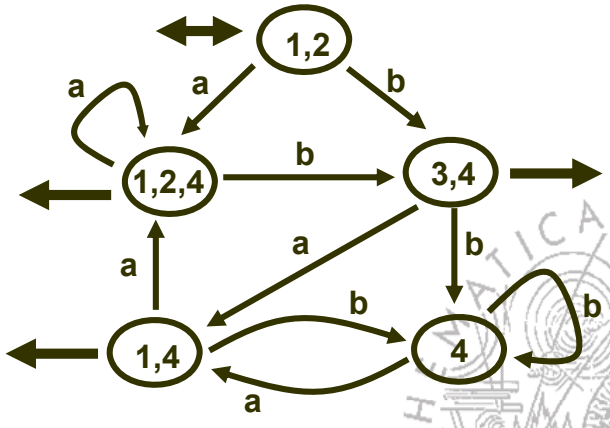
vezměme $q_{n+1} \in F \cap K_{n+1}$, $q_i \in K_i$ tž. $q_{i+1} \in \delta(q_i,x_i), \ldots, q_1 \in K_1 = S$

Ukázka převodu

	a	b
↔ 1	1,2	4
→ 2	4	3
← 3	1	4
4	1,4	4

	a	b
↔ {1,2}	{1,2,4}	{3,4}
← {1,2,4}	{1,2,4}	{3,4}
← {3,4}	{1,4}	{4}
← {1,4}	{1,2,4}	{4}
{4}	{1,4}	{4}





Poznámky k nedeterminismu

Význam:

- teoretický (např. při převodu gramatik na automaty)
- praktický (zjednodušení návrhu automatu)

U konečných automatů vede nedeterminismus ke stejné třídě jazyků jako determinismus.

- neplatí obecně (zásobníkové automaty)!

Převod na determinismus znamená (až) exponenciální nárůst počtu stavů ($Q \rightarrow P(Q)$).

- obecně je tento nárůst nezbytný!

  $L_n = \{ w \mid w \in \{0,1\}^*, w=u1v, |v|=n-1 \}$

- není potřeba explicitně převádět.

Existují také *zobecněné nedeterministické automaty*

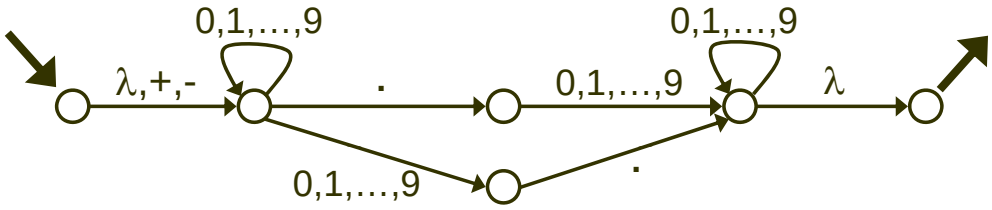
λ-přechod: změna stavu bez čtení vstupu

λ-přechody

Automat může změnit stav bez čtení symbolu.

Hodí se pro zjednodušení popisu automatu.

Příklad: rozpoznání čísla s desetinou tečkou s možností vynechání 0 před/za tečkou a prefixu +/-

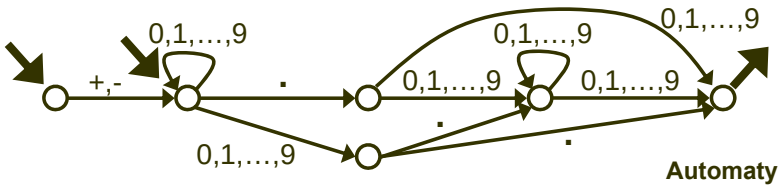


Odstranění λ-přechodů převodem na NKA

λ-uzávěr(q) = stav q a stavy, do kterých se z q dostaneme λ-přechody

nové počáteční stavy: λ-uzávěr(S)

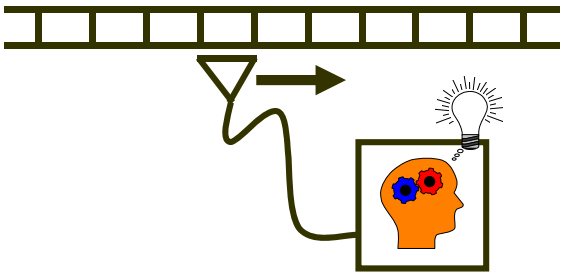
nové hrany: δ'(q, x) = λ-uzávěr(δ(q, x))



Můžeme konečné automaty ještě zobecnit?

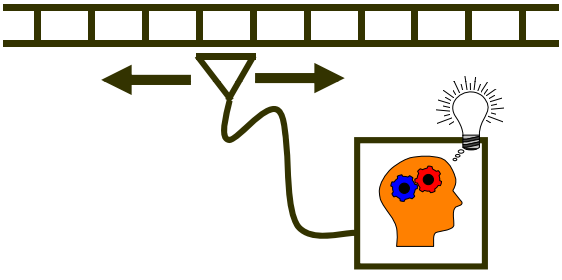
Konečný automat provádí následující činnosti:

- přečte písmeno
- změní stav vnitřní jednotky
- posune čtecí hlavu doprava



Čtecí hlava se nesmí vracet!

Co když automatu povolíme ovládání hlavy?



Pozor! Automat na pásku nic nepíše!

Dvousměrné (dvoucestné) konečné automaty

Dvousměrným (dvoucestným) konečným automatem nazýváme pětici $A = (Q,X,\delta,q_0,F)$, kde:

- Q - konečná neprázdná množina stavů (*stavový prostor*)
- X - konečná neprázdná množina symbolů (*vstupní abeceda*)
- δ - zobrazení $Q \times X \rightarrow Q \times \{-1,0,+1\}$ (*přechodová funkce*)
  - přechodová funkce určuje i pohyb čtecí hlavy
- $q_0 \in Q$ (*počáteční stav*)
- $F \subseteq Q$ (*množina přijímacích stavů*)

Reprezentace:

stavový diagram, tabulka, stavový strom

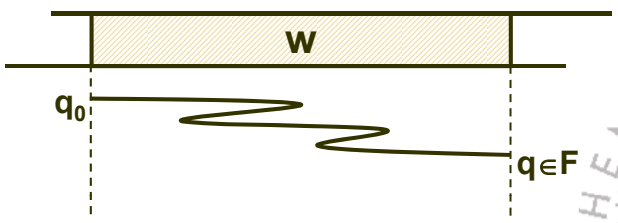
Počítání s dvousměrnými automaty

Kdy dvousměrný automat přijímá slovo?

Co se děje, je-li hlava mimo čtené slovo?

Slovo w je *přijato dvousměrným konečným automatem*, pokud:

- výpočet začal na prvním písmenu slova w vlevo v počátečním stavu
- čtecí hlava poprvé opustila slovo w vpravo v některém přijímacím stavu
- mimo čtené slovo není výpočet definován (výpočet zde končí a slovo není přijato)



Příklad dvousměrného automatu

Nejprve poznámka:

ke slovům si můžeme přidat speciální koncové znaky $\#\notin X$

je-li $L(A)= \{\#w\# \mid w\in L\subseteq X^*\}$ regulární, potom i L je regulární

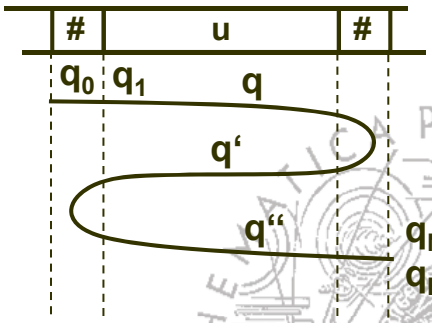
$L = \partial_{\#} \partial^{R}_{\#} (L(A) \cap \#X^*\#)$

Příklad:

$L(B) = \{\#u\# \mid uu\in L(A)\}$ Pozor! Toto není levý ani pravý kvocient!

Nechť $A= (Q,X,\delta,q_1,F)$, definujme dvousměrný konečný automat $B=(Q\cup Q'\cup Q''\cup \{q_0, q_N, q_F\}), X, \delta', q_0, \{q_F\})$ takto:

δ'	$x \in X$	#	poznámka
q_0	q_N,-1	q_1,+1	q_1 je počátek A
q	p,+1	q',-1	p= δ(q,x)
q'	q',-1	q'',+1	
q''	p'',+1	q_F,+1	$q \in F$, p= δ(q,x)
q''	p'',+1	q_N,+1	$q \notin F$, p= δ(q,x)
q_N	q_N,+1	q_N,+1	
q_F	q_N,+1	q_N,+1	



Věta o dvousměrných automatech

Jazyky přijímané dvousměrnými konečnými automaty jsou právě jazyky přijímané konečnými automaty.

Možnost pohybovat čtecí hlavou po pásce nezvětšila sílu konečného automatu!

Pozor, na pásku nic nepíšeme!

Pokud můžeme na pásku psát, dostaneme Turingův stroj.

Zřejmé: konečný automat $\rightarrow$ dvousměrný konečný automat

dvousměrný automat vždy posouvá hlavu doprava

KA $A=(Q,X,\delta,q_0,F) \rightarrow$ 2KA $B=(Q,X,\delta',q_0,F)$, $\delta'(q,x)=(\delta(q,x),+1)$

Zbývá: dvousměrný konečný automat $\rightarrow$ konečný automat

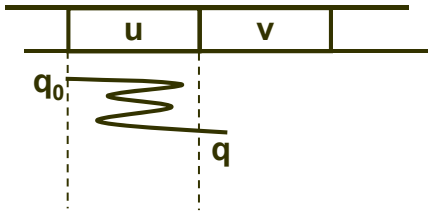
Důkaz věty o dvousměrných automatech (1)

1) Formální popis vlivu slova *u* na výpočet nad slovem *v*

(i) kdy poprvé opustíme slovo *u* vpravo
(v jakém stavu poprvé vstoupíme nad *v*)

$f(q'_0) = q$ poprvé přejdeme na *v* ve stavu q

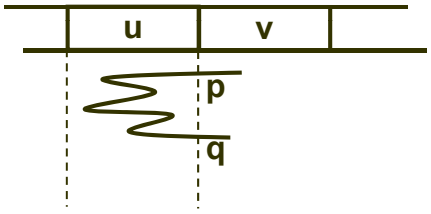
$f(q'_0) = 0$ nikdy neopustíme *u* vpravo



(ii) pokud opustíme slovo *v* vlevo, kdy se nad *v* opět vrátíme

$f(p) = q$ vrátíme se nad *v* ve stavu q

$f(p) = 0$ nikdy už se nevrátíme



2) Výpočet nad *u* máme popsaný funkcí f_u

$f_u: Q \cup \{q'_0\} \to Q \cup \{0\}$

$f_u(q'_0)$ popisuje situaci (i): v jakém stavu poprvé odejdeme vpravo, pokud začneme výpočet vlevo v počátečním stavu q_0

$f_u(p)$ $(p \in Q)$ popisuje situaci (ii): v jakém stavu opět odejdeme vpravo, pokud začneme výpočet vpravo v p

symbol 0 značí, že daná situace nenastane (odejdeme vlevo nebo cyklus)

Důkaz věty o dvousměrných automatech (2)

Pro každé slovo *u* máme funkci f_u popisující výpočet dvousměrného automatu A nad *u*

Definujme ekvivalenci slov takto: $u \sim w \Leftrightarrow_{def} f_u = f_w$

tj. slova jsou ekvivalentní, pokud mají stejné „výpočtové“ funkce

Vlastnosti ~:

- je to ekvivalence (zřejmé, definováno pomocí =)
- má konečný index (maximální počet různých funkcí je $(n+1)^{n+1}$ pro n-stavový dvousměrný automat)
- je to pravá kongruence (zřejmě $u \sim w \Rightarrow uv \sim wv$, protože rozhraní u|v a w|v je stejné a nad v se automat chová stejně)
- L(A) je sjednocením jistých tříd rozkladu X*/~

  stačí si uvědomit, že $w \in L(A) \Leftrightarrow f_w(q'_0) \in F$

  $u \sim w \Rightarrow f_u(q'_0) = f_w(q'_0) \Rightarrow (u \in L(A) \Leftrightarrow w \in L(A))$

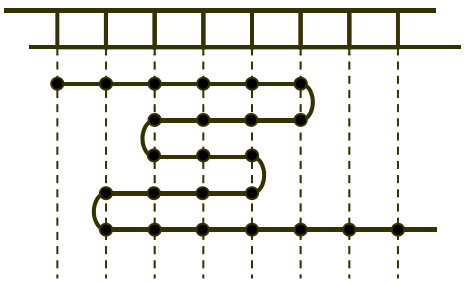
Podle Nerodovy věty je L(A) regulární jazyk.

Převod 2KA na NKA

Konstruktivní důkaz věty o dvousměrných automatech.

Jak výpočet s návraty převést na lineární výpočet?

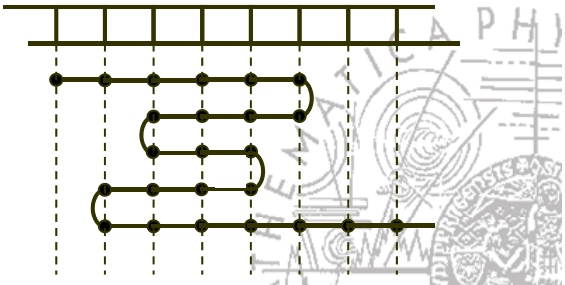
- zajímají nás jen přijímací výpočty
- díváme se na přechody mezi symboly (v jakém stavu se přechází na další políčko)



Pozorování:

- stavy se v přechodu (řezu) střídají (doprava/doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav

1. Najdeme všechny možné řezy - posloupnosti stavů (je jich konečně mnoho).
2. Mezi řezy definujeme (nedeterministické) přechody podle čteného symbolu.
3. Rekonstruujeme výpočet skládáním řezů (jako puzzle).



Formální převod 2KA na NKA

Nechť $A=(Q,X,\delta,q_0,F)$ je dvousměrný konečný automat.

Definujme ekvivalentní nedeterministický konečný automat $B=(Q',X,\delta',(q_0),F')$, kde:

Q' = všechny korektní přechodové posloupnosti

posloupnosti stavů $(q^1,\ldots,q^k)$ z Q takové, že

- délka posloupnosti je lichá (k=2m+1)
- žádný stav se neopakuje na liché ani na sudé pozici $(\forall i\neq j\ q^{2i}\neq q^{2j}) \wedge (\forall i\neq j\ q^{2i+1}\neq q^{2j+1})$

$F' = \{(q) \mid q\in F\}$ přechodové posloupnosti délky 1 obsahující koncový stav

$\delta'(c,x) = \{ d \mid d\in Q' \wedge c\rightarrow d$ je lokálně konzistentní přechod pro x$\}$

L(A)=L(B)?

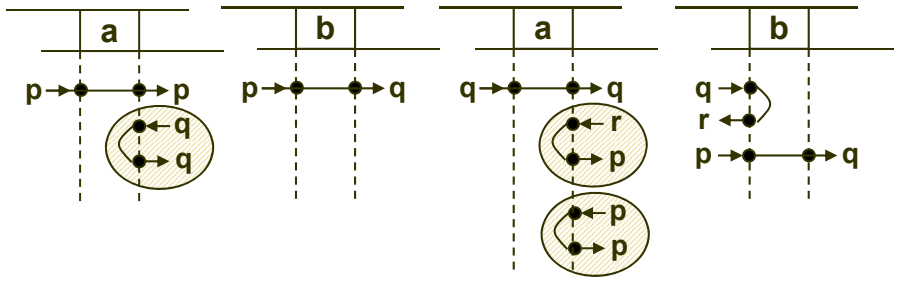
„trajektorie 2KA A odpovídá řezům KA B“

Příklad převodu 2KA na NKA

Mějme následující dvousměrný konečný automat:

	a	b
→ p	p,+1	q,+1
← q	q,+1	r,-1
r	p,+1	r,-1

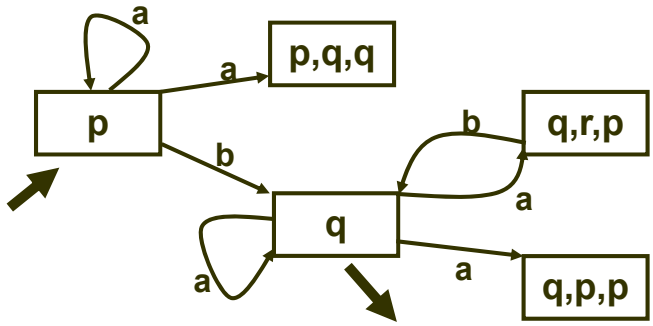
Možné řezy a jejich konzistentní přechody:



Ukázka výpočtu:

```
aabaabaabb
pppqqq
      r
       pqqq
          r
           pq
          rr
           pq
          rr
           . .
```

Výsledný nedeterministický KA:



Množinové operace nad jazyky

Sjednocení jazyků $L_1 \cup L_2 = \{ w \mid w \in L_1 \vee w \in L_2 \}$

Příklad: jazyk obsahuje slova začínající baba nebo končící baa

Průnik jazyků $L_1 \cap L_2 = \{ w \mid w \in L_1 \wedge w \in L_2 \}$

Příklad: jazyk obsahuje slova se sudým počtem nul a každý symbol 1 je bezprostředně následován 0

Rozdíl jazyků $L_1 - L_2 = \{ w \mid w \in L_1 \wedge w \notin L_2 \}$

Příklad: jazyk obsahuje slova začínající baba a neobsahující abb

Doplněk jazyka $-L = \{ w \mid w \notin L \} = X^* - L$

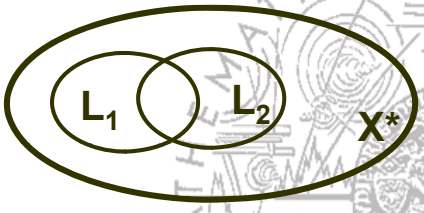
Příklad: slova jazyka neobsahují posloupnost tří symbolů 1

Platí tradiční de Morganova pravidla

$L_1 \cap L_2 = -(-L_1 \cup -L_2)$

$L_1 \cup L_2 = -(-L_1 \cap -L_2)$

$L_1 - L_2 = L_1 \cap -L_2$



Uzavřenost na množinové operace

Nechť L_1 a L_2 jsou jazyky rozpoznávané konečnými automaty.
Potom $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$ a $-L_1$ jsou také jazyky rozpoznávané konečnými automaty (třída $\mathcal{F}$ je uzavřena na uvedené operace).

Konstruktivní důkaz:

doplněk

stačí prohodit koncové a nekoncové stavy přijímajícího det. automatu

sjednocení, průnik a rozdíl

idea: paralelní běh přijímajících automatů

$A_1 = (Q_1,X,\delta_1,q_1,F_1)$, $A_2 = (Q_2,X,\delta_2,q_2,F_2)$

uděláme spojený automat $A = (Q,X,\delta,q,F)$

$Q = Q_1 \times Q_2$, $q = (q_1,q_2)$

$\delta((p_1,p_2),x) = (\delta_1(p_1,x), \delta_2(p_2,x))$

sjednocení $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

průnik $F = F_1 \times F_2$

rozdíl $F = F_1 \times (Q_2 - F_2)$

Množinové operace v příkladě

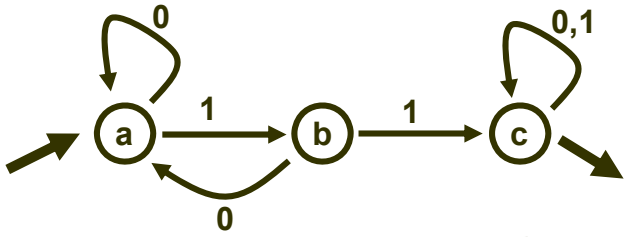
Navrhněte konečný automat přijímající slova, která obsahují 3k+2 symbolů 1 a neobsahují posloupnost 11.

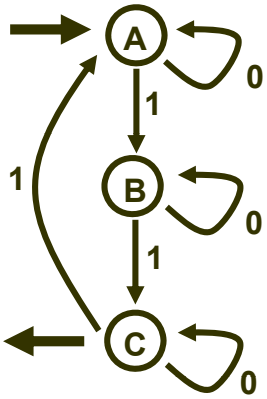
Přímá konstrukce komplikovaná!

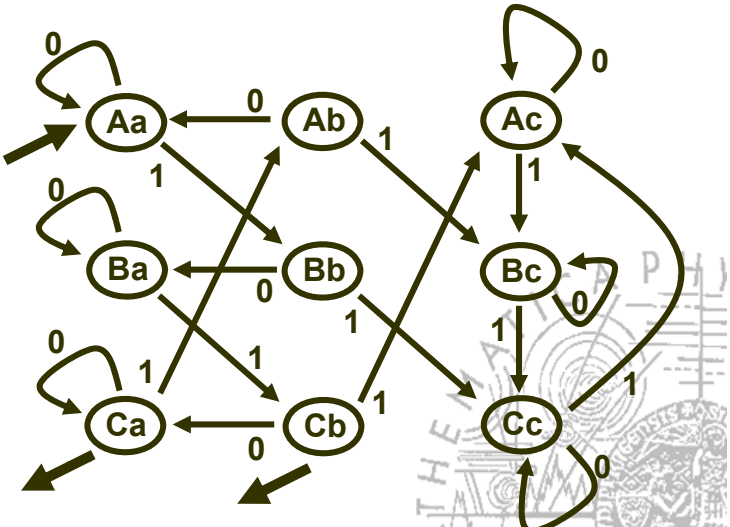
$L_1 = \{w \mid w \in \{0,1\}^* \wedge |w|_1 = 3k+2\}$

$L_2 = \{w \mid u,v \in \{0,1\}^* \wedge w = u11v\}$

$L = L_1 - L_2$



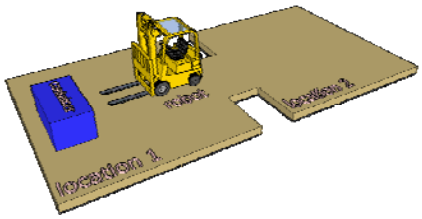


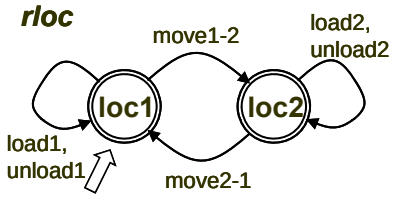


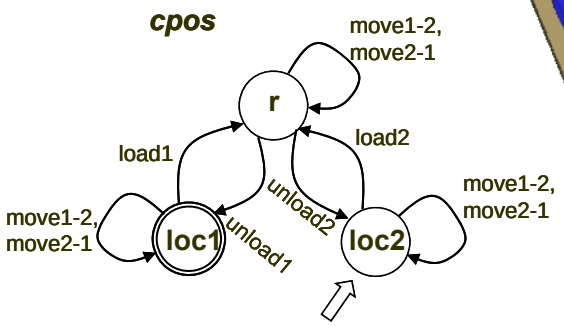
K čemu to je?

Můžeme operace s automaty někde přímo využít?

Například v plánování, kde automat popisuje, jak se mění hodnota nějaké stavové proměnné.

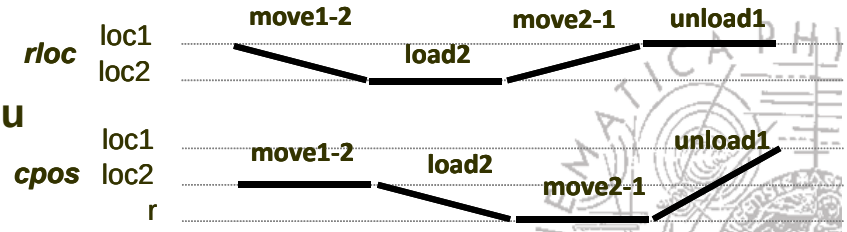






Plán se potom může hledat jako průnik automatů.

V každém stavovém diagramu se provede stejná posloupnost akcí.



