

TEORETIČNE OSNOVE RAČUNALNIŠTVA 2

zapiski predavanj
poletni semester 2007

Predavatelj: prof. dr. Borut Robič

Zapiske digitaliziral: Jernej Petaros

Zapiski so nastali kot digitalna oblika zapiskov s predavanj, zato je način razlage in podajanja snovi izključno predavateljev. Zapiski so bili pregledani, vendar za popolnost vsebine ter za morebitne napake, ki so še ostale v zapiskih, ne prevzemam odgovornosti.

ver 1.1.0

Kazalo

1 Turingov stroj	4
1.1 Uvod	4
1.2 Model Turingovega stroja	4
1.3 Izračunljivi jeziki in funkcije	6
1.4 Konstrukcija Turingovega stroja	9
1.5 Različice Turingovih strojev	11
1.5.1 Turingov stroj z dvosmernim trakom	12
1.5.2 Večtračni Turingov stroj	13
1.5.3 Nedeterministični Turingov stroj	15
1.5.4 Večdimenzionalni Turingov stroj	16
1.5.5 Turingov stroj z več okni	17
1.6 Stroj z naslovljivim pomnilnikom - RAM	17
1.7 Splošno rekurzivne funkcije	18
1.8 Turingovi stroji kot generatorji	20
2 Problemi in jeziki	23
2.1 Prevedba problemov	24
2.2 Osnovne lastnosti Turingovih jezikov	25
2.3 Kodiranje Turingovih strojev	27
2.4 Jezik, ki ni Turingov	29
2.5 Problem, ki je neodločljiv	29
2.6 Turingov jezik, ki ni rekurziven	30
2.7 Prevedbe	32
2.7.1 Riceov izrek	36
2.8 Postov korespondenčni problem (PKP)	37
2.9 Dvournost kontekstno neodvisne gramatike	37
2.10 Diofantske enačbe	39
2.11 Problem tlakovanja	39
2.12 Problem domin	41
3 Zahtevnost (kompleksnost) izračunov	41
3.1 Prostorska zahtevnost	43
3.2 Časovna zahtevnost	44
3.3 Nedeterministična prostorska in časovna zahtevnost	45
3.4 Razredi zahtevnosti (kompleksnostni razredi)	45
3.5 Linearno stiskanje in druge transformacije	45
3.5.1 Uvod	45
3.5.2 Linearno stiskanje	46
3.5.3 Odpravljanje trakov	46
3.6 Linearna pospešitev in druge transformacije	47
3.6.1 Uvod	47
3.6.2 Linearna pospešitev	48
3.6.3 Odpravljanje trakov in časovna zahtevnost	49
3.7 Hierarhije	50
3.7.1 Hierarhije razredov DSPACE	52
3.7.2 Hierarhije razredov DTIME	53
3.7.3 Relacije med časovno in prostorsko zahtevnostjo	54
3.7.4 Hierarhije razredov NTIME in NSPACE	56
3.7.5 Izrek o vrzeli	57
3.7.6 Izrek o uniji	58
3.7.7 Polinomsko omejen čas in prostor	59
3.8 Splošno o prevedbah	61
3.9 Vprašanje $P \stackrel{?}{=} NP$ in prevedbe	62

3.10 Dokazovanje NP-polnosti (težkosti)	64
3.11 Problem izpolnljivosti Boolovih izrazov	65
3.11.1 Problem izpolnljivosti je NP-poln (Cook)	65

1 Turingov stroj

1.1 Uvod

- je standardna formalizacija intuitivnega pojma algoritma
- verjamemo, da je sposoben računati vse, kar se (intuitivno) računati da (Church–Turingova teza)
⇒ sposoben izračunati vse, kar lahko izračuna (najmočnejši) računalnik
- čeprav: včasih ne tako hitro ali pa ne na tako majhnem prostoru kot računalnik (torej ne za tako nizko ceno)
- Zakaj? Ker ima zelo enostavno zgradbo
- toda: zaenkrat nas cena še ne zanima (zahtevnost), zaenkrat nas zanima le če rezultat sploh lahko izračunamo (izračunljivost)

Turingov stroj si je zamislil Alan Turing leta 1936.

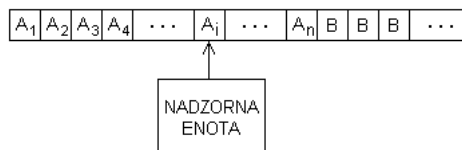
- zamislil si je človeka, ko rešuje nek problem
 - glava, ki razmišlja
 - roka s pisalom
 - medij, kamor zapisuje (papir)
- to je hotel še bolj poenostaviti v obliki nekakšnega stroja
- zaradi lažje formalizacije je zahteval še, da je stroj:
 - sestavljen iz končno mnogo elementov
 - deluje v diskretnih korakih

1.2 Model Turingovega stroja

Osnovni model Turingovega stroja

Osnovni model Turingovega stroja sestavljajo:

- trak (ki ustreza papirju)
 - potencialno **neskončen v eno smer**
 - razdeljen v **celice**
 - v vsaki celici je zapisan **simbol** iz **končne abecede**
 - v vsakem trenutku je le končno mnogo celic “zasedenih”, le v končno mnogo celicah je lahko simbol, ki je različen od **B (blank)**
- nadzorna enota (ki ustreza človekovi glavi)
 - je **končna**
 - v vsakem trenutku je v enem od **končno mnogo stanj**
 - lahko bere in spreminja simbole na traku s pomočjo nekakšnega **čitalnega okna**
 - v vsakem trenutku je preko čitalnega okna dostopna le ena celica na traku
 - na začetku je okno nad prvo celico traku (nad skrajno levo celico)



- čitalno okno (ki ustreza roki s pisalom)

Delovanje

Stroj v enem koraku opravi naslednje:

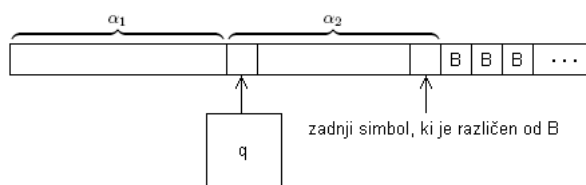
- preide v novo stanje
- zapiše nov simbol v celico pod oknom
- okno premakne levo ali desno

Definicija: Turingov stroj je sedmerka $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kjer je:

- Q končna množica stanj
- Σ končna množica vhodnih simbolov (vhodna abeceda)
- Γ končna množica tračnih simbolov ($\Sigma \subset \Gamma$)
- δ funkcija prehodov: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, D\}$, L in D označujeta premik v levo ali desno, funkcija prehodov je lahko parcialna
- q_0 začetno stanje: $q_0 \in Q$
- B simbol za presledek (blank): $B \in \Gamma$
- F množica končnih stanj: $F \subseteq Q$

Trenutni opis

- $\text{TO} \stackrel{\text{def}}{=} \Gamma^* \times Q \times \Gamma^* \dots$ množica vseh trenutnih opisov
- trenutni opis je trojka $(\alpha_1, q, \alpha_2) \in \text{TO}$, ki opisuje situacijo:



- α_1 in α_2 lahko tudi vsebujeta simbole B (presledke)
- torej:
 - če je $\alpha_1 = \epsilon$ je okno na skrajni levi celici
 - če je $\alpha_2 = \epsilon$ je okno nad B (nad prvim B od nešteto B-jev na desni strani)
- dogovor: namesto (α_1, q, α_2) pišemo kar $\alpha_1 q \alpha_2$ (zato zahtevamo $Q \cap \Gamma = \emptyset$)

Relacija \vdash_M (relacija “neposredno daje/sledi”)

- na množici TO definiramo relacijo \vdash_M
- **Definicija:** Naj bosta $x, y \in \text{TO}$. Tedaj:

$$x \vdash_M y \quad \text{če } y \text{ neposredno sledi iz } x\text{-a z enim korakom stroja } M$$

- Natančneje: Naj bo $x_1 x_2 \dots x_{i-1} q x_i x_{i+1} \dots x_n$ trenutni opis. Če $\delta(q, x_i) = (p, Y, L)$, potem:
 - če je $i = 1$ se izračun konča (ker je trak levo omejen)

- če je $i > 1$: $x_1x_2 \dots x_{i-1}qx_ix_{i+1} \dots x_n \vdash_M x_1x_2 \dots x_{i-2}px_{i-1}Yx_{i+1} \dots x_n$
- Če $\delta(q, x_i) = (p, Y, D)$, potem:
 - $x_1x_2 \dots x_{i-1}qx_ix_{i+1} \dots x_n \vdash_M x_1x_2 \dots x_{i-1}Ypx_{i+1} \dots x_n$

Tranzitivno reflektivna ovojnica \vdash_M^*

- **Definicija:** $x \vdash_M^* y$ če $\exists k \geq 0$ in \exists zaporedje $x = x_0, x_1, x_2, \dots, x_k = y$, da je $x_0 \vdash_M x_1 \wedge x_1 \vdash_M x_2 \wedge \dots \wedge x_{k-1} \vdash_M x_k$

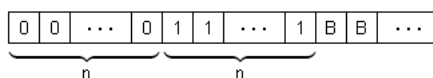
1.3 Izračunljivi jeziki in funkcije

Jezik Turingovega stroja

- **Definicija:** Jezik Turingovega stroja M je $L(M) = \{w | w \in \Sigma^* \wedge q_0w \vdash_M^* w_1qw_2 \wedge q \in F \wedge w_1, w_2 \in \Gamma^*\}$
- Opomba:
 - Če je beseda v jeziku $w \in L(M) \iff$ se M znajde v končno korakih v nekem končnem stanju (če mu na vhod damo besedo w)
 - Če pa beseda ni v jeziku $w \notin L(M)$, tedaj se lahko zgodi dvoje:
 - * M se ustavi v nekem nekončnem stanju (in ne more dalje)
 - * M se nikoli ne ustavi

Primer: Turingov stroj, ki sprejema jezik $\{0^n1^n | n \geq 1\}$

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- na začetku trak vsebuje besedo 0^n1^n



- stroj ponavlja osnovni cikel:
 - zamenjaj skrajno levo 0 z 1
 - premakni okno v desno do prve 1
 - zamenjaj to 1 z Y
 - vrni se levo do skrajno desnega X
 - premakni se v desno za eno mesto
- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
 - $q_0 \dots$ začetno stanje
 - $q_1 \dots$ premikanje v desno do 1
 - $q_2 \dots$ po zamenjavi 1 z Y se premika v levo do X
 - $q_3 \dots$ v to stanje preide, ko najde X
 - $q_4 \dots$ končno stanje

• δ :

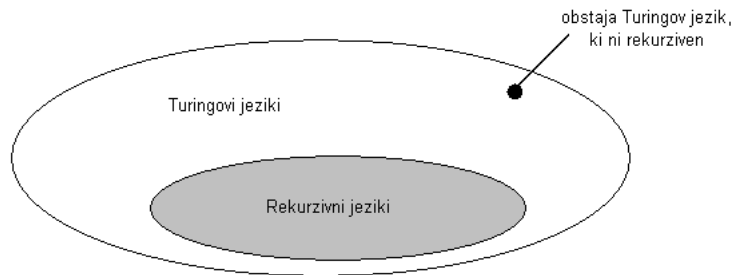
	0	1	X	Y	B
q_0	(q_1, X, D)	-	-	(q_3, Y, D)	-
q_1	$(q_1, 0, D)$	(q_2, Y, L)	-	(q_1, Y, D)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, D)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, D)	(q_4, B, D)
q_4	-	-	-	-	-

Primer: Izračun za besedo 0^21^2 :

$$q_00011 \vdash_M Xq_1011 \vdash_M X0q_111 \vdash_M Xq_20Y1 \vdash_M \dots$$

Izračunljivi jeziki in funkcije

- Turingovi jeziki
 - so jeziki, ki jih sprejemajo Turingovi stroji
 - **Definicija:** Jezik je Turingov jezik, če obstaja Turingov stroj M , da velja $L(M) = L$ (angl. recursively enumerable, r.e.)
 - Med Turingovimi jeziki so nekateri takšni, da pri njih ne moremo algoritmično odločiti o pripadnosti besede temu jeziku, torej ne obstaja algoritem (ne obstaja Turingov stroj), ki bi za vsako besedo x odgovoril z da ali ne na vprašanje “Ali je x v tem jeziku?”
 - Naj bo jezik L takšen in naj bo $L = L(M)$, tedaj se M ne ustavi vsaj na eni besedi $x \in \bar{L}$
 - Med Turingovimi jeziki pa so tudi takšni, pri katerih lahko algoritmično odločimo o pripadnosti neke besede temu jeziku, torej obstaja algoritem (obstaja Turingov stroj), ki za vsako besedo x odgovori z da ali ne na vprašanje “Ali je beseda x v tem jeziku?”. Take jezike imenujemo rekurzivni.
 - Kasneje bomo pokazali, da obstaja Turingov jezik, ki ni rekurziven.



Turingov stroj kot računalnik funkcij

- Imamo Turingov stroj M . Na trak zapišemo $0^{i_1}10^{i_2}1 \dots 10^{i_k}$, kjer so $i_j \in \mathbb{N}$
- Lahko se zgodi, da se M ustavi in ima tedaj na traku 0^m (m ničel, katerim sledijo sami presledki B), tedaj si lahko mislimo, da je stroj M izračunal neko funkcijo (oz. njeno vrednost) $f : \mathbb{N}^k \rightarrow \mathbb{N}$ pri argumentih i_1, i_2, \dots, i_k , torej, da je $m = f(i_1, i_2, \dots, i_k)$
- Funkcija f ni nujno definirana pri vseh možnih k -terkah, v splošnem je parcialna. To je takrat, ko:
 - se pri neki k -terki M ustavi, toda na traku ni beseda oblike 0^m
 - se M ne ustavi
- Opazimo: isti M računa eno funkcijo $\mathbb{N} \rightarrow \mathbb{N}$, drugo $\mathbb{N}^2 \rightarrow \mathbb{N}$, tretjo $\mathbb{N}^3 \rightarrow \mathbb{N}$ itd.
- Funkcijam, ki jih Turingovi stroji računajo na tak način, pravimo parcialno rekurzivne funkcije

Parcialno rekurzivne funkcije

- **Definicija:** Funkcija $f : \mathbb{N}^k \rightarrow \mathbb{N}$ je parcialno rekurzivna, če obstaja Turingov stroj M , da velja, da M računa vrednosti funkcije f na zgoraj opisani način
- rekurzivna funkcija je lahko tudi totalna, tedaj je totalno rekurzivna funkcija
- Vse običajne aritmetične funkcije so totalne, npr. $+$, $*$, $n!$, 2^n , $\lceil \log n \rceil$, itd.

Primer: Seštevanje naravnih števil je parcialno rekurzivna funkcija ($\mathbb{N}^2 \rightarrow \mathbb{N}$).

Dokaz: Sestavimo TS, ki računa vsoto $m + n$; $m, n \in \mathbb{N}$

Skica:

1. na začetku je na traku beseda $0^m 10^n$, na koncu bi radi na traku imeli $0^{(m+n)}$
2. stroj zbriše prvi znak 0 iz dela 0^m
3. nato premakne glavo nad znak 1
4. namesto njega zapiše 0
5. in konča, na traku je $0^{(m+n)}$

Primer: Množenje naravnih števil je parcialno rekurzivna funkcija ($\mathbb{N}^2 \rightarrow \mathbb{N}$).

Dokaz: Sestavimo TS, ki računa produkt $m \cdot n$; $m, n \in \mathbb{N}$

Skica:

1. na začetku je na traku beseda $0^m 10^n$, na koncu bi radi na traku imeli $0^{(m \cdot n)}$
2. stroj na konec besede zapiše 1, na traku imamo $0^m 10^n 1$
3. premakne glavo na začetek (na najbolj levi od blank različni znak)
4. zbriše znak 0, če tega ne more storiti (pod glavo je znak 1), nadaljuje s korakom 6.
5. premakne glavo na naslednjo 1
6. skopira podzaporedje 0^n na konec besede
7. nadaljuje s korakom 3.
8. pobriše del $10^n 1$
9. in konča, na traku je $0^{(m \cdot n)}$

Primer: Potenciranje m^n ; $m, n \in \mathbb{N}$ je parcialno rekurzivna funkcija.

Primer: Funkciji $n!$ in $\lceil \log n \rceil$ sta parcialno rekurzivni funkciji.

Primer: Funkcija $\dot{-}$ (čisto odštevanje) je parcialno rekurzivna funkcija.

Definicija:

$$m \dot{-} n = \begin{cases} m - n & ; \text{če } m \geq n \\ 0 & ; \text{sicer} \end{cases}$$

Dokaz: Sestavimo Turingov stroj

Skica:

1. na začetku je na traku beseda $0^m 10^n$
2. stroj zamenja skrajno levo 0 z B
3. v desno poišče podzaporedje 10
4. zamenja 0 (iz podzaporedja 10) z 1
5. v levo išče B
6. nadaljuje s korakom 2.
7. na koncu bo ostalo na traku $B^m 11^n 0^{(m-n)}$

Cikel se konča, če:

- v desno ne najde 10, na traku imamo $0^{(m-n)}$
- ne najde skrajno leve ničle ($m < n$), vse je spremenil v B

1.4 Konstrukcija Turingovega stroja

- V moč Turingovih strojev se lahko prepričamo tako, da z njimi rešimo čim več nalog (t.j. da na njem “sprogramiramo” čimveč nalog)
- Toda: Turingov stroj je (zaradi svoje enostavnosti) zelo okoren za programiranje

⇒ Obstaja nekaj prijemov, s katerimi olajšamo programiranje

- uporaba nadzorne enote kot pomnilnika
- uporaba večslednega traku
- prestavljanje vsebine traku (shiftanje)
- uporaba podprogramov
- ...

Uporaba nadzorne enote za pomnjenje

- Zamisel: Vsako stanje je sestavljeno iz dveh komponent
 - ena je pravo, čisto stanje
 - druga je le shramba (za en tračni simbol)

⇒ $Q = K \times \Gamma$ (K je množica čistih stanj)

Primer: Turingov stroj za razpoznavanje besed, katerih prvi znak se v besedi ne ponovi.

Skica: $M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, \bar{q}_0, B, F)$, kjer je začetno stanje \bar{q}_0 urejen par $\bar{q}_0 = [q_0, B]$.

- $Q = \{q_0, q_1\} \times \{0, 1, B\} = \{[q_0, 0], [q_0, 1], [q_0, B], [q_1, 0], [q_1, 1], [q_1, B]\}$
 - $F = \{[q_1, B]\}$
 - δ funkcija opravlja nekaj nalog
 - spravi prvi znak v “stanje”:
 - $\delta([q_0, B], 0) = ([q_1, 0], 0, D)$
 - $\delta([q_0, B], 1) = ([q_1, 1], 1, D)$
 - dokler ne najde enakega znaka, se pomika v desno:
 - $\delta([q_1, 0], 1) = ([q_1, 0], 1, D)$
 - $\delta([q_1, 1], 0) = ([q_1, 1], 0, D)$
 - če pride do konca besede, gre v končno stanje:
 - $\delta([q_1, 0], B) = ([q_1, B], B, L)$
 - $\delta([q_1, 1], B) = ([q_1, B], B, L)$
 - če pa najde isti znak, kot je v spominu, se zablokira:
 - $\delta([q_1, 0], 0) = \text{nedefinirano}$ (pustimo brez definicije prehodov)
 - $\delta([q_1, 1], 1) = \text{nedefinirano}$
- ne ve kaj narediti, ne sprejme, a se ustavi

Večsledni trak

- Zamisel: Trak si predstavljamo kot da je sestavljen iz več sledi, kot prikazuje skica.

				...
				...
				...
				...

- **Definicija:** Tračna abeceda je:

$$\Gamma = \Gamma_1 \times \Gamma_2 \times \Gamma_3 \times \dots \times \Gamma_k,$$

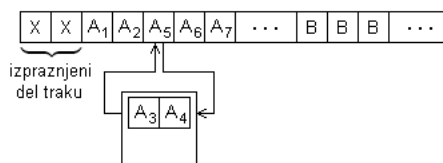
kjer je Γ_i tračna abeceda i -te sledi.

Primer: Želimo sestaviti Turingov stroj, ki preveri ali je število na vhodu praštevilo. Skica Turingovega stroja:

- Trak sestavljajo 3 sledi. Začetna situacija je sledeča:
 - na 1. sledi je (binarno) zapisano vhodno število
 - na 2. sledi je (binarno) zapisano število 2
 - 3. sled je prazna
- Koraki:
 - v prvem koraku prepiše število iz 1. sledi na 3. sled
 - nato odšteva število, ki je na 2. sledi od števila na 3. sledi
 - če se odštevanje zaključi z ostankom 0, stroj konča in zavrne vhod (ni praštevilo), če pa se odštevanje konča z ostankom ($0 < \text{ostanek} < \text{delitelj}$) pa poveča število na 2. traku (delitelj) za 1.
 - če povečano število še ni enako vhodnemu, ponovi vse korake znova, če pa je enako vhodnemu pa stroj konča in sprejme vhod (je praštevilo)

Prestavljanje vsebine traku

- Včasih želimo prestaviti vsebino traku in tako narediti prostor za kasnejšo rabo
- Kako to narediti? Zamisel:



Primer: Za premik tračne vsebine v desno za 2 mesti, so stanja oblike:

$$Q = K \times \Gamma^2$$

Opomba: premikanje se konča pri znaku B.

Skica Turingovega stroja:

- Q vsebuje stanja oblike $[q, A_1, A_2]$, kjer je $q \in \{q_1, q_2\}$ in $A_1, A_2 \in \Gamma$
- X je dodaten, poseben tračni simbol, s katerim označimo sproščena mesta
- δ (zanimivi deli):
 - $\delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, D) \dots$ začetek premikanja, 1. znak si zapomni, na traku pa ga nadomesti z X
 - $\delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, D) \dots$ tudi 2. znak si zapomni in ga na traku nadomesti z X
 - $\delta([q_1, A_i, A_{i+1}], A_{i+2}) = ([q_1, A_{i+1}, A_{i+2}], A_i, D) \dots$ znak si zapomni, prvega v vrsti pomnjenja pa zapiše na trak
 - $\delta([q_1, A_{n-1}, A_n], B) = ([q_1, A_n, B], A_{n-1}, D)$

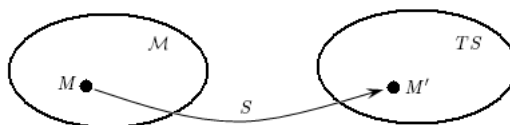
$$- \delta([q_1, A_n, B], B) = ([q_2, B, B], A_n, L)$$

Podprogrami

- Zamisel: Del Turingovega stroja služi kot podprogram
- Zato ima podprogram tudi svoje vhodno in izhodno stanje
- Ko glavni program želi izvedbo podprograma, naredi naslednje:
 - zapomni si tekoče stanje (npr. ga zapiše na posebno sled)
 - preide v vhodno stanje podprograma
 - izvaja podprogram
 - ob koncu podprograma preide v izhodno stanje
 - iz izhodnega stanja preide v (prejšnje) tekoče stanje in nadaljuje glavni program
- parametre lahko prenaša v podprogram in iz njega na dva načina:
 - preko posebne sledi (ki je neomejena)
 - preko pomnilnika v nadzorni enoti (ki pa je omejen)

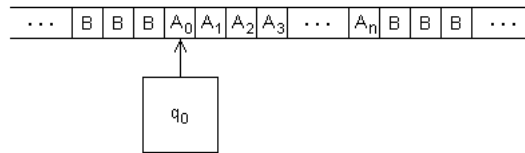
1.5 Različice Turingovih strojev

- Ponovimo: V moč Turingovih strojev se skušamo prepričati tako, da čimveč problemov rešimo z njimi
- Druga možnost: da pokažemo, da so navidez drugačni modeli računanja kvečjemu enakovredni osnovnemu Turingovemu stroju
- Drugačni modeli:
 - razne vrste Turingovih strojev
 - * dvosmerni
 - * večtračni
 - * nedeterminističen
 - * večdimenzionalen
 - * ...
 - RAM (Random Access Machine)
 - splošno rekurzivne funkcije
 - lambda račun
 - ...
- Kako doseči to drugo možnost? Metoda je naslednja:
 - Naj bo \mathcal{M} razred računskih modelov (strojev) za katerega želimo pokazati, da je (kvečjemu) enakovreden razredu Turingovih strojev
 - tedaj moramo najti sistematično proceduro S (simulacija), ki vsakemu (poljubnemu) modelu $M \in \mathcal{M}$ priredi Turingov stroj $M' = S(M)$, ki je sposoben simulirati M , se pravi izračunati enak rezultat kot M .



