Skriptá "Výpočtová zložitosť" neobsahujú všetko, čo treba vedieť na skúške z predmetu "Výpočtová zložitosť a vypočítateľnosť" a tiež obsahujú niečo naviac, čo netreba vedieť. V ďalšom je uvedené, čo a z ktorého študijného materiálu (**skriptá a Dodatok**) treba vedieť. (Študijné poznámky netreba vôbec použiť.)

Zo skrípt treba vedieť všetko, okrem Kapitoly 3.2 (Praktické NP-úplné problémy), Dôkazu Vety 4.2.1, Kapitoly 6 (Modulárna aritmetika), Kapitoly 7 (Kolmogorovská zložitosť) a Kapitoly 8 (Informačná vzdialenosť). Z dôkazu Cook-Levinovej Vety stačí vedieť hlavné idey, treba poznať vzťah medzi reťazcami D a premennými $y_{j,s}$ a treba vedieť zostrojiť výraz H.

Z Dodatku treba vedieť všetko s nasledujúcimi obmedzeniami:

(a): Netreba vedieť odvodiť dôkaz pre odhad nepresnosti aproximačného algoritmu pre 0-1 knapsack problém (kapitola 1); stačí mu rozumieť.

(b): Stačí vedieť dôkazy iba pre F_2 a F_4 vo Vete 2 a dôkaz iba pre F_8 vo Vete 5 (kapitola 5.1).

(c): Stačí vedieť idey dôkazu Lemy B (kapitola 5.2) a Lemy D (kapitola 5.3); pre dôkaz Lemy D treba ešte vedieť, ako sa číslujú inštrukcie, T-stroje, konfigurácie a výpočty a aký je vzťah medzi funkciami Tun, $tobs$ a tvp.

(d): Tabuľku v kapitole 6 netreba vedieť, stačí jej rozumieť.

1 Aproximačný algoritmus pre 0-1 knapsack problém

V tejto časti si najprv ukážeme presný algoritmus na riešenie konštrukčného 0-1 knapsack problému, ktorého časová zložitosť môže byť na niektorých vstupoch až exponenciálna a potom ho jednoduchou úpravou prerobíme na kvalitný aproximačný algoritmus pre tento problém, ktorého časová zložitosť je vždy polynomiálna.

NP-optimalizačný 0-1 knapsack problém je daný cieľom max, reláciou R a hodnotovou funkciou m, kde:

$$R = \{(x,y)\,|\,x = d(v_1)\#\ldots\#d(v_n)\#d(w_1)\#\ldots\#d(w_n)\#d(W),$$
$$y = d(S), S \subseteq \{1,2,\ldots,n\}, \sum_{i\in S} w_i \leq W\},$$

$$m(x,y) = \sum_{i\in S} v_i,$$

pričom v_i je cena i-teho objektu, w_i je váha i-teho objektu a W je nosnosť batoha, $v_i, w_i, W \in N\,\forall i$ a $d(u)$ je binárny zápis čísla/množiny u.

Konštrukčný 0-1 knapsack problém je daný nasledovne:
Pre vstup $v_1,\ldots,v_n,w_1,\ldots,w_n,W$ (formálne, pre vstup x)
nájdi výstup $S \subseteq \{1,2,\ldots,n\}$ (formálne výstup $y = d(S)$) tak, že
$(x,y) \in R$ a $m(x,y) = \sum_{i\in S} v_i = \max_{S'\subseteq\{1,\ldots,n\}}\{\sum_{i\in S'} v_i|\sum_{i\in S'} w_i \leq W\}$.

Rozhodovací 0-1 knapsack problém je daný jazykom:
$L = \{x\#d(k)|\exists S \subseteq \{1,2,\ldots,n\} : \sum_{i\in S} w_i \leq W, \sum_{i\in S} v_i \geq k\}$.

Veta. *Jazyk L (pre 0-1 knapsack problém) je NP-úplný.*

Nasledujúci presný algoritmus pre konštrukčný 0-1 knapsack problém má dve časti. V prvej časti sa dynamickým programovaním postupne vypĺňa tabuľka $W(\cdot,\cdot)$ a indukciou na i možno dokázať, že $W(i,v)$ je váha najľahšieho výberu z prvých i objektov s cenou v. Idea indukcie je nasledovná:

(a) Ak najľahší výber z prvých $i+1$ objektov s cenou v neobsahuje $(i+1)$-vý objekt, potom takýto výber je tiež najľahší výber z prvých i objektov s cenou v a teda $W(i+1,v) = W(i,v)$.

(b) Ak od najľahšieho výberu z prvých $i+1$ objektov s cenou v odoberieme $(i+1)$-vý objekt, (ak ho obsahuje), potom dostaneme najľahší výber z prvých i objektov s cenou $v - v_{i+1}$, ktorý je o váhu w_{i+1} ľahší, teda $W(i+1, v) = W(i, v - v_{i+1}) + w_{i+1}$.

Čiže, v oboch prípadoch platí (*) z presného algoritmu nižšie.

V druhej časti presného algoritmu sa najprv zistí maximálna cena $u = \max\{v | W(n, v) \leq W\}$ optimálneho výberu S, (pozri presný algoritmus). Optimálny výber S nie je v tom čase behu algoritmu ešte známy - známa je len jeho cena a váha a potom sa v cykle postupne zisťuje, či n-tý, $(n-1)$-vý, ... objekt patrí do S a to takto:

Ak $W(n-1, u) > W(n-1, u-v_n)+w_n$, potom podľa (*), ($a$) a ($b$) dostaneme, že n-tý objekt patrí do S. Algoritmus v takom prípade pridá n do S, zníži u o v_n a analogicky pokračuje v cykle pre $(n-1)$-vý, $(n-2)$-hý, ... objekt.

Ak $W(n-1, u) \leq W(n-1, u-v_n)+w_n$, potom podľa (*), ($a$) a ($b$) dostaneme, že existuje taký optimálny výber S, do ktorého n-tý objekt nepatrí. Algoritmus v takom prípade v ďalšom taký výber S nájde a analogicky pokračuje v cykle pre $(n-1)$-vý, $(n-2)$-hý, ... objekt.

Presný algoritmus pre konštrukčný 0-1 knapsack problém

Algoritmus nájde optimálny výber $S \subseteq \{1, \dots, n\}$ taký, že:
$\sum_{i \in S} v_i = \max_{S' \subseteq \{1,\dots,n\}} \{\sum_{i \in S'} v_i | \sum_{i \in S'} w_i \leq W\}$.

(1) Vyplň tabuľku $W(0..n, 0..nV)$, kde $V = \max\{v_1, \dots, v_n\}$,
použi: $W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}$, (*)
začni s: $W(i, v) \leftarrow \infty \, \forall v, i$, ale $W(0,0) \leftarrow 0$.
($W(i, v)$ je váha najľahšieho výberu s cenou v z prvých i objektov.)

(2) $u \leftarrow \max\{v | W(n, v) \leq W\}$ (zrejme $u = \sum_{i \in S} v_i$ pre optimálny výber S, pozri vyššie)
$S \leftarrow \emptyset$
for $i \leftarrow n-1$ to 0 do (rekonštrukcia S pomocou $W(i, v)$)
  if $W(i, u) > W(i, u-v_{i+1})+w_{i+1}$ then $S \leftarrow S \cup \{i+1\}$ a $u \leftarrow u - v_{i+1}$
end

Časová zložitosť uvedeného algoritmu je zrejme $\Theta(n^2V)$, čo nie je polynomiálna zložitosť pre všetky vstupy, napríklad pre vstupy s veľkými cenami: $x = d(v_1)\#\dots\#d(v_n)\#d(w_1)\#\dots\#d(w_n)\#d(W)$, $|d(v_1)| = \cdots = |d(v_n)| = |d(w_1)| = \cdots = |d(w_n)| = |d(W)| = n$, $V = 2^{\Theta(n)}$, $|x| = (2n+1)(n+1) - 1$, $n = \Theta(\sqrt{|x|})$; v takomto prípade je časová zložitosť $\Theta(n^2V) = \Theta(n^2 2^{\Theta(n)}) = \Theta(|x| 2^{\Theta(\sqrt{|x|})})$.

Nasledujúci aproximačný algoritmus pre konštrukčný 0-1 knapsack problém zostrojíme z presného algoritmu pre tento problém jednoducho tak, že vydelíme ceny objektov vhodnou konštantou d, čím dosiahneme, že časová zložitosť takéhoto aproximačného algoritmu bude polynomiálna pre všetky vstupy a naviac, takouto úpravou tiež dosiahneme α-aproximovateľnosť 0-1 knapsack problému pre ľubovoľný koeficient $\alpha > 1$, teda jeho dobrú aproximovateľnosť.

Aproximačný algoritmus pre konštrukčný 0-1 knapsack problém

Pre ľubovoľné α, $1 < \alpha \leq 2$, algoritmus nájde aproximačný výber $\tilde{S} \subseteq \{1,\dots,n\}$ taký, že $\sum_{i\in S} v_i \leq \alpha \sum_{i\in\tilde{S}} v_i$ a $\sum_{i\in\tilde{S}} w_i \leq W$, kde $S \subseteq \{1,\dots,n\}$ je optimálny výber, pre ktorý: $\sum_{i\in S} v_i = \max_{S'\subseteq\{1,\dots,n\}}\{\sum_{i\in S'} v_i | \sum_{i\in S'} w_i \leq W\}$.

(1) $d \leftarrow \lfloor(\alpha-1)V/(n+1)\rfloor$, (kde $V = \max\{v_1,\dots,v_n\} = v_l$ pre najaké l)

(2) Nájdi optimálny výber $\tilde{S}$ pomocou **presného** algoritmu pre 0-1 knapsack problém pre **vstup:** $\tilde{v}_1,\dots,\tilde{v}_n, w_1,\dots,w_n, W$, kde

$$\tilde{v}_i = \lfloor v_i/d \rfloor \text{ pre každé } i. \tag{3}$$

(Ak $d = 0$, potom použi presný algoritmus pre vstup: $v_1,\dots,v_n$, $w_1,\dots,w_n,W$.)

end

Časová zložitosť aproximačného algoritmu:
Ak $d \geq 1$, potom časová zložitosť aproximačného algoritmu je zrejme $\Theta(n^2\tilde{V})$, kde $\tilde{V} = \max\{\tilde{v}_1,\dots,\tilde{v}_n\} = \tilde{v}_l = \lfloor v_l/d\rfloor = \lfloor V/d\rfloor \leq V/d$, (pozri časť (1) aproximačného algoritmu a (3)). Teda, táto časová zložitosť $\Theta(n^2\tilde{V}) = O(n^2V/d) = O(n^3/(\alpha-1))$ je polynomiálna, keďže $(\alpha-1)V/(2n) \leq (\alpha-1)V/(n+1) \leq d+1 \leq 2d$, (pozri časť (1) aproximačného algoritmu).

Podobne, ak $d = 0$, potom $(\alpha - 1)V < n + 1$ a preto časová zložitosť aproximačného algoritmu je $\Theta(n^2 V) = O(n^3/(\alpha - 1))$, čiže tiež polynomiálna.

Nepresnosť aproximačného algoritmu:
Našim cieľom je teraz dokázať, že pre α, S a $\tilde{S}$ z aproximačného algoritmu pre 0-1 knapsack problém platí:

$$\sum_{i \in S} v_i \leq \alpha \sum_{i \in \tilde{S}} v_i. \tag{4}$$

Z toho dôvodu najprv dokážme nasledujúcu sériu odhadov:

$$\begin{aligned}
\sum_{i \in \tilde{S}} v_i &\geq d \sum_{i \in \tilde{S}} \tilde{v}_i && \text{(podľa (3))} \\
&\geq d \sum_{i \in S} \tilde{v}_i && \text{(optimálnosť } \tilde{S} \text{ vzhľadom na } \tilde{v}_i\text{)} \\
&\geq \sum_{i \in S} (v_i - d) && \text{(podľa (3), použi } \tilde{v}_i + 1 \geq v_i/d\text{)} \\
&\geq (\sum_{i \in S} v_i) - nd && \text{(lebo } |S| \leq n\text{)}
\end{aligned}$$

a teda

$$\sum_{i \in \tilde{S}} v_i \geq (\sum_{i \in S} v_i) - nd. \tag{5}$$

Bez ujmy na všeobecnosti môžeme predpokladať, že žiadny objekt nie je ťažší, než W, (ťažšie objekty sú pre riešenie 0-1 knapsack problému nepoužiteľné) a za takého predpokladu platí:

$$\tilde{v}_j \leq \sum_{i \in \tilde{S}} \tilde{v}_i \quad \text{pre každé } j, \tag{6}$$

lebo cena jedného objektu nemôže byť väčšia, než cena optimálneho výberu. Ďalej tiež dokážme, že platí:

$$\begin{aligned}
nd/(\alpha - 1) &\leq V - d/(\alpha - 1) && \text{(pozri časť (1) aprox. algoritmu)} \\
&\leq V - d && \text{(lebo podľa predpokladu: } 1 < \alpha \leq 2\text{)} \\
&= v_l - d && \text{(pozri časť (1) aprox. algoritmu)} \\
&\leq d\lfloor v_l/d \rfloor = d\tilde{v}_l && \text{(podľa (3), použi } v_l/d \leq \lfloor v_l/d \rfloor + 1\text{)}
\end{aligned}$$

$$\begin{aligned} &\leq \; d\sum_{i\in\tilde{S}} \tilde{v}_i && \text{(podľa (6))} \\ &\leq \; \sum_{i\in\tilde{S}} v_i && \text{(podľa (3))} \end{aligned}$$

a preto

$$nd/(\alpha - 1) \leq \sum_{i\in\tilde{S}} v_i. \tag{7}$$

Z (5) a (7) dostaneme

$$\sum_{i\in S} v_i \leq nd + \sum_{i\in\tilde{S}} v_i \leq (\alpha - 1)\sum_{i\in\tilde{S}} v_i + \sum_{i\in\tilde{S}} v_i = \alpha \sum_{i\in\tilde{S}} v_i,$$

čím sme dokázali (4). Dôkaz nasledujúcej Vety teraz vyplýva z (4) a z toho, že časová zložitosť aproximačného algoritmu pre konštrukčný 0-1 knapsack problém je polynomiálna, (pozri vyššie).

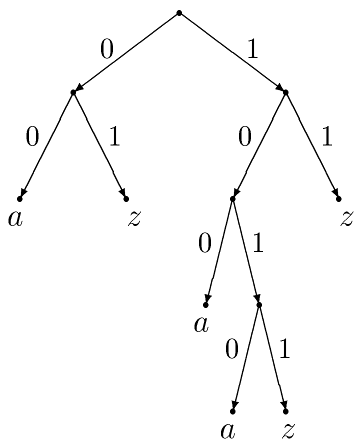
Veta. *NP-optimalizačný 0-1 knapsack problém je dobre aproximovateľný.*

2 Pravdepodobnostné algoritmy

V tejto kapitole sa oboznámime s niektorými základnými pojmami a vlastnosťami pravdepodobnostných algoritmov. Pravdepodobnostné algoritmy používajú generátory náhodných čísel, (obvykle počítajúce náhodné čísla z intervalu $<a, b>$) a v dôsledku toho môže mať pravdepodobnostný algoritmus na tom istom vstupe viacero rôznych výpočtov aj s rôznymi výsledkami. To znamená, že pravdepodobnostný algoritmus môže v niektorých výpočtoch robiť chyby, ale to nemusí z praktického hľadiska vadiť, keď je pravepodobnosť výskytu chyby (z praktického hľadiska) zanedbateľná.

Ukazuje sa, že pomocou pravdepodobnostných algoritmov možno riešiť určité typy problémov efektívnejšie, než pomocou deterministických algoritmov.

Z technických dôvodov nebudeme v ďalšom používať generátor náhodných čísel počítajúci náhodné čísla z intervalu $<a, b>$, ale namiesto toho budeme používať procedúru *brand*, ktorá vráti 0 s pravdepodobnosťou 1/2 a vráti 1 tiež s pravdepodobnosťou 1/2. (Pomocou procedúry *brand* môžeme generovať náhodné m-bitové čísla z intervalu $< 0, 1 >$ nasledovne: m-krát zavoláme *brand* s výsledkami $b_1, \ldots, b_m$ a číslo s binárnym zápisom $0.b_1 \ldots b_m$ je vygenerované náhodné číslo z intervalu $<0, 1>$.)



