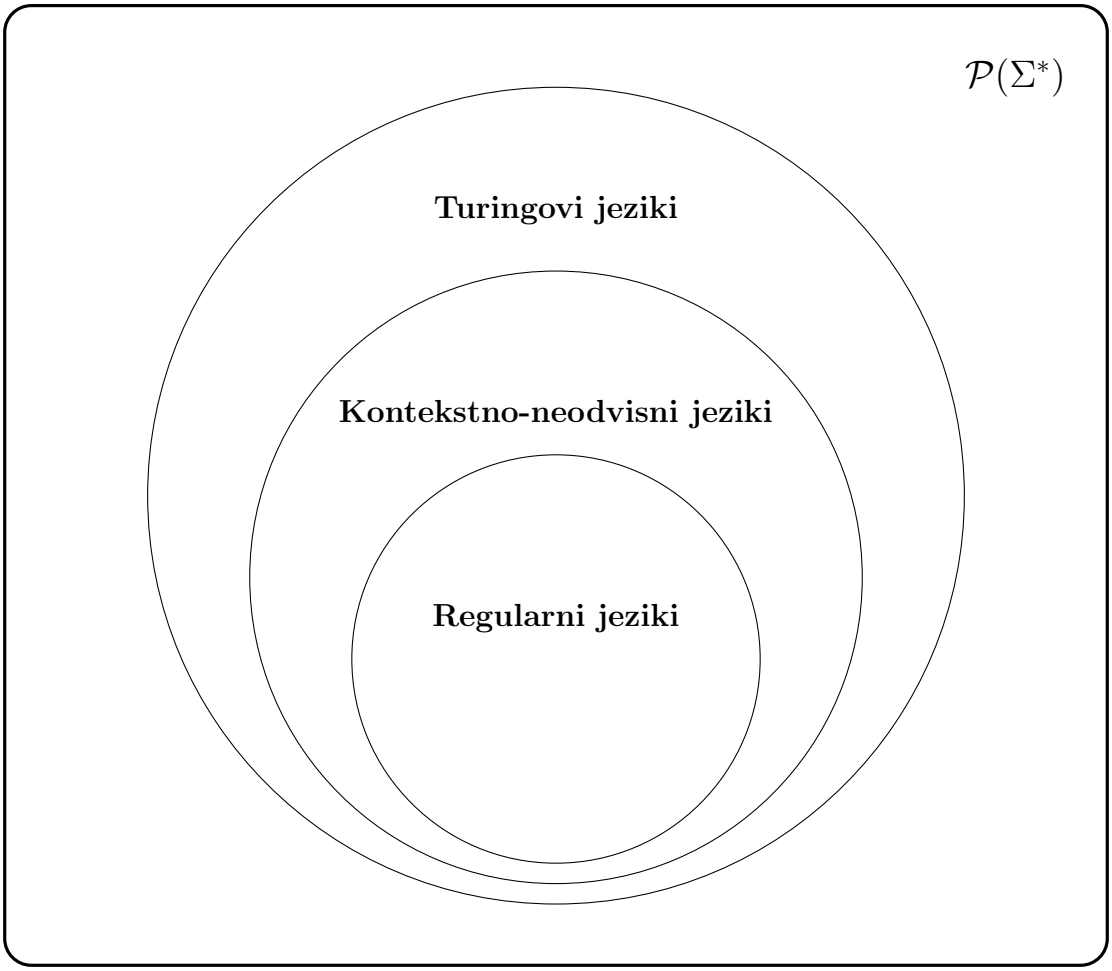
Teoretične osnove računalništva

27. september 2011



Kazalo

Poglavje 1

Uvod

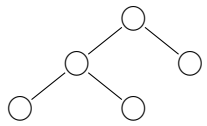
1.1 Osnove dokazovanja

1.1.1 Dokaz s konstrukcijo

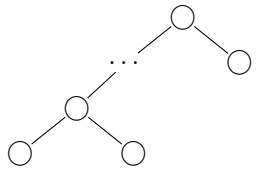
Dokaz obstoja nekega matematičnega objekta je to, da nam ga uspe sestaviti.

Primeri:

Primer 1: Za vsak $n > 4$, obstaja dvojiško drevo, ki ima natanko 3 liste.
Primer za $n = 5$:



Primer za $n > 5$, pri čemer je "...", poljubno vejitev samo v levo:



Primer 2: $|\mathbb{R}| = |[0,1)|$.

- Množici imata enako moč, kadar med njima obstaja bijektivna preslikava.
- Vsako realno število r lahko zapišemo kot:

$$r = \pm d_1 d_2 \cdots d_n . \overline{d_1 d_2} \cdots \overline{d_m} \cdots ; \; d_1 \neq 0$$

- Definiramo preslikavo:

$$\mathbb{R} \to [0,1) : r \to 0.s\overline{d_1} d_n \overline{d_2} d_{n-1} \cdots \overline{d_{n-1}} d_2 \overline{d_n} d_1 \overline{d_{n+1}} 0 \overline{d_{n+2}} 0 \cdots$$

  kjer s določa predznak ($s = 0$, če $r \geq 0$ in $s = 1$, sicer).
- Vidimo:
  - $|\mathbb{R}| \geq |[0,1)|$, ker velja $[0,1) \subset \mathbb{R}$
  - $|\mathbb{R}| \leq |[0,1)|$, •
- Iz tega lahko sklepamo, da velja $|\mathbb{R}| = |[0,1)|$

1.1.2 Dokaz z indukcijo

Če je množica induktivni razred, lahko z matematično indukcijo dokazujemo neko lastnost članov množice. Induktivni razred I sestavlja:

- Baza indukcije - najbolj osnovna množica elementov (osnovni razred)
- Pravila generiranja - kako iz elementov baze gradimo nove elemente (množico)

Primeri:

Primer 1: Induktivni razred naravnih števil ($\mathbb{N}$)

- Baza: $1 \in \mathbb{N}$
- Pravila generiranja: $n \in \mathbb{N} \Longrightarrow n + 1 \in \mathbb{N}$

Primer 2: Hilbertove krivulje •

1.1.3 Dokaz s protislovjem

Vzamemo nasprotno trditev, od tiste, ki jo želimo preveriti in pokažemo, da to vodi v protislovje.

Primeri:

Primer 1: Praštevil je končno mnogo.

- Predpostavimo, da poznamo vsa praštevila:
  $P = \{2, 3, 5, ..., p\}$, kjer je p zadnje praštevilo
- Po definiciji obstajajo le praštevila in sestavljena števila (to so taka, ki jih lahko razstavimo na prafaktorje).
- Če pomnožimo vsa znana praštevila iz P in prištejemo 1 dobimo število, ki se ga ne da razstaviti na prafaktorje iz množice P:
  $q = 2 * 3 * 5 * ... * p + 1$
- Torej je q ali praštevilo (ker ni sestavljeno), ali pa število, sestavljeno iz prafaktorjev, ki jih ni v množici P.
- Oboje kaže na to, da v množici P nimamo vseh praštevil, ter, da to velja za vsako končno množico praštevil.

Primer 2: $\sqrt[3]{2}$ je racionalno število.

- Če je $\sqrt[3]{2}$ racionalno število, ga je moč zapisati kot ulomek $\frac{a}{b}$.
- Predpostavimo, da je ulomek $\frac{a}{b}$ okrajšan (torej, da velja: $GCD(a, b) = 1$):

$$\begin{aligned} \sqrt[3]{2} &= \frac{a}{b} \\ 2 &= \left(\frac{a}{b}\right)^3 \\ 2b^3 &= a^3 \end{aligned}$$

- Opazimo, da je a sodo število, torej lahko pišemo $a = 2k$:

$$\begin{aligned} 2b &= (2k)^3 \\ 2b &= 8k \\ b &= 4k \end{aligned}$$

- Ker se je pokazalo, da je tudi b sodo število, $GCD(a, b) = 1$ ne more držati, torej smo prišli v protislovje in s tem dokazali, da $\sqrt[3]{2}$ ni racionalno število.

1.2 Osnove teorije jezikov

1.2.1 Uporabljene oznake

- a - znak ali simbol
- Σ - abeceda (končna neprazna množica znakov)
- w - niz ali beseda (poljubno končno zaporedje znakov), $w = a_1 a_2 \dots a_n$
- $|w|$ - dolžina niza
- ε - prazen niz, $|w| = 0$
- Σ^* - vsi nizi nad abecedo

1.2.2 Operacije nad jeziki

Stik nizov

$$\begin{aligned} w &= a_1 a_2 \dots a_n \\ x &= b_1 b_2 \dots b_m \\ wx &= a_1 a_2 \dots a_n b_1 b_2 \dots b_m \end{aligned}$$

Stik množic

$$\begin{aligned} A &= \{w_1,\ w_2,\ \dots,\ w_n\} \\ B &= \{x_1,\ x_2,\ \dots,\ x_m\} \\ A \cdot B &= \{w_i x_j \mid w_i \in A \ \wedge \ x_i \in B\} \end{aligned}$$

Potenciranje

$$\begin{aligned} A^0 &= \{\varepsilon\} \\ A^k &= A \cdot A \cdot \ldots \cdot A \ = \ \bigcirc_{i=1}^{k} A \end{aligned}$$

Iteracija

$$A^* = A^0 \cup A^1 \cup A^2 \cdots \ = \ \bigcup_{i=0}^{\infty} A^i$$

Obrat

$$\begin{aligned} w \ &= a_1 a_2 \dots a_{n-1} a_n \\ w^R &= a_n a_{n-1} \dots a_2 a_1 \end{aligned}$$

Poglavje 2

Regularni jeziki

2.1 Regularni izrazi

Def.: Imamo tri osnovne izraze:

- $\underline{\emptyset}$ je opisuje prazen jezik $L(\underline{\emptyset}) = \{\}$
- $\underline{\varepsilon}$ opisuje jezik $L(\underline{\varepsilon}) = \{\varepsilon\}$
- $\underline{a}$ opisuje jezik $L(\underline{a}) = \{a\},\ a \in \Sigma$

In tri pravila za generiranje sestavljenih izrazov:

- $(r_1 + r_2)$ opisuje unijo jezikov $L(r_1 + r_2) = L(r_1) \bigcup L(r_2)$
- $(r_1\ r_2)$ opisuje stik jezikov $L(r_1\ r_2) = L(r_1) \cdot L(r_2)$
- (r^*) opisuje iteracijo jezika $(L(r))^*$

Primeri:

Primer 1: Opiši vse nize, ki se končajo z nizom 00 v abecedi $\Sigma = \{0, 1\}$.

$$r = (0 + 1)^*00$$

Primer 2: Opiši vse nize, pri katerih so vsi a-ji pred b-ji in vsi b-ji pred c-ji v abecedi $\Sigma = \{a, b, c\}$.

$$a^*b^*c^*$$

Primer 3: Opiši vse nize, ki vsebujejo vsaj dva niza 'aa', ki se ne prekrivata v abecedi $\Sigma = \{a, b, c\}$.

$$(a + b + c)^*aa(a + b + c)^*aa(a + b + c)^*$$

Primer 4: Opiši vse nize, ki vsebuje vsaj dva niza 'aa' ki se lahko prekrivata v abecedi $\Sigma = \{a, b, c\}$

$$(a + b + c)^*aa(a + b + c)^*aa(a + b + c)^* + (a + b + c)^*aaa(a + b + c)^*$$

Primer 5: Opiši vse nize, ki ne vsebujejo niza 11 v abecedi $\Sigma = \{0, 1\}$

$$(\varepsilon + 1)(0^*01)^*0^*$$

$$(\varepsilon + 1)(0^* + 01)^*$$

Primer 6: S slovensko abecedo opiši besedo "Ljubljana"v vseh sklonih in vseh mešanicah velikih in malih črk.

$$(L + l)(J + j)(U + u)(B + b)(L + l)(J + j)(A + a)(N + n)((A + a)(O + o)(E + e)(I + i))$$

Koliko različnih nizov opišemo s tem regularnim izrazom?

$2^8 \cdot 2^3 = 2^{11}$ nizov

2.2 Končni avtomati

2.2.1 Nedeterministični končni avtomati z ε-prehodi

Def.: εNKA je definiran kot peterka $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, kjer je:

- Q - končna množica stanj
- Σ - vhodna abeceda
- δ - funkcija prehodov, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$
- q_0 - začetno stanje
- F - množica končnih stanj

$2^Q = P(Q)$ je tu potenčna množica stanj avtomata. To pomeni da je so v 2^Q vse možne kombinacije stanj. Recimo da se nahajamo v stanju A, potem nas funkcija prehodov δ pripelje v vsa mozna stanja do katerih pridemo iz A z določenim znakom abecede in z vsemi ε prehodi, naprimer $\{A_1, A2, \ldots, A_n\}$. Tukaj je množica stanj $\{A_1, A2, \ldots, A_n\}$ element potenčne množice $P(Q)$

2.2.2 Nedeterministični končni avtomati

Def.: NKA je definiran kot peterka $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, kjer je:

- Q - končna množica stanj
- Σ - vhodna abeceda
- δ - funkcija prehodov $\delta : Q \times \Sigma \to 2^Q$
- q_0 - začetno stanje
- F - množica končnih stanj

Def.: Funkcija ε-closure(q) nam pove, do katerih stanj lahko pridemo iz stanja q po ε prehodih.

$$\varepsilon\text{-closure}(q) = \{q_k \mid \exists q_1, q_2, \ldots q_n \in Q,\ q = q_1 \wedge q_i \in \delta(q_{i-1}, \varepsilon)\}$$

Def.: Posplošena funkcija prehodov $\hat{\delta}$ nam pove, do katerih stanj pridemo po nekem nizu.

$$\hat{\delta}(q, \varepsilon) = \varepsilon\text{-closure}(q)$$

$$\hat{\delta}(q, a) = \delta(q, a)$$

$$\hat{\delta}(q, wa) = \varepsilon\text{-closure}(\{q'' \mid q' \in \hat{\delta}(q, w) \wedge q'' \in \delta(q', a)\})$$

2.2.3 Deterministični končni avtomat

Def.: DKA je definiran kot petorka $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, kjer je:

- Q - končna množica stanj
- Σ - vhodna abeceda
- δ - funkcija prehodov, $\delta : Q \times \Sigma \to Q$
- q_0 - začetno stanje
- F - množica končnih stanj

2.2.4 Jeziki končnih avtomatov

Def.: Jezik εNKA ter NKA je definiran kot:

$$L = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

kjer je $\hat{\delta}(q, w)$ posplošena funkcija prehodov v večih korakih.