Řetězcové operace nad jazyky

Zřetězení jazyků	$L_1 . L_2 = \{ uv \mid u \in L_1 \wedge v \in L_2 \}$
Mocniny jazyka	$L^0 = \{\lambda\}$
	$L^{i+1} = L^i . L$
Pozitivní iterace	$L^+ = L^1 \cup L^2 \ldots = \cup_{i \geq 1} L^i$
Obecná iterace	$L^* = L^0 \cup L^1 \ldots = \cup_{i \geq 0} L^i$
zřejmě $L^* = L^+ \cup \{\lambda\}$	
Otočení jazyka	$L^R = \{ u^R \mid u \in L \}$
reverse, zrcadlový obraz	$(x_1 x_2 \ldots x_n)^R = x_n \ldots x_2 x_1$
Levý kvocient L_1 podle L_2	$L_2 \setminus L_1 = \{ v \mid uv \in L_1 \wedge u \in L_2 \}$
Levá derivace L podle w	$\partial_w L = \{w\} \setminus L$
Pravý kvocient L_1 podle L_2	$L_1 / L_2 = \{ u \mid uv \in L_1 \wedge v \in L_2 \}$
Pravá derivace L podle w	$\partial^R_w L = L / \{w\}$

Uzavřenost zřetězení

$L_1, L_2 \in \mathcal{F} \Rightarrow L_1.L_2 \in \mathcal{F}$

idea:

nejprve počítá automat $A_1=(Q_1,X,\delta_1,q_1,F_1)$ potom $A_2=(Q_2,X,\delta_2,q_2,F_2)$

realizace:

pomocí nedeterministického konečného automatu $B=(Q,X,\delta,S,F)$

nedeterminismus slouží při rozhodování kdy přepnout do A_2

$Q = Q_1 \cup Q_2$ (předpokládáme různá jména stavů, jinak přejmenuj)

$S = \{q_1\}$ pokud $\lambda \notin L_1$ ($q_1 \notin F_1$)

$= \{q_1, q_2\}$ pokud $\lambda \in L_1$ ($q_1 \in F_1$), tj. rovnou přejdeme také do A_2

$F = F_2$ končíme až po přečtení slova z L_2

$\delta(q,x) = \{\delta_1(q,x)\}$ pokud $q \in Q_1 \wedge \delta_1(q,x) \notin F_1$ (počítáme v A_1)

$= \{\delta_1(q,x), q_2\}$ pokud $q \in Q_1 \wedge \delta_1(q,x) \in F_1$ (přechod A_1 do A_2)

$= \{\delta_2(q,x)\}$ pokud $q \in Q_2$ (počítáme v A_2)

DCV: ověřit $L(B) = L(A_1) . L(A_2)$

Uzavřenost iterace

$L \in \mathcal{F} \Rightarrow L^* \in \mathcal{F}$

idea: opakovaný výpočet automatu $A=(Q,X,\delta,q_0,F)$

realizace: nedeterministické rozhodnutí, zda pokračovat nebo restart

pozor! $\lambda \in L^*$ i když $\lambda \notin L$, řešíme pomocí speciálního stavu

hledáme nedeterministický automat $B=(Q',X,\delta',S,F')$

$Q' = Q \cup \{s\}$ přidáme nový stav pro příjem λ

$S = \{q_0, s\}$ nový stav

$F' = F \cup \{s\}$ končíme po přečtení slova z L nebo v s (pro λ)

$\delta'(q,x) = \{\delta(q,x)\}$ pokud $q \in Q \wedge \delta(q,x) \notin F$ (počítáme uvnitř A)

$= \{\delta(q,x), q_0\}$ pokud $q \in Q \wedge \delta(q,x) \in F$ (možný restart)

$\delta'(s,x) = \{\}$ žádné přechody z nového stavu

$L \in \mathcal{F} \Rightarrow L^+ \in \mathcal{F}$

stejná konstrukce, pouze bez použití stavu s

Uzavřenost reverse

$L \in \mathcal{F} \Leftrightarrow L^R \in \mathcal{F}$

zřejmě $(L^R)^R = L$, a tedy stačí ukázat $L \in \mathcal{F} \Rightarrow L^R \in \mathcal{F}$

idea: obrátíme „šipky“ ve stavovém diagramu

realizace: nedeterministický konečný automat

$A=(Q,X,\delta,q_0,F) \quad \rightarrow \quad B=(Q,X,\delta',F,\{q_0\})$

$\delta'(q,x) = \{p \mid \delta(p,x)=q\} \qquad (\delta(p,x)=q \Leftrightarrow p \in \delta'(q,x))$

$w = x_1x_2\ldots x_n$

$q_0, q_1, \ldots q_n$ je přijímající výpočet pro w automatu A

tj. $\delta(q_i,x_{i+1})=q_{i+1}$ a $q_n \in F$

$\Leftrightarrow$

$q_n, q_{n-1}, \ldots q_0$ je přijímající výpočet pro w^R automatu B

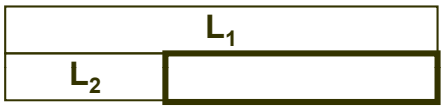
$q_i \in \delta'(q_{i+1},x_{i+1})$

Poznámka:

někdy L nebo L^R má výrazně jednodušší přijímající automat

Uzavřenost kvocientu

$L_1,L_2 \in \mathcal{F} \Rightarrow L_2 \setminus L_1 \in \mathcal{F}$



idea: automat A_1 budeme startovat ve stavech, do kterých se lze dostat slovem z L_2

realizace: nedeterministický automat B „téměř“ totožný s A_1 (rozdíl ve startovních stavech)

$S = \{q \mid q \in Q_1 \; \exists u \in L_2 \; q=\delta_1(q_1,u)\}$ nové startovní stavy

lze nalézt algoritmicky ($A_q=(Q_1,X,\delta_1,q_1,\{q\})$, pak $q \in S \Leftrightarrow L(A_q) \cap L_2 \neq \emptyset$)

$v \in L_2 \setminus L_1$

$\Leftrightarrow \exists u \in L_2 \; uv \in L_1$

$\Leftrightarrow \exists u \in L_2 \; \exists q \in Q_1 \;\; \delta_1(q_1,u)=q \wedge \delta_1(q,v) \in F_1$

$\Leftrightarrow \exists q \in S \;\; \delta_1(q,v) \in F_1$

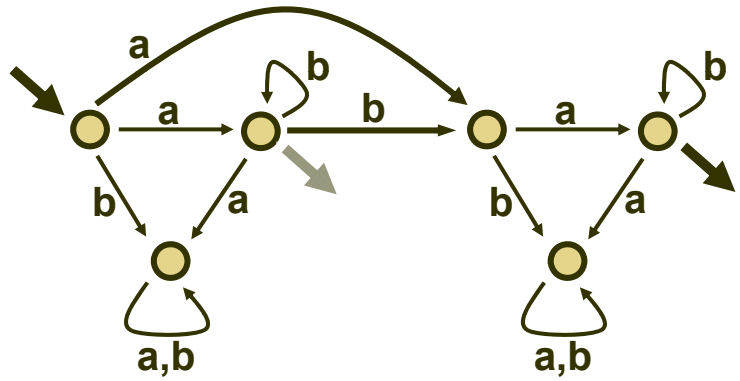
$\Leftrightarrow v \in L(B)$

$L_1,L_2 \in \mathcal{F} \Rightarrow L_1 / L_2 \in \mathcal{F}$

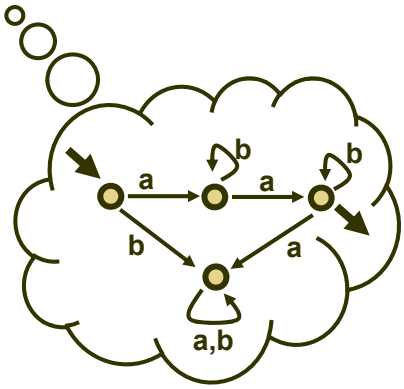
obdobně nebo pomocí $L_1 / L_2 = (L_2^R \setminus L_1^R)^R$

Příklady řetězcových operací

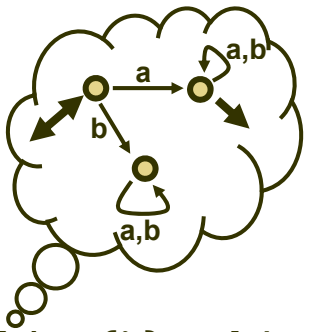
$L = \{ab^i , i \geq 0\}$



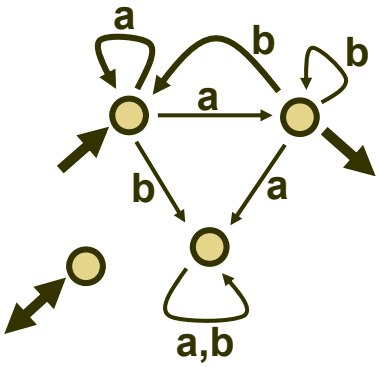
$L.L = \{ab^iab^j, i \geq 0, j \geq 0\}$

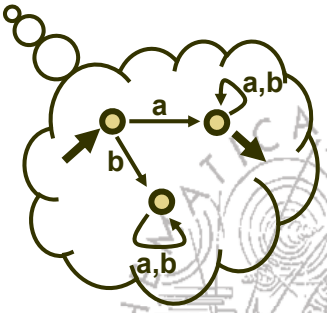


$L^+ = \{ab^{i_1} ab^{i_2} \ldots ab^{i_n}, n>0, i_j \geq 0\}$



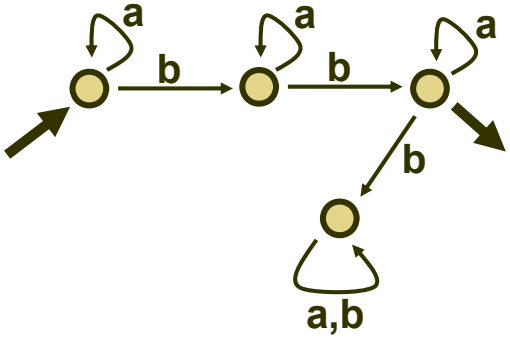
$L^* = \{\lambda\} \cup L^+$



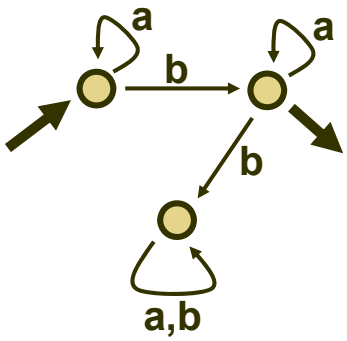


Příklad kvocientu

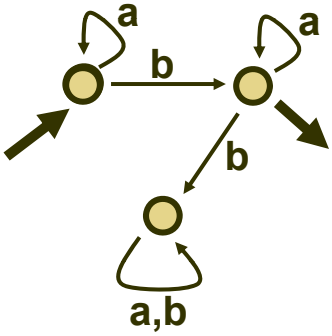
$L_1 = \{a^{i_1}ba^{i_2}ba^{i_3}, i_j \geq 0\}$



$L_2 = \{a^{i_1}ba^{i_2}, i_j \geq 0\}$



$L_2 \setminus L_1 = \{a^{i_1}ba^{i_2}, i_j \geq 0\} = L_2$



Substituce jazyků

nechť X je konečná abeceda

pro každé $x \in X$ budiž $\sigma(x)$ jazyk v nějaké abecedě Y_x

dále položme:

$\sigma(\lambda) = \{\lambda\}$

$\sigma(u.v) = \sigma(u) \,.\, \sigma(v)$

zobrazení $\sigma: X^* \to P(Y^*)$, kde $Y = \cup_{x \in X} Y_x$ se nazývá *substituce*

$\sigma(L) = \cup_{w \in L} \sigma(w)$

nevypouštějící substituce, žádné $\sigma(x)$ neobsahuje λ

Příklad:

$\sigma(0) = \{a^i b^j, i,j \geq 0\}$, $\sigma(1) = \{cd\}$

$\sigma(010) = \{a^i b^j c d a^k b^l, i,j,k,l \geq 0\}$

homomorfismus h: $h(x) = w_x$ (speciální případ substituce)

nevypouštějící homomorfismus: $w_x \neq \lambda$

Věta: $L \in \mathcal{F}$, $\forall x \in X\ \sigma(x) \in \mathcal{F} \Rightarrow \sigma(L), h(L), h^{-1}(L) \in \mathcal{F}$

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

Poznámky k uzávěrovým vlastnostem

Zjednodušení návrhu automatů

$L.\varnothing = \varnothing.L = \varnothing$

$\{\lambda\}.L = L.\{\lambda\} = L$

$(L^*)^* = L^*$

$(L_1 \cup L_2)^* = L_1^*(L_2.L_1^*)^* = L_2^*(L_1.L_2^*)^*$

$(L_1.L_2)^R = L_2^R \,.\, L_1^R$

$\partial_w(L_1 \cup L_2) = \partial_w L_1 \cup \partial_w L_2$

$\partial_w(X^* - L) = X^* - \partial_w L$

$h(L_1 \cup L_2) = h(L_1) \cup h(L_2)$

Důkaz regulárnosti

$L = \{w \mid w \in \{0,1\}^*, |w|_1 = |w|_0\}$ není regulární

$L \cap \{0^i 1^j, i,j \geq 0\} = \{0^i 1^i, i \geq 0\}$

Trochu motivace

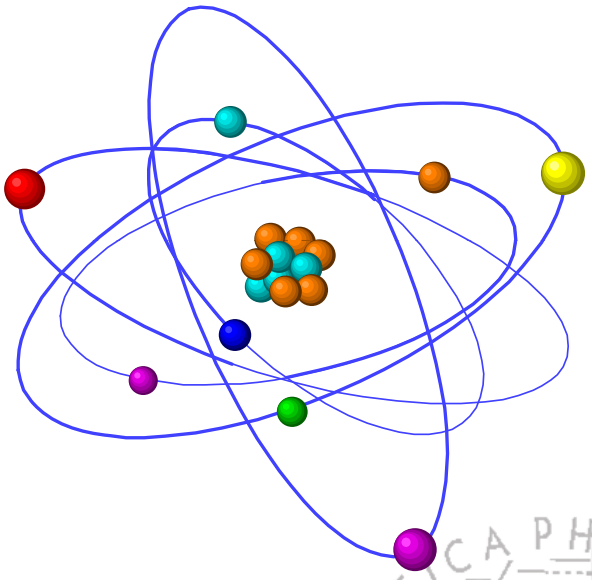
$L = \{ w \mid w=babau \vee w=uabbv \vee w=ubaa, u,v \in \{a,b\}^* \}$

$L = L_1 \cup L_2 \cup L_3$, kde

$L_1 = \{ w \mid w=babau, u \in \{a,b\}^* \}$,

$L_2 = \{ w \mid w=uabbv, u,v \in \{a,b\}^* \}$

$L_3 = \{ w \mid w=ubaa, u \in \{a,b\}^* \}$



Můžeme jít ještě dál!

$L_1 = \{baba\} . \{a,b\}^*$

$L_2 = \{a,b\}^* . \{abb\} . \{a,b\}^*$

$L_3 = \{a,b\}^* . \{baa\}$

Pojďme ještě dál

$L_3 = (\{a\} \cup \{b\})^* . \{b\}.\{a\}.\{a\}$

Nešlo by všechny regulární jazyky „poskládat“ z nějakých triviálních jazyků?

Regulární jazyky

Třída regulárních jazyků RJ(X) nad konečnou neprázdnou abecedou X je nejmenší třída jazyků, která:

- obsahuje prázdný jazyk $\varnothing$
- pro každé písmeno $x \in X$ obsahuje jazyk $\{x\}$
- $A,B \in RJ(X) \Rightarrow A \cup B \in RJ(X)$ uzavřená na sjednocení
- $A,B \in RJ(X) \Rightarrow A.B \in RJ(X)$ uzavřená na zřetězení
- $A \in RJ(X) \Rightarrow A^* \in RJ(X)$ uzavřená na iteraci

Vlastně algebraický popis jazyků!

Speciálně:

$\{\lambda\} \in RJ(X)$ protože $\{\lambda\} = \varnothing^*$

$X \in RJ(X)$ protože $X = \cup_{x \in X} \{x\}$ (pozor! je to konečné sjednocení)

$\{x_{i_1},\ldots,x_{i_k}\} \in RJ(X)$

$X^* \in RJ(X)$

Kleeneova věta

Libovolný jazyk je regulární právě když je rozpoznatelný konečným automatem.

Konečnými automaty lze rozpoznávat jen triviální jazyky (prázdný a jednopísmenné) a jazyky, které z nich lze složit operacemi sjednocení, zřetězení a iterace.

Důkaz RJ $\Rightarrow \mathcal{F}$

regulární jazyky jsou rozpoznatelné konečnými automaty

- **triviální jazyky jsou rozpoznatelné konečným automatem**
- **operace sjednocení, zřetězení a iteraci dávají opět jazyk rozpoznatelný konečným automatem**

Důkaz Kleeneovy věty

jazyky rozpoznatelné konečnými automaty jsou regulární

máme automat $A=(Q,X,\delta,q_1,F)$, který přijímá jazyk $L(A)$
chceme ukázat, že $L(A)$ dostaneme z elementárních jazyků a operací

definujme $R_{ij} = \{w \in X^* | \delta^*(q_i,w)=q_j\}$ — slova převádějící stav q_i na q_j

potom $L(A) = \cup_{q_i \in F} R_{1i}$ — slova převádějící počáteční stav q_1 na nějaký koncový stav q_i

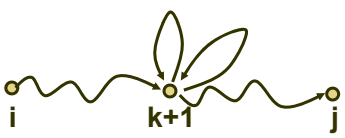
jsou jazyky R_{ij} regulární?

pokud ano, potom $L(A)$ je také regulární, protože $\cup$ zachovává regulárnost

definujme R^k_{ij} =slova převádějící stav q_i na q_j bez meziprůchodu stavy q_m $m>k$
zřejmě $R_{ij} = R^n_{ij}$ (n je počet stavů automatu)

jsou jazyky R^k_{ij} regulární?

- **R^0_{ij} je regulární (žádné mezistavy, tj. maximálně jednopísmenná slova)**
- **$R^{k+1}_{ij} = R^k_{ij} \cup R^k_{i,k+1}.(R^k_{k+1,k+1})^*. R^k_{k+1,j}$ je regulární (sjednocení a iterace regulárních jazyků)**

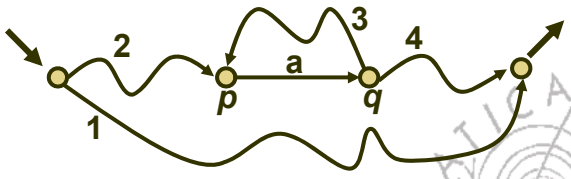


Alternativní důkaz Kleeneovy věty

jazyky rozpoznatelné konečnými automaty jsou regulární

Indukcí podle počtu hran v nedeterministickém automatu $A = (Q,X,\delta,S,F)$ pro daný jazyk $L(A)$

- *žádná hrana*
  - pouze jazyky $\varnothing$ nebo $\{\lambda\}$
- *(n+1) hran*
  - vybereme si jednu hranu: $p \rightarrow^a q$ tj. $q \in \delta(p,a)$
  - sestrojíme čtyři automaty bez této hrany (δ')
    - $A_1 = (Q,X,\delta',S,F)$
    - $A_2 = (Q,X,\delta',S,\{p\})$
    - $A_3 = (Q,X,\delta',\{q\},\{p\})$
    - $A_4 = (Q,X,\delta',\{q\},F)$



Potom $L(A) = L(A_1) \cup (L(A_2).a).(L(A_3).a)^*L(A_4)$

Jazyky $L(A_1)$, $L(A_2)$, $L(A_3)$, $L(A_4)$ jsou regulární (n hran)

Regulární výrazy

Množina regulárních výrazů RV(X) nad konečnou neprázdnou abecedou $X=\{x_1,\ldots,x_n\}$ je nejmenší množina slov v abecedě $\{x_1,\ldots,x_n,\varnothing, \lambda, +, . ,^*, (,)\}$, která:

- obsahuje výraz $\varnothing$ a výraz λ $\varnothing \in RV(X)$, $\lambda \in RV(X)$
- pro každé písmeno $x \in X$ obsahuje výraz x $x \in RV(X)$
- $\alpha,\beta \in RV(X) \Rightarrow (\alpha+\beta) \in RV(X)$
- $\alpha,\beta \in RV(X) \Rightarrow (\alpha.\beta) \in RV(X)$
- $\alpha \in RV(X) \Rightarrow \alpha^* \in RV(X)$

Příklad: ((a+((b.c)+d)*)+e)

Konvence:

- vnější závorky lze vynechat (a+((b.c)+d)*)+e
- závorky lze vynechat u . a + díky asociativitě a+((b.c)+d)*+e
- tečku lze vynechat a+((bc)+d)*+e
- priorita operací (nejvyšší) *, . , + (nejnižší) a+(bc+d)*+e

Hodnota regulárního výrazu

Hodnotou regulárního výrazu $\alpha \in RV(X)$ **je množina slov** $[\alpha]$ **(jazyk) definovaná následovně:**

- $[\emptyset] = \emptyset$, $[\lambda] = \{\lambda\}$, $[x] = \{x\}$
- $[(\alpha+\beta)] = [\alpha] \cup [\beta]$
- $[(\alpha.\beta)] = [\alpha] \,.\, [\beta]$
- $[\alpha^*] = [\alpha]^*$

Regulární výrazy odpovídají regulárním jazykům

- **hodnotou regulárního výrazu je regulární jazyk**
- **každý regulární jazyk lze reprezentovat pomocí regulárního výrazu (jazyk je hodnotou tohoto výrazu)**

Příklady:

$[baba(a+b)^* + (a+b)^*abb(a+b)^* + (a+b)^*baa] =$

$= \{ w \mid w=babau \vee w=uabbv \vee w=ubaa,\ u,v \in \{a,b\}^* \}$

$[(0^*10^*10^*1)^*0^*] =$

$= \{w \mid w \in \{0,1\}^* , |w|_1 = 3k \}$

Použití regulárních výrazů

Praktický

přehledný zápis jazyka

Teoretický

zjednodušení některých důkazů

Věta: $L \in \mathcal{F},\ \forall x \in X\ \sigma(x) \in \mathcal{F} \ \Rightarrow\ \sigma(L) \in \mathcal{F}$

L a $\sigma(x)$ **jsou regulární jazyky, lze je tedy reprezentovat regulárními výrazy**

každý výskyt x ve výrazu pro L stačí nahradit výrazem pro $\sigma(x)$

Rozšířené regulární výrazy

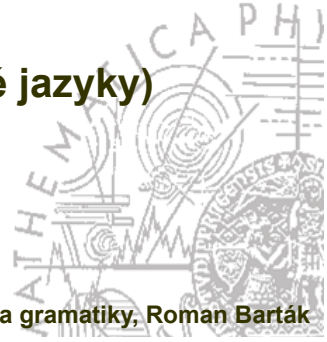
máme i další „regulární“ operace, např. průnik $(\alpha \,\&\, \beta)$

Ekvivalence regulárních výrazů

$\alpha \equiv \beta$ **jestliže** $[\alpha] = [\beta]$ **(tj. výrazy reprezentují stejné jazyky)**

Příklad: $(0^*1)^* \equiv \lambda + (0+1)^*1$

Jak to zjistíme?



Převod regulárního výrazu na konečný automat

Metoda 1 (inkrementální):

- převeď elementární jazyky (prázdný, jednopísmenné)
- spoj použitím regulárních operací podle výrazu

Metoda 2 (přímá)

a+(bc+d)*+a

- očísluj symboly ve výrazu (zleva do doprava) — $a_1+(b_2c_3+d_4)^*+a_5$
- zjisti všechny možné páry symbolů, které se mohou vyskytovat za sebou — b_2c_3, c_3d_4, c_3b_2, d_4d_4, d_4b_2
- zjisti symboly, které mohou být první ve slově — a_1, b_2, d_4, a_5
- zjisti symboly, které mohou být poslední ve slově — a_1, c_3, d_4, a_5
- zjisti, zda jazyk obsahuje prázdné slovo — ANO
- vytvoř nedeterministický automat

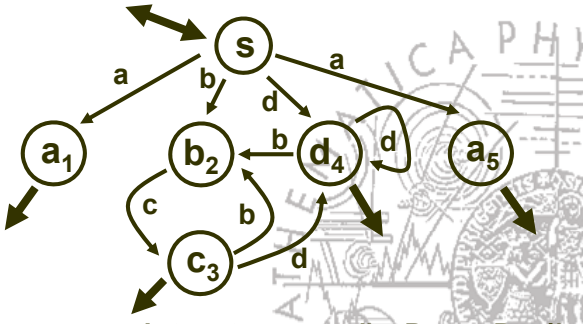
  stavy: s + očíslované symboly

  počátek = s

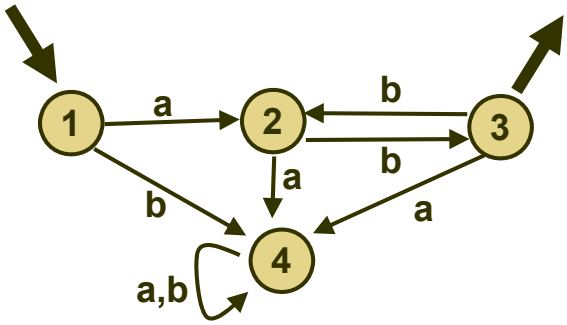
  konec = poslední symboly (+s pro λ)

  přechody: s→první symbol

  $x_i \rightarrow x_j$, pokud je pár $x_i x_j$



Od automatu k regulárnímu výrazu



Pomocí Kleeneovy věty:

- R^0_{ij}
- $R^{k+1}_{ij} = R^k_{ij} \cup R^k_{i,k+1}.(R^k_{k+1,k+1})^*.\ R^k_{k+1,j}$

Pozn.: uzel 4 můžeme ignorovat (nevedou přes něj žádné cesty do ostatních uzlů)

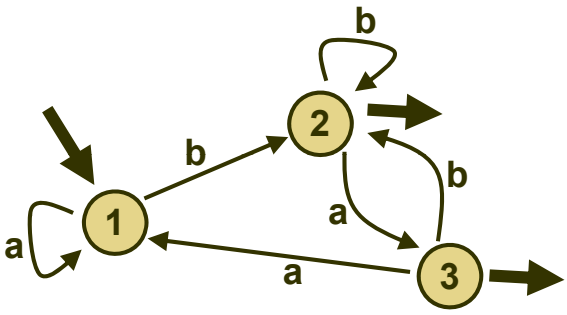
R^0	1	2	3
1	λ	a	∅
2	∅	λ	b
3	∅	b	λ

R^1	1	2	3
1	λ	a	∅
2	∅	λ	b
3	∅	b	λ

R^2	1	2	3
1	λ	a	ab
2	∅	λ	b
3	∅	b	$\lambda+b^2$

R^3	1	2	3
1	λ	$a(b^2)^*$	$ab(b^2)^*$
2	∅	$(b^2)^*$	$b(b^2)^*$
3	∅	$b(b^2)^*$	$(b^2)^*$

Od automatu k regulárnímu výrazu (příklad 2)



Pomocí Kleeneovy věty:

- R^0_{ij}
- $R^{k+1}_{ij} = R^k_{ij} \cup R^k_{i,k+1}.(R^k_{k+1,k+1})^*. R^k_{k+1,j}$

R^0	1	2	3
1	a+λ	b	∅
2	∅	b+λ	a
3	a	b	λ

R^1	1	2	3
1	a*	a*b	∅
2	∅	b+λ	a
3	a^+	a*b	λ

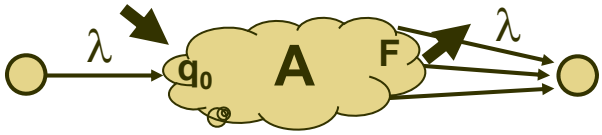
R^2	1	2	3
1	a*	a^*b^+	a^*b^+a
2	∅	b*	b*a
3	a^+	a^*b^+	a^*b^+a

R^3	1	2	3
1	?	(a+b)*b	(a+b)*ba
2	?	?	?
3	?	?	?