Na obrázku vľavo je strom pravdepodobnostných výpočtov algoritmu A na vstupe x. Vnútorné vrcholy zodpovedajú volaniam procedúry *brand*. Z vnútorného vrcholu pokračuje výpočet s pravdepodobnosťou 1/2 vľavo alebo vpravo (podľa hodnoty *brand*). V každom liste končí výpočet buď akceptovaním (a), alebo zamietnutím (z).

Rôzne pravdepodobnostné výpočty A na x zodpovedajú rôznym cestám z koreňa do listov stromu, (pozri obrázok vyššie). Teda pravdepodobnosť výskytu konkrétneho výpočtu obsahujúceho l volaní procedúry *brand* je 2^{-l}.

Označenie. Zápisom $Pr[U]$ označíme pravdepodobnosť toho, že nastala udalosť U.

Definícia 1. *Nech A je pravdepodobnostný algoritmus a x je vstup pre A. Nech $v_1, \ldots, v_m$ sú všetky pravdepodobnostné, akceptujúce výpočty A na x a nech p_i je pravdepodobnosť výskytu výpočtu v_i pre $i = 1, 2, \ldots, m$. Potom pravdepodobnoť toho, že A akceptuje x, je definovaná nasledovne:*
$Pr[A$ akceptuje $x] = \Sigma_{i=1}^{m} p_i$. Podobne, $Pr[A$ zamietne $x]$ je suma pravdepodobností výskytov všetkých zamietajúcich výpočtov A na x. Zrejme tiež platí: $Pr[A$ akceptuje $x] = 1 - Pr[A$ zamietne $x]$.

Príklad: Pre pravdepodobnostné výpočty z obrázku vyššie paltí: A akceptuje x s pravdepodobnosťou $Pr[A$ akceptuje $x] = 2^{-2} + 2^{-3} + 2^{-4} = 7/16$ a A zamietne x s pravdepodobnosťou $Pr[A$ zamietne $x] = 2^{-2}+2^{-4}+2^{-2} = 9/16$.

Definícia 2. *Hovoríme, že pravdepodobnostný algoritmus A akceptuje jazyk $L \subseteq \Sigma^*$ s chybou ϵ, $(0 \leq \epsilon < 1/2)$, ak $\forall x \in \Sigma^*$ platí: Ak $x \notin L$, potom $Pr[A$ akceptuje $x] \leq \epsilon$ a ak $x \in L$, potom $Pr[A$ zamietne $x] \leq \epsilon$.*

Definícia 3. *Hovoríme, že časová zložitosť pravdepodobnostného algoritmus A je $T(n)$, ak A vykoná v každom výpočte na každom vstupe dĺžky n najviac $T(n)$ krokov.*

Veta (o vylepšovaní). *Nech A je pravdepodobnostný algoritmus akceptujúci jazyk L v čase $T(n)$ s chybou ϵ, $(0 < \epsilon < 1/2)$. Potom pre každé ϵ', $(0 < \epsilon' < \epsilon)$, existuje pravdepodobnostný algoritmus A' akceptujúci L v čase $O(T(n))$ s chybou ϵ'.*

Dôkaz: Nech m je nepárne prirodzené číslo také, že

$$\frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2} \leq \epsilon'. \tag{1}$$

Také m existuje pre ϵ a ϵ', lebo $\lim_{n\to\infty} \beta^n = 0$ pre každé β, $0 \leq \beta < 1$. V našom prípade $\beta = 4(1-\epsilon)\epsilon = 4(1/2+\alpha)(1/2-\alpha) = 1-4\alpha^2$ pre $\alpha = 1/2-\epsilon$ a teda $0 \leq \beta < 1$, lebo $0 < \alpha < 1/2$, keďže $0 < \epsilon < 1/2$.

Pravdepodobnostný algoritmus A' na vstupe x náhodne vyberie a simuluje m pravdepodobnostných výpočtov A na x a A' akceptuje [zamietne] x, ak je väčšina náhodne vybraných výpočtov akceptujúcich [zamietajúcich].

Dokážme, že platí (2) a (3):

$$\text{Ak } x \in L, \text{ potom } Pr[A' \text{ zamietne } x] \leq \frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2}, \tag{2}$$

$$\text{ak } x \notin L, \text{ potom } Pr[A' \text{ akceptuje } x] \leq \frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2}. \tag{3}$$

Uvažujme dva prípady.

Prípad 1. Nech $x \in L$. Nech ϵ_x je pravdepodobnosť toho, že náhodne vybraný výpočet algoritmu A na x je zamietajúci. Teda $0 \leq \epsilon_x \leq \epsilon$ a $1 - \epsilon_x$ je pravdepodobnosť toho, že náhodne vybraný výpočet algoritmu A na x je akceptujúci. Pre jednoduchosť uvažujme nasledujúci príklad. Nech p je pravdepodobnosť toho, že postupnosť 5 náhodne vybraných výpočtov algoritmu A na x je tvaru: a, z, z, a, z, kde a je akceptujúci a z je zamietajúci výpočet. Zrejme $p = (1-\epsilon_x)\cdot\epsilon_x\cdot\epsilon_x\cdot(1-\epsilon_x)\cdot\epsilon_x = (1-\epsilon_x)^2\epsilon_x^3$. Keďže počet postupností dĺžky 5 s dvomi a a tromi z je $\binom{5}{2}$, potom $\binom{5}{2}(1-\epsilon_x)^2\epsilon_x^3$ je pravdepodobnosť toho, že postupnosť 5 náhodne vybraných výpočtov algoritmu A na x obsahuje 2 akceptujúce a 3 zamietajúce výpočty. Zovšeobecnením dostaneme, že $S_x = \Sigma_{j=0}^{\lfloor m/2 \rfloor}\binom{m}{j}(1-\epsilon_x)^j\epsilon_x^{m-j}$ je pravdepodobnosť toho, že postupnosť m náhodne vybraných výpočtov algoritmu A na x obsahuje najviac $\lfloor m/2 \rfloor$ akceptujúcich výpočtov, (t.j., obsahuje viac zamietajúcich, než akceptujúcich výpočtov). Teda, $Pr[A' \text{ zamietne } x] = S_x$ pre $x \in L$ a dokážme teraz pomocou tejto rovnosti, že platí (2).

Ak $\epsilon_x = 0$ potom $\epsilon_x^{m-j} = 0$ pre každé $j = 0, 1, \ldots, \lfloor m/2 \rfloor$ a preto $Pr[A' \text{ zamietne } x] = S_x = 0$ pre $x \in L$. Teda (2) platí pre $\epsilon_x = 0$. Nech teraz $\epsilon_x > 0$. Potom, podľa predpokladu platí $0 < \epsilon_x \leq \epsilon < 1/2$, z čoho dostaneme (4) a (5):

$$1 < (1-\epsilon_x)/\epsilon_x, \tag{4}$$

$$(1-\epsilon_x)\epsilon_x \leq (1-\epsilon_x)\epsilon_x + \underbrace{(\epsilon-\epsilon_x)}_{\geq 0}\underbrace{(1-\epsilon-\epsilon_x)}_{>0} = (1-\epsilon)\epsilon. \tag{5}$$

Preto

$$\begin{aligned} S_x &\leq \sum_{j=0}^{\lfloor m/2 \rfloor}\binom{m}{j}(1-\epsilon_x)^j\epsilon_x^{m-j}((1-\epsilon_x)/\epsilon_x)^{m/2-j} \qquad \text{(podľa (4))} \\ &= (1-\epsilon_x)^{m/2}\epsilon_x^{m/2}(\sum_{j=0}^{\lfloor m/2 \rfloor}\binom{m}{j}) \end{aligned}$$

$$= \frac{1}{2}(2^2(1-\epsilon_x)\epsilon_x)^{m/2} \qquad \text{(lebo } \sum_{j=0}^{\lfloor m/2 \rfloor} \binom{m}{j} = 2^m/2)$$

$$\leq \frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2} \qquad \text{(podľa (5))}.$$

Teda $Pr[A'$ zamietne $x] = S_x \leq \frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2}$ pre $x \in L$, čiže (2) platí aj pre $\epsilon_x > 0$.
Prípad 2. Nech $x \notin L$. V tomto prípade možno dokázať (3) podobne, ako sme dokázali (2) pre Prípad 1.

Tverdenie Vety dostaneme teraz z (1), (2) a (3). □

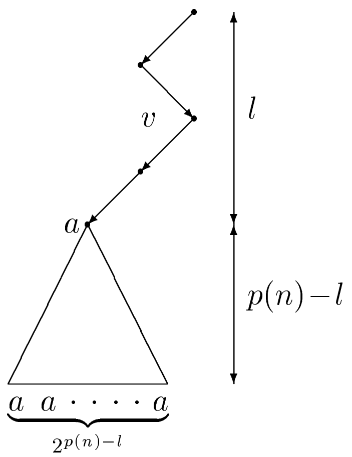
Dôsledok. *Nech A je pravdepodobnostný algoritmus akceptujúci jazyk $L \subseteq \Sigma^*$ s chybou ϵ, $(0 < \epsilon < 1/2)$. Nech A' je pravdepodobnostný algoritmus, ktorý na vstupe $x \in \Sigma^*$ dĺžky n vykoná $2n+1$ náhodne vybraných výpočtov A na x, pričom A' akceptuje [zamietne] x, ak je väčšina vybraných výpočtov akceptujúcich [zamietajúcich]. Potom pravdepodobnosť toho, že A' vykoná na $x \in \Sigma^*$ dĺžky n chybný výpočet, (t.j., akceptujúci, ak $x \notin L$ alebo zamietajúci, ak $x \in L$), je najviac $\frac{1}{2}(4(1-\epsilon)\epsilon)^{n+\frac{1}{2}}$.*

Dôkaz: Stačí dokázať (rovnakým spôsobom) tvrdenia (2) a (3) z dôkazu Vety o vylepšovaní pre $m = 2n+1$. □

Veta. *Nech $L \subseteq \{0,1\}^*$ je ľubovoľný jazyk, pre ktorý existuje pravdepodobnostný algoritmus, ktorý akceptuje L v polynomiálnom čase s chybou ϵ, $(0 < \epsilon < 1/2)$. Potom existuje polynóm $p(n)$ a pravdepodobnostný algoritmus A s polynomiálnou časovou zložitosťou tak, že pre každé n existuje postupnosť $b^n = b_1^n, b_2^n, \ldots, b_{p(n)}^n$, kde $b_i^n \in \{0,1\} \forall i$, taká, že $\forall x \in \{0,1\}^n$ platí: Výpočet algoritmu A na x, v ktorom výsledok i-teho volania procedúry $brand$ je b_i pre každé $i = 1, 2, \ldots, p(n)$, je akceptujúci, ak $x \in L$ a je zamietajúci, ak $x \notin L$, (t.j., A poznajúc b^n neurobí chybu na žiadnom $x \in \{0,1\}^n$).*

Dôkaz: Podľa Vety o vylepšovaní existuje pre L pravdepodobnostný algoritmus A_1 s polynomiálnou časovou zložitosťou akceptujúci L s chybou $\epsilon' = \frac{1}{7}$. Zrejme existuje polynóm $p(n)$ a pravdepodobnostný algoritmus A_2 s časovou zložitosőu $p(n)$ taký, že A_2 na vstupe $x \in \{0,1\}^n$ náhodne vyberie a simuluje $2n+1$ výpočtov algoritmu A_1 na x a A_2 akceptuje [zamietne] x, ak je väčšina vybraných výpočtov akceptujúcich [zamietajúcich].

Upravme algoritmus A_2 tak, aby tesne pred akceptovaním [resp. zamietnutím] vstupu dĺžky n vykonal ešte toľko volaní procedúry *brand*, aby ich celkový počet bol $p(n)$ a až potom, aby vstup akceptoval [resp. zamietol]. Nech A označuje takto upravený algoritmus A_2.



Na obrázku vľavo je znázornený akceptujúci výpočet v algoritmu A_2 na vstupe x dĺžky n, ktorý obsahuje l volaní procedúry *brand*. Po predĺžení výpočtu v o ďalších $p(n) - l$ volaní procedúry *brand* vznikne $2^{p(n)-l}$ akceptujúcich výpočtov algoritmu A na x a každý z nich obsahuje $p(n)$ volaní procedúry *brand*.

Fakt 1. *Nech $x \in \{0,1\}^n$. Potom všetkých chybných výpočtov (t.j. akceptujúcich, ak $x \notin L$ a zamietajúcich, ak $x \in L$) algoritmu A na x je najviac $2^{p(n)} \cdot (\frac{1}{2})^{n+\frac{3}{2}}$.*

Dôkaz: Nech $v_1, \ldots, v_r$ sú všetky chybné výpočty algoritmu A_2 na x a nech l_i je počet volaní procedúry *brand* vo výpočte v_i pre $i = 1, 2, \ldots, r$. Keďže A_2 vykoná na x výpočet v_i s pravdepodobnosťou 2^{-l_i}, potom $\Sigma_{i=1}^{r} 2^{-l_i}$ je pravdepodobnosť toho, že A_2 vykoná na x chybný výpočet a táto pravdepodobnosť je podľa Dôsledku Vety o vylepšovaní najviac $\frac{1}{2}(4(1-\frac{1}{7})\frac{1}{7})^{n+\frac{1}{2}} = \frac{1}{2}(\frac{24}{49})^{n+\frac{1}{2}} < (\frac{1}{2})^{n+\frac{3}{2}}$. Teda $\Sigma_{i=1}^{r} 2^{-l_i} < (\frac{1}{2})^{n+\frac{3}{2}}$.

Keďže predĺžením chybného (t.j. akceptujúceho, ak $x \notin L$ a zamietajúceho, ak $x \in L$) výpočtu v_i o ďalších $p(n) - l_i$ volaní procedúry *brand* vznikne $2^{p(n)-l_i}$ chybných výpočtov A na x, (pozri text pri obrázku vyššie), potom všetkých chybných výpočtov A na x je $\Sigma_{i=1}^{r} 2^{p(n)-l_i} = 2^{p(n)} \cdot \Sigma_{i=1}^{r} 2^{-l_i} < 2^{p(n)} \cdot (\frac{1}{2})^{n+\frac{3}{2}}$; posledná nerovnosť vyplýva z toho, že $\Sigma_{i=1}^{r} 2^{-l_i} < (\frac{1}{2})^{n+\frac{3}{2}}$, (pozri vyššie). $\square$

Nech $x \in \{0,1\}^n$ a nech u je ľubovoľný výpočet algoritmu A na x. Budeme hovoriť, že postupnosť $b = b_1, b_2, \ldots, b_{p(n)}$, kde $b_i \in \{0,1\}$ $\forall i$, sa vyskytuje vo výpočte u, ak výsledok i-teho volania procedúry *brand* vo výpočte u je b_i $\forall i$. Keďže takýchto postupností b je $2^{p(n)}$, čo je viac, než všetkých chybných výpočtov algoritmu A na všetkých $x \in \{0,1\}^n$, (lebo tých je -

podľa Faktu 1 - najviac $2^n \cdot 2^{p(n)} \cdot (\frac{1}{2})^{n+\frac{3}{2}} < 2^{p(n)}$), musí existovať postupnosť $b^n = b_1^n, b_2^n, \ldots, b_{p(n)}^n$, ktorá sa nevyskytuje v žiadnom chybnom výpočte algoritmu A na žiadnom $x \in \{0,1\}^n$. $\Box$

Definícia 4. *Pravdepodobnostný Turingov stroj je podobný nedeterministickému Turingovmu stroju, ktorý má v každej situácii najviac dve možnosti, ako pokračovať vo výpočte a ak má práve dve možnosti, potom každá nastane s pravdepodobnosťou $\frac{1}{2}$; toto zodpovedá volaniu procedúry brand. Naviac, každý konečný výpočet skončí akceptovaním alebo zamietnutím vstupu.*

Je zrejmé, že takto definované Turingove stroje môžeme povazovať za špeciálny druh pravdepodobnostných algoritmov a teda môžeme na ne aplikovať definície 1 až 3.

Definícia 5. *BPP je trieda jazykov, ktoré možno akceptovať pravdepodobnostnými Turingovými strojmi v polynomiálnom čase s nejakou chybou $0 \leq \epsilon < \frac{1}{2}$.*

Veta. $P \subseteq BPP \subseteq PSPACE$.