1.5.1 Turingov stroj z dvosmernim trakom

- edina razlika glede na osnovni Turingov stroj je, da je trak neomejen na obe smeri
- vhodna beseda je nekje na traku in stroj gleda na prvi znak te besede in je v začetnem stanju
- vse celice, ki niso del besede, so prazne (blank)



- Formalno: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ kot pri osnovnem Turingovem stroju, le da funkciji prehodov δ sedaj ni treba skrbeti za levi rob

Trditev: Turingov stroj z dvosmernim trakom ni šibkejši od Turingovega stroja z enosmernim trakom.

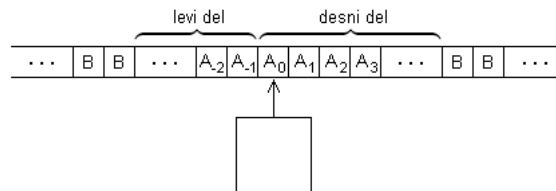
Dokaz: Postavimo mejnik in preko njega ne gremo v levo stran. Na tak način lahko simuliramo nek Turingov stroj z enosmernim trakom.

Vprašanje: Ali je razred dvosmernih Turingovih strojev močnejši?

Odgovor: Ne.

Izrek: Za jezik L obstaja Turingov stroj z dvosmernim trakom natanko tedaj, ko za jezik L obstaja osnovni Turingov stroj (z enosmernim trakom).

Dokaz: Naj bo M (poljubni) dvosmerni Turingov stroj. Označimo:

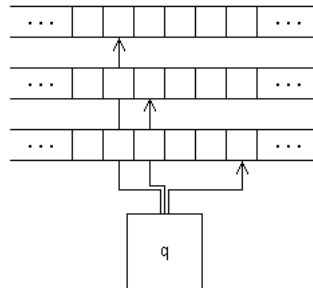


Stroju M priredimo enosmerni dvosledni Turingov stroj M' takole:

- zgornja sled predstavlja M -jev trak od A_0 v desno (desni del)
- spodnja sled predstavlja M -jev trak od A_{-1} v levo (levi del)
- prva celica na spodnjem traku vsebuje znak $\#$ (ki ni del tračne abecede), ki M' pove, da je dosegel levi rob in da mora namesto premika v levo preiti na drugo sled
- Delovanje M' :
 - upošteva vedno le eno sled (izjema pri $\#$)
 - * zgornjo, kadar bi bil M na desnem delu
 - * spodnjo, kadar bi bil M na levem delu
 - po zgornji sledi počne M' isto, kot bi M počel po desnem delu traku (izjema je prehod na levi del)
 - po spodnji sledi počne M' isto, kot bi M počel po levem delu traku (izjema je prehod na desni del), vendar je smer premikov ravno nasprotna
 - če M' po spodnji sledi doseže znak $\#$, preide na zgornjo sled (upošteva po A_0)
 - če M' po zgornji sledi doseže znak $\#$, namesto premika v levo preide na spodnjo sled na celico A_{-1}

1.5.2 Večtračni Turingov stroj

- ima $k \geq 2$ trakov, vsak je neskončen v obe smeri
- nadzorna enota ima k oken, nad vsakim trakom eno, okna niso "povezana", lahko se premikajo neodvisno eno od drugega



- na začetku je vhodna beseda na prvem traku in okno je na njenem prvem znaku, ostali trakovi so takrat prazni
- korak stroja - v odvisnosti od stanja v katerem je nadzorna enota in znakov v oknih stroj naredi sledeče:
 - preide v novo stanje
 - neodvisno vpiše znake v okna
 - neodvisno premakne okna levo ali desno za eno mesto (lahko imamo tudi možnost, da okna ne premakne)

Vprašanje: Ali je večtračni Turingov stroj močnejši od osnovnega?

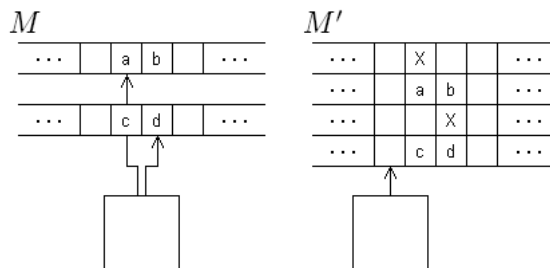
Odgovor: Ne.

Izrek: Za jezik L obstaja večtračni Turingov stroj natanko tedaj, ko za L obstaja osnovni Turingov stroj.

Dokaz:

(\Leftarrow) Večtračni Turingov stroj naj uporablja samo en trak.

(\Rightarrow) Naj bo M večtračni Turingov stroj. Stroju M priredimo $2k$ -sledni osnovni Turingov stroj M' , po spodnji skici.



Lastnosti stroja M' so sledeče:

- ima po dve sledi za vsak trak stroja M :
 - na zgornji sledi je oznaka X , ki pove kje bi bilo okno stroja M
 - na spodnji sledi je vsebina, ki bi bila na tem traku stroja M

- ima nadzorno enoto, ki vsebuje:
 - stanje
 - prostor za k tračnih simbolov
 - števec (ki pove, koliko znakov X je še desno od trenutne pozicije okna stroja M')
- M' simulira en korak stroja M tako:
 - pomika okno v desno in vsakokrat, ko najde X , si zapomni znak pod njim v prostor v nadzorni enoti in zmanjša števec X -ov za 1.
 - ko števec doseže 0, se začne vračati v levo
 - vsakokrat, ko pri vračanju v levo na kaki sledi opazi znak X :
 - * pod njim zamenja tračni simbol, tako kot bi ta znak zamenjal stroj M (na ustreznem traku)
 - * premakne X v levo ali desno (kot bi to storil stroj M)
 - * števec X -ov poveča za 1
 - ko števec doseže k , se premikanje v levo konča
 - tedaj nadzorna enota preide v novo stanje (kot bi to pri stroju M)

★ Upočasnitev:

- Pri simuliranju stroj M' napravi več korakov, da simulira en korak stroja M . Koliko največ?
- Recimo, da je M naredil i korakov, tedaj sta skrajno levo in skrajno desno okno kvečjemu oddaljena za $2i$
- M' bo za simulacijo i -tega koraka porabil:
 - $2i$ korakov za sprehod v desno (do skrajno desnega X)
 - $2i$ korakov za sprehod v levo (do skrajno levega X)
 - $\leq 2k$ dodatnih korakov, če je treba X premakniti v desno
- M' bo za simulacijo m korakov stroja M rabil kvečjemu:

$$\sum_{i=1}^m (4i + 2k) = 4 \sum_{i=1}^m i + 2mk = 4 \cdot \frac{1}{2} \cdot m(m+1) + 2mk = \mathcal{O}(m^2)$$

⇒ Pri simulaciji večtračnega Turingovega stroja z osnovnim Turingovim strojem je potrebna kvečjemu kvadratnaupočasnitev (če večtračni Turingov stroj reši problem v polinomskem času, potem tudi osnovni Turingov stroj reši isti problem v polinomskem času).

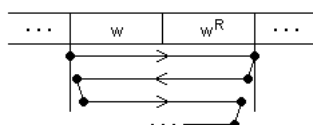
Vprašanje: Ali je kvadratnaupočasnitev nujna? Ali bi se dalo (v splošnem) simulirati npr. v času $\mathcal{O}(n \cdot \log n)$ ali celo $\mathcal{O}(n)$?

Odgovor: V splošnem se to ne da. Obstaja jezik, kjer je kvadratnaupočasnitev nujna (pri simulaciji večtračnega Turingovega stroja z osnovnim).

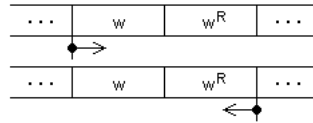
Primer: Jezik palindromov: $L = \{ww^R \mid w \in \Sigma^*\}$

Enotračni osnovni TS razpozna L tako, da teka od enega znaka, do njemu zrcalnega znaka (glede na sredino besede) in ju primerja.

Za to porabi $\mathcal{O}(n^2)$, $n = |ww^R|$.



Dvotračni TS pa najprej skopira besedo s prvega na drugi trak. Nato premakne eno od oken na nasprotno stran besede in nato okni pomika in primerja znake pod njima. Za to porabi $\mathcal{O}(n)$, $n = |ww^R|$.



1.5.3 Nedeterministični Turingov stroj

- $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, D\}}$ ($\mathcal{P}(Q \times \Gamma \times \{L, D\})$)

\Rightarrow pri nekem stanju in tračnem simbolu ima Turingov stroj lahko na izbiro več kot eno potezo (trojko) iz $Q \times \Gamma \times \{L, D\}$

- Toda: na izbiro je vedno končno mnogo potez (lahko tudi nobena). Zakaj? Ker $|\delta(q, a)| \leq |Q| \cdot |\Gamma| \cdot 2$

Primer: Možne so štiri poteze:

$$\delta(q_0, a) = \{(q_1, b, L), (q_2, c, D), (q_3, a, L), (q_4, B, D)\}$$

Vprašanje: Katero potezo bo nedeterministični Turingov stroj (NTS) izbral?

Odgovor: NTS izbere potezo, ki ga vodi k sprejetju vhoda (če je sprejetje možno). NTS se vede, kot da bi imel v nadzorni enoti magični kovanec.

Definicija: NTS sprejme vhodno besedo \Leftrightarrow obstaja končno zaporedje korakov, ki se konča v končnem stanju.

Jasno: NTS zmore vse, kar zmore osnovni TS.

Vprašanje: Ali je NTS močnejši od osnovnega TS? Ali lahko izračuna kaj, česar osnovni TS ne more?

Odgovor: Ne.

Dokaz: : Naj bo M nek NTS. Naj ima njegov program v vsaki množici $\delta(q, a)$ kvečjemu r možnih potez.

Velja: $r \leq |Q| \cdot |\Gamma| \cdot 2 \Rightarrow r$ je končno število.

Stroju M priredimo osnovni 3-sledni TS M' , ki:

- ima na prvi sledi vhodno besedo stroja M
- na drugi sledi generira zaporedje navodil (= besed nad abecedo $\{1, 2, \dots, r\}$) po naraščajoči dolžini
- Npr.: če $r = 3$: 1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, \dots , 12222212
- na tretji sledi simulira stroj M , kot da bi stroj M izbiral svoje poteze skladno s tekočim navodilom (vsakokrat ko potrebuje navodilo ga sproti generira)

...	1	#	2	...	#	1	2	2	X	2	2	2	X	1	2	#	...
-----	---	---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	-----

- Simulacija - stroj M' :

1. na drugi sledi sestavi naslednje navodilo
2. zbriše tretjo sled in nanjo prepíše vhodno besedo, okno postavi na začetek
3. na tretji sledi oponaša stroj M , kot da bi ta izbiral svoje poteze po tekočem navodilu. Pri tem:

- če pride do konca navodila in se znajde v končnem stanju, sprejme vhod in konča
- sicer (če ne pride v končno stanje ali ne pride do konca navodila) nadaljuje s 1. korakom

Trditev: Če M sprejme besedo, jo sprejme tudi M' ($x \in M(L) \Rightarrow x \in L(M')$)

Dokaz: Če $x \in L(M)$, jo M sprejme v končnem številu korakov, npr. v k korakih i_1, i_2, \dots, i_k . Simulator M' bo navodilo i_1, i_2, \dots, i_k sestavil po končno mnogo korakov (po tem, ko bo preizkusil druga navodila, ki so pred i_1, i_2, \dots, i_k). Po tem bo simuliral M in tudi sprejel x . Zato $x \in L(M')$.

Izrek: Za jezik L obstaja NTS natanko tedaj, ko za L obstaja osnovni TS.

D.N.: Koliko največ korakov potrebuje TS, da simulira en korak NTS?

1.5.4 Večdimenzionalni Turingov stroj

- "trak" je k -dimenzionalen ($k \geq 2$), neomejen v vseh $2k$ smereh
- nadzorna enota ima okno nad eno celico
- vhod je zapisan v isti (npr. 1) dimenziji (ni razpršen)
- Korak stroja: na osnovi stanja in znaka nad oknom spremeni vsebino celice, stanje in se premakne v eni od $2k$ smeri
- Ali je večdimenzionalni TS močnejši od osnovnega TS?
- Naj bo M k -dimenzionalni TS. V vsakem trenutku je po končno mnogo korakov le končno mnogo celic polnih (ne blankov). Zato v vsakem trenutku vse polne celice lahko zapremo v neko k -dimenzionalno škatlo (podprostor), ki je končna. Vsebino te škatle pa lahko zapišemo v eni dimenziji, "vrstico za vrstico".
- k -dimenzionalnemu TS M priredimo enodimenzionalni osnovni dvosledni TS M' :
 - na prvi sledi so ena za drugo vrstice iz "škatle" stroja M
 - na drugi sledi pa je znak, ki pove, kje bi bilo okno stroja M

Primer: $k = 2$: M ima v nekem trenutku na traku:

...	B	B	B	a_1	a_2	B	B	a_3	a_4	a_5	B	B	a_6	a_7	B	B	a_8	a_9	B	B	a_{10}	a_{11}	...
	a_4	a_5	B	B	a_6	a_7	B	B	a_8	a_9	B	B	a_{10}	a_{11}									
	a_8	a_9	B	B	B	B	B	a_{10}	a_{11}														

M' ima na svojem traku

	B	B	B	a_1	a_2	B	B	a_3	*	a_4	a_5	B	B	a_6	a_7	B	B	*	a_8	a_9	B	B	B	B	a_{10}	a_{11}	
												X															

Simulacija (pri $k = 2$):

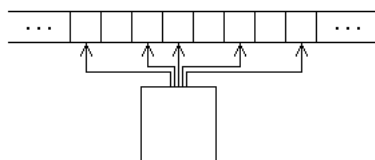
- če bi stroj M zapisal nek znak (v celico), M' zapiše isti znak (v sled nad znakom X)
- če M preide v neko stanje, tudi M' preide v ustrezno stanje
- če M naredi premik znotraj škatle se M' premakne znotraj vrstice ali pa na sosednjo vrstico
- če M zapusti škatlo v smeri gor/dol, stroj M' doda novo vrstico na levo/desno

- če M zapusti škatlo v levo/desno, M' podaljša vsako vrstico na levi/desni (s prestavljanjem vsebine traku)

Izrek: za jezik L obstaja k -dimenzionalni ($k \geq 2$) TS natanko tedaj, ko za jezik L obstaja osnovni TS.

1.5.5 Turingov stroj z več okni

- nadzorna enota ima končno mnogo oken, ki se neodvisno (nepovezano) pomikajo levo ali desno



- **Vprašanje:** Ali osnovni TS zmore vse, kar zmore TS z več okni?

Odgovor: : Da.

- TS M , ki ima $k \geq 2$ oken, simuliramo z osnovnim strojem M' , ki ima $k + 1$ sledi in:
 - na prvi sledi je vsebina, kot bi bila na pri M ,
 - na ostalih sledeh pa beleži, kje bi bila okna stroja M .

Izrek: za jezik L obstaja TS s $k \geq 2$ okni natanko tedaj, ko za jezik L obstaja osnovni TS.

1.6 Stroj z naslovljivim pomnilnikom - RAM

- ima potencialno neskončno pomnilnih celic, ki so oštevilčene
- vsaka celica lahko vsebuje poljubno veliko ampak končno celo število
- ima končno mnogo registrov, ki lahko hranijo poljubno velika cela števila
- nekatera cela števila se lahko dekodirajo v običajne ukaze (aritmetične, logične, razvejitvene)
- delovanje:
 - lokacijski register vsebuje naslov naslednje celice, iz katere bo prenesel vsebino (ukaz) v enega od registrov
 - ukaz se tam dekodira na operacijo ali operand oz. njegov naslov

⇒ če je nabor ukazov primerno izbran, RAM lahko simulira poljuben današnji računalnik

Vprašanje: Ali je RAM močnejši od TS?

Odgovor: Ne.

Izrek: TS lahko simulira RAM (če so ukazi RAM izračunljivi po Turingu).

Dokaz: Uporabimo večtračni TS.

Ukazni cikel:

1. trak za lokacijski register vsebuje i
2. TS bere prvi trak in išče naslov $\#i$
3. če ga najde, sledi analiza vsebine v_i (dekodiranje), nadzorna enota gre v neko stanje (npr. $v_i = \text{ADD } 2 \ j$)
4. sledi izvedba ukaza, npr. poišči naslov j , prenesi vsebino v_j , prištej število 2
5. poveča vsebino lokacijskega registra (traku)

1.7 Splošno rekurzivne funkcije

Definicija: Funkcija $f : \mathbb{N}^n \rightarrow \mathbb{N}$ je dobljena iz funkcij

$$g : \mathbb{N}^m \rightarrow \mathbb{N} \quad \text{in}$$

$$h_i : \mathbb{N}^n \rightarrow \mathbb{N}, \quad i = 1, 2, \dots, m$$

s **kompozicijo**, če je $f(x_1, \dots, x_n) = g(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n))$

Definicija: Funkcija $f : \mathbb{N}^{(n+1)} \rightarrow \mathbb{N}$ je dobljena iz funkcij

$$g : \mathbb{N}^n \rightarrow \mathbb{N} \quad \text{in}$$

$$h : \mathbb{N}^{(n+2)} \rightarrow \mathbb{N}$$

s **primitivno rekurzijo**, če je za $\forall y \in \mathbb{N}$:

$$\begin{aligned} f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n) \\ f(x_1, \dots, x_n, y + 1) &= h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)) \end{aligned}$$

Definicija: Ničelna funkcija $Z : \mathbb{N} \rightarrow \mathbb{N}$ je $Z(x) = 0$ za $\forall x \in \mathbb{N}$.

Definicija: Naslednik je funkcija $N : \mathbb{N} \rightarrow \mathbb{N}$, kjer je $N(x) = x + 1$ za $\forall x \in \mathbb{N}$.

Definicija: Projekcija je funkcija $\pi_i^n : \mathbb{N}^n \rightarrow \mathbb{N}$, kjer $\pi_i^n(x_1, \dots, x_n) = x_i$ (projekcija π_i^n izmed n argumentov vrne i -ti argument).

Iz funkcij Z, N, π_i^n lahko z uporabo kompozicije in primitivne rekurzije sestavimo nove funkcije.

Definicija: Funkcija f je **primitivno rekurzivna funkcija** (PRF), če je ena od funkcij Z, N, π_i^n ali pa je dobljena iz primitivno rekurzivnih funkcij s končno mnogokratno uporabo kompozicije in primitivne rekurzije.

Razred PRF je zelo velik. Vsebuje vse običajne funkcije, ki jih rabimo v praksi, pa tudi druge bolj eksotične funkcije.

Primer:

$$\text{add}(m, n) = m + n$$

$$\text{multiply}(m, n) = m \cdot n$$

$$\text{pow}(m, n) = m^n$$

$$\text{minus}(m, n) = m \dot{-} n$$

$$\max(x_1, \dots, x_n) = m + n$$

$$\min(x_1, \dots, x_n) = m + n$$

$$n!$$

$$\lfloor \log_2 n \rfloor$$

$$n \uparrow m = n \left. \begin{matrix} n \\ n \\ \dots \\ n \end{matrix} \right\} m\text{-krat}$$

$$\text{neg}(x) = \begin{cases} 1 & \text{za } x = 0 \\ 0 & \text{za } x \geq 1 \end{cases}$$

$$\text{and}(x, y) = \begin{cases} 1 & \text{za } x \geq 1 \text{ in } y \geq 1 \\ 0 & \text{sicer} \end{cases}$$

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

Toda: razred PRF ne vsebuje vseh funkcij, ki jih smatramo za intuitivno izračunljive.

Dokaz: konstruiramo funkcijo g in:

1. opišemo, kako jo izračunati
2. pokažemo, da g ni PRF

Korak 1: PRF lahko oštevilčimo, torej obstaja bijekcija med PRF in \mathbb{N} . Kako jih oštevilčimo?

- izberemo abecedo Σ za:
 - opis osnovnih funkcij, npr. Z, N, π_i^n
 - opis uporabe kompozicije in primitivne rekurzije
- sistematično generiramo besede iz Σ^* v kanonskem vrstnem redu
- ko je beseda generirana, preverimo ali je definicija (izpeljava) opis kake PRF
- če je, jo obdržimo, sicer jo zavržemo

\Rightarrow rezultat je zaporedje A_1, A_2, A_3, \dots besed iz Σ^* , kjer je A_i i -ta izpeljava

- obratno: če je f PRF, obstaja izpeljava, ki je končna. To je beseda v Σ^* , ki je element zaporedja A_1, A_2, A_3, \dots
- Opomba: število argumentov, ki jih zahteva i -ta funkcija f_i , je razviden iz opisa A_i -te funkcije, zato bomo pisali kar $f_i(x_1, \dots, x_m)$

Korak 2: Definiramo novo funkcijo $g(n) \stackrel{def.}{=} f_n(n, n, \dots, n) + 1$

Korak 3: $g(n)$ je intuitivno izračunljiva

Algoritem:

- generiraj n -to definicijo A_n
- uporabi A_n za izračun f_n nad argumenti n, n, \dots, n
- prištej 1

Korak 4: Toda g ni primitivno rekurzivna funkcija

Dokaz: Dokažemo s protislovjem. Predpostavimo, da g je PRF.

Tedaj: $\exists k : g = f_k$ (k končen)

Koliko je tedaj $g(k)$?

$$g(k) = \begin{cases} f_k(k, k, \dots, k) \\ f_k(k, k, \dots, k) + 1 \end{cases} > \text{protislovje}$$

Toda: Ali je to nujno protislovje? Kaj pa če je $f_k(k, k, \dots, k) \uparrow$ (nedefiniran), potem je lahko $f_k(k, k, \dots, k) = f_k(k, k, \dots, k) + 1$ saj je oboje nedefinirano.

Ali so PRF lahko nedefinirane?

Odgovor: Ne. PRF je totalna.

Dokaz: Z, N, π_i^n so totalne in kompozicija ter primitivna rekurzija ohranjata totalnost.

Primer: Ackermanova funkcija $A : \mathbb{N}^2 \rightarrow \mathbb{N}$, kjer:

$$\begin{aligned} A(0, y) &= 1 \\ A(1, 0) &= 2 \\ A(x, 0) &= x + 2, \text{ za } x \geq 2 \\ A(x + 1, y + 1) &= A(A(x, y + 1), y) \end{aligned}$$

Ackermanova funkcija ni primitivno rekurzivna.

Če želimo definirati vse (intuitivno) izračunljive funkcije, moramo k kompoziciji in primitivni rekurziji dodati še kako pravilo za definiranje novih funkcij.

Definicija: Če $g(x_1, \dots, x_n, z)$ funkcija potem

$$\mu z[g(x_1, \dots, x_n, z)] = \begin{cases} \text{najmanjši } z \text{ pri katerem je } g(x_1, \dots, x_n, z) = 0 \\ \uparrow, \text{ če takega } z \text{ ni} \end{cases}$$

Definicija: Funkcija $f : \mathbb{N}^n \rightarrow \mathbb{N}$, je dobljena iz funkcije $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ z **minimizacijo**, če je:

$$f(x_1, \dots, x_n) = \mu z[g(x_1, \dots, x_n, z)]$$

Definicija: Funkcija f je **splošno rekurzivna funkcija** (SRF), če je ena od funkcij Z, N, π_i^n ali pa je dobljena iz splošno rekurzivnih funkcij s končno mnogokratno uporabo kompozicije, primitivne rekurzije in minimizacije.

Izrek: Funkcija je splošno rekurzivna natanko tedaj, ko je parcialno rekurzivna (tj. ko je izračunljiva s Turingovim strojem).

