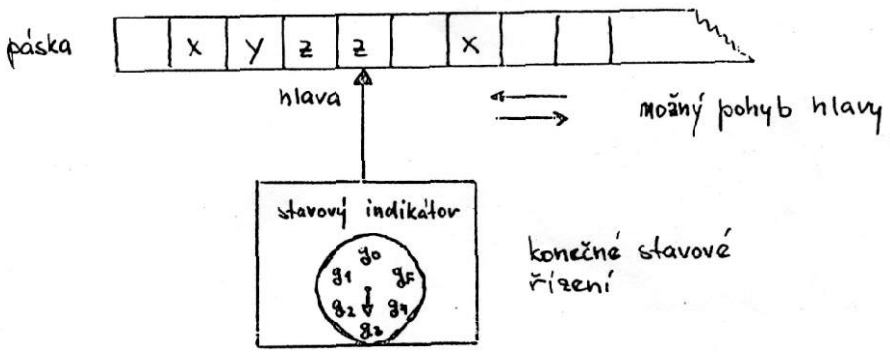
Turingovy stroje a jazyky typu 0

A. Turing 1936 (E. Post 1936)

1. Základní koncepce a definice Turingova stroje



- historie T. stroje
- Turingova téze

Definice

Turingův stroj M je 6-tice tvaru $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde

Q je konečná množina vnitřních stavů

Σ je konečná množina (neblankových) symbolů nazývaná strojová (vstupní) abeceda

Γ je konečná množina, $\Sigma \subseteq \Gamma$, symbolů nazývaná pásková abeceda

δ je zobrazení:

$$\delta: (Q \setminus \{q_F\}) \times \Gamma \to Q \times (\Gamma \cup \{L, R\}),\ L, R \notin \Gamma$$

q_0 je počáteční stav

q_F je koncový stav

Pozn.: $\Delta \in \Gamma$, Δ značí blank $\Delta \notin \Sigma$

Př. zápisu obsahu pásky: $\Delta x y z \underline{z} \Delta x \Delta\Delta \ldots$

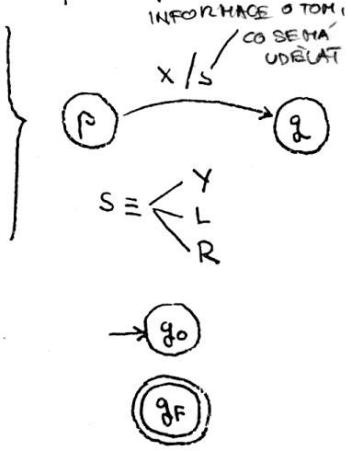
prázdná páska je plná Δ

$\ldots \Delta\Delta\Delta \underbrace{a\ b\ c}_{\Sigma} \Delta\Delta\Delta\Delta \ldots$

Sémantika přechodové funkce δ:

	Význam:
1. $\delta(p,x) = (q,y)$	$p, q \in Q$; $x, y \in \Gamma$ operace zápisu
2. $\delta(p,x) = (q,L)$	operace posuvu hlavy vlevo
3. $\delta(p,x) = (q,R)$	operace posuvu hlavy vpravo
	počáteční stav
	koncový stav

Grafická reprezentace:



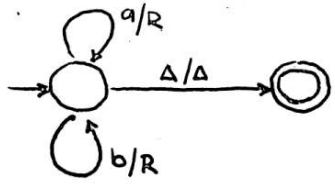
Zastavení T. stroje

a) normální – přechodem do koncového stavu

b) abnormální < pro aktuální stav (p,x) (symbol na pozici hlavy) není δ definována

hlava je na 1. symbolu x pásky, akt. stav je p a současně $\delta(p,x) = (q,L)$

Příklad graf. znázornění T. stroje:



T. S. MUSÍ MÍT ALESPOŇ 2 STAVY NEMOHU SLOUČIT KONCOVÝ A POČÁTEČNÍ

T. stroj, který posune hlavu na první symbol Δ vpravo od akt. pozice čtecí hlavy

např. při obsahu pásky:

$\Delta \underline{a} a b \Delta b a \Delta \ldots$

provede

$\Delta a a b \underline{\Delta} b a \Delta \ldots$

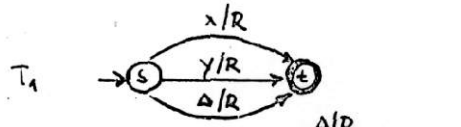
2. Modulární konstrukce T. strojů

Spojování (kombinace) T. strojů

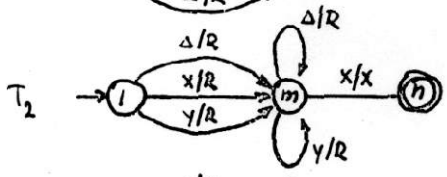
Postupujeme analogicky spojování diagramů přechodů kon. automatů (viz převod reg. výraz → kon. automat)

Budeme uvažovat „sjednocení" a „konkatenaci" diagramů T. strojů:

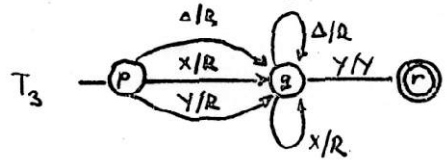
Příklad:



T_1 přesune hlavu o 1 symbol doprava



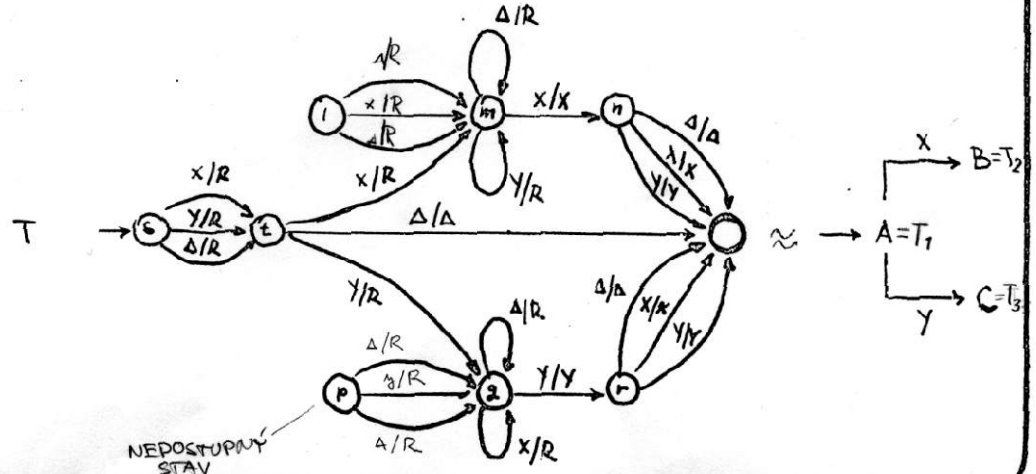
T_2 nalezne 1. výskyt symbolu x vpravo (od akt. pozice hlavy)



T_3 nalezne 1. výskyt symbolu y vpravo

TÉHOŽ

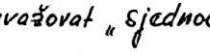
T nalezne 2. výskyt y (neblankového) symbolu vpravo od počáteční pozice hlavy



Konvence pro zjednodušení diagramů

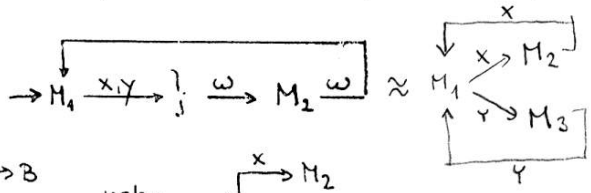
1. Sekvenci strojů $\rightarrow A \rightarrow B \rightarrow C$ budeme zkracovat na $\rightarrow ABC$

2. „Parametrová" konvence:

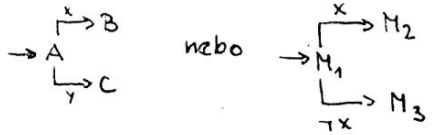


ω nabývá „hodnoty" toho ze symbolu x, y, z, který byl aktuálním symbolem na pásce

Příklad použití:



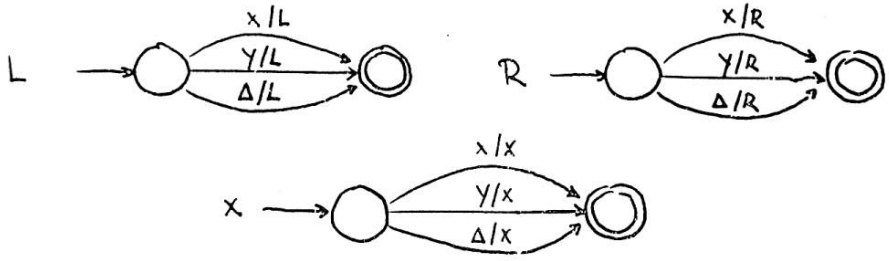
3. „větvení"



(Rozdíl mezi $M_1 \rightarrow M_2$ a $M_1 \xrightarrow{x} M_2$)

Základní stavební bloky

Stroje L, R, x $\quad \Gamma = \{x, y, \Delta\}$

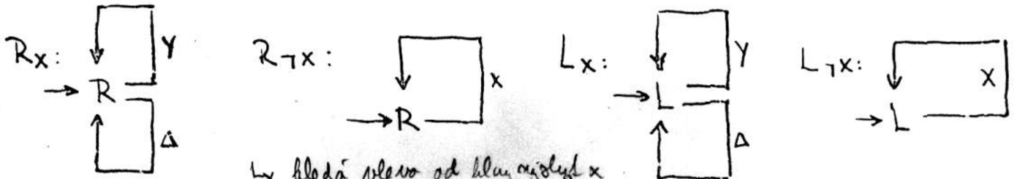


Příklad

$\rightarrow R \rightarrow y \rightarrow L$ nebo zkráceně $\rightarrow RyR$

páska $\quad \Delta x y x y \underline{x} \Delta\Delta \ldots$ „transformuje" na $\Delta x y x y \underline{x} y \Delta \ldots$

Stroje $L_x, L_{\neg x}, R_x, R_{\neg x}$



L_x hledá vlevo od hlavy symbol x
$L_{\neg x}$ hledá vlevo to, co není x
opět přeskakuje, co je na akt. pozici

L posunuje hlavu doleva
R — II — doprava

T. stroje S_R a S_L pro posuv (Shift) obsahu pásky

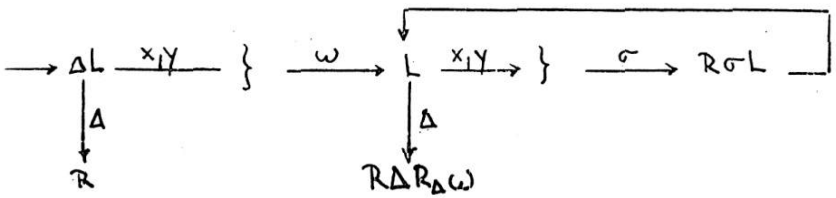
Stroj S_R posune řetěz neblankových symbolů nacházejících se vlevo od aktudl. symbolů o jeden symbol doprava

Stroj S_L – pracuje symetricky

Příklady aktivace těchto strojů:

Δxyy<u>x</u>xΔΔ...	ΔΔxy<u>y</u>xΔΔ...	S_R
Δyxy<u>Δ</u>ΔxxyΔ...	ΔΔyx<u>y</u>ΔxxyΔ...	
xyΔ<u>y</u>xΔΔ...	xyΔ<u>Δ</u>xΔΔ ...	– speciální případ
Δ<u>y</u>yxxΔΔ...	Δ<u>y</u>xxΔΔΔ..	S_L

Realizace stroje S_R: ZAPAMATUJEME SI ω — V TOMTO CYKLU PŘESUNUJEME VŽDY 1 PRVEK DOPRAVA NEŽ PŘIJDE L



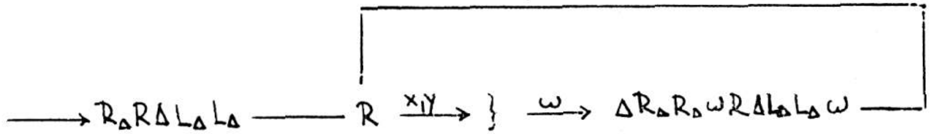
Realizace stroje S_L : analogicky – cvičení MÍSTO R BUDE L

Příklady aplikací

a) kopírovaní řetězce

Stroj provede transformaci

$\underline{\Delta} w \Delta$ na $\Delta w \underline{\Delta} w \Delta$

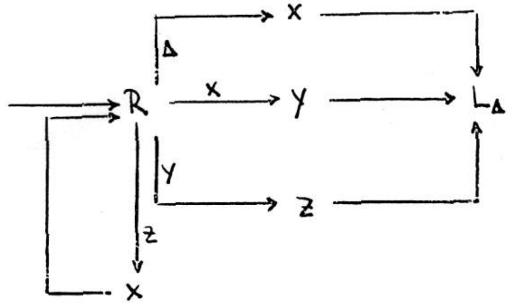


Př:

1. <u>Δ</u>xyΔ (RΔRΔLΔLΔ)
2. <u>Δ</u>xyΔΔ (R) (
3. Δ<u>x</u>yΔΔ (ω = x, Δ)
4. Δ<u>Δ</u>yΔΔ (RΔRΔ)
5. ΔΔyΔ<u>Δ</u> (ωRΔ)
6. ΔΔyΔx<u>Δ</u> (LΔLΔω)
7. Δ<u>x</u>yΔxΔ

⋮

b) generovaní řetězců (*)



Stroj generuje následující řetězec v posloupnosti:

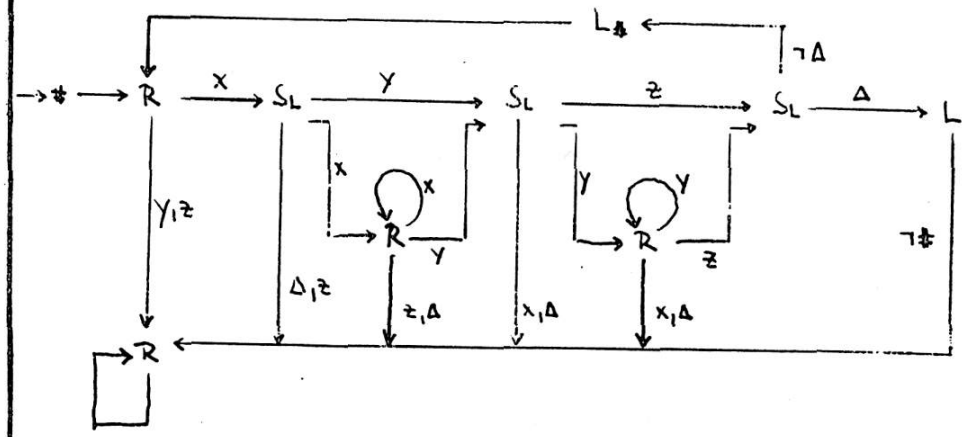
ε, x, y, z, xx, yx, zx, xy, yy, zy, xz, yz, zz, xxx, yxx

3. Turingovy stroje jako akceptory jazyků.

Definice. Řetězec $w \in \Sigma^*$ je *přijat* T. strojem $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, jestliže se při aktivaci M z počáteční konfigurace pásky: $\underline{\Delta} w \Delta\Delta \ldots$ v poč. stavu q_0 stroj M zastaví (přejde do stavu q_F).