Dôkaz: Vzťah $P \subseteq BPP$ je zrejmý z definície 4. Dokážme teraz, že $BPP \subseteq PSPACE$. Nech $L \in BPP$ a nech M je pravdepodobnostný Turingov stroj akceptujúci L v nejakom polynomiálnom čase $p(n)$ s nejakou chybou $0 \leq \epsilon < \frac{1}{2}$. Nech M' je deterministický Turingov stroj s pamäťovou zložitosťou $p(n)$, ktorý bude akceptovať L nasledovne: M' na vstupe x dĺžky n postupne generuje (v lexikografickom poradí) na niektorej pracovnej páske všetky binárne slová dĺžky najviac $p(n)$ a po vygenerovaní každého slova simuluje M' taký výpočet T-stroja M na x, v ktorom postupnosť výsledkov volaní procedúry *brand* zodpovedá aktuálne vygenerovanému slovu. Naviac, v priebehu simulovania všetkých takýchto výpočtov vypočíta M' hodnotu $Pr[M \text{ akceptuje } x]$ a potom M akceptuje x práve vtedy, keď $Pr[M \text{ akceptuje } x] \geq 1 - \epsilon$. Keďže každému vygenerovanému slovu nemusí zodpovedať nejaký výpočet M na x , lebo dĺžka vygenerovaného slova môže byť väčšia alebo menšia, než počet volaní procedúry *brand* v simulovanom výpočte, musí M' takéto simulované výpočty ignorovať, aby vypočítal korektnú hodnotu $Pr[M \text{ akceptuje } x]$. $\Box$

Poznámka:
- vieme, že platí $P \subseteq NP \subseteq PSPACE$
- nie je známy žiadny vzťah medzi NP a BPP
- nie je známe, či $P = BPP$ alebo, či $BPP = PSPACE$

3 Aplikácie RSA šifrovacieho systému

Digitálny podpis.

Dvojica $(M, S_U(M))$ je digitálne podpísaný dokument M účastníkom U, kde S_U je tajný kľúč účastníka U. Keďže verejný kľúč P_U účastníka U je známy, môžno zistiť pravosť podpisu overiac rovnosť $M \stackrel{?}{=} P_U(S_U(M))$.

Utajené riešenie NP-úplného problému.

RSA šifrovací systém umožňuje riešiť aj nasledujúci problém: Predpokladajme, že osoba A chce presvedčiť osobu B o tom, že vie zafarbiť (ak je to možné) vrcholy ľubovoľného grafu $G = (V, E)$ tromi farbami tak, aby žiadne dva vrcholy spojené hranou namali rovnakú farbu. (Problém vrcholového farbenia grafov tromi farbami je NP-úplný.) Osoba B sa však nesmie dozvedieť, ako graf G zafarbiť. Riešenie je nasledujúce:

(1) Osoba A prefarbí vrcholy grafu G podľa náhodne vybranej permutácie troch farieb. (Nech čísla 0,1,2 označujú tieto tri farby).

(2) Osoba A vytvorí verejný a tajný RSA kľúč P_i a S_i pre každé $i \in V$. (P_i resp. S_i je určney dvojicou (e_i, n_i) resp. (d_i, n_i), kde $n_i = p_i \cdot q_i$.)

(3) Nech $b_i \in \{0, 1, 2\}$ je farba vrcholu $i \in V$. Osoba A vyberie náhodné prirodzené číslo $x_i < n_i/3$. Nech $y_i = P_i(3x_i + b_i)$. Pre každé $i \in V$ oznámi osoba A osobe B dvojicu (P_i, y_i).

(4) Potom osoba B vyberie náhodne hranu $(i, j) \in E$ a požiada osobu A, aby oznámila kľúče S_i a S_j na zistenie, či $b_i \stackrel{?}{=} b_j$, kde
$b_i = S_i(y_i) \bmod 3 = (3x_i + b_i) \bmod 3$,
$b_j = S_j(y_j) \bmod 3 = (3x_j + b_j) \bmod 3$.

(5) Uvedený proces opakujú osoby A a B $k|E|$-krát.

Ak A klame, (t.j., ak G nemožno zafarbiť tromi farbami), potom pravdepodobnosť toho, že B neodhalí klamstvo, teda, B nevyberie (v bode (4)) hranu s rovnako zafarbenými vrcholmi počas $k|E|$ výberov, je najviac $(\frac{|E|-1}{|E|})^{k|E|} = (1 - \frac{1}{|E|})^{k|E|} \leq e^{-k}$, (pripomeňme: $e^{-k} = \lim_{n\to\infty}(1 - \frac{1}{n})^{kn}$).

Ak B nevie v rozumne krátkom čase dešifrovať zašifrované farby y_l, potom sa B dozvie o farbení grafu G len toľko, že vrcholy náhodne vybratej hrany (v bode (4)) majú rôznu farbu, keďže A prefarbí (v bode (1)) vrcholy grafu podľa náhodnej permutácie troch farieb.

4 PSPACE

$PSPACE = \bigcup_{k>0} DSPACE(n^k)$.

Definícia. *Jazyk $L \subseteq \Sigma^*$ je polynomiálne transformovateľný na jazyk $l_0 \subseteq \Sigma_0^*$, ak existuje deterministický T-stroj s polynomiálnou časovou zložitosťou, ktorý vstup $w \in \Sigma^*$ pretransformuje na výstup $w' \in \Sigma_0^*$, taký, že $w \in L$ iff $w' \in L_0$.*

Definícia. *Jazyk L_0 je PSPACE-úplný, ak $L_0 \in PSPACE$ a každý $L \in PSPACE$ je polynomiálne tranformovateľný na L_0 .*

Definícia. *Booleovská formula s kvantifikátormi je úplne kvantifikovaná, ak každá jej premenná je v oblasti pôsobnosti niektorého kvantifikátora.*

Príklad. $\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y))$ je úplne kvantifikovaná formula.

Poznámka. Každú úplne kvantifikovanú formulu možno napísať v tvare $Q_1 x_1 Q_2 x_2 \dots Q_n x_n\ \phi(x_1, x_2, \dots, x_n)$, kde každé Q_i je nejaký kvantifikátor a $\phi(x_1, \dots, x_n)$ neobsahuje žiadny kvantifikátor.

Nech $TQBF = \{< \Phi > | \Phi$ je pravdivá, úplne kvantifikovaná formula$\}$.
($< \Phi >$ je kód formuly Φ; formuly s kvantifikátormi sú kódované podobne, ako formuly v jazyku SAT.)

Veta. *Jazyk TQBF je PSPACE-úplný.*

Dôkaz. Jazyk $TQBF$ patrí do $PSPACE$, lebo nasledujúci deterministický algoritmus rozpoznáva $TQBF$ s lineárnou pamäťou.

Algoritmus pre TQBF:

(1) Ak Φ nemá premenné, (t.j. má len konštanty), ohodnoť Φ a akceptuj Φ, ak je pravdivá; inak zamietni Φ.

(2) Ak Φ je tvaru $\exists x \Psi$, rekurzívne 2-krát volaj algoritmus pre vstup Ψ; raz pre $x = 0$ a raz pre $x = 1$. Ak jeden z výpočtov akceptuje, potom akceptuj Φ, inak zamietni.

(3) Ak Φ je tvaru $\forall x \Psi$, postupuj podobne ako v (2).

Pamäťová zložitosť algoritmu je lineárna, lebo hĺbka rekurzie nepresahuje počet premenných, čo je najviac dĺžka kódu formuly a to je dĺžka vstupu. Naviac, na každej úrovni rekurzie stačí v zásobníku blok rozsahu $O(1)$. Preto je celková pamäťová zložitosť lineárna a teda $TQBF \in PSPACE$.

Nech $L \in PSPACE$, $L \subseteq \Sigma^*$. Dokážme teraz, že L je polynomiálne transformovateľný na $TQBF$ (a tým dokážeme, že $TQBF$ je $PSPACE$-úplný).

Podobne, ako v dôkaze Cook-Levinovej Vety možno ukázať, že pre L existuje polynóm $S(n)$ a deterministický T-stroj M akceptujúci L, ktorý nemá pracovné pásky, ale môže prepisovať symboly na vstupnej (smerom doprava nekonečnej) páske, pričom pri rozpoznávaní vstupu dĺžky n môže použiť úsek pásky dĺžky najviac $S(n)$. A podobne, ako v dôkaze Vety o pamäťovej hierarchii možno dokázať, že T-stroj M sa na vstupe dĺžky n dostane najviac do $q(S(n)+1)t^{S(n)} \leq c^{S(n)} \leq 2^{rS(n)}$ rôznych konfigurácií, pre vhodné $c, r \geq 1$, kde q je počet stavov a t je počet páskových symbolov T-stroja M. Teda čas výpočtu T-stroja M na vstupe dĺžky n je najviac $2^{rS(n)}$.

Každú konfiguráciu, do ktorej sa dostane T-stroj M na vstupe dĺžky n, možno zakódovať pomocou $m = (S(n)+2)(t+q)$ booleovských premenných $x_1, \ldots, x_m$ (priradením hodnôt 0/1 týmto premenným) podobným spôsobom, ako je zakódovaný hociktorý reťazec D_i (v dôkaze Cook-Levinovej Vety) pomocou $(p(n)+2)|\Sigma' \cup Q|$ booleovských premenných $y_{j,s}$, kde $i(p(n)+2)+1 \leq j \leq (i+1)(p(n)+2)$ a $s \in \Sigma' \cup Q$. Také priradenie hodnôt 0/1 booleovským premenným $x_1, \ldots, x_m$, ktoré reprezentuje konfiguráciu C, budeme nazývať booleovský kód konfigurácie C a budeme ho označovať $\tilde{C}$.

Našim cieľom je pre každé $l = 1, 2, 4, 8, \ldots, 2^{rS(n)}$ zostrojiť booleovskú formulu $\Phi^l(\tilde{x}, \tilde{y})$, kde $\tilde{x} = x_1, \ldots, x_m$ [resp. $\tilde{y} = y_1, \ldots, y_m$] je postupnosť booleovských premenných pre kódovanie konfigurácie C_1 [resp. C_2], pričom bude platiť, že $\Phi^l(\tilde{C}_1, \tilde{C}_2)$ je úplne kvantifikovaná a je pravdivá *iff* M prejde z C_1 do C_2 počas najviac l krokov, kde $\tilde{C}_1$ [resp. $\tilde{C}_2$] je booleovský kód konfigurácie C_1 [resp. C_2]. Taktiež uvidíme, že každé slovo $w \in \Sigma^*$ dĺžky n možno transformovať vhodným deterministickým T-strojom v polynomiálnom čase (vzhľadom na n) na kód úplne kvantifikovanej formuly $\Phi^{2^{rS(n)}}(\tilde{C}_0^w, \tilde{C}_{acc})$, kde $\tilde{C}_0^w$, [resp. $\tilde{C}_{acc}$] je booleovský kód počiatočnej konfigurácie C_0^w pre vstup w, [resp. akceptačnej konfigurácie C_{acc}]. Vyššie uvedené tvrdenia potom použijeme na dôkaz polynomiálnej transformovateľnosti jazyka L na jazyk $TQBF$.

Konštrukcia $\Phi^1(\tilde{x},\tilde{y})$, kde $\tilde{x} = x_1,\ldots,x_m$ [resp. $\tilde{y} = y_1,\ldots,y_m$] je postupnosť booleovských premenných pre kódovanie konfigurácie C_1 [resp. C_2]. Použijeme podobné formuly, ako H_i^*, H_i^{**}, (pre $i = 0$) a F_j z dôkazu Cook-Levinovej Vety, pričom polynóm $p(n)$ je nahradený polynómom $S(n)$ a premenné $y_{j,s}$ (pre $i(S(n)+2)+1 \le j \le (i+1)(S(n)+2)$) sú nahradené premennými $x_1,\ldots,x_m$ a premenné $y_{j,s}$ (pre $(i+1)(S(n)+2)+1 \le j \le (i+2)(S(n)+2)$) sú nahradené premennými $y_1,\ldots,y_m$:

$$\Phi^1(\tilde{x},\tilde{y}) = ((H_i^* \wedge H_i^{**}) \vee (\tilde{x} \equiv \tilde{y})) \wedge (\bigwedge_{i(S(n)+2)+1\le j\le(i+2)(S(n)+2)} F_j).$$

Podformula $H_i^* \wedge H_i^{**}$ kontroluje, či M prejde v 1 kroku z konfigurácie C_1 do konfigurácie C_2, podformula $\tilde{x} \equiv \tilde{y}$ kontroluje, či $C_1 = C_2$ a podformula

$$\bigwedge_{i(S(n)+2)+1\le j\le(i+2)(S(n)+2)} F_j$$

kontroluje zmysluplnosť kódovania.

Konštrukcia $\Phi^{2l}(\tilde{u},\tilde{v})$ pomocou $\Phi^l(\tilde{x},\tilde{y})$. Keďže M prejde z konfigurácie C_1 do konfigurácie C_2 počas najviac $2l$ krokov *iff* M prejde z C_1 do nejakej konfigurácie C_3 počas najviac l krokov a z C_3 do C_2 počas najviac l krokov, ponúka sa nasledujúce tzv. dlhé riešenie.
Dlhé riešenie:

$$\Phi^{2l}(\tilde{u},\tilde{v}) = \exists\tilde{z}(\Phi^l(\tilde{u},\tilde{z}) \wedge \Phi^l(\tilde{z},\tilde{v})),$$

pre $l = 1,2,4,8,\ldots,2^{rS(n)-1}$ a kde $\tilde{u} = u_1,\ldots,u_m$, resp. $\tilde{v} = v_1,\ldots,v_m$, resp. $\tilde{z} = z_1,\ldots,z_m$ je postupnosť booleovských premenných pre kódovanie konfigurácie C_1 resp. C_2 resp C_3. Ale počet výskytov booleovských premenných v formule $\Phi^{2^{rS(n)}}(\tilde{u},\tilde{v})$ je aspoň $2^{rS(n)}$, lebo počet výskytov booleovských premenných v formule $\Phi^{2l}(\tilde{u},\tilde{v})$ je aspoň 2-krát taký, ako v $\Phi^l(\tilde{x},\tilde{y})$ pre každé $l = 1,2,\ldots,2^{rS(n)-1}$ a $\Phi^1(\tilde{x},\tilde{y})$ obsahuje aspoň jednu booleovskú premennú. To ale znamená, že takéto riešenie je nepoužiteľné, keďže slovo $w \in \Sigma^*$ dĺžky n nemožno transformovať v polynomiálnom čase (vzhľadom na n) na kód formuly $\Phi^{2^{rS(n)}}(\tilde{C}_0^w,\tilde{C}_{acc})$, ktorého dĺžka je zrejme tiež aspoň $2^{rS(n)}$.
Krátke riešenie:

$$\Phi^{2l}(\tilde{u},\tilde{v}) = \exists\tilde{z}\,\forall\tilde{x}\,\forall\tilde{y}((((\tilde{x}\equiv\tilde{u}) \wedge (\tilde{y}\equiv\tilde{z})) \vee ((\tilde{x}\equiv\tilde{z}) \wedge (\tilde{y}\equiv\tilde{v}))) \Rightarrow \Phi^l(\tilde{x},\tilde{y})),$$

pre $l = 1, 2, 4, 8, \dots, 2^{rS(n)-1}$ a kde $\tilde{u} = u_1, \dots, u_m$, resp. $\tilde{v} = v_1, \dots, v_m$, resp. $\tilde{z} = z_1, \dots, z_m$ je postupnosť booleovských premenných pre kódovanie konfigurácie C_1 resp. C_2 resp C_3 a $\tilde{x} = x_1, \dots, x_m$ a $\tilde{y} = y_1, \dots, y_m$ sú pomocné postupnosti booleovských premenných. Počet výskytov booleovských premenných v formule $\Phi^{2^{rS(n)}}(\tilde{u}, \tilde{v})$ je $O(11m \cdot (rS(n)-1)) + O(S(n)) = O(S^2(n))$, teda polynomiálny (vzhľadom na n), lebo počet výskytov booleovských premenných v formule $\Phi^{2l}(\tilde{u}, \tilde{v})$ je o 11m väčší, než v $\Phi^{l}(\tilde{x}, \tilde{y})$ pre každé $l = 1, 2, \dots, 2^{rS(n)-1}$ a počet výskytov booleovských premenných v formule $\Phi^{1}(\tilde{x}, \tilde{y})$ je $O(m) = O(S(n))$. To ale znamená, že takéto riešenie je použiteľné, keďže je relatívne jednoduché a slovo $w \in \Sigma^*$ dĺžky n možno transformovať vhodným deterministickým T-strojom v polynomiálnom čase (vzhľadom na n) na kód formuly $\Phi^{2^{rS(n)}}(\tilde{C}_0^w, \tilde{C}_{acc})$, ktorého dĺžka je zrejme tiež polynomiálna (vzhľadom na n).