Def.: Jezik DKA je definiran kot:

$$L = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

Definicije želijo povedati, da so v jeziku točno tisti nizi, po katerih je iz začetnega stanja mogoče priti do nekega končnega stanja.

2.3 Levo in desno-linearne gramatike

Posebnost linearnih gramatik je v tem, da imajo na desni strani produkcij največ en vmesni simbol, ampak ta model je že nekoliko močnejši od regularnih jezikov (glej **??**), če pa se omejimo le na tiste produkcije, ki imajo ta edini vmesni simbol vedno na skrajno levi strani, ali pa vedno na skrajni desni strani niza, dobimo model, ki opisuje regularne jezike.

Def.: Linearna gramatika je definirana kot četvorček $G = \langle N, T, P, S \rangle$, kjer je:

- N - množica spremenljivk oz. vmesnih simbolov, $N \subseteq \Sigma$
- T - množica znakov oz. končnih simbolov, $T \subset \Sigma,\ N \cap T = \emptyset$
- P - množica produkcij
- S - začetni simbol, $S \in N$

Pri tem je abeceda $\Sigma = N \cup T$ in $N \cap T = \emptyset$.

2.3.1 Produkcije

Def.: Pri levo in desno-linearnih gramatikah, s produkcijami slikamo nek vmesni simbol v niz, ki ima lahko vmesni simbol le na skrajno levi pri levo-linearnih, oz. le na skrajno desni pri desno-linearnih:

- $P \subset N \times ((N \cup \{\varepsilon\})T^*)$ - pri levo-linearnih gramatikah
- $P \subset N \times (T^*(N \cup \{\varepsilon\}))$ - pri desno-linearnih gramatikah

2.3.2 Relacija izpeljave $\Rightarrow$

Def.: Relacija izpeljave pri levo in desno-linearnih gramatikah prek neke produkcije iz P, slika trenutni niz v nov niz, tako, da ima novi niz vmesne simbole lahko le na skrajni levi, pri desno-regularnih pa le na skrajno desni strani, torej:

- $[A \to B\beta]$ - pri levo-linearnih gramatikah
- $[A \to \beta B]$ - pri desno-linearnih gramatikah

Pri tem je $A \in N,\ B \in (N \cup \{\varepsilon\}),\ \beta \in T^*$

Def.: Kadar želimo pokazati, da je mogoče z enim ali več koraki mogoče priti iz enega niza do drugega, to lahko zapišemo s posplošeno relacijo izpeljave $\Rightarrow^*$.

$$\alpha \Rightarrow^* \beta \quad \text{n.t.k.} \quad \alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_k = \beta;\ k > 0$$

2.3.3 Jezik gramatik

2.4 Jezik regularnih jezikov

Def.: Jezik ki ga opisuje poljubni regularni izraz, končni avtomat, levo ali desno-linearna gramatika, je regularni jezik.

Regularni jeziki ne vsebujejo informacije o prejšnjih znakih vhodnega niza in se z njimi ne da opisati poljubnega jezika. (za postopke dokazovanja regularnosti glej 2.7).

Primeri:

Primer 1: $L = \{\}$ - prazen jezik
Primer 2: $L = \{\varepsilon\}$ - jezik, ki vsebuje ε (ni prazen)
Primer 3: $L = \{a, aa, ab\}$ - jezik, ki vsebuje nize "a, aa, ab"
Primer 4: $L = \{0^n 1^n \mid n > 0\}$ - jezik, ki ni regularen (ne moremo si zapomniti poljubnega števila n)

2.5 Operacije, ki ohranjajo regularnost jezikov

Regularnost že po definiciji ohranjajo operacije:

- **Unija** – $L_1 \cup L_2$
- **Stik** – $L_1 \cdot L_2$
- **Iteracija** – L^*

Obstajajo postopki za konstrukcijo, ki kažejo, da regularnost ohranjajo tudi:

- **Presek** – $L_1 \cap L_2$
  Iz avtomatov za jezika L_1 in L_2 zgradimo t.i. produktni avtomat:

  $$\begin{aligned} M_{L_1} &= \{Q_1, \Sigma, \delta_1, q_{1_0}, F_1\} \\ M_{L_2} &= \{Q_2, \Sigma, \delta_2, q_{2_0}, F_2\} \\ M_{L_1} * M_{L_2} &= \{Q_1 \times Q_2, \Sigma, \delta_*, \langle q_{1_0}, q_{2_0}\rangle, F_1 \times F_2\} \end{aligned}$$

  Stanja produktnega avtomata so pari stanj starih dveh avtomatov. Prehode dobimo tako, da hkrati gledamo prehode obeh starih avtomatov in v katere pare stanj nas to pelje. Končna stanja produktnega avtomata so tista, ki so končna v obeh starih avtomatih.

  $$\delta_*(\langle q_1, q_2\rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a)\rangle$$

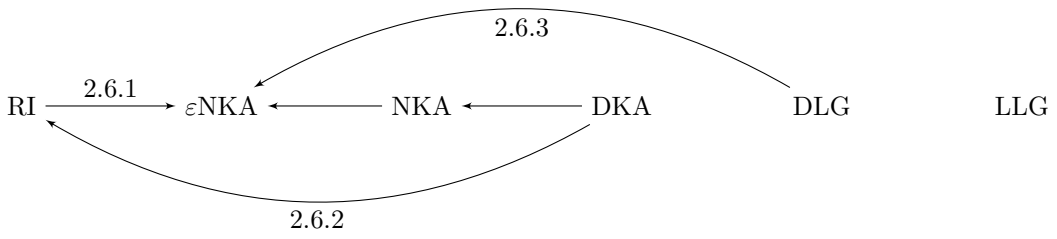
- **Obrat** – L^R
  Avtomat, ki sprejema obrnjene besede lahko konstruiramo tako, da obrnemo vse povezave, ustvarimo novo začetno stanje, ki gre po ε-prehodih v stara končna stanja, staro začetno stanje pa spremenimo v edino končno stanje.
- **Substitucija** – $f(L)$
  Imamo preslikavo $f(a) = R_a$, ki vsak znak $a \in \Sigma_1$ nadomesti z nekim regularnim jezikom R_a (morda v neki drugi abecedi): $R_a \subset \Sigma_2^*$. Ta korak ohranja regularnost – lahko si predstavljamo končni avtomat, ki preprosto vsak pojav simbola a nadomesti z avtomatom za jezik R_a. Če tako nadomestimo vse znake v nizu, dobimo rekurzivno preslikavo za nize: $f(\varepsilon) = \varepsilon$, ter $f(x \cdot a) = f(x) \cdot R_a$. Substitucija nad jeziki pa je nato unija vseh takih preslikav nizov v regularne jezike: $f(L) = \bigcup_{x \in L} R_x$.

Regularnost ohranjajo tudi operacije sestavljene iz zgoraj opisanih, npr.:

- **Razlika** – $L_1 \setminus L_2 = L_1 \cap \overline{L}_2$
- **Komplement** – $\overline{L} = \Sigma^* \setminus L$
- **XOR** – $L_1 \veebar L_2 = (L_1 \cup L_2) \setminus (L_1 \cap L_2)$

2.6 Prevedbe med modeli regularnih jezikov

Regularni izrazi, regularne gramatike in končni avtomati so enako močni modeli in je mogoče pretvarjati med njimi. V tem odseku bomo predstavili naslednje prevedbe:



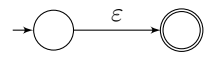
2.6.1 Regularni izraz → Nedeterministični končni avtomat z ε-prehodi

Pretvoriti moramo le osnovne in sestavljene regularne izraze, nato pa ustrezne avotmate samo povezujemo skupaj.

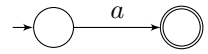
Osnovni izrazi:

- $r = \emptyset$

- $r = \varepsilon$

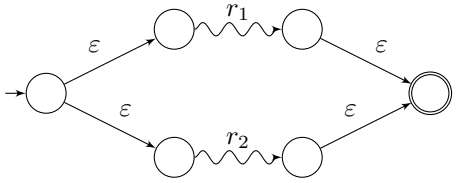


- $r = a$

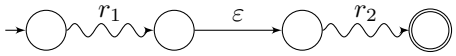


Sestavljeni izrazi:

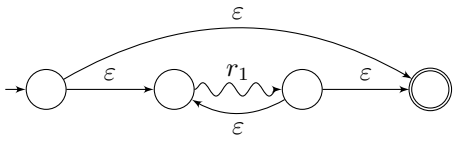
- $r = r_1 + r_2$



- $r = r_1 \cdot r_2$



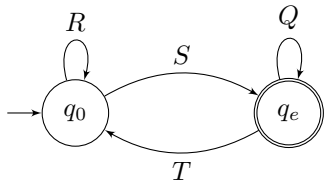
- $r = r_1^*$



2.6.2 Končni avtomat → Regularni izraz

Končni avtomat v regularni izraz prevedemo po metodi z eliminacijo. Pri tej metodi izberemo neko vozlišče za eliminacijo, nato pa njegove sosede povežemo med seboj, tako, da na nove povezave zapišemo regularne izraze, ki opisujejo dogajanje v tistem vozlišču. Eliminacijo ponavljamo, dokler nam v avtomatu ne ostanta le dve stanji, nato pa za končni zapis uporabimo naslednji recept:

Na povezavah avtomata imamo zapisane regularne izraze R, S, Q in T,

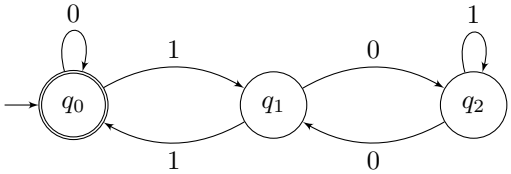


ki jih prepišemo v en sam regularni izraz oblike:

$$(R + SQ^*T)^*SQ^*$$

Primeri:

Primer 1: Zapiši DKA za preverjanje deljivosti s 3 v binarnem sistemu? Zapiši še regularni izraz.



Iz grafa eliminiramo eno stanje in zapišemo regularni izraz:

$$(0 + 1(01^*0)^*1)^*$$

Možna pa je še ena rešitev, če eliminiramo drugo stanje.

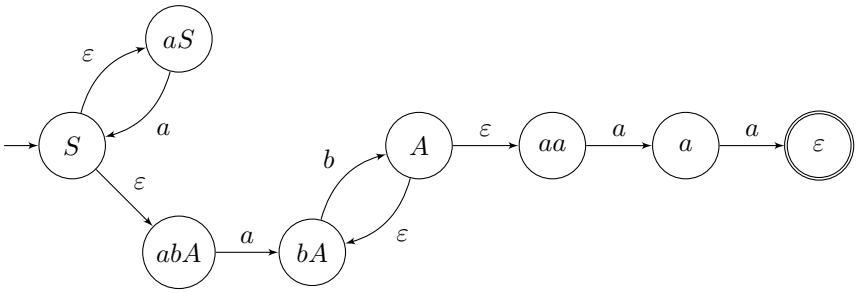
2.6.3 Desno-linearna gramatika → Nedeterministični končni avtomat z ε-prehodi

Vhodno stanje avtomata je začetni simbol gramatike, nato pa stanja označujemo glede na končne in vmesne simbole, ki jih moramo še porabiti. Produkcije predstavljajo ε prehode v avtomatu, preostali prehodi avtomata pa so črke, ki jih generiramo.

Primer: Pretvori podano desno-linearno gramatiko v nedeterministični končni avtomat z ε-prehodi.

$$S \to abA \mid aS$$
$$A \to aa \mid bA$$

Po zgoraj opisanem postopku dobimo:



2.7 Dokazovanje regularnosti jezika

Kadar ugotavljamo, ali je nek jezik regularen, to lahko naredimo na več načinov:

- Pokažemo da je regularen:
  - Jezik skonstruiramo v enem izmed modelov, ki sprejemajo regularne jezike:
    - Končni avtomati
    - Regularni izrazi
    - Levo in desno-linearne gramatike
- Dokažemo da ni regularen:
  - Z uporabo leme o napihovanju za regularne jezike
  - Z uporabo izreka Myhill-Nerode
  - Dokažemo, da jezik ne spada niti v nek širši razred jezikov (glej 3.3)

2.7.1 Lema o napihovanju za regularne jezike

Lemo o napihovanju za regularne jezike uporabljamo za dokazovanje, da nek jezik ne spada v razred regularnih jezikov.

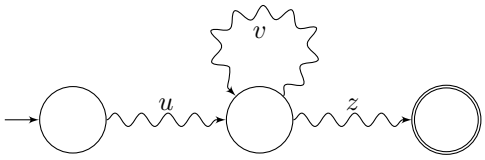
Def.: Za vsak regularni jezik obstaja neka konstanta n, taka, da lahko vsako besedo w iz jezika, daljšo od n, razbijemo na tri dele:

$$w = u\ v\ z$$

Pri čemer velja:

- $|uv| \leq n$
- $|v| > 0$
- $uv^i z \in L,\ \forall i \geq 0$ (napihovanje)

Ker dokazujemo da jezik ni regularen, moramo torej najti neko besedo, za katero pri napihovanju ne ostanemo znotraj jezika. Če nam tega z izbrano besedo ne uspe dokazati, še nismo dokazali da je jezik regularen – edini pravi dokaz tega je konstrukcija jezika v enem izmed modelov, ki opisujejo regularne jezike.