Množinu $L(M) = \{ w \mid w$ je přijat T. strojem $M \}$ nazýváme *jazyk přijímaný T. strojem M*.

Příklad



T. stroj M, pro který $L(M) = \{ x^n y^n z^n \mid n \in \mathbb{N} \}$

(M provádí opakovaně redukce

$$x^n y^n z^n \to x^{n-1} y^{n-1} z^n \to x^{n-1} y^{n-1} z^{n-1}$$

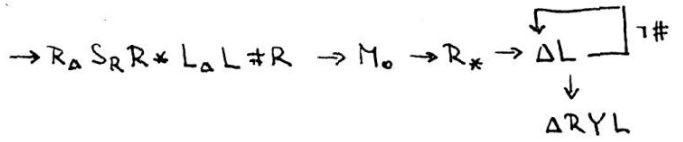
a končí zredukováním vst. řetězce na prázdný řetězec)

$\# \in \Gamma$ - marker označující levý konec pásky

Alternativní způsob definice přijetí řetězce:

$$\underline{\Delta} w \Delta\Delta \quad \to \quad \underline{\Delta} Y \Delta\Delta \qquad Y \in \Gamma$$

Realizace:



M_0 je T. stroj přijímající řetězce 1. způsobem; začíná pracovat s páskou tvaru: $\# \underline{\Delta} w * \Delta\Delta$

Vícepáskové Turingovy stroje

Uvažujme T. stroj, který má k-pásek a k odpovídajících hlav s přechodovou funkcí δ

$$\delta: (Q \setminus \{q_F\}) \times (\Gamma_1 \times \Gamma_2 \times \ldots \times \Gamma_k) \to Q \times (\Gamma_i \cup \{L, R\}) \times \{1, \ldots, k\}$$

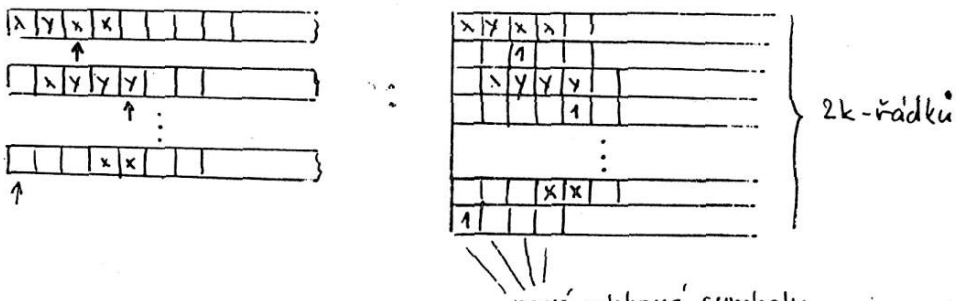
$\Gamma_1, \ldots, \Gamma_k$ - abecedy páskových symbolů, $i \in \{1, \ldots, k\}$

VĚTA: Pro každý k-páskový T. stroj M existuje jednopáskový T. stroj M' takový, že $L(M) = L(M')$.

Důkaz: (základní myšlenka)

Budeme simulovat činnost automatu M ekvivalentním jednopáskovým automatem M'.

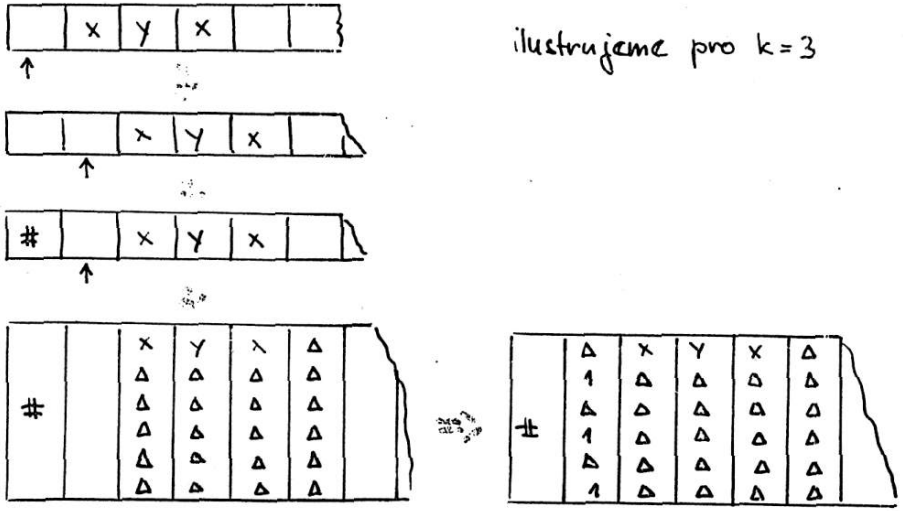
Nejprve reprezentujeme (informaci) o stavu pásek a pozici čtecích hlav 2k-ticemi, které budou tvořit nové páskové symboly automatu M'.



lichý index : obsah pův. pásek
sudý index : pozice hlav

Předpokládáme, že vícepáskový automat M měl přijímaný řetězec umístěn na 1. pásce, obsah ostatních pásek byl blankový, hlavy na 1. pozici

a) automat M' provede nejprve konverzi 1. pásky do 2k-tic reprezentujících počáteční obsah pásky automatu M:

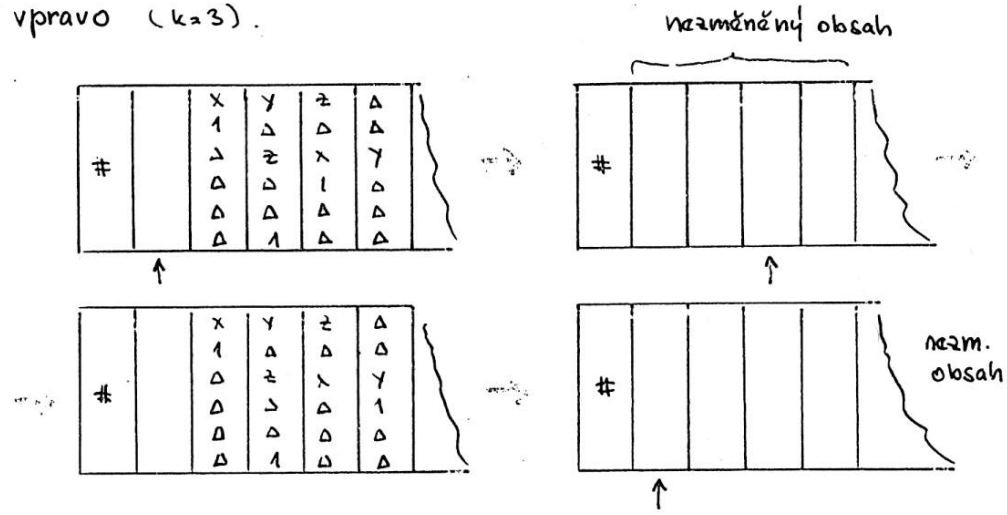


ilustrujeme pro k = 3

b) Nyní konstruujeme přech. funkci automatu M', která pracuje se „složenými" stavy $(p, x_1, x_2, \dots, x_k)$ korespondujícími s konfigurací stroje M

simulaci podle této funkce ilustrujme na příkladě.

Posloupnost přechodů automatu M' ekvivalentní posuvu 2. hlavy vpravo (k=3).



Závěr: Zvětšením paměť. možností T. stroje nerozšíříme jeho schopnosti přijímat jazyky.

Nedeterministické Turingovy stroje

Definice Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ se nazývá nedeterministickým T. strojem, jestliže δ má tvar

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

VĚTA Pro každý nedeterministický T. stroj M existuje deterministický T. stroj D takový, že $L(M) = L(D)$.

Důkaz.

Nedeterministický T. stroj M budeme simulovat deterministickým 3-páskovým T. strojem M' takto:

Účel pásek stroje M':

Páska 1. - bude obsahovat přijímaný vst. řetězec

2. - pracovní páska; obsahuje kopii pásky 1; po indikaci neúspěšné volby přechodů bude její obsah obnovován

3. - obsahuje kódovanou volbu posloupnosti přechodů; při neúspěchu bude nahrazena novou posloupností

Dopl. pozn.: - Řetězec na 2. pásce je ohraničen z obou stran sp. markery

- zvolená posl. přechodů je kódována posloupností čísel přiřazených hranám stroje M; generátor nové posloupnosti lze realizovat T. podstrojem z předchozího příkladu (*).

Algoritmus simulace:

1. Okopíruj obsah 1. pásky na 2. pásku
2. Generuj příští posloupnost přechodů na pásku 3
3. Simuluj tuto posloupnost s využitím pásky 2
4. Vede-li tato posloupnost do konc. stavu stroje M, zastav - vst. řetězec je přijat. V opačném případě smaž obsah pásky 2 a vrať se ke kroku (1).

Závěr:

4. Jazyky přijímané T. stroji a jazyky typu 0

názvosloví: jazyky typu 0 (neomezené jazyky), frázově strukturované jazyky

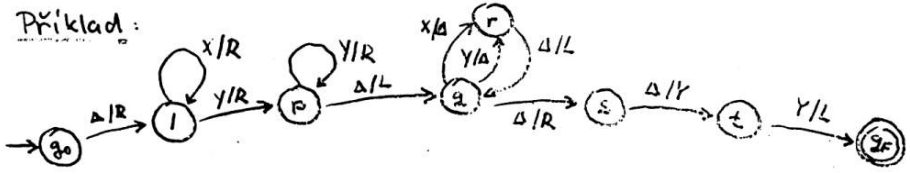
konfigurace T. stroje:

Je dána (1) stavem kon. řízení, (2) obsahem pásky, (3) pozicí hlavy

Konvence: $p, p \in Q \wedge \Delta x y x \Delta \ldots$ zapisujeme jako konfiguraci

$[\Delta x p \overset{\downarrow}{x} \Delta]$ případně $[\Delta x p \overset{\downarrow}{x} \Delta\Delta]$ atd.

Příklad:



Kostra T. stroje přijímajícího jazyk $\{x^m y^n \mid m \geq 0, n > 0\}$

Posloupnost konfigurací při přijetí řetězce xxy:

1. $[q_0 \Delta x x y \Delta]$
2. $[\Delta 1 x x y \Delta]$
3. $[\Delta x 1 x y \Delta]$
4. $[\Delta x x 1 y \Delta]$
5. $[\Delta x x y p \Delta]$
6. $[\Delta x x q y \Delta]$
7. $[\Delta x x r \Delta]$
8. $[\Delta x q x \Delta]$
9. $[\Delta x r \Delta]$
10. $[\Delta q x \Delta]$
11. $[\Delta r \Delta]$
12. $[q \Delta]$
13. $[\Delta s \Delta]$
14. $[\Delta t Y \Delta]$
15. $[q_f \Delta Y \Delta]$

VĚTA Každý jazyk přijímaný T. strojem je jazykem typu 0.

Důkaz:

Sestrojme gramatiku typu 0, která vytváří derivaci odpovídající reverzi posloupnosti konfigurací, která vede k přijetí vst. řetězce.

Nechť $G = (N, \Sigma, P, S)$ je předpokládaná gramatika

(1) $N = \{S\} \cup \{Q\} \cup \{P\} \cup \{\Gamma,]\}$ ($\Gamma \setminus \Sigma$) S bude startovací nonterminál

(2) Pravidla gramatiky:

(a) $S \to [q_F \Delta Y \Delta]$

(b) $\Delta] \to \Delta\Delta]$ umožňuje rozvinout řetězec $[q_F Y \Delta]$ na požadovanou délku

(c) $qy \to px$ jestliže $\delta(p,x) = (q,y)$

(d) $xq \to px$ — " — $\delta(p,x) = (q,R)$

(e) $qyx \to ypx$ pro každý páskový symbol y, jestliže $\delta(p,x) = (q,L)$

(f) $[q_0 \Delta \to e$
$\Delta\Delta] \to \Delta]$
$\Delta] \to e$
} umožňující odstranit symboly $q_0, [,]$ a Δ

$L(M) = L(G)$ t.j.

$w \in L(M) \iff$ ex. derivace $S \Rightarrow [q_F \Delta Y \Delta] \Rightarrow \cdots [q_0 \Delta w \Delta] \Rightarrow w\Delta] \Rightarrow w$

VĚTA Každý jazyk typu 0 je jazykem přijímaným T. strojem

Důkaz:

Nechť $L = L(G)$ je jazyk typu 0. Sestrojíme nedeterministický 2-páskový T. stroj N takový, že $L(G) = L(N)$

Základní akce: simulace použití přepisovacího pravidla tvaru $\alpha \to \beta$

Páska 1. – obsahuje testovaný vst. řetězec
– " – 2. – vytváří se derivace tohoto řetězce substitucemi $\alpha \to \beta$ (na počátku obs. S)

Rozsah jazyků typu 0

VĚTA Pro každou abecedu Σ existuje jazyk nad touto abecedou, který není jazykem typu 0.

Důkaz:

1. Nejdříve pochopíme, že lib. jazyk typu 0 nad abecedou Σ může být přijat T. strojem M, jehož symboly na pásce patří do množiny $\Sigma \cup \{\Delta\}$.