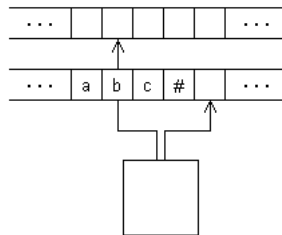
1.8 Turingovi stroji kot generatorji

Turingov stroj lahko uporabljamo kot:

- sprejemnik jezika
- računalnik funkcij
- generator jezika

Naj bo M dvotračni (ali večtračni) TS, ki ima:

- vsaj en delovni trak
- en izhodni trak:
 - kamor lahko le piše (če nekaj zapiše, tega ne more več zbrisati), nanj zapisuje besede iz Σ^*
 - okno se premika le v desno
 - ko izpiše neko besedo, za njo zapiše ločilo (#)



Definicija: Jezik, ki ga generira tak TS M je $G(M) \stackrel{def.}{=} \{w \mid w \in \Sigma^* \wedge M \text{ izpiše } w \text{ na izhodni trak}\}$
Opombe:

- če M teče končno dolgo je $G(M)$ končna (\Rightarrow regularen jezik)
- če se M nikoli ne ustavi je $G(M)$ lahko končna:
 - kadar sicer ne izpisuje več, a vseeno nekaj računa

- kadar izpisuje le besede, ki jih je že prej izpisal
- ali pa je $G(M)$ neskončna
- besede lahko izpiše večkrat
- besede se ne izpisujejo v kakem posebnem vrstnem redu

Kakšni so jeziki, ki jih generirajo TS?

Lema: Če TS generira nek jezik L potem je jezik L Turingov jezik.

Dokaz: M' (sprejemnik) je tak kot stroj M (generator), le da ima še dodaten vhodni trak. Ko M' dobi vhodno besedo x :

- začne simulirati generator M
- vsakokrat, ko generira besedo, jo primerja z vhomom x
- če sta besedi enaki, pomeni, da je beseda x tudi v jeziku in M' vhod sprejme

Očitno: $x \in G(M) \Leftrightarrow x \in L(M')$, torej velja: $G(M) = L(M')$

Opomba: če $x \notin G(M)$, potem:

- če je $G(M)$ neskončna, se M' nad x ne ustavi
- če je $G(M)$ končna:
 - se M' ustavi in zavrne x ali
 - se M' ne ustavi, kadar generator M brez prestanka generira ene in iste besede ali pa računa brez prestanka in več ne izpisuje

Kaj pa obratno? Ali velja: Če je jezik L je Turingov, potem nek TS generira jezik L .

Naj bo L Turingov jezik, tj. $L = L(M)$ za nek TS M . Kako sestaviti TS M' , da bo $L = G(M')$? Zamisel 1:

- M' naj na delovnem traku po vrsti sestavlja besede iz Σ^* v nekem vrstnem redu, npr. v leksikografskem vrstnem redu.
- ko sestavi besedo w_j :
 - začne simulirati M nad w_j
 - če M sprejme w_j , M' izpiše w_j na izhodni trak
- Problem: če $w_j \notin L(M)$ (če $L(M)$ ni rekurziven) se lahko zgodi, da se simulacija M nad w_j ne konča, zato M' naslednjih besed w_{j+1}, w_{j+2}, \dots nikoli ne sestavi in preveri, zato celo tistih, kiso v $L(M)$, nikoli ne generira

\Rightarrow Preprečiti je treba neskončno simulacijo stroja M nad neko besedo

Zamisel 2:

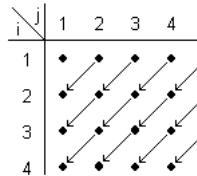
- M' naj po vrsti sestavlja pare $(i, j) \in \mathbb{N}^2$, in sicer vsak par natanko enkrat
- ko sestavi par (i, j) :
 - simulira M nad w_j kvečjemu i korakov (število korakov $\leq i$)
 - če M sprejme w_j natanko v i -tem koraku, potem M' izpiše w_j na izhodni trak

Podrobnosti: Generiranje parov

Sestavljanje parov v zaporedju $(1, 1), (1, 2), (1, 3), \dots$ ni v redu, ker se nekateri pari nikoli ne pojavijo, npr. par $(2, 1)$

Vprašanje: Kako sestavljati pare $(i, j) \in \mathbb{N}^2$, da se vsak par pojavi po končno korakih?

Odgovor: Tako kot prikazuje spodnja slika



Vsak par se pojavi v končnem času, tj. par (i, j) bo $\frac{1}{2}(i+j-1)(i+j-2) + i$ par po vrsti. Ta preslikava $\mathbb{N}^2 \rightarrow \mathbb{N}$ je bijektivna.

TS za generiranje parov:

- na traku ima $i + j$ ničel in enko med njima
- enico vsakokrat prestavi v desno
- ko pride do zadnje ničle, doda eno ničlo in vse ponovi

Kako vzpostaviti bijekcijo med Σ^* in \mathbb{N} (kako zagotoviti, da bodo w_1, w_2, \dots res vse besede iz Σ^*)?

Podrobnosti: Kanonska razvrstitev besed iz Σ^*

- najprej po naraščujoči dolžini besed
- enako dolge besede pa po njihovi vrednosti
- npr. $\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots$

Lema: Če je L Turingov jezik, ga generira nek TS.

Dokaz: Naj bo $w \in L(M) \Rightarrow M$ sprejme w v $i < \inf$ korakih. Naj bo $w = w_j$ v kanonski razvrstitvi. M' bo v končno korakih sestavil par (i, j) in po simulaciji stroja M nad w_j , izpisal w_j

Izrek: Jezik L je Turingov natanko tedaj, ko L generira nek TS.

Torej, če je L Turingov jezik, lahko njegove besede naštejemo (generiramo) z nekim TS.

Vprašanje: Kaj pa, če je L rekurziven? Ali lahko njegove besede generiramo v kakem posebnem vrstnem redu?

Odgovor: Da.

Lema: Če je L rekurziven, ga nek TS generira v kanonskem vrstnem redu.

Dokaz: Ker je L rekurziven, zamisel 1 postane uporabna.

Naj bo $L = L(M)$. Generator M' deluje tako:

- na delovnem traku v kanonskem vrstnem redu sestavlja besede iz Σ^*
- ko sestavi besedo, nad njo simulira stroj M
- če M besedo:

- sprejme, jo M' izpiše
- zavrne, jo M' ne izpiše
- besede, ki jih M' izpiše so v kanonskem vrstnem redu (in so natanko besede iz L) in vsaka beseda je izpisana natanko enkrat

Vprašanje: Ali velja tudi obratno?

Odgovor: Da.

Lema: Če lahko jezik L generiramo v kanonskem vrstnem redu, je L rekurziven.

Dokaz: $L = L(G)$, L v kanonskem vrstnem redu

Sprejemnik M deluje tako:

- ima vhodno besedo x na vhodnem traku
- simulira generator M in po vrsti generira besede
- ko generira novo besedo w jo primerja z x :
 - če $w = x$, M' sprejme x in konča
 - če $w > x$ ali je w za besedo x v kanonskem vrstnem redu, M' zavrne x in konča

Izrek: Jezik L je rekurziven natanko tedaj, ko L generira nek TS v kanonskem vrstnem redu.

2 Problemi in jeziki

Naj bo P odločitveni problem (odgovor je DA ali NE).

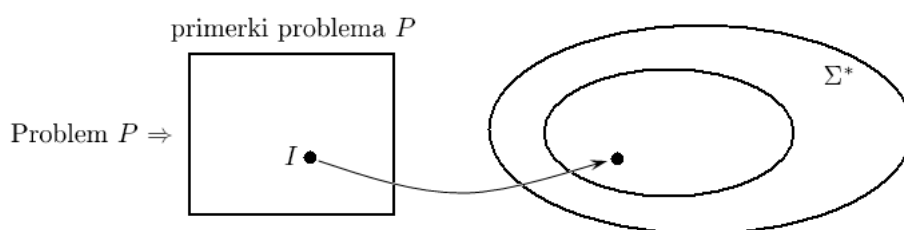
Npr. "Ali je graf Hamiltonov?"

Primerki I (instance) problema P dobimo, če formalne parametre v P nadomestimo z dejanskimi.

Npr. "Ali je nek točno določen graf Hamiltonov?"

Če izberemo neko kodirno shemo, lahko vsak $I \in P$ zapišemo z besedo v Σ^* . Kode vseh primerkov tvorijo nek jezik v Σ^* (vsaka beseda iz Σ^* ni koda kakega primerka $I \in P$).

Torej:



Vendar: Nekateri primerki $I \in P$ imajo odgovor DA, drugi NE (čeprav ni nujno, da odgovor poznamo).

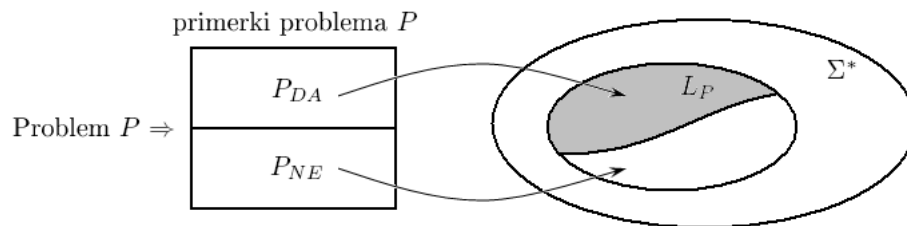
Naj bo

$$P_{DA} = \{I \mid I \in P \wedge \text{odgovor na } I \text{ je DA}\}$$

in

$$P_{NE} = \{I \mid I \in P \wedge \text{odgovor na } I \text{ je NE}\}$$

Tedaj:



Jezik L_P (množica kod pozitivnih primerkov problema P) je jezik, ki pripada problemu P .

Vprašanje “Ali ima nek primerek $I \in P$ odgovor DA?” je enkovredno vprašanju “Ali je neka beseda $w \in L_P$?”, kjer je w koda primerka I .

Uspešnost odgovarjanja na vprašanje, le je neka beseda w v jeziku L_P ($w \in L_P$), pa je odvisna od L_P :

- če je L_P rekurziven, potem \exists TS (= algoritem), ki sprejme ali zavrne besedo w v končno mnogo korakov, ne glede na to, kje je beseda w (v L_P ali $\overline{L_P}$)
- Torej: če je L_P rekurziven, potem obstaja algoritem, ki za $\forall I \in P$ v končno mnogo korakov **odloči** ali $I \in P_{DA}$ ali $I \in P_{NE}$
- če pa L_P ni rekurziven sta dve možnosti:
 1. L_P je Turingov, ne pa rekurziven
 2. L_P ni niti Turingov

Če velja 1.: \forall TS $\in M$: $\exists w \notin L_P$: M se ne ustavi nad w , kar pomeni, da $\exists I \in P_{NE}$: M nikoli ne ugotovi, da je odgovor NE.

Če velja 2.: \forall TS $\in M$: $\exists w \in L_P \wedge \exists w' \in \overline{L_P}$: M se ne ustavi nad w in niti ne nad w' , kar pomeni, da: $\exists I \in P_{DA} \wedge \exists J \in P_{NE}$: M nikoli ne ugotovi odgovorov na I in J .

Definicija: Problem P je **odločljiv** (decidable), kadar je L_P rekurziven. Če pa L_P ni rekurziven, pravimo, da P **ni odločljiv** (v literaturi zasledimo tudi pol-odločljive (semi-decidable) probleme).

2.1 Prevedba problemov

Naj bosta A in B dva problema.

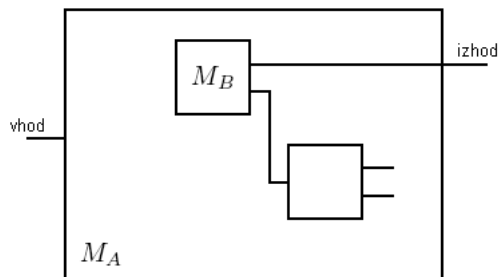
Denimo, da znamo sestaviti TS (= algoritem) M_A za reševanje problema A , če predpostavimo, da imamo na voljo TS M_B za reševanje problema B .

M_A vsebuje hipotetični stroj M_B in izkorišča njegove odgovore:

- lahko so to DA ali NE, če je M_B algoritem
- lahko so samo DA, če je M_B TS

M_A te odgovore izkorišča, tako da:

- preoblikuje v svoj odgovor
- jih uporabi kot vhod v nekdrug TS



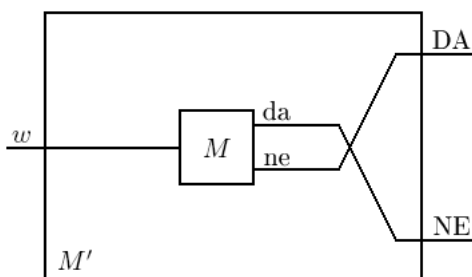
Sestavljanje M_A za problem A z uporabo stroja M_B (za problem B) imenujemo **prevedba** problema A na problem B .

Če problem A prevedemo na problem B pokažemo, da je problem B vsaj toliko težak kot problem A .

2.2 Osnovne lastnosti Turingovih jezikov

Izrek 1: Komplement rekurzivnega jezika je rekurzivni jezik.

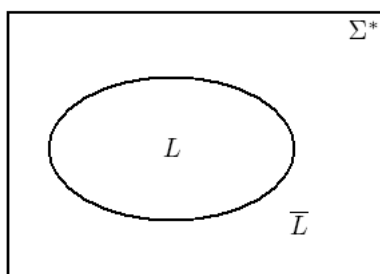
Dokaz: Naj bo L rekurziven in naj bo M TS z lastnostjo $L(M) = L$ (torej M sprejema jezik L , odloča z DA ali NE). Uporabimo M pri sestavljanju novega stroja M' (ki sprejema \bar{L}) tako:



Očitno:

- M' je algoritem (torej se ustavi za vsak vhod)
- $\Rightarrow L(M')$ je rekurziven
- $L(M') = \bar{L}$

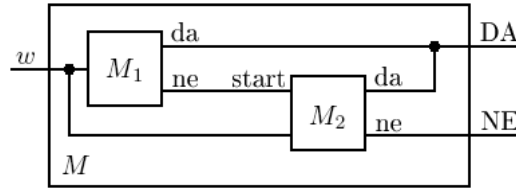
Opomba: V tem primeru smo problem: "Ali je neka beseda w v jeziku \bar{L} ?" prevedli na problem: "Ali je neka beseda w v jeziku L ?"



Izrek 2: Unija rekurzivnih (Turingovih) jezikov je rekurzivni (Turingov) jezik.

Dokaz:

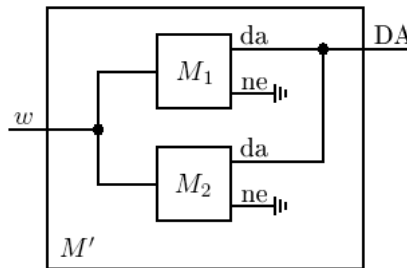
- a) Naj bosta $L_1 = L(M_1)$ in $L_2 = L(M_2)$ rekurzivna (M_1 in M_2 sta algoritma). Sestavimo nov algoritem M :



Očitno:

- M je algoritem, zato je $L(M)$ rekurziven
- $L(M) = L_1 \cup L_2$
- iz tega sledi, da je unija $L_1 \cup L_2$ tudi rekurziven jezik

- b) Naj bosta $L_1 = L(M_1)$ in $L_2 = L(M_2)$ Turingova. Sestavimo nov TS M :

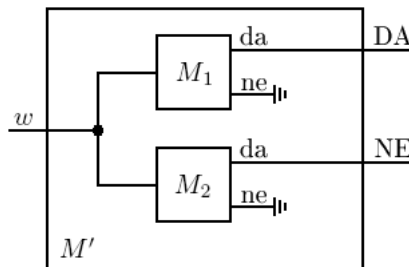


Za odgovor NE se sploh ne menimo (ni se nam treba, saj trdimo, da je jezik $L(M)$ Turingov). Očitno:

- M je Turingov
- $w \in (L_1 \cup L_2) \Leftrightarrow w \in L(M)$

Izrek 3: Če sta L in \bar{L} oba Turingova, potem je jezik L rekurziven (in kot posledica je tudi \bar{L} rekurziven).

Dokaz: Naj bo $L = L(M_1)$ in $\bar{L} = L(M_2)$. Sestavimo nov TS M :



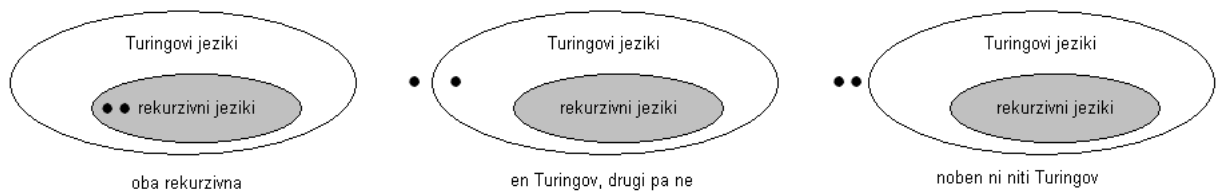
Očitno:

- M je algoritem, torej je jezik $L(M)$ rekurziven
- $L(M) = L$, torej je L rekurziven

D.N. Presek dveh rekurzivnih (Turingovih) jezikov je rekurzivni (Turingov) jezik.

Posledice izrekov 1, 2 in 3:

- L Turingov $\wedge \bar{L}$ Turingov $\stackrel{\text{Izrek 3}}{\Rightarrow} L$ rekurziven $\stackrel{\text{Izrek 1}}{\Rightarrow} \bar{L}$ rekurziven $\Rightarrow L$ rekurziven $\wedge \bar{L}$ rekurziven
 - če obrnemo: L ni rekurziven $\vee \bar{L}$ ni rekurziven $\Rightarrow L$ ni Turingov $\vee \bar{L}$ ni Turingov
 - Torej: če vsaj eden od L in \bar{L} ni rekurziven, potem vsaj eden od njiju ni niti Turingov
- \Rightarrow Naj bo L poljuben jezik. Za L in \bar{L} so možni le tri primeri:
1. primer: oba rekurzivna
 2. primer: en Turingov, drugi pa ne
 3. primer: noben ni niti Turingov



Uporaba: Naj bo dan nek problem P . Poglejmo njegov pripadajoči jezik L_P . Denimo, da nam za komplement \bar{L}_P uspe dokazati, da ni Turingov. Tedaj je \bar{L}_P bodisi iz 2. primera ali iz 3. primera, v obeh primerih pa L_P ni rekurziven. Ker L_P ni rekurziven, je P **neodločljiv problem** (za problem P ni algoritma).

Tako bomo dokazali, da problem $U =$ "Ali TS M sprejme besedo w ?" ni odločljiv.

2.3 Kodiranje Turingovih strojev

Naj bo $L \in \Sigma^*$ Turingov.

1. Vsak simbol iz Σ lahko zakodiramo z abecedo $\{0, 1\}$. Zato lahko rečemo, da je $L \subseteq \{0, 1\}^*$.
2. **Izrek:** Če je $L \subseteq \{0, 1\}^*$ Turingov, ga sprejme nek TS, ki ima tračno abecedo $\Gamma = \{0, 1, B\}$.
3. V takem stroju lahko brez škode predpostavimo, da je q_1 začetno stanje, q_2 pa edino končno stanje.

\Rightarrow Vsak Turingov jezik sprejme nek TS oblike $M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$.

Vse take TS bomo zakodirali. Kako?

Naj bo dan nek TS $M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$:

- treba je zakodirati le Q in δ
- stanja, ki so v Q bomo zakodirali že kar pri kodiranju δ (ne pa posebej)

\Rightarrow kodirati bo treba le δ

- Kako? To naredimo v treh korakih:

1. Uvedemo: $X_1 \equiv 0, X_2 \equiv 1, X_3 \equiv B, S_1 \equiv L, S_2 \equiv D$.

Tedaj prehod $\delta(q_i, X_j) = (q_k, X_l, S_m)$ zakodiramo z $0^i 10^j 10^k 10^l 10^m$.

Primer: $\delta(q_3, 1) = (q_2, 0, D)$ zakodiramo z 00010010010100

Opazimo, da se ne pojavi podzaporedje 11.

2. Kode prehodov ločimo med sabo z 11. Vrstni red prehodov pa je lahko poljuben.

Primer:

$$1: \delta(q_1, 1) = (q_2, 0, D)$$

$$2: \delta(q_3, 0) = (q_1, 1, D)$$

...

zakodiramo z

$$\overbrace{010010010100}^1 11 \overbrace{0001010100100}^2 11 \dots$$

ali

$$\overbrace{0001010100100}^2 11 \overbrace{010010010100}^1 11 \dots$$

Opazimo:

- vrstni red prehodov ni predpisan
- nikoli se ne pojavi podzaporedje 111

3. Celo kodo stroja M začnemo in končamo z 111.

Primer: TS $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$, kjer je:

$$\delta(q_1, 1) = (q_3, 0, D)$$

$$\delta(q_3, 0) = (q_1, 1, D)$$

$$\delta(q_3, 1) = (q_2, 0, D)$$

$$\delta(q_3, B) = (q_3, 1, L)$$

zakodiramo z

$$\underline{11101001000101001100010101001001100010010010100110001000100010010111}$$

Opombe:

- Nek stroj M lahko zakodiramo z več besedami (npr. če spremenimo vrstni red kod prehodov med sabo)
- Množico stanj Q lahko dobimo, če pregledamo vse kode prehodov (in sicer 1. in 3. blok ničel)
- M je determinističen, če se kvečjemu ena koda prehoda začne z nizom oblike $0^i 10^j 1$ pri določenem i in j
- Jezik vseh veljavnih opisov TS je regularen, opisuje ga regularni izraz

$$111111 + 111E(11E)^*111$$

kjer je

$$E = (0^+1(0 + 00 + 000)10^+1(0 + 00 + 000)1(0 + 00)).$$

⇒ preverjanje, ali je neka beseda veljavna koda kakega TS, lahko opravi končni avtomat

2.4 Jezik, ki ni Turingov

Besede $w \in \{0, 1\}^*$ razvrstimo v kanonskem vrstnem redu:

$$\begin{array}{cccccc} w_1, & w_2, & w_3, & w_4, & w_5, & \dots \\ \epsilon, & 0, & 1, & 00, & 01, & \dots \end{array}$$

Naj pomeni M_w TS, katerega koda je beseda w . Neka beseda w_k je ali pa ni koda kakega TS.

Dogovor: če w_k ni koda nobenega TS, se dogovorimo, da je oznaka M_{w_k} oznaka za "pokvarjen" TS, torej M_{w_k} ne sprejema nobene besede: $L(M_{w_k}) = \emptyset$

Sedaj definiramo tabelo:

	M_{w_1}	M_{w_2}	M_{w_3}	\dots	M_{w_j}	\dots
w_1						\vdots
w_2						\vdots
w_3						\vdots
\vdots						\vdots
w_i	\dots	\dots	\dots	\dots	$f(i, j)$	
\vdots						

kjer je

$$f(i, j) = \begin{cases} 0 & \text{če } w_i \notin L(M_{w_j}) \\ 1 & \text{če } w_i \in L(M_{w_j}) \end{cases}$$

Pozor: Na vsakem mestu tabele je 0 ali 1 (tabela je dobro definirana), ni pa nujno, da se da vedno ugotoviti katera je ta vrednost.

Na podlagi diagonalnih vrednosti, definiramo diagonalni jezik L_d :

$$L_d \stackrel{\text{def.}}{=} \{w_k \mid w_k \notin L(M_{w_k})\} \quad *$$

Torej, če je na diagonalni vrednost 0, ustrežna beseda sodi v L_d .

Izrek: L_d ni Turingov.

Dokaz: Izrek dokažemo s protislovjem. Denimo, da je L_d Turingov. Tedaj:

$$\exists w_n : L_d = L(M_{w_n}) \quad *$$

Poglejmo, kako je s pripadnostjo ravno te besede w_n :

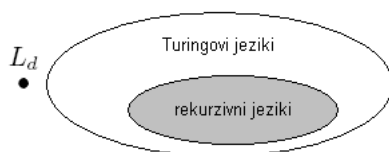
1. če $w_n \in L_d \stackrel{*}{\Rightarrow} w_n \notin L(M_{w_n}) \Rightarrow L(M_{w_n}) \neq L_d$, to pa je protislovje s predpostavko *
2. če $w_n \notin L_d \stackrel{*}{\Rightarrow} w_n \in L(M_{w_n}) \Rightarrow L(M_{w_n}) \neq L_d$, to pa je protislovje s predpostavko *

Tretje možnosti ni.

Sklep: Predpostavko, da je L_d Turingov, smo prisiljeni opustiti. Torej L_d ni Turingov jezik.

2.5 Problem, ki je neodločljiv

L_d je umeten jezik, toda prvi, za katerega vemo, da ni Turingov.



Posledica: Jezik $\overline{L_d} = \{w_k \mid w_k \in L(M_{w_k})\}$ ni rekurziven.

$\overline{L_d}$ je jezik, ki pripada problemu $P = \text{“Ali Turingov stroj } M_w \text{ sprejme lastno kodo?”}$ (lahko bi tudi rekli “Ali program sprejme lastno kodo?”). Ker $\overline{L_d}$ ni rekurziven, problem P ni odločljiv.