Če zgornjo definicijo pogledamo v kontekstu končnih avtomatov, vidimo, da je n gotovo večji od števila stanj, saj mora za napihovanje v avtomatu obstajati nek cikel, sicer bi bi veljalo $|v| = 0$.

2.7.2 Izrek Myhill-Nerode

2.7.3 Minimizacija končnih avtomatov

Poglavje 3

Kontekstno-neodvisni jeziki

3.1 Kontekstno-neodvisne gramatike

Def.: Kontekstno-neodvisna gramatika je definirana kot četvorček $G = \langle N, T, P, S \rangle$, kjer je:

- N - množica spremenljivk oz. vmesnih simbolov
- T - množica znakov oz. končnih simbolov
- P - množica produkcij
- S - začetni simbol

Def.: Kontekstno-neodvisna gramatika je dvoumna, kadar do nekega končnega niza lahko pridemo po več različnih izpeljavah.

Def.: Kontekstno-neodvisna gramatika je deterministična, kadar za jezik, ki ga gramatika opisuje, obstaja vsaj ena gramatika, ki ni dvoumna. Ni nujno, da je taka gramatika, ki jo imamo - važno je, da taka gramatika obstaja.

3.1.1 Chomskyeva normalna oblika

Def.: Kontekstno-neodvisna gramatika je v Chomskyevi normalni obliki, kadar nima nekoristnih simbolov, ter so vse produkcije naslednjih dveh oblik:

$$A \to a$$
$$A \to BC$$
$$a \in T, \quad B, C \in N$$

3.1.2 Greibachina normalna oblika

Def.: Kontekstno-neodvisna gramatika je v Greibachini normalni obliki, kadar so vse produkcije oblike:

$$A \to a\gamma$$
$$a \in T, \quad \gamma \in N^*$$

3.2 Skladovni avtomati

Def.: Skladovni avtomat je definiran kot sedmerka $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, kjer je:

- Q - končna množica stanj
- Σ - vhodna abeceda
- Γ - skladovna abeceda
- δ - funkcija prehodov, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$

- q_0 - začetno stanje, $q_0 \in Q$
- Z_0 - začetni skladovni simbol, $Z_0 \in \Gamma$
- F - množica končnih stanj

3.2.1 Trenutni opis

Def.: Trenutni opis je trojka $\langle q, w, \gamma \rangle \in Q \times \Sigma^* \times \Gamma^*$, pri čemer je q trenutno stanje, w preostanek vhodnega niza, ter γ trenutna vsebina sklada

3.2.2 Relacija $\vdash$

Def.: Relacija $\vdash$ nas pelje iz enega trenutnega opisa v drugega, če je ta prehod predviden v funkciji prehodov δ:

$$\langle q, aw, Z\gamma \rangle \vdash \langle p, w, \gamma'\gamma \rangle \iff \langle p, \gamma' \rangle \in \delta(q, a, Z)$$

Uporabljamo tudi posplošeno relacijo $\vdash^*$, ki je ubistvu samo ena ali več-kratna uporaba relacije $\vdash$. Pove nam to, da pridemo iz enega trenutnega opisa do drugega, prek enega ali večih prehodov, pod pogojem, da vse vmesne prehode predvideva funkcija prehodov δ.

3.2.3 Jezik skladovnega avtomata

3.3 Dokazovanje kontekstne-neodvisnosti jezika

3.3.1 Lema o napihovanju za kontekstno-neodvisne jezike

3.3.2 Ogdenova lema za kontekstno-neodvisne jezike

Poglavje 4

Turingovi jeziki

4.1 Zgodovina

Leta 1900 je Nemški matematik David Hilbert objavil seznam triindvajsetih nerešenih problemov v matematiki. Eden izmed Hilbertovih problemov (deseti po vrsti), je vprašanje, ali obstaja postopek, po katerem ugotovimo rešljivost poljubne Diofantske enačbe – torej, ali lahko ugotovimo, če ima polinom s celoštevilskimi koeficienti $P(x_1, x_2, \ldots, x_n) = 0$, celoštevilsko rešitev. Kljub temu, da je Emil Post že leta 1944 slutil, da je problem nerešljiv, je to dokončno dokazal rus Jurij Matijaševič šele leta 1970 v svojem doktorskem delu. Med reševanjem problema pa so se matematiki že prej začeli ukvarjati s formalizacijo pojma postopka oz. algoritma. Intuitivna definicija tega se glasi nekako tako:

Def.: Algoritem je zaporedje ukazov, s katerimi se v končnem številu korakov opravi neka naloga.

Pri tem pa ostaja še kar nekaj vprašanj, npr.:

- Kakšni naj bodo ukazi?
  - Osnovni - algoritem ima veliko korakov
  - Kompleksni - prezapleteni ukazi so že sami algoritmi
- Koliko ukazov naj bo?
  - Končno - ali je s končno množico res mogoče rešiti vsako nalogo?
  - Neskončno - kakšen izvajalec ukazov je sposoben izvršiti neskončno različnih ukazov?
- So ukazi zvezni ali diskretni?
- V kakšnem pomnilniku so ukazi shranjeni?
  - Končnem - ali s končnim zaporedjem ukazov res lahko mogoče rešimo vsako nalogo?
  - Neskončnem - ali je to praktično uporaben model pomnilnika?

Nekateri zgodnji poskusi formalizacije pojma algoritma so:

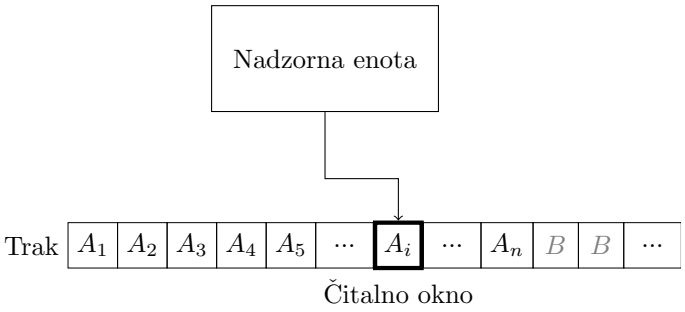
- Rekurzivne funkcije (Kurt Gödel, Stephen Kleene)
- Splošne rekurzivne funkcije (Jacques Herbrand, Kurt Gödel)
- Algoritmi Markova (Andrey Markov, ml.)
- Produkcijski sistem (Emil Post)
- Lambda račun (Alonso Church, 1936)
- Turingov stroj (Alan Turing, 1936)

4.2 Turingovi stroji

Turingov stroj se je uveljavil kot uporaben in preprost model računanja, ki zna izračunati vse kar se izračunati da (pod pogojem, da Church-Turingova teza drži). Alan Turing je svoj stroj izpeljal iz razmišljanja o tem, kako človek rešuje miselne probleme na papir. Pri tem je izbral tri sestavne dele:

- Nadzorno enoto (glava)
- Čitalno okno (roka in vid)
- Trak (papir)

V postopku formalizacije, pa je zaradi večje preprostosti, zahteval še, da je stroj sestavljen iz končno mnogo elementov, ter da deluje v diskretnih korakih.



Def.: Turingov stroj je definiran kot sedmerka $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$, kjer je:

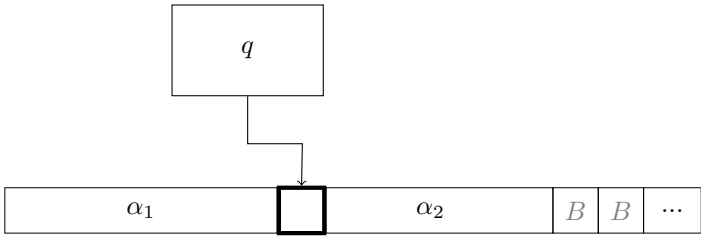
- Q končna množica stanj
- Σ končna množica vhodnih simbolov, $Q \cap \Sigma = \emptyset$
- Γ končna množica tračnih simbolov, $\Sigma \subset \Gamma$
- δ funkcija prehodov: $Q \times \Gamma \to Q \times \Gamma \times \{L, D\}$, kjer L in D označujeta premik levo ali desno
- q_0 začetno stanje, $q_0 \in Q$
- B prazen simbol, $B \in \Gamma$
- F množica končnih stanj, $F \subseteq Q$

Stroj deluje tako, da v vsakem koraku opravi naslednje:

- preide v neko stanje
- zapiše nov simbol v celico, ki je pod oknom
- okno premakne eno celico levo ali desno

4.2.1 Trenutni opis

Def.: $TO = \Gamma^* \times Q \times \Gamma^*$ je množica vseh trenutnih opisov.
Nek trenutni opis $\langle \alpha_1, q, \alpha_2 \rangle$, ali krajše $\alpha_1\ q\ \alpha_2$ opisuje konfiguracijo Turingovega stroja.



Čitalno okno je nad prvim znakom niza α_2, iz tega lahko razberemo:

- če je $\alpha_1 = \varepsilon$, je okno skrajno levo
- če je $\alpha_2 = \varepsilon$, je okno nad B in so naprej sami B-ji

4.2.2 Relacija $\vdash$

Def.: Če sta u, v trenutna opisa iz množice TO, ter v neposredno sledi iz u v enem koraku Turingovega stroja, tedaj pišemo $u \vdash v$.

Naj bo $x_1 \dots x_{i-1}\ q\ x_i \dots x_n$ trenutni opis:

- če je $\delta(q, x_i) = \langle p, Y, D \rangle$:
  $x_1 \dots x_{i-1}\ q\ x_i \dots x_n \vdash x_1 \dots x_{i-1}\ Y\ p\ x_{i+1} \dots x_n$
- če je $\delta(q, x_i) = \langle p, Y, L \rangle$:
  - če je okno na robu $(i = 1)$, se Turingov stroj ustavi, ker je trak na levi omejen.
  - če okno ni na robu $(i > 1)$, potem: $x_1 \dots x_{i-2} x_{i-1}\ q\ x_i \dots x_n \vdash x_1 \dots\ x_{i-2} p\ x_{i-1}\ Y\ x_{i+1} \dots x_n$

Imamo pa tudi posplošeno relacijo $u \vdash^* v$, ki pove, da trenutni opis v sledi iz u v enem ali večih korakih.

Def.: $u \vdash^* v$, če obstaja tako zaporedje $x_i, (i \in [0, 1, \dots, k], k \geq 0)$, da velja $u = x_0, v = x_k$ in $x_0 \vdash x_1 \wedge x_1 \vdash x_2 \wedge \dots \wedge x_{k-1} \vdash x_k$

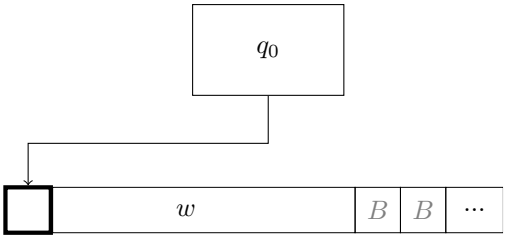
Torej, trenutni opis v sledi iz u, v k korakih Turingovega stroja.

4.3 Jezik Turingovega stroja

Def.: Jezik Turingovega stroja je:

$$L(M) = \{w \mid w \in \Sigma^* \wedge \varepsilon\ q_0\ w \vdash^* \alpha_1\ q_F\ \alpha_2 \wedge \alpha_1, \alpha_2 \in \Gamma^*, q_F \in F\}$$

Z besedami to pomeni, da je jezik Turingovega stroja množica besed, ki če jih damo na vhod stroja, povzročijo, da se ta v končno mnogo korakih znajde v končnem stanju.



Začetna konfiguracija Turingovega stroja.

Def.: Jezik L je Turingov jezik, če obstaja Turingov stroj M, tak, da je $L = L(M)$.

4.3.1 Ugotavljanje pripadnosti besed Turingovemu jeziku

Pri vprašanju ali je neka beseda v jeziku, Turingove jezike ločimo na:

- Odločljive - obstaja algoritem, s katerim se lahko za poljubno besedo odločimo, ali pripada jeziku.
- Neodločljive - v splošnem ni algoritma, ki bi za poljubno vhodno besedo z DA ali NE odgovoril na vprašanje pripadnosti.
  - če je odgovor DA, to ugotovimo v nekem končnem številu korakov.
  - če je odgovor NE, pa ni nujno, da se bo stroj kdaj ustavil.

- - vennov diagram odločljivi jeziki znotraj Turingovih?

Primer: Zapiši Turingov stroj, ki sprejema jezik $L = \{0^n 1^n | n \geq 1\}$
Skica izvajanja stroja:

- $0^n 1^n$ - vhodna beseda
- $X 0^{n-1} 1^n$ - zamenjamo najbolj levo 0 z X

- $X0^{n-1}Y1^{n-1}$ - premaknemo okno desno do najbolj leve 1 in jo zamenjamo z Y
- $XX0^{n-2}Y1^{n-1}$
  $XX0^{n-2}YY1^{n-2}$ - ponovimo in vidimo, da bomo niz sprejeli, če je prave oblike.