   (Pokud M pracuje s více symboly, pak tyto symboly zakódujeme jako posloupnosti symbolů $x \in \Sigma$ a sestrojíme automat M', s $\Gamma = \Sigma \cup \{\Delta\}$, který simuluje automat M

2. Nyní můžeme systematicky „vypisovat" všechny T. stroje s páskovými symboly z množiny $\Sigma \cup \{\Delta\}$:
   začneme vš. stroji se 2 stavy, pokračujeme vš. stroji se 3 stavy, 4 stavy atd.
   Závěr: množina všech takových strojů a tudíž i jazyků typu 0 je spočetná

3. Množina Σ^* obsahuje nekonečně mnoho řetězců a proto je množina 2^{Σ^*} všech jazyků nespočetná
   (Viz Cantorův důkaz nespočetnosti mn. reálných čísel – diagonalizace)

Budeme předpokládat, že množina všech jazyků nad ab Σ je spočetná.
Pak existuje bijekce f

$$f: N \to 2^{\Sigma^*}$$

Pokusme se tuto bijekci zobrazit nekonečnou maticí takto:
(Předpokl., že $\Sigma = \{x, y, z\}$)

	ε	x	y	z	xx	xy	xz	...		xxx	xxy	...
$f(1)$	a_{11}	a_{12}	a_{13}	...						a_{1i}	a_{1i+1}	...
$f(2)$	a_{21}	a_{22}	a_{23}	...						a_{2i}	a_{2i+1}	...
$f(3)$	a_{31}	a_{32}	a_{33}	...								
⋮												
$f(k)$	a_{k1}	a_{k2}	...			a_{kk}	...			a_{ki}	a_{ki+1}	...
⋮												

Jazyk $L_n = f^{-1}(n)$ je zakódován binárním vektorem (nekonečným) tvaru

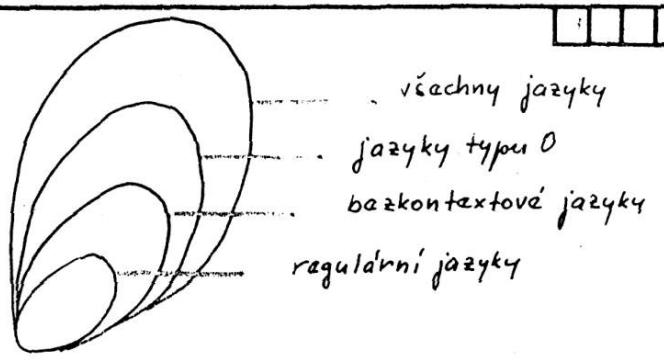
$(a_{n1}, a_{n2}, a_{n3}, \dots)$ kde $a_{nj} = \begin{cases} 0 & \text{pokud } j\text{-tý řetězec neleží v } L_n \\ 1 & \text{pokud } j\text{-tý řetězec leží v } L_n \end{cases}$

Nyní uvažujme jazyk

$\bar{L} = (\bar{a}_1, \bar{a}_2, \bar{a}_3, \dots)$ kde $\bar{a}_i = \begin{cases} 0 & \text{pokud } a_{ii} = 1 \\ 1 & \text{pokud } a_{ii} = 0 \end{cases}$

$\bar{L}$ se liší od každého jazyka $f(n), n \in N$, přitom $\bar{L} \in 2^{\Sigma^*}$ tedy f není surjektivní a není tudíž hledanou bijekcí.

Závěr:



Důsledky Turingovy téze:

1. Jazyky bez gramatického základu nemohou být rozpoznatelné žádným algoritmickým procesem.
2. Mohou existovat systémy pro porozumění přirozenému jazyku?
3. Mohou existovat opravdu inteligentní stroje?
   (Přirozená versus umělá inteligence)

5. Jazyky mimo rozsah frázově strukturovaných jazyků

Kódovací systém pro Turingovy stroje

Zahrnuje kódování:
(1) stavů
(2) symbolů z Γ a symboly L, R
(3) přechodové funkce δ

Výsledkem bude binární řetězec reprezentující celý T. stroj.

kódování stavů: Množinu stavů Q uspořádáme: $Q = \{q_0, q_1, q_2, q_3, r, s \dots t\}$
j-tý stav kódujeme řetězcem 0^j

t.j.: $q_0 \approx 0$, $q_1 \approx 00$, 5-tý stav $\approx$ 00000

Kódování symbolů

(předpokládáme páskové symboly $\in \Sigma \cup \{\Delta\}$

- opět uspořádáme ub. $\Sigma = \{a_1, a_2, \dots, a_n\}$ a zvolíme tyto kódy

$\Delta \approx e$ (pr. řetězec)

$L \approx 0$

$R \approx 00$

$a_i \approx 0^{i+2} \quad i = 1, \dots, n$

Kódování přechodů:

přechod $\delta(p,x) = (q,y) \quad y \equiv \begin{cases} x \in \Gamma \\ L \\ R \end{cases}$

reprezentujeme čtveřicí

(p, x, q, y) a kódujeme zřetězením kódů

p, x, q, y a symbolu 1 v roli separátoru:

„kód p" „kód x" „kódy" „kódy"

Příklad

Přechod $\delta(q_0, x) = (q_F, R)$ kde kód x je 000 bude zakódován řetězcem 01000100100

$\delta(q_0, \Delta) = (q_F, \Delta) \approx 011001$

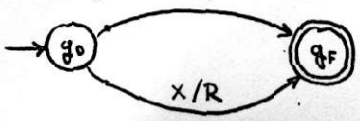
Kódování celého T. stroje

zřetězením posloupnosti všech přechodů (v syst. pořadí) a symbolu 1:

„kód 1. přechodu" 1 „kód 2. přechodu" 1 ... 1 „kód m-tého přechodu" 1

Příklad:

T. stroj



Jeho kód:

101100100010100010010011

Příklad jazyka který není přijímán T. strojem

zavedený kódovací systém nám umožňuje zobrazit každý T stroj jako binární číslo, opak však neplatí.

Uvažujme nyní třídu $\mathcal{T}$ všech T. strojů se vst. abecedou Σ a páskovou abecedou $\Sigma \cup \{\Delta\}$ a zobrazení

$$\tau : \mathbb{N} \to \mathcal{T}$$

takové, že

$$\tau(n) = \begin{cases} T & \text{právě když binární zápis čísla } n \text{ je kódem stroje } T \\ \bar{T} & \text{jinak; } \bar{T} \text{ je triviální T. stroj} \end{cases}$$

- τ je zřejmě surjektivní zobrazení

Vytvořme nyní jinou funkci $\varsigma : \Sigma^* \to \mathcal{T}$ takovou, že pro lib $w \in \Sigma^*$

$$\varsigma(w) = \tau(|w|) \equiv M_w$$

($|w|$ čteme: „číslo shodnotou")

t.j. funkci, která k řetězci w přiřadí T. stroj, jehož kódem je b. číslo shodnotou

ς je opět surjekce.

Protože symboly w i abecedy M_w jsou shodná, má smysl uplikovat stroj M_w na řetězec w. Definujme jazyk L_0 takto:

$$L_0 = \{w \mid M_w \text{ nepřijímá řetězec } w\}$$

Ukážeme, že jazyk L_0 nemůže být přijímán žádným T. strojem. Budeme postupovat tak, že předpokládáme opak, t.j. ex. $T_0 \in \mathcal{T}$, tak, že $L_0 = L(T_0)$. Pak také existuje $w_0 \in \Sigma^*$ tak, že $L_0 = L(M_{w_0})$. ($T_0 \equiv M_{w_0}$)

Nyní se ptáme, zda $w_0 \in L(M_{w_0})$. Buď platí (a) $w_0 \in L(M_{w_0})$, nebo (b) $w_0 \notin L(M_{w_0})$

Z definice L_0 plyne:

$w_0 \in L(M_{w_0}) \Rightarrow w_0 \notin L_0$ a $w_0 \notin L(M_{w_0}) \Rightarrow w_0 \in L_0$. Protože $L_0 = L(M_{w_0})$

jsou obě implikace sporné a tudíž $L_0 \notin \mathcal{L}_0$.

Universální Turingovy stroje

Je to koncept „programovatelného" stroje, který umožňuje ve tvaru vstupního řetězce specifikovat konkrétní T. stroj i data, nad nimiž má tento konkrétní stroj pracovat.

Pro specifikaci T. stroje využijeme jeho binární zakódování z předch. odstavce. Ve stejném duchu budeme kódovat i vst. řetězec, který má být zpracováván:

1000010001000001

kódy symbolů

Celý „problém" je pak zadán zřetězením kódu stroje a kódu dat, např:

10110010001010001001001100010001 0001

1. přechod — 2. přechod — symboly

kódovaný stroj — kódovaná data

Univerzální T. stroj, který zpracuje toto zadání může být navržen jako 3-paskový T. stroj takto:

1. páska - program + data (+ výst. data)
2. páska - pracovní páska
3. páska - registruje stav stroje, který je simulován (stav programu)

Víme, že k takovému stroji můžeme sestrojit ekvivalentní 1-páskový Turingův stroj. Označme tento 1-páskový T. stroj symbolem T_U.

Jazyky přijatelné versus rozhodnutelné Turingovy stroji

Schopnost T. strojů přijmout jazyk není symetrická se schopností nepřijmout jeho komplement. S pomocí univerzálního T. stroje ukážeme tento fakt na příkladě jazyka L_0; zkonstruujeme T. stroj, který přijímá komplement jazyka L_0:

Nejprve vytvoříme stroj M'_w, který pro každý řetězec $w \in \Sigma^*$

1. vytvoří b. číslo $\zeta^{-1}(\varsigma(w))$, t.j. kód automatu M_w
2. zřetězí tento kód s kódem účižce w

Pak vytvoříme složení T. strojů $\rightarrow M'_w \rightarrow T_U$

Výsledný stroj přijímá jazyk L_1

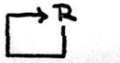
$$L_1 = \{w \mid M_w \text{ přijímá řetězec } w\} = \overline{L_0}$$

To tedy znamená, že existují jazyky přijímané T. strojem, pro které nelze sestrojit T. stroj, který končí zprávou „Y" pro lib. $w \in L$ a zprávou „N" pro vš. $w \notin L$. Tyto jazyky se nazývají jazyky přijatelné (acceptable) T. strojem. Jazyky, pro které lze takový T. stroj sestrojit se nazývají jazyky rozhodnutelné (decidable) T. strojem, či krátce rozhodnutelné jazyky.

Příklady

1. Jazyk L_1 je přijatelný T. strojem, nevšak rozhodnutelný.
2. Bezkontextové jazyky jsou rozhodnutelné jazyky.
3. Jazyk $L = \{x^n y^n z^n \mid n \in \mathbb{N}\}$ je rozhodnutelný

(V uvedeném diagramu T. stroje pro tento jazyk nahradíme stroj



strojem $\rightarrow L_\# \rightarrow RNL\Delta$).

Terminologická poznámka:

Jazyky přijatelné T. strojem se nazývají také jazyky rekurzivně vyčíslitelné (recursively enumerable languages)

Jazyky rozhodnutelné T. strojem také jazyky rekurzivní jazyky (recursive languages).

Problém zastavení T. stroje

Dokážeme klasický nerozhodnutelný problém T. strojů nazývaný problém zastavení (Halting problem)

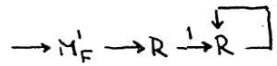
Předpokládejme, že abecedy T. stroje jsou binární t.j. $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\Delta\}$ a označme kódovanou verzi T. stroje M symbolem $\sigma(M)$. $\sigma(M)$ je b. řetězec, na který může být aplikován T. stroj M ($\sigma(M)$ dáme na vstup stroje M). Zajímá nás, zda se T. stroj M pro vstupní řetězec $\sigma(M)$ zastaví (M je selfterminating), nebo ne (M je nonselfterminating).

Nyní definujme jazyk L_F nad abecedou $\Sigma = \{0,1\}$ takto

$$L_F = \{\sigma(M) \mid M \text{ je selfterminating}\}$$

Problém zastavení je takto formulován jako problém rozhodnutelnosti jazyka L_F. Jazyk L_F je bohužel nerozhodnutelný T. strojem, což dokážeme, podobně jako v případě jazyka L_0, sporem.

Předpokládejme tedy, že existuje T. stroj, označme ho jako M_F, který rozhoduje jazyk L_F. Nyní modifikujme stroj M_F tak, že modifikovaný T. stroj M'_F dává na výstup 0, resp. 1, právě když stroj M_F dává zprávu N, resp. Y. M'_F má takto páskovou abecedu $\{0,1,\Delta\}$ a může být použit k vytvoření stroje

M_0: 

Stroj M_0 se zastaví právě když M'_F dosáhne koncový stav s výstupem 0.

Nyní položím otázku, zda je stroj M_0 selfterminating. Pokud je L_F rozhodnutelný jazyk, pak existuje odpověď ANO, nebo NE a žádná jiná možnost.

a) Předpokládejme, že M_0 je selfterminating. Pak pro vstup $\sigma(M_0)$ stroj M'_F zastaví s výstupem 1 a stroj M_0 přejde po hraně $R \xrightarrow{1} R$ a nezastaví se. Tedy „M_0 je selfterminating $\Rightarrow$ M_0 je nonselfterminating"

b) Předpokládejme, že M_0 je nonselfterminating. Pak pro vstup $\sigma(M_0)$ se stroj M'_F zastaví s výstupem 0 a stroj M_0 se zastaví. Tedy „M_0 není selfterminating $\Rightarrow$ M_0 je selfterminating"

Inconzistence obou implikací nás vede k závěru, že předpoklad, že L_F je rozhodnutelný T. strojem je mylný a tedy problém zastavení T. stroje je nerozhodnutelný.

Postův problém přiřazení

Definice Postův systém nad abecedou $\Sigma = \{a,b\}$ je dán množinou uspořádaných dvojic (α_i, β_i), $i = 1,2,\dots,k$ řetězců nad abecedou Σ, t.j.

$$S = \{(\alpha_1,\beta_1),(\alpha_2,\beta_2),\dots(\alpha_k,\beta_k),\ k \geq 1$$

Řešením Postova systému je každá neprázdná posloupnost přirozených čísel $i_1, i_2, \dots, i_m$ $(i_j \leq k)$ taková, že

$$\alpha_{i_1}\alpha_{i_2}\dots\alpha_{i_m} = \beta_{i_1}\beta_{i_2}\dots\beta_{i_m}$$

Postův problém přiřazení zní: Existuje pro daný (libovolný) Postův systém řešení?

VĚTA Postův problém přiřazení je nerozhodnutelný.

Příklady: 1. Uvažujme Postův systém nad abecedou $\{a,b\}$:

$$S = \{(b,bbb),(babbb,ba),(ba,a)\}$$

Tento systém má řešení $i_1 = 2,\ i_2 = 1,\ i_3 = 1,\ i_4 = 3$ (tedy $m = 4$):

$$\underbrace{b\,a}_{\alpha_2}\,\underbrace{b\,b\,b}_{\alpha_1}\,\underbrace{b\,a}_{\alpha_1}\,\underbrace{b\,a}_{\alpha_3} = \underbrace{b\,a}_{\beta_2}\,\underbrace{b\,b\,b}_{\beta_1}\,\underbrace{b\,b\,b}_{\beta_1}\,\underbrace{a}_{\beta_3}$$

2. Naopak Postův systém $S = \{(ab,abb),(a,ba),(b,bb)\}$ nemá řešení protože $|\alpha_i| < |\beta_i|$ pro $i = 1,2,3$ □

Postův problém přiřazení se používá k důkazu nerozhodnutelnosti důležitých problémů:

1. Problém ekvivalence bezkontextových gramatik
2. Problém neprázdnosti jazyka generovaného kontextovou gramatikou
3. Problém derivace věty v gramatice typu 0

Vyčíslitelnost - Computability

- Operační přístup x Funkcionální přístup
- Churchova téze

Základy teorie rekurzivních funkcí

Budeme se snažit identifikovat takové funkce, které jsou „spočitatelné", t.j. vyčíslitelné v obecném smyslu (bez ohledu na konkrétní výpočetní systém). Abychom snížili extrémní velikost třídy těchto funkcí, která je dána také varietou definičních oborů a oborů hodnot, omezíme se, uvažujíce možnost kódování, na funkce tvaru:

$$f: N^m \to N^n$$

kde $N = \{0,1,2,\dots\}$, $m,n \in N$

Klasifikace:

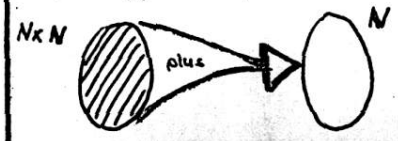
- Parciální funkce
  - Totální funkce
  - Striktně parciální funkce

Příklad:

Totální funkce

plus : $N \times N \to N$

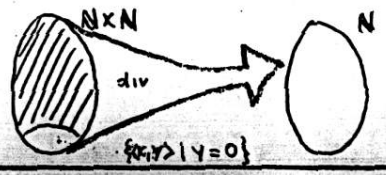
plus$(x,y) = x + y$



Striktně parciální funkce:

div : $N \times N \to N$

div(x,y) = celá část x/y, je-li $y \neq 0$



Konvence:

n-tici $(x_1, x_2, \dots, x_n) \in \mathbb{N}^n$ budeme označovat jako $\bar{x}$

Počáteční funkce

Hierarchie vyčíslitelných funkcí je založena na dostatečně elementárních tzv. počátečních funkcí, tvořících „stavební kameny" vyšších funkcí.