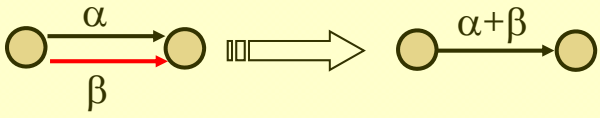
Od automatu k regulárnímu výrazu jinak

Ohodnocení hran regulárním výrazem

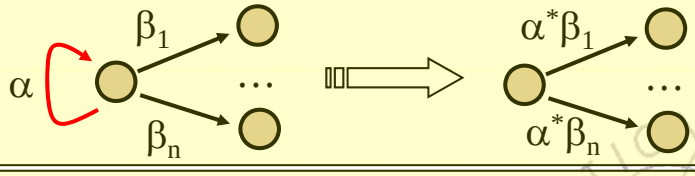
Nejprve vytvoříme automat s jedním vstupem a jedním výstupem



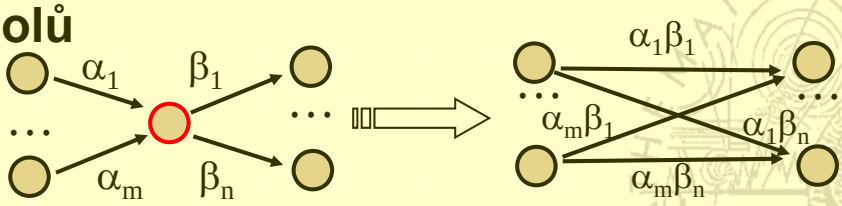
- **spojení hran**



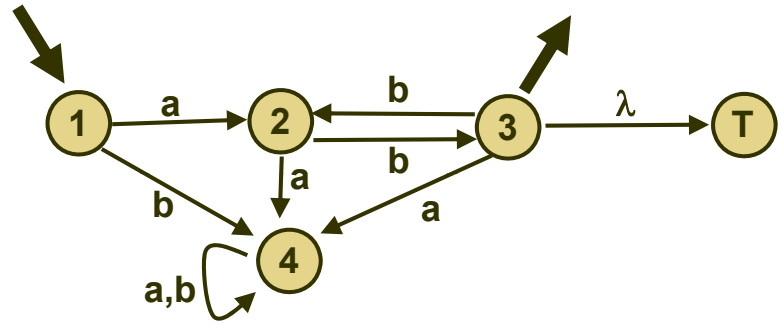
- **eliminace smyček**



- **eliminace vrcholů**



Od automatu k regulárnímu výrazu v příkladě

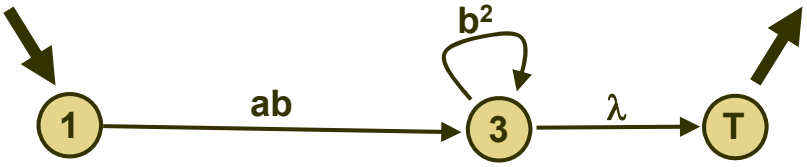


Stačí přidat pouze nový koncový stav.

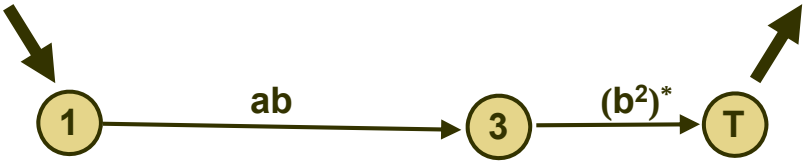
Eliminujeme smyčku 4.

Eliminujeme uzel 4.

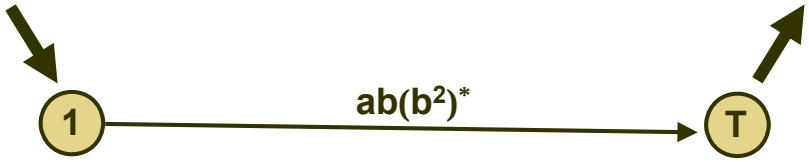
Eliminujeme uzel 2.



Eliminujeme smyčku 3.



Eliminujeme uzel 3.



Automaty s výstupem (motivace)

… aneb jak zaznamenat výpočet automatu?

Dosud jediná zpráva z automatu - jsme v přijímajícím stavu.
Můžeme z konečného automatu získat více informací?
Můžeme zaznamenat trasu výpočtu?

1) indikace stavů (všech, nejen koncových)
   v každé chvíli víme, kde se automat nachází
   Příklad: různé (regulární) čítače

2) indikace přechodů
   po přečtení každého symbolu víme, co automat udělal
   Příklad: (regulární) překlad slov

Automat už není tak docela černá skříňka.

Mooreův stroj

Mooreovým (sekvenčním) strojem nazýváme šestici
$A = (Q,X,Y,\delta,\mu,q_0)$ resp. pětici $A = (Q,X,Y,\delta,\mu)$, kde:

- Q - konečná neprázdná množina stavů (*stavový prostor*)
- X - konečná neprázdná množina symbolů (*vstupní abeceda*)
- Y - konečná neprázdná množina symbolů (*výstupní abeceda*)
- δ - zobrazení $Q \times X \to Q$ (*přechodová funkce*)
- μ - zobrazení $Q \to Y$ (*značkovací funkce*)
- $q_0 \in Q$ (*počáteční stav*)

Poznámky:

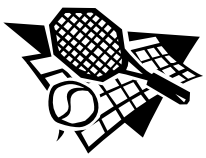
- někdy nás nezajímá počáteční stav, ale jen práce automatu
- značkovací funkce umožňuje suplovat roli koncových stavů

  $F \subseteq Q$ nahradíme značkovací funkcí $\mu : Q \to \{0,1\}$ takto:

  $\mu(q) = 0$, pokud $q \notin F$

  $= 1$, pokud $q \in F$

Příklad Mooreova stroje



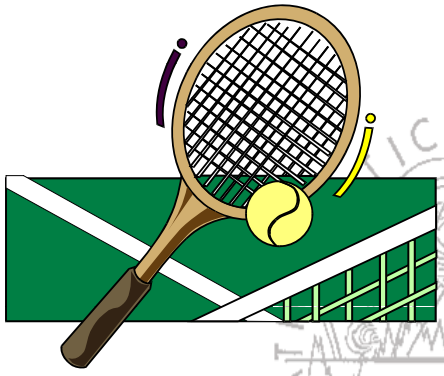
Navrhněte automat počítající tenisové skóre.

Vstupní abeceda: ID hráče, který uhrál bod

Výstupní abeceda/stavy: skóre (tj. $Q=Y$ a $\mu(q)=q$)

Stav/výstup	*A*	*B*
00:00	15:00	00:15
15:00	30:00	15:15
15:15	30:15	15:30
00:15	15:15	00:30
30:00	40:00	30:15
30:15	40:15	30:30
30:30	40:30	30:40
15:30	30:30	15:40
00:30	15:30	00:40
40:00	A	40:15
40:15	A	40:30
40:30	A	shoda
30:40	shoda	B
15:40	30:40	B
00:40	15:40	B

Stav/výstup	*A*	*B*
shoda	A:40	40:A
A:40	A	shoda
40:A	shoda	B
A	15:00	00:15
B	15:00	00:15



Mealyho stroj

Mealyho (sekvenčním) strojem nazýváme šestici
$A = (Q,X,Y,\delta,\lambda,q_0)$ resp. pětici $A = (Q,X,Y,\delta,\lambda)$, kde:

- Q - konečná neprázdná množina stavů (*stavový prostor*)
- X - konečná neprázdná množina symbolů (*vstupní abeceda*)
- Y - konečná neprázdná množina symbolů (výstupní abeceda)
- δ - zobrazení $Q \times X \to Q$ (*přechodová funkce*)
- λ - zobrazení $Q \times X \to Y$ (*výstupní funkce*)
- $q_0 \in Q$ (*počáteční stav*)

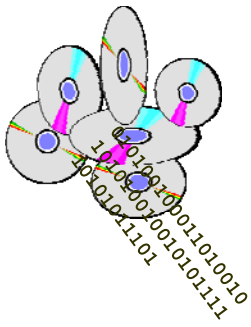
Poznámka:

výstup je určen stavem a vstupním symbolem

tj. Mealyho stroj je obecnějším prostředkem než stroj Mooreův

značkovací funkci $\mu : Q \to Y$ lze nahradit výstupní funkcí $\lambda: Q \times X \to Y$
například takto:
$\forall x \in X \ \lambda(q,x) = \mu(q)$
nebo takto $\forall x \in X \ \lambda(q,x) = \mu(\delta(q,x))$

Příklad Mealyho stroje

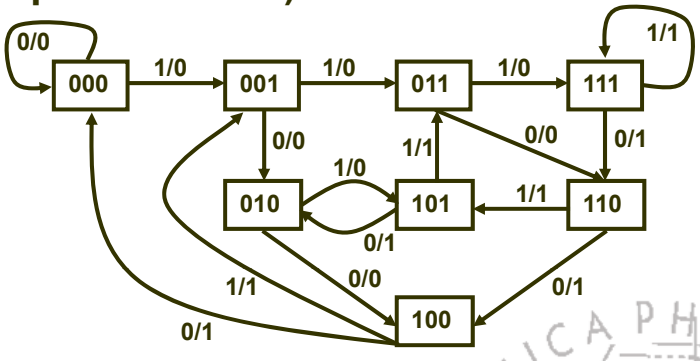


Navrhněte automat, který dělí vstupní slovo v binárním tvaru číslem 8 (celočíselně).

Realizace:

- posun o tři bity doprava (1101010 → 0001101)
- potřebujeme si pamatovat poslední trojici bitů (vlastně dynamická tříbitová paměť'-buffer)

Stav\symbol	0	1
000	000/0	001/0
001	010/0	011/0
010	100/0	101/0
011	110/0	111/0
100	000/1	001/1
101	010/1	011/1
110	100/1	101/1
111	110/1	111/1



Vadí nám, když nevíme, kde automat startuje?
NE - po třech symbolech začne počítat správně

Výstup sekvenčních strojů

slovo ve vstupní abecedě → slovo ve výstupní abecedě

Mooreův stroj

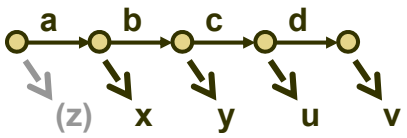
značkovací funkce $\mu: Q \to Y$

$\mu^* : Q \times X^* \to Y^*$

$\mu^*(q,\lambda) = \lambda$ (někdy $\mu^*(q,\lambda) = \mu(q)$)

$\mu^*(q,wx) = \mu^*(q,w) \,.\, \mu(\delta^*(q,wx))$

Příklad: $\mu^*(00{:}00,AABA) = (00{:}00\ .)\ 15{:}00\ .\ 30{:}00\ .\ 30{:}15\ .\ 40{:}15$



Mealyho stroj

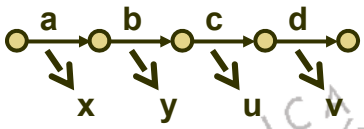
výstupní funkce $\lambda: Q \times X \to Y$

$\lambda^* : Q \times X^* \to Y^*$

$\lambda^*(q,\lambda) = \lambda$

$\lambda^*(q,wx) = \lambda^*(q,w) \,.\, \lambda(\delta^*(q,w),x)$

Příklad: $\mu^*(,000',1101010) = 0001101$



Převod Mooreova stroje na Mealyho

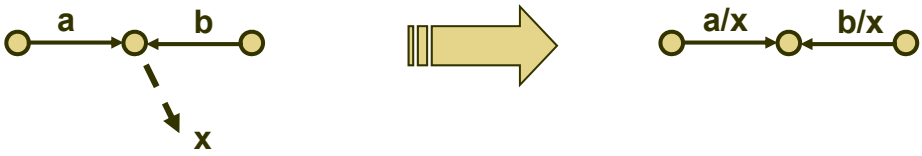
Nechť $A = (Q,X,Y,\delta,\mu,q_0)$ je Mooreův stroj.

Umíme najít Mealyho stroj B tak, že $\forall q,w\ \mu^*(q,w) = \lambda^*(q,w)$?

ANO!

položme $B = (Q,X,Y,\delta,\lambda,q_0)$, kde $\lambda(q,x) = \mu(\delta(q,x))$

tj. λ vrací značku stavu, do kterého přejdeme



Příklad:

stav	0	1	výstup
a	a	b	0
b	b	c	1
c	c	a	2

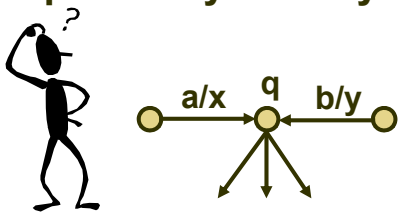
stav	0	1
a	a/0	b/1
b	b/1	c/2
c	c/2	a/0

Převod Mealyho stroje na Mooreův

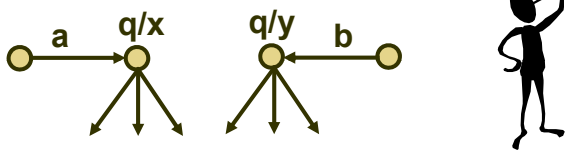
Nechť $A = (Q,X,Y,\delta,\lambda,q_0)$, je Mealyho stroj.

Sestrojme Mooreův stroj B tak, že $\forall q,w\ \lambda^*(q,w) = \mu^*(q,w)$.

Problém: do jednoho stavu mohou vést přechody s různým výstupem!



Řešení: stav rozdělíme na více stavů (podle počtu výstupních symbolů).

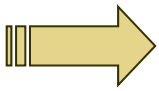


Teď už je to jednoduché! $B = (Q\times Y,X,Y,\delta',\mu,(q_0,\_))$, kde

$\delta'((q,y),x) = (\delta(q,x), \lambda(q,x))$ a $\mu((q,y)) = y$

Příklad:

stav	0	1
a	a/0	b/0
b	a/1	b/1



stav	0	1	výstup
(a,0)	(a,0)	(b,0)	0
(a,1)	(a,0)	(b,0)	1
(b,0)	(a,1)	(b,1)	0
(b,1)	(a,1)	(b,1)	1

Konečné automaty - shrnutí

Konečný automat

- jednoznačný redukovaný automat
- nedeterminismus (2^n), dvousměrný KA (n^n)

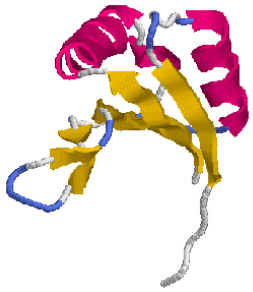
Automaty a jazyky

- regulární jazyky
- uzavřenost na množinové operace
- uzavřenost na řetězcové operace
- uzavřenost substituce

Charakteristika regulárních jazyků

- Nerodova věta (kongruence)
- Kleeneova věta (elementární jazyky a operace)
- Iterační lemma (iterace podslov, jen nutná podmínka)

Celulární automaty aneb hra na život

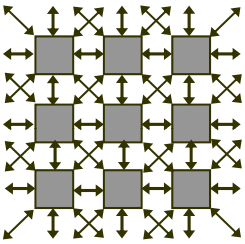


Buňka = konečný automat

vstup automatu = stavy okolních buněk

je definováno uniformní propojení automatů

automaty pracují synchronně



Conwayova hra „Life“

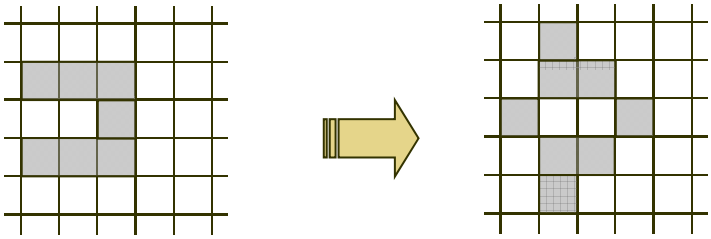
stav 1 (živá buňka), stav 0 (mrtvá buňka)

přechody (dle počtu živých buněk v okolí):

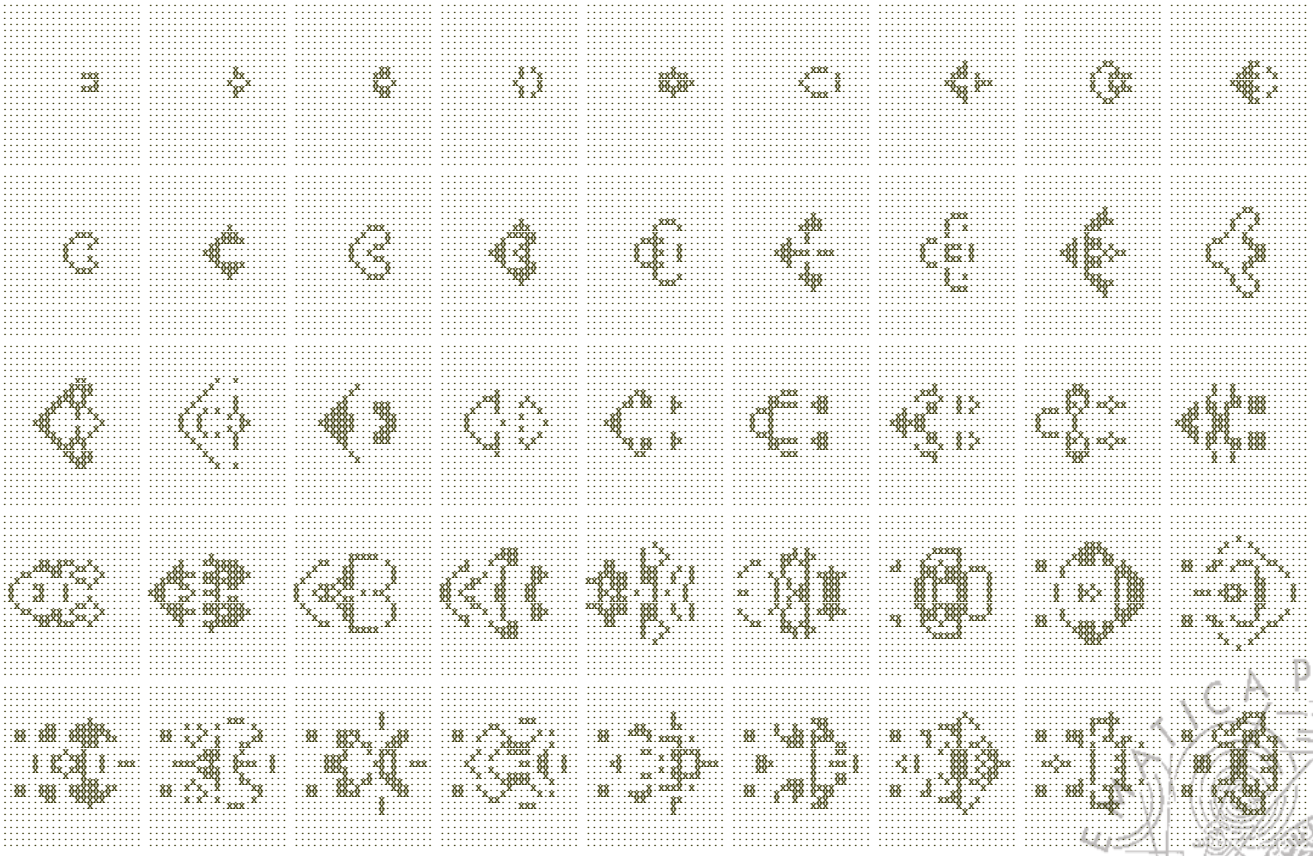
zrození: 3-4 živé buňky v okolí

úmrtí: 0-1 živá buňka v okolí („je mi smutno“)

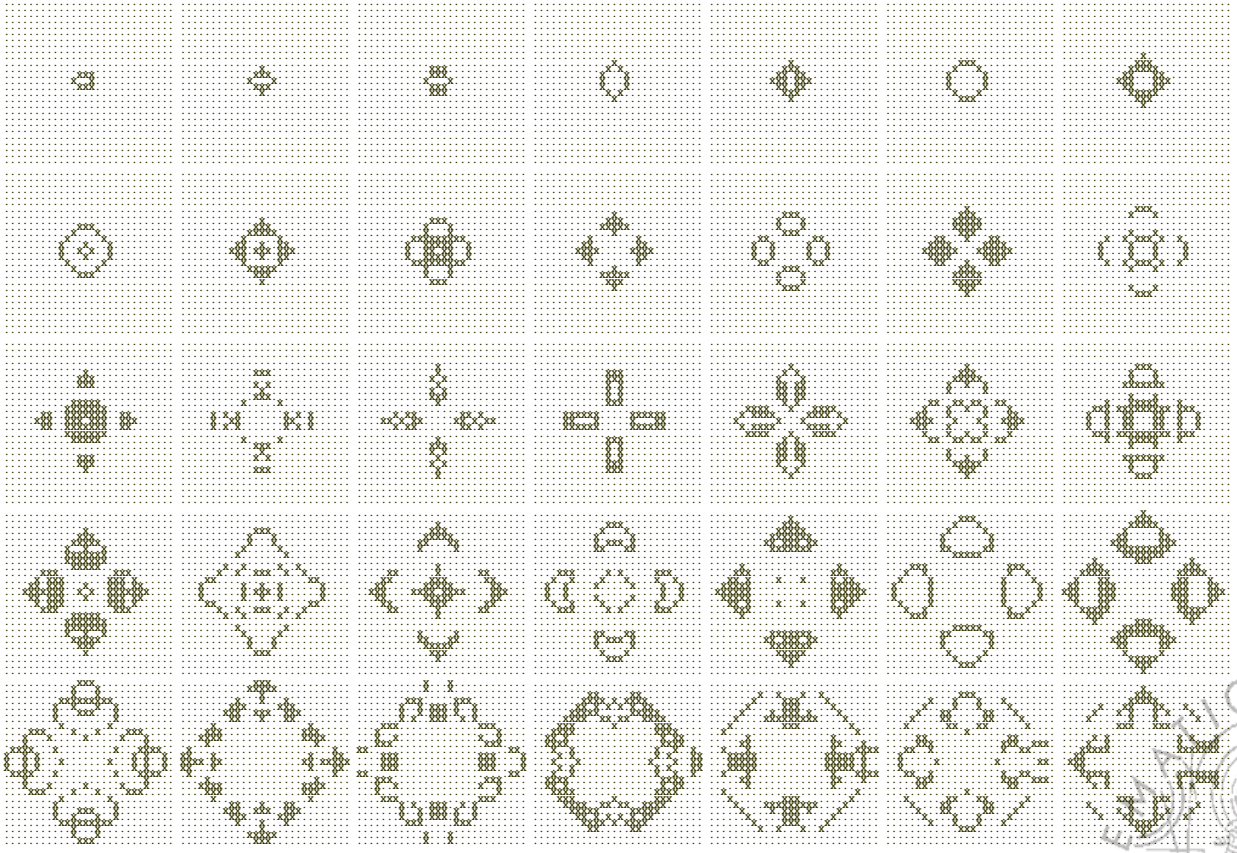
4-8 živých buněk v okolí („je mi těsno“)



Život („Life“)



Jedna buňka navíc



Úvod do formálních gramatik

Gramatiky, všichni je známe, ale co to je?

Popis jazyka pomocí pravidel, podle kterých se vytvářejí všechny řetězce daného jazyka.

Původně pro popis přirozených jazyků

<věta> → <podmětná část> <přísudková část>

Zadání syntaxe vyšších programovacích jazyků

od dob Algolu 60

Backus-Naurova normální forma (BNF)

<číslo> :== <číslo bez zn.> | +<číslo bez zn.> |-<číslo bez zn.>

<číslo bez zn.> :== <číslice> | <číslice><číslo bez znam>.

<číslice> :== 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Příklady gramatik

1) Gramatika správných uzávorkování

$V \rightarrow VV \mid (V) \mid ()$

Výraz (()())(()) je generován posloupností přepisů:

$V \rightarrow VV \rightarrow (V)V \rightarrow (VV)V \rightarrow (()V)V \rightarrow (()())V \rightarrow (()())(V) \rightarrow (()())(())$

2) Gramatika generující všechny výrazy s operacemi + a *, závorkami a jedinou konstantou c.

$V \rightarrow T+V \mid T$

$T \rightarrow F*T \mid F$

$F \rightarrow (V) \mid c$

Výraz c+c*c+c je generován posloupností přepisů:

$V \rightarrow T+V \rightarrow F+V \rightarrow c+V \rightarrow c+T+V \rightarrow c+F*T+V \rightarrow c+c*T+V \rightarrow$
$c+c*F+V \rightarrow c+c*c+V \rightarrow c+c*c+T \rightarrow c+c*c+F \rightarrow c+c*c+c$

Přepisovací systémy - základní pojmy

Přepisovacím (produkčním) systémem nazýváme dvojici

$R = (V,P)$,

kde

V - konečná abeceda

P - konečná množina přepisovacích pravidel

přepisovací pravidlo (produkce) je uspořádaná dvojice (u,v), kde $u,v \in V^*$ (zpravidla píšeme $u \rightarrow v$)

Říkáme, že *w se přímo přepíše na z* (píšeme $w \Rightarrow z$), jestliže:

$\exists u,v,x,y \in V^*$ tž. $w=xuy$, $z=xvy$ a $(u \rightarrow v) \in P$.

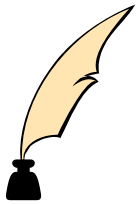
Říkáme, že *w se přepíše na z* (píšeme $w \Rightarrow^* z$), jestliže:

$\exists u_1,\ldots,u_n \in V^* \quad w=u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_n=z$.

Posloupnost $u_1,\ldots,u_n$ nazýváme *odvozením* (derivací).

Pokud $\forall i \neq j \; u_i \neq u_j$, potom hovoříme o *minimálním odvození*.

Přepisovací systémy



Příklad:

$V = \{0,1\}$

$P = \{01 \to 10,\ 10 \to 01\}$

$00110 \Rightarrow^* 00011$ dostaneme z $00\underline{10} \Rightarrow 00\underline{01}1 \Rightarrow 00011$

$00110 \Rightarrow^* 01100$ dostaneme z $0\underline{01}10 \Rightarrow 0\underline{10}10 \Rightarrow 01100$

libovolné slovo přepíše na libovolné jiné slovo
(se stejným počtem výskytů 0 a 1)

Produkční systémy slouží jako programovací nástroj v UI

program = systém produkcí

data = slova v abecedě

- OPS5, TOPS, CLIPS, JBoss Rules, Jess
- Constraint Handling Rules (CHR)
- Definite Clause Grammars (DCG)

Formální (generativní) gramatiky

Generativní gramatikou nazýváme čtveřici $G=(V_N,V_T,S,P)$:

V_N - konečná množina neterminálních symbolů

V_T - konečná množina terminálních symbolů

obě abecedy jsou neprázdné a disjunktní!