Turingov stroj zapišemo kot $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F\rangle$:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, B, X, Y\}$
- $F = \{q_4\}$
- δ bomo definirali s tabelo

Pomen stanj:

- q_0 - začetno stanje in stanje pred zamenjavo 0 z X
- q_1 - premikanje desno do 1
- q_2 - zamenjava 1 z Y in premikanje levo do X
- q_3 - najde X in se premik desno
- q_4 - končno stanje

Tabela prehajanja stanj:

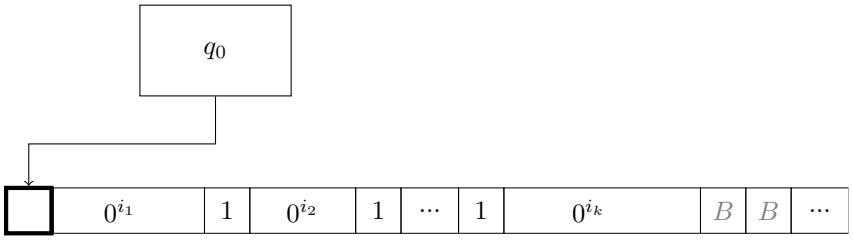
	0	1	X	Y	B
q_0	$\langle q_1, X, D\rangle$	–	–	$\langle q_3, Y, D\rangle$	–
q_1	$\langle q_1, 0, D\rangle$	$\langle q_2, Y, L\rangle$	–	$\langle q_1, Y, D\rangle$	–
q_2	$\langle q_2, 0, D\rangle$	–	$\langle q_0, X, D\rangle$	$\langle q_2, Y, L\rangle$	–
q_3	–	–	–	$\langle q_3, Y, D\rangle$	$\langle q_4, B, D\rangle$
q_4	–	–	–	–	–

Izvajanje stroja s trenutnimi opisi:

$$q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash \dots$$

4.3.2 Turingov stroj kot računalnik funkcij

Imamo Turingov stroj, ki ima na traku neko število ničel, ki predstavljajo pozitivna naravna števila, ločena z enicami:



Recimo, da se stroj po nekem številu korakov ustavi in ima na traku skupino ničel 0^m, na levi in desni strani skupine pa same B-je. S tem je stroj morda izračunal neko funkcijo:

$$f^{(k)} : \mathbb{N}_+^k \to \mathbb{N}_+ \quad \text{oz.} \quad f(i_1, i_2, \dots, i_k) = m$$

Funkcija f ni nujno definirana za vsako k-terico iz $\mathbb{N}_+^k$, torej je parcialna funkcija, kadar pa je definirana povsod, pravimo da je totalna. Stroj se pri nedefiniranih k-tericah pač na neki točki ustavi in pri tem na traku ne pusti le ene skupine ničel, ali pa se sploh ne ustavi. Isti turingov stroj hkrati računa več funkcij: $f^{(1)}, f^{(2)}, \dots f^{(k)}$.

Parcialna rekurzivna funkcija

Def.: Vsaka funkcija $f^{(k)} : \mathbb{N}_+^k \to \mathbb{N}$, ki jo lahko izračuna nek Turingov stroj, je parcialna rekurzivna funkcija. Če je $f^{(k)}$ definirana za vse k-terice, jo imenujemo totalna rekurzivna funkcija (včasih samo rekurzivna funkcija)

Vse običajne aritmetične funkcije so parcialne ali celo totalne rekurzivne funkcije. V primerih si bomo pogledali nekaj primerov, tu pa jih nekaj naštejmo: $m+n,\ m*n,\ n!,\ 2^n,\ \lceil\log(n)\rceil,\ m^n,\ \ldots$.

Primeri:

Primer 1: Ali je $f(m,n) = m+n$ (parcialno) rekurzivna?
Skica stroja, ki računa $m+n$:

- 0^m10^m - vhodna beseda
- $B0^{m-1}10^m$ - izbriši prvo ničlo
- $B0^{m+n}$ - premakni se do 1 in jo zamenjaj z 0

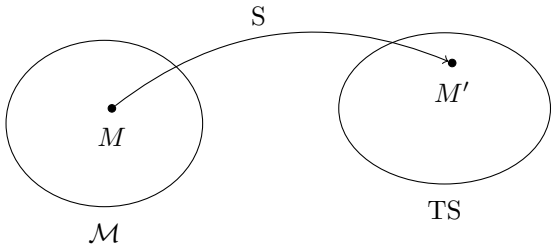
Primer 2: Ali je $f(m,n) = m*n$ (parcialno) rekurzivna?
Skica stroja, ki računa $m*n$:

- 0^m10^n - vhodna beseda
- 0^m10^n1 - premakni se na konec in zapiši 1 (ločnica za rezultat)
- $B0^{m-1}10^n1$ - premakni se na začetek in izbriši 0
- $B0^{m-1}10^m10^n$ - prekopiraj n ničel za ločnico (in ničle)
- $B^m10^m10^{m*n}$ - ponavljaj tadva koraka, dokler ni več ničel pred prvo 1
- $B^{m+n+2}0^{m*n}$ - izbriši del, ki ne spada v rezultat

4.4 Razširitve Turingovega stroja

V tem odseku bomo spoznali nekaj razširitev Turingovega stroja in pokazali da so enako močne kot osnovni model. Poleg tega bomo naredili še pregled alternativnih modelov, za katere se je tudi izkazalo, da so enako močni.

Postopek dokazovanja: Recimo, da je $\mathcal{M}$ razred modelov, za katerega želimo dokazati, da je ekvivalenten razredu Turingovih strojev. Poiskati moramo sistematičen, končen postopek S, ki poljubnemu stroju $M \in \mathcal{M}$ priredi nek Turingov stroj M', ki je sposoben simulirati M.



4.4.1 Nadzorna enota kot pomnilnik

Vsako stanje stroja je sestavljeno iz dveh delov – stanja avtomata in shrambe za tračne znake. Novo množico stanj zapišemo kot $Q = K \times \Gamma$, kjer je K stara množica stanj in Γ tračna abeceda.

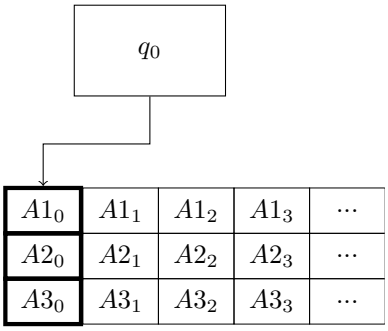
Primer: Sestavi Turingov stroj za razpoznavanje besed, pri katerih se prvi znak ne ponovi:
Stroj $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F\rangle$ zapišemo kot:

- $M = \langle Q, \{0,1\}, \{0,1,B\}, \delta, \langle q_0, B\rangle, B, F\rangle$
- $Q = \{q_0, q_1\} \times \{0,1,B\} = \{\langle q_0,0\rangle, \langle q_0,1\rangle, \langle q_0,B\rangle, \langle q_1,0\rangle, \langle q_1,1\rangle, \langle q_1,B\rangle\}$
- $F = \{\langle q_1, B\rangle\}$
- δ zapišemo kot:
  - Shrani prvi znak besede v stanje stroja:
    $\delta(\langle q_0,B\rangle, 0) = \langle\langle q_1,0\rangle, 0, D\rangle$
    $\delta(\langle q_0,B\rangle, 1) = \langle\langle q_1,1\rangle, 1, D\rangle$

- Premakni okno v desno do prvega znaka, enakega shranjenemu:
  $\delta(\langle q_1, 0\rangle, 1) = \langle\langle q_1, 0\rangle, 1, D\rangle$
  $\delta(\langle q_1, 1\rangle, 0) = \langle\langle q_1, 1\rangle, 0, D\rangle$
- Če prebereš B, pojdi v končno stanje:
  $\delta(\langle q_1, 0\rangle, B) = \langle\langle q_1, B\rangle, karkoli\rangle$
  $\delta(\langle q_1, 1\rangle, B) = \langle\langle q_1, B\rangle, karkoli\rangle$
- Sicer se ustavi. To dosežemo tako, da ne definiramo prehodov:
  $\delta(\langle q_1, 0\rangle, 0)$ in $\delta(\langle q_1, 1\rangle, 1)$

4.4.2 Večsledni trak

Na traku imamo več kot eno sled, kar pomeni, da s traku beremo k-terice tračnih znakov, kar zapišemo kot: $\Gamma = \Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_k$.



Primer: Sestavi Turingov stroj, ki preveri, ali je vhodno število praštevilo.
Skica stroja:

- Trak ima tri sledi:
  - na prvi sledi je vhodno število
  - na drugi sledi je števec, ki na začetku hrani število 2
  - tretjo sled uporabimo za delovno sled, na začetku je lahko prazna.
- Stroj deluje tako:
  - prepiši število s prve sledi na tretjo sled
  - odštevaj število iz druge sledi od števila na tretji sledi
  - če se odštevanje konča z 0, se ustavi (ni praštevilo)
  - sicer število na drugi sledi povečaj za 1
  - če je število na drugi sledi enako tistemu na prvi, sprejmemo (je praštevilo)
  - sicer, ponovimo postopek

4.4.3 Prestavljanje vsebine traku

Recimo, da bi s traku radi vzeli nekaj zaporednih znakov tako, kot da bi jih izrezali iz traku in nato trak zlepili nazaj skupaj, izrezane simbole pa bi si pri tem seveda radi nekako zapomnili. Tudi to metodo realiziramo s pomočjo shrambe za tračne simbole v nadzorni enoti, a moramo pri tem paziti, da je funkcija prehodov pravilno napisana.

- slika "gube"na traku in slika nadzorne enote

Primer: Sestavi Turingov stroj, ki premakne vsebino traku za 2 celici v desno.
Skica stroja:

- Q vsebuje stanja oblike: $\langle q, A_1, A_2\rangle$; $q \in \{q_1, q_2\}$, $A_1, A_2 \in \Gamma$
- Γ poleg ostalih znakov, vsebuje še poseben znak X, ki označuje izpraznjeno celico na traku
- $F = \{q_2\}$

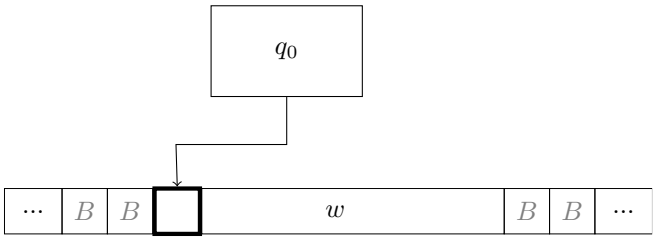
- δ zapišemo kot:
  - Prva koraka – zapomni si in izprazni prvi in drugi znak:
    $\delta(\langle q_1, B, B\rangle, A_1) = \langle\langle q_1, B, A_1\rangle, X, D\rangle$
    $\delta(\langle q_1, B, A_1\rangle, A_2) = \langle\langle q_1, A_1, A_2\rangle, X, D\rangle$
  - Zapomni si nov znak in prvega iz shrambe zapiši na trak:
    $\delta(\langle q_1, A_i, A_{i+1}\rangle, A_{i+2}) = \langle\langle q_1, A_{i+1}, A_{i+2}\rangle, A_i, D\rangle$
  - Zadnja koraka – zapiši vsebino shrambe na trak:
    $\delta(\langle q_1, A_{n-1}, A_n\rangle, B) = \langle\langle q_1, A_n, B\rangle, A_{n-1}, D\rangle$
    $\delta(\langle q_1, A_n, B\rangle, B) = \langle\langle q_2, B, B\rangle, A_n, L\rangle$

4.4.4 Podprogrami

•

4.4.5 Turingov stroj z dvosmernim trakom

Imamo Turingov stroj, ki ima trak neomejen v obe smeri. Vhodna beseda je na začetku napisana nekje na traku, okno pa je na prvem znaku besede.



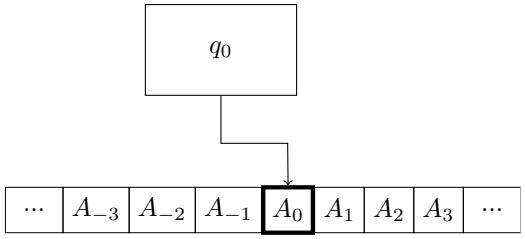
Stroj je definiran skoraj enako kot osnovni Turingov stroj, le funkcija prehodov δ je enostavnejša, saj ni treba skrbeti, kaj se zgodi, če zadanemo levi rob, kot pri običajnem Turingovemu stroju.

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

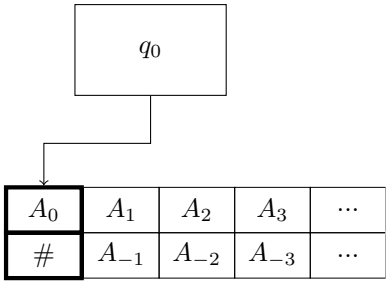
Dokaz: Stroj se lahko vede kot da je omejen na levi. Na začetku izvajanja se premaknemo levo, zapišemo poseben znak, ki nam pomeni konec traku. Nato se premaknemo desno in stroj normalno izvajamo.

Trditev: Turingov stroj z dvosmernim trakom ni močnejši od osnovnega.

Dokaz: Imamo Turingov stroj M z dvosmernim trakom:



Stroju M priredimo dvosledni Turingov stroj M'. Zgornja sled nam bo predstavljala celice od A_0 naprej, spodnja pa vse tiste, levo od A_0:

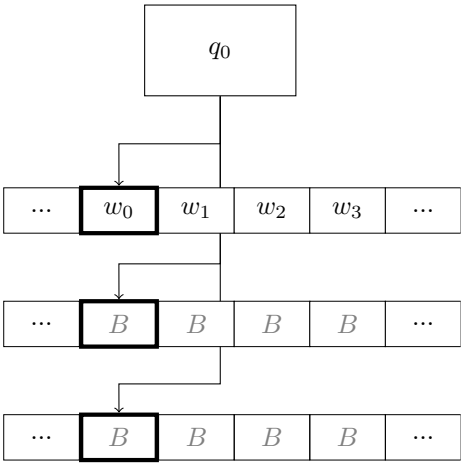


Stroj M' deluje tako:

- naeenkrat dela le z eno sledjo
- ko na drugi sledi vidi #, zamenja aktivno sled
- na zgornji sledi dela enako kot M
- na spodnji se obrne smer premikanja

4.4.6 Večtračni Turingov stroj

Večtračni Turingov stroj ima $k > 1$ trakov, ki so neomejeni v obe smeri. Vsak trak ima svoje okno, ki se lahko na vsakem koraku premakne neodvisno od ostalih. Na prvem traku imamo na začetku ponovno vhodno besedo, okno prvega traku pa na prvem znaku vhodne besede. Ostali trakovi so prazni.