Torej: za problem $P = \text{“Ali } M_w \text{ sprejme } w\text{?”}$ ni algoritma (ni TS, ki bi za vsak primerek w problema P odgovoril z DA ali NE).

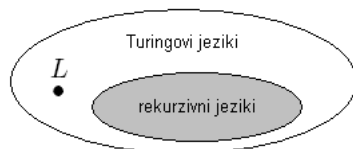
Komentar: Morebitno iskanje algoritma za P je obsojeno na neuspeh. Vsak predlagani “algoritem” bo vrnil napačen odgovor ali pa odgovora sploh ne bo vrnil pri vsaj enem vhodu w .

Vemo: $\overline{L_d}$ ni rekurziven, ampak za $\overline{L_d}$ sta dve možnosti:

1. $\overline{L_d}$ ni niti Turingov
2. $\overline{L_d}$ je Turingov



Ali je možna taka situacija?



2.6 Turingov jezik, ki ni rekurziven

Vprašanje: Kako najti Turingov jezik, ki ni rekurziven?

Odgovor: Izhajamo iz jezika $\overline{L_d}$

TODO: diagram
 $\overline{L_d}$

Definicija: Naj bo par $\langle M, w \rangle$ beseda, ki vsebuje kodo TS M in besedo w . Tedaj:

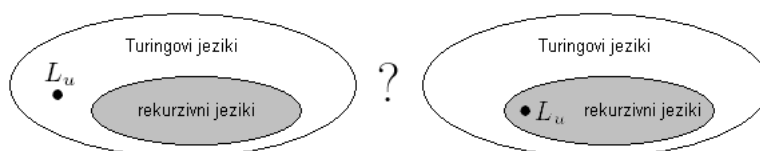
$$L_u \stackrel{\text{def.}}{=} \{ \langle M, w \rangle \mid w \in L(M) \}$$

Izrek: Jezik L_u je Turingov.

Dokaz: Opišimo TS M_u , ki sprejema L_u . Ko M_u dobi na vhod $\langle M, w \rangle$

- preveri ali $\langle M, w \rangle$ vsebuje veljaven opis nekega TS M in besedo w
- če ugotovi, da opis stroja M ni veljaven, M_u zavrne par $\langle M, w \rangle$ saj po dogovoru nobena beseda ne more biti v jeziku "pokvarjenega" stroja
- sicer:
 - M_u začne simulirati stroj M nad w
 - na poseben trak zapiše začetni TO (trenutni opis) q_1w , kjer je q_1 začetno stanje stroja M
 - na podlagi $\langle M \rangle$ generira še naslednje TO
 - če je generirani TO oblike xq_2y , vemo, da smo prišli v končno stanje (q_2 je končno stanje v M), takrat M_u sprejme vhod $\langle M, w \rangle$ in konča
 - sicer nadaljuje generiranje TO, morda v nedogled

Opomba: M_u imenujemo Univerzalni Turingov stroj, ker lahko opravi delo vsakega TS.

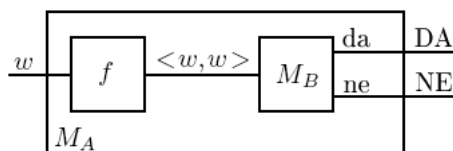


Izrek: L_u ni rekurziven

Dokaz: Denimo, da bi L_u bil rekurziven. Tedaj

$$\exists \text{ TS } M_B (= \text{algoritem}) : L(M_B) = L_u$$

Toda: s pomočjo M_B bi lahko sestavili naslednji algoritem M_A :



- naj bo w poljuben vhod v M_A
- f preslika w v par $\langle w, w \rangle$ (očitno: f je totalno rekurzivna funkcija, izvaja jo nek algoritem M_f)
- hipotetični algoritem M_B , ki dobi $\langle w, w \rangle$ kot vhod, razume (interpretira) $\langle w, w \rangle$ kot:
 - kodo nekega TS M_w
 - in vhod w v ta stroj
- ker je M_B po predpostavki algoritem, odvrne bodisi z:
 - "da" $\Leftrightarrow w \in L(M_w)$
 - "ne" $\Leftrightarrow w \notin L(M_w)$
 To pa pomeni, da je M_A algoritem, ampak za jezik $\overline{L_d}$ rekurzivni jezik.
 To pa je protislovje, saj smo za $\overline{L_d}$ dokazali, da ni rekurziven.

Torej L_u ni rekurziven.

Posledica: $\overline{L_u}$ ni Turingov.

Dokaz: če bi $\overline{L_u}$ bil Turingov, bi bila oba, L_u in $\overline{L_u}$ Turingova, zato bi bila oba tudi rekurzivna. To pa je protislovje, saj L_u ni rekurziven.

Komentar: L_u je jezik, ki pripada problemu $U = \text{“Ali TS } M \text{ sprejme } w\text{?”}$
 Ker L_u ni rekurziven je problem U neodločljiv \Rightarrow za U ni algoritma.

2.7 Prevedbe

Prevedba je metoda, ki smo jo uporabili pri dokazu, da L_u ni rekurziven.

Vemo: L_A ni rekurziven. *

Trdimo: L_B ni rekurziven.

Metoda: dokazati skušamo “ L_B rekurziven $\Rightarrow L_A$ rekurziven”, kar povzroči protislovje s predpostavko *

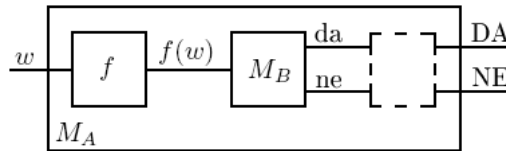
Postopek:

- predpostavimo, da je L_B rekurziven

\Rightarrow obstaja algoritem M_B za prepoznavanje jezika L_B

- sestavimo nek nov TS M_A :

- ki uporablja domnevni algoritem M_B , tako da mu podtakne svoj (ustrezno preoblikovani) vhod
- izhode M_B uporabi kot svoj odgovor (izhode lahko tudi preoblikuje, npr. negira)



- pokažemo, da je M_A algoritem za $L_A \Rightarrow L_A$ rekurziven \Rightarrow protislovje s predpostavko * (predpostavka ne velja)

Sprejemanje jezika L_A smo **prevedli** na sprejemanje jezika L_B oz. reševanje problema A smo prevedli na reševanje problema B , tako da velja: B odločljiv $\Rightarrow A$ odločljiv.

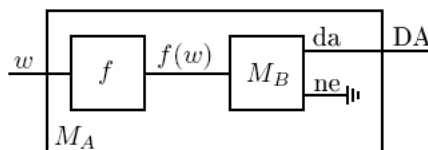
Prevedbo lahko uporabimo za dokaz, da je nek jezik Turingov.

Vemo: nek L_B je Turingov ($= \exists$ TS M_B za L_B)

Trdimo: L_A je Turingov

Metoda: dokažemo, da \exists TS M_B za $L_B \Rightarrow \exists$ TS M_A za L_A

Postopek: sestavimo M_A tako:



Za dokaz, da je L_A Turingov, izhod za NE ni pomemben.

Za funkcijo f pa zahtevamo:

- da je totalno rekurzivna (da vedno preoblikuje vhod w v $f(w)$)
- da ima lastnost: $f(w) \in L_B \Leftrightarrow w \in L_A$

Prevedbo lahko uporabimo tudi za dokaz, da nek jezik **ni** Turingov.

Vemo: L_A ni Turingov.

Trdimo: L_B ni Turingov

Metoda: dokazati želimo, da \exists TS M_B za $L_B \Rightarrow \exists$ TS M_A za L_A

Postopek:

- predpostavimo obstoj TS M_B
- s pomočjo TS M_B sestavimo TS M_A
- s tem pridemo v protislovje s predpostavko, da L_A ni Turingov $\Rightarrow M_B$ ni Turingov

Definicija: $L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$

L_{ne} (not empty) vsebuje kode TS, ki sprejemajo vsaj eno besedo. Lepo bi bilo, če bi bil L_{ne} rekurziven.

Vprašanje: Kakšen je L_{ne} ?

Odgovor: L_{ne} je Turingov.

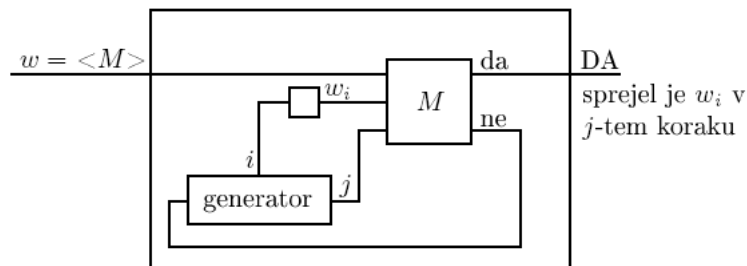
Izrek: L_{ne} je Turingov.

Dokaz: Za L_{ne} sestavimo TS M_{ne} :

- na vhod dobi besedo w
- to besedo interpretira kot kodo $\langle M \rangle = w$ nekega TS
- in "nadzorovano" simulira stroj M s pomočjo generatorja parov
- generira naslednji par (i, j)
- simulira stroj M nad besedo w_i j korakov (w_i je i -ta beseda v kanonskem vrstnem redu)
- če M sprejme besedo w_i na j -tem koraku, M_{ne} sprejme $\langle M \rangle = w$ in konča
- sicer generira naslednji par (i, j)

Torej:

če $\left\{ \begin{array}{l} \langle M \rangle \in L_{ne} \Leftrightarrow \exists(i_0, j_0) : M \text{ sprejme } w_{i_0} \text{ v } j_0\text{-tem koraku} \Rightarrow M_{ne} \text{ sprejme } \langle M \rangle \\ \langle M \rangle \notin L_{ne} \Leftrightarrow M_{ne} \text{ se ne ustavi} \end{array} \right\} L_{ne} \text{ je Turingov}$



Na vse skupaj lahko gledamo kot na prevedbo L_{ne} na L_u .

S to prevedbo \exists TS za $L_u \Rightarrow \exists$ TS L_{ne} .

Vemo, da TS za L_u obstaja.

Izrek: L_{ne} ni rekurziven.

Dokaz: Prevedba jezika L_u na L_{ne} . Če bi bil L_{ne} rekurziven, bi bil rekurziven tudi L_u , toda vemo, da L_u ni rekurziven.

Problem: "Ali TS M sprejme kako besedo?" ni odločljiv (zanj ni algoritma, postopka, ki bi za vsak primer pravilno odgovoril z DA ali NE).

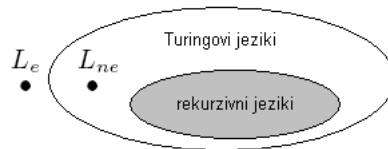
Definicija: $L_e = \{ \langle M \rangle \mid L(M) = \emptyset \}$

Jezik L_e ustreza problemu: "Ali M ne sprejema nobene besede?"

Opomba: v L_e so tudi besede, ki niso kode kakega TS.

Zato: $L_e = \overline{L_{ne}}$

Izrek: L_e ni Turingov.



Definicija: $L_r = \{ \langle M \rangle \mid L(M) \text{ je rekurziven} \}$

Opomba:

Naj bo $\langle M \rangle \in L_r$

Torej: besede, za katere se stroj M ustavi, tvorijo rekurzivni jezik.

Toda: če stroju M damo na vhod $w \notin L(M)$, se lahko zgodi (če npr. M ni dobro konstruiran za sprejemanje $L(M)$), da se M ne ustavi.

Zato ne smemo reči, da je L_r enak jeziku $\{ \langle M \rangle \mid M \text{ se ustavi na vseh vhodih} \}$.

Velja le: $L_r \supsetneq \{ \langle M \rangle \mid M \text{ se ustavi na vseh vhodih} \}$

Toda: če $\langle M \rangle \in L_r$, gotovo obstaja nek (morda drug) TS, ki se ustavi na vseh vhodih in tudi sprejema ravno $L(M)$

Torej: $L_r = \{ \langle M \rangle \mid \exists M' : M' \text{ se ustavi na vseh vhodih} \wedge L(M') = L(M) \}$

Definicija: $L_{nr} = \{ \langle M \rangle \mid L(M) \text{ ni rekurziven} \}$

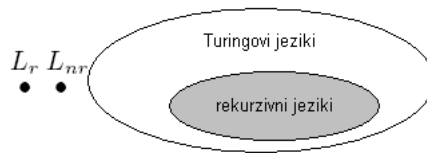
Opomba: $L_{nr} = \overline{L_r}$

Izrek: L_r ni Turingov.

Dokaz: Prevedba $\overline{L_u}$ na L_r .

Izrek: L_{nr} ni Turingov.

Dokaz: Prevedba $\overline{L_u}$ na L_{nr} .



Motivacija: Doslej smo obravnavali:

- $L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$
- $L_e = \{ \langle M \rangle \mid L(M) = \emptyset \}$
- $L_r = \{ \langle M \rangle \mid L(M) \text{ je rekurziven jezik} \}$
- $L_{nr} = \{ \langle M \rangle \mid L(M) \text{ ni rekurziven jezik} \}$

Za vse se je izkazalo, da niso rekurzivni. S prevedbami lahko dokažemo, da niso rekurzivni niti naslednji jeziki:

- $L_{fin} = \{ \langle M \rangle \mid L(M) \text{ je končen} \}$
- $L_{inf} = \{ \langle M \rangle \mid L(M) \text{ ni končen} \}$
- $L_{reg} = \{ \langle M \rangle \mid L(M) \text{ je regularen jezik} \}$
- $L_{cf} = \{ \langle M \rangle \mid L(M) \text{ je kontekstno neodvisen jezik} \}$
- $L_{even} = \{ \langle M \rangle \mid L(M) \text{ ima sodo mnogo besed} \}$
- \vdots

Vsi ti jeziki imajo podobno definicijo:

$$L_S = \{ \langle M \rangle \mid L(M) \text{ ima lastnost } S \}$$

Vprašanje: Za katere lastnosti S je L_S nerekurziven, za katere pa rekurziven?

Posplošimo: vzamimo nekaj Turingovih jezikov in jih strnimo v družino S .

Definicija: Pravimo:

- da je S **jezikovna lastnost** (ki jo imajo jeziki, ki so v S)
- da ima nek jezik L jezikovno lastnost S , če $L \in S$, sicer je nima
- da je jezikovna lastnost S **trivialna**, če je nima noben Turingov jezik ($S = \emptyset$) ali pa jo imajo vsi Turingovi jeziki ($S = \{L \mid L \text{ je Turingov jezik} \}$)

Opomba: trivialne lastnosti niso zanimive. Zanimive so netrivialne lastnosti. Pri teh je zanimivo vprašanje: "Ali ima L lastnost S ?"

Definicija: $L_S = \{ \langle M \rangle \mid L(M) \text{ ima lastnost } S \}$

Opomba: ali $\langle M \rangle$ je ali ni v L_S je odvisno od $L(M)$, ne pa od strukturnih lastnosti M direktno.

Definicija: Pravimo:

- da je jezikovna lastnost S **rekurzivna**, če je L_S rekurziven jezik
- da je jezikovna lastnost S **Turingova**, če je L_S Turingov jezik

Vprašanje: Katere jezikovne lastnosti so rekurzivne?

Vprašanje je pomembno, ker bi nam odgovor povedal za katere lastnosti S je problem "Ali ima L lastnost S ?" odločljiv (za katere jezikovne lastnosti S obstaja algoritem, ki odgovori na vprašanje "Ali ima L lastnost S ?").

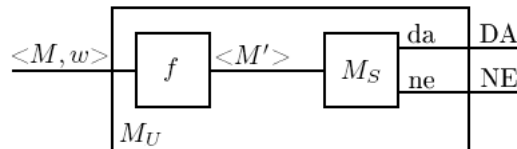
2.7.1 Riceov izrek

Riceov izrek: Jezikovna lastnost je rekurzivna natanko tedaj, ko je trivialna.

Opomba: slaba novica - čim je jezikovna lastnost S taka, da jo nekateri jeziki imajo, drugi pa ne, za ugotavljanje, kateri jo imajo in kateri ne, ni algoritma.

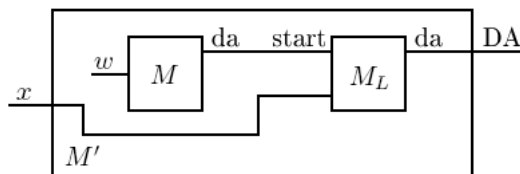
Dokaz:

- \emptyset ima ali pa nima lastnosti S . Vzamimo, da \emptyset nima lastnosti S , torej $\emptyset \notin S$ (v nasprotnem primeru bi namesto S obravnavali \bar{S} in bi bil pogoj izpolnjen)
- Naj bo S netrivialna. Torej $\exists L : L \in S$ in \exists TS $M_L : M_L$ sprejema L
- Predpostavka: S je rekurzivna
- Tedaj $\exists M_S$ algoritem, ki sprejema jezik L_S
- s pomočjo domnevnega algoritma M_S konstruiramo TS M_U , ki:
 - na vhod dobi poljuben par $\langle M, w \rangle$



- vhod transformira v kodo $\langle M' \rangle$
- ki jo ponudi domnevnemu M_S
- M_S ugotovi ali je $\langle M' \rangle \in L_S$
- M_U je celo algoritem, če je funkcija f totalno rekurzivna

Kakšen naj bo M' ?



Kakšen je $L(M')$?

Očitno:

$$L(M') = \begin{cases} L \in S & , \text{ če } w \in L(M) \\ \emptyset \notin S & , \text{ če } w \notin L(M) \end{cases}$$

Kakšen je $L(M_U)$?

- če je izhod iz M_U DA $\Rightarrow L(M')$ ima lastnost $S \Rightarrow L(M') = L \Rightarrow w \in L(M) \Rightarrow \langle M, w \rangle \in L_U$
- če je izhod iz M_U NE $\Rightarrow L(M')$ nima lastnosti $S \Rightarrow L(M') = \emptyset \Rightarrow w \notin L(M) \Rightarrow \langle M, w \rangle \notin L_U$

Torej: $L(M_U) = L_u$

Toda: M_U je algoritem $\Rightarrow L_u$ rekurziven, kar pa je protislovje.

Intuitivno: O programih (Turingovih strojih) lahko marsikaj povemo, toda o tem, kaj porogram počne (o jeziku, ki ga TS sprejema) lahko vedno algoritemično določimo, le trivialne lastnosti.

Vprašanje: Ali obstaja kak bolj naraven neodločljiv problem? **Odgovor:** Veliko jih je.

2.8 Postov korespondenčni problem (PKP)

Definicija: $PKP(x, y)$

Ali za dana seznama $X = \langle x_1, x_2, \dots, x_n \rangle$ in $Y = \langle y_1, y_2, \dots, y_n \rangle$ besed nad neko abecedo obstaja zaporedje i_1, i_2, \dots, i_k indeksov (kjer je $k \geq 1$ in $i_j \in \{1, 2, \dots, n\}$), da je $x_{i_1} x_{i_2} \dots x_{i_n} = y_{i_1} y_{i_2} \dots y_{i_n}$

Intuitivno: Ali lahko sestavimo neko besedo na dva podobna načina iz dveh skupin gradnikov?

Primer:

	1.	2.	3.	4.	5.
x	011	0	101	1010	010
y	1101	00	01	00	0

Ta $PKP(x, y)$ ima rešitev, npr.:

	2.	1.	1.	4.	1.	5.
x :	0011	1011	1101	0011	010	
y :	0011	1011	1101	0011	010	
	2.	1.	1.	4.	1.	5.

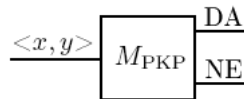
Primer:

	1.	2.	3.
x	10	011	101
y	101	11	011

Ta $PKP(x, y)$ nima rešitve.

\Rightarrow od seznamov X in Y je odvisno ali $PKP(x, y)$ ima/nima rešitev.

Vprašanje: Ali \exists algoritem M_{PKP} , ki odloči (reče DA ali NE) ali ima $PKP(x, y)$ rešitev?



Ali je problem $PKP(x, y)$ odločljiv?

Izrek: : PKP je neodločljiv.

2.9 Dvournost kontekstno neodvisne gramatike

Definicija: $AMB \equiv$ "Ali je dana KNG dvournna?"

Trditev: Problem AMB je neodločljiv.

Dokaz: (prevedba PKP na AMB)

Naj bo dan par seznamov x, y (del problema PKP). Preslikamo jih v KNG $f(x, y)$ z lastnostjo:

$$f(x, y) \text{ dvoumna} \Leftrightarrow \text{PKP}(x, y) \text{ ima rešitev}$$

Kako? Definirajmo jezika:

$$L_x = \{x_{i_1}x_{i_2} \dots x_{i_m}a_{i_m} \dots a_{i_2}a_{i_1} \mid m \geq 1, i_j \in \{1, 2, \dots, n\}\}$$

in

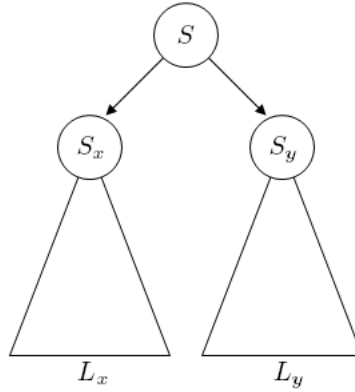
$$L_y = \{y_{i_1}y_{i_2} \dots y_{i_m}a_{i_m} \dots a_{i_2}a_{i_1} \mid m \geq 1, i_j \in \{1, 2, \dots, n\}\}$$

Sestavimo KNG za jezik $L_x \cup L_y$:

$$\text{KNG } G = (\{S, S_x, S_y\}, \Sigma \cup \{a_1, a_2, \dots, a_n\}, P, S)$$

kjer P vsebuje:

$$\begin{aligned} S &\rightarrow S_x \mid S_y \\ S_x &\rightarrow x_i S_x a_i \mid x_i a_i \quad \text{za vse } i = 1, 2, \dots, n \\ S_y &\rightarrow y_i S_y a_i \mid y_i a_i \quad \text{za vse } i = 1, 2, \dots, n \end{aligned}$$



$$\Rightarrow L(G) = L_x \cup L_y$$

a) Trditev: $\text{PKP}(x, y)$ ima rešitev $\Rightarrow G$ dvoumna.

Dokaz: Če ima $\text{PKP}(x, y)$ rešitev i_1, i_2, \dots, i_m je

$$x_{i_1}x_{i_2} \dots x_{i_m} = y_{i_1}y_{i_2} \dots y_{i_m},$$

zato je

$$x_{i_1}x_{i_2} \dots x_{i_m}a_{i_m} \dots a_{i_2}a_{i_1} = y_{i_1}y_{i_2} \dots y_{i_m}a_{i_m} \dots a_{i_2}a_{i_1}.$$

To pa je beseda v $L(G)$, ki ima dve izpeljavi (preko S_x oz. S_y) $\Rightarrow G$ je dvoumna.

b) Trditev: $\text{PKP}(x, y)$ ima rešitev $\Leftrightarrow G$ dvoumna.

Dokaz: Naj bo G dvoumna $\Rightarrow \exists$ beseda z dvema izpeljavama, preko S_x oz. S_y .

Če je ta beseda $xa_{i_m} \dots a_{i_2}a_{i_1}$ (kjer $x \in \Sigma^*$), potem je $i_1 i_2 \dots i_m$ ravno rešitev problema $\text{PKP}(x, y)$. Torej je $\text{PKP}(x, y)$ rešljiv.

AMB odločljiv \Leftrightarrow PKP odločljiv.

PKP pa ni odločljiv.

Torej: AMB je **neodločljiv**.

Izrek: Naj bodo G, G_1 in G_2 poljubne KNG, R pa poljubna regularna množica. Naslednji problemi so neodločljivi:

- $L(G_1) \cap L(G_2) \stackrel{?}{=} \emptyset \dots$ "Ali dve KNG generirata disjunktna jezika?"
- $L(G_1) \stackrel{?}{=} L(G_2) \dots$ "Ali dve KNG generirata isti jezik?"
- $L(G_1) \stackrel{?}{\subseteq} L(G_2) \dots$ "Ali KNG generira podjezik jezika, ki ga generira druga KNG?"
- $L(G) \stackrel{?}{=} R \dots$ "Ali neka KNG generira celo regularno množico?"
- $R \stackrel{?}{\subseteq} L(G) \dots$ "Ali neka KNG generira jezik, ki vsebuje regularno množico R?"
- $L(G) \stackrel{?}{=} \Sigma^* \dots$ "Ali KNG generira vse možne besede?"