Jsou to tyto funkce:

1. Nulová funkce (zero function)

$$\zeta() = 0$$

zobrazuje „prázdnou n-tici" $\mapsto 0$

2. Funkce následníka (Successor function)

$$\sigma : \mathbb{N} \to \mathbb{N} \quad ; \quad \sigma(x) = x + 1$$

3. Projekce

$$\pi^n_k : \mathbb{N}^n \to \mathbb{N}$$

Vybírá z n-tice k-tou složku, např.:

$\pi^3_2(7,6,4) = 6$, $\pi^2_1(5,17) = 5$

Speciální případ: $\pi^n_0 : \mathbb{N}^n \to \mathbb{N}^0$ t.j. např $\pi^3_0(1,2,3) = ()$

Primitivně rekurzivní funkce

Nyní definujeme 3 způsoby vytváření nových, složitějších funkcí:

1. Kombinace

kombinací dvou funkcí $f: \mathbb{N}^k \to \mathbb{N}^m$ a $g: \mathbb{N}^k \to \mathbb{N}^n$ získáme funkci

$$f \times g : \mathbb{N}^k \to \mathbb{N}^{m+n}$$

pro kterou $f \times g(\bar{x}) = (f(\bar{x}), g(\bar{x}))$

Např.: $\pi^3_1 \times \pi^3_3 (4,12,8) = (4,8)$

2. Kompozice

Kompozice dvou funkcí $f: \mathbb{N}^k \to \mathbb{N}^m$ a $g: \mathbb{N}^m \to \mathbb{N}^n$ je funkce,

$$f \circ g : \mathbb{N}^k \to \mathbb{N}^n$$

pro kterou $f \circ g(\bar{x}) = g(f(\bar{x}))$ $\quad \bar{x} \in \mathbb{N}^k$

Např.: $\sigma \circ \zeta() = 1$, $\sigma \circ \sigma \circ \zeta() = 2$

3. Primitivní rekurze

Příklad: Předpokládejme, že chceme definovat funkci $f: \mathbb{N}^2 \to \mathbb{N}$, jejíž hodnoty $f(x,y)$ udávají počet vrcholů x-arního stromu hloubky y, který je pravidelný (úplný):

zřejmě
(a) $f(x,0) = 1$

(b) strom hloubky n má x^n listů; zvětšením hloubky o jedničku na $y+1$ musíme přidat $x^{y+1} = x^y \cdot x$ vrcholů

Takže funkci f můžeme definovat násl. předpisem:

$$f(x,0) = x^0$$
$$f(x,y+1) = f(x,y) + x^y \cdot x$$

(Např. $f(3,2) = 13$)

Primitivní rekurze je technika, která umožňuje vytvořit funkci f

$$f: \mathbb{N}^{k+1} \to \mathbb{N}^m$$

na základě jiných dvou funkcí g a h; $g: \mathbb{N}^k \to \mathbb{N}^m$, $h: \mathbb{N}^{k+m+1} \to \mathbb{N}^m$

rovnicemi

$$f(\bar{x}, 0) = g(\bar{x})$$
$$f(\bar{x}, y+1) = h(\bar{x}, y, f(\bar{x}, y)) \quad , \bar{x} \in \mathbb{N}^k$$

Ilustrace schématu vyčíslení:

$$f(\bar{x},3) = h(\bar{x},2,f(\bar{x},2))$$
$$f(\bar{x},2) = h(\bar{x},1,f(\bar{x},1))$$
$$f(\bar{x},1) = h(\bar{x},0,f(\bar{x},0))$$
$$f(\bar{x},0) = g(\bar{x})$$

Příklad

Uvažujme funkci $plus: \mathbb{N}^2 \to \mathbb{N}$. Může být definována pomocí primitivní rekurze takto:

$$plus(x,0) = \pi_1^1(x)$$
$$plus(x,y+1) = \sigma \circ \pi_3^3(x,y,plus(x,y))$$

což vyjadřuje:

(1) $x+0 = x$

(2) $x+(y+1) =$ následníku $x+y$

□

Definice: Třída primitivních rekurzivních funkcí obsahuje všechny funkce, které mohou být z počátečních funkcí vytvořeny kombinací, kompozicí a pr. rekurzí.

Věta: Každá primitivní rekurzivní funkce je totální funkcí.

D. Poč. funkce jsou totální, aplikací kombinace, kompozice i pr. rekurze na totální funkce dostaneme totální funkce.

Příklady primitivně rekurzivních funkcí

Třída p.r. funkcí zahrnuje většinu funkcí typických v aplikacích počítačů

Konvence: Namísto funkcionálních zápisů typu

$$h \equiv plus \circ (\pi_1^3 \times \pi_2^3)$$

budeme někdy používat standard. zápisů $h(x,y,z) = plus(x,z)$ nebo $h(x,y,z) = x+z$

Konstantní funkce

Zavedeme funkci K_m^n, která lib. n-tici $\bar{x} \in \mathbb{N}^n$ přiřadí konst. hodnotu $m \in \mathbb{N}$

$$K_m^0 \equiv \underbrace{\sigma \circ \sigma \circ \ldots \circ \sigma}_{m\text{-krát}} \circ$$

Také pro $n>0$ je K_m^n funkce rek. primitivní:

$$K_m^n(\bar{x},0) = K_m^{n-1}(\bar{x})$$
$$K_m^n(\bar{x},y+1) = \pi_{n+1}^{n+1}(\bar{x},y,K_m^n(\bar{x},y))$$

Např.: $K_3^2(1,1) = \pi_3^3(1,0,K_3^2(1,0)) = K_3^2(1,0) = K_3^1(1) = K_3^1(0) = K_3^0() = 3$

Kombinací funkcí K_m^n dostáváme konstanty z $\mathbb{N}^n$, $k>1$

Např. $K_1^3 \times K_5^2\ (x,y,z) = (2,5)$

Násobení

$$mult(x,0) = K_0^1(x)$$

$$mult(x,y+1) = plus(x, mult(x,y))$$

Umocňování

$exp: \mathbb{N}^2 \to \mathbb{N}$ – analogicky – cvičení

Funkce předchůdce

$$pred(0) = \xi()$$

$$pred(y+1) = \Pi_1^2(y, pred(y))$$

Pozn.: pred je totální f., $pred(0) = 0$

Funkce monus

$$monus(x,0) = \Pi_1^1(x)$$

$$monus(x,y+1) = pred(monus(x,y))$$

Význam:
$$monus(x,y) = \begin{cases} x-y & \text{je-li } x \geq 0 \\ 0 & \text{jinak} \end{cases}$$

$\boxed{monus(x,y) \equiv x \dot{-} y}$

Funkce eq

$$eq(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq 0 \end{cases}$$

Definice:
$$eq(x,y) = 1 \dot{-} ((y \dot{-} x) + (x \dot{-} y))$$

nebo formálněji:
$$eq \equiv monus \circ (K_1^2 \times (plus \circ ((monus \circ (\Pi_2^2 \times \Pi_1^2)) \times monus \circ (\Pi_1^2 \times \Pi_2^2))))$$

Příklad:
$$eq(5,3) = 1 \dot{-} ((3-5)+(5-3)) = 1 \dot{-} (0+2) = 1 \dot{-} 2 = 0$$

Funkce ¬eq

$$\neg eq \equiv monus \circ (K_1^2 \times eq) \qquad (\equiv 1 \dot{-} eq)$$

Tabulární funkce

Uvažujme funkce typu

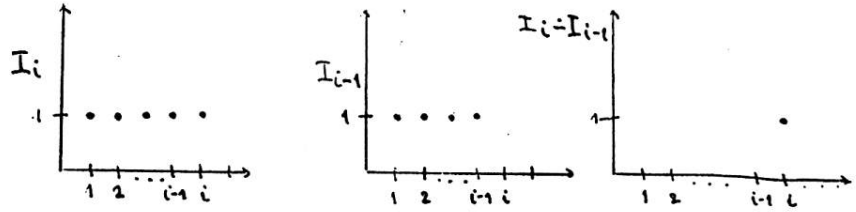
$$f(x) = \begin{cases} 3 & \text{je-li } x=0 \\ 5 & \text{je-li } x=4 \\ 2 & \text{v ostatních případech} \end{cases}$$

které bývají zadávány tabulkou. Tyto funkce lze tvořit pomocí charakt. funkce

$$\varphi_i(x) = \begin{cases} 1 & \text{je-li } x=i \\ 0 & \text{jinak} \end{cases}$$

která může být vyjádřena jako $monus(I_i, I_{i-1})$, kde

$$I_i(x) = eq(x \dot{-} i, 0)$$



Tabulární funkce mohou být nyní tvořeny konečným součtem násobků konstant, funkcí φ_i a $\neg\varphi_i$

Např. nahoře uvedená funkce f(x) je vyjádřitelná ve tvaru

$$mult(3, \varphi_0) + mult(5, \varphi_4) + mult(2, mult(\neg\varphi_0, \neg\varphi_4))$$

Funkce quo

$$quo(x,y) = \begin{cases} \text{celá část podílu } x/y & \text{je-li } y \neq 0 \\ 0 & \text{je-li } y = 0 \end{cases}$$

Tato funkce může být definována primitivní rekurzí:

$$quo(0,y) = 0$$

$$quo(x+1,y) = quo(x,y) + eq(x+1, mult(quo(x,y), y) + y)$$

Funkce mimo primitivní rekurzivní funkce

Existují funkce, které jsou vyčíslitelné a nejsou p.r. funkcemi. Jsou to všechny striktně parciální funkce (jako div), ale i totální funkce. Taková totální funkce byla prezentována W. Ackermannem (1928) a nazývá se Ackermannova funkce. Je dána rovnicemi:

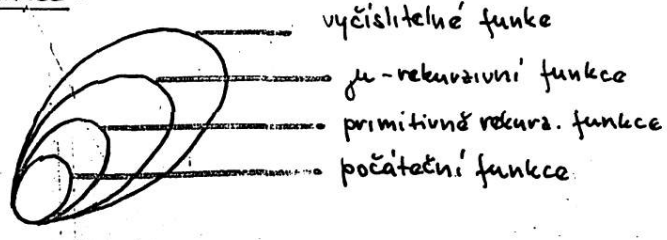
$$A(0,y) = y+1$$
$$A(x+1,0) = A(x,1)$$
$$A(x+1,y+1) = A(x, A(x+1,y))$$

VĚTA Existuje totální funkce z $\mathbb{N}$ do $\mathbb{N}$, která není prim. rekurzivní.

D. Definice funkcí, které jsou pr. rekurzivní, budeme chápat jako řetězce a můžeme je uspořádat v lexikografickém pořadí s označením $f_1, f_2, \dots f_n, \dots$

Definujme nyní funkci $f: \mathbb{N} \to \mathbb{N}$ tak, že $f(n) = f_n(n) + 1$ pro vš. $n \in \mathbb{N}$. f je jasně totální a vyčíslitelná. f však není primitivně rekurzivní (kdyby byla pak $f \equiv f_m$ pro nějaké $m \in \mathbb{N}$. Pak ale $f(m) = f_m(m)$ ale ne $f_m(m)+1$ jak vyžadují definice funkce f.

Definice Třída totálních vyčíslitelných funkcí se nazývá *μ-rekurzivní funkce*.



Parciálně rekurzivní funkce

K rozšíření třídy vyčíslitelných funkcí za totální vyčíslitelné funkce zavedeme techniku známou pod názvem *minimalizace*.

Tato technika umožňuje vytvořit funkci $f: \mathbb{N}^n \to \mathbb{N}$ z jiné funkce

$$g: \mathbb{N}^{n+1} \to \mathbb{N}$$

předpisem, v němž $f(\bar{x})$ je nejmenší y takové, že

(1) $g(\bar{x}, y) = 0$

(2) $g(\bar{x}, z)$ je definována pro vš. $z < y$, $z \in \mathbb{N}$

Tuto konstrukci zapisujeme notací:

$$f(\bar{x}) = \mu y [g(\bar{x}, y) = 0]$$

Příklady:

1. $f(x) = \mu y [plus(x,y) = 0]$ t.j. $f(x) = \begin{cases} 0 \text{ pro } x = 0 \\ \text{nedef. jinak} \end{cases}$
2. $div(x,y) = \mu t [((x+1) \dot{-} (mult(t,y) + y)) = 0]$
3. $i(x) = \mu y [monus(x,y) = 0]$ t.j. identická funkce

Funkce definovaná minimalizací je skutečně vyčíslitelná. Výpočet hodnoty $f(\bar{x})$ zahrnuje výpočet $g(\bar{x},0), g(\bar{x},1), \dots$ tak dlouho, pokud nedostaneme $g(\bar{x},y) = 0$, nebo $g(\bar{x},z)$ je nedef.

Definice: Třída parciálně rekurzivních funkcí je třída parciálních funkcí, které mohou být vytvořeny z počátečních funkcí aplikací kombinací, kompozicí, primitivní rekurzí a minimalizací

Turingovsky vyčíslitelné funkce

Definice: Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ **vyčísluje (počítá)** p. funkci $f: \Sigma_1^{*m} \to \Sigma_1^{*n}$ $(\Sigma_1 \subseteq \Gamma,\ \Delta \notin \Sigma_1)$, jestliže pro každé $(w_1, w_2, \dots, w_m) \in \Sigma^{*m}$ a odpovídající poč. konfiguraci $\underline{\Delta} w_1 \Delta w_2 \Delta \dots \Delta w_m \Delta\Delta\Delta$ stroj M

1. v případě, že $f(w_1, \dots, w_m)$ je definována, pak M zastaví u páska obsahuje $\Delta v_1 \Delta v_2 \Delta \dots \Delta v_n \Delta\Delta\Delta$, kde $(v_1, v_2, \dots, v_n) = f(w_1, \dots w_m)$
2. v případě, že $f(w_1, \dots, w_m)$ není definována, M cykluje (nikdy nezastaví) nebo zastaví abnormálně.