Def.: Korak stroja δ opišemo kot:

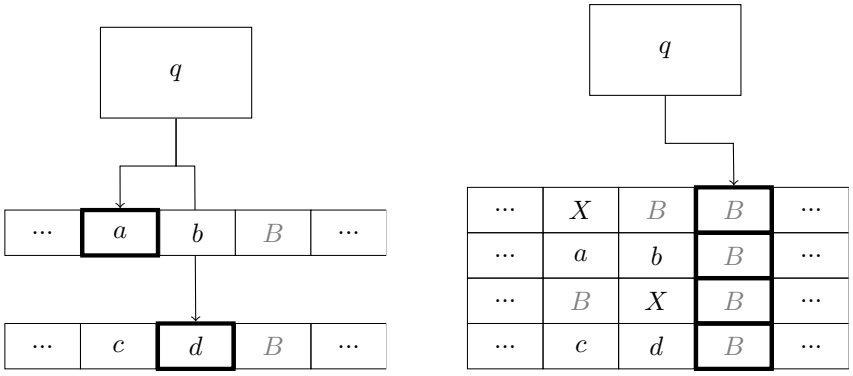
$$\delta = Q \times \Gamma^k \to Q \times (\Gamma \times \{L, D, -\})^k$$

Na vsakem koraku torej iz trenutnega stanja in tračnega simbola na vsakem traku dobimo neko novo stanje ter za vsak trak neodvisno nov simbol in premik.

Trditev: Večtračni Turingov stroj je enako močen kot osnovni model.

Dokaz: **($\Rightarrow$):** Večtračni Turingov stroj uporabi le prvi trak.

($\Leftarrow$): Turingovemu stroju M s k-trakovi priredimo $2k$-sledni v obe strani neskončni Turingov stroj M'. Za vsak trak stroja M imamo tako dve sledi v M' – na zgornji sledi je zapisana oznaka X, ki pove, kje naj bi bilo okno na tem traku stroja M, na spodnji sledi pa je zapisana vsebina tega traku stroja M.



k-tračni stroj M $2k$-sledni stroj M'

Stroj M' torej hrani trenutni položaj na trakovih z dodatno sledjo, ki ima v ustrezni celici zapisan simbol X.

Poleg tega pa pri M' potrebujemo še drugačno nadzorno enoto, ki hrani:

- Stanje stroja
- k tračnih simbolov
- Števec na intervalu $[0, k]$, ki nam pove, koliko simbolov X je še desno od trenutnega položaja okna.

Stroj M' simulira en korak stroja M tako da:

- Okno pomika v desno
- Ko na neki sledi naleti na simbol X:
  - V nadzorno enoto shrani simbol iz naslednje sledi
  - Zmanjša števec za 1.
- Ko števec doseže 0, se začne pomikati v levo
- Ko naleti na simbol X
  - Zamenja tračni simbol na naslednji sledi, enako kot bi naredil stroj M
  - Premakne se levo ali desno, enako kot bi naredil stroj M
  - Poveča števec za 1
- Ko števec doseže k, nadzorna enota preide v novo stanje, enako kot bi stroj M

En korak stroja M torej simuliramo s končno dolgim sprehodom v desno in levo.

4.4.7 Nedeterministični Turingov stroj

Pri nedeterminističnemu Turingovemu stroju, imamo lahko v določeni konfiguraciji več možnosti, kako bo stroj nadaljeval z delom. Za formalni zapis tega je potrebno spremeniti le funkcijo prehodov δ, ki sedaj slika v množico konfiguracij, namesto v eno samo:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L, D, -\}}$$

Def.: Nedeterministični Turingov stroj sprejme besedo natanko tedaj, kadar obstaja končno zaporedje korakov, po katerem pridemo do končnega stanja.

Primer: $\delta(0, a) = \{\langle q_1, b, L\rangle, \langle q_2, c, D\rangle, \langle q_3, a, L\rangle, \langle q_2, B, L\rangle\}$
Stroj bo izbral tisto preslikavo, ki ga vodi k sprejetju vhodne besede. Če to ni mogoče, se bo stroj ali ustavil v nekončnem stanju, ali pa se sploh ne bo ustavil.

Trditev: Nedeterministični Turingov stroj zmore vse kar zmore osnovni Turingov stroj.

Dokaz: Funkcija prehodov δ osnovnega Turingovega stroja je le poseben primer funkcije δ nedeterminističnega Turingovega stroja.

Trditev: Nedeterministični Turingov stroj ni močnejši od osnovnega modela.

Dokaz: Simulacija nedeterminističnega Turingovega stroja z osnovnim:
• slika
Naj bo M nek nedeterministični Turingov stroj in naj ima njegov program v vsaki mmnožici $\delta(q, a)$ največ r možnih potez, kjer je r končno število.
Stroju M priredimo osnovni 3-sledni Turingov stroj M', z naslednjimi lastnostmi:

- na prvi sledi ima abecedo stroja M
- na drugi sledi generira zaporedje navodil besed nad abecedo $\{1, 2, \ldots, r\}$ v leksikografskem vrstnem redu (npr. $r = 3 : 1, 2, 3, 11, 12, 13, \ldots$)
- na tretji sledi simulira stroj M, kot da bi ta izbral poteze skladno s tekočimi navodili

Natančneje stroj M' deluje tako:

- na drugi sledi sestavi novo, naslednje navodilo
- prepiše vhodno besedo s prve na tretjo sled
- na tretji sledi simulira stroj M, kot da bi ta izbral svoje poteze skladno s tekočimi navodili. Pri tem:
  - če M' pride do konca navodila in je tedaj v končnem stanju, vhodno besedo sprejme in konča, kot bi to storil tudi M
  - sicer nadaljuje s prvim korakom ali pa se ustavi v končnem stanju

Trditev: Če M sprejme besedo jo sprejme tudi M'.

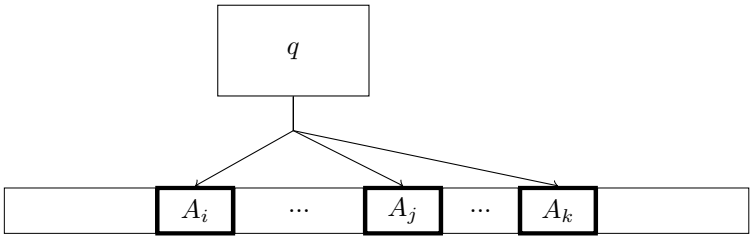
Izrek: Za jezik L obstaja nedeterministični Turingov stroj natanko tedaj, ko za L obstaja osnovni Turingov stroj.

4.4.8 Večdimenzionalni Turingov stroj

Pri tem modelu je trak k-dimenzionalen ($k \geq 2$) in je v vsaki dimenziji neomejen v obe smeri.
Za tridimenzionalni Turingov stroj bi to pomenilo, da je celica kocka in se po vsakem koraku stroja premaknemo v eno izmed šestih sosednjih celic.

4.4.9 Turingov stroj z več okni

Nadzorna enota hkrati lahko bere iz večih oken na istem traku.
Okna so ves čas enako oddaljena med seboj.



4.5 Alternative Turingovemu stroju

Alternative smo našteli že na začetku poglavja. Tu v uvodu na hitro povejmo malo o tem, prek katerih metafor so različni raziskovalci prišli do svojih modelov. Turing je poskušal formalizirati kako človek rešuje problem na papir, ostali pa so se zgledovali po jeziku (npr. kako iz opisa problema dobiti opis rešitve), ali pa kar po funkcijah.

4.5.1 Rekurzivne funkcije (Gödel, Kleene)

Gödel si je zamislil, da bi iz nekaj osnovnih funkcij z nekaj pravili lahko izpeljal kompleksnejše funkcije. Definiral je: ničelno funkcijo, funkcijo naslednik in projekcijo, ter pravili za sestavljanje: kompozicijo in primitivno rekurzijo.

• začetne in drevo ven

Konstrukcija funkcije opisuje tudi mehanični postopek za izračun vrednosti funkcije tako, da je s tem tudi kandidat za formalni opis pojma algoritma.

Kasneje se je izkazalo, da ta pravila ne zadostujejo, saj obstaja npr. Ackermannova funkcija, ki narašča hitreje od vsake funkcije, ki pripada primitivno rekuzivnim:

$$A(m,n) = \begin{cases} n+1 & ; \quad m = 0 \\ A(m-1,1) & ; \quad m > 0 \wedge n = 0 \\ A(m-1, A(m, n-1)) & ; \quad m > 0 \wedge n > 0 \end{cases}$$

Kleene zato doda Gödelovim pravilom še eno pravilo sestavljanja – minimizacijo oz. μ-operacijo.

Def.: Totalne naravne funkcije $f : N^k \to N$, ki jih je moč konstruirati iz treh začetnih funkcij, s končno mnogo uporabami treh pravil sestavljanja imenujemo **rekurzivne funkcije.**

Konstrukcija rekurzivne funkcije f je končno zaporedje $f_1, f_2, \ldots, f_L$ in je $f_L = f$, ter je vsaka funkcija f_i na poti, bodisi začetna, bodisi po pravilih sestavljena iz predhodnic v tem zaporedju.

Začetne funkcije:

- Ničelna funkcija: $\zeta(n) = 0$
- Naslednik: $\sigma(n) = n + 1$
- Projekcija: $\pi_i^k(n_1, n_2, \ldots, n_k) = n_i$

Pravila sestavljanja:

- Kompozicija:

  $g : \mathbb{N}^m \to \mathbb{N}$

  $h_i : \mathbb{N}^n \to \mathbb{N}$

  $f(x_1, x_2, \ldots, x_n) = g(h_1(x_1, x_2, \ldots, x_n), h_2(x_1, x_2, \ldots, x_n), \ldots, h_m(x_1, x_2, \ldots, x_n))$

- Primitivna rekurzija:

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

  $h : \mathbb{N}^{n+2} \to \mathbb{N}$

  $f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n)$

  $f(x_1, x_2, \ldots, x_n, y+1) = h(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y))$

- Minimizacija:

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

  $f(x_1, x_2, \ldots, x_n) = \mu_y(g(x_1, x_2, \ldots, x_n, y)) = z$

  Pri tem je z najmanjše število, za katerega velja $g(x_1, x_2, \ldots, x_n, z) = 0$. Če tak z ne obstaja je funkcija f tam nedefinirana.

Povzetek: Algoritem po Gödel-Kleenu(GK) je ravno konstrukcija rekurzivne funkcije.
Računanje po GK je izračun vrednosti funkcije tako, kot jo narekuje njena konstrukcija.
Funkcija je izračunljiva po GK, če je rekurzivna.

4.5.2 Splošne rekurzivne funkcije (Herbrand, Gödel)

Herbrand je razmišljal o tem, kako poljubno naravno funkcijo definirati s sistemom enačb.

Gödelu je pisal: *„Recimo, da je f neznana funkcija, $g_1, g_2, \ldots, g_m$ pa znane. Funkcije f in g_i poljubno vstavljamo kot argumente v druge funkcije, nato pa nekatere dobljene izraze izenačimo. Če ima dobljeni sistem natanko eno rešitev za funkcijo f, potem je f rekurzivna."*

Ta pa je dodal še dve dodatni zahtevi:

- Na levi strani enačb se funkcija f ne sme pojaviti kot eden izmed argumentov f.
- Funkcija f naj bo totalna.

Def.: Če se funkcijo f da zapisati na zgoraj zapisani način, je **splošno rekurzivna**.
Sistem označimo z $\varepsilon(f)$ in mu pravimo **standardni sistem**.

Pravila za računanje vrednosti $f(n_1, n_2, \ldots, n_k)$ iz $\varepsilon(f)$:

- V enačbi vse pojave neke spremenljivke zamenjamo z istim naravnim številom
- V enačbi pojave funkcije zamenjamo z njeno vrednostjo

Povzetek: Algoritem po Herbrant-Gödlu(HG) je $\varepsilon(f)$.
Računanje po HG je izračun vrednosti funkcije $f(n_1, \ldots, n_k)$ iz $\varepsilon(f)$.
Funkcija je izračunljiva po HG, če je splošno rekurzivna.

4.5.3 Netipizirani λ-račun (Alonso Church)

Pri λ-računu imamo začetni izraz oz. začetni λ-term, ki opisuje neko funkcijo f in njene argumente $n_1, \ldots, n_k$. Cilj računanja je preoblikovati začetni λ-term v tak končni λ-term, ki bo opisoval ravno vrednost funkcije $f(n_1, n_2, \ldots, n_k)$.

Preoblikovanje dosežemo z uporabo redukcij:

- α-redukcija preimenuje spremenljivko v λ-termu,
- β-redukcija uporabi neko funkcijo nad njenimi argumenti.

$t_0 \to t_1 \to \cdots \to t_k = f(n_1, n_2, \ldots, n_k)$

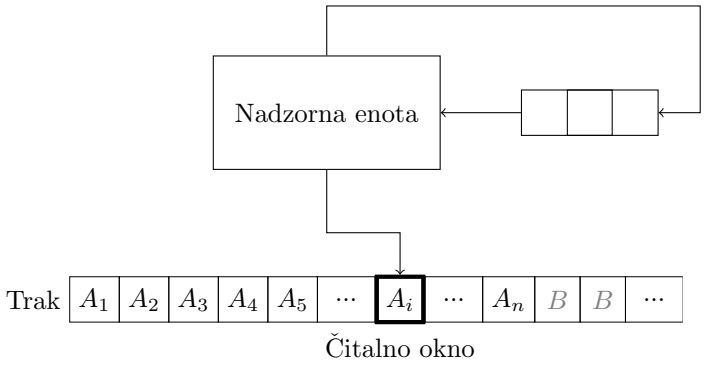
Def.: funkcija, ki jo je moč predstaviti in računati v *lambda*-računu, je **λ-definabilna**

Povzetek: Algoritem po Churchu je λ-term.
Računanje po Churchu je preoblikovanje začetnega λ-terma v končni λ-term z redukcijami.
Funkcija je izračunljiva po Churchu, če je λ-definabilna.