To, že L je polynomiálne transformovateľný na $TQBF$, vyplynie teraz z nasledujúcich úvah: Indukciou na l (postupne pre $l = 1, 2, 4, 8, \dots, 2^{rS(n)}$) možno dokázať, že ak M prejde z nejakej konfigurácie C_1 do nejakej konfigurácie C_2 počas najviac l krokov, potom je formula $\Phi^{l}(\tilde{C}_1, \tilde{C}_2)$ pravdivá a úplne kvantifikovaná, kde $\tilde{C}_1$ [resp. $\tilde{C}_2$] je booleovský kód konfigurácie C_1 [resp. C_2]. Indukciou na takéto l možno tiež dokázať, že ak je formula $\Phi^{l}(\bar{D}_1, \bar{D}_2)$ pravdivá a úplne kvantifikovaná, pričom $\bar{D}_1$ je booleovský kód nejakej konfigurácie D_1, potom $\bar{D}_2$ je booleovský kód nejakej konfigurácie D_2, do ktorej sa M dostane z D_1 počas najviac l krokov. (C_1, C_2, D_1 a D_2 sú konfigurácie v rámci pamäte $S(n)$.) Pre $l = 2^{rS(n)}$ preto platí: Slovo w dĺžky n patrí do L *iff* M prejde z počiatočnej konfigurácie C_0^w (pre vstup w) do akceptačnej konfigurácie C_{acc} počas najviac $2^{rS(n)}$ krokov *iff* $\Phi^{2^{rS(n)}}(\tilde{C}_0^w, \tilde{C}_{acc})$ je pravdivá a úplne kvantifikovaná *iff* kód formuly $\Phi^{2^{rS(n)}}(\tilde{C}_0^w, \tilde{C}_{acc})$ patrí do $TQBF$.

To znamená, že L je polynomiálne transformovateľný na $TQBF$, keďže každé $w \in \Sigma^*$ dĺžky n sa dá transformovať na kód formuly $\Phi^{2^{rS(n)}}(\tilde{C}_0^w, \tilde{C}_{acc})$ deterministickým T-strojom v polynomiálnom čase, pozri vyššie. $TQBF$ je preto $PSPACE$-úplný, keďže $TQBF \in PSPACE$, pozri vyššie. □

Hra BF

- daná je booleovská formula $\phi(x_1, \dots, x_n)$, (bez kvantifikátorov $\forall$, $\exists$)
- hru hrajú striedavo dvaja hráči H_1 a H_2; začína H_1
- hráči postupne a striedavo určujú hodnoty $b_1, b_2, \dots, b_n$ premenným $x_1, x_2, \dots, x_n$ (v každom kroku hry jednu b_i; H_1 nepárnym premenným a H_2 párnym)

- hru vyhral hráč H_1 *iff* $\phi(b_1, \ldots, b_n) = 1$

Príklad 1: $\phi(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$
$H_1 : x_1 = 1\ (= b_1)$
$H_2 : x_2 = 0\ (= b_2)$
$H_1 : x_3 = 1\ (= b_3)$
teda $\phi(1, 0, 1) = 1$, čiže H_1 vyhral.

Definícia. *Hráč H_1 má* ***víťaznú stratégiu*** *pre $\phi(x_1, \ldots, x_n)$, ak H_1 môže vyhrať pre každú postupnosť krokov hráča H_2, (t.j., ak formula $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1, \ldots, x_n)$ je pravdivá).*

Príklad 2: Hráč H_1 má nasledujúcu víťaznú stratégiu pre ϕ z Príkladu 1:
$H_1 : x_1 = 1$
$H_2 : x_2 = b_2$
$H_1 : x_3 = \bar{b}_2$

Nech $BF = \{< \phi(x_1, \ldots, x_n) > | \phi(x_1, \ldots, x_n)$ je booleovská formula, pre ktorú má H_1 víťaznú stratégiu$\}$.
($< \phi(x_1, \ldots, x_n) >$ je kód formuly $\phi(x_1, \ldots, x_n)$.)

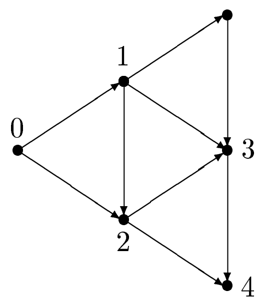
Veta. *Jazyk BF je PSPACE-úplný.*

Dôkaz. Stačí dokázať, že jazyk $BF \in PSPACE$, (to sa dokáže podobne, ako pre $TQBF$) a že $PSPACE$-úplný jazyk $TQBF$ je polynomiálne transformovateľný na jazyk BF. Nech Φ je úplne kvantifikovaná booleovská formula. Ak Φ je tvaru $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1, \ldots, x_n)$, potom $< \Phi >$ možno deterministicky a v polynomiálnom čase pretransformovať na $< \phi >$ a zrejme platí: $< \Phi > \in TQBF$ *iff* H_1 má víťaznú stratégiu pre ϕ *iff* $< \phi > \in BF$.
Ak Φ je iného tvaru, napríklad $\Phi = \forall x_1 \exists x_2 \exists x_3 \forall x_4 \phi(x_1, \ldots, x_4)$, potom $< \Phi >$ možno deterministicky a v polynomiálnom čase pretransformovať na $< \phi'(y_1, x_1, x_2, y_2, x_3, x_4) > = < \phi(x_1, \ldots, x_4) \wedge \underbrace{(y_1 \vee \bar{y}_1)}_{1} \wedge \underbrace{(y_2 \vee \bar{y}_2)}_{1} >$ a zrejme platí: $< \Phi > \in TQBF$ *iff* Φ je pravdivá *iff* formula $\exists y_1 \forall x_1 \exists x_2 \forall y_2 \exists x_3 \forall x_4$ $\phi'(y_1, x_1, x_2, y_2, x_3, x_4)$ je pravdivá *iff* H_1 má víťaznú stratégiu pre ϕ' *iff* $< \phi' > \in BF$. $\square$

Hra GG
- daný je orientovaný graf G a jeho vrchol b

- hru hrajú striedavo dvaja hráči H_1 a H_2; začína H_1 vo vrchole b
- hráči postupne a striedavo navštevujú (z momentálne navštíveného vrchulu) ešte nenavštívené susedné vrcholy, (v každom kroku jeden vrchol)
- hru prehral hráč, ktorý už nemôže navštíviť žiadny ešte nenavštívený vrchol

Príklad 3:



Hru začal hráč H_1 vo vrchole 0.
Čísla pri vrcholoch udávajú poradie,
v akom boli navštevované vrcholy grafu.
Hráč H_1 hru prehral.

Definícia. *Hráč H_1 má víťaznú stratégiu pre $< G, b >$, ak H_1 môže vyhrať pre každú postupnosť krokov hráča H_2.*

Nech $GG = \{< G, b > | G$ je orientovaný graf, b je jeho vrchol a
H_1 má víťaznú stratégiu pre $< G, b >\}$.

Veta. *Jazyk GG je PSPACE-úplný.*

5 Rekurzívne funkcie, registrovvé stroje a Turingove stroje

V tejto kapitole bude našim hlavným cieľom dokázať, že čiastočne rekurzívne funkcie, registrové stroje a Turingove stroje sú z hľadiska vypočítateľnosti ekvivalentné.

5.1 Rekurzívne funkcie

V tejto podkapitole budeme uvažovať len funkcie typu $f : N^n \to N$, $(N = \{0, 1, 2, \ldots\})$, pričom funkcia f nemusí byť totálna.

Označenia.

(a) *Funkcia I^n_m, $(1 \leq m \leq n)$, je n-árna funkcia, pre ktorú platí:* $I^n_m(x_1, \ldots, x_n) = x_m$ *pre všetky* $x_1, \ldots, x_n \in N$.

(b) *Funkcia s je unárna funkcia, pre ktorú platí: $s(x) = x + 1$, $\forall x \in N$. (Funkcia s je nasledovník.)*

(c) *Symbol* 0 *označuje (okrem iného aj) nula-árnu funkciu s konštantnou hodnotou* 0.

Definícia. *Nech f je n-árna funkcia a $g, f_1, \ldots, f_n$ sú m-árne funkcie. Funkcia g vznikne* ***skladaním*** *z funkcií $f, f_1, \ldots, f_n$, ak pre všetky $x_1, \ldots, x_m \in N$ platí:*

$$g(x_1, \ldots, x_m) = f(f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m)).$$

Poznámka. Všimnime si, že zápis funkcie g napríklad v tvare $g(x_1, x_2, x_3) = f(x_1, f_2(x_1, x_3), f_3(x_1, x_2, x_3))$ nezodpovedá celkom požiadavkám z definície operácie skladania funkcií, keďže podľa tejto definície má byť počet argumentov výslednej funkcie g a tiež všetkých funkcií $f_1, \ldots, f_n$ rovnaký, (v našom prípade 3). Toto sa však dá napraviť pomocou funkcií I^n_m napríklad takto: $g(x_1, x_2, x_3) = f(I^3_1(x_1, x_2, x_3), f_2(I^3_1(x_1, x_2, x_3), I^3_3(x_1, x_2, x_3)), f_3(x_1, x_2, x_3))$. Preto ani v ďalšom nebudeme veľmi dbať na rovnakú aritu funkcií pri operácii skladania.

Definícia. *Nech h je n-árna funkcia, g je $(n+2)$-árna funkcia a f je $(n+1)$-árna funkcia. Funkcia f vznikna z funkcií g a h operáciou* ***primitívnej rekurzie****, ak pre všetky $x_1, \ldots, x_n, y \in N$ platí:*

$$f(0, x_1, \ldots, x_n) = h(x_1, \ldots, x_n),$$
$$f(y+1, x_1, \ldots, x_n) = g(y, f(y, x_1, \ldots, x_n), x_1, \ldots, x_n).$$

Z definície primitívnej rekurzie vyplýva, že ak vieme vypočítať funkcie h a g, potom dokážeme (rekurzívne) vypočítať aj funkciu f.

Príklad 1: Funkcia $f(y, x) = y + x$ vznikne operáciou primitívnej rekurzie z funkcií $g(y, z, x) = z + 1$ a $h(x) = x$, lebo platí:

$$f(0, x) = x = h(x),$$
$$f(y+1, x) = y + 1 + x = f(y, x) + 1 = g(y, f(y, x), x).$$

Definícia. *Funkcia f je* ***primitívne rekurzívna****, ak vznikne z funkcí 0, s a I^n_m konečným počtom operácií skladania funkcií a primitívnej rekurzie. (Primitívne rekurzívne funkcie sú preto totálne.)*

Príklad 2: Funkcie h a g z Príkladu 1 sú primitívne rekurívne, lebo $h(x) = x = I^1_1(x)$ a $g(y, z, x) = z+1 = s(I^3_2(y, z, x))$. Preto je aj funkcia f z Príkladu 1 primitívne rekurzívna.

Veta 1. *Konštantná funkcia $K_0(x) = 0$, $\forall x \in N$, je primitívne rekurzívna.*

Dôkaz. Je ľahko vidieť, že funkcia $K_0(x)$ vznikne operáciou primitívnej rekurzie z primitívne rekurzívnych funkcií g a h, kde h je 0-árna funcia 0, (pozri bod (c) vyššie) a funkcia $g(y, z) = I^2_2(y, z) = z$, lebo platí:

$$K_0(0) = 0 = h,$$
$$K_0(y+1) = 0 = K_0(y) = I^2_2(y, K_0(y)) = g(y, K_0(y)). \ \square$$

Dôsledok 1. *Nech $j \geq 1$. Potom konštantná funkcia $K_j(x) = j$, $\forall x \in N$, je primitívne rekurzívna.*

Dôkaz. Keďže obe funkcie $K_0(x)$ a s (nasledovník) sú primitívne rekurzívne, potom aj funkcia $K_j(x) = j = \overbrace{s(s(\ldots s}^{j-\text{krát}}(K_0(x))\ldots))$, ktorá vynikne opakovaným sladaním z funkcií s a $K_0(x)$, musí byť primitívne rekurzívna. $\square$

Veta 2. *Funkcie F_1 až F_7 sú primitívne rekurzívne.*

$$F_1(x, y) = x + y$$

$$F_2(x, y) = x \cdot y$$

$$F_3(x, y) = y^x$$

$$F_4(x) = sg(x) = \begin{cases} 0, & ak\ x = 0 \\ 1, & inak \end{cases}$$

$$F_5(x) = \overline{sg}(x) = 1 - sg(x)$$

$$F_6(x, y) = y \dot{-} x = \begin{cases} 0, & ak\ y < x \\ y - x, & ak\ y \geq x \end{cases}$$

$$F_7(x, y) = |x - y| = (x \dot{-} y) + (y \dot{-} x)$$

Dôkaz.

Pre F_1: Pozri Príklad 2 vyššie.

Pre F_2: Keďže funkcia F_1 je primitívne rekurzívna, (pozri vyššie), je aj funkcia $g(x, z, y) = F_1(I_2^3(x, z, y), I_3^3(x, z, y)) = z+y$ primitívne rekurzívna. Preto je aj F_2 primitívne rekurzívna, keďže vznikne operáciou primitívnej rekurzie z primitívne rekurzívnych funkcií K_0 (z Vety 1) a $g(x, z, y)$, lebo platí:
$F_2(0, y) = 0 = K_0(y)$,
$F_2(x + 1, y) = (x + 1)y = x \cdot y + y = F_2(x, y) + y = g(x, F_2(x, y), y)$.

Pre F_3: Dôkaz je podobný, ako pre F_2.

Pre F_4: Funkcia $F_4(x) = sg(x)$ je primitívne rekurzívna, keďže vznikne operáciou primitívnej rekurzie z primitívne rekurzívnych funkcií 0 a $g(x, y)$, kde 0 je nula-árna funkcia z bodu (c) vyššie, $g(x, y) = K_1(I_1^2(x, y)) = 1$, ($K_1$ je z Dôsledku Vety 1), lebo platí:
$sg(0) = 0$, (nula vpravo je nula-árna funkcia 0),
$sg(x + 1) = 1 = g(x, sg(x))$.

Pre F_5: Dôkaz je podobný dôkazu pre F_4, keďže platí:
$\overline{sg}(0) = 1 = s(0)$, (vo výraze $s(0)$ sú funkcie s a 0 z bodov (b) a (c) vyššie),
$\overline{sg}(x + 1) = 0 = K_0(I_1^2(x, \overline{sg}(x)))$, ($K_0$ je z Vety 1)

Pre F_6: (Idea.) Podobne, ako pre F_4 alebo F_5 možno dokázať, ze funkcie $F_6'(x) = x \dot{-} 1$ a F_6 sú primitívne rekurzívne, lebo platí:
$F_6'(0) = 0$, (nula vpravo je nula-árna funkcia z bodu (c) vyššie),
$F_6'(x + 1) = x = I_1^2(x, F_6'(x))$ a
$F_6(0, y) = y = I_1^1(y)$,
$F_6(x + 1, y) = (y \dot{-} x) \dot{-} 1 = F_6'(I_2^3(x, F_6(x, y), y))$.

Pre F_7: Funkcia F_7 je primitívne rekurzívna, lebo vznikne (opakovanou)

operáciou sladania z primitívne rekurívnych funkcií F_1, F_6, I_1^2 a I_2^2, keďže platí:
$F_7(x,y) = F_1(F_6(I_2^2(x,y), I_1^2(x,y)), F_6(x,y))$. $\square$

Dôsledok 2. *Funkcie* $\bar{F}_3(x,y) = x^y$ *a* $\bar{F}_6(x,y) = x \dot{-} y$ *sú primitívne rekurzívne.*

Dôkaz: Je podobný Dôkazu pre F_7 (z Vety 2), lebo platí:
$\bar{F}_3(x,y) = F_3(I_2^2(x,y), I_1^2(x,y)) = F_3(y,x) = x^y$ a
$\bar{F}_6(x,y) = F_6(I_2^2(x,y), I_1^2(x,y)) = F_6(y,x) = x \dot{-} y$. $\square$

Veta 3. *Nech* $h(y, x_1, \ldots, x_n)$ *je primitívne rekurzívna funkcia. Potom aj funkcie*

$$f_1(y, z, x_1, \ldots, x_n) = \sum_{i=z}^{y+z} h(i, x_1, \ldots, x_n),$$

$$f_2(y, z, x_1, \ldots, x_n) = \prod_{i=z}^{y+z} h(i, x_1, \ldots, x_n),$$

sú primitívne rekurzívne.