Parciální funkce, kterou může počítat nějaký T. stroj se nazývá funkcí Turingovsky vyčíslitelnou.

Příklad:

1. 

T. stroj, který počítá funkci $f(w_1, w_2, w_3) = (w_1, w_3)$

2. Funkce $f(w) = \begin{cases} Y & \text{jestliže } w \in L_0 \quad (L_0 \text{ je jazyk z předch. odst.}) \\ \text{nedef} & \text{jinak} \end{cases}$

$g(x) = \begin{cases} 1 & \text{je-li } w = \sigma(M) \text{ pro selfterminating stroj M} \\ 0 & \text{jinak} \end{cases}$

nejsou T. vyčíslitelné funkce

Pozn.: Definice výpočtu funkce T. strojem nepředpokládala speciální polici hlavy v konc. konfiguraci. Můžeme předpokládat, že M končí v konfiguraci $\underline{\Delta} v_1 \Delta \dots \Delta v_n \Delta\Delta\Delta$ bez újmy na obecnosti.

Turingovská vyčíslitelnost parciálně rekurzivních funkcí

Budeme uvažovat T. stroj s abecedou $\Sigma = \{0, 1\}$ a p. funkce tvaru $f: N^m \to N^n$. m (n) - tice budeme kódovat podle vzoru:

$\Delta 11 \Delta 10 \Delta 100 \Delta\Delta$ reprezentuje trojici $(3, 2, 4)$

VĚTA Každá parciálně rekurzivní funkce je Turingovsky vyčíslitelná.

Důkaz: 1. Nejprve je třeba nalézt T. stroje, které vyčíslují počáteční funkce ζ, σ, π.

a) $\zeta: \to$ ROL

b, c) - cvičení

2. Dále popíšeme konstrukci T. strojů pro aplikaci kombinace, komporice, pr. rekurze a minimalizace

a) kombinace

Nechť T. stroj M_1, resp. M_2 vyčísluje p. funkci g_1, resp. g_2. Stroj M, který vyčísluje $g_1 \times g_2$ bude 3-páskový T. stroj, který začne duplikováním vstupu na 2. a 3. pásku. Pak M simuluje M_1 s využitím pásky 2 a M_2 s použitím pásky 3. Skončí-li M_1 i M_2 řádně, M vyčistí pásku 1 a okopíruje na ni obsah pásek 2 a 3 (oddělených blankem) a zastaví.

b. komporice

Funkci $g_1 \circ g_2$ realizuje stroj $\to M_1 M_2$

c. Primitivní rekurze

Uvažujme schema pr. rekurze

$$f(\bar{x},0) = g(\bar{x})$$
$$f(\bar{x},y+1) = h(\bar{x},y,f(\bar{x},y))$$

kde g p. funkce, kterou vyčísluje stroj M_1 a h parc. funkce, kterou vyčísluje stroj M_2. Funkci f vyčísluje T. stroj, který pracuje takto:

I. Je-li poslední složkou vstupu 0, pak vymaže tuto složku, vrátí hlavu na začátek a simuluje stroj M_1.

II. Není-li poslední složkou vstupu 0, pak páska musí obsahovat sekvenci

$\Delta\,\bar{x}\,\Delta\,y+1\,\Delta\Delta\Delta$ pro nějaké $y+1 > 0$. Pak

(α) S využitím strojů pro kopírování a dekrement (viz cvičení) transformuj obsah pásky na sekvenci

$\Delta\,\bar{x}\,\Delta\,y\,\Delta\,\bar{x}\,\Delta\,y-1\,\Delta\ldots\Delta\,\bar{x}\,\Delta\,0\,\Delta\,\bar{x}\,\Delta\Delta$

Pak přesuň hlavu bezprostředně za 0 a simuluj stroj M_1

(β) Nyní bude páska obsahovat

$\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y-1\Delta\ldots\Delta\bar{x}\Delta 0\Delta\, g(\bar{x})\,\Delta\Delta$ což je ekvivalentní

$\sim\sim$ ‖ $\sim\sim$ $f(\bar{x},0)\Delta\Delta$.

Přesuň hlavu před poslední $\bar{x}$ a simuluj stroj M_2. To vede k

$\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y-1\Delta\ldots\Delta\bar{x}\Delta 1\Delta\, h(\bar{x},0,f(\bar{x},0)\Delta\Delta$ což je ekvival.

— ‖ — $f(\bar{x},1)\,\Delta\Delta$

(γ) Pokračuj v aplikaci stroje M_2 na zbytek pásky už je M_2 aplikován na $\Delta\bar{x}\Delta y\Delta f(\bar{x},y)$ a páska je zredukována do tvaru $\Delta\, h(\bar{x},y,f(\bar{x},y)\Delta\Delta$, což je ekvivalentní žádanému výstupu $\Delta\, f(\bar{x},y+1)\,\Delta\Delta$

d. minimalizace

Uvažujme funkci $\mu y[g(\bar{x},y)=0]$, kde p. funkci g vyčísluje stroj M_1, zkonstruujeme 3-páskový stroj M, který pracuje takto:

1. Zapiše 0 na pásku 2
2. Kopíruje obsah pásky 1 a následně pásky 2 na pásku 3
3. Simuluje stroj M_1 na pásce 3
4. Jestliže páska 3 obsahuje 0, vymaže pásku 1, okopíruje pásku 2 na pásku 1 a zastaví. Jinak inkrementuje obsah na pásce 2, vymaže pásku 3 a pokračuje krokem (2).

Reprezentace T. stroje parciálně rekurzivními funkcemi

Abychom dokázali, že Turingova a Churchova třída jsou ekvivalentní, zbývá ukázat, že výpočetní síla T. strojů je omezena na možnost vyčíslovat parciálně rekurzivní funkce. K tomu účelu uvažujme T. stroj $M=(Q,\Sigma,\Gamma,\delta,q_0,q_F)$ a položme $b=|\Gamma|$. Nyní budeme obsah pásky interpretovat jako v opačném pořadí zapsané nezáporné celé číslo při základu b. Např: Bude-li $\Gamma=\{x,y,\Delta\}$ a interpretujeme-li $x\approx 1$, $y\approx 2$, $\Delta\approx 0$ (vždy) pak páska obsahující $\Delta yx\Delta\Delta y\Delta\Delta\ldots$ je ekvivalentní 02100200... což po reverzi reprezentuje číslo ...00200120 při základu 3 a tedy 501.

S touto interpretací můžeme činnost každého T. stroje chápat jako výpočet funkce $f:\mathbb{N}\to\mathbb{N}$, která číslu odpovídajícímu počátečnímu obsahu pásky „přiřazuje" číslo odpovídající obs. pásky po zastavení stroje. Charakter funkce f specifikuje následující věta.

VĚTA

Každý výpočetní proces prováděný T. strojem je procesem vyčíslení nějaké parciálně rekurzivní funkce.

Důkaz:

Budeme vycházet z právě zavedené interpretace činnosti T. stroje M, t.j. pro každé $n \in N$, $f(n)$ je definována obsahem pásky při zastavení stroje M. Dále provedeme zakódování stavů: $0 \approx q_0$, $1 \approx q_F$, ostatní stavy čísly $2, 3, \ldots, k-1$ za předpokladu $|Q| = k$. Tedy jak stavy, tak symboly mají přiřazené num. hodnoty a můžeme definovat funkce, které sumarizují diagram přechodů stroje M' (M' je zakódovaný stroj M):

$$mov(p,x) = \begin{cases} 2 & \text{jestliže } \delta(p,x) = (*,R) \\ 1 & \text{jestliže } \delta(p,x) = (*,L) \\ 0 & \text{jinak} \end{cases}$$

$$sym(p,x) = \begin{cases} Y & \text{jestliže } \delta(p,x) = (*,Y) \\ x & \text{jinak} \end{cases}$$

$$state(p,x) = \begin{cases} q & \text{jestliže } \delta(p,x) = (q,*) \\ k & \text{jestliže } p=0, \text{ nebo } \delta(p,x) \text{ není definována} \end{cases}$$

Funkce sym, mov a state jsou tabulární funkce (totální) a jsou tedy primitivně rekurzivní.

Nyní uvažujme konfiguraci T. stroje M' ve tvaru trojice

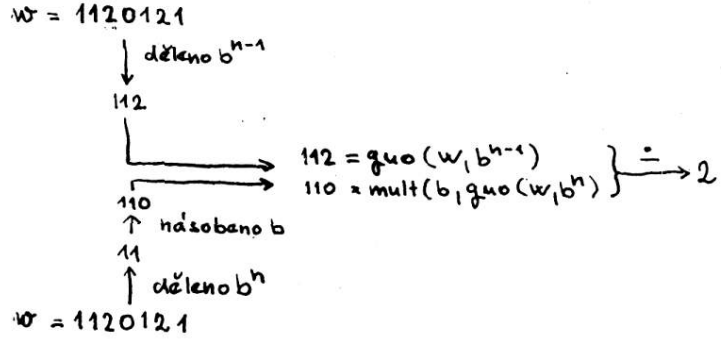
$$(w, p, n)$$

kde w je obsah pásky, p přítomný stav, n pozice hlavy ($n \geq 1$, nejlevější pozice je rovna 1)

z informace obsažené v konfiguraci může být vypočítán symbol, který se nachází pod hlavou, p.r. funkcí cursym:

$$cursym(w,p,n) = quo(w, b^{n-1}) \dot{-} mult(b, quo(w, b^n))$$

jak ilustruje násl. obrázek:



Obr. Nalezení n-té číslice řetězce $w = 1120121$ pro $n = 5$ a $b = 3$

Další funkce, které definujeme na množině konfigurací jsou:

$$nexthead(w,p,n) = n \dot{-} eq(mov(p, cursym(w,p,n)), 1) + eq(mov(p, cursym(w,p,n)), 2)$$

určující příští pozici hlavy (0 indikuje abnormální posuv z 1. pozice vlevo)

$$nextstate(w,p,n) = state(p, cursym(w,p,n)) + mult(k, \overset{1 \dot{-} 0}{\neg} nexthead(w,p,n))$$

nulhead = 0

(Normálně je 2. sčítanec roven 0; při abnormálním posuvu tato funkce vyčísli nelegální stav větší než $k-1$.

4 konečné funkce

$$nexttape(w,p,n) = (w \dot{-} mult(b^n, cursym(w,p,n))) + mult(b^n, sym(p, cursym(w,p,n)))$$

která vyčísluje celé číslo reprezentující nový obsah pásky, po provedení přechodu z konfigurace (w,p,n).

Kombinací 3 předchozích funkcí dostaneme funkci (step), která modeluje 1 krok T. stroje, t.j. přechod do nové konfigurace:

$$step = nexttape \times nextstate \times nexthead$$

Nyní definujme funkci $run : N^4 \to N^3$; $run(w,p,n,t)$ realizuje t přechodů z konfigurace (w,p,n). Opět užijeme pr. rekurze

$$run(w,p,n,0) = (w,p,n)$$
$$run(w,p,n,t+1) = step(run(w,p,n,t))$$

Výstupní funkce vyčíslovaná strojem M' (při vstupu w) je hodnota pásky po dosažení koncového stavu (stavu 0). Počet požadovaných kroků stroje M je dán funkcí stoptime

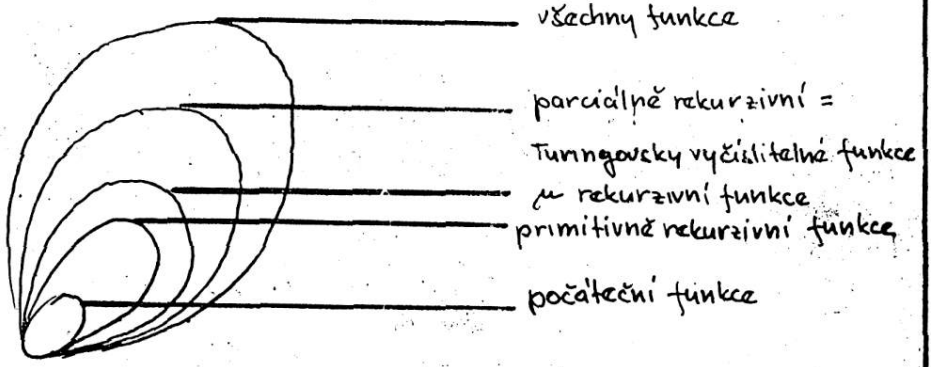
$$stoptime(w) = \mu t\,[\pi_2^3(run(w,1,1,t)) = 0]$$

Nakonec tedy, je-li $f : N \to N$ p. funkce počítaná strojem M, pak

$$f(w) = \pi_1^3(run(w,1,1,stoptime(w)))$$

z jejíž konstrukce plyne, že f je parciálně rekurzivní funkce.

Shrňme v obrázku získané informace:



Složitost - Complexity

Zavedení T. stroje nám umožnilo klasifikovat problémy do 2 tříd:

(a) problémy neřešitelné

(b) problémy řešitelné

Nyní se budeme zabývat třídou (b) v souvislosti s otázkou složitosti řešení problému.

Co je to složitost problému?

Klasifikace složitosti ← časová složitost
paměťová složitost
(prostorová složitost)

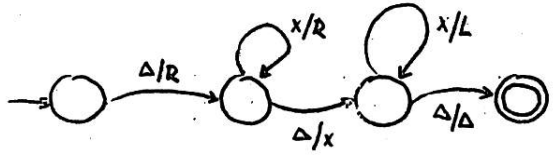
Složitost výpočtů prováděných T. strojem

Časová složitost - počet kroků T. stroje provedený od počátku do konce výpočtu

Paměťová složitost - počet „buněk" pásky T. stroje požadovaný pro daný výpočet

Příklad:

T:



pro vstup ΔxxxxΔΔ je ← časová složitost výpočtu stroje T rovna 9
paměťová — " — 5

Jednoduchá implikace:

Je-li časová složitost výpočtu prováděného T. strojem rovna n, pak paměťová složitost tohoto výpočtu není větší než n+1.

Složitost algoritmů

Základní teoretický přístup vychází z T. tézе:

Každý algoritmus je implementovatelný jistým T. strojem

Analýza složitosti algoritmu je pak analýzou složitosti příslušného T. stroje, jejíž cílem je vyjádřit (kvantifikovat) požadované zdroje (čas, paměť) funkcí závisející na délce vstupního řetězce.

Obvykle rozeznáváme
(1) analýzu složitosti nejhoršího případu (worst-case analysis)
(2) analýzu nejlepšího případu (best-case analysis)
(3) analýzu složitosti průměrného případu (average-case analysis)

Případ (3) nazývaný také průměrná výkonnost algoritmu je definován přirozeně:

Jestliže aplikace algoritmu (T. stroje) vede k m různým výpočtům (případům) se složitostí $c_1, c_2, \dots, c_m$, jež nastávají s pravděpodobností $p_1, p_2, \dots, p_m$, pak průměrná složitost (výkonnost) algoritmu je dána

$$\sum_{i=1}^{m} p_i c_i$$

Analýza složitosti algoritmů v jiném (méně přesném) prostředí, než jsou T. stroje je založena na předpokladu známém jako

uniform cost assumption

který umožňuje používat stejných konceptů.