Izrek: Naj bosta G_1 in G_2 poljubni gramatiki. Naslednja problema sta neodločljiva:

- "Ali je $\overline{L(G_1)}$ KNJ?"
- "Ali je $L(G_1) \cap L(G_2)$ KNJ?"

2.10 Diofantske enačbe

Definicija: $D(k) \equiv$ "Ali ima polinom s celimi koeficienti in $k (\geq 1)$ spremenljivkami celoštevilsko rešitev?"

Primer: $3x^5 + 4y + 15z - 89v^4 - w = P(x, y, z, v, w) = 0$

Vprašanje: Ali $\exists (x_0, y_0, z_0, v_0, w_0) \in \mathbb{Z}^5 : P(x, y, z, v, w) = 0$

Odgovor: Leta 1970 je bilo pokazano, da je $D(k)$ neodločljiv (že pri $k = 13$).

Posebni podproblemi pa so lahko odločljivi.

Primer:

- $D'(k) \equiv$ "Ali ima polinom $ax^k = c$ ($a, c \in \mathbb{Z}$) celo rešitev?"
- $D''(k) \equiv$ "Ali ima polinom $\sum_{i=1}^k a_i x_i = c$ ($a_i, c \in \mathbb{Z}$) celo rešitev?"

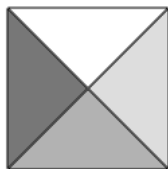
sta problema, ki **sta odločljiva**, imata polinomska algoritma.

- $D'''(2) \equiv$ "Ali ima polinom $ax^2 + by = c$ ($a, b, c \in \mathbb{Z}$) celo rešitev?"

je odločljiv problem, ampak je NP-poln (gotovo rešljiv v eksponentnem času, vprašanje pa če je rešljiv tudi hitreje).

2.11 Problem tlakovanja

Ploščica, npr.



- razdeljena na štiri dele, vsak del je obarvan z neko barvo (ali pa oštevilčen)
- se ne sme zavrteti, niti simetrično preslikati (ima predpisano orientacijo)

Definicija: Dana je končna množica $T = \{t_1, t_2, \dots, t_n\}$ razpoložljivih tipov ploščic. Vsakega tipa je na zalogi potencialno neomejeno ploščic.

Problem 1: “Ali s T lahko tlakujemo vsako končno veliko ploskev (v \mathbb{Z}^2)?” (dve sosednji ploščici se morata ujemati v barvi/številki vseh štirih stičnih delov)

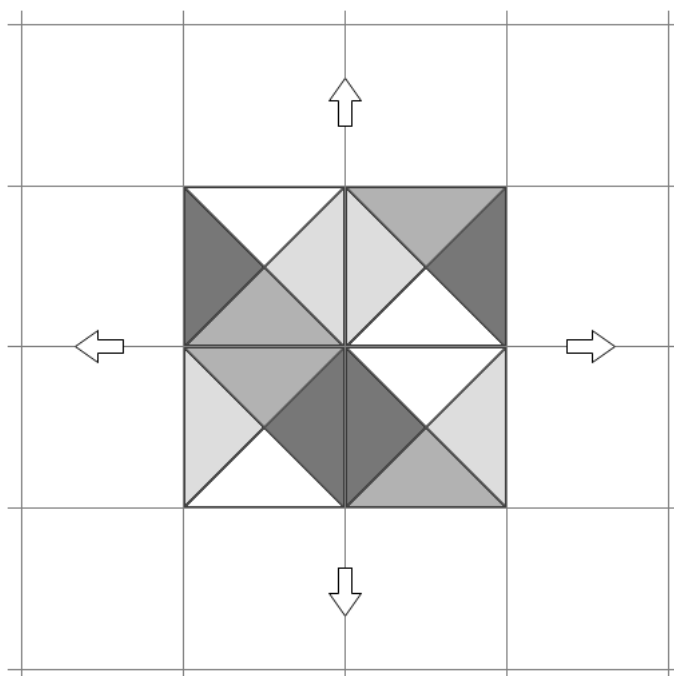
Pozor: Vprašanje ni, ali lahko tlakujemo neko konkretno končno ploskev, temveč če lahko tlakujemo vsako končno ploskev.

Pozor: Množica T je v tem vprašanju parameter problema (= vhodni podatek v morebitni algoritmu za problem 1).

Izrek: Problem 1 je neodločljiv.

Opomba: Vsak “algoritem”, ki si ga zamislimo za reševanje problema 1, bo pri nekem T (v resnici pri nešteto mnogo T -jih) bodisi računal brez prestanka ali pa vrnil napačen odgovor.

Problem 2: “Ali lahko s T tlakujemo celo \mathbb{Z}^2 ?”



Izrek: Problem 1 in problem 2 sta ekvivalentna (oba sta neodločljiva).

Dokaz: (skica dokaza)

(\Leftarrow) če znamo tlakovati \mathbb{Z}^2 , znamo tlakovati tudi končno ploskev v \mathbb{Z}^2

(\Rightarrow) bolj zapleteno

Posledica: Problem 2 je neodločljiv.

Posledica: Obstaja množica T , s katero lahko tlakujemo celo \mathbb{Z}^2 , toda tlakovanje je neperiodično (tj. ne vsebuje končno velikega vzorca, ki bi se neskončno mnogokrat ponavljal v vseh smereh).

Dokaz: (skica dokaza)

če to ne bi bilo res, ki bilo vsako tlakovanje cele \mathbb{Z}^2 pač preiodično.

Toda: tedaj bi lahko našli algoritem za problem 2 (\Rightarrow problem 2 bi postal odločljiv \Rightarrow protislovje).

Torej: je res.

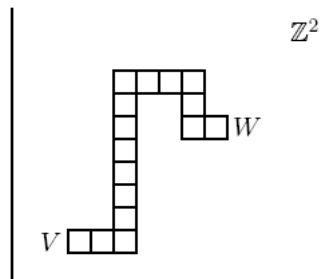
Podrobneje o algoritmu:

- ```
/* sistematično generira končno velike ploskve $P \in \mathbb{Z}^2$ */
1 generiraj naslednjo ploskev P
/* sistematično išče vsa tlakovanja za P */
2 skušaj generirati naslednje tlakovanje ploskve P s T
3 če naslednjega tlakovanja ni
4 če P doslej še ni bila tlakovana
 /* \mathbb{Z}^2 se ne da tlakovati s T , zato je odgovor NE */
5 odgovor NE in končaj
6 sicer če je P bila tlakovana in P se ne da nadaljevati
7 nadaljuj pri 1
8 sicer če lahko repliciraš tlakovanje P v vseh smereh
 /* našel periodično tlakovanje \mathbb{Z}^2 , zato je odgovor DA */
9 odgovor DA in končaj
```

## 2.12 Problem domin

**Definicija:** Dana je množica  $T = \{t_1, t_2, \dots, t_n\}$  in točki  $V, W \in \mathbb{Z}^2$ .

**Vprašanje:** "Ali med  $V$  in  $W$  lahko tlakujemo končno pot?"



Opombe:

- včasih se je treba močno oddaljiti od  $V$  preden se obrnemo proti  $W$
- oddaljitvev je lahko poljubno velika

**Odgovor:** Odločljivost/neodločljivost problema je odvisna od omejitev pri speljevanju poti:

- če gre lahko pot kamorkoli, potem je problem odločljiv
- če gre lahko pot le po polovici  $\mathbb{Z}^2$ , potem je problem neodločljiv
- če gre lahko pot kamorkoli, razen čez točko  $U$  ( $U \neq V, W$ ), potem je problem neodločljiv

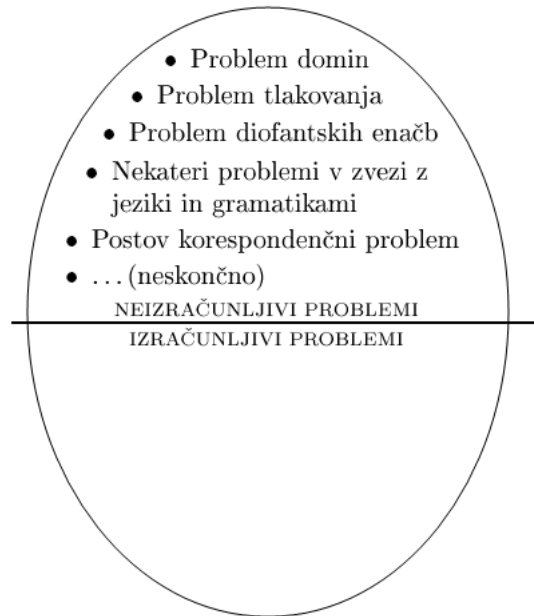
## 3 Zahtevnost (kompleksnost) izračunov

Uvod: doslej smo obravnavali

- kaj se da in česa se ne da izračunati
- na kakšnem modelu
- ne glede na to, koliko je potrebno časa/prostora

Ugotovili smo, da obstajajo neizračunljivi problemi (neodločljivi). Za nobenega od njih ne obstaja algoritem (ne glede na čas/prostor, ki nam je na voljo).

⇒ Prva delitev:



med seboj niso čisto enakovredni

Relativizacija: nekateri so “bolj neizračunljivi” kot drugi. Če predpostavimo, da bi za katerega obstajal prerok, bi kateri postali izračunljivi, drugi pa bi bili še vedno neizračunljivi.

Stopnje (degrees) neodločljivosti ⇒ hierarhija neodločljivosti

Drugi del so izračunljivi problemi, za katere obstaja algoritem.

Toda: intuicija/praksa nam pokaže, da je možno rešiti

- večje primerke teh problemov
- oz. bolj zapletene probleme

le, če je na voljo več računskega vira (npr. časa, prostora, procesorjev, energije, če je lahko računska napaka večja, ...).

**Vprašanje:** Kako ta intuitivni občutek (da imajo računski viri pri računanju pomembno vlogo) formalizirati?

**Odgovor:** uporabljata se dva pristopa

1. povečevanje konkretnih računskih virov

- prostor, čas
- procesorji, čas (vzporedni algoritmi)
- odstopanje od točne rešitve, čas (aproksimativni algoritmi)
- verjetnost točne rešitve (verjetnostni algoritmi)

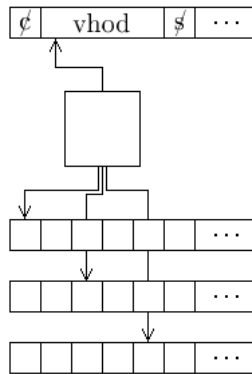
2. aksiomatski pristop (začetnik: Manuel Blum)

- postuliramo lastnosti (aksiome), ki jih ima vsak smiselen računski vir
- izpeljujemo posledice

### 3.1 Prostorska zahtevnost

Dan je TS  $M$  z:

- 1 vhodnim trakom
  - na njem je vhodna beseda med mejnima znakoma  $\phi$  in  $\$$
  - vhodno besedo lahko le beremo
- $k$  ( $\geq 1$ ) delovnimi trakovi



**Definicija:**  $M$  je TS s **prostorsko omejitvijo** (zahtevnostjo)  $S(n)$ , če za vsak vhod dolžine  $n$  stroj obiše  $\leq S(n)$  celic (na vsakem delovnem traku).

Pravimo, da ima tedaj jezik  $L(M)$  **prostorsko zahtevnost**  $S(n)$ .

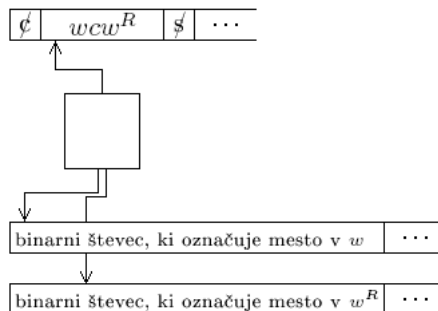
Intuitivno: Za odgovarjanje na vprašanje " $w \in L(M)$ " zadošča prostor  $S(|w|)$ .

Opombe:

- šteje le poraba celic na **delovnih trakovih**.  
Razlog: če bi upoštevali tudi prostor na vhodnem traku, bi bila  $S(n) \geq n$ .  
Toda: mi bi radi preučevali tudi npr.  $S(n) = \lceil \log n \rceil$
- vsak TS porabi vsaj 1 celico na vsakem delovnem traku, zato  $S(n) \geq 1$ .  
Zato: ko rečemo, da ima TS prostorsko omejitvev  $S(n)$ , mislimo na  $\max\{1, \lceil S(n) \rceil\}$   
(sicer bi pri počasi rastočih  $S$  dobili pri nekaterih  $n$ :  $S(n) = 0$ )
- namesto  $k$  delovnih trakov bi lahko bil 1 delovni trak s  $k$  sledmi

**Primer:**  $L = \{w c w^R \mid w \in (0 + 1)^*\}$  ima prostorsko zahtevnost  $\lceil \log n \rceil$ ,  $n = |w c w^R|$ , ker  $\exists$  TS  $M$  s prostorsko omejitvijo  $\lceil \log n \rceil$ , da je  $L = L(M)$ .

Kakšen je  $M$ ?



$M$  skače levo-desno in primerja zrcalna znaka (v  $w$  in  $w^R$ ).

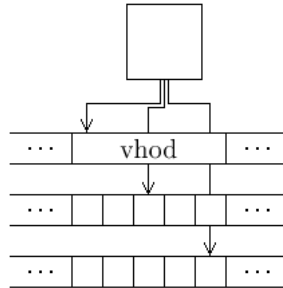
$M$  ima logaritemsko prostorsko omejitev, ker vsak od števcov rabi  $\lceil \log |w| \rceil = \lceil \log \frac{n-1}{2} \rceil \leq \lceil \log n \rceil$  celic.

### 3.2 Časovna zahtevnost

Dan je TS  $M$ :

- z  $k (\geq 1)$  delovnimi trakovi (na enem bo vhodna beseda, tudi na ta trak lahko piše)
- ki so na obe strani neomejeni

Slika:



**Definicija:**  $M$  je TS s **časovno omejitvijo** (zahtevnostjo)  $T(n)$ , če za vsak vhod dolžine  $n$  naredi  $\leq T(n)$  korakov (preden se ustavi).

Pravimo, da ima tedaj tudi jezik  $L(M)$  **časovno zahtevnost**  $T(n)$ .

Intuitivno: za odgovarjanje na vprašanje “ $w \in L(M)$ ” vedno zadošča  $T(|w|)$  korakov.

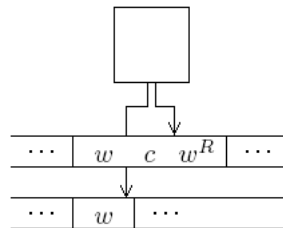
Opomba: predpostavljamo, da mora stroj prebrati **cel vhod**.

Zato:  $T(n) \geq n + 1$ ,  $n = |w|$ ,  $n$  za branje vhoda,  $+1$  za ugotovitev, da je vhoda konec.

Zato: odgovor “časovna omejitev  $T(n)$ ” pomeni  $\max\{n + 1, \lceil T(n) \rceil\}$ .

**Primer:**  $L = \{w c w^R \mid w \in (0 + 1)^*\}$  ima časovno zahtevnost  $n + 1$ , ker  $\exists$  TS  $M$  s časovno omejitvijo  $n + 1$ , da je  $L = L(M)$ .

Kakšen je  $M$ ?



Stroj  $M$  prepíše  $w$  na 2. trak, ko pa prečka znak  $c$ :

- nadaljuje po  $w^R$  po 1. traku
- se hkrati vrača po  $w$  po 2. traku
- in sproti primerja tekoča znaka

Za vse skupaj porabi  $n + 1$  korakov.

### 3.3 Nedeterministična prostorska in časovna zahtevnost

**Definicija:** Nedeterministični TS, ima prostorsko omejitev (zahtevnost)  $S(n)$ , če za vsak vhod dolžine  $n$  obstaja izračun, ki obišče  $\leq S(n)$  celic (na vsakem delovnem traku).

**Definicija:** Nedeterministični TS ima časovno omejitev (zahtevnost)  $T(n)$ , če za vsak vhod dolžine  $n$  obstaja izračun, ki naredi  $\leq T(n)$  korakov.

**Definicija:** Jezik  $L$  ima:

- deterministično prostorsko zahtevnost  $S(n)$ , če zanj obstaja deterministični TS s prostorsko omejitvijo  $S(n)$
- nedeterministično prostorsko zahtevnost  $S(n)$ , če zanj obstaja nedeterministični TS s prostorsko omejitvijo  $S(n)$

**Definicija:** Jezik  $L$  ima:

- deterministično časovno zahtevnost  $T(n)$ , če zanj obstaja deterministični TS s časovno omejitvijo  $T(n)$
- nedeterministično časovno zahtevnost  $T(n)$ , če zanj obstaja nedeterministični TS s časovno omejitvijo  $T(n)$

### 3.4 Razredi zahtevnosti (kompleksnostni razredi)

**Definicija:**

- $\text{DSPACE}(S(n)) \stackrel{\text{def.}}{=} \{L \mid L \text{ ima deterministično prostorsko zahtevnost } S(n)\}$
- $\text{NSPACE}(S(n)) \stackrel{\text{def.}}{=} \{L \mid L \text{ ima nedeterministično prostorsko zahtevnost } S(n)\}$
- $\text{DTIME}(T(n)) \stackrel{\text{def.}}{=} \{L \mid L \text{ ima deterministično časovno zahtevnost } T(n)\}$
- $\text{NTIME}(T(n)) \stackrel{\text{def.}}{=} \{L \mid L \text{ ima nedeterministično časovno zahtevnost } T(n)\}$

**Primer:**  $L = \{wcw^R \mid w \in (0+1)^*\}$

Videli smo, da:

- $L \in \text{DSPACE}(\log n)$
- $L \in \text{DTIME}(n)$

### 3.5 Linearno stiskanje in druge transformacije

#### 3.5.1 Uvod

- za velikost tračne abecede ni omejitve (le da je končna)

$\Rightarrow$  vsebino nekaj zaporednih celic nadomestimo z enim samim simbolom iz neke druge (večje) tračne abecede

**Primer:**

|     |   |   |   |   |     |   |   |   |     |                                  |
|-----|---|---|---|---|-----|---|---|---|-----|----------------------------------|
| ... | 0 | 0 | 0 | 1 | 1   | 1 | 1 | 0 | ... | $M_1, \Gamma_1 = \{0, 1\}$       |
| ... | 0 | 1 | 3 | 2 | ... |   |   |   |     | $M_2, \Gamma_2 = \{0, 1, 2, 3\}$ |

$\Rightarrow$  Tako lahko vedno zmanjšamo število porabljenih celic za nek konstantni faktor

$\Rightarrow$  pri prostorski zahtevnosti nas sme zanimati le **velikostni razred** funkcije  $S(n)$  (lahko uporabljamo  $\mathcal{O}$ -notacijo)

Opomba: tu ne razmišljamo, kako velika bi morala biti nova celica

### 3.5.2 Linearno stiskanje

**Izrek:** Če za jezik  $L$  obstaja  $k$ -tračni TS s prostorsko omejitvijo  $S(n)$ , potem za  $L$  obstaja  $k$ -tračni TS s prostorsko omejitvijo  $c \cdot S(n)$  za **poljuben**  $c > 0$ .

Opomba: Tu je pomembno, da je lahko  $c < 1$ .

**Dokaz:**

- Naj bo  $M_1$  TS s prostorsko omejitvijo  $S(n)$  in  $L = L(M_1)$
- Naj bo dan **poljuben**  $c \in (0, 1)$
- Sestavimo TS  $M_2$ , ki oponaša  $M_1$ , tako da:
  - v eni svoji celici hrani vsebino  $r$  zaporednih celic stroja  $M_1$  ( $r$  določimo kasneje, odvisen je od  $c$ , več kot bi radi stisnili, večji je  $r$ )
  - nadzorna enota  $M_2$  pa si beleži katero od  $r$  celic bi gledal stroj  $M_1$
- Pri simulaciji stroja  $M_1$  stroj  $M_2$  obišče  $\lceil \frac{S(n)}{r} \rceil$  svojih celic (na vsakem traku)

Naj bo  $r$  tak, da je  $\frac{1}{r} \leq \frac{c}{2}$  ( $\Rightarrow \frac{2}{c} \leq r$ )

Če:

- $S(n) \geq r$ , je  $\lceil \frac{S(n)}{r} \rceil \leq \lceil \frac{c}{2} \cdot S(n) \rceil \leq S(n)$
- $S(n) < r$ , ima  $M_2$  vse shranjeno v eni svoji celici, zato porabi le to celico

$\Rightarrow M_2$  razpoznavlja  $L$  in ima prostorsko omejitev  $c \cdot S(n)$

Opomba: Naj bo  $d > 1$ .

Gotovo velja, da  $\text{DSPACE}(S(n)) \subseteq \text{DSPACE}(d \cdot S(n))$ .

Kaj pa obratno? Če je  $L \in \text{DSPACE}(d \cdot S(n))$ , izrek pove, da je tudi  $L \in \text{DSPACE}(S(n))$  (vzeti je potrebno  $c = \frac{1}{d}$ ).

**Posledica:**  $\text{DSPACE}(S(n)) = \text{DSPACE}(c \cdot S(n))$ , za  $\forall c > 0$

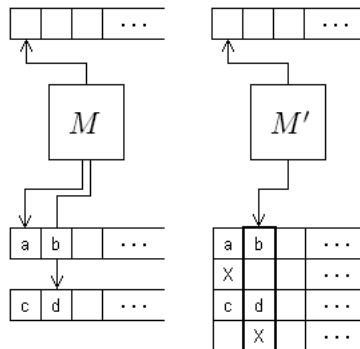
Opomba: S spreminjanjem prostorske omejitve za **konstantni faktor** pri “razpoznavni moči” ne pridobimo nič novega.

**Posledica:** Dovolj je, da nas zanima le **velikostni razred** prostorske omejitve/zahtevnosti (lahko uporabljamo  $\mathcal{O}$ -notacijo).

### 3.5.3 Odpravljanje trakov

**Vprašanje:** “Ali in s kakšnim vplivom na prostorsko zahtevnost lahko  $k > 1$  delovnih trakov nadomestimo z enim samim?”

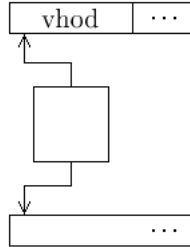
Zamisel:  $k > 1$  trakov nadomestimo z enim trakom, ki ima  $2k$  sledi.



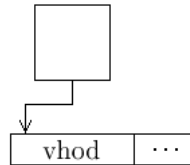
**Izrek:** če je  $L$  jezik  $k$ -tračnega TS s prostorsko omejitvijo  $S(n)$ , je  $L$  tudi jezik 1-tračnega TS z isto prostorsko omejitvijo.

Opombe:

- ko bomo govorili o prostorski omejitvi, bomo (največkrat) predpostavljali kar 1-tračni TS:



- če nas bo zanimala  $S(n) \geq n$  bo zadoščal celo tak TS:



### 3.6 Linearna pospešitev in druge transformacije

#### 3.6.1 Uvod

- **Vprašanje:** Ali za časovno zahtevnost velja kaj podobnega kot za prostorsko (stiskanje)?
- Tudi ta množico stanj pravzaprav ni posebnih omejitev (biti mora končna, sicer poljubne velikosti)

⇒ Zamisel: nekaj zaporednih stanj TS nadomestimo z enim stanjem iz neke druge večje množice stanj.

⇒ vedno lahko zamenjamo število korakov za konstantni faktor

⇒ Tudi pri časovnih omejitvah (zahtevnostih) nas zanima le njihov velikostni razred.  
(Npr. pogosto rečemo  $T(n) = \mathcal{O}(n^2)$ , manjkrat pa rečemo  $T(n) = 3n^2$ ).