4.5.4 Postov Stroj (Emil Post)

Model je podoben Turingovemu stroju, z naslednjimi spremembami:

- Stroj s traku lahko le bere znake
- Uporablja posebno vrsto za znake



Korak stroja: Iz celice pod oknom in iz začetka vrste prebere po en znak. Na podlagi teh dveh znakov in stanja premakne okno, nov znak da na konec vrste in preide v novo stanje.

Povzetek: Algoritem po Postu je program Postovega stroja.
Računanje po Postu je izvajanje programa Postovega stroja.
Funkcija je izračunljiva po Postu, če njeno vrednost lahko izračuna Postov stroj.

4.5.5 Algoritmi Markova (Andrej Markov ml.)

Imamo abecedo Σ, ter končno zaporedje produkcij:

$$x_1 \to y_1$$
$$x_2 \to y_2$$
$$\dots$$
$$x_n \to y_n$$
$$x, y \in \Sigma^*$$

Produkcija preoblikuje besedo tako, da v besedi nadomesti skrajno levi pojav x_i z y_i. Algoritem je zaporedje korakov, ki postopno preoblikuje začetno besedo v končno. V vsakem koraku se trenutna beseda preoblikuje z najbolj levo možno produkcijo.

Povzetek: Algoritem po Markovu je gramatika.
Računanje po Markovu je preoblikovanje vhodne besede z dano gramatiko.
Funkcija je izračunljiva po Markovu, če njeno vrednost računa kaka gramatika

4.5.6 Povzetek modelov

Vsem omenjenim modelom je skupno, da so razumno zmogljivi:

- Končno mnogo ukazov
- Ukaz se izvede v končnem času
- Ukaz opravi končno mnogo računanja
- Učinek ukaza je predvidljiv

Skupno jim je tudi to, da je pomnilnik potencialno neskončen in brez omejitev dostopa.

4.5.7 Church-Turingova teza

Church je postavil domnevo: *„algoritem"* $\iff$ algoritem po Churchu (λ-račun)
Turing je postavil domnevo: *„algoritem"* $\iff$ algoritem po Turingu (Turingovi stroji)

Church-Turingova teza pravi, da je *„algoritem"* opis postopka reševanja nekega problema na Turingovem stroju, ali na kateremkoli izmed ekvivalentnih modelov.

Neurejen del zapiskov... halp!

4.6 Težave s totalnimi funkcijami

Kako bi v splošnem dokazali, da je neka funkcija $f : \mathbb{N}^n \to \mathbb{N}$ totalna?
V najslabšem primeru, je treba za vsak $x \in A$ pogledati, ali je funkcija za ta x definirana (pišemo: $f(x) \uparrow$)
Če je množica A neskončna, potem tudi ta način seveda ni končen. Pojem izračunljivosti funkcije se tako opira na algoritmično težko določljiv pojem totalnosti.

• slika

Po Church-Turingovi tezi je izračunljiva ... rekurziva funkcija

- Moč $N^k : N$ je c (moč realnih števil)
- Rekurzivnih funkcij je $\aleph_0$ (števno mnogo)
- Vsako definira program Turingovega stroja, teh programov je tudi števno mnogo.

Trditev: Obstaja izračunljiva funkcija, ki ni rekurzivna.

Dokaz: Definicije funkcij so končna zaporedja.
Zaporedja uredimo po dolžini, enako dolga pa leksikografsko
$\Rightarrow$ lahko govorimo o prvem, drugem, ..., n-tem programu Turingovega stroja. Zato lahko tudi prvi, drugi, ..., n-ti rekurzivni funkciji.
Označimo n-to rekurzivno funkcijo s f_n
Definirajmo .....
$g(n, a_1, \ldots, a_k) \stackrel{def}{=} .f_n(a_1, a_2, \ldots, a_k) + 1$, kjer $a_i \in \mathbb{N}$
Funkcija je izračunljiva
Algoritem je ....
....
ali je g tudi rekurzivna
ali obstaja program Turingovega stroja, ki jo izračuna.

predpostavka: g je rekurzivna
Tedaj obstaja nek naravni m, da je $f_m = g$.
Poglejmo vrednost $g(m, m, \ldots, m)$
iz (*) sledi $g(m, m, \ldots, m) = f_m(m, m, \ldots, m)$
iz (**) sledi $f_m(m, m, \ldots, m) = g(m, m, \ldots, m)$
iz tega sledi $f_m(m, m, \ldots, m) = f_m(m, m, \ldots, m) + 1$
in pridemo v protislovje – g ni rekurzivna.
torej obstaja funkcija, ki je izračunljiva in ni rekurzivna.

kaj sedaj?
Ali naj dodajo začetne funkcije, ali pravila za dodajanje, če gledamo Godlov model
Ne, pridemo v isto protislovje.

Ali je to ovrglo Church-Turingovo tezo?
Ne. Ugotovili so, da se je treba odpovedati zahtevi, da so funkcije le totalne. Dopustiti je treba tudi sestavljene parcialne.
Takrat *** ni več nujno protislovje, saj je lahko $f_m(mmmm) \uparrow$.

Povzetek: Funkcija je *„izračunljiva"* $\Longleftrightarrow$ nek Turingov stroj računa f tam, kjer je definirana

4.7 Univerzalni Turingov stroj

Ideja: Turingove stroje bi radi oštevilčili.
Če bi imel vsak Turingov stroj svoj indeks, bi nek drug Turingov stroj lahko računal z drugimi

stroji oz. z njihovimi ideksi.

Kodiranje Turingovih strojev

Kako poljuben Turingov stroj zakodirati z abecedo $\{0,1\}$?
Naj bo $T = \langle Q, \Sigma, \Gamma, \delta, q_1, B_1, q_f \rangle$ poljuben Turingov stroj, kjer je $\delta(q_i, a_j) = \langle q_k, a_l, S_m \rangle$.

Zadostuje, da zakodiramo posamezne ukaze programa δ:

$$K = 0^i 10^j 10^k 10^l 10^m$$

Ko zakodiramo vseh r ukazov programa δ dobimo kode $K_1, K_2, \ldots, K_r$ iz katerih bomo sestavili kodo Turingovega stroja:

$$\langle T \rangle = 111 K_1 11 K_2 11 \ldots 11 K_r 111$$

Na $\langle T \rangle$ lahko gledamo kot na dvojiški zapis nekega naravnega števila in indeks Turingovega stroja T. Nekatera naravna števila pa niso indeksi Turingovega stroja, zato se dogovorimo: če naravno število nima oblike $\langle T \rangle$, rečemo, da je indeks praznega Turingovega stroja – njegova δ je povsod nedefnirana, torej se takoj ustavi in ne sprejme nobene besede.

Posledica: Vsako naravno število, je indeks natanko enega Turingovega stroja. Obratno pa ne velja, saj ima isti Turingov stroj lahko več indeksov (npr. če stroju dodajamo nepotrebne ukaze).

Ideja za univerzalni Turingov stroj

Trditev: Obstaja Turingov stroj, ki izračuna vse kar izračuna katerikoli drug Turingov stroj.

Dokaz: Stroj si bomo zamislili le v grobem in intuitivno zapisali njegov program. Tu se skličemo na Church-Turingovo tezo, ki nam zagotovi obstoj nekega konkretnega Turingovega stroja, ki opravlja to nalogo.

Trakovi: **Vhodni trak** – vsebuje vhodno besedo sestavljeno iz dveh delov:
- Kodo $\langle T \rangle$ poljubnega Turingovega stroja.
- Poljubno besedo $w \in \Sigma^*$

Delovni trak – sprva prazen. Stroj U .....
Pomožni trak – sprva prazen. Stroj U ga bo uporabljal za zapis tekočega stanja stroja T in za primerjanje tega stanja s končnim stanjem stroja T.

Program stroja deluje nekako tako:

- Preveri, ali je vhod oblike $\langle T,\ w \rangle$, kjer je T koda nekega Turingovega stroja. Če ni, se U ustavi (tudi T bi se)
- Iz $\langle T \rangle$ preberi kodo končnega stanja $\langle q_F \rangle$, stroja T. in napisi $\langle q_1, q_f \rangle$ na tretji trak.
- Prepiši w na delovni trak in postavi okno na začetek
- Denimo, da je na pomožnem traku nek par $\langle q_i, q_f \rangle$ in da je v delovnem oknu znak a. Če je $q_i = q_f$, se stroj ustavi (tudi T bi se)
- Na prvem traku poišči v $\langle T \rangle$ kodo ukaza, ki se začne z $\delta(q_i, a) = \ldots$
- Če je ne najdemo se U ustavi (tak T bi se ustavil)
- Denimo, da najdena koda opisuje ukaz $\delta(q_i, a) = \langle q_, b, S \rangle$ ... na drugi trak zapiši b v ... premik v smeri S
- Na pomožni trak namesto $\langle q_i, q_f \rangle$ vpiši $\langle q_j, q_f \rangle$, goto 4

• slika: trije trakovi

$$U = \langle Q_U, \Sigma_U, \Gamma_U, \delta_{U1}, q_{U1}, B, q_{Uf} \rangle$$

4.7.1 Pravi stroji, ki so univerzalni

Naredimo nekaj sprememb: Vsaka celica je neposredno dosegljiva prek naslova Program naj bo na traku, ne v glavi

4.8 Reševanje računskih problemov

Računske probleme lahko delimo na:

- **Odločitvene** – sprašujejo po odgovoru z DA ali NE
- **Iskalne** – sprašujejo po elementih množice, ki imajo neko lastnost
- **Preštevalne** – sprašujejo po številu elementov množice, ki imajo neko lastnost
- **Naštevalne** – zahtevajo generiranje vseh elementov množice, ki imajo neko lastnost

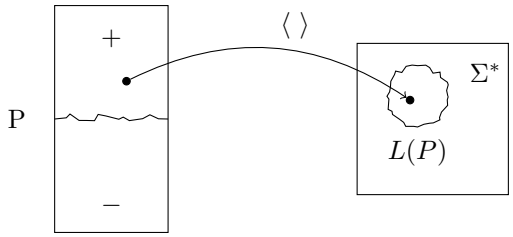
Odločitevni problemi so najenostavnejši, zato se jim bomo tu bolj posvetili.

4.8.1 Jezik odločitvenih problemov

Problemu priredimo formalen jezik (množico besed nad neko abecedo).

Odločitveni problem označujemo s P (npr. $P =$ *,,Ali je število n praštevilo?"*). Ko v problem vstavimo namesto n nek konkretni podatek dobimo nalogo, ki jo označimo s p. (npr. $p =$ *,,Ali je 19871 praštevilo?"*). Vsaka odločitvena naloga je pozitivna ali negativna (ima odgovor DA ali NE). Prek kodirne funkcije $\langle\ \rangle$ dobi vsaka naloga p svojo kodo $\langle p \rangle \in \Sigma^*$, ki jo razume nek Turingov stroj.
Če definiramo kodirno funkcijo kot $\langle\ \rangle : P \to \Sigma^*$ in zahtevamo, da je injektivna $((p \neq p' \Rightarrow \langle p \rangle \neq \langle p' \rangle)$, potem lahko jezik problema P definiramo kot:

$$L(P) \stackrel{def}{=} \{\langle p \rangle \in \Sigma^* \mid p \text{ je pozitivna naloga problema } P\}$$



Naloga $p \in P$ je torej pozitivna, če je koda naloge v jeziku problema. Računanje odgovora na nalogo p tako lahko zamenjamo z ugotavljanjem, ali je beseda $\langle p \rangle$ v jeziku $L(P)$.

Pravimo, da je problem P:

- **Odločljiv**, če je njegov jezik $L(P)$ rekurziven.
- **Neodločljiv**, če njegov jezik $L(P)$ ni rekurziven.
- **Poldločljiv**, če je njegov jezik $L(P)$ Turingov (rekurzivno prešteven).

Neurejen del zapiskov... halp!

4.8.2 Neodločljivi problemi

Težave delajo stroji, ki se ne ustavijo. Ali bi lahko za poljuben par stroja in besede, $\langle T, w\rangle$, preverili, ali se T(w) ustavi.

Def.: Problem ustavitve: $P_{\mathcal{K}_0}$ je odločitveni problem *,,Ali se Turingov stroj T pri besedi w ustavi?“'*

Trditev: Problem ustavitve ni odločljiv.

Dokaz: Jezik problema je: $\mathcal{K}_0 = \{\langle T, w\rangle \mid T \text{ se pri w ustavi}\}$ Nekaj časa nas bo zanimal jezik $\mathcal{K} \subseteq \mathcal{K}_0$, ki ga dobimo iz $\mathcal{K}_0$, če za w vzamemeo kar kodo stroja, torej $\langle T\rangle$. $\mathcal{K} = \{T, T \mid T \text{ se pri } \langle T\rangle \text{ ustavi. }\}$ Temu jeziku pripada odločitveni problem $P_{\mathcal{K}}$ - *,,Ali se Turingov stroj ustavi nad lastno kodo?”*.

Lema: Problem $P_{\mathcal{K}}$ ni odločljiv.