$S \in V_N$ - počáteční neterminální symbol

P - systém produkcí $u \to v$, kde $u,v \in (V_N \cup V_T)^*$
a u obsahuje alespoň jeden neterminální symbol.

Jazyk L(G) generovaný gramatikou G definujeme takto:
$L(G) = \{w \mid w \in V_T^* \wedge S \Rightarrow^* w\}$.

Gramatiky G_1 a G_2 jsou *ekvivalentní*, jestliže $L(G_1)=L(G_2)$.

Příklad:

$G = (\{S\},\{0,1\},S,\{S \to 0S1, S \to 01\})$, $L(G) = \{0^i1^i \mid i \geq 1\}$

Chomského hierarchie

Klasifikace gramatik podle tvaru přepisovacích pravidel.

gramatiky typu 0 (rekurzivně spočetné jazyky $\mathcal{L}_0$)

pravidla v obecné formě

gramatiky typu 1 (kontextové jazyky $\mathcal{L}_1$)

pouze pravidla ve tvaru $\alpha X\beta \to \alpha w\beta$,
$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$

jedinou výjimkou je pravidlo $S \to \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové jazyky $\mathcal{L}_2$)

pouze pravidla ve tvaru $X \to w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

gramatiky typu 3 (regulární/pravé lineární jazyky $\mathcal{L}_3$)

pouze pravidla ve tvaru $X \to wY$, $X \to w$, $X,Y \in V_N$, $w \in V_T^*$

Uspořádanost Chomského hierarchie

Chomského hierarchie definuje uspořádání tříd jazyků:

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

Dokonce vlastní podmnožiny (později):

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

$\mathcal{L}_0 \supseteq \mathcal{L}_1$ (rekurzivně spočetné jazyky zahrnují kontextové jazyky)

obecná pravidla $\supseteq$ pravidla tvaru $\alpha X\beta \to \alpha w\beta$

$\mathcal{L}_2 \supseteq \mathcal{L}_3$ (bezkontextové jazyky zahrnují regulární jazyky)

„$X \to w$, $w \in (V_N \cup V_T)^*$ “ $\supseteq$ „$X \to wY$, $X \to w$, $Y \in V_N$, $w \in V_T^*$ “

$\mathcal{L}_1 \supseteq \mathcal{L}_2$ (kontextové jazyky zahrnují bezkontextové jazyky)

$\alpha X\beta \to \alpha w\beta$, $|w|>0$ vs. $X \to w$, $|w| \geq 0$

problém s pravidly tvaru $X \to \lambda$

Můžeme z bezkontextových gramatik vyřadit pravidla $X \to \lambda$?

Nevypouštějící bezkontextové gramatiky

Bezkontextová gramatika G je *nevypouštějící* právě tehdy, když nemá pravidla ve tvaru $X \to \lambda$.

Věta: Ke každé bezkontextové gramatice G existuje nevypouštějící bezkontextová gramatika G_1 taková, že $L(G_1) = L(G) - \{\lambda\}$ (jazyky se liší maximálně o prázdné slovo).

Je-li $\lambda \in L(G)$, potom existuje BKG G_2 tak, že $L(G_2) = L(G)$ a jediné pravidlo s λ na pravé straně je $S' \to \lambda$ a S' (počáteční neterminál) se nevyskytuje na pravé straně žádného pravidla G_2 (tedy $\mathcal{L}_1 \supseteq \mathcal{L}_2$).

Příklad:

G: $S \to 0S1 \mid \lambda$

G_1: $S \to 0S1 \mid 01$

G_2: $S' \to S \mid \lambda$, $S \to 0S1 \mid 01$

Převod na nevypouštějící BKG

… aneb, jak se zbavit pravidel ve tvaru $X \to \lambda$?

Základní myšlenka:

- pravidlo $X \to \lambda$ se používá pro vyhození X ze slova
- co když X do slova vůbec nezařadíme?

…, $Y \to uXv$, $X \to \lambda$, … $\Rightarrow$ …, $Y \to uXv$, $Y \to uv$, …

1) Nejprve zjistíme neterminály, které se přepisují na λ:

$U = \{X \mid X \in V_N \wedge X \Rightarrow^* \lambda\}$

Proč tak silně (nestačilo by $X \to \lambda$ místo $X \Rightarrow^* \lambda$)?

Řešení derivací $X \Rightarrow^{X \to Y} Y \Rightarrow^{Y \to Z} Z \Rightarrow^{Z \to \lambda} \lambda$

Iterační algoritmus pro získání U:

$U_1 = \{X \mid X \in V_N \wedge (X \to \lambda) \in P\}$ přímý přepis

$U_{i+1} = \{X \mid X \in V_N \wedge (X \to w) \in P, w \in U_i^*\}$ přepis po i+1 krocích

$U_1 \subseteq U_2 \subseteq \ldots \subseteq V_N$ + stabilizace ($\exists k\ U_k = U_{k+1} = \ldots$) + $U = U_k$

Převod na nevypouštějící BKG - pokračování

2) Úprava pravidel

do P_1 dáme pravidla tvaru $X \to u$ taková, že:

- $u \neq \lambda$
- v P je pravidlo $X \to v_1 Y_1 v_2 \ldots v_m Y_m v_{m+1}$, $Y_i \in U$, $v_i \in ((V_N - U) \cup V_T)^*$ a u vzniká z $(v_1 Y_1 v_2 \ldots v_m Y_m v_{m+1})$ vypuštěním některých (všech, žádného) symbolů Y_i.

3) Ještě $L(G_1) = L(G) - \{\lambda\}$

zřejmé: G_1 je nevypouštějící BKG, $L(G_1) \subseteq L(G)$, $\lambda \notin L(G_1)$

nechť $w \in L(G)$ a $w \neq \lambda$, tj. $S \Rightarrow^* w$,

pokud se použilo pravidlo z $P\text{-}P_1$, pak má tvar $X \to \lambda$

v derivaci před ním muselo být užito pravidlo $Y \to uXv$

uděláme novou derivaci s $Y \to uv$ a bez $X \to \lambda$

4) Zbývá situace $\lambda \in L(G)$

$G_2 = (V_N \cup \{S'\}, V_T, S', P_1 \cup \{S' \to \lambda, S' \to S\})$

Příklad - nevypouštějící BKG

$S \to aSc \mid A$

$A \to bAc \mid \lambda$

1) Nejprve zjistíme neterminály, které se přepisují na λ:

$U = \{A, S\}$

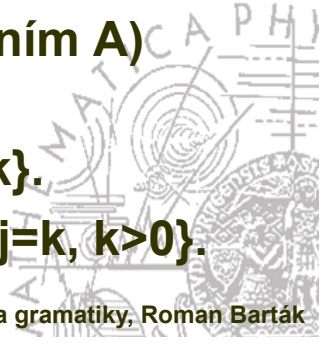
2) Upravíme pravidla:

$S \to aSc \mid A$

$S \to ac$ (vzniklo z $S \to aSc$ vypuštěním S)

$A \to bAc$ (pravidlo $A \to \lambda$ nepřevádíme)

$A \to bc$ (vzniklo z $A \to bAc$ vypuštěním A)



Původní gramatika generuje jazyk $\{a^i b^j c^k \mid i+j=k\}$.

Převedená gramatika generuje jazyk $\{a^i b^j c^k \mid i+j=k, k>0\}$.

Chomského hierarchie

gramatiky typu 0 (rekurzivně spočetné jazyky $\mathcal{L}_0$)

pravidla v obecné formě

gramatiky typu 1 (kontextové jazyky $\mathcal{L}_1$)

pouze pravidla ve tvaru $\alpha X\beta \to \alpha w\beta$,
$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$

jedinou výjimkou je pravidlo $S \to \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové jazyky $\mathcal{L}_2$)

pouze pravidla ve tvaru $X \to w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

gramatiky typu 3 (regulární/pravé lineární jazyky $\mathcal{L}_3$)

pouze pravidla ve tvaru $X \to wY$, $X \to w$, $X,Y \in V_N$, $w \in V_T^*$

Gramatiky typu 3 a regulární jazyky

pouze pravidla ve tvaru $X \to wY$, $X \to w$, $X,Y \in V_N$, $w \in V_T^*$

Podívejme se na derivace generované gramatikami typu 3

P: $S \to 0S \mid 1A \mid \lambda$, $A \to 0A \mid 1B$, $B \to 0B \mid 1S$

$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$

Pozorování:

- každé slovo derivace obsahuje právě jeden neterminál
- tento neterminál je vždy umístěn zcela vpravo
- aplikací pravidla $X \to w$ se derivace uzavírá
- krok derivace = generuje symbol(y) +změní neterminál

Idea vztahu gramatiky a konečného automatu:

neterminál = stav konečného automatu

pravidla = přechodová funkce

Převod konečného automatu na gramatiku

$L \in \mathcal{F} \Rightarrow L \in \mathcal{L}_3$

Důkaz:

L=L(A) pro nějaký konečný automat $A=(Q,X,\delta,q_0,F)$

definujme gramatiku $G=(Q,X,q_0,P)$, kde pravidla mají tvar

$p \to aq$, když $\delta(p,a)=q$

$p \to \lambda$, když $p \in F$

ještě L(A)=L(G)?

1) $\lambda \in L(A) \Leftrightarrow q_0 \in F \Leftrightarrow (q_0 \to \lambda) \in P \Leftrightarrow \lambda \in L(G)$

2) $a_1 \ldots a_n \in L(A) \Leftrightarrow \exists q_0,\ldots,q_n \in Q$ tž. $\delta(q_i,a_{i+1})=q_{i+1}$, $q_n \in F$

$\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow \ldots \Rightarrow a_1 \ldots a_n q_n \Rightarrow a_1 \ldots a_n)$ je derivace pro $a_1 \ldots a_n$

$\Leftrightarrow a_1 \ldots a_n \in L(G)$

QED

A co naopak?

- pravidla $X \to aY$ kódujeme do přechodové funkce a $X \to \lambda$ je konec
- ale co pravidla $X \to a_1 \ldots a_n Y$, $X \to Y$, $X \to a_1 \ldots a_n$?

Příklad převodu KA na gramatiku

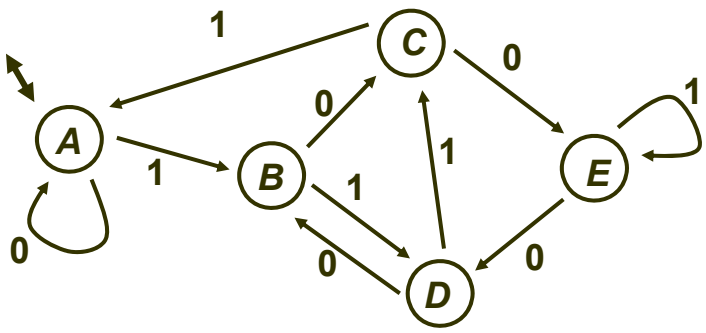
$A \to 1B \mid 0A \mid \lambda$

$B \to 0C \mid 1D$

$C \to 0E \mid 1A$

$D \to 0B \mid 1C$

$E \to 0D \mid 1E$



Příklady derivací:

$A \Rightarrow 0A \Rightarrow 0$ (0)

$A \Rightarrow 1B \Rightarrow 10C \Rightarrow 101A \Rightarrow 101$ (5)

$A \Rightarrow 1B \Rightarrow 10C \Rightarrow 101A \Rightarrow 1010A \Rightarrow 1010$ (10)

$A \Rightarrow 1B \Rightarrow 11D \Rightarrow 111C \Rightarrow 1111A \Rightarrow 1111$ (15)

$L = \{ w \mid w \in \{0,1\}^* \wedge w$ je binární zápis čísla dělitelného 5$\}$

Standardizace pravidel regulární gramatiky

Ke každé gramatice $G=(V_N,V_T,S,P)$ typu 3 existuje ekvivalentní gramatika G‘, která obsahuje pouze pravidla ve tvaru: $X\rightarrow aY$ a $X\rightarrow\lambda$.

Důkaz:

definujme $G‘=(V‘_N,V_T,S,P‘)$, kde pravidla P‘ získáme takto

P	P‘
$X\rightarrow aY$	$X\rightarrow aY$
$X\rightarrow\lambda$	$X\rightarrow\lambda$
$X\rightarrow a_1\ldots a_nY$	$X\rightarrow a_1Y_2,\ Y_2\rightarrow a_2Y_3,\ \ldots,\ Y_n\rightarrow a_nY$
$Z\rightarrow a_1\ldots a_n$	$Z\rightarrow a_1Z_1,\ Z_1\rightarrow a_2Z_2,\ \ldots,\ Z_n\rightarrow\lambda$

($Y_2,\ldots,Y_n$, $Z_1,\ldots,Z_n$ jsou nové neterminály - pro každé pravidlo jiná sada)

zbývá $X\rightarrow Y$

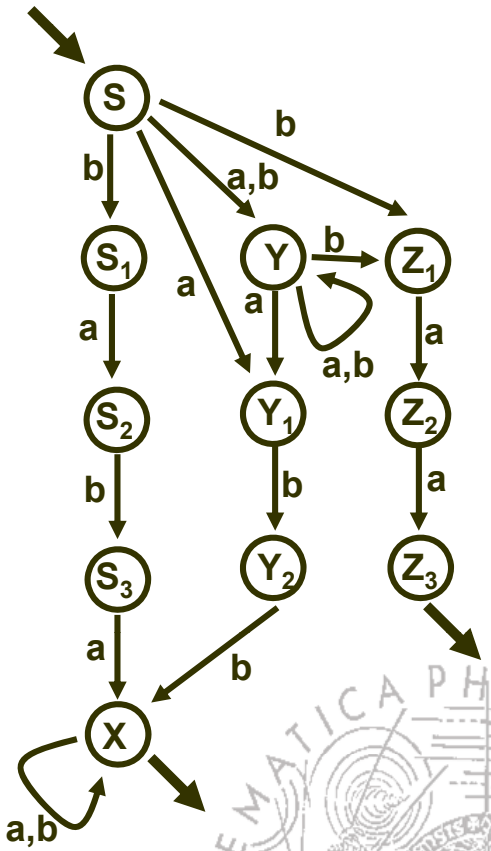
definujme $U(X)=\{Y \mid Y\in V_N\wedge X\Rightarrow^* Y\}$

efektivní postup $U_1=\{Y \mid (X\rightarrow Y)\in P\}$, $U_{i+1}=U_i\cup\{Y \mid (Z\rightarrow Y)\in P, Z\in U_i\}$

$X\rightarrow Y$	$X\rightarrow w$ pro všechna $Z\rightarrow w$ z P‘ a $Z\in U(X)$

Příklad standardizace regulární gramatiky

Originální	*Převedená*
$S\rightarrow babaX$	$S\rightarrow bS_1$ $S_1\rightarrow aS_2$ $S_2\rightarrow bS_3$ $S_3\rightarrow aX$
$S\rightarrow Y$	$S\rightarrow aY \mid bY \mid aY_1 \mid bZ_1$
$Y\rightarrow aY \mid bY$	$Y\rightarrow aY \mid bY$
$Y\rightarrow abbX$	$Y\rightarrow aY_1$ $Y_1\rightarrow bY_2$ $Y_2\rightarrow bX$
$Y\rightarrow baa$	$Y\rightarrow bZ_1$ $Z_1\rightarrow aZ_2$ $Z_2\rightarrow aZ_3$ $Z_3\rightarrow\lambda$
$X\rightarrow aX \mid bX \mid \lambda$	$X\rightarrow aX \mid bX \mid \lambda$



$L=\{ w \mid w=babau \vee w=uabbv \vee w=ubaa,\ u,v\in\{a,b\}^* \}$

Převod gramatiky na konečný automat

$L \in \mathcal{L}_3 \Rightarrow L \in \mathcal{F}$

Důkaz:

$L=L(G)$ pro nějakou gramatiku $G = (V_N,V_T,S,P)$ typu 3 obsahující pouze pravidla ve tvaru: $X \to aY$ a $X \to \lambda$

definujme nedeterministický konečný automat

$A = (V_N,V_T,\delta,\{S\},F)$, kde:

$F = \{ X \mid (X \to \lambda) \in P\}$

$\delta(X,a) = \{ Y \mid (X \to aY) \in P\}$

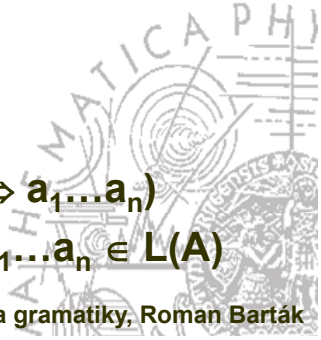
ještě $L(G)=L(A)$?

1) $\lambda \in L(G) \Leftrightarrow (S \to \lambda) \in P \Leftrightarrow S \in F \Leftrightarrow \lambda \in L(A)$

2) $a_1 \ldots a_n \in L(G)$

$\Leftrightarrow$ existuje derivace $(S \Rightarrow a_1X_1 \Rightarrow \ldots \Rightarrow a_1 \ldots a_nX_n \Rightarrow a_1 \ldots a_n)$

$\Leftrightarrow \exists X_0,\ldots,X_n \in V_N$ tž. $\delta(X_i,a_{i+1}) \ni X_{i+1}$, $X_0=S$, $X_n \in F \Leftrightarrow a_1 \ldots a_n \in L(A)$



Levé (a pravé) lineární gramatiky

Gramatiky typu 3 nazýváme také *pravé lineární* (neterminál je vždy vpravo).

Obdobně - gramatika G je *levá lineární*, jestliže má pouze pravidla tvaru $X \to Yw$, $X \to w$, $X,Y \in V_N$, $w \in V_T^*$ (neterminál je vždy vlevo).

Věta: Jazyky generované levou lineární gramatikou jsou právě regulární jazyky.

Důkaz:

- „otočením“ pravidel dostaneme pravou lineární gramatiku $X \to Yw$, $X \to w$ převedeme na $X \to w^RY$, $X \to w^R$
- získaná gramatika generuje jazyk L^R
- víme, že regulární jazyky jsou uzavřené na reverzi tudíž protože L^R je regulární, je i L $(=(L^R)^R)$ regulární
- takto lze získat všechny regulární jazyky

Lineární gramatiky (a jazyky)

Můžeme levě a právě lineární pravidla používat najednou?

Další zobecnění - *gramatika je lineární*, jestliže má pouze pravidla tvaru $X \to uYv$, $X \to w$, $X,Y \in V_N$, $u,v,w \in V_T^*$ (na pravé straně vždy maximálně jeden neterminál).

Lineární jazyky jsou právě jazyky generované lineárními gramatikami.

Zřejmě: regulární jazyky $\subseteq$ lineární jazyky

Platí také: regulární jazyky $\supseteq$ lineární jazyky?

NE!

$\{0^n1^n \mid n \geq 1\}$ není regulární jazyk, ale je lineární ($S \to 0S1 \mid 01$)

Pozorování:

lineární pravidla lze rozložit na levě a pravě lineární pravidla

$S \to 0A$, $A \to S1$

Chomského hierarchie

gramatiky typu 0 (rekurzivně spočetné jazyky $\mathcal{L}_0$)

pravidla v obecné formě

gramatiky typu 1 (kontextové jazyky $\mathcal{L}_1$)

pouze pravidla ve tvaru $\alpha X \beta \to \alpha w \beta$,
$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$

jedinou výjimkou je pravidlo $S \to \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové jazyky $\mathcal{L}_2$)

pouze pravidla ve tvaru $X \to w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

gramatiky typu 3 (regulární/pravě lineární jazyky $\mathcal{L}_3$)

pouze pravidla ve tvaru $X \to wY$, $X \to w$, $X,Y \in V_N$, $w \in V_T^*$

Bezkontextové gramatiky

pouze pravidla ve tvaru $X \to w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

velký praktický význam

- definování syntaxe vyšších programovacích jazyků
- konstrukce kompilátorů

Příklad:

```
<Program> → <Příkaz> | <Příkaz> <Program>
<Příkaz> → <IF-příkaz> | <WHILE-příkaz> | <Přiřazení>
<IF-příkaz> → if <Test> then <Příkaz> else <Příkaz>
<WHILE-příkaz> → while <Test> do <Příkaz>
<Přiřazení> → <Proměnná> := <Výraz>
```

Bude nás zajímat:

- jednoznačnost gramatiky (kvůli překladu)
- analýza pomocí zásobníkových automatů
- vlastnosti BKG obecně

Redukované bezkontextové gramatiky

Redukce (gramatiky) = vyřazení zbytečností

u konečných automatů:

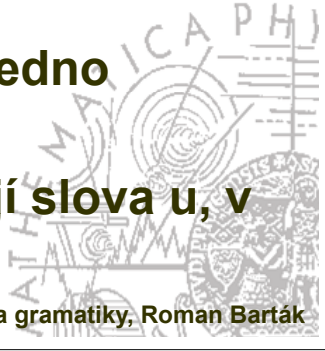
- dosažitelné stavy, které nejsou ekvivalentní
- redukt určen jednoznačně

u bezkontextových gramatik:

- dosažitelné neterminály, které něco generují
- zde slabší význam (nemáme jednoznačnost)

Bezkontextová gramatika G (taková, že $L(G) \neq \emptyset$) se nazývá *redukovaná*, jestliže:

1) pro každý neterminál X existuje alespoň jedno terminální slovo w takové, že $X \Rightarrow^* w$
2) pro každý neterminál X různý od S existují slova u, v tak, že $S \Rightarrow^* uXv$ (dosažitelnost).



Redukce bezkontextových gramatik

Příklad:

$S \to aA \mid ab$
$A \to BC$
$B \to ba$
$D \to ab \mid \lambda$

Zjevně stačí pravidlo
$S \to ab$,
ostatní pravidla (neterminály) jsou zbytečná

Tvrzení: Ke každé bezkontextové gramatice G takové, že $L(G) \neq \emptyset$ lze sestrojit ekvivalentní redukovanou gramatiku.

Důkaz (idea):

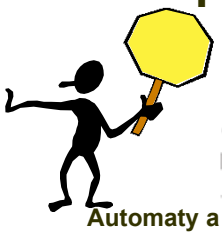
1) vyhoď neterminály, které negenerují terminální slovo

2) vyhoď nedosažitelné neterminály

Poznámka: pořadí redukčních kroků nelze prohodit!

$S \to ab \mid A$
$A \to BC$
$B \to b$

~~$S \to ab$~~
~~$B \to b$~~



Algoritmus redukce - krok 1

hledáme $V = \{X \mid X \in V_N, \exists w \in V_T^* \; X \Rightarrow^* w\}$

obvyklý postup (iterace po krocích):

$V_0 = V_T$

$V_{i+1} = V_i \cup \{X \mid X \in V_N, \exists w \in V_i^* \; (X \to w) \in P\}$

$V_0 \subseteq V_1 \subseteq \ldots \subseteq V_T \cup V_N$ + stabilizace ($\exists k \; V_k = V_{k+1} = \ldots$) + $V = V_k \cap V_N$

Zároveň víme, zda $L(G) \neq \emptyset$ ($L(G) \neq \emptyset \Leftrightarrow S \in V$)

Nyní z gramatiky odstraníme všechna pravidla obsahující na levé či pravé straně neterminál nepatřící do V.

a) máme splněn bod 1) definice redukované gramatiky

pro $X \in V$ víme: $\exists w \in V_T^* \; X \Rightarrow^* w$ + použitá pravidla nebyla odstraněna

b) získaná gramatika G‘ je ekvivalentní s původní gramatikou G

$L(G') \subseteq L(G)$ zřejmé , $L(G') \supseteq L(G)$ lze ukázat sporem

Příklad:

$S \to aA \mid ab$, $A \to BC$, $B \to ba$, $D \to ab \mid \lambda$

$V = \{S,B,D\}$

redukovaná pravidla: $S \to ab$, $B \to ba$, $D \to ab \mid \lambda$

Algoritmus redukce - krok 2 (dosažitelnost)

hledáme $U = \{X \mid X \in V_N, S \Rightarrow^* uXv\}$

obvyklý postup (iterace po krocích):

$U_0 = \{S\}$

$U_{i+1} = U_i \cup \{X \mid X \in V_N, \exists Y \in U_i\ (Y \to uXv) \in P\}$

$U_0 \subseteq U_1 \subseteq \ldots \subseteq V_N$ + stabilizace ($\exists k\ U_k = U_{k+1} = \ldots$) + $U = U_k$

Podobně jako v kroku 1 odstraníme z gramatiky všechna pravidla obsahující na levé či pravé straně neterminál nepatřící do U.

a) máme splněn bod 2) definice redukované gramatiky

pro $X \in U$ víme: $S \Rightarrow^* uXv$ + použitá pravidla nebyla odstraněna

b) získaná gramatika G'' je ekvivalentní s gramatikou G'

$L(G'') \subseteq L(G')$ zřejmé , $L(G'') \supseteq L(G')$ lze ukázat sporem

c) platnost bodu 1) definice redukované gramatiky nebyla narušena

spojením $X \Rightarrow^* w$ a $S \Rightarrow^* uXv$

Příklad:

$S \to ab,\quad B \to ba,\quad D \to ab \mid \lambda$ $\qquad U = \{S\}$

finální redukovaná pravidla: $S \to ab$

Bezkontextové gramatiky a derivace

pouze pravidla ve tvaru $X \to w$, $\quad X \in V_N, w \in (V_N \cup V_T)^*$

Úmluva

neterminály	= velká písmena
terminály	= malá písmena
pravidla	= $X \to u \mid v \mid w \mid \ldots$ (pro stejnou levou stranu)

Derivace

$w \Rightarrow z$, jestliže: $\exists x,y,v \in (V_N \cup V_T)^*$ tž. $w = xAy$, $z = xvy$ a $(A \to v) \in P$

G: $S \to aSX \mid \lambda, \qquad X \to XbSb \mid c$

$S \Rightarrow aSX \Rightarrow aSXbSb \Rightarrow aSXbb \Rightarrow aXbb \Rightarrow acbb$

$S \Rightarrow aSX \Rightarrow \quad aX \quad \Rightarrow aXbSb \Rightarrow acbSb \Rightarrow acbb$

$S \Rightarrow aSX \Rightarrow aSXbSb \Rightarrow aSXbb \Rightarrow aScbb \Rightarrow acbb$

Pozorování

- stejná délka derivací (počet pravidel) + použita stejná pravidla
- liší se pořadím aplikace pravidel
- přepis neterminálu neovlivňuje derivaci ve zbytku slova

Kanonické derivace

Zdá se zbytečné zabývat se derivacemi s různým pořadím pravidel.

