

Automaty a gramatiky

Roman Barták, KTIML

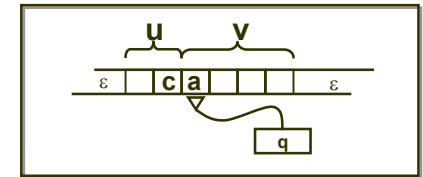
bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Výpočet Turingova stroje

Turingovým strojem nazýváme pěticu $T=(Q,X,\delta,q_0,F)$

– prázdné políčko ε

Konfigurace TS popisuje aktuální stav výpočtu - uqv .



Krok výpočtu (přímá změna konfigurace): $uqv \mapsto wpz$

$v=av', w=u, z=bv' \quad q,a \rightarrow p,b,0$

$v=av', w=ub, z=v' \quad q,a \rightarrow p,b,+1$

$v=av', u=wc, z=cbv' \quad q,a \rightarrow p,b,-1$

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 $uqv \mapsto^* wpz$

Automaty a gramatiky, Roman Barták

Turingovy stroje a jazyky

Slovo w je přijímáno Turingovým strojem T , pokud

$q_0w \mapsto^* upv, p \in F$

někdy je na konci výpočtu vyžadováno smazání pásky ($q_0w \mapsto^* \lambda p \varepsilon$)

Jazyk přijímaný Turingovým strojem T

$L(T) = \{w \mid w \in (X - \{\varepsilon\})^* \text{ \& } q_0w \mapsto^* upv, p \in F\}$.

Jazyk L nazveme **rekurzivně spočítelným**, pokud je přijímán nějakým Turingovým strojem T ($L=L(T)$).

Příklad: $\{a^{2n}\}$

$q_0, \varepsilon \rightarrow q_F, \varepsilon, 0$

$q_0, a \rightarrow q_1, a, +1$

$q_1, a \rightarrow q_0, a, +1$

prázdné slovo (konec výpočtu)

zvětší čítač (2k+1 symbolů)

nuluje čítač (2k symbolů)

Automaty a gramatiky, Roman Barták

Od Turingova stroje ke gramatice

Každý rekurzivně spočítelný 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

Automaty a gramatiky, Roman Barták

Od Turingova stroje ke gramatice - pokračování

Ještě $L(T) = L(G)$?

$w \in L(T)$

existuje konečný výpočet stroje T (konečný prostor)
gramatika vygeneruje dostatečně velký prostor pro výpočet
simulujeme výpočet a smažeme dvojníky

$w \in L(G)$

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

$S \rightarrow D q_0$

$D \rightarrow a D a \mid \varepsilon$

$\varepsilon q_0 \rightarrow C$

$a q_0 \rightarrow q_1 a$

$a q_1 \rightarrow q_0 a$

$C a \rightarrow C$

$C \rightarrow \lambda$

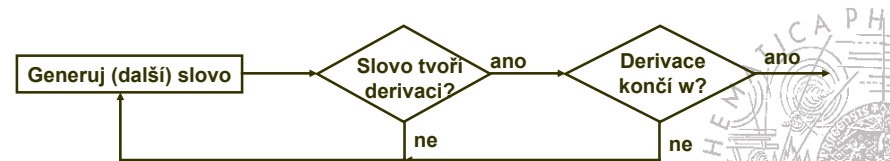
Automaty a gramatiky, Roman Barták

Od gramatik k Turingově stroji

Každý jazyk typu 0 je rekurzivně spočetný.

Důkaz (neformálně):

- idea: Turingův stroj postupně generuje všechny derivace
derivaci $S \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n = w$ kódujeme jako slovo $\#S\#w_1\#\dots\#w\#$
TS postupně generuje všechna slova $\#S\#w_1\#\dots\#w_k\#$
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 $\#w_1\#\dots\#w_k\#$, kde $w_1 \Rightarrow^* w_k$
 - umíme udělat TS postupně generující všechna slova
 - stroje spojíme do „while“ cyklu



Automaty a gramatiky, Roman Barták

Nedeterministické Turingovy stroje

Nedeterministickým Turingovým strojem nazýváme pěticu $T=(Q, X, \delta, q_0, F)$, kde Q, X, q_0, F jsou jako u TS a $\delta : (Q-F) \times X \rightarrow P(Q \times X \times \{-1, 0, 1\})$.

Slovo w je přijímáno nedeterministickým Turingovým strojem T , pokud existuje nějaký výpočet $q_0 w \xrightarrow{*} upv, p \in F$.

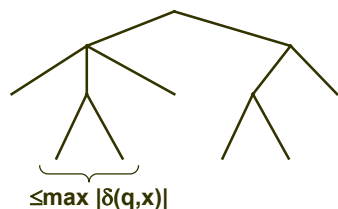
Tvrzení: N Turingovy stroje přijímají právě rekurzivně spočetné jazyky.

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

Automaty a gramatiky, Roman Barták

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 ($\underline{l}, \underline{r}$). Tyto symboly nelze při výpočtu přepsat a nesmí se jít nalevo od \underline{l} a napravo od \underline{r} .

Slovo w je přijímáno lineárně omezeným automatem, pokud $q_0 \underline{l} w \underline{r} \xrightarrow{*} upv, p \in F$.

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

Automaty a gramatiky, Roman Barták

Od kontextových jazyků k LOA

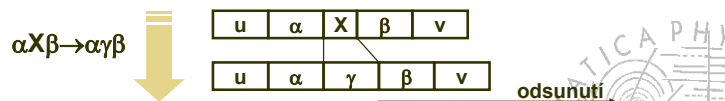
Každý kontextový jazyk lze přijímat pomocí LOA.