Dokaz: Predpostavimo, da je množica $\mathcal{K}$ rekurzivna (torej, je njen problem odločljiv). Potem gotovo obstaja Turingov stroj R, ki ta jezik razpoznava. $\exists TS : R_{\mathcal{K}}$, ki za poljuben T odgovori $R(\langle T, T\langle)$,

kjer je $R(\langle T, T\rangle) = \begin{cases} DA & ; \text{ če se } T \text{ ustavi nad } \langle T\rangle \\ NE & ; \text{ če se ne ustavi} \end{cases}$

Sestavimo nov Turingov stroj N s pomočjo domnevnega $R_{\mathcal{K}}$: • škatla N. vhod $\langle T\rangle$,v škatli naredi $\langle T, T\rangle$, preda kosmati (ne vemo če obstaja) škatli $R_{\mathcal{K}}$ z izhodi DA/NE... če DA ga vprašamo šeenkrat isto.. če ne, ne speljemo na izhod N

Trditev: domnevni $R_{\mathcal{K}}$ napove prihodnost stroja T, N pa bo razgalil nesposobnost $R_{\mathcal{K}}$ pri tem napovedovanju na primeru T=N

Dokaz: • $\langle N\rangle$ gre v $\langle N, N\rangle$ Domnevni $R_{\mathcal{K}}$ ne bi rešil naloge p - *,,Ali se Turingov stroj N pri vhodu $\langle N\rangle$ ustavi?”* Sklep: K ni rekurziven jezik, zato $R_{\mathcal{K}}$ ni odločljiv problem.

Tudi $P_{\mathcal{K}_0}$ ni odločljiv, sicer bi bil zagotovo odločljiv tudi podproblem $P_{\mathcal{K}}$. Sklep: Obstaja odločittveni problem, za katerega ni Turingovega stroja, ki bi ga vedno rešil.Turingov stroj je po Church-Turingovi tezi algoritem, torej za nekatere probleme ni algoritma, ki bi ga rešil.

4.8.3 Primeri nerešljivih problemov

• napol presekana množica - odločljivi/neodločljivi

Problemi o Turingovih strojih

Problem garača (angl. busy beaver): Vzamemo Turingove stroje, ki imajo n nekončnih stanj. Takih strojev je končno mnogo. Garač je turingov stroj z natanko n nekončnimi stanji, ki začne s praznim trakom in zapiše največ enic (od vseh takih strojev), preden se ustavi. *,,Ali je Turingov stroj T, garač?”*.

Še nekaj primerov za ustavitev Turingovega stroja: *,,Ali se T pri w ustavi?”*, *,,Ali se T pri praznem vhodu ustavi?”*, *,,Ali se T pri vsakem vhodu ustavi?”*, *,,Ali se dva stroja ustavita pri istih vhodih?”*

Razpoznavanje jezikov:
,,Ali Turingov stroj T razpoznava rekurziven jezik?”,
,,Ali Turingov stroj T razpoznava regularen jezik?”

Problemi o algoritmih in programih

P je problem, $\langle P\rangle$ opis problema.
A je algoritem, $\langle A\rangle$ označuje program.
,,Ali program $\langle A\rangle$ rešuje problem $\langle P\rangle$?”

Pravimo, da je program $\langle A'\rangle$ ekvivalenten $\langle A\rangle$, če da za vsak vhod enako vrednost kot $\langle A\rangle$.
,,Ali obstaja program $\langle A'\rangle$, ki je krajši in hkrati ekvivalenten $\langle A\rangle$?”

Problemi o izračunljivih funkcijah

$\varphi : A \to B$ je izračunljiva funkcija.
„Ali ima φ neprazno domeno?",
„Ali ima φ neskončno domeno?",
„Ali ima φ končno domeno?",
„Ali je φ surjektivna?",
„Ali je φ totalna?"

Problemi iz matematike

„Ali ima Diofantska enačba $p(x_1, x_2, \ldots, x_n)$ celoštevilsko rešitev?"

Dana je množica matrik $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$. Vse matrike so reda $n \times n$ in imajo celoštevilske koeficiente.
„Ali je mogoče zmnožiti matrike M_i tako, da dobimo ničelno matriko?"

Problemi o gramatikah in jezikih

„Ali je dana kontekstno-neodvisna gramatika dvoumna?",
„Ali sta dve kontekstno-neodvisni gramatiki ekvivalentni?"

Problemi matematične logike

„Ali so vse formule predikatnega računa prvega reda odločljive?"

Razni problemi

Enakost besed: E naj bo končna množica enačb nad besedami. npr.
1. bc=cba
2. ba=abc
3. ca=ac

Od tu lahko izpeljemo enakost abcc=cacacbaa
„Ali iz E sledi $u = v$?"

Tlakovanje: Imamo končno množico tlakovcev $T = \{$ • križ-kraž kvadrati z različno pobarvanimi četrtinami $\}$. Radi bi tlakovali ravnino razdeljeno na kvadrate, tako, da se sosednja tlakovca vedno ujemata v barvi.
„Ali lahko tlakujemo vsak poligon v $\mathbb{Z}^2$?"
„Ali lahko tlakujemo pot v ravini od točke A do C, s tem, da ne obiščemo točke B?"

4.8.4 Dokazovanje neodločljivosti problemov

Neodločljivost problema lahko dokazujemo z naslednjimi metodami:

- **Neposredno dokazovanje**
- **Dokazovanje s prevedbo**
- **Dokazovanje z Riceovim izrekom**

Dokazovanje s prevedbo

Naj bo Q odločitveni problem, za katerega sumimo, da je neodločljiv.Dokazujemo v treh korakih:

- Izberemo nek problem P, ki ni odločljiv.
- Dokažemo, da velja: če bi bil Q odločljiv, bi bil tudi P odločljiv.
- Če prejšnji korak uspe lahko sklenemo da Q ni odločljiv.

V koraku 2 z domnevnim Turingovim strojem R_Q, je treba sestaviti Turingov stroj R_P, ki se vedno ustavi.

Dokazovanje z Riceovim izrekom

- Za funkcije - Vsaka netrivialna lastnost izračunljivih funkcij je neodločljiva
- Za jezike - Vsaka netrivialna lastnost Turingovih jezikov je tudi neodločljiva

Podrobneje: Naj bo L neka lastnost za funkcije in naj bo φ poljubna izračunljiva funkcija.

P_L*eq,,Ali ima* φ *lastnost* L*?"*

Def.: Funkcijska lastnost L je odločljiva, če je P_L odločljiv.

?: Kakšne smiselne lastnosti pa nas zanimajo?
Samo take lastnosti L, da so neodvisne od ...(=lg,TS,...), s ..... njihove vrednosti. npr. *,,L eq ,,bodi totalna""*
Def.: Lastnost funkcij L je trivialna, če jo ima bodisi vsaka ali pa nobena funkcija.

Riceov prvi izrek za funkcije

Lastnost izračunljivih funkcij je odločljiva $\iff$ je ta lastnost trivialna.

Slaba novica: Problem P_L bo odločljiv le za trivialne funkcijske lastnosti, ki navadno niso zanimive za obravnavo.

Dobra novica: Ugotavljanje odločljivosti P_L je enostaven, ker ga lahko nadomestimo z ugotavljanjem ali je ustrezna lastnost L trivialna.

Primer: P_L eq *,,Ali je* φ *totalna?"*
Preverimo:
Ali je lastnost smiselna? DA.
Ali je lastnost trivialna? NE.
Torej, P_L ni odločljiv problem.

Primer: Ali ima φ neprazno domeno?
Ali ima φ končno domeno?
Ali ima φ neskončno domeno?
Ali ima φ pri vsaj enem argumentu vrednost 3?
Ali je φ surjektivna/injektivna?
Ali je φ definirana pri 2?
Ali je φ enaka drugi funkciji ψ?

Dokaz: Lastnost funkcij ... od ,,,
Naj bo $L = \{\psi | \psi$ ima lastnost $L\}$.
Problem P_L je sedaj vprašanje: $\varphi = ?\mathcal{L}$.
Naj bo $\Theta\mathcal{L} = \{$ vsi programi(indeksi Turingovih strojev), ki računajo funkcijo iz$\mathcal{L}\} = \cup_{\psi \in \mathcal{L}} \Theta\psi$.
Ker je po predpostavki lastnost L neodvisna od načina računanja funkcije, so v množici $\Theta\mathcal{L}$ bodisi vsi programi, ki računajo neko funkcijo φ, ali pa noben.
Zato je vprašanje ali je $\varphi \in ?\mathcal{L}$ ekvivalentno vprašanju: $x \in ?\Theta\mathcal{L}$, kjer je x indeks poljubnega Turingovega stroja, ki računa funkcijo φ.
Torej: Problem $\varphi \in ?\mathcal{L}$ oz. P_L je odločljiv, $\iff$ jezik $\Theta\mathcal{L}$ odločljiv.

Rice je dokazal, da je $\Theta\mathcal{L}$ odločljiv $\iff \Theta\mathcal{L}$, bodisi $\emptyset$, ali pa $\mathbb{N}$.

Dokaz: $\emptyset$ in $\mathbb{N}$ sta obe odločljivi, torej je taka tudi $\Theta\mathcal{L}$, če je enaka kateremu od njiju.
Zato vzamemo, da $\Theta\mathcal{L}$ ni niti $\emptyset$, niti $\mathbb{N}$. Sledi: $\exists a, b : a \in \Theta\mathcal{L} \wedge b \in \overline{\Theta\mathcal{L}}$ Naj bo φ_n povsod definirana funkcija.
..Ker je n naraven, je bodisi v $\Theta\mathcal{L}$, bodisi izven.
Predpostavimo: $n \in \overline{\theta\mathcal{L}}$
..Naj bo f funkcija, ki vsak $x \in \mathcal{K}$ preslika v a, vsak $x \in \overline{\mathcal{K}}$ pa v n.
Potem:$x \in \mathcal{K} \iff f(x) \in \Theta\mathcal{L}$To pomeni: če bi bil $\Theta\mathcal{L}$ odločljiv, je tudi $\mathcal{K}$, torej $\Theta\mathcal{L}$ ni odločljiv.Predpostavimo še: $n \in \Theta\mathcal{L}$ in dokažemo na analogen način.

Riceov prvi izrek za jezike

L je lastnost, smiselna za jezike, $\mathcal{M}$ pa naj bo poljuben Turingov jezik.
Problem: Q eq *„Ali ima Turingov jezik $\mathcal{M}$ lastnost L?"*.

Def.: L je odločlvijv, če ke QL odločljiv problem

Def.: L je trivialna, če jo ima bodisi vsak Turingov jezik, ali pa noben

Riceov izrek: Lastnost Turingovega jezika je odločljiva $\iff$ lastnost je trivialna.

Primer: *„Ali Turingov jezik $\mathcal{M}$ vsebuje npr. besedo aabca?"*
„Ali je Turingov jezik $\mathcal{M}$ enak Turingovemu jeziku $\mathcal{N}$?"
„Ali Turingov jezik $\mathcal{M}$ regularen?"

4.9 Povzetek

V prejšnjem poglavjo pri izračunljivosti nas je predvsem zanimalo:

- Kaj se da/ne da izračunati
- Kaj lahko izračuna določen model računanja
- Ni nas zanimala časovna in prostorska zatevnost računanja

Pri tem smo ugotovili:

- Obstajajo neizračunljivi(neodločljivi) problemi
- Za te probleme ni splošnega algoritma, ki bi se vedno ustavil
- Enako bi obveljalo če bi imeli neomejeno prostora/časa

• slika elipsa prerezana na pol – odločljivi/neodločljivi(našteti... + načini dokazovanja). jih je neskončno mnogo

4.9.1 Turingov stroj s prerokom

Post, Kleene in drugi so se nato začeli spraševai, kaj bi pomenilo, če bi našli rešitev za enega izmed neodločljivih problemov. Uvedli so Turingov stroj s prerokom, ki zna povedati ali je odgovor na odločitveni problem DA, NE, ali pa NEVEM. Pojavljata se dve implementaciji:

- Dodamo tri dodatna stanja $q_D A, q_N E$ in $q_?$, v katera stroj preide po branju vhodne besede.
- Imamo trak na katerem so zapisane vse rešitve za nek problem.

4.9.2 Polodločljivi jeziki glede na nek jezik

Def.: Naj bo $\mathcal{M}$ nek jezik. Pravimo, da je $\mathcal{M}$ glede na L:

- Polodločljiv: če je $M = L(T^{\mathcal{L}})$ za nek $T^{\mathcal{L}}$
- Rekurziven: če je $M = L(T^{\mathcal{L}})$ za nek $T^{\mathcal{L}}$ in se vedno ustavi.

Tu lahko začnemo odgovarjati na vprašanje kaj bi pomenilo, če bi bil nek problem rešljiv.
Razredi relativne izračunljivosti:

- $\Sigma_1^0 \stackrel{def}{=} \{M \mid \text{M je r.e}\}$
- $\Sigma_{n+1}^0 \stackrel{def}{=} \{M \mid \text{M je r.e. glede na nek jezik L iz prejšnjega razreda } \Sigma_n^0\}$
- $\Delta_1^0 \stackrel{def}{=} \{M \mid \text{M je rekurzivnen}\}$
- $\Delta_{n+1}^0 \stackrel{def}{=} .\{M \mid \text{M je rekurziven glede na nek jezik L iz prejšnjega razreda} \Sigma_n^0\}$

- $\Pi_n^0 \stackrel{def}{=} \{M \mid \overline{M} \in \Sigma_n^0\}$

Ugotovitve:
$\mathcal{M} \in \Sigma_n^0 \iff M = \{\exists x_1 \forall x_2 \exists x_3 \ldots Quant x_n R(w, x_1, \ldots, x_n)\}$
$\mathcal{M} \in \Pi_n^0 \iff M = \{\forall x_1 \exists x_2 \forall x_3 \ldots Quant x_n S(w, x_1, \ldots, x_n)\}$
• spet eliplsa, delitev na $\Sigma\Delta\Pi$

Dobimo aritmetično hierarhijo: $\Sigma_1^0, \Delta_1^0, \Pi_1^0, \ldots$ Torej, če bi bil nek Σ_n^0 rešljiv, bi bili rešljivi tudi Δ_n^0, problemi na višjih nivojih pa še vedno ne.