Složitost problémů

Složitost problému je složitost jeho nejjednoduššího řešení

Příklad : T. stroj pro srovnání 2 řetězců ↑ pessim. existovat

Rates (Orders) of Growth (Rychlosti růstu resp. řády složitosti)

Definice Nechť $\mathcal{F}$ je množina funkcí z $\mathbb{N}$ do $\mathbb{N}$. Pro danou funkci $f \in \mathcal{F}$ definujeme množiny funkcí $O(f)$, $\Theta(f)$ a $\Omega(f)$ takto

$O(f(n)) = \{g(n) \mid$ existují kladná konstanta c a n_0 takové, že $0 \leq g(n) \leq c \cdot f(n)$ pro vš. $n \geq n_0\}$

$\Theta(f(n)) = \{g(n) \mid g(n) \in \mathcal{F}$ a existují kladné konstanty c_1, c_2 a n_0 takové, že $0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$ pro vš. $n \geq n_0\}$

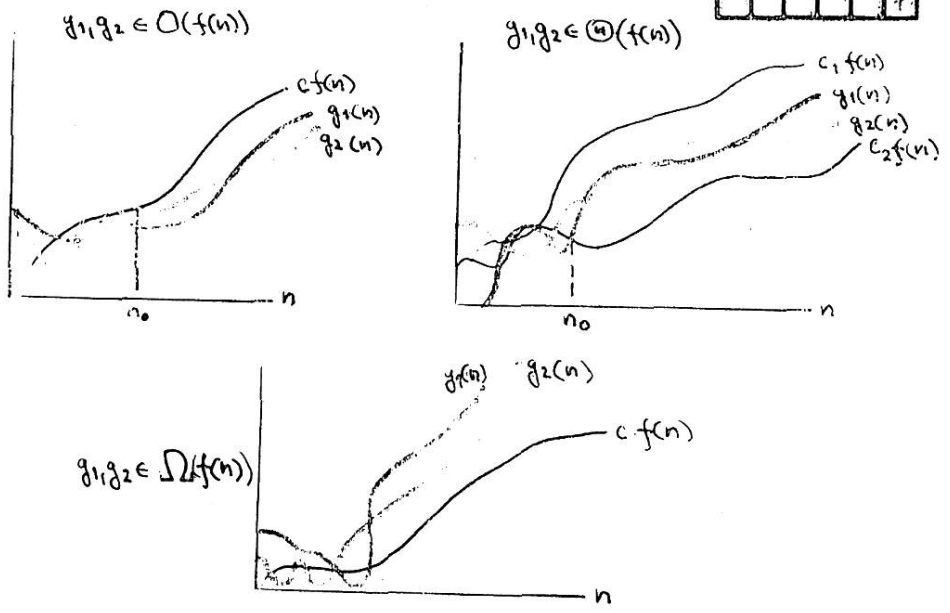
$\Omega(f(n)) = \{g(n) \mid g(n) \in \mathcal{F}$ a existují kladné konstanty c a n_0 takové, že $0 \leq c \cdot f(n) \leq g(n)$ pro vš. $n \geq n_0\}$

Množinu $O(f(n))$ nazýváme asymptotickým horním ohraničením funkce $f(n)$ (asymptotically upper bound),

$\Theta(f(n))$ – asymptotickým (oboustranným) ohraničením funkce $f(n)$ (asymptotically tight bound) a

$\Omega(f(n))$ – asymptotickým dolním ohraničením funkce $f(n)$ (asymptotically lower bound)

Situaci ilustruje následující obrázek



VĚTA : Pro lib. 2 funkce $f(n)$ a $g(n)$ z $\mathcal{F}$ je $f(n) \in \Theta(g(n))$ právě když $f(n) \in O(g(n))$ a $f(n) \in \Omega(g(n))$.

D. Plyne z definice

VĚTA : Nechť $p(n)$ je polynom stupně d. Pak $|p(n)| \in \Theta(n^d)$.

D. Cvičení

Uvedené funkce nám dávají možnost vytvořit jisté klasifikační schéma založené na uzavřené inkluzi na množinách $\Theta(f(n))$

Poněvadž, např., $\Theta(n^3) \subseteq \Theta(n^2)$, přičemž $\Theta(n^3) \leq \Theta(n^2)$, budeme problém se složitostí $\Theta(n^2)$ považovat za méně složitý než problém se složitostí $\Theta(n^3)$.

Omezení asymptotických odhadů

Koncept asymptotických odhadů složitosti byl a je úspěšně používán při klasifikaci složitosti četných problémů a jejich tříd. Přesto však není zcela obecný a univerzální.

1. Existují problémy, pro které každé jejich řešení může být do nekonečna zlepšováno tak, že každé nové řešení padá do jiné třídy složitosti. Tento výsledek je znám jako „Blum's speedup theorem" a byl objeven M. Blumem v r. 1967. Přesněji Blumův theorem říká:

> Pro každou μ-rekurzivní funkci $g: \mathbb{N} \to \mathbb{N}$ existují problém, pro který každé řešení s nějakou složitostí $t(n)$ může být zlepšeno tak, že nové řešení má složitost $g(t(n))$ skoro pro všechny hodnoty $n \in \mathbb{N}$. (a v důsledku toho další řešení má složitost $g(g(t(n)))$ atd.)

Např. nechť g je funkce $\lfloor \log_2 \rfloor$. Podle tohoto výsledku lze problém se složitostí 2^n vylepšit na problém se složitostí n, tenpak na problém se složitostí $\log_2 n$ atd.

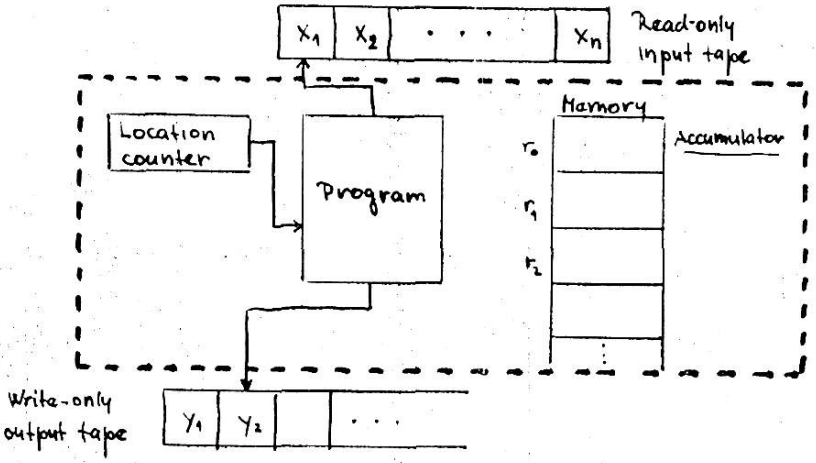
Současně známé problémy, na které lze aplikovat „urychlovací theorém" nepřekračují nepočetnost rámec teoretické informatiky, nemají praktické aplikace.

2. Největší nevýhodou klasifikace problémů podle asymp. ohraničení je jejich značná citlivost na „výpočetní prostředí", t.j. změníme-li výpočetní systém, může složitost řešení problémů padnout do jiné třídy.

Příklad: Lze dokázat, že složitost rozpoznávání řetězců 1-páskovým T. strojem je $\Theta(n^2)$. Uvažujme však 2-páskový T. stroj!

Jiné modely výpočtu

1. Random access machine – RAM model



Instrukce: (všechny pracují s r_0)

LOAD	op	READ	op
STORE	op	WRITE	op
ADD	op	JUMP	lab
SUB	op	JGTZ	lab
MULT	op	JZERO	lab
DIV	op	HALT	

Operandy:

celočíselné (stejné jako x_i, y_i)

Typy adresování:

(a) literál (=5)

(b) přímá adresa (5)

(c) nepřímá adresa (*5)

2. Random access stored program machine – RASP model

Jde o stejný model jako RAM s tou odlišností, že program je uložen v paměti; není tudíž potřeba adresování typu (c) – program se může modifikovat

VĚTA: Pro každý RAM program s časovou složitostí $T(n)$ existuje konst. k taková, že ekvivalentní RASP program má časovou složitost $k \cdot T(n)$.

Časová složitost problémů rozpoznávání jazyků

Definice: Nechť T. stroj M (jednopáskový nebo vícepáskový) vyčísluje parciální funkci $f: \Sigma_1^* \to \Sigma_2^*$. Říkáme, že M vyčísluje funkci f v polynomiálním čase, jestliže existuje polynom p(x) takový, že pro každý řetězec $w \in \Sigma_1^*$, pro který je f(w) definována, stroj M vyčíslí funkci f v nanejvýš p(|w|) krocích. Funkci f pak nazýváme funkcí vyčíslitelnou v polynomiálním čase.

VĚTA Nechť f_1, f_2 jsou funkce vyčíslitelné v polynomiálním čase. Pak funkce $f_2 \circ f_1$ je rovněž vyčíslitelná v polynomiálním čase.

Důkaz: Podle definice existuje T. stroj M_1, který vyčísluje funkci $f_1(v)$ maximálně v $p_1(|v|)$ krocích a stroj M_2 vyčíslující $f_2(w)$ v maximálně $p_2(|w|)$ krocích. Stroj $\to M_1 M_2$, který vyčísluje hodnotu $f_2 \circ f_1(v)$ pro lib. v, má následující složitost:

$$p_1(|v|) + p_2(p_1(|v|)+1) \quad \text{kde:}$$

$p_1(|v|)$ je složitost vyčíslení funkce $f_1(v)$

$p_1(|v|)+1$ maximální délka vstupu pro stroj M_2

$p_2(p_1(|v|)+1)$ je složitost vyčíslení funkce f_2 aplikované na $f_1(v)$.

Výsledný výraz pro složitost kompozice funkcí je polynom.

S využitím tohoto výsledku lze dokázat větu:

> Třída funkcí, které jsou vyčíslitelné vícepáskovými T. stroji v polynom čase, je stejná jako třída funkcí vyčíslitelných v polyn. čase 1-páskovými T. stroji

Třída P

Definice: Nechť M je T. stroj a L = L(M) jazyk, který tento stroj přijímá. Říkáme, že M přijímá jazyk L v polynomiálním čase, jestliže existuje polynom p(n) takový, že pro každé $w \in L$ je počet kroků (přechodů) potřebných k přijetí řetězce w nanejvýš roven číslu p(|w|). Třídu všech jazyků, které lze přijímat T. stroji v polynomiálním čase definujeme jako třídu P.

Pozn. Následující tabulka ilustruje rozdíl mezi polynomiální a exponenciální č. složitostí pro T. stroje M resp. M', které přijímají jazyk se složitostí $|w|^2$ resp $2^{|w|}$. $w \in L$, $|w| = n$.

	n^2	2^n
n = 10	.0001 sec	.0001 sec
20	.0004 sec	1.05 sec
30	.0009 sec.	17.92 minut
40	.0016 sec	12.74 dnů
50	.0025 sec	35.75 roků
60	.0036 sec	36.6 století
70	.0049 sec.	374.8 milionů let

1 krok ≈ 1 mikrosekunda

VĚTA: Jestliže T. stroj přijímá jazyk L v polynomiálním čase, pak existuje jiný T. stroj, který přijímá L rovněž v polynom. čase a navíc indikuje přijetí řetězce konc. obsahem pásky $\Delta Y \Delta\Delta\Delta$.

Důkaz: cvičení

VĚTA: Nechť L je jazyk, který je přijímán vícepáskovým T. strojem v polynomiálním čase. Pak L patří do třídy P.

Důkaz:

Nechť f je funkce: $f(w) = \begin{cases} Y, & \text{jestliže } w \in L \\ \text{nedef.}, & \text{jestliže } w \notin L \end{cases}$

Pozn.: Robustnost třídy P

Jazyky rozhodnutelné v polynomiálním čase

Definice: zřejmá

Otázka: V jakém vztahu jsou jazyky přijímané T. strojem v polyn. čase a jazyky rozhodnutelné T. strojem v polynom. čase ?

VĚTA:

Jestliže jazyk L může být přijat T. strojem v polynomiálním čase, pak existuje T. stroj, který rozhoduje jazyk L v polynomiálním čase.

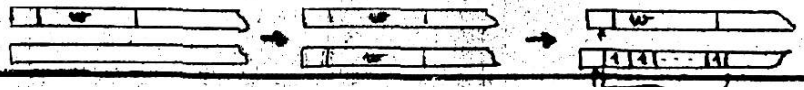
Důkaz:

Předpokládejme, že T. stroj M přijímající jazyk L nepřipouští abnormální zastavení. Pokud ano, provedeme transformaci pásky a přech. funkce:

Existuje polynom $p(n)$, který ohraničuje počet kroků $p(|w|)$ stroje M při přijetí řetězce w.

Nyní vytvořím 2-páskový T. stroj M', který bude rozhodovat jazyk L v polynomiálním čase. M' bude kompozicí strojů M_1 a M_2.

1. Stroj M_1 okopíruje vst. řetězec w na pásku 2 a pak ho nahradí řetězcem $1^{p(|w|)}$:



Lze rigorosně dokázat, že stroj M_1 pracuje v polynomiálním čase.

2. M_2 bude simulovat činnost stroje M s těmito modifikacemi:

Po každém provedení kroku M posune hlavu na 2. pásce o 1 buňku doprava. Jestliže dosáhne konc. stavu M, pak stroj M_2 končí a na 1 pásce bude $\Delta Y \Delta\Delta\Delta$

Jestliže však dříve dosáhne na 2. pásce blank, pak M_2 končí s obsahem 1. pásky $\Delta N \Delta\Delta\Delta$. Zřejmě i stroj M_2 pracuje v polynomiálním čase a tudíž

$\rightarrow$ $M_1 M_2$ rozhoduje jazyk L v polynom. čase.

Pozn.: Rozhodovací problémy (decision problems) řešitelné v polynomial. čase

Třída P obsahuje všechny rozhodovací problémy, které mohou být řešeny T. strojem v polynomiálním čase.

Časová složitost nedeterministických strojů

Definice: Říkáme, že nedeterministický T. stroj M přijímá jazyk $L = L(M)$ v polynomiálním čase, jestliže existuje polynom $p(x)$ takový, že pro každý řetězec $w \in L$ může M přijmout w v maximálně $p(|w|)$ krocích.

Třídu NP definujeme jako třídu jazyků, které mohou být přijímány nedeterministickým T. strojem v polynomiálním čase.

zřejmě platí: $P \subseteq NP$

$P \stackrel{?}{=} NP$ – je stále otevřený problém

Deterministické výpočty v polyn. čase

řešitelné rozhodovací problémy = rozhodnutelné jazyky = jazyky přijímané T. strojem

Nedeterm. výpočty v polynom. čase

řešitelné rozhodovací problémy = rozhodnutelné $\stackrel{?}{=}$ jazyky přijímané T. strojem

Polynomiální redukce (polynomiální transformace)

Definice: Polynomiální redukce (*) jazyka L_1 nad abecedou Σ_1 na jazyk L_2 nad abecedou Σ_2 je funkce $f: \Sigma_1^* \to \Sigma_2^*$, pro kterou platí:

(a) pro vš. $w \in \Sigma_1^*$: $w \in L_1 \iff f(w) \in L_2$

(b) f je Turingovsky vyčíslitelná v polynomiálním čase.