Dôkaz: Zrejme platí:

$$\begin{aligned}
f_1(0, z, x_1, \ldots, x_n) &= h(z, x_1, \ldots, x_n) \\
f_1(y+1, z, x_1, \ldots, x_n) &= \sum_{i=z}^{y+z} h(i, x_1, \ldots, x_n) + h(y+z+1, x_1, \ldots, x_n) \\
&= f_1(y, z, x_1, \ldots, x_n) + h(y+z+1, x_1, \ldots, x_n) \\
&= g(y, f_1(y, z, x_1, \ldots, x_n), z, x_1, \ldots, x_n),
\end{aligned}$$

pre funkciu
$g(y, u, z, x_1, \ldots, x_n) = u + h(y+z+1, x_1, \ldots, x_n) = F_1(u, h(s(F_1(y,z)), x_1, \ldots, x_n))$,
kde F_1 je z Vety 1 a s je nasledovntík, (pozri bod (b) vyššie). Keďže g je primitívne rekurzívna, (vznikne operáciou skladania z primitívne rekurzívnych funkcií F_1, s a h), potom aj f_1 je primitívne rekurzívna funkcia, lebo vznikne z primitívne rekurzívnych funkcií h a g operáciou primitívnej rekurzie.

Dôkaz pre f_2 je podobný. $\square$

Definícia. *Čiastočná funkcia* $f(x_1, \ldots, x_n)$ *vznikne z čiastočnej funkcie* $g(y, x_1, \ldots, x_n)$ ***minimalizáciou****, zapisujeme*

$$f(x_1, \ldots, x_n) = \mu_y(g(y, x_1, \ldots, x_n) = 0),$$

ak pre všetky $x_1, \dots, x_n$ platí: $f(x_1, \dots, x_n) = y$ iff keď pre všetky $z < y$ je $g(z, x_1, \dots, x_n)$ definovaná a kladná a $g(y, x_1, \dots, x_n) = 0$. ($y$ je prvý nulový bod funkcie g vzhľadom na $x_1, \dots, x_n$.)

Ak naviac f a g sú totálne, hovoríme, že f vznikne z g ***regulárnou minimalizáciou****.*

Poznámka. Operácie minimalizácie nezachováva totálnosť a ani primitívnu rekurzívnosť.

Príklad : Nech $f(0) = 0$ a nech $f(x)$ nie je definivaná pre žiadne $x > 0$. Pre f platí: $f(x) = \mu_y(I_2^2(y, x) = 0)$.

Definícia. *Funkcia f je* ***rekurzívna****, ak f vznikne z funkcií 0, s a I_m^n konečným počtom operácií skladania funkcií, pritívnej rekurzie a regulárnej minimalizácie. (Rekurzívne funkcie sú preto totálne.)*

Poznámka. Každá primitívne rekurzívna funkcie je rekurzívna.

Definícia. *Čiastočná funkcia f je* ***čiastočne rekurzívna****, ak f vznikne z funkcií 0, s a I_m^n konečným počtom operácií skladania funkcií, primitívnej rekurzie a minimalizácie.*

Veta 4. *Nech funkcie $g(y, x_1, \dots, x_n)$ a $h(x_1, \dots x_n)$ sú primitívne rekurzívne a nech pre každé $x_1, \dots, x_n$ existuje $u \leq h(x_1, \dots, x_n)$ také, že $g(u, x_1, \dots, x_n) = 0$. Potom je funkcia $f(x_1, \dots, x_n) = \mu_y(g(y, x_1, \dots, x_n) = 0)$ primitívne rekurzívna.*

Dôkaz. Dokážme najprv, že pre každé $x_1, \dots, x_n$ platí:

$$f(x_1, \dots, x_n) = \sum_{y=0}^{h(x_1, \dots, x_n)} sg(\prod_{i=0}^{y} g(i, x_1 \dots, x_n)), \tag{1}$$

kde sg je z Vety 2.

Vyberme ľubovolné $x_1, \dots, x_n$. Podľa predpokladov Vety 4, pre tieto $x_1, \dots, x_n$ existuje $u \leq h(x_1, \dots, x_n)$ také, že $g(u, x_1, \dots, x_n) = 0$. Bez ujmy na všeobecnosti predpokladajme, že u je najmenšie s takouto vlastnosťou. Teda: $g(u, x_1, \dots, x_n) = 0$ a všetky členy postupnosti

$$g(0, x_1, \dots, x_n), g(1, x_1, \dots, x_n), \dots, g(u-1, x_1, \dots, x_n)$$

sú kladné. Preto, podľa definície minimalizácie dostaneme,

$$f(x_1, \dots, x_n) = u \tag{2}$$

a naviac, pre súčin $\prod_{i=0}^{y} g(i, x_1, \ldots, x_n)$ platí: Ak $y \leq u - 1$, potom je tento súčin kladný, lebo všetky jeho činitele $g(i, x_1, \ldots, x_n)$ sú kladné, a ak $y \geq u$, potom má tento súčin hodnotu 0, lebo aspoň jeden jeho činiteľ (a to $g(u, x_1, \ldots, x_n)$) má hodnotu 0. V dôsledku toho:

$$sg(\prod_{i=0}^{y} g(i, x_1, \ldots, x_n)) = \begin{cases} 1, & \text{ak } y \leq u - 1, \\ 0, & \text{ak } y \geq u. \end{cases}$$

Preto z (2) dostaneme:

$$\begin{aligned} \sum_{y=0}^{h(x_1,\ldots,x_n)} sg(\prod_{i=0}^{y} g(i, x_1, \ldots, x_n)) &= \sum_{y=0}^{u-1} sg(\prod_{i=0}^{y} g(i, x_1, \ldots, x_n)) + \sum_{y=u}^{h(x_1,\ldots,x_n)} sg(\prod_{i=0}^{y} g(i, x_1, \ldots, x_n)) \\ &= u + 0 = u = f(x_1, \ldots, x_n). \end{aligned}$$

Tým sme dokázali (1). Z (1) vyplýva, že f je primitívne rekurzívna, lebo ju možno zostrojiť z primitívne rekurzívnych funkcií g, h a sg pomocou operácie skladania a aplikovaním Vety 3 (pre súčin aj súčet). $\square$

Veta 5. *Funkcie F_8 až F_{15} sú primitívne rekurzívne.*

$$F_8(x, y) = \lfloor x/y \rfloor, \qquad (\lfloor x/y \rfloor = 0, \text{ ak } y = 0)$$

$$F_9(x, y) = x \bmod y, \qquad (x \bmod y = x, \text{ ak } y = 0)$$

$$F_{10}(x, y) = \mathrm{div}(x, y) = \begin{cases} 1, & \text{ak } y \text{ delí } x \\ 0, & \text{inak} \end{cases}$$

$$F_{11}(x) = \mathrm{nd}(x) = \begin{cases} \text{počet deliteľov čísla } x, & \text{ak } x \neq 0 \\ 0, & \text{inak} \end{cases}$$

$$F_{12}(x) = \mathrm{chp}(x) = \begin{cases} 0, & \text{ak } x \text{ je prvočíslo} \\ 1, & \text{inak} \end{cases}$$

$$F_{13}(x) = \pi(x) = \text{ počet prvočísel nepresahujúcich } x$$

$$F_{14}(x) = \mathrm{p}(x) = x\text{-té prvočíslo } \ (\text{t.j., } \mathrm{p}(0) = 2, \mathrm{p}(1) = 3, \mathrm{p}(2) = 5, \ldots)$$

$$F_{15}(x, y) = \mathrm{ex}(x, y) = \text{exponent prvočísla } \mathrm{p}(x) \text{ v rozklade čísla } y \text{ na prvočísla, } (\mathrm{ex}(x, y) = 0 \text{ pre } y = 0)$$

Dôkaz. Keďže platí $\lfloor x/y \rfloor = sg(y) \sum_{i=1}^{(x \dot{-} 1)+1} \overline{sg}(i \cdot y \dot{-} x)$, potom je funkcia F_8 primitívne rekurzívna, lebo ju možno zostrojiť z primitívne rekurzívnych funkcií F_2, F_4, F_5 a F_6 (z Vety 2) a K_1 (z Dôsledku Vety 1), pomocou operácií skladania a aplikovaním Vety 3 nasledovne:

Funkcia $h(i,x,y) = F_5(F_6(y, F_2(i,x))) = \overline{sg}(i \cdot x \dot{-} y)$ je primitívne rekurzívna, lebo vznikne skladaním z F_2, F_6 a F_5. Podľa Vety 3 je potom aj funkcia $f(u,v,x,y) = \sum_{i=v}^{u+v} h(i,x,y)$ primitívne rekurzívna. Teda aj funkcia

$$\begin{aligned} F_8(x,y) &= F_2(F_4(y), f(F_6(K_1(y),x), K_1(y), x, y)) \\ &= sg(y) \sum_{i=1}^{(x \dot{-} 1)+1} \overline{sg}(i \cdot y \dot{-} x) = \lfloor x/y \rfloor, \end{aligned}$$

ktorá vznikne skladaním z F_2, F_4, f, F_6 a K_1, je primitívne rekurzívna.

Pre funkcie F_9 až F_{13} sú dôkazy podobné, (ale Veta 3 sa použije len pre F_{11} a F_{13}), pričom sa využijú tieto rovnosti:

$x \bmod y = x \dot{-} y \cdot \lfloor x/y \rfloor$, $\operatorname{div}(x,y) = \overline{sg}(x \bmod y)$,
$\operatorname{nd}(x) = sg(x) \sum_{i=1}^{(x \dot{-} 1)+1} \operatorname{div}(x,i)$, $\operatorname{chp}(x) = sg(|\operatorname{nd}(x) - 2|)$ a
$\pi(x) = \sum_{i=0}^{x} \overline{sg}(\operatorname{chp}(i))$.

V dôkaze pre F_{14} je použitá Veta 4 (namiesto Vety 3) a je využitá rovnosť $\operatorname{p}(x) = \mu_y(|\pi(y) - (x+1)| = 0)$ a tiež fakt, že $\operatorname{p}(x) \leq 2^{2^x}$; ($h(x) = 2^{2^x}$ je funkcia h z Vety 4). Podobne, v dôkaze pre F_{15} je použitá Veta 4, ďalej rovnosť $\operatorname{ex}(x,y) = \mu_u(sg(y) \cdot \operatorname{div}(y, (\operatorname{p}(x))^{u+1}) = 0)$ a fakt, že $\operatorname{ex}(x,y) \leq y$; ($h(x,y) = I_2^2(x,y) = y$ je funkcia h z Vety 4). □

5.2 Registrové stroje

V tejto podkapitole dokážeme, že registrové stroje a Turingove stroje sú z hľadiska vypočítateľnosti ekvivalentné. Registrový stroj je podobný Turingovmu stroju, ale namiesto pások má registre.

Registrový stroj má:

- konečne veľa stavov $q_0, q_1, \ldots, q_k$, (q_1 je počiatočný a q_0 je koncový stav)
- konečne veľa registrov $R_0, R_1, \ldots, R_t$, (obsah každého R_i je nejaké číslo z $N = \{0, 1, 2, \ldots\}$)
- konečne veľa inštrukcií typu:

  (q_i, R_j, q_l, q_m), (t.j. v stave q_i vykonaj: <u>if</u> $R_j = 0$ <u>then</u> <u>goto</u> q_l <u>else</u> <u>goto</u> q_m)

  $(q_i, R_j, +1, q_m)$, (t.j. v stave q_i vykonaj: $R_j \leftarrow R_j + 1$ a <u>goto</u> q_m)

  $(q_i, R_j, \dot{-}\, 1, q_m)$, (t.j. v stave q_i vykonaj: $R_j \leftarrow R_j \dot{-}\, 1$ a <u>goto</u> q_m)

Definícia. *Registrový stroj M počíta funkciu $f(x_1, \ldots, x_n)$, ak na začiatku výpočtu je M v stave q_1, pre všetky $i = 1, \ldots, n$ platí $R_i = x_i$ a ostatné registre obsahujú 0 a na konci výpočtu je M vstave q_0 a platí $R_0 = f(x_1, \ldots, x_n)$. Ak hodnota $f(x_1, \ldots, x_n)$ nie je definovaná, potom sa výpočet neskončí.*

Definícia. *Turingov stroj (so vstupnou páskou a s pracovnými páskami) počíta funkciu $f(x_1, \ldots, x_n)$, ak má na začiatku výpočtu na vstupnej páske slovo $01^{x_1}0 \ldots 01^{x_n}0$ a na konci výpočtu má na prvej pracovnej páske slovo $1^{f(x_1,\ldots,x_n)}$. Ak hodnota $f(x_1, \ldots, x_n)$ nie je definovaná, potom sa výpočet na vstupe $01^{x_1}0 \ldots 01^{x_n}0$ neskončí.*

Pre registrové a Turingove stroje počítajúce funkcie platia nasledujúce Lemy A a B.

Lema A. *Každý registrový stroj počítajúci nejakú funkciu $f(x_1, \ldots, x_n)$ možno simulovať nejakým Turingovým strojom.*

Dôkaz. Nech M je registrový stroj, ktorý počíta nejakú funkciu $f(x_1, \ldots, x_n)$ a nech M má t registrov. Turingov stroj T (s t pracovnými páskami) bude simulovať stroj M nasledovne. Na začiatku výpočtu si T skopíruje zo vstupu podslovo 1^{x_i} na $(i+1)$-vú pracovnú pásku pre každé i. Potom inštrukciu $(q_i, R_j, +1, q_m)$ [resp. inštrukciu $(q_i, R_j, \dot{-}\, 1, q_m)$] simuluje tak, že hlavu na $(j+1)$-vej pracovnej páske presunie o jedno políčko vpravo [resp. vľavo, ak je to možné]. Pri simulácii inštrukcie (q_i, R_j, q_l, q_m) musí T zistiť, či $R_j = 0$; T to zistí tak, že hlava na $(j+1)$-vej pracovnej páske je úplne na jej ľavom kraji. □

Lema B. *Každý Turingov stroj počítajúci nejakú funkciu $f(x_1, \dots, x_n)$ možno simulovať nejakým registrovým strojom.*

Dôkaz. Nech T_1 je Turingov stroj (s pracovnými páskami) počítajúci funkciu $f(x_1, \dots, x_n)$. Nech T_2 je Turingov stroj (bez pracovných pások) simulujúci T_1; T_2 má jednu vstupnú, obojstranne nekonečnú prepisovaciu pásku, pričom na začiatku výpočtu má na vstupnej páske slovo $01^{x_1}0 \dots 01^{x_n}0$ a na konci výpočtu slovo $01^{f(x_1,\dots,x_n)}0^l$, pre nejaké $l \geq 1$. (Nulami vpravo za slovom $01^{f(x_1,\dots,x_n)}0$ prepíše T-stroj pred koncom výpočtu úsek pásky, ktorý použil pre výpočet funkcie $f(x_1, \dots, x_n)$, keďže v tomto dôkaze predpokladáme, že T-stroj nemôže prepísať symboly rôzne od B na symbol B, kde B je blank a tiež, že každý čítaný symbol B musí byť následne prepísaný na najaký symbol rôzny od B).

Konečne nech T_3 je Turingov stroj bez pracovných pások, s jednou vstupnou, obojstranne nekonečnou prepisovacou páskou a s páskovou abecedou $\Sigma = \{0, 1, B\}$ simulujúci T_2. T-stroj T_3 kóduje s symbolov T-stroja T_2 pomocou s slov z $\{0,1\}^m$, kde $s \leq 2^m$ pre najaké m. T_3 má na začiatku výpočtu na vstupnej páske slovo $\bar{0}\bar{1}^{x_1}\bar{0} \dots \bar{0}\bar{1}^{x_n}\bar{0}$ a na konci výpočtu slovo $\bar{0}\bar{1}^{f(x_1,\dots,x_n)}\bar{0}^{l'}$, pre nejaké $l' \geq 1$, kde $\bar{0} = 0^m$ a $\bar{1} = 0^{m-1}1$, (t.j. slovo $\bar{0} = 0^m$ resp. $\bar{1} = 0^{m-1}1$ kóduje symbol 0 resp. 1 páskovej abecedy T-stroja T_2). (V tomto dôkaze použijeme T-stroj T_3 namiesto T_1 alebo T_2, lebo T_3 s vyššie uvedenými vlastnosťami je z technických dôvodov pre simuláciu registrovým strojom oveľa vhodnejší, než T_1 alebo T_2.

Budeme hovoriť, že T-stroj T_3 je v konfigurácii $C = uqav$, kde $u, v \in \{0,1\}^*$, $a \in \{0, 1, B\}$ a q je stav T-stroja T_3, ak je na jeho vstupnej páske slovo uav, každý symbol vľavo aj vpravo od slova uav je symbol B, hlava číta symbol a a T_3 je v stave q.

Pre každé $d \in \{0,1\}^+$, nech $num(d)$ je číslo, ktorého binárny zápis je slovo d. Napríklad, $num(101) = 5$.