Priprava:

**Definicija:** Naj bo  $f(n)$  funkcija in  $n \in \mathbb{N}$ .

$$\sup f(n) \stackrel{def.}{=} \lim_{n \rightarrow \infty} \text{lub}\{f(n), f(n+1), f(n+2), \dots\} \dots \text{supremuum}$$

in

$$\inf f(n) \stackrel{def.}{=} \lim_{n \rightarrow \infty} \text{glb}\{f(n), f(n+1), f(n+2), \dots\} \dots \text{infimum}$$

**Primer:** Naj bo  $f(n) = \begin{cases} \frac{1}{n} & , \text{ če } n \text{ sod} \\ n & , \text{ če } n \text{ lih} \end{cases}$

|        |   |               |   |               |   |               |     |                |        |     |
|--------|---|---------------|---|---------------|---|---------------|-----|----------------|--------|-----|
| $n$    | 1 | 2             | 3 | 4             | 5 | 6             | ... | $2k$           | $2k+1$ | ... |
| $f(n)$ | 1 | $\frac{1}{2}$ | 3 | $\frac{1}{4}$ | 5 | $\frac{1}{6}$ | ... | $\frac{1}{2k}$ | $2k+1$ | ... |

$\sup_{n \rightarrow \infty} f(n) = \infty$ , zaradi lihih  $n$   
 $\inf_{n \rightarrow \infty} f(n) = 0$ , zaradi sodih  $n$

**Primer:**  $f(n) = \frac{n}{n+1}$

|        |               |               |               |               |               |               |     |
|--------|---------------|---------------|---------------|---------------|---------------|---------------|-----|
| $n$    | 1             | 2             | 3             | 4             | 5             | 6             | ... |
| $f(n)$ | $\frac{1}{2}$ | $\frac{2}{3}$ | $\frac{3}{4}$ | $\frac{4}{5}$ | $\frac{5}{6}$ | $\frac{6}{7}$ | ... |

Komentar:

$\sup_{n \rightarrow \infty} f(n) = 1$   
 $\inf_{n \rightarrow \infty} f(n) = 1$

### 3.6.2 Linearna pospešitev

**Izrek:** Če za jezik  $L$  obstaja  $k$ -tračni TS s časovno omejitvijo  $T(n)$ , potem za  $L$  obstaja  $k$ -tračni TS s časovno omejitvijo  $c \cdot T(n)$  za poljuben  $c > 0$  **pod pogojem**, da je  $k > 1$  in  $\inf_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$ .

Torej: Tudi pri  $c < 1$

Pomembno: do pospešitve lahko pride, ker je lahko  $c < 1$ .

Opombe:

- Izrek je podoben izreku o stiskanju prostora, toda uvaja dodaten pogoj
- Zahteva  $\inf_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$  pomeni, da mora časovna omejitev  $T(n)$  naraščati "vsaj malo" hitreje od  $n$  (linearne)  
 Drugače povedano: poleg za branje mora biti "vsaj malo" korakov na razpolago za računanje

**Primer:** če bi imeli  $T(n) = \begin{cases} \log n & , \text{ če } n \text{ sod} \\ n & , \text{ če } n \text{ lih} \end{cases}$

Tedaj:  $\inf_{n \rightarrow \infty} \frac{T(n)}{n} = 0 \Rightarrow$  za vhode sode dolžine bi bil čas računanja premajhen  $\Rightarrow$  pospešitve niso smiselne.

S pogojem  $\inf_{n \rightarrow \infty} \frac{T(n)}{n}$  izločimo nekatere (nesmiselne) časovne omejitve.

**Posledica:**  $\text{DTIME}(T(n)) = \text{DTIME}(c \cdot T(n))$ , za poljuben  $c > 0$ , če je  $\inf_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$ .

Če  $T(n)$  zadosti hitro narašča je pomemben le velikostni red funkcije  $T(n)$  (lahko uporabljamo  $\mathcal{O}$ -notacijo).

**Vprašanje:** Kaj pa, če pogoj  $\inf_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$  ne velja?

**Primer:** če je  $T(n) = d \cdot n$  ( $d$  konstanta) je  $\inf_{n \rightarrow \infty} \frac{d \cdot n}{n} = d < \infty$ .

Če je  $d > 1$ , je  $d \cdot n > n$ , torej je čas za branje in razpoznavanje daljši od časa za branje.

**Vprašanje:** Ali v tem primeru izrek res ne velja?

**Odgovor:** Izkaže se, da velja.

**Izrek:** Če za jezik  $L$  obstaja  $k$ -tračni TS s časovno omejitvijo  $d \cdot n$ , kjer  $d > 1$  in  $k > 1$ , potem za jezik  $L$  za vsak  $\epsilon > 0$  obstaja  $k$ -tračni TS s časovno omejitvijo  $(1 + \epsilon) \cdot n$ .

Opomba: pomembno je "za vsak  $\epsilon > 0$ ", saj je  $\epsilon$  lahko poljubno majhen, le da je pozitiven.

**Posledica:**  $\text{DTIME}(d \cdot n) = \text{DTIME}((1 + \epsilon) \cdot n)$  za poljubne  $\epsilon > 0$  in konstanten  $d > 1$ .

Opomba: če pa funkcija  $T(n)$  raste počasneje, npr. z  $\log n$  ali  $\sqrt{n}$ , potem pa izrek več ne velja.

Opomba: Velja tudi za NTIME.

**Primer:**

- a)  $\text{NTIME}(T(n)) = \text{NTIME}(c \cdot T(n))$  za poljuben  $c > 0$ , če je  $\inf_{n \rightarrow \infty} \frac{T(n)}{n} = \infty$
- b)  $\text{NTIME}(d \cdot n) = \text{NTIME}((1 + \epsilon) \cdot n)$  za poljuben  $\epsilon > 0$ , in konstanten  $d > 1$

### 3.6.3 Odpravljajanje trakov in časovna zahtevnost

**Vprašanje:** Ali in za kakšno ceno (za časovno zahtevnost) lahko  $k > 1$  trakov nadomestimo z enim samim?

**Primer:**  $L = \{w c w^R \mid w \in (0 + 1)^*\}$

Če ima TS 2 delovna trakova:

- sprejme jezik  $L$  v linearnem času  $n = |w c w^R|$
- **Dokaz:**
  - TS prepisuje  $w$  s prvega traku na drugi trak
  - ko pride do znaka  $c$  pa primerja  $w^R$  in  $w$  (ki je na drugem traku)

Če pa ima TS le en delovni trak:

- lahko simulira delo dvotračnega TS
  - za kar ima en 4-sledni trak
  - v nadzorni enoti ima še prostor za znake nad  $X$ -i in števec  $X$ -ov, ki so še desno od glave
  - pri simulaciji potuje levo-desno in primerja “simetrična” znaka v  $w$  in  $w^R$
- $\Rightarrow$  pri tem naredi  $\Theta(n^2)$  korakov

Nauk primera: zmanjševanje časovnih trakov na enega **lahko** časovno zahtevnost **poveča s kvadratom**.

Toda: To je največja (najhujša) upočasnitev, ki se lahko zgodi pri prehodu na enotračni TS.

**Izrek:** Če je jezik  $L \in \text{DTIME}(T(n))$  na  $k$ -tračnem TS, potem je  $L \in \text{DTIME}(T^2(n))$  na enotračnem TS.

**Dokaz:** (skica dokaza)

- glej dokaz o simulaciji večtračnega TS z enotračnim
- tam smo ugotovili, da je za simulacijo  $T(n)$  korakov večtračnega TS, potrebnih kvečjemu  $\leq d \cdot T^2(n)$  korakov
- izrek o linearni pospešitvi (v našem primeru za faktor  $\frac{1}{d}$ ) zagotavlja, da je simulacija možna na enotračnem TS v  $T^2(n)$  korakih

Dokaze lahko prilagodimo tudi za nedeterministične TS.

Zato:

**Izrek:** Če je  $L \in \text{NTIME}(T(n))$  na  $k > 1$  tračnem TS, potem je  $L \in \text{NTIME}(T^2(n))$  na enotračnem TS.

**Vprašanje:** Kaj pa, če omejimo TS le na dva trakova?

**Odgovor:** Tudi tedaj se časovna zahtevnost **poveča**, toda **kvečjemu** za faktor  $\log T(n)$ .

**Izrek:** Če je jezik  $L \in \text{DTIME}(T(n))$  na  $k > 2$  tračnem TS, potem je  $L \in \text{DTIME}(T(n) \cdot \log T(n))$  na dvotračnem TS.

Velja tudi za nedeterministične TS:

**Izrek:** Če je jezik  $L \in \text{NTIME}(T(n))$  na  $k > 2$  tračnem nedeterminističnem TS, potem je  $L \in \text{NTIME}(T(n) \cdot \log T(n))$  na dvotračnem nedeterminističnem TS.

### 3.7 Hierarhije

Motivacija:

- Intuicija: le je na voljo več časa ali prostora, pričakujemo, da bomo razpoznali več jezikov (rešili več problemov)
- Toda: izrek o linearnem stiskanju in izrek o linearni pospešitvi pravita, da za povečanje razreda sprejetih jezikov (oz. rešenih problemov) **ne zadošča** povečanje časa ali prostora za **konstantni faktor**.

**1. Vprašanje:** Kaj pa če prostorsko/časovno omejitev pomnožimo z neko zelo počasi naraščujočo funkcijo ( $\log(\log n)$ ,  $\log^*(n)$ )? Ali se lahko zgodi, da celo tedaj ne moremo razpoznati nič več kot prej?

**2. Vprašanje:** Ali obstaja neka "super" prostorska/časovna zahtevnost  $f^*(n)$ , da je vsak rekurziven (odločljiv problem) v razredu  $\text{DSPACE}(f^*(n))$  oz.  $\text{DTIME}(f^*(n))$ ?

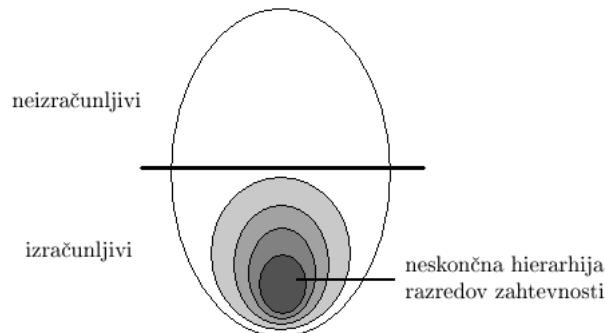
**Odgovor na 1. vprašanje:** Ne. Za vsako prostorsko/časovno omejitev obstaja jezik (problem), ki za sprejetje (rešitev) zahteva več kot toliko prostora/časa.

**Posledica:** Hierarhija razredov deterministične prostorske zahtevnosti je neskončna.

**Posledica:** Hierarhija razredov deterministične časovne zahtevnosti je neskončna.

**Posledica:** Hierarhija razredov nedeterministične prostorske zahtevnosti je neskončna.

**Posledica:** Hierarhija razredov nedeterministične časovne zahtevnosti je neskončna.



**Izrek:** Naj bo  $T(n)$  poljubna totalno rekurzivna časovna omejitev. Potem obstaja takšen rekurzivni jezik  $L$ , da  $L \in \text{DTIME}(T(n))$ .

**Izrek:** Naj bo  $S(n)$  poljubna totalno rekurzivna prostorska omejitev. Potem obstaja takšen rekurzivni jezik  $L$ , da  $L \in \text{DSPACE}(S(n))$ .

**Dokaz:** (samo za časovno omejitev)

- po predpostavki je  $T(n)$  totalno rekurzivna  $\Rightarrow \exists$  algoritem  $M$ , ki izračuna  $T(n)$  za  $\forall n$

- večtračne TS (s poljubnimi abecedami) lahko zakodiramo z abecedo  $\{0,1\}$  (podobno kot enotračne pri neodločljivosti)  $\Rightarrow$  lahko govorimo o  $i$ -tem večtračnem TS s poljubno tračno abecedo  $M_i$
- Naj bo  $x_i$   $i$ -ta beseda v kanonski ureditvi besed iz  $(0+1)^*$
- **Definicija:**  $L \stackrel{def.}{=} \{x_i \mid M_i \text{ ne sprejme besede } x_i \text{ v } \leq T(|x_i|) \text{ korakih}\}$
- **Trditev:**  $L$  je rekurziven.

**Dokaz:** Naj bo  $w$  vhodna beseda.

Za jezik  $L$  sestavimo sledeč algoritem  $\overline{M}$ , ki:

- uporabi algoritem  $M$  in z njim izračuna vrednost  $T(n)$ ,  $n = |w|$
  - poišče  $i$ , za katerega je  $x_i = w$
  - konstruira stroj  $M_i$  (sestavi njegovo kodo)
  - simulira stroj  $M_i$  nad  $w (= x_i)$  za  $\leq T(n)$  korakov
  - sprejme vhod  $w \Leftrightarrow \begin{cases} M_i \text{ se je v } i \text{ korakih ustavil in zavrnil vhod } w \text{ ali} \\ M_i \text{ po } T(n) \text{ korakih bi še delal } (\Rightarrow \text{ni sprejel } w \text{ niti v } T(n) \text{ korakih}) \end{cases}$
- **Trditev:**  $L \in \text{DTIME}(T(n))$

**Dokaz:** Predpostavka:  $L = L(M_j)$  za nek  $j$  in  $M_j$  ima časovno omejitev  $T(n)$ .

Kje je  $x_j$ ?

- $x_j \in L(M_j) \Rightarrow M_j$  sprejme  $x_j$  v  $T(|x_j|)$  korakih  $\stackrel{def.}{\Rightarrow} L \stackrel{predpos.}{\Rightarrow} x_j \notin L(M_j)$
- $x_j \notin L(M_j) \Rightarrow M_j$  ne sprejme  $x_j$  v  $T(|x_j|)$  korakih  $\stackrel{def.}{\Rightarrow} L \stackrel{predpos.}{\Rightarrow} x_j \in L(M_j)$

Odpovedati se je potrebno predpostavki ( $L$  nima časovne zahtevnosti  $T(n)$ )

Opomba:

- Noben razred  $\text{DTIME}(T(n))$  ni "super razred", ki bi zajemal vse rekurzivne jezike (odločljive probleme)
- Naj bo  $T(n)$  poljubna totalna rekurzivna časovna omejitev. Izrek nam pravi, da obstaja rekurzivni jezik  $L$ :  $L \notin \text{DTIME}(T(n))$ . Vemo pa:  $L$  je rekurziven  $\Rightarrow$  obstaja algoritem  $\overline{M}$ :  $L = L(\overline{M})$  (glej dokaz).

Algoritem  $\overline{M}$  je nek TS z neko časovno omejitvijo  $\overline{T(n)}$  (tudi  $\overline{T(n)}$  je totalno rekurzivna funkcija).

Naj bo  $T'(n) = \max\{T(n), \overline{T(n)}\}$  (tudi  $T'(n)$  je totalno rekurzivna funkcija).

Gotovo:  $\text{DTIME}(T(n)) \subseteq \text{DTIME}(T'(n))$

Vemo pa:  $L \notin \text{DTIME}(T(n)) \wedge L \in \text{DTIME}(T'(n)) \Rightarrow \text{DTIME}(T(n)) \subsetneq \text{DTIME}(T'(n))$

**Posledica:** Obstaja neskončna hierarhija razredov  $\text{DTIME}$ :

- $\text{DTIME}(T(n)) \subsetneq \text{DTIME}(T'(n)) \subsetneq \text{DTIME}(T''(n)) \subsetneq \dots$

Podobno:

- $\text{DSPACE}(S(n)) \subsetneq \text{DSPACE}(S'(n)) \subsetneq \text{DSPACE}(S''(n)) \subsetneq \dots$
- $\text{NTIME}(T(n)) \subsetneq \text{NTIME}(T'(n)) \subsetneq \text{NTIME}(T''(n)) \subsetneq \dots$
- $\text{NSPACE}(S(n)) \subsetneq \text{NSPACE}(S'(n)) \subsetneq \text{NSPACE}(S''(n)) \subsetneq \dots$

### 3.7.1 Hierarhije razredov DSPACE

**Vprašanje:** Kako gosta, tesna, je hierarhija razredov DSPACE?

Kako veliko naj bo povečanje prostorske omejitve  $S(n)$ , da dobimo nov, večji razred? (povečanje prostora s konstantnim faktorjem ne prinese ničesar - izrek o linearnem stiskanju).

**Odgovor:** Tu so pomembne "lepe" funkcije  $S(n)$ . Če bo  $S(n)$  "lepa", je dovolj, da  $S'(n)$  narašča le "neznatno" hitreje od  $S(n)$ , pa bo že veljalo  $\text{DSPACE}(S(n)) \subsetneq \text{DSPACE}(S'(n))$ .

**Vprašanje:** Kaj pomeni "lepa" funkcija in kaj "neznatno" hitreje?

**Odgovor:**

"lepa" pomeni prostorsko verna (prostorsko smiselna)

"neznatno" pa pomeni  $\inf_{n \rightarrow \infty} \frac{S(n)}{S'(n)} = 0$

**Definicija:** Funkcija  $S(n)$  je prostorsko verna (space constructible), če  $\exists$  TS  $M$  s prostorsko omejitvijo  $S(n)$ , tak, da za  $\forall n \in \mathbb{N} \exists$  vhod dolžine  $n$ , na katerem  $M$  porabi natanko  $S(n)$  celic.

Opomba:

- zadošča, da  $M$  le na enem vhodu dolžine  $n$  porabi natanko  $S(n)$  celic
- včasih pa celo na vsakem vhodu dolžine  $n$  (kar je v resnici pogosto možno doseči)

Množica prostorsko vernih funkcij je zelo bogata:  $(\log n, n^k, 2^n, n!, \dots)$

Tudi produkt in potenca dveh prostorsko vernih funkcij je prostorsko verna funkcija.

Tudi eksponent prostorsko verne funkcije je prostorsko verna funkcija.

**Definicija:** Funkcije  $S(n)$  je popolnoma prostorsko verna (fully space constructible), če  $\exists$  TS  $M$  s prostorsko omejitvijo  $S(n)$ , tako, da za  $\forall$  vhod dolžine  $n$  porabi natanko  $S(n)$  celic.

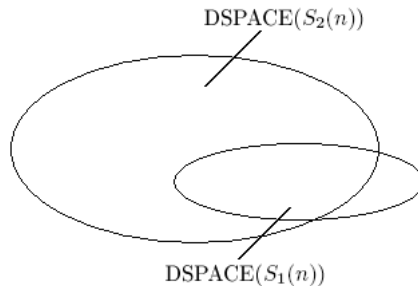
**Trditev:** Vsaka prostorsko verna funkcija  $f(n) \geq n$  za  $\forall n$ , je popolnoma prostorsko verna.

**Izrek:** Če je  $S_1(n) \geq \log n$ ,  $S_2(n) \geq \log n$ ,  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$  in je  $S_2(n)$  popolnoma prostorsko verna, potem  $\exists L: L \in \text{DSPACE}(S_2(n)) \wedge L \notin \text{DSPACE}(S_1(n))$ .

Opomba: Pogoje lahko celo omilimo:

Če je  $S_1(n) \geq \log n$ ,  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$  in  $S_2(n)$  prostorsko verna, potem  $\exists L: L \in \text{DSPACE}(S_2(n)) \wedge L \notin \text{DSPACE}(S_1(n))$ .

Če za  $S_1$  in  $S_2$  iz zgornjega izreka velja še  $\forall n: S_1(n) \leq S_2(n)$ , potem  $\text{DSPACE}(S_1(n)) \subsetneq \text{DSPACE}(S_2(n)) \Rightarrow$  obstaja neskončna hierarhija DSPACE razredov, ki je "precej gosta, tesna" (zaradi zahteve  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)}$ ).



To je lahko posledica situacije, ko je  $S_1(n) > S_2(n)$  za neke  $n$ .

### 3.7.2 Hierarhije razredov DTIME

**Vprašanje:** Kako gosta je hierarhija razredov DTIME?

**Odgovor:** Je nekoliko bolj "redka".

**Definicija:** Funkcija  $T(n)$  je **časovno verna** (time constructible), če  $\exists$  TS  $M$  s časovno omejitvijo  $T(n)$ , da za  $\forall n \in \mathbb{N} \exists$  vhod dolžine  $n$ ,  $M$  naredi natanko  $T(n)$  korakov.

**Definicija:** Funkcija  $T(n)$  je **popolnoma časovno verna** (fully time constructible), če  $\exists$  TS  $M$  s časovno omejitvijo  $T(n)$ , da za  $\forall n \in \mathbb{N}$  za  $\forall$  vhod dolžine  $n$ ,  $M$  naredi natanko  $T(n)$  korakov.

**Izrek:** Če je  $T_2(n)$  popolnoma časovno verna in  $\inf_{n \rightarrow \infty} \frac{T_1(n) \cdot \log T_1(n)}{T_2(n)} = 0$ , potem  $\exists L: L \in \text{DTIME}(T_2(n)) \wedge L \notin \text{DTIME}(T_1(n))$ .

**Primer:**

$$\left. \begin{array}{l} T_1(n) = 2^n \\ T_2(n) = n^2 \cdot 2^n \end{array} \right\} \inf_{n \rightarrow \infty} \frac{T_1(n) \cdot \log T_1(n)}{T_2(n)} = \inf_{n \rightarrow \infty} \frac{2^n \cdot \log 2^n}{n^2 \cdot 2^n} = \inf_{n \rightarrow \infty} \frac{1}{n} = 0$$

Ker:  $T_1(n) \leq T_2(n)$  za  $\forall n$  je tudi  $\text{DTIME}(2^n) \subsetneq \text{DTIME}(n^2 \cdot 2^n)$ .

**Primer:**

$$\left. \begin{array}{l} T_1(n) = 2^n \\ T_2(n) = n \cdot 2^n \end{array} \right\} \inf_{n \rightarrow \infty} \frac{T_1(n) \cdot \log T_1(n)}{T_2(n)} = \inf_{n \rightarrow \infty} \frac{2^n \cdot \log 2^n}{n \cdot 2^n} = 1 > 0$$

Pogoj za izrek ni izpolnjen. Kljub temu velja  $\text{DTIME}(2^n) \subsetneq \text{DTIME}(n \cdot 2^n)$ .

**Vprašanje:** Kako to dokažemo?

**Odgovor:** Dokažemo s pomočjo leme o pomiku.

Torej pogoj  $\inf_{n \rightarrow \infty} \frac{T_1(n) \cdot \log T_1(n)}{T_2(n)} = 0$  je le zadosten pogoj ni pa potreben pogoj.

**Izrek:** Če je  $T_2(n)$  časovno verna in  $\inf_{n \rightarrow \infty} \frac{T_1(n) \cdot \log^* T_1(n)}{T_2(n)} = 0$ , potem za  $\forall k \in \mathbb{N} \exists L:$

- $\exists k$ -tračni TS  $M$  s časovno omejitvijo  $T_2(n)$ :  $L(M) = L$
- $\neg \exists k$ -tračni TS  $M'$  s časovno omejitvijo  $T_1(n)$ :  $L(M') = L$

Opombe:

- $\log^* n \stackrel{\text{def.}}{=} \text{najmanjše število } m \in \mathbb{N}: \underbrace{\log \log \dots \log n}_{m\text{-krat}} \leq 1$

- $\log^* n$  zelo počasi narašča, npr.  $\log^* 2^{65536} = 5$

$\Rightarrow T_2$  sme komajda hitreje naraščati od  $T_1$ , pa že lahko razpoznamo nov jezik.

**Primer:** Naj bo  $S_1(n) = n^2$  in  $S_2(n) = \begin{cases} n^3 & \text{če je } n \text{ sod} \\ n & \text{če je } n \text{ lih} \end{cases}$ .

Uporabimo izrek o prostorski hierarhiji (z omiljenimi pogoji):

Če je  $S_1(n) \geq \log n$ ,  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$  in  $S_2(n)$  je prostorsko verna, potem  $\exists L: L \in \text{DSpace}(S_2(n)) \wedge L \notin \text{DSpace}(S_1(n))$ .

Velja:

- $S_1(n) \geq \log n$
- $S_2(n)$  je prostorsko verna

Vprašamo se:  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$ ?

Računamo:  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = \inf_{n \rightarrow \infty} \begin{cases} \frac{1}{n} & , \text{ če je } n \text{ sod} \\ n & , \text{ če je } n \text{ lih} \end{cases} = \text{/glej primer nazaj/} = 0$   
 $\Rightarrow \exists L': L \in \text{DSPACE}(S_1(n)) \wedge L' \notin \text{DSPACE}(S_2(n))$

Prostorski omejitvi  $S_1(n)$  in  $S_2(n)$  pogledjmo v obratnih vlogah:

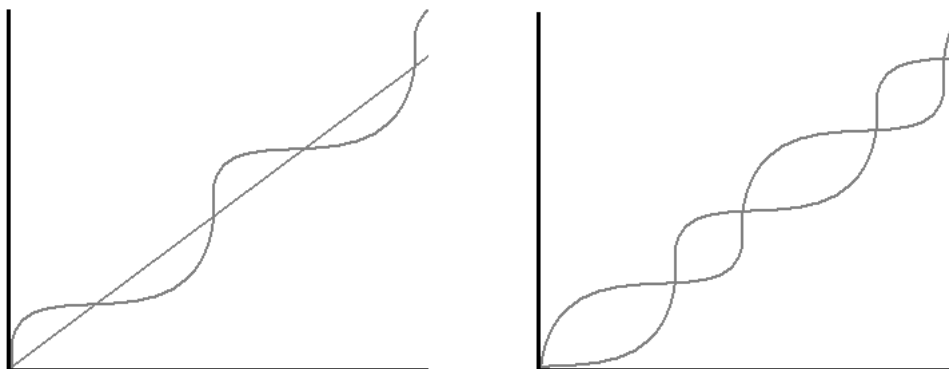
- $S_2(n) \geq \log n$
- $S_1(n)$  je prostorsko verna
- $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$ ?