Existuje-li polynomiální redukce jazyka L_1 na L_2, říkáme, že L_1 se redukuje na L_2 a píšeme $L_1 \propto L_2$.

Věta: Je-li $L_1 \propto L_2$ a L_2 je ve třídě P, pak L_1 je ve třídě P

Důkaz: Nechť M_f je T. stroj, který provádí redukci f jazyka L_1 na L_2 a nechť $p(x)$ jeho časová složitost. Pro lib $w \in L_1$ výpočet $f(w)$ vyžaduje nanejvýš $p(|w|)$ kroků a produkuje výstup maximální délky $p(|w|) + |w|$.

Nechť M_2 přijímá jazyk L_2 v polynom. čase daném polynomem $q(x)$. Uvažujme T. stroj, který vznikne kompozicí $M_f M_2$. Tento stroj přijímá jazyk L_1 tak, že pro každé $w \in L_1$ udělá stroj $M_f M_2$ maximálně $p(|w|) + q(p(|w|) + |w|)$ kroků, což je polynom ve $|w|$ a tedy L_1 leží ve třídě P.

Příklad: Funkce $f: \{x,y\}^* \to \{x,y,z\}^*$ definovaná jako $f(v) = vzv$ je polynom. redukcí jazyka $L_1 = \{w \mid w$ je palindrom nad $\{x,y\}\}$ na jazyk $L_2 = \{wzw^R \mid w \in \{x,y\}^*\}$.

Předchozí věta nám dává praktickou možnost jak ukázat, že určitý jazyk je ve třídě P. Navíc, přeformulujeme-li tuto větu takto:

„Jestliže platí $L_1 \propto L_2$ a L_1 neleží v P, pak L_2 také neleží v P",

můžeme dokazovat, že určitý jazyk neleží v P.

Cookův theorem

Cookův theorem stanovuje „reprezentativní" NP problém.

Problém splnitelnosti – SAT problem (SATisfiability problem):

Nechť $V = \{v_1, v_2, \dots, v_n\}$ je konečná množina Boolovských proměnných (prvotních formulí výr. počtu)

Literálem nazveme každou proměnnou v_i nebo její negaci $\overline{v}_i$

Klausuli nazveme výr. formuli obsahující pouze literály spojené výr. spojkou $\vee$ (nebo)

Příklad klausulí: $v_1 \vee \overline{v}_2$

$v_2 \vee v_3$

$\overline{v}_1 \vee \overline{v}_3 \vee v_2$

SAT-problem lze formulovat takto:

> Je dána množina proměnných V a množina klausulí nad V; je tato množina klausulí splnitelná?

Každý konkrétní SAT-problém můžeme zakódovat jediným řetězcem takto: Nechť $V = \{v_1, v_2, \dots, v_m\}$, každý literál v_i zakódujeme řetězcem délky m, který obsahuje samé 0 s výjimkou i-té pozice, která obsahuje symbol 1, jde-li o literál v_i, nebo m, jakti o literál $\overline{v}_i$. Klausuli reprezentujeme

seznamem zakódovaných literálů, oddělených symbolem /. SAT-problém bude seznam klausulí uzavřených v aritm. závorkách.

Příklad:

SAT-problém obsahuje proměnné v_1, v_2, v_3 a klausule

$v_1 \vee \bar{v}_2$; $v_2 \vee v_3$; $\bar{v}_1 \vee \bar{v}_3 \vee v_2$

bude reprezentován řetězcem:

(p00/0n0)(0p0/00p)(n00/00n/0p0)

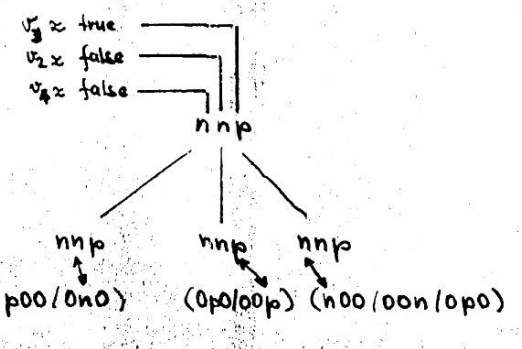
Označme L_{SAT} jazyk obsahující řetězce tohoto typu, které reprezentují splnitelné množiny klauzulí. Ohmotivovaný řetězec (např.) je prvkem L_{SAT} ($v_1 = F, v_2 = F, v_3 = T$), na rozdíl od řetězce (p00/0p0)(n00/0p0)(p00/0n0)(n00/onc který je kódem nesplnitelné mn. klauzulí $v_1 \vee v_2$; $\bar{v}_1 \vee v_2$; $v_1 \vee \bar{v}_2$; $\bar{v}_1 \vee \bar{v}_2$

Přiřazení pravd. hodnot budeme reprezentovat řetězcem z $\{p,n\}^+$, kde

p v i-té pozici představuje přiřazení $v_i \approx$ true a

n — " — $v_i \approx$ false

Pak test, zda určité ohodnocení je modelem množiny klauzulí (množina klauzulí je pro toto ohodnocení splněna), je velmi jednoduché a ilustruje ho obrázek:



Na uvedeném principu můžeme zkonstruovat nedeterministický T. stroj, který přijímá jazyk L_{SAT} v polynom. čase. Zvolíme 2-páskový T. stroj který:

1. začíná kontrolou, zda vstup reprezentuje množinu klauzulí
2. na 2. pásku nageneruje řetězec z $\{n,p\}^m$ nedeterministickým způsob
3. posouvá hlavu na 1. pásce a testuje, zda pro dané ohodnocení (na 2. pásce je množina klauzulí splnitelná

Tento proces může být snadno implementován s polynom. složitostí přijetí v závislosti na délce vst. řetězce a tedy $L_{SAT} \in NP$.

VĚTA (Cookův theorém):

Je-li L lib. jazyk z NP, pak $L \propto L_{SAT}$.

Důkaz: Protože $L \in NP$, existuje nedeterm. Turingův stroj M a polynom p(x) tak, že pro každé $w \in L$ stroj M přijímá w v maximálně p(|w|) krocích. Jádro důkazu tvoří konstrukce polynom. redukce f z L na L_{SAT}: Pro každý řetězec $w \in L$ bude f(w) množina klauzulí, které jsou splnitelné právě když M přijímá w.

NP-úplné jazyky

Po objevení Cookova theorému se ukázalo, že mnoho dalších NP jazyků má vlastnost podobnou jako L_{SAT}, t.j. jsou polynom. redukcemi ostatních NP jazyků. Tyto jazyky se nazývají NP-úplné (NPcomplete) jazyky. Kdyby se ukázalo, že lib. z těchto jazyků je v P, pak by muselo platit P=NP; naopak, důkaz, že některý z nich leží mimo P by znamenal $P \subset NP$.

Význačné NP-úplné problémy

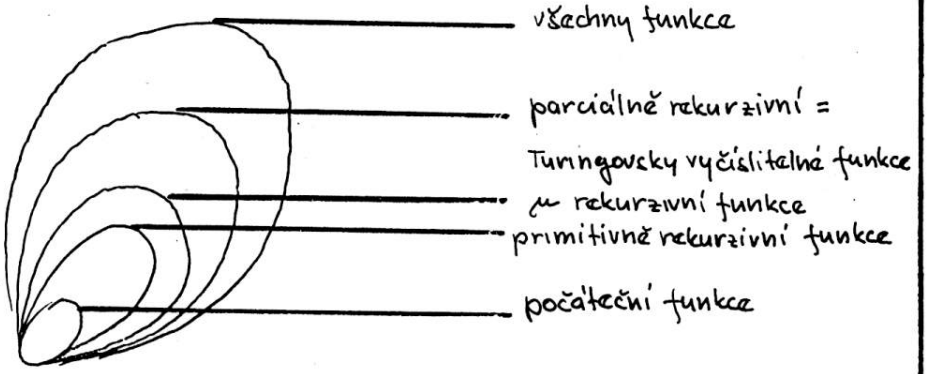
Definice: Nechť $\vec{G} = (H, U, \sigma)$ je orientovaný graf.

1. zpětnovazebná množina vrcholů (feedback vertex set) je podmnožina $S \subseteq U$ taková, že každý cyklus grafu $\vec{G}$ obsahuje nějaký vrchol z S.
2. zpětnovazebná množina hran (feedback edge set) je podmnožina $F \subseteq H$ taková, že každý cyklus v $\vec{G}$ obsahuje nějakou hranu z F.

VĚTA: Následující problémy jsou NP-úplné problémy:

1. (Satisfiability) Je booloveský výraz splnitelný?
2. (Clique) Obsahuje neor. graf kliku velikosti k?
3. (Vertex cover) Má neorient. graf dominantní množinu mohutnosti k?
4. (Hamilton circuit) Má neor. graf Hamiltonovskou kružnici.
5. (Colorability) Má neorientovaný graf chromatické číslo k?
6. (Feedback vertex set) Má orientovaný graf zpětnovazebnou množinu vrcholů mohutnosti k?
7. (Feedback edge set) — " — hran mohutnosti k?
8. (Directed Hamilton circuit) Má orient. graf Hamiltonovský cyklus?
9. (Set cover) Je dána třída množin $S_1, S_2, \ldots, S_n$. Existuje podtřída k množin $S_{i_1}, S_{i_2}, \ldots S_{i_k}$ taková, že
$$\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j$$
10. (Exact cover). Je dána třída množin $S_1, S_2, \ldots, S_n$. Existuje množinové pokrytí (set cover) tvořené podtřídou po dvojicích disjunktních množin?

Shrňme v obrázku získané informace:



Vyčíslitelnost a programovací jazyky

Triviálizovaný programovací jazyk (Bare-Bones Programming Languages)

Zavedeme maximálně zjednodušený programovací jazyk PL_{BB} s následujícími výrazovými prostředky:

Datové struktury: jednoduché proměnné typu integer označované identifikátory

Příkazy:

(a) `incr` name;
(b) `decr` name;
(c) `while` name $\neq 0$ `do`
⋮
`end`;

name je identif. proměnné typu integer

VĚTA: Každá primitivně rekurzivní funkce může být naprogramována v jazyce PL_{BB} a každý program v PL_{BB} popisuje primitivně rekurzivní funkci.

Důkaz: Cvičení

ACKERMANN FUNCTION (1928)

$A(0,y) = y+1$

$A(x,0) = A(x-1,1)$

$A(x,y) = A(x-1, A(x,y-1))$

x\y	0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	5	7	9	11	13	15	17
3	5	13	29	61	125	253	509	1021

x\y	8	9	10	11	12	13	14	15
0	9	10	11	12	13	14	15	16
1	10	11	12	13	14	15	16	17
2	19	21	23	25	27	29	31	33
3	2045	4093	8189	16381	32765	65533	131069	262141

x\y	16	17	18	19	20	21	22	23
0	17	18	19	20	21	22	23	24
1	18	19	20	21	22	23	24	25
2	35	37	39	41	43	45	47	49
3	524285	1048573	2097149	4194301	8388605	16777213	33554429	67108861

x\y	24	25	26	27	28	29	30	31
0	25	26	27	28	29	30	31	32
1	26	27	28	29	30	31	32	33
2	51	53	55	57	59	61	63	65
3	134217725	268435453	536870909	1073741821	2147483645	-	-	-

x\y	0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	5	7	9	11	13	15	17
3	5	13	29	61	125	253	509	1021
4	13	65533	**********					

S 65533

Lemma 1

Parciálně rekurzivní fce jsou programovatelné ve skeletovém jazyku.

Důkaz

Konvence: Pro výpočet fce $f: \mathbb{N}^m \to \mathbb{N}^n$ budeme používat identifikátory $x_1, x_2, \dots x_m$ pro uložení vstupních hodnot a identifikátory $z_1, z_2, \dots z_n$ pro uložení výstupních hodnot fce f.

A) Počáteční fce:

1. ξ se počítá programem `clear` z_1;
2. σ - - - - - - - $z_1 \leftarrow x_1$; `incr` z_1;
3. Π_j^m - - - - - - - $z_1 \leftarrow x_j$;

B) Primitivní rekurzivní fce

1. Kombinace $\underbrace{f: \mathbb{N}^k \to \mathbb{N}^m}_{\text{programy } F}$ $\underbrace{g: \mathbb{N}^k \to \mathbb{N}^n}_{G}$

F má vstupy $x_1 \dots x_k$ a výstupy $z_1 \dots z_m$

G - - - $x_1 \dots x_k$ - - - $z_{m+1}, \dots z_{m+n}$

F neporuší $x_1 \dots x_k$, G neporuší $z_1 \dots z_m$

$f \times g$ se počítá programem F G

Zbytky (makra):

```
clear name;

while name ≠ 0 do
    decr name;
end;
```

```
name 1 ← name 2

clear aux;
clear name 1;
while name 2 ≠ 0 do
    incr aux;
    decr name;
end;
while aux ≠ 0 do
    incr name 1;
    incr name 2;
    decr aux;
end;
```

Nechť program $\mathcal{B}$ počítá parc. rek. fci $h: \mathbb{N}^t \to \mathbb{N}^k$. Nemění-li X v $\mathcal{B}$ zpracovávána, program vždy nekončí a tedy počítá parc. rek. fci. Jinak, uvažujme fci

$$f(\bar{x}, 0) = ident(\bar{x})$$
$$f(\bar{x}, y+1) = h(f(\bar{x}, y))$$

která odpovídá y-násobnému provedení cyklu.
Počet opakování cyklu je

$$\mu y \left[\Pi_j^k \circ f(\bar{x}, y) = 0 \right]$$

Cyklus while $x \neq 0$ do $\mathcal{B}$ end tedy počítá fci $g: \mathbb{N}^k \to \mathbb{N}^k$ definovanou takto:

$$g(\bar{x}) = f\left(\bar{x}, \mu y \left[\Pi_j^k \circ f(\bar{x}, y) = 0 \right]\right).$$

A to je parciálně rek. fce.

Lemma 2

Funkce programovatelné ve skeletovém jazyku jsou parciálně rekurzivní.

Důkaz

Ať-li program k proměnných, počítá fci
$f: \mathbb{N}^k \to \mathbb{N}^k$.

4) Obsahuje-li program 1 příkaz, jsou 3 možnosti:

incr, decr — primitivně rekurzivní

while name ≠ 0 do
end

počítá fci $f(name) = \begin{cases} 0 & \text{if } name = 0 \\ \text{nedefinováno jinak.} \end{cases}$

která je ekvivalentní

$\mu z [plus(name, z) = 0]$

B) Předpokládejme, že program o $n-1$ příkazech počítá parciálně rekurzivní fci. Uvažujme program o n příkazech. Ten vznikne $\begin{cases} \text{zřetězením dvou kratších programů} & (1) \\ \text{zacyklením příkazem while.} & (2) \end{cases}$

V případě (1) jde o kompozici funkcí.