Predpokladajme, že T_3 prejde v 1 kroku výpočtu z konfigurácie $C = uqav$ do konfigurácie $C' = u'q'a'v'$, kde napríklad:

$C = \overbrace{010}^{u}q0\overbrace{1100}^{v}$, $C' = \overbrace{0101}^{u'}q'1\overbrace{100}^{v'}$, t.j. $a = 0$ a $a' = 1$.

Je zrejmé, že obsah vstupnej pásky T-stroja T_3 vľavo od jeho hlavy je v konfigurácii C jednoznačne kódovaný číslom $num(1u)$, (v našom príklade $num(1010)$). Vedúca 1 vľavo od u je pridaná preto, aby bola zaručená jednoznačnosť kódovania; bez nej by rôzne slová, napr. 010 a 00010, mali rovnaký kód $num(010) = num(00010) = 2$.

Podobne obsah vstupnej pásky T-stroja T_3 vpravo od jeho hlavy je v konfigurácii C jednoznačne kódovaný číslom $num(1v^R)$, (v našom príklade $num(10011)$). Aj v tomto prípade je z rovnakých dôvodov pridaná vedúca 1 a naviac, namiesto slova v je použitý jeho zrkadlový obraz v^R a to preto, aby boli najmenej významné bity binárneho zápisu čísla $num(1v^R)$ najbližšie k hlave T-stroja T_3. (V prípade čísla $num(1u)$ to platí priamo pre slovo $1u$.)

To znamená, že štvorica $num(1u), q, a, num(1v^R)$ jednoznačne kóduje konfigiuráciu C a štvorica $num(1u'), q', a', num(1v'^R)$ jednoznačne kóduje konfigiuráciu C'.

Preto môže registrový stroj M simulovať 1 krok výpočtu T-stroja T_3 nasledovne: Z čísel $num(1u)$, $num(1v^R)$, (uložených v registroch) a z hodnôt q, a, vypočíta M čísla $num(1u')$, $num(1v'^R)$, (uloží ich do registrov) a zistí hodnoty q' a a'. Naviac, M má pre každú dvojicu (q, a) podprogram, ktorým simuluje inštrukciu $\delta(q, a) = (q', b, \text{posun hlavy})$ T-stroja T_3, pričom hodnoty q', b a "posun hlavy" sú už vopred zakomponované do simulujúceho podprogramu.

Ukážme si teraz, ako možno z hodnôt $num(1u), q, a, num(1v^R)$ zistiť hodnoty $num(1u'), q', a', num(1v'^R)$ pre prípad, keď sa hlava T-stroja T_3 posunie vpravo. (Keď sa hlava posunie vľavo alebo zostane na mieste, postupuje sa podobne.)

Výpočet $num(1u')$: Je zrejmé, že keď sa hlava posunie vpravo, potom platí: $u' = ub$, kde b je symbol, na ktorý je prepísaný symbol a čítaný v konfigurácii C. (V našom príklade vyššie: $u = 010$, $a = 0$, $b = 1$, $u' = 0101$.) Preto tiež platí $1u' = 1ub$, čiže:

$$num(1u') = 2 \cdot num(1u) + b.$$

Pripomeňme, že M pozná hodnoty b a "posun hlavy", lebo tieto sú už vopred zakomponované do podprogramu simulujúceho inštrukciu $\delta(q, a)$, pozri vyššie.

Výpočet $num(1v'^R)$: Ak $v \in \{0, 1\}^+$ a hlava sa posunie vpravo, potom slovo v' dostaneme zo slova v odstránením jeho najľavejšieho symbolu, čo je najpravejší symbol slova v^R a preto tiež slovo $1v'^R$ dostaneme zo slova $1v^R$ odstránením jeho najpravejšieho symbolu. (V našom príklade vyššie: $v = 1100$ a $v' = 100$.) A ak $v = \epsilon$, (t.j., všetky symboly vpravo od hlavy sú B) a hlava sa posunie vpravo, potom $v' = \epsilon$, (lebo opäť všetky symboly vravo od hlavy sú B). Z vyššie uvedeného dostaneme:

$$num(1v'^R) = \begin{cases} \lfloor num(1v^R)/2 \rfloor, & \text{ak } v \in \{0,1\}^+, \text{ (t.j., ak } num(1v^R) \geq 2) \\ 1, & \text{ak } v = \epsilon, \text{ (t.j., ak } num(1v^R) = 1) \end{cases}$$

Zistenie symbolu a': Ak $v \in \{0,1\}^+$ a hlava sa posunie vpravo, potom a' je najľavejší symbol slova v, čo je najpravejší symbol slova v^R a teda aj slova $1v^R$. Ak $v = \epsilon$, (t.j., všetky symboly vpravo od hlavy sú B) a hlava sa posunie vpravo, potom $a' = B$. Teda:

$$a' = \begin{cases} 0, & \text{ak } num(1v^R) \text{ je párne číslo a } v \in \{0,1\}^+ \\ 1, & \text{ak } num(1v^R) \text{ je nepárne číslo a } v \in \{0,1\}^+ \\ B, & \text{ak } v = \epsilon \end{cases}$$

Zistenie stavu q' a zistenie ďalšej simulovanej inštrukcie: Podprogram pre simulovanie inštrukcie $\delta(q,a) = (q', b, \text{posun hlavy})$, (do ktorého sú už vopred zakomponované hodnoty $q', b, \text{posun hlavy}$), je naprogramovaný tak, že po skončení simulácie prejde na podprogram pre simulovanie inštrukcie $\delta(q', a')$.

Ukážme si teraz, ako M násobí a delí dvomi pri výpočte $num(1u')$ a $num(1v'^R)$. M delí dvomi nasledovne: V cykle priebežne z registra R (obsahujúceho na začiatku cyklu $num(1v^R)$) odpočítava 2-krát jednotku a do registra R' pripočítava 1-krát jednotku; keď bude $R = 0$, potom bude $R' = \lfloor num(1v^R)/2 \rfloor = num(1v'^R)$. Podobne postupuje M pri násobení dvomi.

M zisťuje paritu čísla $num(1v^R)$ - pri zisťovaní symbolu a' - nasledovne: Z registra R obsahujúceho $num(1v^R)$ odpočítava v cykle jednotku; ak bude R obsahovať 0 v párnom [nepárnom] kroku cyklu, potom je $num(1v^R)$ párne [nepárne] číslo. Ak nechceme z pamäte stroja M vymazať číslo $num(1v^R)$, (aby sme ho mohli použiť v ďalších výpočtoch), môžeme - pri odpočítavaní jednotky z R - súčasne pripočítavať jednotku do pomocného registra R'; keď bude $R = 0$, potom bude $R' = num(1v^R)$. (Týmto spôsobom si M môže vytvárať záložné kópie rôznych čísel v rôznych výpočtoch.)

Z vyššie uvedeného je zrejmé, že M dokáže simulovať výpočet T-stroja T_3 z počiatočnej konfigurácie $C_0 = q_1\bar{0}\bar{1}^{x_1}\bar{0}\ldots\bar{0}\bar{1}^{x_n}\bar{0} = u_0q_10v_0$ do koncovej konfigurácie $C_t = q_0\bar{0}\bar{1}^{f(x_1,\ldots,x_n)}\bar{0}^{l'} = u_tq_00v_t$ za predpokladu, že na začiatku simulácie má M, v niektorých dvoch registroch, čísla $num(1v_0^R)$ a $num(1u_0)$ a že M pozná stav a čítaný symbol v konfigurácii C_0. M pozná posledné tri zo štyroch uvedených hodnôt, lebo $u_0 = \epsilon$, (keďže vľavo od hlavy sú len symboly B), t.j., $num(1u_0) = 1$, ďalej T_3 začína výpočet v stave q_1 a jeho hlava číta (v konfigurácii C_0) najľavejší symbol najľavejšieho podslova $\bar{0}$ na páske, čo je 0. Teda, M si musí pred začiatkom simulácie ešte vypočítať číslo $num(1v_0^R)$ z čísel $x_1, \ldots, x_n$, ktoré má na začiatku výpočtu v registroch $R_1, \ldots, R_n$.

Výpočet $num(1v_0^R)$: Zrejme $v_0 = 0^{m-1}\bar{1}^{x_1}\bar{0}\ldots\bar{0}\bar{1}^{x_n}\bar{0}$ a teda $num(1v_0^R) = 1\bar{0}(\bar{1}^R)^{x_n}\bar{0}\ldots\bar{0}(\bar{1}^R)^{x_1}0^{m-1}$, kde $\bar{1}^R = 10^{m-1}$ a $\bar{0} = 0^m$. M si najprv vypočíta a do registrov uloží čísla $r_1, \ldots, r_n$ a $z_1, \ldots, z_{n+1}$, kde:

$$r_i = num((\bar{1}^R)^{x_i}) = 2^{m-1} \cdot (\sum_{j=0}^{x_i-1} 2^{mj}), \text{ pre } i = 1, 2, \ldots, n,$$

$$z_1 = m - 1, \quad z_{i+1} = z_i + m(x_i + 1), \text{ pre } i = 1, 2, \ldots, n.$$

(z_i je počet bitov vpravo za podslovom $(\bar{1}^R)^{x_i}$ v slove $1v_0^R$ pre $i = 1, 2, \ldots, n$. z_{n+1} je počet bitov vpravo za vedúcou 1 v slove $1v_0^R$.) M vypočíta $num(1v_0^R)$ takto:

$$num(1v_0^R) = 2^{z_{n+1}} + \sum_{i=1}^{n} r_i 2^{z_i}.$$

Potom M simuluje výpočet T-stroja T_3 z konfigurácie C_0 do C_t a po ukončení simulácie musí M ešte vypočítať číslo $f(x_1, \ldots, x_n)$ pomocou čísla $num(1v_t^R)$, kde $v_t = 0^{m-1}\bar{1}^{f(x_1,\ldots,x_n)}\bar{0}^{l'}$. Keďže v slove v_t je práve $f(x_1, \ldots, x_n)$ symbolov 1, (v každom podslove $\bar{1} = 0^{m-1}1$ je jeden symbol 1), potom v slove $1v_t^R$ je práve $f(x_1, \ldots, x_n) + 1$ symbolov 1. Preto môže M vypočítať číslo $f(x_1, \ldots, x_n)$ pomocou nasledujúceho algoritmu:

$R \leftarrow num(1v_t^R)$; $\quad R_0 \leftarrow 0$;
L: ak R obsahuje nepárne číslo, potom $R_0 \leftarrow R_0 + 1$; (zistená ďalšia 1 v v_t^R)
$R \leftarrow \lfloor R/2 \rfloor$;
ak $R \geq 1$ goto L;
$R_0 \leftarrow R_0 - 1$; (odpočítaná vedúca 1 v slove $1v_t^R$)

Po vykonaní algoritmu platí: $R_0 = f(x_1, \ldots, x_n)$. $\Box$

5.3 Ekvivalencia čiastočne rekurzívnych funkcií a Turingových strojov

V tejto podkapitole dokážeme, že čiastočne rekurzívne funkcie a Turingove stroje sú z hľadiska vypočítateľnosti ekvivalentné.

Fakt 1.

(a) *Pre každú funkciu* 0, s *a* I^n_m *existuje Turingov stroj, ktorý ju počíta.*

(b) *Ak pre každú funkciu* $f, f_1, \ldots, f_m$ *existuje Turingov stroj, ktorý ju počíta a funkcia* g *vznikne z funkcií* $f, f_1, \ldots, f_m$ *operáciou skladania, potom aj pre funkciu* g *existuje Turingov stroj, ktorý ju počíta.*

(c) *Ak pre funkcie* g *a* h *existujú Turingove stroje, ktoré ich počítajú a funkcia* f *vznikne z* g *a* h *operáciou primitívnej rekurzie, potom aj pre funkciu* f *existuje Turingov stroj, ktorý ju počíta.*

(d) *Ak pre funkciu* g *existuje Turingov stroj, ktorý ju počíta a funkcia* f *vznikne z* g *operáciou minimalizácie, potom aj pre* f *existuje Turingov stroj, ktorý ju počíta.*

Dôkaz. Je zrejmý. □

Lema C. *Pre každú čiastočne rekurzívnu funkciu existuje Turingov stroj, ktorý ju počíta.*

Dôkaz. Vyplýva z Faktu 1. □

Lema D. *Ak pre funkciu* $f : N^n \to N$ *existuje Turingov stroj, ktorý ju počíta, potom* f *je čiastočne rekurzívna.*

Dôkaz. Bez ujmy na všeobecnosti a pre jednoduchosť budeme uvažovať deteministické Turingove stroje bez pracovných pások a s obojstranne nekonečnou vstupnou páskou. T-stroj má páskovú abecedu $\Sigma = \{a_0, a_1, \ldots, a_r\}$, pre nejaké $r \geq 1$, kde $a_0 = B$ (blank symbol) a množinu stavov $Q = \{q_0, q_1, \ldots, q_s\}$, pre nejaké $s \geq 1$, kde q_1 je počiatočný a q_0 je koncový stav. Takýto T-stroj má na začiatku výpočtu funkcie $f(x_1, \ldots, x_n)$ na vstupnej páske slovo $01^{x_1}0 \ldots 01^{x_n}0$ a na konci výpočtu slovo $01^{f(x_1 \ldots, x_n)}0$.

Našim cieľom je zostrojiť vhodné čiastočne rekurívne funkcie, pomocou ktorých možno simulovať výpočty T- strojov. K tomu potrebujeme prirodzenými číslami kódovať rôzne štruktúry súvisiace s výpočtami T-strojov.

Číslovanie inštrukcií: Nech $c(i, j)$ je poradové číslo usporiadanej dvojice (i, j) v postupnosti $(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (2, 0), (0, 3), (1, 2), (2, 1), (3, 0), \ldots$, kde $c(0, 0) = 0$. (Dá sa dokázať, že $c(i, j)$, $l(m)$ a $r(n)$ sú primitívne rekurzívne funkcie, kde $l(c(i,j)) = i$ a $r(c(i,j)) = j$ pre všetky i,j.) Nech p_i je i-te prvočíslo; ($p_0 = 2, p_1 = 3, \ldots$). Potom $p_{c(i,j)}^{3c(m,k)+x+1}$ je číslo inštrukcie $\delta(q_i, a_j) = (q_m, a_k, x)$, kde $x = 1$ [resp. $x = 2$ resp. $x = 0$], keď sa hlava posunie vľavo [resp. vpravo resp. zostane na mieste].

Číslovanie T-strojov: Číslo T-stroja je súčin čísel jeho inštrukcií.

Číslovanie konfigurácií:

Príklad: Konfigurácia $a_3a_2q_3a_1a_2$ má číslo $2^{2\cdot 3} \cdot 3^{2\cdot 2} \cdot 5^{2\cdot 3+1} \cdot 7^{2\cdot 1} \cdot 11^{2\cdot 2}$, pričom hlava číta symbol vpravo od nej (t.j. a_1), párne exponenty kódujú páskové symboly a nepárny exponent kóduje stav a pozíciu hlavy. Všimnime si, že pri kódovaní sú použité postupne prvočísla $2, 3, 5, 7, 11$.

Vo všeobecnosti platí: konfigurácia $a_{l_0}a_{l_1}\ldots a_{l_{i-1}}q_ja_{l_{i+1}}\ldots a_{l_n}$, kde $l_0 \neq 0$ a $l_n \neq 0$, má číslo : $2^{2l_0} \cdot 3^{2l_1} \cdots p_{i-1}^{2l_{i-1}} \cdot p_i^{2j+1} \cdot p_{i+1}^{2l_{i+1}} \cdots p_n^{2l_n}$, kde p_i je i-te prvočíslo.