Računamo:  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = \inf_{n \rightarrow \infty} \begin{cases} n & , \text{ če je } n \text{ sod} \\ \frac{1}{n} & , \text{ če je } n \text{ lih} \end{cases} = 0$   
 $\Rightarrow \exists L': L' \in \text{DSPACE}(S_1(n)) \wedge L' \notin \text{DSPACE}(S_2(n))$

To izvira iz tega, da je **neskončno mnogokrat**  $S_1(n) < S_2(n)$  in tudi  $S_1(n) > S_2(n)$

(zato sta  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)}$  in  $\inf_{n \rightarrow \infty} \frac{S_2(n)}{S_1(n)}$  oba lahko = 0)

To se zgodi, če se funkciji  $S_1$  in  $S_2$  "prepletata" neskončno dolgo.



Že zaradi definicije razreda zahtevnosti bo veljalo:

$\text{DSPACE}(S_1(n)) \subseteq \text{DSPACE}(S_2(n))$  (če je  $\forall n: S_1(n) \leq S_2(n)$ )

Če pa so izpolnjeni še pogoji  $\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0$ , pa bo veljalo:

$\text{DSPACE}(S_1(n)) \subsetneq \text{DSPACE}(S_2(n))$

### 3.7.3 Relacije med časovno in prostorsko zahtevnostjo

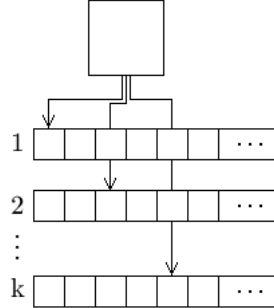
- Videli smo, da obstaja neskončno razredov prostorske in časovne zahtevnosti
- Videli smo kako "tesne" so te hierarhije
- **Vprašanje:** Kakšen je odnos med razredi različnih zahtevnosti (prostor in čas)?

**Izrek:**

1.  $\text{DTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$



- $k$  trakov,  $s$  stanj,  $t$  tračnih simbolov in prostorsko omejitev  $f(n)$
- ⇒ stroj  $M_1$  ima  $\leq s \cdot (f(n) + 1)^k \cdot t^{k \cdot f(n)}$  različnih trenutnih opisov  
 $t^{k \cdot f(n)}$  - vse možne vsebine trakov  
 $(f(n) + 1)^k$  - vse možne lege oken  
 $s$  - vsa možna stanja



- Tedaj obstaja konstanta  $d$ :  $d^{f(n)} \geq s \cdot (f(n) + 1)^k \cdot t^{k \cdot f(n)}$  za  $\forall n \geq 1$ . Za  $d$  lahko vzamemo kar  $d = s \cdot (t + 1)^{3k}$
- Sestavimo TS  $M_2$ :
  - ki sestavi seznam TO, ki bi jih lahko dosegel  $M_1$  v  $f(n)$  korakih
  - teh TO je  $\leq d^{f(n)}$
 ⇒  $M_2$  jih lahko zapiše v času, ki je  $\leq d^{f(n)} \cdot (1 + k(f(n) + 1))$   
 $(1 + k(f(n) + 1))$  je dolžina enega TO, in sicer:  
 1 za stanje  
 $k$  za vsak trak  
 $(f(n) + 1)$  je največja dolžina traka  
 ⇒  $M_2$  za vse to rabi čas  $\leq c^{f(n)}$ , kjer je  $c$  neka konstanta

d) (dokaz smo izpustili)

### 3.7.4 Hierarhije razredov NTIME in NSPACE

Motivacija:

- hierarhiji DTIME in DSPACE sta “gosti”
- kako je s NTIME in NSPACE

Koristna je lema o pomiku (translation lema).

**Lema:** Naj bodo  $S_1(n)$ ,  $S_2(n)$  in  $f(n)$  popolnoma prostorsko verne in  $S_2(n) > n$  ter  $f(n) \geq n$ .  
 Tedaj: Če je  $\text{NSPACE}(S_1(n)) \subseteq \text{NSPACE}(S_2(n))$ , potem je  $\text{NSPACE}(S_1(f(n))) \subseteq \text{NSPACE}(S_2(f(n)))$ .

Opomba:

- Namesto  $S_2(n) \geq n$  lahko zahtevamo  $S_2(n) \geq \log n \wedge S_2(n)$  popolnoma prostorsko verna.
- Lema o pomiku (v enaki obliki) velja tudi za NTIME, DSPACE in DTIME.

Uporaba: S pomočjo leme o pomiku (za DTIME) lahko dokažemo, da velja

$$\text{DTIME}(2^n) \subsetneq \text{DTIME}(n \cdot 2^n).$$

Ker je  $\inf_{n \rightarrow \infty} \frac{2^n \log 2^n}{n \cdot 2^n} = 1 > 0$ , izreka o hierarhiji razredov DTIME ne moremo uporabiti za dokaz. Lotiti se moramo drugače, z uporabo leme o pomiku (translaciji).

**Trditev:**  $\text{DTIME}(2^n) \subseteq \text{DTIME}(n \cdot 2^n)$

**Dokaz:** (z lemo o pomiku)

Predpostavimo, da  $\text{DTIME}(n \cdot 2^n) \subseteq \text{DTIME}(2^n)$ . Uporabimo lemo o pomiku:

- $S_1(n) = n \cdot 2^n$   
 $S_2(n) = 2^n$   
 $f(n) = n + 2^n$   
 dobimo, da velja:  $\text{DTIME}((n + 2^n) \cdot 2^n \cdot 2^{2^n}) \subseteq \text{DTIME}(2^n \cdot 2^{2^n})$
- $S_1(n) = n \cdot 2^n$   
 $S_2(n) = 2^n$   
 $f(n) = 2^n$   
 dobimo, da velja:  $\text{DTIME}(2^n \cdot 2^n) \subseteq \text{DTIME}(2^{2^n})$
- zaradi tranzitivnosti dobimo:  $\text{DTIME}((n + 2^n) \cdot 2^n \cdot 2^{2^n}) \subseteq \text{DTIME}(2^{2^n})$

Toda:  $\inf_{n \rightarrow \infty} \frac{2^{2^n} \log 2^{2^n}}{(n + 2^n) \cdot 2^n \cdot 2^{2^n}} = \inf_{n \rightarrow \infty} \frac{1}{(n + 2^n)} = 0 \Rightarrow$   
 $\Rightarrow \exists L: L \in \text{DTIME}((n + 2^n) \cdot 2^n \cdot 2^{2^n}) \wedge L \notin \text{DTIME}(2^{2^n})$ ,  
 kar pomeni, da predpostavka ne more veljati.

Torej velja:

$$\neg(\underbrace{\text{DTIME}(n \cdot 2^n)}_A \subseteq \underbrace{\text{DTIME}(2^n)}_B)$$

$$\neg(A \subseteq B) \Leftrightarrow \neg(A \subset B \vee A = B) \Leftrightarrow \neg(A \subset B) \wedge A \neq B$$

torej  $\text{DTIME}(n \cdot 2^n) \neq \text{DTIME}(2^n)$ .

Vendar po definiciji časovne zahtevnosti je  $\text{DTIME}(2^n) \subseteq \text{DTIME}(n \cdot 2^n)$ .

Iz tega sledi:

$$\text{DTIME}(2^n) \subsetneq \text{DTIME}(n \cdot 2^n)$$

Lemo o pomiku uporabimo tudi pri dokazovanju naslednjih izrekov o hierarhiji NSPACE.

**Izrek:** Če je  $r \geq 0$  in  $\epsilon > 0$ , potem  $\text{NSPACE}(n^r) \subsetneq \text{NSPACE}(n^{r+\epsilon})$ .

**Izrek:** Če je  $c > 1$  in  $\epsilon > 0$ , potem  $\text{NSPACE}(c^n) \subsetneq \text{NSPACE}((c + \epsilon)^n)$ .

Opomba: Zgornjih izrekov **ne moremo** zapisati za NTIME, ker se pri dokazu uporablja Savitchev izrek.

### 3.7.5 Izrek o vrzeli

Motivacija:

- Hierarhiji razredov DTIME in DSPACE sta "precej gosti" (že majhna sprostitev omejitev razredu doda nove jezike/probleme)
- To zagotavljata dva izreka (o hierarhiji), ki oba zahtevata, da so omejitve časovno/prostorsko verne funkcije
- **Vprašanje:** Ali bi lahko to zahtevo spustili, ne da bi vplivalo na izrek?
- **Odgovor:** Ne.

Priprava:

- izberimo poljubno totalno rekurzivno funkcijo  $g(n)$ , ki raste vsaj linearno  $\forall n: n \leq g(n) \star$
- izberimo poljubno totalno rekurzivno funkcijo  $S(n)$

- iz  $\star$  sledi:  $\forall n: S(n) \leq g(S(n))$

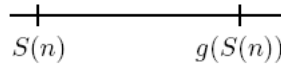
$\Rightarrow \text{DSPACE}(S(n)) \subsetneq \text{DSPACE}(g(S(n)))$

- pričakujemo, da bo velikokrat (največkrat) strogo vsebovanje:  $\subsetneq$ , saj zaradi  $\star$  funkcija  $g$  lahko narašča zelo hitro (npr.  $g(n) = 2^n$ ), da je gotovo  $\inf_{n \rightarrow \infty} \frac{S(n)}{g(S(n))} = 0$ , kar je (zadosten) pogoj v izreku o hierarhiji DSPACE.  
 Toda: izrek zahteva tudi, da je  $g(S(n))$  prostorsko verna. Mi pri definiciji in  $S$  tega nismo zahtevali.

**Vprašanje:** Ali lahko to izniči veljavnost izreka o hierarhiji DSPACE oz. naše pričakovanje, da je  $\text{DSPACE}(S(n)) \subsetneq \text{DSPACE}(g(S(n)))$ ?

**Odgovor:** Da. Lahko se zgodi, da povečanje prostora od  $S(n)$  na  $g(S(n))$  ne “prinese nič novega”.

Med  $S(n)$  in  $g(S(n))$  je kdaj “vrzel”.



Povečanje prostora omejenosti tu ne omogoči reševanja novih problemov

**Izrek:** Borodin: Za  $\forall$  totalno rekurzivno funkcijo  $g(n)$ , kjer  $\forall n: n \leq g(n) \exists$  totalna rekurzivna funkcija  $S(n)$ , da je  $\text{DSPACE}(S(n)) = \text{DSPACE}(g(S(n)))$ .

Opombe:  $g$  lahko poljubno izberemo. Ni nujno, da bo za **vsak**  $S$  vrzel med  $S(n)$  in  $g(S(n))$ . Izrek pravi le, da **obstaja** (vsaj ena taka  $S$ , ki je odvisna od  $g$ ), da bo med  $S(n)$  in  $g(S(n))$  vrzel.

Aksiomska teorija zahtevnosti (Blum) pokaže, da so vrzeli tudi pri DTIME, NTIME in NSPACE (in tudi pri vseh merah zahtevnosti, ki zadoščajo dvema začetnima aksiomoma).

Posledice izreka o vrzeli:

Primer 1:  $\exists$  totalno rekurzivna funkcija  $f(n)$ , da velja  $\text{DTIME}(f(n)) = \text{NTIME}(f(n)) = \text{DSPACE}(f(n)) = \text{NSPACE}(f(n))$ .

Primer 2: Naj bosta  $\mathcal{M}_1$  in  $\mathcal{M}_2$  dva poljubna univerzalna modela računalnika (TS ali RAM), tedaj  $\exists$  totalna rekurzivna funkcija  $T(n)$ , da je  $\forall$  funkcija, ki je izračunljiva v času  $T(n)$  na  $\mathcal{M}_1$  izračunljiva v istem času  $T(n)$  tudi na  $\mathcal{M}_2$ .

### 3.7.6 Izrek o uniji

Motivacija:

- $\text{DSPACE}(p(n))$ , kjer je  $p(n) = n$  ali  $n^2$  ali  $n^3, \dots$  so različni razredi, vsebovani vsak v nasledniku (Dokaz: izrek o hierarhiji DSPACE).
- podobno velja tudi za DTIME, NTIME, NSPACE.
- **Vprašanje:** Ali obstaja neka funkcija  $S(n)$ , da  $\text{DSPACE}(S(n))$  vsebuje vse  $\text{DSPACE}(p(n))$  in nič drugega? Ali  $\exists S(n): \text{DSPACE}(S(n)) = \bigcup_{p \in \text{polinom}} \text{DSPACE}(p(n))$ ?

Očitno bi moral tak  $S(n)$ :

- skoraj povsod (a.e., p.p.) hitreje naraščati od vsakega konkretnega polinoma
- naraščati dovolj počasi, da ne bi zajel superpolinomskih funkcij (kot npr.  $n^{\log n}, 2^n, \dots$ )
- **Odgovor:** Da, obstaja. To zagotavlja izrek o uniji

**Izrek o uniji:** Naj bo  $F = \{f_i(n) \mid i = 1, 2, \dots\}$  množica nekih rekurzivnih funkcij in naj bo  $F$  Turingova (torej  $\exists$  TS  $M: M$  oštevilči vse kode strojev  $M_1, M_2, \dots$ , kjer  $M_i$  računa funkcijo  $f_i$ ).

Opomba: torej lahko za vsako funkcijo  $f_i \in F$  generiramo kodo njenega stroja  $M_i$  in nato ta  $M_i$  poženemo, da z njim izračunamo vrednost  $f_i(n)$ .

Tedaj obstaja rekurzivna funkcija  $S(n)$ , da velja:

$$\text{DSPACE}(S(n)) = \bigcup_{i \geq 1} \text{DSPACE}(f_i(n))$$

**Dokaz:** (H.U., str. 310)

Uporaba:

- za  $F$  lahko vzamemo  $F = \{p_i(n) \mid p_i \text{ je polinom}\}$

$\Rightarrow$  (po izreku o uniji) torej  $\exists S(n): \text{DSPACE}(S(n)) = \bigcup_{p_i \in F} \text{DSPACE}(p_i(n)) \dots$  to je tudi razred neke deterministične prostorske zahtevnosti

- podobno naredimo tudi pri DTIME, NTIME, NSPACE

Posledica: Tako definiramo naslednja nova imena razredov zahtevnosti:

- $P \stackrel{\text{def.}}{=} \bigcup_{i \geq 1} \text{DTIME}(n^i)$

jeziki, ki jih TS **deterministično sprejemajo** v polinomsko omejenem času  
 problemi, ki jih **deterministično rešimo** v polinomsko omejenem času  
 problemi, kjer rešitev **deterministično preverimo** v polinomsko omejenem času

- $NP \stackrel{\text{def.}}{=} \bigcup_{i \geq 1} \text{NTIME}(n^i)$

jeziki, ki jih TS **nedeterministično sprejemajo** v polinomsko omejenem času  
 problemi, ki jih **nedeterministično rešimo** v polinomsko omejenem času  
 problemi, kjer rešitev **deterministično preverimo** v polinomsko omejenem času

- $PSPACE \stackrel{\text{def.}}{=} \bigcup_{i \geq 1} \text{DSPACE}(n^i)$

jeziki, ki jih TS **deterministično sprejemajo** na polinomsko omejenem prostoru  
 problemi, ki jih **deterministično rešimo** na polinomsko omejenem prostoru  
 problemi, kjer rešitev **deterministično preverimo** na polinomsko omejenem prostoru

- $NSPACE$  ali  $NPSPACE \stackrel{\text{def.}}{=} \bigcup_{i \geq 1} \text{NSPACE}(n^i)$

jeziki, ki jih TS **nedeterministično sprejemajo** na polinomsko omejenem prostoru  
 problemi, ki jih **nedeterministično rešimo** na polinomsko omejenem prostoru  
 problemi, kjer rešitev **deterministično preverimo** na polinomsko omejenem prostoru

### 3.7.7 Polinomsko omejen čas in prostor

**Vprašanje:** V kakšnem odnosu so  $P, NP, PSPACE$  in  $NSPACE$ ?

**Trditev:**  $P \subseteq NP \subseteq PSPACE = NSPACE$

**Dokaz:** (za  $PSPACE = NSPACE$ )

Gotovo:  $PSPACE \subseteq NSPACE$

Obratno:  $NSPACE \stackrel{\text{def.}}{=} \bigcup_{i \geq 1} \text{NSPACE}(n^i) \stackrel{\text{Savitch}}{\subseteq} \bigcup_{i \geq 1} \text{DSPACE}(n^{2i}) \subseteq PSPACE$

Torej: PSPACE = NSPACE

**Dokaz:** (za  $NP \subseteq PSPACE$ )

Če  $L \in NP \Rightarrow \exists k: L \in NTIME(n^k) \Rightarrow L \in NSPACE(n^k) \xrightarrow{\text{Savitch}} L \in DSPACE(n^{2k}) \Rightarrow L \in PSPACE$ .

**Dokaz:** (za  $P \subseteq NP$ )

Vsak polinomsko časovno omejen deterministični TS je tudi polinomsko omejen nedeterministični TS.

Komentar:

- PSPACE = NSPACE, če je prostor polinomsko omejen, dodajanje nedeterminizma ne poveča računske moči
- **Vprašanje:** Ali kaj takega velja tudi pri polinomsko omejenem času? Ali velja  $P = NP$ ?  
**Vprašanje:** Ali lahko vsak polinomsko časovno omejen nedeterministični izračun, deterministično simuliramo v polinomskem času (vemo že, da ga lahko deterministično simuliramo v eksponentnem času)?  
**Vprašanje:** Ali vsako rešitev, ki je preverljiva v polinomskem času, lahko tudi konstruiramo v polinomskem času?  
**Vprašanje:** Ali magična kocka (kovanec) nič ne pomaga, če je čas polinomsko omejen?
- Domnevamo, da to ne more biti tako.  
Domneva:  $P \neq NP$ , toda dokaza nimamo.
- Vprašanje je pomembno, ker je ogromno praktičnih problemov ravno v NP

**Trditev:**  $DSPACE(\log n) \subseteq P$

**Dokaz:** Če je  $L \in DSPACE(\log n) \xrightarrow{\text{izrek}} \exists c_L: L \in DTIME(c_L^{\log n}) \Rightarrow / \log n = k_L \cdot \log_{c_L} n / \Rightarrow L \in DTIME(c_L^{k_L \cdot \log_{c_L} n}) = DTIME(n^{k_L}) \Rightarrow L \in P$

**Trditev:**  $DSPACE() \subsetneq PSPACE$

**Dokaz:**  $\log n$  in  $n$  sta prostorsko verni funkciji in tudi  $\inf_{n \rightarrow \infty} \frac{\log n}{n} = 0$ , zato (po izreku o hierarhiji) velja:  $DSPACE(\log n) \subsetneq DSPACE(n) \subsetneq PSPACE$

**Posledica:** Vsaj eno vsebovanje izmed  $DSPACE(\log n) \subseteq P \subseteq NP \subseteq PSPACE$  je strogo vsebovanje ( $\subsetneq$ ).

**Vprašanje:** Ali je strogo morda  $DSPACE(\log n) \subsetneq P$ ?

**Odgovor:** Še ni znan.

Razmišljanje:

- $DSPACE(\log n) \subsetneq DSPACE(\log^2 n) \subsetneq DSPACE(\log^3 n) \subsetneq \dots$  (dokaz z izrekom o hierarhiji DSPACE)
- torej je  $\bigcup_{i \geq 1} DSPACE(\log^i n)$  "precej velik"
- te unije ne poveča niti dodatek nedeterminizma:  
$$\bigcup_{i \geq 1} DSPACE(\log^i n) = \bigcup_{i \geq 1} NSPACE(\log^i n)$$
  
(Dokaz:  $\bigcup_i DSPACE(\log^i n) \subsetneq \bigcup_i NSPACE(\log^i n)$   
Obratno: uporabimo Savitchev izrek)

- Zamisel: Morda pa je  $\bigcup_i \text{DSPACE}(\log^i n) = \text{P}$   
Če bi to dokazali, bi sledilo  $\text{DSPACE}(\log n) \subsetneq \text{P}$
- Toda: V resnici je znano le, da je  $\bigcup_i \text{DSPACE}(\log^i n) \neq \text{P}$ , ni pa znano kdo od njiju drugega vsebuje, če sploh.

**Vprašanje:**  $\text{P} \stackrel{?}{=} \text{NP}$

Zakaj nas zanima odgovor?

**Odgovor:** Ker:

- ogromno praktičnih problemov sodi v razred NP (gotovo njihove rešitve lahko **preverimo** v polinomskem času)
- vsakega od njih gotovo lahko **rešimo** kvečjemu v eksponentnem času  
(**Dokaz:**  $L \in \text{NP} \Rightarrow \exists k: L \in \text{NTIME}(n^k) \xrightarrow{\text{izrek}} \exists c_L: L \in \text{DTIME}(c_L n^k)$ )
- Toda: to je v praksi prepočasi

$\Rightarrow$  **Vprašanje:** Ali takega problema (ki je v NP) res ne moremo rešiti deterministično v polinomskem času?

**Vprašanje:** Ali je nedeterministično polinomsko omejeni TS močnejši od determinističnega polinomsko omejenega TS?

**Vprašanje:** Ali je čudežni kovanec močnejši od običajnega kovanca, če želimo računanje v polinomskem času?

**Vprašanje:** Ali je konstrukcija rešitve v splošnem "težja" od njenega preverjanja, če želimo oboje zaključiti v polinomsko omejenem času?

**Vprašanje:** Kako iskati odgovor na vprašanje  $\text{P} \stackrel{?}{=} \text{NP}$ ?

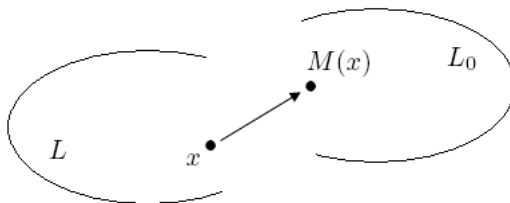
Ker domnevamo, da je  $\text{P} \subsetneq \text{NP}$ , bomo skušali dokazati ravno to.

Metoda:

- Išči najtežje probleme v NP in skušaj za vsaj enega dokazati, da ni v P.  
Namreč: če je nek problem v NP in ni v P ( $\text{NP} \setminus \text{P}$ ) je gotovo najtežji v NP.
- **Vprašanje:** Kdaj je nek problem v NP najtežji?  
**Odgovor:** Kadar je vsak drug problem v NP kvečjemu tako težek, t.j. kadar vsak drug problem v NP lahko prevedemo nanj.
- **Vprašanje:** Kako problem prevedemo?

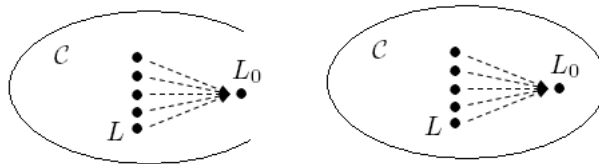
### 3.8 Splošno o prevedbah

**Definicija:** Jezik (problem)  $L$  je  $R$ -prevedljiv na jezik  $L_0$  ( $L \leq^R L_0$ ), če  $\exists$  deterministični TS  $M$  z lastnostjo  $R$ , ki za vsak vhod  $x$  vrne izhod  $M(x)$ , za katerega velja:  $M(x) \in L_0 \Leftrightarrow x \in L$ .



Komentar: s pomočjo stroja  $M$  vprašanje  $x \stackrel{?}{\in} L$  prevedemo na vprašanje, če  $M(x) \stackrel{?}{\in} L_0$  (ker je odgovor na drugo vprašanje tudi odgovor na prvo vprašanje).

**Definicija:** Naj bo  $\mathcal{C}$  razred nekih jezikov (problemov). Jezik  $L_0$  je  $\mathcal{C}$ -težek (glede na  $\stackrel{R}{\leq}$ ), če za  $\forall L \in \mathcal{C}$ :  $L \stackrel{R}{\leq} L_0$ .