V případě (2) uvažujme program

while X ≠ 0 do
B
end;

Registr	Obsah		Význam	
100	3	}	STORE 1	uschova akumulátoru
101	1			
102	1	}	LOAD r+i	obsah RAM registru i
103	r+i			
104	5	}	ADD =r	inkrementace čísla registru
105	r			
106	3	}	STORE 111	modifikace instrukce SUB
107	111			
108	1	}	LOAD 1	obnova akumulátoru
109	1			
110	6	}	SUB b ←	b je obsah RAM registru i
111	–			(je to číslo registru s operandem)

Simulace SUB *i v RASP.

Každá RAM instrukce tedy vyžaduje maximálně 6 RASP instrukcí. Při uniformním cenovém kriteriu je tedy časová složitost RASP programu maximálně $6T(n)$.

Obdobně lze analýzou simulace nepřímého adresování při logaritmickém cenovém kriteriu zjistit, že časová složitost RASP programu je maximálně $f(r)T(n)$, přičemž $f(r)$ je pro daný program konstanta. □

Druhá věta platí i obráceně. Její důkaz se opírá o simulaci RASP programu pomocí RAM. RAM čte program z registrů a na základě jejich obsahu simuluje odpovídající RASP instrukce.

Složitost RASP programů

Lze použít { uniformní cenové kritérium => jako RAM
logaritmické c.k. => je třeba vzít v úvahu čtení instrukcí.

Například, je-li ADD =i umístěna v registrech j a j+1, cena této instrukce je $\ell(j) + \ell(c(0)) + \ell(i)$. [Uvažujeme $\ell(j)+\ell(j+1) \sim \ell(j)$]

Věta. Pro každý RAM program s čas. složitostí $T(n)$ existuje konstanta k taková, že ekvivalentní RASP program má čas. složitost $k.T(n)$, a to jak pro uniformní, tak i pro logaritmické cenové kritérium.

Důkaz. Ukážeme simulaci RAM programu P pomocí RASP programu P_S:

(1) Registr 1 RASP bude sloužit k přechodnému uschování akumulátoru RAM

(2) P_S obsadí r-1 registrů RASP (r závisí na P)

(3) $\forall i, i \geq 1$: obsah RAM registru i bude uložen v RASP registru $r+i$.

(4) Každá RAM instrukce bez nepřímé adresy je mapována přímo do odpovídající RASP instrukce (s inkrementovanými odkazy na registry).

(5) Každá RAM instrukce s nepřímou adresou je mapována do sekvence 6-ti RASP instrukcí, které simulují nepřímé adresování pomocí modifikace instrukcí.

Logaritmická prostorová složitost RAM programu je definována jako

$$\sum_{\substack{r_i \\ i \in \langle 0;\infty)}} \ell(x_i)$$

, kde x_i je největší číslo, které se v průběhu výpočtu vyskytne v registru r_i

---

Je jasné, že program může mít radikálně rozdílné složitosti pod uniformním a logaritmickým cenovým kriteriem.

Uniformní cenové kritérium má opodstatnění v případě, kdy každé číslo, které se zpracovává, je umístěno v 1 slově počítače. Jinak (obecně) je realističtější logaritmické cenové kritérium.

Stroj RASP (random access stored program machine)

Vypadá podobně jako RAM, ale program je umístěn v registrech.

Každá instrukce obsazuje 2 registry: | OP.CODE | ADDR |

Operační kód kóduje instrukci a typ adresování (i, =i).

Pozn.: *i není třeba, dá se obejít modifikací programu.

Pozn.: STORE, READ, JUMP, JGTZ, JZERO nepřipouštějí adresu =i (literál).

Programový čítač je nastavený na určitý registr. Po provedené instrukce se jeho hodnota zvýší o 2 nebo se nastaví podle operandů skokových instrukcí.

Složitost RAM programů

A. Uniformní cenové kriterium

Každá RAM instrukce vyžaduje 1 jednotku času, každý registr vyžaduje 1 jednotku prostoru.

B. Logaritmické cenové kriterium

Definujeme fci

$$l(i) = \begin{cases} \lfloor \log|i| \rfloor + 1 & \text{pro } i \neq 0 \\ 1 & \text{pro } i = 0 \end{cases}$$

Předpokládáme, že pro reprezentaci čísla n je potřeba $\lfloor \log|n| \rfloor + 1$ bitů. Protože číslo n může být libovolně velké, nelze ho zpracovat naráz, a tedy časová složitost (cena) přečtení operandu instrukce závisí na jeho hodnotě (počtu bitů):

Operand a	Cena t(a)
=i	$l(i)$
i	$l(i) + l(c(i))$
*i	$l(i) + l(c(i)) + l(c(c(i)))$

Příklady operací a jejich cen:

ADD i	$l(c(0)) + l(i) + l(c(i))$
ADD *i	$l(c(0)) + l(i) + l(c(i)) + l(c(c(i)))$
STORE i	$l(c(0)) + l(i)$
STORE *i	$l(c(0)) + l(i) + l(c(i))$

C) Parciální rekurzivní fce — minimalizace

$g: \mathbb{N}^{n+1} \to \mathbb{N}$ ... program G

$\mu y [g(\bar{x}, y) = 0]$

je počítán programem

```
clear x_{k+1};
G
while z_1 ≠ 0 do
    incr x_{k+1};
    G
end;
z_1 ← x_{n+1};
```

2. Kompozice $\underbrace{f : \mathbb{N}^k \to \mathbb{N}^\ell}_{F}$ $\underbrace{g : \mathbb{N}^\ell \to \mathbb{N}^m}_{G}$

F má vstupy $x_1 \dots x_k$ a výstupy $y_1 \dots y_\ell$

G - - - $y_1 \dots y_\ell$ - - - $z_1 \dots z_m$

$f \circ g$ se počítá programem F ; G

3. Primitivní rekurze $\underbrace{g : \mathbb{N}^k \to \mathbb{N}^m}_{G}$ $\underbrace{h : \mathbb{N}^{k+m+1} \to \mathbb{N}^m}_{H}$

$f : \mathbb{N}^{k+1} \to \mathbb{N}^m$ definované primitivní rekurzí

$$f(\bar{x}, 0) = g(\bar{x})$$
$$f(\bar{x}, y+1) = h(\bar{x}, y, f(\bar{x}, y))$$

se počítá programem

```
G
aux ← x_{k+1};
clear x_{k+1};
while aux ≠ 0 do
    x_{k+2} ← z_1 ;
    x_{k+3} ← z_2 ;
    - - - - - -
    x_{k+m+1} ← z_m ;
    H
    incr x_{k+1} ;
    decr aux ;
end;
```

n-krokový RAM program (bez vstupu) je schopen počítat číslo 2^{2^n}:

```
x := 2
x := x * x      n-krát
```

$x = 2, 4, 16, 256, \dots$

T. stroj potřebuje 2^n buněk pásky (bitů) jen k uložení takového čísla. K jeho přečtení je třeba 2^n kroků T.s., což exponenciálně převyšuje počet kroků RAM.

Pro uniformní e.k. tedy neexistuje polynomiální vazba mezi T.s. a RAM.

Ale pro logaritmické e.k. platí:

> **Věta.** RAM a RASP pod logaritmickým cenovým kriteriem a Turingův stroj jsou polynomiálně vázané modely.

Důkaz. Lze ho někde najít...

Definice. Funkce $f_1(n)$ a $f_2(n)$ jsou *polynomiálně vázané*, existují-li polynomy $p_1(x)$ a $p_2(x)$ takové, že pro všechny hodnoty n je $f_1(n) \leq p_1(f_2(n))$ a $f_2(n) \leq p_2(f_1(n))$.

Příklad: (1) $f_1(n) = 2n^2$, $f_2(n) = n^5$ jsou polynomiálně vázané, protože: $f_1(x) = 2x \Leftrightarrow 2n^2 \leq 2n^5$, $f_2(x) = x^3 \Leftrightarrow n^5 \leq (2n^2)^3$.
(2) n^2 a 2^n nejsou polyn. vázané, protože neexistuje $p(x)$ takový, že $p(n^2) \geq 2^n$, $\forall n$.

Souvislost Turingových strojů a RAM (RASP)

1. Simulace Turingova stroje pomocí RAM:

   i-tá buňka j-té pásky k-páskového T. stroje je uložena v registru $ki + j + c$, kde c je konstanta určující počet pomocných registrů (obsahují mj. pozice k hlav atd...).

   Nechť program T. stroje má složitost $T(n) \geq n$. RAM může přečíst vstup, uložit ho do registrů a simulovat T.s. v čase $O(T(n))$ v případě uniformního ekn., resp. v čase $O(T(n) \log T(n))$ pro logaritmické e.k. Obecně je tedy horní ohraničení času simulace T.s. $O(T^2(n))$.

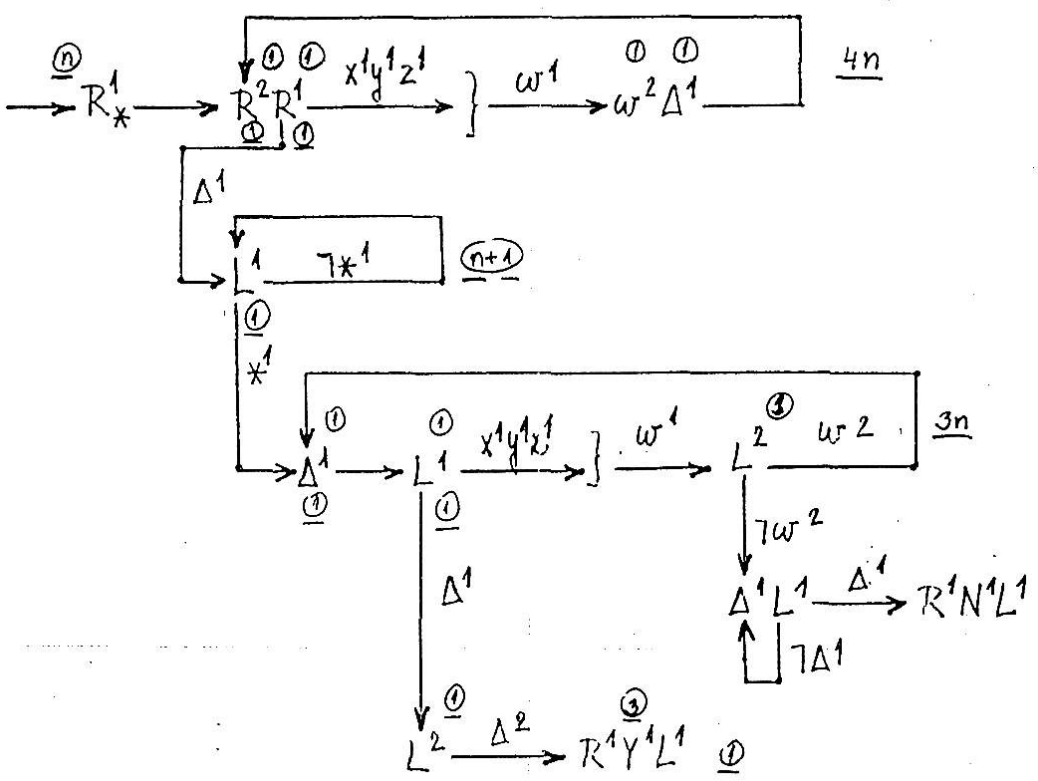
2. Simulace RAM pomocí Turingova stroje.

   Problém je v uložení čísel, které musí respektovat konečnou páskovou abecedu.

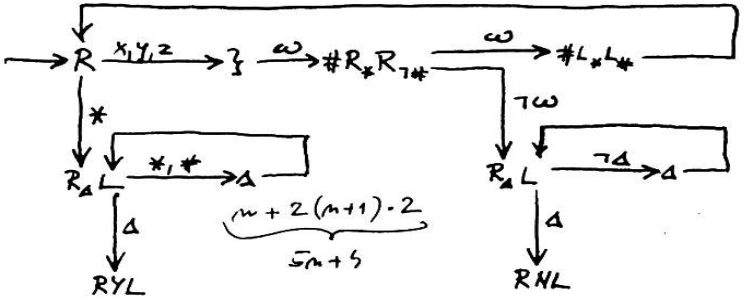
V nejhorším případě (řetězce jsou identické) stroj provede $9n + 11$ kroků.

Časová složitost problému v kontextu 2-páskových Turingových strojů:

$$\Theta(n)$$



$1+1+(n+1)+(n+1)$



$n(2n+5)+1$

$2n^2+5n+1$

TS pro srovnání řetězců stejné délky, oddělené hvězdičkou

(△ v * w △)

Nejhorší případ (shodné řetězce) :

$2n^2+10n+9 = 2n^2+5n+1$ ... srovnání řetězců

$+5n+3$ ... posuv hlavy na zač. a vymazání

$+3$ ... zápis Y

$+1$ ... příchod do konc. stavu

Průměr z shodných symbolů : $2(2n+3)+6n+10$

Nejlepší případ : $6n+10$

Algoritmus je možné zrychlit srovnáváním většího počtu symbolů najednou.

```
void insertSort ( int * a , int n )
{   int i, j, x ;

    for ( i = 1 ; i < n ; i++ )                    (n-1) krát
    {
        x = a[i] ;
        j = i ;
        while ( j > 0 && a[j-1] > x )              1+2+...+n-1
        {                                          průměr = (1+(n-1))/2 = n/2
            a[j] = a[j-1] ;
            j-- ;
        }
        a[j] = x ;
    }
}
```

Složitost algoritmu je $\underbrace{(n-1)\cdot\frac{n}{2}}_{\text{součet řady}} = \frac{n^2-n}{2} \in O(n^2)$

Příklad:

```
program InsertSort (inout:List; in:ListLength)
    var PivotPosition, I: integer;
        Pivot : ListEntryType;

    begin
      if (ListLength ≥ 2) then
          begin
            PivotPosition := 2;
            repeat                          {(n-1)-krát
                Pivot := List[PivotPosition];
                I := PivotPosition;
            while ( I > 1 and List[I-1] > Pivot ) do
                    begin
                        List[I] := List[I-1];
                        I := I - 1;
                    end
                List[I] := Pivot;
                PivotPosition := PivotPosition + 1;
            until (PivotPosition > ListLength);
          end;
    end.
```

nejhorší případ: opačně setříděný seznam

n... počet položek seznamu

počet opakování cyklu: $\underset{\text{while}}{1+2+\dots+n-1} = \frac{1+(n-1)}{2} = \frac{n}{2}$

časová složitost $\sim$ (repeat * while) $\sim (n-1)\cdot\frac{n}{2} \sim n^2 - n$

$\sim O(n^2)$