Definice:

Levé přepsání $w \Rightarrow z$, jestliže se přepisuje nejlevější neterminál:
$\exists v,y \in (V_N \cup V_T)^* \; \exists x \in V_T^* \; \exists A \in V_N$ tž. $w=xAy$, $z=xvy$ a $(A \rightarrow v) \in P$

Levá derivace vzniká použitím pouze levých přepsání.

Pravé přepsání a *pravá derivace* se definuje obdobně (vždy se přepisuje nejpravější neterminál)

Lemma: Pro bezkontextové gramatiky platí:
$X \Rightarrow^* w$ právě tehdy, když existuje levá (pravá) derivace w z X

Důkaz:

stačí ukázat, že existence derivace implikuje existenci levé derivace

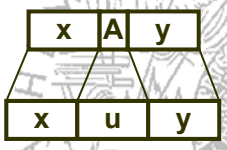
$xAy \Rightarrow xuy$, použitím $A \rightarrow u$

přepsání $A \rightarrow u$ neovlivní řetězce x a y ani jejich další přepisování

části x, A, y se přepisují nezávisle na sobě

můžeme preferovat aplikaci některých pravidel

formální důkaz (indukcí dle délky derivace)

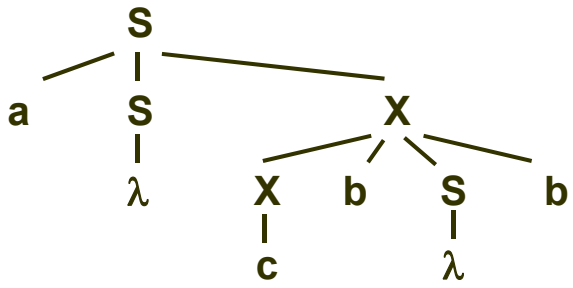


Derivační strom

Můžeme „výpočet“ zachytit jinak než sekvencí pravidel?

Příklad:

G: $S \rightarrow aSX \mid \lambda$, $X \rightarrow XbSb \mid c$



Definice:

Derivační strom je takový strom, že:

- každý vrchol je ohodnocen prvkem z $V_N \cup V_T \cup \{\lambda\}$
- kořen je ohodnocen S (počáteční neterminál)
- vnitřní vrcholy jsou ohodnoceny prvkem z V_N
- je-li A ohodnocení vrcholu a $u_1,\ldots,u_n$ jsou ohodnocení jeho potomků (bráno zleva doprava), potom $(A \rightarrow u_1,\ldots,u_n) \in P$
- je-li vrchol ohodnocen λ , potom je to list, který je jediným potomkem svého rodiče.

V derivačním stromu hraje roli jak stromové uspořádání dané hranami, tak uspořádání zleva doprava.

Derivace a derivační stromy

Říkáme, že *derivační strom dává slovo w*, jestliže *w* je slovo složené z ohodnocení listů (bráno zleva doprava).

Několik zřejmých tvrzení:

- $S \Rightarrow^* w$ potom existuje derivační strom, který dává w (jednoznačně daný derivací)
- máme-li derivační strom, který dává w, potom $S \Rightarrow^* w$ (derivace ale není určena jednoznačně)
- každý derivační strom jednoznačně určuje levou (a pravou) derivaci

Derivační strom zastupuje derivace slova získané „stejným způsobem“ (význam slova).

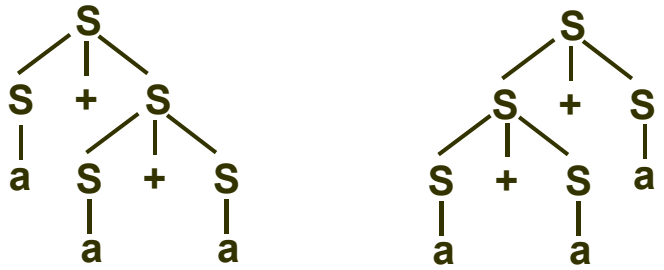
Je možné mít různé derivační stromy dávající stejné slovo (pro stejnou gramatiku)?

Jednoznačnost a víceznačnost BKG

Příklad:

$S \rightarrow S+S \mid a$

slovo a+a+a



Definice:

- *Bezkontextová gramatika G je víceznačná* (nejednoznačná), jestliže $\exists w \in L(G)$, které má dvě různé levé derivace.
- V ostatních případech *bezkontextová gramatika je jednoznačná*.
- *Bezkontextový jazyk L je jednoznačný*, jestliže existuje jednoznačná bezkontextová gramatika G tak, že L=L(G).
- *Bezkontextový jazyk L je (podstatně) nejednoznačný*, jestliže každá BKG G, taková že L=L(G) je nejednoznačná.

Příklad:

jazyk $\{a^i b^j c^k \mid i=j \vee j=k\}$ je podstatně nejednoznačný

pro slovo $a^i b^i c^i$ existují z principielních důvodů dva způsoby odvození

Od víceznačnosti k jednoznačnosti

Víceznačnost je potenciálním zdrojem potíží.

jedno slovo = více významů

U programovacích jazyků je víceznačnost nepřípustná!

Příklad 1:

S → S+S | a …víceznačná gramatika

S → a+S | a …ekvivalentní jednoznačná gramatika

Příklad 2:

S → if then S else S | if then S | λ

slovo „if then if then else“ má dva významy
„if then (if then else)“ nebo „if then (if then) else“

Řešení:

- **syntaktická chyba (Algol 60)**
- **else patří k bližšímu if (preference pořadí pravidel)**
- **závorky begin-end (asi nejčistší řešení)**

Jednoznačnost a kompilátory

E → E+E | E*E | (E) | a … nejednoznačné

E → E+E |T, T → T*T | F, F → (E) | a … řeší prioritu operací

Kompilace výrazu (zásobník na mezivýsledky+dva registry):

(1) E → E+T … pop r1; pop r2; add r1,r2; push r2

(2) E → T

(3) T → T*F … pop r1; pop r2; mul r1,r2; push r2

(4) T → F

(5) F → (E)

(6) F → a … push a

a+a*a, získáme postupnou aplikací pravidel 1,2,4,6,3,4,6,6

$\underline{E} \Rightarrow \underline{E}+T \Rightarrow \underline{T}+T \Rightarrow \underline{F}+T \Rightarrow a+\underline{T} \Rightarrow a+\underline{T}*F \Rightarrow a+\underline{F}*F \Rightarrow a+a*\underline{F} \Rightarrow a+a*a$

posloupnost obrátíme a vybere pouze pravidla generující kód
6,6,3,6,1

nyní pravidla nahradíme příslušným kódem

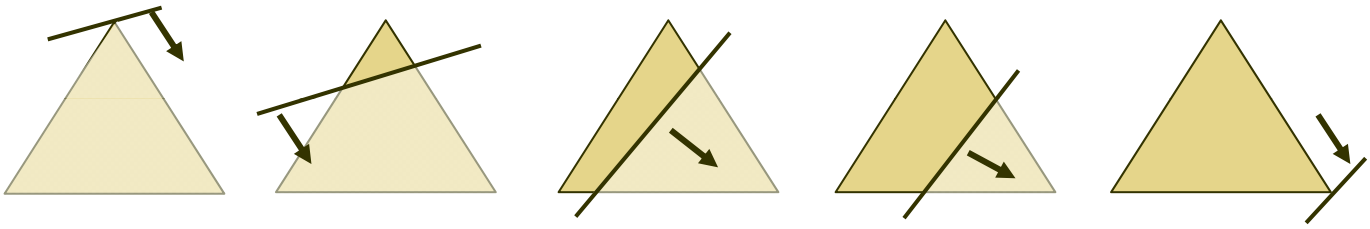
push a; push a; pop r1; pop r2; mul r1,r2; push r2; push a;
pop r1; pop r2; add r1,r2; push r2

Analýza shora bezkontextových jazyků

Jak k danému slovu a zvolené bezkontextové gramatice najdeme odpovídající derivační strom?

Analýza shora

- konstruujeme levou derivaci
- dosud vygenerované slovo kontrolujeme se vstupem



V každém kroku derivace můžeme slovo psát ve tvaru *uv*, kde

- *u* obsahuje pouze terminály (již přečtená část slova)
- *v* začíná neterminálem (zatím nehotová část)

Postup hledání derivace:

1) vezmi první neterminál *A* z *v* a nahraď ho *w*, dle pravidla $A \rightarrow w$
2) vzniklé slovo *v* rozlož na *xy*, kde *x* obsahuje pouze terminály a *y* začíná neterminálem
3) zkontroluj *x* oproti vstupu a pokud je v pořádku, přidej *x* za *u*, polož *v* rovno *y* a opakuj od 1 dokud $v \neq \lambda$

Realizace analyzátoru

slovo generujeme na zásobník (LIFO struktura)

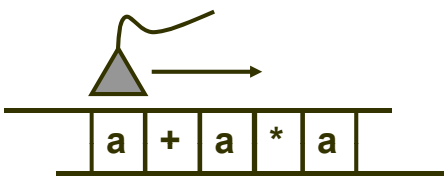
je-li vrchol zásobníku terminál, srovnáme ho se vstupem (čteme znak)

je-li vrchol zásobníku neterminál, nahradíme ho slovem dle pravidla

končíme, když je zásobník prázdný (musí být přečteno celé slovo)

Příklad:

(1) $E \rightarrow E+T$
(2) $E \rightarrow T$
(3) $T \rightarrow T*F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow a$



zásobník	zbytek vstupu	pravidlo
E	a+a*a	
E+T	a+a*a	(1)
T+T	a+a*a	(2)
F+T	a+a*a	(4)
a+T	a+a*a	(6)
+T	+a*a	krácení
T	a*a	krácení
T*F	a*a	(3)
F*F	a*a	(4)
a*F	a*a	(6)
*F	*a	krácení
F	a	krácení
a	a	(6)
λ	λ	krácení

Zásobníkový automat

Zásobníkovým automatem nazýváme sedmici

$$M = (Q,X,Y,\delta,q_0,Z_0,F),$$

kde

- Q - neprázdná konečná množina stavů
- X - neprázdná konečná vstupní abeceda
- Y - neprázdná konečná zásobníková abeceda
- δ - přechodová funkce $Q\times(X\cup\{\lambda\})\times Y \rightarrow P_{FIN}(Q\times Y^*)$
- $q_0 \in Q$ - počáteční stav
- $Z_0 \in Y$ - počáteční zásobníkový symbol
- F - množina koncových stavů

Poznámky:

- ZA je z principu nedeterministický
- vždy nahrazujeme vrchol zásobníku
- nemusíme číst vstupní symbol

Výpočet zásobníkového automatu

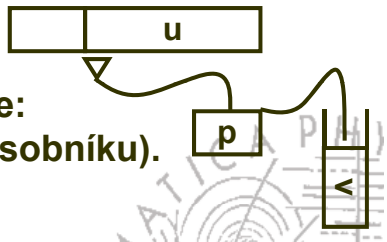
Instrukci $(p,a,Z) \rightarrow (q,w)$ ($(q,w)\in\delta(p,a,Z)$) lze vykonat, pokud:

- stav automatu je p
- na vstupu je symbol *a* (pouze pokud $a\neq\lambda$)
- na vrcholu zásobníku je symbol Z

Vykonání instrukce $(p,a,Z) \rightarrow (q,w)$ znamená:

- změnu stavu automatu z *p* na *q*
- je-li $a\neq\lambda$, posun čtecí hlavy (přečtení písmene *a*)
- smazání vrchního symbolu zásobníku (symbolu Z)
- přidání slova *w* na zásobník (nejvýše bude první písmeno z *w*)

Formalizace kroku zásobníkového automatu:



Situace zásobníkového automatu je trojice (p,u,v), kde:
$p\in Q$, $u\in X^*$ (zbytek čteného slova), $v\in Y^*$ (obsah zásobníku).

Situace $E_1=(p,au,Zv)$ *vede bezprostředně na situaci*
$E_2=(q,u,wv)$, když $p,q\in Q$, $u\in X^*$, $v,w\in Y^*$, $a\in X\cup\{\lambda\}$, $Z\in Y$, $(q,w)\in\delta(p,a,Z)$.
Píšeme $E_1 \vdash E_2$.

Situace E *vede na situaci* E‘ ($E \vdash^* E'$), právě když $E\vdash E_1, E_1\vdash E_2 \dots E_n\vdash E'$

Zásobníkové automaty a jazyky

Kdy končí výpočet zásobníkového automatu:

- zásobník je prázdný
- není definována žádná instrukce

Přijímání koncovým stavem
slovo je celé přečteno a jsme v koncovém stavu

Přijímání prázdným zásobníkem
slovo je celé přečteno a zásobník je vyprázdněný

Nechť M je zásobníkový automat.

Jazyk rozpoznávaný automatem M *koncovým stavem* definujeme takto: $L(M) = \{w \mid w \in X^*, v \in Y^*, q \in F\ (q_0,w,Z_0) \vdash^* (q,\lambda,v)\}$.

Jazyk rozpoznávaný automatem M *prázdným zásobníkem* definujeme takto: $N(M) = \{w \mid w \in X^*, q \in Q\ (q_0,w,Z_0) \vdash^* (q,\lambda,\lambda)\}$.

Koncové stavy nás tady nezajímají, proto klademe $F=\emptyset$.

Zásobníkový automat v příkladě

$L = \{0^n1^n \mid n>0\}$

Přijímání prázdným zásobníkem

p-počáteční stav, Z-počáteční zásobníkový symbol

$\delta(p,0,Z) = \{(p,A)\}$ … čte první symbol 0
$\delta(p,0,A) = \{(p,AA)\}$ … čte další symboly 0
$\delta(p,1,A) = \{(q,\lambda)\}$ … čte první symbol 1
$\delta(q,1,A) = \{(q,\lambda)\}$ … čte další symboly 1

Přijímání koncovým stavem

p-počáteční stav, q_F-koncový stav, Z-počáteční zásobníkový symbol

$\delta(p,0,Z) = \{(p,AZ)\}$ … čte první symbol 0
$\delta(p,0,A) = \{(p,AA)\}$ … čte další symboly 0
$\delta(p,1,A) = \{(q,\lambda)\}$ … čte první symbol 1
$\delta(q,1,A) = \{(q,\lambda)\}$ … čte další symboly 1
$\delta(q,\lambda,Z) = \{(q_F,\lambda)\}$ … končí

Přijímání zásobníkem ⇒ přijímání stavem

Pro každý zásobníkový automat M_1 existuje ekvivalentní zásobníkový automat M_2 tak, že $N(M_1)=L(M_2)$ (prázdný zásobník → koncový stav).

Důkaz (konstruktivní):

idea:

- na zásobník přidáme speciální symbol,
- běžíme stejně jako M_1,
- je-li na zásobníku speciální symbol, končíme

formálně:

$M_1 = (Q_1,X,Y_1,\delta_1,q_1,Z_1,\{\}) \rightarrow M_2 = (Q_2,X,Y_2,\delta_2,q_2,Z_2,\{q_F\})$,

$q_2,q_F \notin Q_1$, $Q_2 = Q_1 \cup \{q_2,q_F\}$,

$Z_2 \notin Y_1$, $Y_2 = Y_1 \cup \{Z_2\}$,

δ_2 „=" $\delta_1 + \delta_2(q_2,\lambda,Z_2)=\{(q_1,Z_1Z_2)\} + \forall q \in Q_1\ \delta_2(q,\lambda,Z_2)=\{(q_F,\lambda)\}$

$N(M_1)=L(M_2)$?

$w \in N(M_1) \Leftrightarrow (q_1,w,Z_1) \vdash^*_{M1} (q,\lambda,\lambda)$

$\Leftrightarrow (q_2,w,Z_2) \vdash_{M2} (q_1,w,Z_1Z_2) \vdash^*_{M2} (q,\lambda,Z_2) \vdash_{M2} (q_F,\lambda,\lambda)$

$\Leftrightarrow w \in L(M_2)$

Příklad převodu

$L = \{0^n1^n \mid n \geq 0\}$

$M_1 = (\{p,q\}, \{0,1\}, \{Z,A\}, \delta, p, Z, \{\})$, $L=N(M_1)$

$\delta(p,\lambda,Z) = \{(p,\lambda)\}$ … čte prázdné slovo

$\delta(p,0,Z) = \{(p,A)\}$

$\delta(p,0,A) = \{(p,AA)\}$

$\delta(p,1,A) = \{(q,\lambda)\}$

$\delta(q,1,A) = \{(q,\lambda)\}$

$M_2 = (\{p,q,q_2,q_F\}, \{0,1\}, \{Z,A,Z_2\}, \delta_2, q_2, Z_2, \{q_F\})$, $L=L(M_2)$

$\delta_2(q_2,\lambda,Z_2) = \{(p,ZZ_2)\}$ … nastartování výpočtu M_1

$\delta_2(p,\lambda,Z) = \{(p,\lambda)\}$

$\delta_2(p,0,Z) = \{(p,A)\}$

$\delta_2(p,0,A) = \{(p,AA)\}$

$\delta_2(p,1,A) = \{(q,\lambda)\}$

$\delta_2(q,1,A) = \{(q,\lambda)\}$

$\delta_2(p,\lambda,Z_2) = \{(q_F,\lambda)\}$ … ukončení výpočtu

$\delta_2(q,\lambda,Z_2) = \{(q_F,\lambda)\}$ … "

Přijímání stavem ⇒ přijímání zásobníkem

Pro každý zásobníkový automat M_1 existuje ekvivalentní zásobníkový automat M_2 tak, že $L(M_1)=N(M_2)$ (koncový stav → prázdný zásobník).

Důkaz (konstruktivní):

idea:

- na zásobník přidáme speciální symbol (proti vyprázdnění),
- běžíme stejně jako M_1,
- v koncovém stavu smažeme zásobník (nedeterminismus!)

formálně:

$M_1 = (Q_1,X,Y_1,\delta_1,q_1,Z_1,F) \rightarrow M_2 = (Q_2,X,Y_2,\delta_2,q_2,Z_2,\{\})$,

$q_2,q_M \notin Q_1$, $Q_2 = Q_1 \cup \{q_2,q_M\}$,

$Z_2 \notin Y_1$, $Y_2 = Y_1 \cup \{Z_2\}$,

δ_2 „=" $\delta_1 + \forall q_F \in F\ \forall Z \in Y_2\ \delta_2(q_F,\lambda,Z)=(\delta_1(q_F,\lambda,Z) \cup \{(q_M,\lambda)\})$,

$+ \delta_2(q_2,\lambda,Z_2)=\{(q_1,Z_1Z_2)\} + \forall Z \in Y_2\ \delta_2(q_M,\lambda,Z)=\{(q_M,\lambda)\}$

$L(M_1)=N(M_2)$?

$w \in L(M_1) \Leftrightarrow (q_1,w,Z_1) \vdash^*_{M1} (q_F,\lambda,v)$

$\Leftrightarrow (q_2,w,Z_2) \vdash_{M2} (q_1,w,Z_1Z_2) \vdash^*_{M2} (q_F,\lambda,vZ_2) \vdash^*_{M2} (q_M,\lambda,\lambda)$

$\Leftrightarrow w \in N(M_2)$

Příklad převodu

$L = \{ w \mid w \in \{0,1\}^*, |w|_0 = |w|_1 \}$

$M_1 = (\{p,q\}, \{0,1\}, \{Z,N,J\}, \delta, p, Z, \{p\})$, $L=L(M_1)$

$\delta(p,0,Z) = \{(q,NZ)\}$ … čte první nulu

$\delta(p,1,Z) = \{(q,JZ)\}$ … čte první jedničku

$\delta(q,0,N) = \{(q,NN)\}$ … přidává další nulu

$\delta(q,0,J) = \{(q,\lambda)\}$ … krátí nulu oproti předchozí jedničce

$\delta(q,1,N) = \{(q,\lambda)\}$ … krátí jedničku oproti předchozí nule

$\delta(q,1,J) = \{(q,JJ)\}$ … přidává další jedničku

$\delta(q,\lambda,Z) = \{(p,Z)\}$ … počet nul a jedniček vyrovnán

$M_2 = (\{p,q,q_2,q_M\}, \{0,1\}, \{Z,N,J,Z_2\}, \delta_2, q_2, Z_2, \{\})$, $L=N(M_2)$

$\delta_2(q_2,\lambda,Z_2) = \{(p,ZZ_2)\}$ … nastartování výpočtu M_1

" … stejné instrukce jako u M_1

$\delta_2(p,\lambda,X) = \{(q_M, \lambda)\}$ $\forall X \in \{Z,N,J,Z_2\}$ … přechod do mazacího stavu

$\delta_2(q_M,\lambda,X) = \{(q_M, \lambda)\}$ $\forall X \in \{Z,N,J,Z_2\}$ … mazání zásobníku

Od gramatiky k automatu

Každý bezkontextový jazyk je rozpoznáván zásobníkovým automatem prázdným zásobníkem.

Důkaz (konstruktivní):

idea:

- ze vstupu čteme terminály, na zásobníku terminály i neterminály
- pravidla gramatiky → pravidla automatu (opracování zásobníku)
- přidáme pravidla pro krácení terminálu na vstupu a na zásobníku
- stačí jediný stav

formálně:

$G=(V_N,V_T,S,P) \rightarrow M=(\{p\},V_T,V_N \cup V_T,\delta,p,S,\{\})$

$\delta(p,\lambda,X)=\{(p,w) \mid (X \rightarrow w) \in P\} \quad \forall X \in V_N$ … opracování zásobníku

$\delta(p,a,a)=\{(p,\lambda)\} \quad \forall a \in V_T$ … krácení terminálů

Příklad: $A \rightarrow bAc \mid \lambda$ … $\delta(p,\lambda,A) = \{ (p,bAc), (p,\lambda) \}$

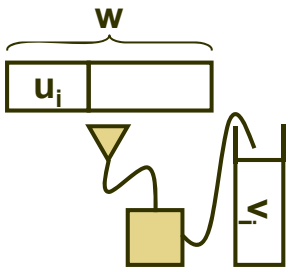
Od gramatiky k automatu - pokračování

$L(G) \supseteq N(M)$?

„Situaci“ v kroku *i* popíšeme pomocí slova $w_i=u_iv_i$

u_i - dosud přečtená část vstupního slova

v_i - ještě nezpracovaná část (zásobník)



provedení kroku

a) $v_i = Zv_i'$ kde $Z \in V_N$, potom
$u_{i+1} = u_i$ (nic nečteme), $v_{i+1} = vv_i'$ (dle pravidla gramatiky $Z \rightarrow v$)
zřejmě tedy $w_i \Rightarrow w_{i+1}$ (přímý přepis v gramatice)

b) $v_i = av_i'$ kde $a \in V_T$, potom $u_{i+1} = u_ia$, $v_{i+1} = v_i'$, tedy $w_{i+1} = w_i$

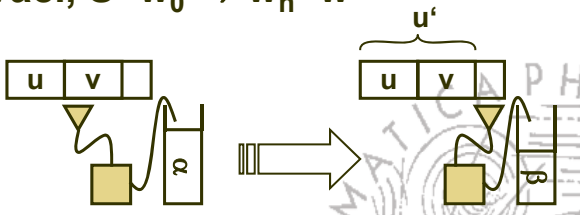
výpočet automatu tedy definuje derivaci, $S=w_0 \Rightarrow w_n=w$

$L(G) \subseteq N(M)$?

vezmeme levé derivace

$u\alpha \Rightarrow^* u'\beta$, kde $u'=uv$, $u,u' \in V_T^*$ potom $(p,v,\alpha) \vdash^* (p,\lambda,\beta)$

dohromady $S \Rightarrow^* w$ a tedy $(p,w,S) \vdash^* (p, \lambda, \lambda)$



Od gramatiky k automatu - příklad

$G=(V_N,V_T,S,P) \rightarrow M=(\{p\},V_T,V_N \cup V_T,\delta,p,S,\{\})$

$\delta(p,\lambda,X)=\{(p,w) \mid (X \rightarrow w) \in P\} \quad \forall X \in V_N$ … opracování zásobníku

$\delta(p,a,a)=\{(p,\lambda)\} \quad \forall a \in V_T$ … krácení terminálů

Příklad: $L= \{a^i b^j c^k \mid i+j=k\}$

Gramatika	*Zásobníkový automat*
$S \rightarrow aSc \mid A$ $A \rightarrow bAc \mid \lambda$	$\delta(p,\lambda,S) = \{ (p,aSc), (p,A) \}$ $\delta(p,\lambda,A) = \{ (p,bAc), (p,\lambda) \}$ $\delta(p,x,x) = \{ (p,\lambda) \} \quad \forall x \in \{a,b,c\}$

$S \Rightarrow aSc \Rightarrow aAc \Rightarrow abAcc \Rightarrow abbAccc \Rightarrow abbccc$

$(p,abbccc,S) \vdash (p,abbccc,aSc) \vdash (p,bbccc,Sc) \vdash (p,bbccc,Ac)$
$\vdash (p,bbccc,bAcc) \vdash (p,bccc,Acc) \vdash (p,bccc,bAccc)$
$\vdash (p,ccc,Accc) \vdash (p,ccc,ccc) \vdash (p,cc,cc) \vdash (p,c,c) \vdash (p, \lambda, \lambda)$

Od automatu ke gramatice

Pro jednostavový ZA, stačí reverzní proces k BKG→ZA.