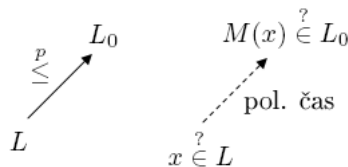
**Definicija:** Jezik (problem)  $L_0$  je  $\mathcal{C}$ -poln (glede na  $\stackrel{R}{\leq}$ ), če je  $L_0$   $\mathcal{C}$ -težek in hkrati  $L \in \mathcal{C}$ .

### 3.9 Vprašanje $P \stackrel{?}{=} NP$ in prevedbe

Za reševanje vprašanja  $P \stackrel{?}{=} NP$  postavimo:

- $\mathcal{C} \equiv NP$
- $R \equiv$  polinomsko časovno omejen TS ali logaritmsko prostorsko omejen pretvornik
- $\stackrel{R}{\leq} \equiv$  polinomska časovna prevedba ( $\stackrel{p}{\leq}$ ) ali logaritmska prostorska prevedba ( $\stackrel{\log}{\leq}$ )

**Definicija:** Jezik (problem)  $L$  je polinomsko časovno prevedljiv na jezik  $L_0$  ( $L \stackrel{p}{\leq} L_0$ ), če  $\exists$  polinomsko časovno omejen deterministični TS  $M$ , ki za vsak vhod  $x$  vrne izhod  $M(x)$ , za katerega velja:  $M(x) \in L_0 \Leftrightarrow x \in L$ .



**Vprašanje:** Zakaj smo izbrali samo takšno lastnost  $R$ ?

**Odgovor:** Zato ker takšna prevedba omogoča, da:

$M(x)$  sestavimo v polinomskem času  $p(|x|)$ , kar pomeni, da je dolžina  $|M(x)| \leq p(|x|)$ .

**Trditev:** Naj bo  $L \stackrel{p}{\leq} L_0$  ( $L$  je polinomsko časovno prevedljiv na  $L_0$ ). Tedaj velja:

- $L_0 \in P \Rightarrow L \in P$
- $L_0 \in NP \Rightarrow L \in NP$

**Dokaz:**

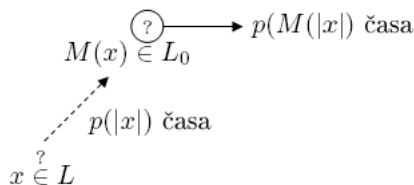
- Naj bo  $L \stackrel{p}{\leq} L_0 \wedge L_0 \in P$ . Ker  $L_0 \in P \Rightarrow \exists$  polinom  $p_0(n)$ :  $L_0 \in \text{DTIME}(p_0(n))$ . Po predpostavki pa  $\exists$  polinom  $p(n)$ :  $L \stackrel{p}{\leq} L_0$

Naj bo  $x$  poljubna beseda. Na vprašanje  $x \stackrel{?}{\in} L$  lahko odgovorimo z naslednjim algoritmom:

- s prevedbo izračunaj besedo  $M(x)$  (za to rabimo  $\leq p(|x|)$  časa)
- odgovi na vprašanje  $M(x) \stackrel{?}{\in} L_0$  (za to rabimo  $\leq p_0(p(|x|))$  časa)

Ugotovimo: na vprašanje  $x \in L$  lahko odgovarjamo v času  $p(|x|) + p_0(p(|x|))$ , kar je polinomski čas  $\Rightarrow L \in P$ .

b) Dokaz je podoben dokazu pod točko a).

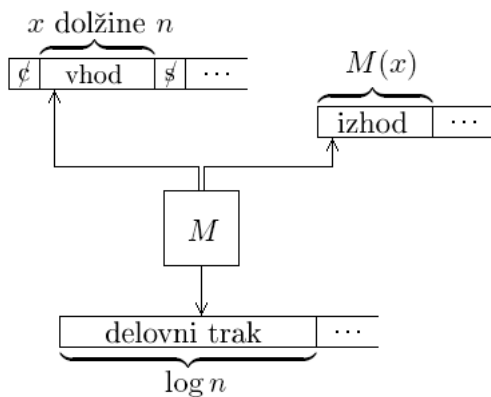


**Vprašanje:** Kaj pa druga lastnost  $R$ ?

**Definicija:** Pretvornik (transducer) z logaritmsko prostorsko omejitvijo je deterministični TS, ki:

- ima en vhodni trak (z njega sme le brati)
- ima en izhodni trak (nanj sme le pisati, okna ne sme premakniti v levo)
- ima en delovni trak (ki ima logaritmsko prostorsko omejitev)
- se vedno ustavi (za vsak vhod)

Opomba: ideja je v tem, da če sme porabiti le logaritmsko velik prostor (na delovnem traku), lahko gradi izhod le polinomsko dolgo časa. Po daljši gradnji bi se namreč moral nek trenutni opis ponoviti  $\Rightarrow$  stroj bi se zaciklal in nikoli ustavil, kar pa bi bilo v nasprotju z definicijo pretvornika. To pomeni, da je izhod polinomsko dolg glede na vhod.



**Definicija:** Jezik (problem)  $L$  je logaritmsko prostorsko prevedljiv na jezik  $L_0$  ( $L \stackrel{\log}{\leq} L_0$ ), če  $\exists$  logaritmsko prostorsko omejen pretvornik  $M$ , ki za vsak vhod  $x$  vrne izhod  $M(x)$ , za katerega velja:  $M(x) \in L_0 \Leftrightarrow x \in L$ .

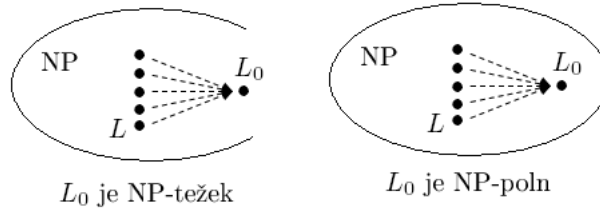
**Trditev:** : Naj bo  $L \stackrel{\log}{\leq} L_0$ . Tedaj velja:

- $L_0 \in P \Rightarrow L \in P$
- $L_0 \in \text{DSPACE}(\log^k n) \Rightarrow L \in \text{DSPACE}(\log^k n)$
- $L_0 \in \text{NSPACE}(\log^k n) \Rightarrow L \in \text{DSPACE}(\log^k n)$

### 3.10 Dokazovanje NP-polnosti (težnosti)

**Definicija:** Jezik  $L_0$  je NP-težek, če za  $\forall L \in \text{NP}$ :  $L \stackrel{p}{\leq} L_0 \vee L \stackrel{\log}{\leq} L_0$ .

**Definicija:** Jezik  $L_0$  je NP-poln, če je  $L_0$  NP-težek  $\wedge L_0 \in \text{NP}$ .



**Trditev:** Relaciji  $\stackrel{p}{\leq}$  oz.  $\stackrel{\log}{\leq}$  sta tranzitivni. Kompozitum dveh polinomskih časovnih (logaritamskih prostorskih) prevedb je spet polinomska časovna (logaritamska prostorska) prevedba.

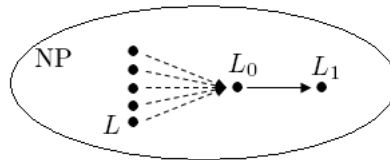
$$L \stackrel{p}{\leq} L_0 \wedge L_0 \stackrel{p}{\leq} L_1 \Rightarrow L \stackrel{p}{\leq} L_1$$

$$L \stackrel{\log}{\leq} L_0 \wedge L_0 \stackrel{\log}{\leq} L_1 \Rightarrow L \stackrel{\log}{\leq} L_1$$

Tranzitivnost  $\stackrel{p}{\leq}$  in  $\stackrel{\log}{\leq}$  odpira možnost za dokazovanje NP-težnosti/polnosti nekega jezika  $L_1$  tako, da nanj prevedemo nek NP-težek/poln jezik  $L_0$ .

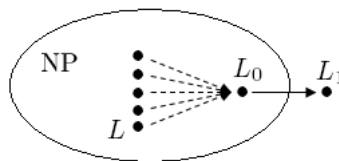
Denimo, da vemo, da je  $L_0$  NP-poln. Če  $L_0$  prevedemo ( $\stackrel{p}{\leq}$  ali  $\stackrel{\log}{\leq}$ ) na  $L_1$ , so (zaradi tranzitivnosti prevedb) na  $L_1$  prevedljivi vsi ostali jeziki iz NP.

Torej:  $L_0$  NP-poln  $\wedge L_0 \stackrel{p, \log}{\leq} L_1 \Rightarrow L_1$  NP-poln



Podobno velja za NP-težnost  $L_1$ .

Torej:  $L_0$  NP-težek  $\wedge L_0 \stackrel{p, \log}{\leq} L_1 \Rightarrow L_1$  NP-težek



**Vprašanje:** Kako pa dokazati NP-polnost prvega jezika (problema)  $L_0$ ? Ker je jezikov v NP neskončno mnogo, ne moremo prevesti na  $L_0$  vsakega posebej.

**Vprašanje:** Ali se sploh da dokazati NP-polnost prvega NP-polnega jezika (problema)?

**Odgovor:** Da. To je naredil S. Cook leta 1971, v Rusiji pa nekje ob istem času L. Levin, kar pa se je odkrilo šele pozneje.

### 3.11 Problem izpolnljivosti Boolovih izrazov

**Definicija:** Boolov izraz sestavljajo:

- Boolove spremenljivke:  $x_1, x_2, \dots$  (ki lahko zavzamejo vrednost 0 ali 1)
- povezane z Boolovimi operacijami ( $\neg, \wedge, \vee$ )

**Definicija:** Boolov izraz:

- $\neg E$  ima vrednost  $\begin{cases} 1 & \text{če ima } E \text{ vrednost } 0 \\ 0 & \text{če ima } E \text{ vrednost } 1 \end{cases}$
- $E_1 \wedge E_2$  ima vrednost  $\begin{cases} 1 & \text{če imata } E_1 \text{ in } E_2 \text{ vrednost } 1 \\ 0 & \text{če ima } E_1 \text{ ali } E_2 \text{ vrednost } 0 \end{cases}$
- $E_1 \vee E_2$  ima vrednost  $\begin{cases} 1 & \text{če ima } E_1 \text{ ali } E_2 \text{ vrednost } 1 \\ 0 & \text{če imata } E_1 \text{ in } E_2 \text{ vrednost } 0 \end{cases}$

**Definicija:** Boolov izraz je izpolnljiv (satisfiable), če lahko njegove spremenljivke zavzamejo take vrednosti, da je vrednost izraza 1 (problem izpolnljivosti).

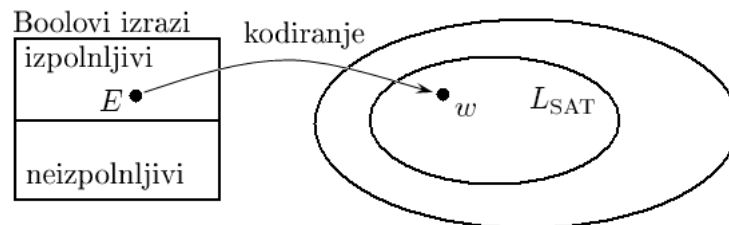
Opombe:

- Ni vsak Boolov izraz izpolnljiv
- Če ima Boolov izraz  $n$  spremenljivk lahko te zavzamejo vrednosti na  $2^n$  načinov.

**Definicija:** Problem izpolnljivosti: “Ali je dani Boolov izraz izpolnljiv?” (Boolean satisfiability problem - SAT).

#### 3.11.1 Problem izpolnljivosti je NP-poln (Cook)

**Definicija:** Problemu pripada jezik  $L_{SAT}$  (ki vsebuje kode izpolnljivih Boolovih izrazov).



$$L_{SAT} = \{w \in \{0, 1\}^* \mid w \text{ je koda nekega izpolnljivega Boolovega izraza}\}$$

Opombe:

- Simboli  $\neg, \wedge, \vee, ( \text{ in } )$  se enostavno zakodirajo (s konstantno mnogo 0 oz. 1).
- Pri kodiranju spremenljivk  $x_i$  pa moramo poleg  $x$  zakodirati tudi indeks  $i$ .
- Ker je  $1 \leq i \leq m$  bo  $x$  zakodiran z  $c_x + \lceil \log_2 m \rceil$  simboli.
- Naj bo  $n$  dolžina izraza  $E$ . Tedaj  $m \leq n$ . Torej je dolžina  $|w| \leq n \cdot (c + \lceil \log_2 m \rceil) = \mathcal{O}(n \cdot \log n)$  simbolov.

**Trditev:**  $L_{\text{SAT}}$  je NP-težek.

**Dokaz:** Dokazati je treba:

$$\forall L (L \in \text{NP}): L \stackrel{\log}{\leq} L_{\text{SAT}}.$$

Če prevedemo na TS:

$$\forall \text{TS } T (L(T) \in \text{NP}): L(T) \stackrel{\log}{\leq} L_{\text{SAT}}.$$

Namesto, da je  $L(T) \in \text{NP}$  lahko zahtevamo, da je  $T$  nedeterministični in polinomsko časovno omejen:

$$\forall \text{ pol. čas. omejen NTS } T: L(T) \stackrel{\log}{\leq} L_{\text{SAT}},$$

kar pomeni:

$$\forall \text{ pol. čas. omejen NTS } T: \exists \text{ log. prost. omejen pretvornik } M: x \in L(T) \Leftrightarrow M(x) \in L_{\text{SAT}}$$

Situacijo " $x \in L(T)$ " izrazimo s sprejemajočim izračunom stroja  $T$  pri vходу  $x$ .

$\exists$  nedeterministični polinomsko časovno omejen izračun v katerem  $T$  sprejme  $x$ .

Šele na podlagi tega sprejemajočega izračuna  $\#\beta_0\#\beta_1\#\dots\#$  bo pretvornik  $M$  sestavil izpolnljiv Boolov izraz  $M(x)$ .

Torej moramo dokazati tole:

$$\forall \text{TS } T \exists M: \exists \text{ nedet. pol. čas. omejen izračun, v katerem TS } T \text{ sprejme vhod } x \Leftrightarrow \\ \Leftrightarrow \exists \text{ prireditve vrednosti spremenljivkam Boolovega izraza } M(x), \text{ da ima ta vrednost } 1$$

Velja:

- $T$  je NTS s polinomsko omejenim časom
- $M$  je logaritemsko prostorsko omejen pretvornik

**Vprašanje:** Kako na podlagi izračuna (= zaporedja trenutnih opisov TO) sestaviti Boolov izraz, ki bo izpolnljiv natanko tedaj, ko je izračun sprejemajoč (ko se zaključi v končnem stanju).

Koraki:

1. Izračun NTS  $T$  pri vходу  $x$  je neko zaporedje TO  $\beta_i, i = 0, 1, \dots, z$ .  
Dolžina vhoda je  $n = |x|$ . Ker je  $T$  pol. čas. omejen z nekim polinomom  $p$ , zato je  $z \leq p(n)$  (trenutnih opisov ne more biti več kot  $p(n)$ ).

2. Poenostavimo zapis izračunov, da bo lažja splošna obravnava

- 2.1. vsak TO  $\beta_i$  sestavljajo

- $\leq z$  tračnih simbolov (ker na vsakem koraku računanja lahko  $T$  doda le po en simbol)
- simbol za stanje
- številka poteze zaradi katere bo sledil naslednji TO  $\beta_{i+1}$

V  $\beta_i$  bi bilo torej kvečjemu  $z + 2$  simbolov, toda simbola za stanje in tračni simbol pod oknom združimo s simbolom za izbrano potezo v en sam simbol.

**Posledica:** : zato je  $|\beta_i| \leq z$ .

- 2.2 Izračun je zapisan kot beseda  $\#\beta_0\#\beta_1\#\dots\#\beta_z$ , kjer je  $z \leq p(n)$  in  $|\beta_i| \leq z$ .

Vsak  $\beta_i$  po potrebi dopolnimo s presledki, da bo dolg natanko  $p(n)$  simbolov.

Zadnji TO  $\beta_z$  še ponavljamo, da bo vseh opisov  $p(n) + 1$ .

**Posledica:** izračun stroja  $T$  nad vходом  $x$  opisuje beseda  $\#\beta_0\#\beta_1\#\dots\#\beta_{p(n)}$ , ki je dolga natanko  $(p(n)+1)^2$  simbolov (simbole označujemo z indeksi od 0 do  $(p(n)+1)^2 - 1$ ).

3. Uvedemo množico Boolovih spremenljivk  $c_{is}$ , kjer je  $0 \leq i \leq (p(n) + 1)^2 - 1$  in  $s \in \Gamma \cup \{\#\}$  ( $c_{is}$  bo povedal ali je v besedi  $\#\beta_0\#\beta_1\#\dots\#\beta_{p(n)}$  na  $i$ -tem mestu simbol  $s$ ).
4. Iz spremenljivk  $c_{is}$  bomo sestavili Boolov izraz  $M(x)$  z naslednjo lastnostjo: "M(x) je resničen pri neki prireditvi vrednosti spremenljivkam  $c_{is}$ , natanko tedaj, ko spremenljivke  $c_{is}$  z vrednostjo 1 opisujejo izračun, v katerem stroj  $T$  sprejme vhod  $x$ ".
5. Izraz  $M(x)$  bomo sestavili iz spremenljivk  $c_{is}$  tako, da bo konjunkcija sestavljena iz 4 Boolovih izrazov, ki bodo povedali/zahtevali sledeče:
 

$A \equiv$  spremenljivke  $c_{is}$  z vrednostjo 1 opisujejo besedo  $\#\beta_0\#\beta_1\#\dots\#\beta_{p(n)}$   
(za  $\forall i = 0, 1, 2, \dots, (p(n) + 1)^2 - 1$  bo natanko en  $c_{is}$  enak 1, za nek  $s \in \Gamma \cup \{\#\}$ )

$B \equiv \beta_0$  vsebuje (ima vpisano) začetno stanje  $q_0$  stroja  $T$  in vhodno besedo  $x$ .

$C \equiv \beta_{p(n)}$  vsebuje (ima vpisano) neko končno stanje stroja  $T$ .

$D \equiv$  Vsak  $\beta_i$  sledi iz predhodnega  $\beta_{i-1}$  ( $1 \leq i \leq p(n)$ ) z eno od možnih potez stroja  $T$  (ta poteza bo vpisana v predhodniku  $\beta_{i-1}$ ).
6. Izrazi, ki sestavljajo  $M(x)$  so:

6.1.

$$A \equiv \bigwedge_{0 \leq i < (p(n)+1)^2} \left[ \bigvee_s c_{is} \wedge \neg \left( \bigvee_{r \neq s} (c_{is} \wedge c_{ir}) \right) \right]$$

$\bigvee_s c_{is}$ : na  $i$ -tem mestu mora biti vsaj en simbol

$\neg \left( \bigvee_{r \neq s} (c_{is} \wedge c_{ir}) \right)$ : na  $i$ -tem mestu ne smeta biti dva različna simbola, ampak kveč-

$\Rightarrow$  na  $i$ -tem mestu je **natanko 1 simbol** (iz  $\Gamma \cup \{\#\}$ )

6.2. Naj bo vhodna beseda  $x = a_1 a_2 \dots a_n$

$$B \equiv c_{0\#} \wedge c_{p(n)+1,\#} \wedge \left( \bigvee_k c_{1Y_k} \right) \wedge \bigwedge_{2 \leq i \leq n} c_i a_i \wedge \bigwedge_{n \leq i \leq p(n)} c_i B$$

- $c_{0\#} \wedge c_{p(n)+1,\#}$ : simbola na mestih 0 in  $p(n) + 1$  sta  $\#$
- $\bigvee_k c_{1Y_k}$ : je prvi simbol v  $\beta_0$ ,  $Y_k$  je sestavljen iz  $a_1, q_0$  in številke  $k \in \delta(q_0, q_1)$ , t.j. prve poteze stroja  $T$
- $\bigwedge_{2 \leq i \leq n} c_i a_i$ : naslednjih  $n - 1$  simbolov v  $\beta_0$  je preostanek vhodne besede  $x$
- $\bigwedge_{n \leq i \leq p(n)} c_i B$ : so presledki

6.3.

$$C \equiv \bigvee_{p(n)(p(n)+1) < i < (p(n)+1)^2} \bigvee_{X \in F} c_i X$$

kjer je  $F = \{X \mid X \text{ sestavljen simbol in vsebuje neko končno stanje stroja } T\}$

Zgornja disjunkcija zahteva, da naj eno mesto v TO  $\beta_p(n)$  zaseda sestavljen simbol, ki vsebuje končno stanje stroja  $T$

6.4. Uvedemo predikat  $f(W, X, Y, Z)$ , ki bo povedal ali se na kakem mestu kakega TO lahko pojavi simbol  $Z$ , če je bil na tem mestu v prejšnjem TO simbol  $X$  med simboloma  $W$  in  $Y$ .

Torej:

$$f(W, X, Y, Z) = 1 \Leftrightarrow \exists i, j : \# \beta_0 \# \dots \# \underbrace{\dots WXY \dots}_{\beta_{i-1}} \# \underbrace{\dots Z \dots}_{\beta_i} \# \dots$$

$X$  ima indeks  $j - p(n) - 1$ ,  $Z$  pa indeks  $j$  in sta na na razdalji  $p(n) + 1$

Opomba:  $f(W, X, Y, Z)$  je lahko 1:

- če je  $T$  v eni potezi spremenil simbol  $X$  v  $Z$
- če  $T$  v tej potezi ni vplival na  $X$  (in je zato  $Z = X$ )

Tedaj:

$$D \equiv \bigwedge_{p(n) < j < (p(n)+1)^2} \left( \bigvee_{W, X, Y, Z: f(W, X, Y, Z)=1} c_{j-p(n)-2, W} \wedge c_{j-p(n)-1, X} \wedge c_{j-p(n), Y} \wedge c_{j, Z} \right)$$

Trije zaporedni simboli  $(c_{j-p(n)-2, W} \wedge c_{j-p(n)-1, X} \wedge c_{j-p(n), Y})$  v  $\beta_{j-1}$  in soležni simbol  $(c_{j, Z})$  v  $\beta_j$  morajo biti v zvezi  $f(W, X, Y, Z)$ . Torej simbol  $Z$  mora biti posledica nekega prehoda stroja  $T$  ali pa mora ostati enak kot prej.

7. Postavimo:  $M(x) = A \wedge B \wedge C \wedge D$

**Trditev:** Izračun  $\# \beta_0 \# \beta_1 \# \dots \# \beta_{p(n)}$  sprejme  $x \Leftrightarrow M(x)$  je izpolnljiv

**Dokaz:**

( $\Rightarrow$ ) Če na osnovi  $\# \beta_0 \# \beta_1 \# \dots \# \beta_{p(n)}$  priredimo vrednosti spremenljivkam  $c_{is}$ , te izpolnijo izraz  $M(x)$ .

( $\Leftarrow$ ) Če prireditev  $\{c_{is}\} \rightarrow \{0, 1\}$  izpolni izraz  $M(x)$ , izrazi  $A, B, C$  in  $D$  zagotavljajo, da obstaja nek sprejemajoči izračun stroja  $T$  nad  $x$  (dobimo ga iz spremenljivk  $c_{is}$ , ki so 1).

8. Dokazati je treba še, da stroj  $M$  za konstrukcijo Boolovega izraza  $M(x)$  potrebuje kvečjemu  $\mathcal{O}(\log |x|)$  prostora (da je  $M$  lahko logaritemsko prostorsko omejen pretvornik).

Skica dokaza:

- za  $A$ :  $\mathcal{O}(p^2(n))$
- za  $B$ :  $\mathcal{O}(p(n))$
- za  $C$ :  $\mathcal{O}(p(n))$
- za  $D$ :  $\mathcal{O}(p^2(n))$
- $|M(x)| = \mathcal{O}(p^2(n))$ ,  $n = |x|$
- Izrazi  $A, B, C$  in  $D$  so "relativno" enostavni, ne predstavljajo resne težave za računanje s TS  $M$ .
- Vendar mora stroj  $M$  med sestavljanjem izrazov  $A, B, C$  in  $D$  večkrat šteti do  $(p(n) + 1)^2$ .
- Prostor, ki ga za to nujno potrebuje je velik  $\log((p(n) + 1)^2) = \Theta(\log p(n)) = /$  ker je  $p(n)$  polinom  $/ = \Theta(\log n) \Rightarrow$  zadošča logaritemsko omejen prostor za delo stroja  $M$