Jeden krok výpočtu, keď sa hlava T-stroja presunie vľavo, kontroluje funkcia

$$prech_L(x,y,z) = \begin{cases} 0, & \text{ak platí } (a) \text{ - v tomto prípade prejde T-stroj s číslom } z \\ & \quad \text{v 1 kroku (s presunom hlavy vľavo) z konfigurácie} \\ & \quad \text{s číslom } x \text{ do konfigurácie s číslom } y \\ 1, & \text{inak.} \end{cases}$$

(a) ((x je číslo nejakej konfigurácie)$\wedge$(y je číslo nejakej konfigurácie)$\wedge$
(z je číslo nejakého T-stroja)$\wedge$
($\exists m_1, m_2, v, i, j, k, m, n \leq x + y$ také, že:
((($x = m_1 \cdot p_i^{2v} \cdot p_{i+1}^{2j+1} \cdot p_{i+2}^{2k} \cdot m_2$)$\vee$(hlava je celkom vľavo))$\wedge$
($y = m_1 \cdot p_i^{2m+1} \cdot p_{i+1}^{2v} \cdot p_{i+2}^{2n} \cdot m_2$)$\wedge$
(ani p_i ani p_{i+1} ani p_{i+2} nedelí číslo $m_1 \cdot m_2$)$\wedge$
(T-stroj s číslom z obsahuje inštrukciu $\delta(q_j, a_k) = (q_m, a_n, 1)$)))

Príklad: Nech C a C' sú nasledujúce konfigurácie:

$C = a_{l_0}\ldots a_{l_{i-1}}a_v\, q_ja_ka_{l_{i+3}}\ldots a_{l_t}$,

$C' = a_{l_0}\ldots a_{l_{i-1}}q_ma_va_na_{l_{i+3}}\ldots a_{l_t}$,

teda,

C má číslo $x = \underbrace{2^{l_0}\cdots p_{i-1}^{2l_{i-1}}}_{m_1} \cdot p_i^{2v} \cdot p_{i+1}^{2j+1} \cdot p_{i+2}^{2k} \cdot \underbrace{p_{i+3}^{2l_{i+3}}\cdots p_t^{2l_t}}_{m_2}$,

C' má číslo $y = \quad m_1 \cdot p_i^{2m+1} \cdot p_{i+1}^{2v} \cdot p_{i+2}^{2n} \cdot m_2$.

Ak T-stroj s číslom z obsahuje inštrukciu $\delta(q_j, a_k) = (q_m, a_n, 1)$, potom tento T-stroj prejde v 1 kroku z C do C', čiže $prech_L(x,y,z) = 0$.

Podobne môžeme zostrojiť funkcie $prech_N(x,y,z)$ resp. $prech_P(x,y,z)$ pre prípad, keď sa hlava nepohybuje resp. pre pohyb hlavy vpravo. Dá sa dokázať, že $prech_L(x,y,z)$, $prech_N(x,y,z)$, $prech_P(x,y,z)$ a $prech(x,y,z)$ sú primitívne rekurzívne funkcie, kde

$$prech(x,y,z) = \begin{cases} 0, & \text{ak } prech_L(x,y,z) = 0 \vee prech_N(x,y,z) = 0 \vee prech_P(x,y,z) = 0 \\ 1, & \text{inak.} \end{cases}$$

<u>Číslo výpočtu</u> $C_0, C_1, C_2, \ldots, C_t$, kde C_i je konfigurácia pre každé i, je číslo

$$2^{cislo(C_0)} \cdot 3^{cislo(C_1)} \cdots p_t^{cislo(C_t)},$$

kde p_i je i-te prvočíslo a $cislo(C_i)$ je číslo konfigurácie C_i pre každé i.

Nech $ex(i,x)$ je exponent prvočísla p_i v rozklade čísla x na prvočísla. (Funkcia $ex(i,x)$ je primitívne rekurzívna, pozri Vetu 5.) Teda, ak x je číslo výpočtu $C_0, C_1, \ldots, C_t$, potom $ex(i,x) = cislo(C_i)$ pre $1 \leq i \leq t$.

To, či postupnosť $C_0, C_1, \ldots, C_t$ je výpočet T-stroja, kontroluje funkcia

$$vyp(x,y,z) = \begin{cases} 0, & \text{ak platí } (b) \text{ - v tomto prípade } x \text{ je číslo výpočtu} \\ & \quad \text{T-stroja s číslom } y \text{ začínajúceho konfiguráciou s číslom } z \\ 1, & \text{inak.} \end{cases}$$

(b) $(\forall i \leq x)(ex(i+1,x) \neq 0 \Rightarrow (prech(\overbrace{ex(i,x)}^{cislo(C_i)}, \overbrace{ex(i+1,x)}^{cislo(C_{i+1})}, y) = 0)) \wedge (\underbrace{ex(0,x)}_{cislo(C_0)} = z)$

Dá sa dokázať, že funkcia vyp a ďalšie funkcie tvp a $tobs$, sú primitívne rekurzívne, kde

$$tvp(x,y,x_1,\ldots,x_n) = \begin{cases} 0, & \text{ak } x \text{ je číslo výpočtu T-stroja s číslom } y\text{, ktorý začína} \\ & \text{konfiguráciou } q_1 0 1^{x_1} 0 \ldots 0 1^{x_n} 0 \text{ a končí nejakou} \\ & \text{konfiguráciou } q_0 0 1^z 0\text{, t.j., } tobs(x) = z; \\ & \text{(funkcia } tobs \text{ je definovaná nižšie)} \\ 1, & \text{inak.} \end{cases}$$

Funkcia tvp je zostrojená pomocou vyp a ďalších primitívne rekurzívnych funkcií a funkcia $tobs$ je definovaná nasledovne:

$tobs(x) =$ počet jednotiek v poslednej konfigurácii výpočtu s číslom x.

Keďže *tobs* a *tvp* sú primitívne rekurzívne, potom nasledujúca funkcia Tun musí byť čiastočne rekurzívna.

$$Tun(y, x_1, \ldots, x_n) = tobs(\mu_x(tvp(x, y, x_1, \ldots, x_n) = 0)). \qquad (*)$$

Teraz už ľahko dokážeme Lemu D. Nech $f : N^n \to N$ je ľubovoľná funkcia, pre ktorú existuje T-stroj (nech má číslo j), ktorý ju počíta. Ak je f definovaná pre $x_1, \ldots, x_n$, potom existuje jediný a konečný výpočet (nech má číslo $\hat{x}$) T-stroja s číslom j z konfigurácie $q_1 0 1^{x_1} 0 \ldots 0 1^{x_n} 0$ do konfigurácie $q_0 0 1^{f(x_1,\ldots,x_n)} 0$. Preto $tvp(\hat{x}, j, x_1, \ldots, x_n) = 0$, $\hat{x} = \mu_x(tvp(x, j, x_1, \ldots, x_n) = 0)$, $tobs(\hat{x}) = f(x_1, \ldots, x_n)$ a teda, podľa (*) dostaneme:

$$Tun(j, x_1, \ldots, x_n) = tobs(\mu_x(tvp(x, j, x_1, \ldots, x_n) = 0)) = f(x_1, \ldots, x_n).$$

To ale znamená, že f je čiastočne rekurzívna, lebo Tun je čiastočne rekurzívna.

Z (*) tiež vyplýva, že $Tun(y, x_1, \ldots, x_n)$ je univerzálna funkcia pre všetky T-stroje počítajúce n-árne funkcie, kde y je číslo T-stroja. □

Veta 1. *Každú čiastočne rekurzívnu funkciu možno zostrojiť pomocou najviac jednej minimalizácie.*

Dôkaz. Nech $f(x_1, \ldots, x_n)$ je ľubovoľná, čiastočne rekurzívna funkcia. Podľa Lemy C existuje pre f T-stroj (nech má číslo j), ktorý ju počíta. Rovnako, ako v dôkaze Lemy D platí:

$$Tun(j, x_1, \ldots, x_n) = tobs(\mu_x(tvp(x, j, x_1, \ldots, x_n) = 0)) = f(x_1, \ldots, x_n).$$

Teda funkciu f možno zostrojiť pomocou najviac jednej minimalizácie, lebo funkcie *tobs* a *tvp* sú primitívne rekurzívne a dajú sa zostrojiť bez minimalizácie. □

Veta 2. *Každá totálna, čiastočne rekurzívna funkcia je rekurzívna.*

Dôkaz. Nech $f(x_1, \ldots, x_n)$ je ľubovoľná, čiastočne rekurzívna funkcia, ktorá je naviac totálna. Rovnako ako v dôkaze Vety 1 platí:

$$Tun(j, x_1, \ldots, x_n) = tobs(\mu_x(tvp(x, j, x_1, \ldots, x_n) = 0)) = f(x_1, \ldots, x_n),$$

kde j je číslo T-stroja, ktorý počíta f. Keďže f je totálna, musí existovať jediný a konečný výpočet (s nejakým číslom x) T-stroja s číslom j na každom vstupe $x_1, \ldots, x_n$. To ale znamená, že funkcia

$$g_j(x_1, \ldots, x_n) = \mu_x(tvp(x, j, x_1, \ldots, x_n) = 0)$$

je totálna. Naviac, *tvp* je primitívne rekurzívna a teda aj totálna a preto je minimalizácia (použitá v g_j) regulárna. Čiže, f sa dá zostrojiť pomocou jedinej minimalizácie, ktorá je regulárna a preto je f rekurzívna. □

Hlavná veta o ekvivalencii výpočtových modelov

Veta. *Pre každú funkciu $f : N^n \to N$ platí: f je čiastočne rekurzívna iff f je vypočítateľná nejakým T-strojom iff f je vypočítateľná nejakým registrovým strojom.*

Dôkaz: Pozri Lemy A až D.

Hlavná veta o rekurzívnych funkciách

Veta.

(a) *Existuje čiastočne rekurzívna funkcia, krorú nemožno dodefinovať na žiadnu rekurzívnu funkciu. (Teda, existuje čiastočne rekurzívna funkcia, ktorá nie je rekurzívna.)*

(b) *Existuje funkcia $f : N^n \to N$, ktorá nie je čiastočne rekurzívna.*

(c) *Existuje rekurzívna funkcia, ktorá nie je primitívne rekurzívna.*

Dôkaz (a)*:* Diagonalizáciou. *Kód* čiastočne rekurzívnej funkcie f je slovo (nad vhodnou abecedou Σ) obsahujúce popis konštrukcie funkcie f operáciami skladania, primitívnej rekurzie a minimalizácie z funkcií 0, s a I^n_m. Nech $w_0, w_1, w_2, \ldots$ sú lexikograficky usporiadané všetky kódy unárnych čiastočne rekurzívnych funkcií. Nech f_i je čiastočne rekurzívna funkcia s kódom w_i pre $i = 0, 1, 2, \ldots$ Možno dokázať, že existuje T-stroj M, ktorý vstup 01^j01^x0 pretransformuje na výstup $01^{f_j(x)}0$; (M najprv nájde w_j (postupne generujúc a testujúc všetky slová nad Σ) a potom zostrojí výstup $01^{f_j(x)}0$ poznajúc konštrukciu f_j popísanú v kóde w_j). Teda M počíta funkciu $G(j, x) = f_j(x)$ pre všetky j a x.

Z Lemy D dostaneme, že G je čiastočne rekurzívna funkcia. (G je univerzálna funkcia pre všetky unárne čiastočne rekurzívne funkcie.) Nech

$$H(j) = G(j, j) + 1 = f_j(j) + 1 \text{ pre všetky } j. \tag{1}$$

Funkcia H je čiastočne rekurzívna, lebo s, G a I_1^1 sú čiastočne rekurzívne a platí: $H(j) = s(G(I_1^1(j), I_1^1(j)))$.

Nech $\bar{H}$ je funkcia, ktorá vznikne z H jej dodefinovaním (ľubovoľným spôsobom) na nejakú totálnu funkciu. Dokážme teraz (sporom), že $\bar{H}$ nemôže byť čiastočne rekurzívna a teda ani rekurzívna. Ak by $\bar{H}$ bola čiastočne rekurzívna, potom by mala kód w_l pre najaké l, t.j. $\bar{H} = f_l$. Funkcia f_l je totálna, (lebo $f_l = \bar{H}$ a $\bar{H}$ je totálna) a preto, (podľa(1)), je hodnota $H(l)$ pre l definovaná a platí $H(l) = f_l(l) + 1$. Keďže $\bar{H}$ vznikla z H jej dodefinovaním na totálnu a hodnota $H(l)$ je pre l definovaná, musí pre l tiež platiť: $\bar{H}(l) = H(l)$. Z vyššie uvedených rovností dostaneme:

$$f_l(l) = \bar{H}(l) = H(l) = f_l(l) + 1,$$

čo je spor!

Preto funkcia $\bar{H}$ nie je čiastočne rekurzívna a teda ani rekurzívna. To znamená, že čiastočne rekurzívnu funkciu H nemožno dodefinovať na rekurzívnu.

Dôkaz (b)*:* Funkcia $\bar{H}$ z dôkazu (a) nie je čiastočne rekurzívna.

Iný dôkaz pre (b)*:* Množina všetkých unárnych čiastočne rekurzívnych funkcií je spočítateľná, (lebo množina ich kódov je spočítateľná), ale množina všetkých funkcií $f : N \to N$ nie je spočítateľná (jej kardinalita je c.)

Dôkaz (c)*:* Je podobný dôkazu pre (a); rozdiel je v tom, že $w_0, w_1, w_2, \ldots$ sú lexikograficky usporiadané všetky kódy unárnych primitívne rekurzívnych funkcií, (tieto kódy neobsahujú minimalizácie). Teda funkcie $f_0, f_1, f_2, \ldots$ sú primitívne rekurzívne. Rovnako, ako v dôkaze (a) sa dokáže, že funkcia

$$H(j) = f_j(j) + 1 \tag{2}$$

je čiastočne rekurzívna. Naviac, H je aj totálna, lebo každá f_j je primitívne rekurzívna a teda totálna. Preto je H rekurzívna, (podľa Vety 2). Ak by H bola primitívne rekurzívna, potom by mala kód w_l pre nejaké l, t.j., $H = f_l$. Táto rovnosť a (2) nám dáva:

$$f_l(l) = H(l) = f_l(l) + 1,$$

čo je spor! Preto je funkcia H rekurzívna, ale nie je primitívne rekurzívna.

Iný dôkaz pre (c)*:* O Ackermanovej funkcii, ktorá je pomerne ľahko definovateľná (a nie je tak abstrakná, ako funkcia H), sa dá tiež dokázať, že je rekurzívna, ale nie je primitívne rekurzívna. Dôkaz je ale komplikovanejší.

□

Churchova téza

Pojem *algoritmus* nemá presnú formálnu definíciu. Existujú len charakteristické črty tohto pojmu, napríklad:

(1) **Diskrétnosť:** Algoritmus je proces postupného zostrojovania veličín (t.j. čísel, slov,...), ktorý postupuje v *diskrétnom* čase. Na začiatku je daný konečný systém veličín a v každom ďalšom diskrétnom čase vznikne systém veličín zo systému veličín z predchádzajúceho diskrétneho času podľa istého pravidla/zákona/programu.

(2) **Determinističnosť:** (Zrejmé.)

(3) **Elementárnosť krokov:** Pravidlo, podľa ktorého vznikne nasledujúci systém veličín z predchádzajúceho, musí byť *jednoduché.*

(4) **Rezultatívnosť:** Ak sa vo výpočte už nedá pokračovať, musí byť určené, čo je výsledok.

(5) **Hromadnosť:** Počiatočný systém veličín sa môže vyberať z (potenciálne) nekonečnej množiny (t.j. existuje (potenciálne) nekonečne veľa vstupných inštancií).

Churchova téza. *Systém všetkých algoritmicky (čiastočne) vypočítateľných funkcií $f : N^n \to N$ je totožný so systémom všetkých (čiastočne) rekurzívnych funkcií.*

6 RAM počítače

RAM počítač je výpočtový model, ktorý je (na rozdiel od Turingovho stroja alebo Registrového stroja) dosť podobný reálnemu počítaču, ale má aj črty Turingovho stroja.

RAM počítač má:

- vstupnú konečnú pásku (rozdelenú na políčka), na ktorej sa pohybuje (ale len vpravo) čítacia hlava riadená inštrukciou READ (pozri tabuľku nižšie)

- výstupnú (doprava nekonečnú) pásku (rozdelenú na políčka), na ktorej sa pohybuje (tiež len vpravo) zapisovacia hlava riadená inštrukciou WRITE (pozri tabuľku nižšie)

- riadiacu jednotku, v ktorej je program zostavený z inštrukcií uvedených v tabuľke nižšie

- nekonečnú pamäť, ktorá pozostáva z registrov r_0, r_1, r_2,... Každý register a aj každé políčko vstupnej aj výstupnej pásky môže obsahovať ľubovoľne veľké celé číslo. Register r_0 je špeciálny - vykonávajú sa v ňom aritmetické operácie.