Pro více-stavový ZA:

- převést na jednostavový ZA
- přímo na gramatiku

Převod více-stavového ZA na gramatiku:

pravidla gramatiky zachycují všechny možné výpočty

neterminální symboly: $[q,Z,p]$, kde $p,q \in Q$, $Z \in Y$

q je stav automatu těsně před tím, než se Z přepíše

p je stav automatu, když začínáme počítat pod Z

pravidla gramatiky:

$S \rightarrow [q_0,Z_0,p]$ nastartování výpočtu (p - nevíme, kde skončí)

$[q,A,p] \rightarrow a\ [q_1,B_1,q_2]\ [q_2,B_2,q_3] \ldots [q_m,B_m,p]$

$\delta(q,a,A) \ni (q_1, B_1 B_2 \ldots B_m)$

$q_2,\ldots, q_m,p$ libovolné stavy - nevíme, jak výpočet vypadá

speciálně: $[q,A,p] \rightarrow a$, pro $\delta(q,a,A) \ni (p,\lambda)$

Výpočet automatu → derivace

$(q,w,A) \vdash^* (p,\lambda,\lambda) \quad \Rightarrow \quad [q,A,p] \Rightarrow^* w$

indukcí dle délky výpočtu

k=1

$w \in X \cup \{\lambda\}$, $\delta(q,w,A) \ni (p,\lambda)$, tj. máme pravidlo $[q,A,p] \to w$

k>1 (pro výpočty kratší než k platí)

$(q,au_1 \ldots u_l,A) \vdash (q_1, u_1 \ldots u_l, B_1 \ldots B_l)$, $l \geq 1$ první krok výpočtu dle přechodu $\delta(q,a,A) \ni (q_1, B_1 B_2 \ldots B_l)$

u_i jsou slova nutná ke zpracování zásobníkového symbolu B_i

tj. $(q_i,u_i,B_i) \vdash^* (q_{i+1},\lambda,\lambda)$, pro vhodná q_i ($q_{l+1}=p$)

tyto výpočty jsou nutně kratší než *k*

tj. dle indukčního předpokladu $[q_i,B_i,q_{i+1}] \Rightarrow^* u_i$

dohromady:

$[q,A,p] \to a\,[q_1,B_1,q_2]\,[q_2,B_2,q_3] \ldots [q_l,B_l,p]$

$[q,A,p] \Rightarrow^* a \quad u_1 \quad u_2 \quad \ldots \quad u_l$

Derivace → výpočet automatu

$[q,A,p] \Rightarrow^* w \quad \Rightarrow \quad (q,w,A) \vdash^* (p,\lambda,\lambda)$

indukcí dle délky (levé) derivace

k=1

jediné pravidlo $[q,A,p] \to w$ muselo vzniknout z $\delta(q,w,A) \ni (p,\lambda)$

k>1 (pro derivace kratší než k platí)

$[q,A,p] \to a\,[q_1,B_1,q_2]\,[q_2,B_2,q_3] \ldots [q_l,B_l,p]$ první použité pravidlo vzniklo z přechodu $\delta(q,a,A) \ni (q_1, B_1 B_2 \ldots B_l)$

potom $w = a\,u_1 \ldots u_l$, kde $[q_i,B_i,q_{i+1}] \Rightarrow^* u_i$ ($q_{l+1}=p$)

tyto derivace jsou nutně kratší než *k*

tj. dle indukčního předpokladu $(q_i,u_i,B_i) \vdash^* (q_{i+1},\lambda,\lambda)$

dohromady: „slepíme“ výpočty a dostaneme $(q,w,A) \vdash (q_1, u_1 \ldots u_l, B_1 B_2 \ldots B_l) \vdash^* (p,\lambda,\lambda)$

Derivace vždy začíná nějakým pravidlem $S \to [q_0,Z_0,q]$, tj. $L(G)=N(M)$.

Příklad převodu automatu na gramatiku

$\delta(q,x,A) \ni \{(q_1,B^1..B^m)\}$ ..... ${}_qA_p \to x\,[{}_{q_1}B^1{}_{q_2}] \ldots [{}_{q_m}B^m{}_p]$

Příklad: $L = \{0^n1^n \mid n \geq 0\}$

Automat	*Gramatika*
	$S \to {}_pZ_p \mid {}_pZ_q$
$\delta(p,\lambda,Z) = \{(p,\lambda)\}$	${}_pZ_p \to \lambda$
$\delta(p,0,Z) = \{(p,A)\}$	~~${}_pZ_p \to 0\ {}_pA_p$~~ ${}_pZ_q \to 0\ {}_pA_q$
$\delta(p,0,A) = \{(p,AA)\}$	~~${}_pA_p \to 0\ {}_pA_p\ {}_pA_p \mid 0\ {}_pA_q\ {}_qA_p$~~ ~~${}_pA_q \to 0\ {}_pA_p\ {}_pA_q \mid 0\ {}_pA_q\ {}_qA_q$~~
$\delta(p,1,A) = \{(q,\lambda)\}$	${}_pA_q \to 1$
$\delta(q,1,A) = \{(q,\lambda)\}$	${}_qA_q \to 1$

$S \Rightarrow {}_pZ_q \Rightarrow 0\ {}_pA_q \Rightarrow 00\ {}_pA_q\ {}_qA_q \Rightarrow 001\ {}_qA_q \Rightarrow 0011$

Deterministické zásobníkové automaty

Jak je to s nedeterminismem zásobníkových automatů?

- Je nutný!
- Pro rozpoznání ww^R potřebujeme nedeterministicky uhádnout střed.

Kde je skryt nedeterminismus zásobníkového automatu?

- množina možných přechodů
- opracování zásobníku bez čtení vstupu (λ-přechod)

Definice: Říkáme, že *zásobníkový automat* $M=(Q,X,Y,\delta,q_0,Z_0,F)$, je *deterministický*, jestliže platí:

- $\forall p \in Q,\ \forall a \in X \cup \{\lambda\},\ \forall Z \in Y \quad |\delta(p,a,Z)| \leq 1$
- $\forall p \in Q,\ \forall Z \in Y\ (\ \delta(p,\lambda,Z) \neq \emptyset \Rightarrow \forall a \in X\ \delta(p,a,Z) = \emptyset\ )$

Každý krok výpočtu je přesně určen.

Příklady zásobníkových automatů

Deterministický zásobníkový automat (prázdný zásobník)

$L = \{0^n1^n \mid n>0\}$

$\delta(p,0,Z) = \{(p,A)\}$ … čte první symbol 0
$\delta(p,0,A) = \{(p,AA)\}$ … čte další symboly 0
$\delta(p,1,A) = \{(q,\lambda)\}$ … čte první symbol 1
$\delta(q,1,A) = \{(q,\lambda)\}$ … čte další symboly 1

(Klasický) zásobníkový automat (prázdný zásobník)

$L = \{ww^R \mid w\in\{0,1\}^* \}$

$\delta(p,0,X) = \{(p,NX)\}$ … čte 0 v první půlce (uschovává na zásobníku)
$\delta(p,1,X) = \{(p,JX)\}$ … čte 1 v první půlce (uschovává na zásobníku)
$\delta(p,\lambda,X) = \{(q,X)\}$ … překlopení do druhé půlky (nedeterminismus)
$X\in\{Z,N,J\}$
$\delta(q,0,N) = \{(q,\lambda)\}$ … čte 0 symetricky v druhé půlce
$\delta(q,1,J) = \{(q,\lambda)\}$ … čte 1 symetricky v druhé půlce
$\delta(q,\lambda,Z) = \{(q,\lambda)\}$ … ukončuje výpočet (není nedeterminismus)

Deterministické a bezprefixové jazyky

Již „víme“, že determinismus je u ZA slabší než nedeterminismus (ww^R).

Jak je to s příjímáním slov?

Deterministické bezkontextové jazyky jsou jazyky rozpoznávané DZA koncovým stavem.

Tvrzení: Regulární jazyk je deterministický BKJ.

Zásobník nemusíme využívat.

Bezprefixové bezkontextové jazyky jsou jazyky rozpoznávané DZA prázdným zásobníkem.

Pozorování: M je DZA, $u\in N(M) \Rightarrow \forall w\in X^+ \; uw\notin N(M)$

Jakmile jednou vyprázdníme zásobník (po přečtení *u*, které je přijato), nemůže výpočet pokračovat (čtením *w*).

Tvrzení: Bezprefixový BKJ je deterministický BKJ.

Převod nepřidává nedeterminismus.

Koncové stavy vs. prázdný zásobník

Je také každý deterministický jazyk bezprefixový?

NE!

Z vlastnosti M je DZA, $u \in N(M) \Rightarrow \forall w \in X^+ \; uw \notin N(M)$ snadno sestrojíme příslušný jazyk.

Potřebujeme:

základní deterministický jazyk, který

obsahuje slovo, jež je prefixem jiného přijímaného slova

Například $\{0^n1^m \mid 0<n\leq m\}$ (uděláme DZA, q_F je koncový stav).

$\delta(p,0,Z) = \{(p,AZ)\}$ … čte první symbol 0

$\delta(p,0,A) = \{(p,AA)\}$ … čte další symboly 0

$\delta(p,1,A) = \{(q,\lambda)\}$ … čte první symbol 1

$\delta(q,1,A) = \{(q,\lambda)\}$ … čte další symboly 1

$\delta(q,\lambda,Z) = \{(q_F,Z)\}$ … nyní se počet 0 a 1 vyrovnal

$\delta(q_F,1,Z) = \{(q_F,Z)\}$ … dále čteme jen 1
to už „prázdný zásobník“ nemůže

Jak na bezprefixové jazyky?

Jak se od deterministického jazyka dostaneme k bezprefixovému?

Co nám vadí?

po přečtení prefixu, který patří do jazyka, máme prázdný zásobník.

Řešení:

přidáme speciální znak na konec slova (až po přečtení tohoto znaku vyprázdníme zásobník)

Nechť $L \subseteq X^*$ je deterministický jazyk a $\# \notin X$, potom $L\#$ je bezprefixový jazyk.

opraví nedeterministické λ -kroky při převodu automatů

Příklad: $\{0^n1^m\# \mid 0<n\leq m\}$

$\delta(p,0,Z) = \{(p,AZ)\}$ … čte první symbol 0

$\delta(p,0,A) = \{(p,AA)\}$ … čte další symboly 0

$\delta(p,1,A) = \{(q,\lambda)\}$ … čte první symbol 1

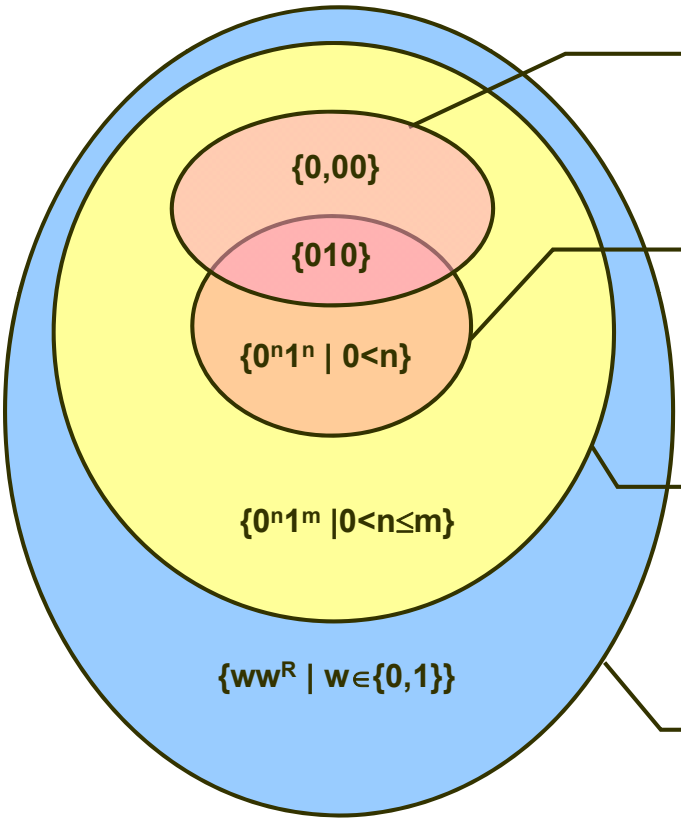
$\delta(q,1,A) = \{(q,\lambda)\}$ … čte další symboly 1

$\delta(q,\lambda,Z) = \{(q_{OK},Z)\}$ … nyní se počet 0 a 1 vyrovnal

$\delta(q_{OK},1,Z) = \{(q_{OK},Z)\}$ … dále čteme jen 1

$\delta(q_{OK}, \#,Z) = \{(q_{OK}, \lambda)\}$ … na konci vyprázdníme zásobník

Jazyky a automaty - přehled



REGULÁRNÍ JAZYKY
konečný automat

BEZPREFIXOVÉ BK JAZYKY
deterministický zásobníkový automat
přijímání prázdným zásobníkem

DETERMINISTICKÉ BK JAZYKY
deterministický zásobníkový automat
přijímání koncovým stavem

BEZKONTEXTOVÉ JAZYKY
zásobníkový automat

Greibachové normální forma

Co nám vadí při analýza slova?

Nevíme, jaké pravidlo vybrat!

Speciálně vadí pravidla tvaru $A \rightarrow Au$ (levá rekurze).

Definice: Říkáme, že gramatika je v *Greibachové normální formě* (tvaru), jestliže všechna pravidla mají tvar:
$A \rightarrow au$, kde $a \in V_T$, $u \in V^*_N$.

K čemu je tento tvar dobrý?

Srovnáním terminálu na pravé straně pravidel a čteného symbolu můžeme zjistit, jaké pravidlo použít,

pokud je ovšem takové pravidlo jediné.

Věta: Ke každému bezkontextovému jazyku L existuje bezkontextová gramatika G v Greibachové normální formě taková, že $L(G) = L - \{\lambda\}$.

Spojení pravidel a odstranění levé rekurze

Lemma (spojení pravidel):

Nechť $A \to uBv$ je pravidlo gramatiky G a $B \to w_1, \ldots, B \to w_k$ jsou všechna pravidla pro B. Potom nahrazením pravidla $A \to uBv$ pravidly $A \to uw_1v, \ldots, A \to uw_kv$ dostaneme ekvivalentní gramatiku.

Důkaz:

$A \Rightarrow uBv \Rightarrow^* u'Bv \Rightarrow u'w_iv$ v původní gramatice

$A \Rightarrow uw_iv \Rightarrow^* u'w_iv$ v nové gramatice

Lemma (odstranění levé rekurze):

Nechť $A \to Au_1, \ldots, A \to Au_k$ jsou všechna levě rekurzivní pravidla gramatiky G pro A a $A \to v_1, \ldots, A \to v_m$ jsou všechna ostatní pravidla pro A. Potom nahrazením všech těchto pravidel pravidly:

1) $A \to v_i$, $A \to v_iZ$, $Z \to u_j$, $Z \to u_jZ$, nebo
2) $A \to v_iZ$, $Z \to u_jZ$, $Z \to \lambda$

(Z je nový neterminál) dostaneme ekvivalentní gramatiku.

Důkaz:

$A \Rightarrow Au_{i_n} \Rightarrow \ldots \Rightarrow Au_{i_1}\ldots u_{i_n} \Rightarrow v_ju_{i_1}\ldots u_{i_n}$ (G)

$A \Rightarrow v_jZ \Rightarrow v_ju_{i_1}Z \Rightarrow \ldots \Rightarrow v_ju_{i_1}\ldots u_{i_{n-1}}Z \Rightarrow v_ju_{i_1}\ldots u_{i_n}$ (1)

$A \Rightarrow v_jZ \Rightarrow v_ju_{i_1}Z \Rightarrow \ldots \Rightarrow v_ju_{i_1}\ldots u_{i_n}Z \Rightarrow v_ju_{i_1}\ldots u_{i_n}$ (2)

Převod na Greibachové NF

Věta: Libovolnou bezkontextovou gramatiku lze převést na gramatiku v Greibachové normální formě.

Důkaz:

spojování pravidel a odstraňování levé rekurze

1) neterminály libovolně očíslujeme $\{A_1, \ldots, A_n\}$
2) povolíme rekurzivní pravidla pouze tvaru $A_i \to A_ju$, kde $i<j$

   postupnou iterací od 1 do n

   $A_i \to A_ju$ pro $j<i$ odstraníme spojováním pravidel
   pro $j=i$ odstraníme levou rekurzi

   získáme pravidla tvaru $A_i \to A_ju$ $(i<j)$, $A_i \to au$ $(a \in V_T)$, $Z_i \to u$
3) pravidla s A_i (původní neterminály) pouze tvaru $A_i \to au$

   postupným spojování pravidel od n do 1 (pro n již platí)
4) pravidla s Z_i (nové neterminály) pouze tvaru $Z_i \to au$

   žádné pravidlo Z_i pro nezačíná vpravo Z_j

   buď je v požadovaném tvaru nebo se spojí z pravidlem $A_j \to au$
5) odstranění terminálů uvnitř pravidel

Příklad převodu na Greibachové NF

Původní gramatika

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid a$

Odstranění levé rekurze

$E \to T \mid TE'$

$E' \to +T \mid +TE'$

$T \to F \mid FT'$

$T' \to *F \mid *FT'$

$F \to (E) \mid a$

(téměř) Greibachové normální forma

$E \to (E) \mid a \mid (E)T' \mid aT' \mid (E)E' \mid aE' \mid (E)T'E' \mid aT'E'$

$E' \to +T \mid +TE'$

$T \to (E) \mid a \mid (E)T' \mid aT'$

$T' \to *F \mid *FT'$

$F \to (E) \mid a$

Chomského normální forma

Podívejme se nyní na derivační stromy.

Jak odhadnout výšku stromu podle délky slova?

Definice: Říkáme, že gramatika je v *Chomského normální formě* (tvaru), jestliže všechna pravidla mají tvar:
$X \to YZ$ nebo $X \to a$, kde $a \in V_T$, $X,Y,Z \in V_N$.

K čemu je tento tvar dobrý?

Derivační strom je (skoro) binární.

Je-li maximální cesta délky k, potom terminální slovo $\leq 2^{k-1}$.

Pumping lemma pro bezkontextové jazyky

Věta: Ka každému bezkontextovému jazyku L existuje bezkontextová gramatika G v Chomského normální formě taková, že $L(G) = L - \{\lambda\}$.

Převod na Chomského NF

Povoleno pouze $X \to YZ$ nebo $X \to a$, kde $a \in V_T$, $X,Y,Z \in V_N$.

Víme:

pravidla $A \to B$ lze odstranit (viz regulární gramatiky)

pravidla $A \to \lambda$ lze odstranit (maximálně přijdeme o λ)

Dále:

pravidla tvaru $X \to a$, kde $a \in V_T$ jsou v pořádku

zbývají pravidla tvaru $X \to B_1 \ldots B_k$, $k \geq 2$, $B_i \in V_N \cup V_T$

vytvoříme pravidlo $X \to C_1 \ldots C_k$, kde:

C_i $= B_i$ je-li $B_i \in V_N$

$= B'_i$ je-li $B_i \in V_T$ (B'_i je nový neterminál)

+ přidáme pravidla $B'_i \to B_i$

pravidlo $X \to C_1 \ldots C_k$, $k \geq 3$ nahradíme pravidly:

$X \to C_1D_1$, $D_1 \to C_2D_2$, …, $D_{k-2} \to C_{k-1}C_k$,

D_i jsou nové neterminály

Příklad převodu na Chomského NF

Původní gramatika

$A \to B \mid C$

$B \to 0B1 \mid 01$

$C \to D \mid E$

$D \to 1D0 \mid 1$

$E \to 0E \mid 0$

Po odstranění $X \to Y$

$A \to 0B1 \mid 01 \mid 1D0 \mid 1 \mid 0E \mid 0$

$B \to 0B1 \mid 01$

~~$C \to 1D0 \mid 1 \mid 0E \mid 0$~~

$D \to 1D0 \mid 1$

$E \to 0E \mid 0$

Po nahrazení terminálů

$A \to NBJ \mid NJ \mid JDN \mid 1 \mid NE \mid 0$

$B \to NBJ \mid NJ$

$D \to JDN \mid 1$

$E \to NE \mid 0$

$N \to 0$

$J \to 1$

Chomského normální forma

$A \to NA_1 \mid NJ \mid JA_2 \mid 1 \mid NE \mid 0$

$A_1 \to BJ$

$A_2 \to DN$

$B \to NB_1 \mid NJ$

$B_1 \to BJ$

$D \to JD_1 \mid 1$

$D_1 \to DN$

$E \to NE \mid 0$

$N \to 0$

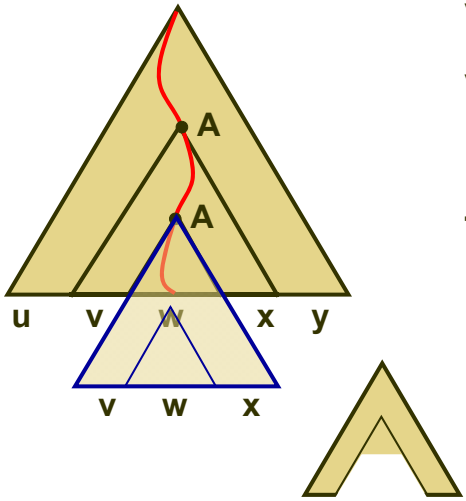
$J \to 1$

Pumping lemma pro bezkontexové jazyky

Lemma o vkládání: **Nechť L je bezkontextový jazyk. Potom existují přirozená čísla p, q taková, že každé slovo $z \in L$, $|z|>p$ lze psát ve tvaru $z = uvwxy$ a platí:**

1) $|vwx| \leq q$ (pumpovací část není moc dlouhá)
2) buď $v \neq \lambda$ nebo $x \neq \lambda$ (lze psát $vx \neq \lambda$)
3) $\forall i \geq 0\ uv^iwx^iy \in L$

Idea důkazu:



vezmeme derivační strom pro slovo z

v derivačním stromu najdeme nejdelší cestu

na této cestě najdeme dva stejné neterminály

tyto neterminály určí dva podstromy

podstromy definují rozklad slova

nyní můžeme větší podstrom posunout (i>1)

nebo nahradit menším podstromem (i=0)

Důkaz lemma o vkládání pro BKJ

$|z|>p : z = uvwxy, |vwx| \leq q, vx \neq \lambda, \forall i \geq 0\ uv^iwx^iy \in L$

vezmeme gramatiku v Chomského NF (slova λ nevadí)

$|V_N|=k$, položme $p = 2^{k-1}$, $q = 2^k$

$|z| > 2^{k-1}$, v libovolném derivačním stromu je cesta délky >k

na této (nejdelší) cestě musí ležet dva stejné neterminály a terminál *t*

vezmeme dvojici A^1, A^2 nejblíže k *t* (určuje podstromy T^1 a T^2)

cesta z A^1 do *t* je nejdelší v podstromu T^1 a má délku maximálně k+1

tedy slovo dané stromem T^1 není delší než 2^k ($|vwx| \leq q$))

z A^1 vedou dvě cesty (ChNF), jedna do T^2 druhá do zbytku vx

ChNF je nevypouštějící, tedy $vx \neq \lambda$

derivace slova ($A^1 \Rightarrow^* vA^2x$, $A^2 \Rightarrow^* w$)

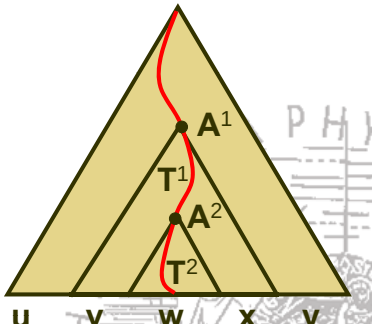
$S \Rightarrow^* uA^1y \Rightarrow^* uvA^2xy \Rightarrow^* uvwxy$

posuneme-li A^2 do A^1 (i=0)

$S \Rightarrow^* uA^2y \Rightarrow^* uwy$

posuneme-li A^1 do A^2 (i=2,…)

$S \Rightarrow^* uA^1y \Rightarrow^* uvA^1xy \Rightarrow^* uvvA^2xxy \Rightarrow^* uvvwxxy$



Použití lemma o vkládání

Jak ukázat, že daný jazyk není bezkontextový?

Příklad 1: $L = \{a^n b^n c^n \mid n \geq 1\}$ není bezkontextový jazyk sporem

- zvolme $k=\max(p,q)$, potom $|a^k b^k c^k| > p$
- pumpovací slovo není delší než q
- tj. vždy lze pumpovat maximálně dva různé symboly
- poruší se rovnost počtu symbolů - SPOR

Příklad 2: $L = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$ není bezkontextový jazyk sporem

- zvolme $n=\max(p,q)$, potom $|a^n b^n c^n| > p$
- pumpovací slovo není delší než q
- tj. vždy lze pumpovat maximálně dva různé symboly
- pokud pumpujeme *a* (případně *b*), pumpujeme nahoru
- pokud pumpujeme *c* (případně *b*), pumpujeme dolu
- potom $i>k$ ($a\uparrow$, $c\downarrow$) nebo $j>k$ ($b\uparrow$, $c\downarrow$) nebo $i>j$ ($a\uparrow$, $b\downarrow$) - SPOR

Kdy lemma o vkládání nezabere

Pozor! Lemma o vkládání je pouze implikace!

- BKJ $\Rightarrow$ lze pumpovat (nutná podmínka bezkontextovosti)
- nejedná se o podmínku postačující

Příklad:

$L = \{a^i b^j c^k d^l \mid i=0 \vee j=k=l\}$ není bezkontextový jazyk

přesto lze pumpovat

- $i=0$: $b^j c^k d^l$ lze pumpovat v libovolném písmenu
- $i>0$: $a^i b^n c^n d^n$ lze pumpovat v části obsahující *a*

Jak na to?