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)

Automaty a gramatiky, Roman Barták

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 $(a_0, [q_0, \perp, a_0]), (a_1, a_1), \dots, (a_n, [a_n, \perp])$
stav a okraje musíme ukryt 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

Automaty a gramatiky, Roman Barták

Rekurzivní jazyky

Co se stane, když TS nepřijímá nějaké slovo?

- a) výpočet skončí v nekonečném 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 \bar{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 $w \in L$

Automaty a gramatiky, Roman Barták

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)

Automaty a gramatiky, Roman Barták

Postův korespondenční problém

Postovým korespondenčním problémem (PKP) nazýváme konečný seznam dvojic neprázdných slov $[u_1, v_1], \dots, [u_n, v_n]$.

Říkáme, že **Postův korespondenční problém má řešení**, pokud existují indexy i_1, \dots, i_k tak, že $1 \leq i_j \leq n$ a $u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k}$

Říkáme, že **Postův korespondenční problém má iniciální řešení**, pokud existují indexy i_1, \dots, i_k tak, že $1 \leq i_j \leq n$ a $u_{i_1} u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_1} v_{i_2} \dots v_{i_k}$

Věta: PKP je algoritmicky rozhodnutelný, právě když je algoritmicky rozhodnutelné zda PKP má iniciální řešení.

Důkaz:

PKP s iniciálním řešením \Rightarrow PKP (stačí vyzkoušet všechny začátky)

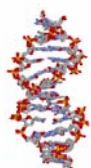
PKP \Rightarrow PKP s iniciálním řešením

značení $a_1 a_2 \dots a_n = a_1 \cdot a_2 \cdot \dots \cdot a_n$ $\cdot a_1 a_2 \dots a_n = \cdot a_1 \cdot a_2 \cdot \dots \cdot a_n$

$x_1 = \cdot a_1$, $x_{j+1} = a_j \cdot$, $x_{n+2} = \cdot$

$y_1 = a_1$, $y_{j+1} = \cdot a_j$, $y_{n+2} = \cdot$

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.

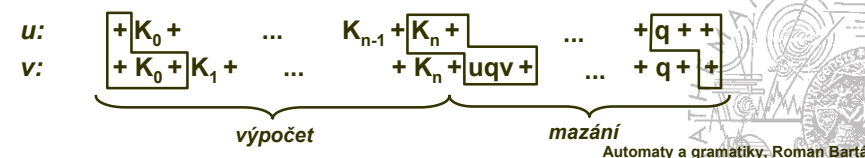
Důkaz:

výpočet TS pro slovo w převedeme na PKP

u	v	
+	$+\epsilon q_0 w +$	
x	x	$x \in X$
+	+	
px	qy	$\delta(p, x) = (q, y, 0)$
p+	qy+	$\delta(p, \epsilon) = (q, y, 0)$
px	yq	$\delta(p, x) = (q, y, +1)$
p+	yq+	$\delta(p, \epsilon) = (q, y, +1)$
zpx	qzy	$\delta(p, x) = (q, y, -1)$
zp+	qzy+	$\delta(p, \epsilon) = (q, y, -1)$
+px	+qey	$\delta(p, x) = (q, y, -1)$

u	v	
xqy	q	$q \in F$
xq+	q+	
+qy	+q	
q++	+	

PKP má iniciální řešení
 \Leftrightarrow 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 ($X \rightarrow YZ, X \rightarrow a$)
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 $L(G_1) \cap L(G_2) = \emptyset$.

Důkaz: převedeme PKP na daný problém

máme PKP $[u_1, v_1], \dots, [u_n, v_n]$

zvolíme nové terminály a_1, \dots, a_n pro kódy indexů

$G_1: S \rightarrow u_i S a_i \mid u_i a_i$ generuje slova $u_{i_1} \dots u_{i_k} a_{i_k} \dots a_{i_1}$

$G_2: S \rightarrow v_i S a_i \mid v_i a_i$ generuje slova $v_{i_1} \dots v_{i_k} a_{i_k} \dots a_{i_1}$

PKP má řešení právě když $L(G_1) \cap L(G_2) \neq \emptyset$

u -část = v -část + složky a_i zajišťují stejné pořadí

Je algoritmicky nerozhodnutelné, zda je bezkontextová gramatika víceznačná.

Důkaz:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow u_i S_1 a_i \mid u_i a_i$

$S_2 \rightarrow v_i S_2 a_i \mid v_i a_i$

PKP má řešení právě když je gramatika víceznačná

Další algoritmicky nerozhodnutelné problémy

Je algoritmicky nerozhodnutelné, zda $L(G)=X^*$ pro BKG G .

Důkaz:

$G_1: S \rightarrow u_i S a_i \mid u_i a_i$ generuje slova $u_{i_1} \dots u_{i_k} a_{i_k} \dots a_{i_1}$

$G_2: S \rightarrow v_i S a_i \mid v_i a_i$ generuje slova $v_{i_1} \dots v_{i_k} a_{i_k} \dots a_{i_1}$

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: necht' G_1 generuje X^*)

$L(G_1) \subseteq L(G_2)$, pro BKG G_1 a G_2 (důkaz: necht' 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

Automaty a gramatiky, Roman Barták

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!



Automaty a gramatiky, Roman Barták