Nech $v(a)$ označuje hodnotu operanda a a nech $c(i)$ označuje obsah i-teho registra r_i. (Teda pamäť RAM počítača si môžeme predstaviť ako nekonečné celočíselné pole c.) Inštrukcie RAM počítača môžu mať operandy troch nasledujúcich typov:

$$= i \quad \text{alebo} \quad i \quad \text{alebo} \quad *i$$

kde i je celé číslo pre prvý typ a prirodzené číslo pre druhý a tretí typ. Hodnota takýchto operandov je určená nasledovne:

$$v(a) = \begin{cases} i, & \text{ak } a \text{ je typu ''}= i\text{''}, \\ c(i), & \text{ak } a \text{ je typu } i, \\ c(c(i)), & \text{ak } a \text{ je typu } *i, \text{ (nepriama adresácia).} \end{cases}$$

Príklad: Inštrukcia ADD a spôsobí pripočítanie hodnoty $v(a)$ do registra r_0 t.j. $c(0) \leftarrow c(0) + v(a)$. Teda,

$c(0) \leftarrow c(0) + 7$, ak a je operand $= 7$,
$c(0) \leftarrow c(0) + c(7)$, ak a je operand 7,
$c(0) \leftarrow c(0) + c(c(7))$, ak a je operand $*7$.

Cena inštrukcie RAM počítača.

Pre RAM počítače sa používa buď:

- *jednotková cena inštrukcie* (t.j., vykonanie každej inštrukcie trvá rovnakú jednotku času a každé celé číslo zaberá v pamäti, na vstupe aj výstupe rovnakú jednotku, alebo

- *logaritmická cena instrukcie* - tá je uvedená v tabuľke nižšie a je založená na tom, že na zápis celého čísla i (vrátane znamienka) v pamäti, na vstupe aj výstupe treba $l(i)$ bitov a stačí $l(i)+1$ bit , kde

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

Nech $t(a)$ je *logaritmická cena operanda* a, pre ktorú platí:

$t(=i) = l(i)$,
$t(i) = l(i) + l(c(i))$,
$t(*i) = l(i) + l(c(i)) + l(c(c(i)))$.

V nasledujúcej tabuľke sú uvedené inštrukcie RAM počítača, význam a logaritmická cena inštrukcií. Pod pojmom *vstup* (v riadkoch READ i a READ $*i$) myslíme obsah aktuálne čítaného políčka vstupnej pásky a podobne, pod pojmom *výstup* (v riadku WRITE a) myslíme aktuálne snímané políčko výstupnej pásky.

inštrukcia	význam	logaritmická cena
LOAD a	$c(0) \leftarrow v(a)$	$t(a)$
STORE i	$c(i) \leftarrow c(0)$	$l(c(0)) + l(i)$
STORE $*i$	$c(c(i)) \leftarrow c(0)$	$l(c(0)) + l(i) + l(c(i))$
ADD a	$c(0) \leftarrow c(0) + v(a)$	$l(c(0)) + t(a)$
SUB a	$c(0) \leftarrow c(0) - v(a)$	$l(c(0)) + t(a)$
MULT a	$c(0) \leftarrow c(0) \cdot v(a)$	$l(c(0)) + t(a)$
DIV a	$c(0) \leftarrow \lfloor c(0)/v(a) \rfloor$	$l(c(0)) + t(a)$
READ i	$c(i) \leftarrow$ čítaný vstup; tiež posun hlavy vpravo	$l(\text{vstup}) + l(i)$
READ $*i$	$c(c(i)) \leftarrow$ čítaný vstup; tiež posun hlavy vpravo	$l(\text{vstup}) + l(i) + l(c(i))$
WRITE a	hodnota $v(a)$ je zapísaná na výstup; tiež posun hlavy vpravo	$t(a)$
JUMP b	goto b	1
JGTZ b	if $c(0) > 0$ then goto b	$l(c(0))$
JZERO b	if $c(0) = 0$ then goto b	$l(c(0))$
HALT	stop	1

Časová zložitosť RAM programu rozpoznávajúceho jazyk

RAM program môže rozpoznávať jazyk $L \subseteq \Sigma^*$. V takom prípade očíslujeme symboly abecedy Σ číslami $1, 2, \ldots, |\Sigma|$ a namiesto slova $w = w_1 w_2 \ldots w_m$, kde $w_i \in \Sigma$ pre každé i, má RAM na vstupe zodpovedajúcu postupnosť $kod(w) = kod(w_1), \ldots, kod(w_m)$, kde $kod(w_i)$ je číselný kód symbolu w_i zapísaný v i-tom políčku vstupnej pásky.

Definícia. *RAM program rozpoznáva jazyk $L \subseteq \Sigma^*$, ak pre každé slovo $w \in \Sigma^*$ platí: $w \in L$ iff RAM program sa na zodpovedajúcom vstupe $kod(w)$ zastaví a na výstupe má zapísanú 1.*

Nech RAM program rozpoznáva jazyk $L \subseteq \Sigma^*$. Nech w je ľubovoľné slovo z Σ^* a nech $S_{kod(w)}$ je suma logaritmických cien všetkých inštrukcií vykonaných RAM programom počas výpočtu na vstupe $kod(w)$.

Definícia. *Časová zložitosť RAM programu rozpoznávajúceho jazyk* $L \subseteq \Sigma^*$ *s logaritmickou cenou inštrukcií je funkcia* $T(n) = max\{S_{kod(w)} | w \in \Sigma^n\}$. *Ak v sume* $S_{kod(w)}$ *je jednotková cena inštrukcií, namiesto logaritmickej ceny, potom* $T(n)$ *je časová zložitosť RAM programu rozpoznávajúceho jazyk* $L \subseteq \Sigma^*$ *s jednotkovou cenou inštrukcií.*

Veta A. *Nech* L *je jazyk rozpoznávaný deterministickým T-strojom* M *v čase* $T(n) \geq n$. *Potom existuje RAM program rozpoznávajúci* L *s časovou zložitosťou* $O(T(n))$ *resp.* $O(T(n) \log T(n))$ *v prípade jednotkovej resp. logaritmickej ceny inštrukcií.*

Dôkaz: RAM program simuluje (krok za krokom) T-stroj M nasledovne. Pre každý vnútorný stav T-stroja M má RAM program vhodný podprogram simulujúci činnosť T-stroja M v tomto stave. Nech má M vstupnú pásku a $k-1$ pracovných pások (spolu k pások). Obsah i-teho políčka j-tej pásky T-stroja M si RAM program pamätá v $(k \cdot i + j + d)$-tom registri; vhodná konštanta d umožňuje RAM programu pamätať si v prvých d registroch informácie potrebné na simuláciu T-stroja M, (napríklad pozície hláv na páskach). Teda počas simulácie $T(n)$ krokov nepoužije RAM program viac než $k \cdot T(n) + k + d$ registrov, lebo $i \leq T(n)$, (keďže žiadna z hláv T-stroja M sa nedostane vpravo za $T(n)$-té políčko pásky) a $j \leq k$. V $(j+k)$-tom registri, $(1 \leq j \leq k)$, si RAM program pamätá aktuálnu pozíciu hlavy na j-tej páske a v j-tom registri si pamätá číslo $k \cdot c(j+k) + j + d$, čo je adresa registra obsahujúceho aktuálne čítané políčko j-tej pásky. Obsah toho registra zistí RAM program pomocou nepriamej adresácie, keďže $c(c(j)) = c(k \cdot c(j+k) + j + d)$.

Na začiatku výpočtu RAM program prečíta všetky políčka jeho vstupnej pásky a uloží ich do registrov reprezentujúcich vstupnú pásku T-stroja M. Potom RAM program simuluje T-stroj M spôsobom uvedeným vyššie, pričom jeden krok výpočtu T-stroja M dokáže RAM program simulovať pomocou $O(1)$ svojich inštrukcií. Teda $O(T(n))$ je časová zložitosť RAM programu simulujúceho M v prípade jednotkovej ceny inštrukcií, keďže M vykoná $T(n)$ krokov.

V prípade logaritmickej ceny inštrukcií je situácia trochu iná. Keďže RAM program použije počas simulácie najviac $kT(n)+k+d$ registrov, (pozri vyššie), potom adresy použitých registrov sú čísla najviac $O(T(n))$. Pri simulácii sú páskové symboly T-stroja M kódované prirodzenými číslami a preto obsahy registrov simulujúcich políčka pások T-stroja M sú čísla rozsahu $O(1)$. Ostatné čísla, ktoré sa vyskytnú počas výpočtu RAM programu v rámci simulácie, (napríklad výpočty pozícií hláv na páskach, výpočty adries pre nepriamu adresáciu, zisťovanie číselných kódov páskových symbolov čítaných hlavami T-stroja M), sú tiež čísla najviac $O(T(n))$.

Preto $O(\log T(n))$ [resp. $O(T(n) \log T(n))$] je časová zložitosť RAM programu s logaritmickou cenou inštrukcií simulujúceho jeden krok výpočtu [resp. celý výpočet] T-stroja M. □

Veta B. *Nech L je jazyk rozpoznávaný RAM programom s logaritmickou cenou inštrukcie v čase $T(n)$. Potom L možno rozpoznávať deterministickým T-strojom v čase $O(T^3(n))$. Naviac, ak RAM nepoužíva inštrukciu MULT ani DIV, potom L možno rozpoznávať deterministickým T-strojom v čase $O(T^2(n))$.*

Dôkaz. RAM program je simulovaný T-strojom nasledovne. T-stroj má simulovaný program zakódovaný vo svojej riadiacej jednotke a pre každú inštrukciu v RAM programe má T-stroj podprogram, ktorým ju simuluje. T-stroj má, okrem vstupnej pásky, ešte 4 pracovné pásky. Na prvej pracovnej páske má T-stroj zapísaný aktuálny obsah registra r_0. Obsah 2. pracovnej pásky je tvaru

$$\#\#i_1\#c(i_1)\#\#i_2\#c(i_2)\#\#\ldots\#\#i_u\#c(i_u)BB\ldots$$

pre nejaké u, kde B je blank symbol, i_j je číslo registra a $c(i_j)$ je jeho obsah; čísla i_j a $c(i_j)$ sú zapísané v binárnom tvare. Ďalej, $i_j \geq 1$ pre každé j, lebo obsah registra r_0 je na prvej páske. Ak simulovaná inštrukcia zapisuje do m-tého registra, pre nejaké $m \geq 1$, potom T-stroj pripíše slovo $\#\#m\#c(m)$ na koniec neblankového úseku 2. pracovnej pásky (ešte počas simulácie tejto inštrukcie). Teda, T-stroj nájde aktuálny obsah i-teho tegistra tak, že najprv na 2. pracovnej páske nájde najľavejší blank a potom prechádza túto pásku doľava hľadajúc najpravejší výskyt slova $\#\#i\#$. Potom číslo $c(i)$ bezprostredne vpravo za nájdeným slovom je aktuálny obsah i-teho regista. Tretiu pracovnú pásku používa T-stroj na pomocné výpočty a na 4. pracovnej páske simuluje výstupnú pásku RAMu.

Ukážme teraz, ako T-stroj simuluje inštrukcie ADD $*40$ a STORE 33 a z toho bude jasný princíp simulácie aj ďalších inštrukcií.

Pri simulácii inštrukcie ADD $*40$, T-stroj nájde na 2. pracovnej páske (spôsobom popísaným vyššie) aktuálny obsah 40-teho registra, t.j. číslo $c(40)$ a číslo $c(40)$ skopíruje na 3. pracovnú pásku. Rovnakým spôsobom potom nájde na 2. pracovnej páske aktuálny obsah $c(40)$-teho registra, t.j. číslo $c(c(40))$, ktoré tiež skopíruje na 3. pracovnú pásku. Na záver simulácie inšrukcie ADD $*40$ pripočíta číslo $c(c(40))$ z 3. pracovnej pásky k číslu $c(0)$ na prvej pracovnej páske. Je zrejmé, že postačujúci čas pre T-stroj na simulovanie tejto inštrukcie, (ale bez času potrebného na prehľadávanie na 2. pracovnej páske), je $O(l(40) + l(c(40)) + l(c(c(40))) + l(c(0)))$, pričom $l(40)+l(c(40))+l(c(c(40)))+l(c(0))$ je logaritmická cena inštrukcie ADD $*40$.

Pri simulácii inštrukcie STORE 33, T-stroj najprv nájde koniec neblankového úseku 2. pracovnej pásky a tam pripíše slovo ##33#, za ktoré následne skopíruje číslo $c(0)$ z 1. pracovnej pásky. Tiež je zrejmé, že postačujúci čas pre T-stroj na simulovanie tejto inštrukcie, (ale bez času potrebného na prehľadávanie na 2. pracovnej páske), je $O(l(33) + l(c(0)))$, pričom $l(33) + l(c(0))$ je logaritmická cena inštrukcie STORE 33.

Nech $I_1, I_2, \ldots, I_h$ je postupnosť inštrukcií vykonaných RAM programom na vstupe $kod(w)$, kde $w \in \Sigma^n$ a nech $cena(I_t)$ je logaritmická cena inštrukcie I_t pre $t = 1, \ldots, h$. Podľa definície časovej zložitosti pre logaritmickú cenu inštrukcií platí: $S_{kod(w)} = \sum_{t=1}^{h} cena(I_t) \leq T(n)$ a preto $h \leq T(n)$, keďže $cena(I_t) \geq 1$ pre všetky t.

Dá sa ľahko dokázať, že ak je na koniec neblankového úseku 2. pracovnej pásky pripísané nejaké slovo $\#\#j\#c(j)$ počas simulácie inštrukcie I_t, potom platí:

$$|\#\#j\#c(j)| \leq cena(I_t) + 5. \tag{3}$$

(Pripomeňme, že j aj $c(j)$ sú na páske zapísané v binárnom tvare a na ich zápis stačí $l(j)+1$ resp. $l(c(j))+1$ bit vrátane znamienka.) Napríklad, ak I_t je STORE $*i$, potom na 2. pracovnú pásku je pripísané slovo $\#\#c(i)\#c(c(i))$, kde $c(c(i)) = c(0)$ a teda

$$|\#\#c(i)\#c(c(i))| = |\#\#c(i)\#c(0)| \leq l(c(i))+l(c(0))+5 \leq cena(\text{STORE}*i)+5.$$

V prípade inštrukcí STORE i, READ $*i$ a READ i je dôkaz podobný; v prípade iných inštrukcií sa obsah 2. pracovnej pásky nemení, lebo iné inštrukcie nezapisujú do žiadneho registra r_i, kde $i \geq 1$.

Z (3) teda dostaneme, že celková dĺžka neblankového úseku 2. pracovnej pásky je najviac $\Sigma_{t=1}^{h}(cena(I_t) + 5) \leq T(n) + 5h = O(T(n))$, lebo $h \leq T(n)$), pozri vyššie.

Ak RAM program nepoužíva inštrukcie MULT ani DIV, potom T-stroj dokáže simulovať každú inštrukciu I_t v čase $O(T(n)) + O(cena(I_t))$, lebo:

1. Čas potrebný na nájdenie najľavejšieho blanku na 2. pracovnej páske alebo na nájdenie aktuálneho obsahu i-teho registra (prípadne $c(i)$-teho registra v prípade nepriamej adresácie) na tejto páske je $O(T(n))$, lebo dĺžka neblankového úseku tejto pásky je $O(T(n))$, (pozri vyššie).
2. Čas potrebný na ostatné činnosti v rámci simulácie inštrukcie I_t je $O(cena(I_t))$, čo možno zdôvodniť podobne, ako pri popise simulácie inštrukcií ADD $*40$ a STORE 33, pozri vyššie.

Teda, celkový čas potrebný na simuláciu postupnosti $I_1, I_2, \ldots, I_h$ je $O(hT(n)) + O(\Sigma_{t=1}^{h} cena(I_t)) = O(T^2(n)) + O(T(n)) = O(T^2(n))$.

V prípade, keď RAM program používa inštrukcie MULT a DIV je T-stroj schopný simulovať každú inštrukciu I_t v čase $O(cena^2(I_t)) + O(T(n)) = O(T^2(n))$, lebo $cena(I_t) \leq \sum_{i=1}^{h} cena(I_i) \leq T(n)$, pozri vyššie a teda, celkový čas potrebný na simuláciu postupnosti $I_1, I_2, \ldots, I_h$ je $O(T^3(n))$ □

Dôsledok. *RAM programy s logaritmickou cenou inštrukcií a deterministické T-stroje sú polynomiálne ekvivalentné.*

Dôkaz. Vyplýva z Vety A a z Vety B. □

Poznámka. Na T-stroji je problematické efektívne simulovať výpočet RAM programu s jednotkovou cenou inštrukcií, lebo v takom prípade môže RAM program počas $O(n)$ krokov vypočítať čísla veľkosti až $\Omega(2^{2^n})$, napríklad takýmto programom:

$$x \leftarrow 2;\ \text{for } i \leftarrow 1 \text{ to } n \text{ do } x \leftarrow x * x$$

ale na zápis takýchto čísel na pásku T-stroja treba až $\Omega(2^n)$ políčok pásky a teda aj toľko času.