- zobecnění pumping lemmatu (Ogdenovo lemma)
  - pumpování vyznačených symbolů
- uzávěrové vlastnosti

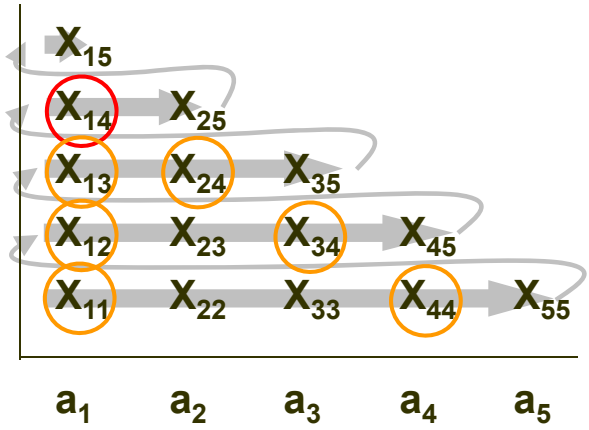
Algoritmus CYK (Cocke, Younger, Kasami)

Jak zjistíme příslušnost slova $a_1a_2\ldots a_n$ do BKJ?

vezmeme gramatiku v ChNF

a podle ní vyplníme následující tabulku (dynamické programování)

- myšlenka: $X_{i,j} = \{A \mid A \Rightarrow^* a_i a_{i+1}\ldots a_j\}$
- začneme: $X_{i,i} = \{A \mid (A \to a_i) \in P\}$
- pokračujeme: $X_{i,j} = \{A \mid \exists k: i \le k < j \;\; B \in X_{i,k} \wedge C \in X_{k+1,j} \wedge (A \to BC) \in P\}$
- pokud $S \in X_{1,n}$, potom $a_1a_2\ldots a_n$ patří do jazyka



S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{S,A,C}				
-	{S,A,C}			
-	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

Nekonečnost bezkontextových jazyků

Pro každý BKJ L existují přirozená čísla m, n taková, že:
L je nekonečný $\Leftrightarrow \exists z \in L \; m < |z| \le n$.

Důkaz:

z lemmatu o vkládání máme *p* a *q*, položme: m = p, n = p+q

„$\Leftarrow$“

p<|z|, tedy *z* lze pumpovat $\Rightarrow$ jazyk je nekonečný

„$\Rightarrow$ “

jazyk je nekonečný $\Rightarrow \exists z \in L \; p = m < |z|$

vezmeme nejkratší takové *z* a potom $|z| \le n = p+q$

sporem: nechť p+q < |z|, lze pumpovat dolu, tj. |z‘| < |z|
odstraňujeme část o max. velikosti q, tedy p < |z‘| - SPOR

Rychlejší algoritmus:

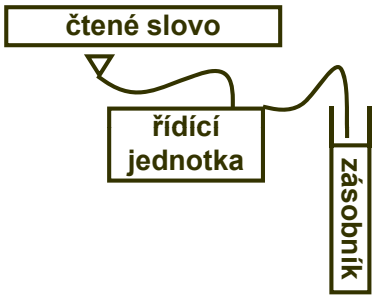
vezmeme redukovanou gramatiku G v ChNF tž. L=L(G)

uděláme orientovaný graf

vrcholy = neterminály, hrany = {(A,B), (A,C) pro $(A \to BC) \in P(G)$}

hledáme orientovaný cyklus (existuje $\Rightarrow$ jazyk je nekonečný)

Jak charakterizovat bezkontextové jazyky?



Pokud do zásobníku pouze přidáváme
potom si stačí pamatovat poslední symbol.
Stačí konečná paměť → konečný automat.

Potřebujeme ze zásobníku také odebírat (čtení symbolu)!
Takový proces nelze zaznamenat v konečné struktuře.
Přidávání a odebírání ale není zcela libovolné
jedná se o zásobník tj. LIFO (last-in first-out) strukturu
Roztáhněme si výpočet se zásobníkem do lineární struktury
X - symbol přidán do zásobníku
X^{-1} - symbol odebrán se zásobníku

$Z\ Z^{-1}\ B\ A\ A^{-1}\ C\ C^{-1}\ B^{-1}$

Přidávaný a odebíraný symbol tvoří pár, který se v celé posloupnosti chová jako závorka!

Dyckovy jazyky

Dyckův jazyk D_n je definován nad abecedou
$Z_n = \{a_1,a'_1,\ldots,a_n,a'_n\}$ následující gramatikou:
$S \rightarrow \lambda \mid SS \mid a_1Sa'_1 \mid \ldots \mid a_nSa'_n$.

Úvodní poznámky:

- Jedná se zřejmě o jazyk bezkontextový.
- Dyckův jazyk D_n popisuje správně uzávorkované výrazy s *n* druhy závorek.
- Tímto jazykem lze popisovat výpočty libovolného zásobníkového automatu.
- Pomocí Dyckova jazyka lze popsat libovolný bezkontextový jazyk.

$L = h(D \cap R)$

Regulární jazyk
popisuje všechny kroky výpočtu

Dyckův jazyk
vybírá pouze korektní výpočty

Homomorfismus
čistí pomocné symboly

Dyckovy jazyky a bezkontextové jazyky

Pro každý bezkontextový jazyk L existuje regulární jazyk R tak, že L=h(D∩R) pro vhodný Dyckův jazyk D a homomorfismus h.

Důkaz:

máme zásobníkový automat přijímající L prázdným zásobníkem

BÚNO instrukce tvaru $\delta(q,a,Z) \ni (p,w)$, $|w| \leq 2$

nechť R‘ obsahuje všechny výrazy

$q^{-1}aa^{-1}Z^{-1}BAp$ pro instrukci $\delta(q,a,Z) \ni (p,AB)$

podobně pro instrukce $\delta(q,a,Z) \ni (p,A)$ a $\delta(q,a,Z) \ni (p,\lambda)$

je-li $a=\lambda$, potom dvojici aa^{-1} nezařazujeme

definujme R takto $Z_0 q_0 R'^* Q^{-1}$

Dyckův jazyk je definován nad abecedou $X \cup X^{-1} \cup Q \cup Q^{-1} \cup Y \cup Y^{-1}$

$D \cap Z_0 q_0 R'^* Q^{-1}$ popisuje korektní výpočty

$$Z_0\ q_0\ q_0^{-1}\ a\ a^{-1}\ Z_0^{-1}\ B\ A\ p\ \ p^{-1}\ b\ b^{-1}\ A^{-1}\ q\ \ q^{-1}\ c\ c^{-1}\ B^{-1}\ r\ \ r^{-1}$$

homomorfismus h vydělí přečtené slovo, tj.

$h(a) = a$ pro vstupní (čtené) symboly

$h(y) = \lambda$ pro ostatní symboly

Průnik bezkontextových jazyků

Bezkontextové jazyky nejsou uzavřené na průnik.

Důkaz:

stačí najít dva BKJ, jejichž průnik není BKJ

$L_1 = \{a^i b^i c^j \mid 0 \leq i,j\}$ $\{S \rightarrow AC,\ A \rightarrow aAb \mid \lambda,\ C \rightarrow cC \mid \lambda\}$

$L_2 = \{a^i b^j c^j \mid 0 \leq i,j\}$ $\{S \rightarrow AB,\ A \rightarrow aA \mid \lambda,\ B \rightarrow bBc \mid \lambda\}$

$L_1 \cap L_2 = \{a^i b^i c^i \mid 0 \leq i\}$ není BKJ (víme z pumping lemmatu)

Pozorování:

paralelní běh dvou zásobníkových automatů

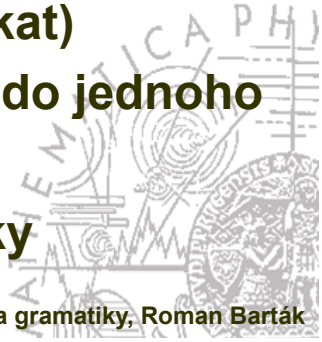
řídící jednotky umíme spojit (viz konečné automaty)

čtení umíme spojit (jeden automat může čekat)

bohužel dva zásobníky nelze obecně spojit do jednoho

dva zásobníky = Turingův stroj

= rekurzivně spočetné jazyky



Průnik bezkontextového a regulárního jazyka

(Deterministické) bezkontextové jazyky jsou uzavřené na průnik s regulárním jazykem.

Důkaz:

zásobníkový a konečný automat můžeme spojit

konečný automat $A_1 = (Q_1, X, \delta_1, q_1, F_1)$

zásobníkový automat (přijímání stavem) $M_2 = (Q_2, X, Y, \delta_2, q_2, Z_0, F_2)$

nový automat $M = (Q_1 \times Q_2, X, Y, \delta, (q_1,q_2), Z_0, F_1 \times F_2)$

$((p',q'), u) \in \delta((p,q),a,Z)$ právě když

- $a \neq \lambda$: $p' \in \delta_1(p,a) \wedge (q',u) \in \delta_2(q,a,Z)$ … automaty čtou vstup
- $a = \lambda$: $(q',u) \in \delta_2(q,\lambda,Z)$ … ZA mění zásobník
  $p'=p$ … KA stojí

zřejmě $L(M) = L(A_1) \cap L(M_2)$

paralelní běh automatů

Použití uzavřenosti průniku BKJ a RJ

$L = \{a^i b^j c^k d^l \mid i=0 \vee j=k=l\}$ není bezkontextový jazyk

SPOREM:

nechť L je bezkontextový jazyk

$L_1 = \{ab^i c^j d^k \mid 0 \leq i,j,k\}$ je regulární jazyk

$S \rightarrow aB,\quad B \rightarrow bB \mid C,\quad C \rightarrow cC \mid D,\quad D \rightarrow dD \mid \lambda$

$L \cap L_1 = \{ab^i c^i d^i \mid 0 \leq i\}$ není bezkontextový jazyk - SPOR

L je kontextový jazyk

$S \rightarrow B' \mid aA$

$B' \rightarrow bB' \mid C',\quad C' \rightarrow cC' \mid D',\quad D' \rightarrow dD' \mid \lambda$

$A \rightarrow aA \mid P$

$P \rightarrow bPCD \mid \lambda$

$DC \rightarrow CD$ $\quad \{DC \rightarrow XC, XC \rightarrow XY, XY \rightarrow CY, CY \rightarrow CD\}$

$bC \rightarrow bc,\ cC \rightarrow cc,\ cD \rightarrow cd,\ dD \rightarrow dd$

Sjednocení a doplněk BKJ

Bezkontextové jazyky jsou uzavřené na sjednocení.

použijeme gramatiky (pro ZA nedeterministický rozeskok na startu)

$L_1=L(G_1)$ $G_1 = (V_{N1},V_{T1},S_1,P_1)$

$L_2=L(G_2)$ $G_2 = (V_{N2},V_{T2},S_2,P_2)$

můžeme předpokládat, že $V_{N1} \cap V_{N2} = \varnothing$ (jinak přejmenuj)

uděláme gramatiku:

$G = (V_{N1} \cup V_{N2} \cup \{S\}, V_{T1} \cup V_{T2}, S, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\})$

zřejmě $L(G) = L(G_1) \cup L(G_2)$

„počítá“ jedna nebo druhá gramatika

Bezkontextové jazyky nejsou uzavřené na doplněk.

sporem podle de Morganových pravidel

$L_1 \cap L_2 = -(-L_1 \cup -L_2)$

bezkontextové jazyky nejsou uzavřené na průnik - SPOR

v ZA nestačí prohodit koncové a nekoncové stavy!

Zrcadlový obraz, zřetězení a iterace

Bezkontextové jazyky jsou uzavřené na zrcadlový obraz, zřetězení, iteraci a pozitivní iteraci.

1) zrcadlový obraz $L^R = \{w^R \mid w \in L\}$

G: $X \to w^R$ (obrátíme pravou stranu pravidel)

ZA: slova do zásobníku dáváme „opačně“

2) zřetězení $L_1 . L_2$

G: $S \to S_1S_2$ (nejprve generujeme první slovo, potom druhé)

ZA: nejprve běží první automat, potom druhý

3) iterace $L^* = \cup_{i \geq 0} L^i$

G: $S' \to SS' \mid \lambda$ (opakovaně spouštíme generování slov z jazyka)

ZA: na konci výpočtu můžeme restartovat + prázdné slovo

4) pozitivní iterace $L^+ = \cup_{i \geq 1} L^i$

G: $S' \to SS' \mid S$ (opakovaně spouštíme generování slov z jazyka)

ZA: na konci výpočtu můžeme restartovat

Substituce a homomorfismus

Substituce σ **převádí slova na jazyky**

$\sigma(\lambda) = \{\lambda\}$, $\sigma(x) =$ jazyk

$\sigma(uv) = \sigma(u).\ \sigma(v)$ $\quad\quad$ $\sigma: X^* \to P(Y^*)$

$\sigma(L) = \cup_{w \in L}\ \sigma(w)$

Třída T jazyků je **uzavřena na substituci**, když:

$\forall a \in X\ \sigma(a) \in T \wedge\ L \in T \Rightarrow \sigma(L) \in T$

Homomorfismus převádí slova na slova

$h(\lambda) = \lambda$, $h(x) =$ slovo

$h(uv) = h(u)\ .\ h(v)$ $\quad\quad$ $h: X^* \to Y^*$

$h(L) = \{h(w) \mid w \in L\}$

Inverzní homomorfismus převádí slova zpět

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

Uzavřenost BKJ na substituci

Bezkontextové jazyky jsou uzavřeny na substituci.

intuitivně: listy v derivačním stromu generují další stromy

formálně:

máme bezkontextový jazyk L_0, tj. gramatiku $G_0 = (V_{N0}, V_{T0}, S_0, P_0)$,

pro každý terminál a_i z V_{T0}, $\sigma(a_i)$ je bezkontextový jazyk - gramatika G_i

předpokládejme, že množiny neterminálů jsou navzájem disjunktní a že žádný terminál není v jiné gramatice neterminálem

definujme $G = (\cup_{i \geq 0} V_{Ni}, \cup_{i \geq 1} V_{Ti}, S_0, P)$, kde $P = \cup_{i \geq 1} P_i \cup P'$ a $P' = P_0$, kde každý terminál a_i je nahrazen S_i

zřejmě: $L(G) = \sigma(L_0)$

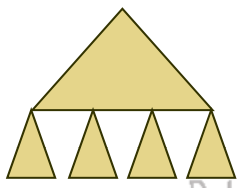
Příklad:

$L_0 = \{a^i b^j \mid 0 \leq i \leq j\}$ $\quad$ $S_0 \to aS_0b \mid S_0b \mid \lambda$

$\sigma(a) = L_1 = \{c^i d^i \mid 0 \leq i\}$ $\quad$ $S_1 \to cS_1d \mid \lambda$

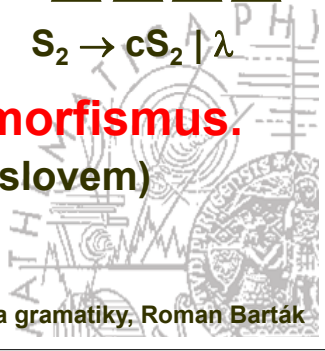
$\sigma(b) = L_2 = \{c^i \mid 0 \leq i\}$ $\quad$ $S_2 \to cS_2 \mid \lambda$

$\sigma(L_0)$: $\quad$ $S_0 \to S_1S_0S_2 \mid S_0S_2 \mid \lambda$, $\quad$ $S_1 \to cS_1d \mid \lambda$, $\quad$ $S_2 \to cS_2 \mid \lambda$



Bezkontextové jazyky jsou uzavřeny na homomorfismus.

přímý důsledek předchozí věty (terminál nahradíme slovem)



Inverzní homomorfismus

Bezkontextové jazyky jsou uzavřeny na inverzní homomorfismus.

$h^{-1}(L) = \{ w \mid h(w) \in L \}$ - máme zásobníkový automat M pro L a čteme w

idea:

- přečteme písmeno x a do vnitřního bufferu dáme h(x)
- simulujeme výpočet M, kdy vstup bereme z bufferu
- po vyprázdnění bufferu načteme další písmeno ze vstupu
- slovo je přijato, když je buffer prázdný a M je v koncovém stavu

formálně:

buffer je konečný, můžeme ho tedy modelovat ve stavu

pro L máme $M = (Q,X,Y,\delta,q_0,Z_0,F)$ (přijímání koncovým stavem)

$h: A^* \to X^*$

definujme $M' = (Q', A, Y, \delta', [q_0,\lambda], Z_0, F\times\{\lambda\})$, kde

$Q' = \{[q,u] \mid q\in Q, u\in X^*, \exists a\in A\ \exists v\in X^*\ h(a)=vu\}$, u je buffer

$\delta'([q,u], \lambda, Z) = \{ ([p,u],\gamma) \mid (p,\gamma) \in \delta(q, \lambda, Z) \}$

$\cup \{ ([p,v],\gamma) \mid (p,\gamma) \in \delta(q, b, Z) \}$ … u=bv (čte buffer)

$\delta'([q, \lambda], a, Z) = \{ ([q,h(a)], Z) \}$ … naplňuje buffer

Kvocienty s regulárním jazykem

Bezkontextové jazyky jsou uzavřené na levý (pravý) kvocient s regulárním jazykem.

$R\backslash L = \{ w \mid \exists u\in R\ uw\in L\}$, $L/R = \{ u \mid \exists w\in R\ uw\in L\}$

idea:

- ZA běží na prázdno (nečte vstup) paralelně s KA
- je-li KA v koncovém stavu, můžeme začít číst vstup

formálně:

konečný automat $A_1 = (Q_1,X, \delta_1,q_1,F_1)$

zásobníkový automat $M_2 = (Q_2,X,Y,\delta_2,q_2,Z_0,F_2)$ (přijímání koncovým stavem)

definujme nový automat $M = (Q', X, Y, \delta, (q_1,q_2), Z_0, F_2)$, kde

$Q' = (Q_1\times Q_2) \cup Q_2$ (dvojice stavů pro paralelní běh ZA a KA)

$\delta'((p,q), \lambda, Z) = \{ ((p',q'),u) \mid \exists a\in X\ p'\in\delta_1(p, a) \wedge (q',u)\in\delta_2(q, a, Z)\}$

$\cup \{ ((p,q'),u) \mid (q',u) \in \delta_2(q, \lambda, Z) \}$

$\cup \{ (q,Z) \mid p\in F_1\}$

$\delta'(q, a, Z) = \delta_2(q, a, Z)$ $a\in X\cup\{\lambda\}$, $q\in Q_2$

zřejmě $L(M) = L(A_1)\backslash L(M_2)$

Uzávěrové vlastnosti deterministických BKJ

Rozumné programovací jazyky jsou deterministické BKJ.

Deterministické bezkontextové jazyky:

- nejsou uzavřené na průnik,
- jsou uzavřené na průnik s regulárním jazykem,
- jsou uzavřené na inverzní homomorfismus.

Doplněk deterministického BKJ je opět deterministický BKJ!

„prohodíme koncové a nekoncové stavy“

potíže:

- nemusí přečíst celé vstupní slovo
  - krok není definován (např. vyprázdnění zásobníku)
    snadno ošetříme „podložkou“ na zásobníku
  - cyklus (zásobník roste, zásobník pulsuje)
    odhalíme pomocí čítače
- po přečtení slova prochází koncové a nekoncové stavy
  stačí si pamatovat, zda prošel koncovým stavem

Uzávěrové vlastnosti DBKJ v praxi

DBKJ nejsou uzavřené na sjednocení (BKJ ano)!

Příklad:

$L = \{a^i b^j c^k \mid i \neq j \vee j \neq k \vee i \neq k\}$ je BKJ, ale není DBKJ

sporem: nechť L je DBKJ

potom -L (doplněk) je DBKJ

$-L \cap a^*b^*c^* = \{a^i b^j c^k \mid i=j=k\}$ je DBKJ - SPOR

DBKJ nejsou uzavřené na homomorfismus (BKJ ano)!

Příklad:

$L_1 = \{a^i b^j c^k \mid i=j\}$ je DBKJ

$L_2 = \{a^i b^j c^k \mid j=k\}$ je DBKJ

$0L_1 \cup 1L_2$ je DBKJ, $1L_1 \cup 1L_2$ není DBKJ

položme $h(0) = 1$

$h(x) = x$ pro ostatní symboly

$h(0L_1 \cup 1L_2) = 1L_1 \cup 1L_2$

Uzávěrové vlastnosti v kostce

	RJ	BKJ	DBKJ
Sjednocení	✓	✓	✗
Průnik	✓	✗	✗
Průnik s RJ	✓	✓	✓
Doplněk	✓	✗	✓
Substituce/ homomorfismus	✓	✓	✗
Inverzní homomorfismus	✓	✓	✓

Chomského hierarchie

gramatiky typu 0 (rekurzivně spočetné jazyky $\mathcal{L}_0$)

pravidla v obecné formě

gramatiky typu 1 (kontextové jazyky $\mathcal{L}_1$)

pouze pravidla ve tvaru $\alpha X\beta \rightarrow \alpha w\beta$,
$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$

jedinou výjimkou je pravidlo $S \rightarrow \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové jazyky $\mathcal{L}_2$)

pouze pravidla ve tvaru $X \rightarrow w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

gramatiky typu 3 (regulární/pravé lineární jazyky $\mathcal{L}_3$)

pouze pravidla ve tvaru $X \rightarrow wY$, $X \rightarrow w$, $X,Y \in V_N$, $w \in V_T^*$

Kontextové gramatiky

pouze pravidla ve tvaru $\alpha X\beta \to \alpha w\beta$,

$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$

jedinou výjimkou je pravidlo $S \to \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

Poznámky:

- neterminál X se přepisuje na w pouze v kontextu α a β
- pravidlo $S \to \lambda$ slouží pouze pro přidání λ do jazyka

Příklad:

$L = \{a^n b^n c^n \mid n \geq 1\}$ je kontextový jazyk (není BKJ)

$S \to aSBC \mid abC$

$CB \to BC$ pozor, není kontextové pravidlo!

$bB \to bb$

$bC \to bc$

$cC \to cc$

Separované gramatiky

Gramatika je **separovaná**, pokud obsahuje pouze pravidla tvaru $\alpha \to \beta$, kde:
buď $\alpha,\beta \in V^+_N$ (neprázdné posloupnosti neterminálů)
nebo $\alpha \in V_N$ a $\beta \in V_T \cup \{\lambda\}$.

Lemma: **Ke každé gramatice G lze sestrojit ekvivalentní separovanou gramatiku G‘.**

Důkaz:

nechť $G = (V_N, V_T, S, P)$

pro každý terminál $x \in V_T$ zavedeme nový neterminál X‘

v pravidlech z P nahradíme terminály odpovídajícími neterminály a přidáme pravidla $X‘ \to x$

$G‘ = (V_N \cup V‘_T, V_T, S, P‘ \cup \{X‘ \to x \mid x \in V_T\})$

zřejmě $L(G) = L(G’)$

Od monotonie ke kontextovosti

Gramatika je monotónní (nevypouštějící), jestliže pro každé pravidlo $(u \to v) \in P$ platí $|u| \leq |v|$.

Monotónní gramatiky slovo v průběhu generování nezkracují.

***Věta:* Ke každé monotónní gramatice lze nalézt ekvivalentní gramatika kontextovou.**

Důkaz:

nejprve převedeme gramatiku na separovanou
tím se monotonie neporuší (+ pravidla $X \to x$ jsou kontextová)
zbývají pravidla $A_1 \ldots A_m \to B_1 \ldots B_n$ (kde $m \leq n$)
převedeme na kontextová pravidla s novými neterminály C

$A_1 A_2 \ldots A_m \to C_1 A_2 \ldots A_m$
$C_1 A_2 \ldots A_m \to C_1 C_2 \ldots A_m$ …
$C_1 \ldots C_{m-1} A_m \to C_1 \ldots C_{m-1} C_m$
$C_1 \ldots C_m \to B_1 \ldots C_m$
$B_1 C_2 \ldots C_m \to B_1 B_2 \ldots C_m$ …
$B_1 \ldots B_{m-1} C_m \to B_1 \ldots B_{m-1} B_m \ldots B_n$

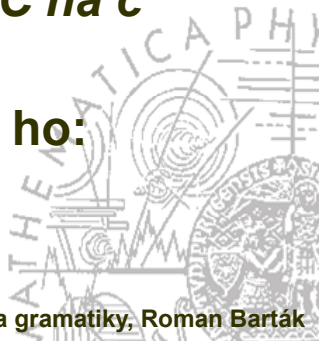
Příklad kontextového jazyka

$L = \{a^i b^j c^k \mid 1 \leq i \leq j \leq k\}$ je kontextový jazyk (není BKJ)

$S \to aSBC \mid aBC$	*generování symbolů a*
$B \to BBC$	*množení symbolů B*
$C \to CC$	*množení symbolů C*
$CB \to BC$	*uspořádání symbolů B a C*
$aB \to ab$	*začátek přepisu B na b*
$bB \to bb$	*pokračování přepisu B na b*
$bC \to bc$	*začátek přepisu C na c*
$cC \to cc$	*pokračování přepisu C na c*

$CB \to BC$ není kontextové pravidlo, nahradíme ho:

$CB \to XB, XB \to XY, XY \to BY, BY \to BC$



Chomského hierarchie

gramatiky typu 0 (rekurzivně spočetné jazyky $\mathcal{L}_0$)

pravidla v obecné formě

gramatiky typu 1 (kontextové jazyky $\mathcal{L}_1$)

pouze pravidla ve tvaru $\alpha X\beta \to \alpha w\beta$,
$X \in V_N$, $\alpha,\beta \in (V_N \cup V_T)^*$, $w \in (V_N \cup V_T)^+$

jedinou výjimkou je pravidlo $S \to \lambda$, potom se ale S nevyskytuje na pravé straně žádného pravidla

gramatiky typu 2 (bezkontextové jazyky $\mathcal{L}_2$)

pouze pravidla ve tvaru $X \to w$, $X \in V_N$, $w \in (V_N \cup V_T)^*$

gramatiky typu 3 (regulární/pravé lineární jazyky $\mathcal{L}_3$)

pouze pravidla ve tvaru $X \to wY$, $X \to w$, $X,Y \in V_N$, $w \in V_T^*$

Turingovy stroje - historie a motivace

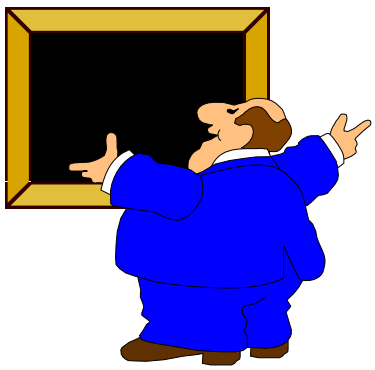
1931 - 1936 pokusy o formalizaci pojmu algoritmu

Gödel, Kleene, Church, Turing

Turingův stroj

zachycení práce matematika

- (nekonečná) tabule
  - lze z ní číst a lze na ni psát
- mozek (řídící jednotka)



Formalizace TS:

- místo tabule oboustranně nekonečná páska
- místo křídy čtecí a zapisovací hlava, kterou lze posouvat
- místo mozku konečná řídící jednotka (jako u ZA)

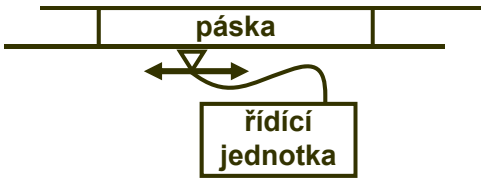
Další formalizace:

- λ-kalkul, částečně rekurzivní funkce, RAM

Definice Turingova stroje

Turingovým strojem nazýváme pětici $T=(Q,X,\delta,q_0,F)$, kde

- Q - neprázdná konečná množina stavů
- X - neprázdná konečná množina symbolů
  - obsahuje symbol ε pro prázdné políčko
- δ - přechodová funkce $\delta : (Q-F)\times X \rightarrow Q\times X\times \{-1,0,1\}$
  - popisuje změnu stavu, zápis na pásku a posun hlavy
- $q_0 \in Q$ - počáteční stav
- $F \subseteq Q$ - množina koncových stavů



1) výpočet začíná ve stavu q_0
2) v každém taktu dojde
   - ke změně stavu
   - k přepisu políčka na pásce
   - k posunu hlavy
3) výpočet končí, když není definována žádná instrukce
(speciálně platí pro koncové stavy)

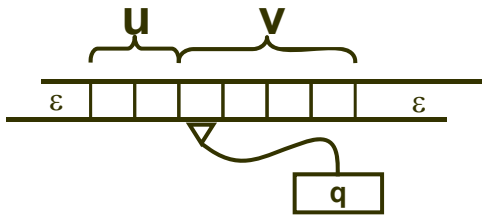
Turingovy stroje - konfigurace a modifikace

Konfigurace Turingova stroje je souhrn údajů přesně popisující stav výpočtu. Obsahuje:

- nejmenší souvislou část pásky, která obsahuje
  - všechny neprázdné buňky
  - čtenou buňku
- vnitřní stav
- polohu čtené buňky (hlavy)

TS postupně přepracovává konfigurace.

obvyklý zápis: uqv



Modifikace Turingova stroje:

- více pásek, více hlav
- jednostranná páska
- omezené činnosti v taktu
- omezený počet stavů, omezená abeceda
- dva zásobníky

Příklad Turingova stroje

Navrhněte Turingův stroj převádějící konfiguraci q_0w na q_Fw^R, kde $w \in \{a_1,\ldots,a_n\}^*$ (tj. obrácení slova).

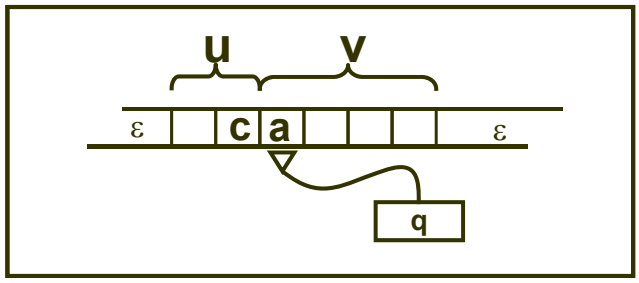
$q_0,\varepsilon \to q_F,\varepsilon,0$	prázdné slovo
$q_0,a_i \to q_{i,r},a'_i,+1$	přečte písmeno, pamatuje si ve stavu
$q_0,a'_i \to q_R,a'_i,+1$	konec (slovo sudé délky)
$q_{i,r},a_j \to q_{i,r},a_j,+1$	běží doprava
$q_{i,r},\varepsilon \to q_{i,w},\varepsilon,-1$	na konci se otočí
$q_{i,r},a'_j \to q_{i,w},a'_j,-1$	"
$q_{i,w},a_j \to q_{j,l},a'_i,-1$	vymění písmena
$q_{i,w},a'_i \to q_R,a'_i,+1$	konec (slovo liché délky)
$q_{i,l},a_j \to q_{i,l},a_j,-1$	a běží zpět (doleva)
$q_{i,l},a'_j \to q_0,a'_i,+1$	na zarážce uloží písmeno a začne znova
$q_R,a'_j \to q_R,a'_j,+1$	běží doprava
$q_R,\varepsilon \to q_C,\varepsilon,-1$	na konci se otočí
$q_C,a'_j \to q_C,a_j,-1$	při běhu doleva ruší označení
$q_C,\varepsilon \to q_F,\varepsilon,+1$	slovo je obráceno

Výpočet Turingova stroje

Turingovým strojem nazýváme pětici $T=(Q,X,\delta,q_0,F)$

- prázdné políčko ε

Konfigurace TS popisuje aktuální stav výpočtu - uqv.



Krok výpočtu (přímá změna konfigurace): uqv |— wpz

$v=av'$, $w=u$, $z=bv'$	$q,a \to p,b,0$
$v=av'$, $w=ub$, $z=v'$	$q,a \to p,b,+1$
$v=av'$, $u=wc$, $z=cbv'$	$q,a \to p,b,-1$

Poznámky:

- technicky je potřeba ošetřit případy, kdy $v=\lambda$ nebo $u=\lambda$
- s *u* a *v* lze pracovat jako se dvěma zásobníky

Výpočet je posloupnost přímých kroků uqv |—* wpz

Turingovy stroje a jazyky

Slovo w je přijímáno Turingovým strojem T, pokud

$q_0 w \vdash^* upv, p \in F$

někdy je na konci výpočtu vyžadováno smazání pásky ($q_0 w \vdash^* \lambda p \varepsilon$)

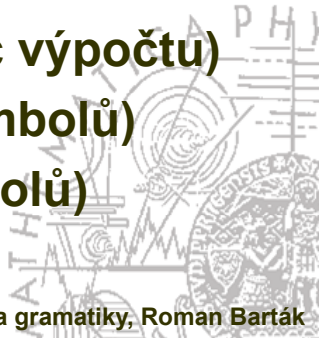
Jazyk přijímaný Turingovým strojem T

$L(T) = \{w \mid w \in (X-\{\varepsilon\})^* \wedge q_0 w \vdash^* upv, p \in F\}.$

Jazyk L nazveme rekurzivně spočetným, pokud je přijímán nějakým Turingovým strojem T (L=L(T)).

***Příklad*: $\{a^{2^n}\}$**

$q_0,\varepsilon \rightarrow q_F,\varepsilon,0$	prázdné slovo (konec výpočtu)
$q_0,a \rightarrow q_1,a,+1$	zvětší čítač (2k+1 symbolů)
$q_1,a \rightarrow q_0,a,+1$	nuluje čítač (2k symbolů)



Od Turingova stroje ke gramatice

Každý rekurzivně spočetný jazyk je typu 0.

Důkaz:

pro Turingův stroj T najdeme gramatiku G tak, že L(T)=L(G)

- **gramatika nejdříve vygeneruje pásku stroje + kopii slova**
- **potom simuluje výpočet (stavy jsou součástí slova)**
- **v koncovém stavu smažeme pásku, necháme pouze kopii slova**

$w\ \underline{\varepsilon}^n\ \underline{w}^R\ q_0\ \underline{\varepsilon}^n$ ($\underline{\varepsilon}^n$ představují volný prostor pro výpočet)

I) $S \rightarrow D\ Q_0\ E$

$D \rightarrow x\ D\ \underline{X} \mid E$ generuje slovo a jeho reverzní kopii pro výpočet

$E \rightarrow \underline{\varepsilon}\ E \mid \underline{\varepsilon}$ generuje volný prostor pro výpočet

II) $\underline{X}\ P\ \underline{Y} \rightarrow \underline{X}'\ Q\ \underline{Y}$ pro $\delta(p,x)=(q,x',0)$

$\underline{X}\ P\ \underline{Y} \rightarrow Q\ \underline{X}'\ \underline{Y}$ pro $\delta(p,x)=(q,x',+1)$

$\underline{X}\ P\ \underline{Y} \rightarrow \underline{X}'\ \underline{Y}\ Q$ pro $\delta(p,x)=(q,x',-1)$

III) $P \rightarrow C$ pro $p \in F$

$C\ \underline{A} \rightarrow C$ mazání pásky

$\underline{A}\ C \rightarrow C$ mazání pásky

$C \rightarrow \lambda$ konec výpočtu

Od Turingova stroje ke gramatice - pokračování

Ještě L(T) = L(G)?

w∈*L(T)*

existuje konečný výpočet stroje T (konečný prostor)

gramatika vygeneruje dostatečně velký prostor pro výpočet

simulujeme výpočet a smažeme dvojníky

w∈*L(G)*

pravidla v derivaci nemusí být v pořadí, jakém chceme

derivaci můžeme přeuspořádat tak, že pořadí je I, II, III

podtržené symboly smazány, tj. vygenerován koncový stav

Příklad:

$\delta(q_0,\varepsilon) = (q_F,\varepsilon,0)$

$\delta(q_0,a) = (q_1,a,+1)$

$\delta(q_1,a) = (q_0,a,+1)$

Gramatika po zjednodušení

$S \to D\ q_0$

$D \to a\ D\ \underline{a} \mid \underline{\varepsilon}$

$\underline{\varepsilon}\ q_0 \to C$

$\underline{a}\ q_0 \to q_1\ \underline{a}$

$\underline{a}\ q_1 \to q_0\ \underline{a}$

$C\ \underline{a} \to C$

$C \to \lambda$

Od gramatik k Turingově stroji

Každý jazyk typu 0 je rekurzivně spočetný.

Důkaz **(neformálně):**

idea: Turingův stroj postupně generuje všechny derivace

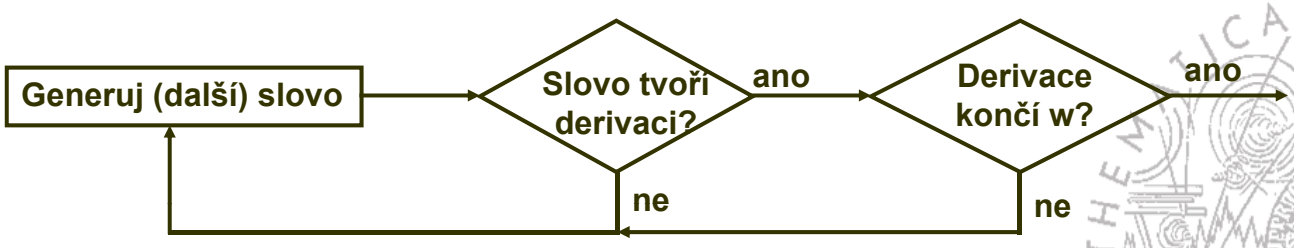
derivaci $S \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = w$ kódujeme jako slovo $\#S\#w_1\#\ldots\#w\#$

TS postupně generuje všechna slova $\#S\#\ w_1\#\ldots\#w_k\#$

pokud $w_n = w$, výpočet končí

jinak, TS generuje další derivaci

- umíme udělat TS, který přijímá slova #u#v#, kde $u \Rightarrow v$
- umíme udělat TS, který přijímá slova $\#w_1\#\ldots\#w_k\#$, kde $w_1 \Rightarrow^* w_k$
- umíme udělat TS postupně generující všechna slova
- stroje spojíme do „while“ cyklu



Nedeterministické Turingovy stroje

Nedeterministickým Turingovým strojem nazýváme pětici $T=(Q,X,\delta,q_0,F)$, kde Q,X,q_0,F jsou jako u TS a $\delta : (Q-F)\times X \to P(Q\times X\times \{-1,0,1\})$.

Slovo w je přijímáno nedeterministickým Turingovým strojem T, pokud existuje nějaký výpočet $q_0w \vdash^* upv$, $p\in F$.

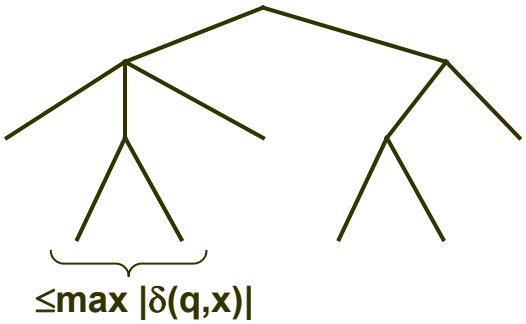
Tvrzení: **N Turingovy stroje přijímají právě rekurzivně spočetné jazyky.**

Důkaz (neformálně):

Ukážeme, že výpočty NTS lze modelovat pomocí TS.

Pozor! Nelze použít podmnožinovou konstrukci (kvůli pásce)!

TS modeluje všechny výpočty NTS prohledáváním do šířky



- Na pásce můžeme mít všechny konfigurace v hloubce *k* (páska je nekonečná), nebo
- můžeme generovat „popis“ výpočtu (posloupnost pravidel) a vždy k němu dopočítat výslednou konfiguraci

Lineárně omezené automaty

Ještě potřebujeme **ekvivalent pro kontextové gramatiky.**

Připomeňme, že kontextovou gramatiku dostaneme z libovolné monotónní gramatiky

Lineárně omezený automat (LOA) je nedeterministický TS, kde na pásce je označen levý a pravý konec (l, r). Tyto symboly nelze při výpočtu přepsat a nesmí se jít nalevo od l a napravo od r.

Slovo w je přijímáno lineárně omezeným automatem, pokud $q_0lwr \vdash^* upv$, $p\in F$.

Prostor výpočtu je definován vstupním slovem a automat při jeho přijímání nesmí překročit jeho délku

u monotónních (kontextových) derivací to není problém:

žádné slovo v derivaci není delší než výstupní slovo

Od kontextových jazyků k LOA

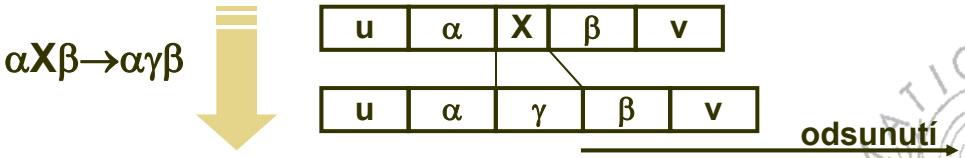
Každý kontextový jazyk lze přijímat pomocí LOA.

Důkaz:

derivaci gramatiky budeme simulovat pomocí LOA

použijeme pásku se dvěma stopami (větší abeceda)

1) slovo w dáme do horní stopy a na začátek dolní stopy dáme S
2) přepisujeme slovo ve druhé stopě podle pravidel G
   2.1) nedeterministicky vybereme část k přepsání
   2.2) provedeme přepsání dle pravidla (pravá část se odsune)



3) pokud jsou ve druhé stopě samé terminály, porovnáme ji s první stopou (slovo přijmeme či zamítneme)

Od LOA ke kontextovým jazykům

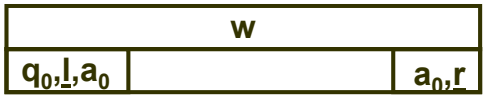
LOA přijímají pouze kontextové jazyky.

***Důkaz*:**

potřebujeme převést LOA na monotónní gramatiku
tj. gramatika nesmí generovat nic navíc!

výpočet ukryjeme do „dvoustopých“ neterminálů

1) generuj slovo ve tvaru $(a_0,[q_0,\underline{l},a_0]),(a_1,a_1),\ldots,(a_n,[a_n,\underline{r}])$
   stav a okraje musíme ukrýt to neterminálů



2) simuluj práci LOA ve „druhé“ stopě (stejně jako u TS)
3) pokud je stav koncový, smaž „druhou“ stopu

speciálně je potřeba ošetřit přijímání prázdného slova
pokud LOA přijímá λ, přidáme speciální startovací pravidlo

Rekurzivní jazyky

Co se stane, když TS nepřijímá nějaké slovo?

a) výpočet skončí v nekoncovém stavu

b) výpočet nikdy neskončí

protože výpočet neskončil, nevíme, zda slovo do jazyka patří

Říkáme, že TS T rozhoduje jazyk L, pokud L=L(T) a pro každé slovo w je výpočet stroje nad w konečný.

Jazyky rozhodnutelné TS nazýváme rekurzivní jazyky.

Věta (Postova): Jazyk L je rekurzivní, právě když L a doplněk L jsou rekurzivně spočetné.

Důkaz:

máme Turingovy stroje T_1 pro L a T_2 pro -L

pro dané slovo w naráz simulujeme výpočet T_1 i T_2

T_1 a T_2 rozpoznávají komplementární jazyky, tedy po konečném počtu kroků víme zda $w \in L$

Problém zastavení TS

Existuje rekurzivně spočetný jazyk, který není rekurzivní?

ANO

Problém zastavení Turingova stroje (halting problem) je algoritmicky nerozhodnutelný.
Neexistuje algoritmus, který by pro daný kód TS a daný vstup rozhodl, zda se TS zastaví.

Důkaz (neformálně):

- vychází z existence univerzálního TS (Turingův stroj, který simuluje výpočet jiného TS nad daným vstupem)
  U(T,X) = T(X) T je kód stroje, X jsou vstupní data
- můžeme udělat stroj P(X), který se na datech X zastaví právě když U(X,X) se nezastaví
- U(P,P) vede ke sporu: $P(P)\downarrow \Leftrightarrow U(P,P)\uparrow \Leftrightarrow P(P)\uparrow$ (diagonální metoda)

Postův korespondenční problém

Postovým korespondenčním problémem (PKP) nazýváme konečný seznam dvojic neprázdných slov $[u_1,v_1],\ldots,[u_n,v_n]$.

Říkáme, že **Postův korespondenční problém má řešení**, pokud existují indexy $i_1,\ldots i_k$ tak, že $1\leq i_j \leq n$ a $u_{i_1}u_{i_2}\ldots u_{i_k} = v_{i_1}v_{i_2}\ldots v_{i_k}$

Říkáme, že **Postův korespondenční problém má iniciální řešení**, pokud existují indexy $i_1,\ldots i_k$ tak, že $1\leq i_j \leq n$ a $u_1u_{i_1}u_{i_2}\ldots u_{i_k} = v_1v_{i_1}v_{i_2}\ldots v_{i_k}$

Věta: **PKP je algoritmicky rozhodnutelný, právě když je algoritmicky rozhodnutelné zda PKP má iniciální řešení.**

Důkaz:

PKP s iniciálním řešením ⇒ PKP (stačí vyzkoušet všechny začátky)

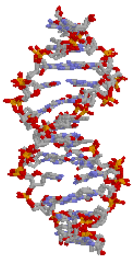
PKP ⇒ PKP s iniciálním řešením

značení $\underline{a_1a_2\ldots a_n}^\bullet = a_1\bullet\, a_2\bullet \ldots a_n\bullet$ $^\bullet\underline{a_1a_2\ldots a_n} = \bullet\, a_1\bullet\, a_2\bullet \ldots a_n$

$x_1 = \bullet\,\underline{u_1}^\bullet$, $x_{j+1} = \underline{u_j}^\bullet$, $x_{n+2} = \blacklozenge$

$y_1 = {}^\bullet\underline{v_1}$, $y_{j+1} = {}^\bullet\underline{v_j}$, $y_{n+2} = \bullet\,\blacklozenge$

PKP s u,v má iniciální řešení právě když PKP s x,y má řešení



Algoritmická nerozhodnutelnost PKP

Existence iniciálního řešení PKP není algoritmicky rozhodnutelná.

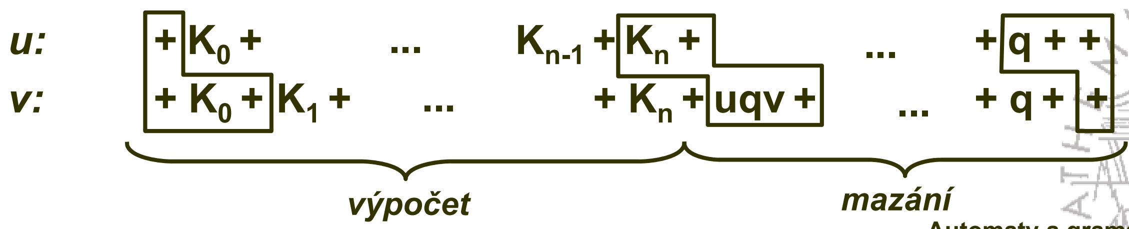
Důkaz:

výpočet TS pro slovo w převedeme na PKP

u	v	
+	$+\varepsilon q_0 w+$	
x	x	$x\in X$
+	+	
px	qy	$\delta(p,x)=(q,y,0)$
p+	qy+	$\delta(p,\varepsilon)=(q,y,0)$
px	yq	$\delta(p,x)=(q,y,+1)$
p+	yq+	$\delta(p,\varepsilon)=(q,y,+1)$
zpx	qzy	$\delta(p,x)=(q,y,-1)$
zp+	qzy+	$\delta(p,\varepsilon)=(q,y,-1)$
+px	$+q\varepsilon y$	$\delta(p,x)=(q,y,-1)$

u	v	
xqy	q	$q\in F$
xq+	q+	
+qy	+q	
q++	+	

PKP má iniciální řešení ⇔ TS se zastaví nad w



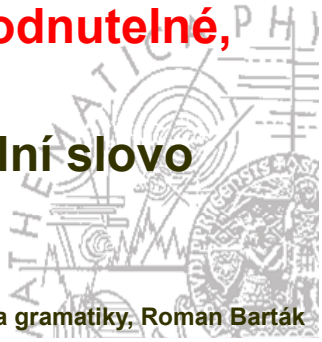
Algoritmická rozhodnutelnost u BKJ

Pro bezkontextové jazyky je algoritmicky rozhodnutelné, zda dané slovo patří či nepatří do jazyka.

- umíme $\lambda \in L(G)$ ($S \Rightarrow^* \lambda$)
- pro ostatní slova použijeme ChNF ($X \to YZ$, $X \to a$)
  - v každé derivaci se délka slova zvětšuje nebo roste počet terminálních symbolů (tj. v derivaci není cyklus!)
  - na terminální slovo délky n použijeme právě (2n-1) pravidel
  - derivací pro všechna slova délky n je konečně
  - můžeme postupně vyzkoušet všechny derivace vedoucí ke slovům dané délky, například prohledáváním do hloubky

Pro bezkontextové jazyky je algoritmicky rozhodnutelné, zda je jazyk prázdný.

umíme zjistit, zda z S lze generovat terminální slovo (algoritmus redukce)



Algoritmicky nerozhodnutelné problémy

Pro bezkontextové gramatiky G_1, G_2 je algoritmicky nerozhodnutelné zda $L(G_1) \cap L(G_2) = \varnothing$.

Důkaz: převedeme PKP na daný problém

máme PKP $[u_1,v_1],\ldots,[u_n,v_n]$

zvolíme nové terminály $a_1,\ldots,a_n$ pro kódy indexů

G_1: $S \to u_i S a_i \mid u_i a_i$ generuje slova $u_{i_1} \ldots u_{i_k} a_{i_k} \ldots a_{i_1}$

G_2: $S \to v_i S a_i \mid v_i a_i$ generuje slova $v_{i_1} \ldots v_{i_k} a_{i_k} \ldots a_{i_1}$

PKP má řešení právě když $L(G_1) \cap L(G_2) \neq \varnothing$

u-část = v-část + složky a_i zajišťují stejné pořadí

Je algoritmicky nerozhodnutelné, zda je bezkontextová gramatika víceznačná.

Důkaz:

$S \to S_1 \mid S_2$

$S_1 \to u_i S_1 a_i \mid u_i a_i$

$S_2 \to v_i S_2 a_i \mid v_i a_i$

PKP má řešení právě když je gramatika víceznačná

Další algoritmicky nerozhodnutelné problémy

Je algoritmicky nerozhodnutelné, zda $L(G)=X^*$ pro BKG G.

Důkaz:

G_1: $S \to u_i S a_i \mid u_i a_i$ generuje slova $u_{i_1} \ldots u_{i_k} a_{i_k} \ldots a_{i_1}$

G_2: $S \to v_i S a_i \mid v_i a_i$ generuje slova $v_{i_1} \ldots v_{i_k} a_{i_k} \ldots a_{i_1}$

jazyky $L(G_1)$, $L(G_2)$ jsou deterministické, tedy $-L(G_1)$ a $-L(G_2)$ jsou deterministické BKJ a $-L(G_1) \cup -L(G_2)$ je BKJ

máme BKG G takovou, že $L(G) = -L(G_1) \cup -L(G_2)$

PKP má řešení $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G) = -L(G_1) \cup -L(G_2) \neq X^*$

Poznámka: **$L(G) = \emptyset$ je algoritmicky rozhodnutelné.**

Důsledky: **Nelze algoritmicky rozhodnout**, zda

$L(G) = R$, pro BKG G a regulární jazyk R (důkaz: za R zvolme X^*)

$R \subseteq L(G)$, pro BKG G a regulární jazyk R (důkaz: za R zvolme X^*)

$L(G_1) = L(G_2)$, pro BKG G_1 a G_2 (důkaz: nechť G_1 generuje X^*)

$L(G_1) \subseteq L(G_2)$, pro BKG G_1 a G_2 (důkaz: nechť G_1 generuje X^*)

Poznámka: **$L(G) \subseteq R$ je algoritmicky rozhodnutelné**

$L(G) \subseteq R \Leftrightarrow L(G) \cap -R = \emptyset$ + $(L(G) \cap -R)$ je BKJ

Shrnutí

popis nekonečných objektů konečnými prostředky

regulární jazyky

- konečné automaty (NKA,2KA)
- Nerode, Kleene, pumpování

bezkontextové jazyky

- zásobníkové automaty (DZA)
- Dyckovy jazyky, pumpování

kontextové jazyky

- lineárně omezené automaty
- monotonie

rekurzivně spočetné jazyky

- Turingovy stroje
- algoritmická nerozhodnutelnost

použití nejen pro práci s jazyky!