Nekaj primerov in kam v hierarhijo spadajo:

- Problem ustavitve $\mathcal{K}_0 \in \Sigma_1^0$
- Stroji, ki ne sprejmejo nobene besede $\in \Pi_1^0$
- Vprašanje, ali TS razpoznava končen jezik $\in \Sigma_1^0$
- Vprašanje, ali je funkcija totalna $in \Pi_2^0$

Poglavje 5

Zahtevnost računanja

5.1 Deterministična časovna in prostorska zahtevnost

5.1.1 Prostorska zahtevnost

Dan je Turingov stroj M, z enim vhodnim trakom, na katerem je vhodna beseda omejena z mejnikoma za začetek/konec. Vhodni trak je namenjen le branju. Imamo še $k \geq 1$ delovnih trakov, ki so neskončni v eno smer.

Def.: Turingov stroj M ima prostorsko omejitev $S(n)$, če za vsak vhod dolžine n obišče kvečjemu $S(n)$ celic na vsakem delovnem traku. Pravimo, da ima jezik stroja $L(M)$ prostorsko omejitev/zahtevnost $S(n)$

Primer: $L = \{wcw^R | w \in (0+1)^*\}$ ima prostorsko zahtevnost $log(n)$(zaokroženo gor) $\exists TSM$ s prostorsko omejitvijo ... tko, da je L=L(M). Kakšen je M?

5.1.2 Časovna zahtevnost

Dan je Turingov stroj M s $k \geq 1$ trakovi. Na prvem traku je vhodna beseda. Vsak trak je neomejen v obe smeri.

Def.: Turingov Stroj M ima časovno omejitev $T(n)$, če za vsak vhod dolžine n naredi kvečjemu $T(n)$ korakov, preden se ustavi. Pravimo, da ima jezik stroja $L(M)$ časovno omejitev/zahtevnost $T(n)$

Primer: $L = \{wcw^R | w \in (0+1)^0\}$ ker obstaja TS M z časovno zahtevnostjo n+1 z L=(M). Kakšen je M?

5.2 Nedeterministična časovna in prostorska zahtevnost

Def.: Nedeterministični Turingov stroj ima:

- Prostorsko zahtenost/omejitev $S(n)$, če za vsak vhod dolžine n obstaja izračun, ki obišče kvečjemu $S(n)$ celic(na ?vsakem delu traku)
- Časovno zahtenost/omejitev $T(n)$, če za vsak vhod dolžine n obstaja izračun, ki naredi kvečjemu $T(n)$ korakov.

Jezik L ima deterministično prostorsko zahtevnost/omejitev, S(n), če zanj obstaja deterministični turingov stroj ki ima prostorsko omejitev S(n).
Jezik L ima nedeterministično prostorsko zahtevnost/omejitev, S(n), če zanj obstaja nedeterministični turingov stroj ki ima prostorsko omejitev S(n).
Jezik L ima deterministično časovno zahtevnost/omejitev, T(n), če zanj obstaja deterministični turingov stroj ki ima časovno omejitev T(n).
Jezik L ima nedeterministično časovno zahtevnost/omejitev, T(n), če zanj obstaja nedeterministični turingov stroj ki ima časovno omejitev T(n).

5.2.1 Razredi zahtevnosti

- Jeziki, ki jih lahko deterministično razpoznamo v $S(n)$:

  $$\text{DSPACE}(S(n)) \stackrel{def}{=} \{L \mid L \text{ ima deterministično prostorsko omejitev } S(n)\}$$

- Jeziki, ki jih lahko nedeterministično razpoznamo v $S(n)$:

  $$\text{NSPACE}(S(n)) \stackrel{def}{=} \{L \mid L \text{ ima nedeterministično prostorsko omejitev } S(n)\}$$

- Jeziki, ki jih lahko deterministično razpoznamo v $T(n)$:

  $$\text{DTIME}(T(n)) \stackrel{def}{=} \{L \mid L \text{ ima deterministično časovno omejitev } T(n)\}$$

- Jeziki, ki jih lahko nedeterministično razpoznamo v $T(n)$:

  $$\text{NTIME}(T(n)) \stackrel{def}{=} \{L \mid L \text{ ima nedeterministično časovno omejitev } T(n)\}$$

Primer: $L = \{wcw^R \mid ...\}$
$L \in \text{DSPACE}(log(n))$ $L \in \text{DTIME}(n)$

5.2.2 Lastnosti prostorske zahtevnosti

Linearno stiskanje oz. krajšanje trakov

Tračna abeceda je končna, a poljubno velika.

Ideja: Več zaporednih vrednosti na traku zamenjamo z znaki neke nove abecede

Tako lahko vedno zmanjšamo število celic za konstantni faktor.
Pri prostorski omejitvi $S(n)$, nas zanima le njen velikostni razred. (Ki ga lahko opišemo z notacijo z O, Θ in Ω.)

Izrek: Če za jezik L obstaja tračni Turingov stroj, s prostorsko omejitvijo $S(n)$, potem za L obstaja k-tračni Turingov stroj s konstantno prostorsko omejitvijo $c > 0$: $c \cdot S(n)$

Dokaz: Naj bo M_1 Turingov stroj, s prostorsko omejitvijo $S(n)$ in $L = L(M_1)$, $c \in (0, 1)$.
Sestavimo Turingov stroj M_2, ki simulira stroj M_1, tako da:

- V eni celici M_2 hrani r zaporednih celic M_1
- Nadzorna enota stroja M_2 si beleži katero od r celic bi gledal M_1 in se primerno obnaša.

Pri simulaciji stroja M_1, M_2 obišče kvečjemu $\lceil \frac{S(n)}{r} \rceil$.
Določimo r: Naj bo r tak, da je $\frac{1}{r} = \frac{c}{2}$, torej $r = \frac{2}{c}$.
Če je:

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

Stroj M_2 razpoznava L in ima prostorsko omejitev $c \cdot S(n)$.

Posledica: S spreminjanjem prostorske omejitve ne pridobimo na razpoznavni moči.

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

Enako velja tudi za nedeterministično računanje:

$$\text{NSPACE}(S(n)) = \text{NSPACE}(c \cdot S(n)),\ \forall c > 0$$

Odpravljanje trakov

Ali, ter za kakšno ceno za prostorsko zahtevnost, lahko k delovnih trakov nadomestimo z enim samim?

Ideja: k trakov nadomestimo z enim $2k$-slednim trakom

Izrek: Če je L jezik k-tračnega Turingovega stroja, s prostorsko omejitvijo $S(n)$, je taka tudi omejitev eno-tračnega Turingovega stroja.

5.2.3 Lastnosti časovne zahtevnosti

Linearna pospešitev

Množica stanj je omejena, a poljubno velika.

Ideja: Nekaj zaporednih stanj med računanjem nadomestimo z enim novim stanjem iz neke nove množice stanj

Tako lahko vedno zmanjšamo število korakov za konstantni faktor.
Tudi pri časovni omejitvi $T(n)$, nas zanima le velikostni razred. (Ki ga lahko opišemo z notacijo z O, Θ in Ω.)

Def.: Naj bo $f : \mathbb{N} \to \mathbb{N}$ funkcija, kjer velja:

$$\sup_{n\to\infty}(f(n)) \stackrel{def}{=} \lim_{n\to\infty}(\text{LowestUpperBound}\{f(n), f(n+1), \dots\})$$

$$\inf_{n\to\infty}(f(n)) \stackrel{def}{=} \lim_{n\to\infty}(\text{GreatestLowerBound}\{f(n), f(n+1), \dots\})$$

Primer: Naj bo $f(n) = \frac{1}{n}$, če je n sod, ter $f(n) = n$, če je n lih.

•

Velja: Če $f(n)$ konvergira, je limita enaka $\sup_{n\to\infty} f(n)$ in $\inf_{n\to\infty} f(n)$.

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

Posledica: Razred jezikov, ki jih lahko razpoznamo v času $T(n)$, ostane enak, če ga pomnožimo s poljubno konstanto $c > 0$ (če je $\inf_{n\to\infty} \frac{T(n)}{n} = \infty$).

$$\text{DTIME}(T(n)) = \text{DTIME}(c \cdot T(n))$$

Kaj pa če je $\inf_{n\to\infty} \frac{T(n)}{n} < \infty$?

Če je $T(n) = d \cdot n, d > 1$(Torej, da imamo vsaj nekaj korakov časa za računanje), je $\inf_{n\to\infty} \frac{T(n)}{n} = d$ in ne vemo, ali velja zgornja posledica. Bilo bi čudno, če ne bi veljalo podobno, saj imamo d-krat več časa.

Izrek: Če za jezik L obstaja k-tračni Turingov stroj s časom $d \cdot n, d > 1$ in je $k > 1$, potem za jezik za $\forall \varepsilon > 0$ obstaja k-tračni Turingov stroj, s časovno omejitvijo $(1 + \varepsilon) \cdot n$

Posledica: $\text{DTIME}(d \cdot n) = \text{DTIME}((1 + \varepsilon) \cdot n)$ za poljubni konstanti $d > 1$ in $\varepsilon > 0$.

Kaj pa če bi $T(n)$ naraščal počasneje od $d \cdot n$?
Tedaj bi imeli $\inf_{n\to\infty} \frac{T(n)}{n} = 0$ in bi za take funkcije tisti zgornji izrek ne veljal.

Tudi za NTIME velja podobno:

$$\text{NTIME}(T(n)) = \text{NTIME}(c \cdot T(n)),\ c > 0$$

$$\text{NTIME}(d \cdot n) = \text{DTIME}((1 + \varepsilon) \cdot n),\ \text{za poljubni konstanti } d > 1 \text{ in } \varepsilon > 0$$

Odpravljanje trakov

Ali lahko $k > 1$ delovnih trakov nadomestimo z enim samim? Kako bi to vplivalo na časovno zahtevnost?

Primer: $L = \{ww^R \mid w \in \{0,1\}^*\}$
Če je $k = 2$, razpoznamo L v linearnem času.
Če je $k = 1$
Časovna zahtevnost je reda $\Theta(n^2)$

Zmanjšanje števila trakov na enega časovno zahtevnost poveča s kvadratom.

Trditev: To je največja možna upočasnitev pri prehodu s k-trakov na en trak.

Izrek: Če je $L \in \text{DTIME}(T(n))$ na k-tračnem ($k > 1$) Turingovem stroju, potem je $L \in \text{DTIME}(T(n)^2)$ na enotračnem Turingovem stroju.

Torej L zagotovo lahko rešimo v $T(n)^2$.
Enako velja tudi za nedeterministične stroje.

Kaj pa zmanjšanje k-tračnega ($k > 2$) stroja na dva delovna trakova?
Tudi tukaj lahko pride do upočasnitve, a le za faktor $\log T(n)$.

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

Analogno velja za nedeterministične stroje.

5.3 Hierarhije

Po intuiciji: če bi imeli na voljo več prostora ali časa, bi nam to omogočalo rešiti več problemov oz. prepoznati več jezikov.
Toda: izrek o linearni pospešitvi in linearnem stiskanju.

?: Kaj se zgodi z močjo reševanja/prepoznavanja, če prostor ali čas pomnožimo z neko (zelo) počasi rastočo funkcijo? Ali je možno, da nam to še vedno ne bi omogočilo rešiti večjega števila problemov?

Ali obstaja neka *„super"* prostorska ali časovna omejitev $f^*(n)$, da je vsak rekurziven jezik rešljiv v razredu $\text{DSPACE}(f(n))$ oz. $\text{DTIME}(f(n))$?
Taka omejitev ne obstaja niti za prostor, niti za čas. Za vsako prostorsko ali časovno omejitev obstaja jezik, ki za razpoznavanje zahteva več kot toliko prostora ali časa.

Posledica: Obstaja neskončna hierarhija razredov zahtevnosti

Izrek: Naj bo $T(n)$ poljubna totalno rekurzivna časovna omejitev.
Obstaja rekurziven jezik $L \notin \text{DTIME}(T(n))$

Izrek: Naj bo $S(n)$ poljubna totalno rekurzivna prostorska omejitev.
Obstaja rekurziven jezik $L \notin \text{DSPACE}(S(n))$

Dokaz: Če je $T(n)$ totalno izračunljiva, obstaja Turingov stroj M, ki izračuna $T(n)$ za vsak n.
Večtračne Turingove stroje lahko kodiramo z abecedo $\{0,1\}$ in lahko govorimo o i-tem večtračnem Turingovem stroju M_i
Naj bo x_i, i-ta beseda v običajni ureditvi besed v $\{0,1\}^*$.
$L \stackrel{def}{=} \{x_i \mid M_i$ ne sprejme x_i v času največ $T(|x_i|)$ korakov $\}$ Dokažimo, da je L rekurziven:
Za L sestavimo algoritem $\overline{M}$, ki uporabi zgoraj omenjeni M za izračun $T(n)$, $n = |w|$.
poišče kateri po vrsti je w, torej poišče i za katerega velja $x_i = w$.
sestavi kodo stroja M.
simulira stroj M_i nad w za največ $T(n)$ korakov.

$\overline{M}$ sprejme vhod $w \iff M_i$ se je v i korakih ustavil in zavrnil vhod w, ali pa $T(n)$ korakih še vedno računa.
Dokažimo, da $L \in \text{DTIME}(T(n))$:
Predpostavimo: $L = L(M_j)$ za nek j in ima M_j časovno omejitev $T(n)$.
Kje je x_j:
$x_j \in L(M_j) \Rightarrow x_j \notin L$ (po def.)
$x_j \in L(M_j) \Rightarrow x_j \notin L(M_j)$ (po predpostavki)
in
$x_j \in L(M_j) \Rightarrow x_j \notin L$ (po def.)
$x_j \in L(M_j) \Rightarrow x_j \in L(M_j)$ (po predpostavki)

Torej smo v protislovju in ne obstaja tak stroj, ki bi sprejel jezik L s časovno omejitvijo $T(n)$.

5.3.1 Hierarhija razredov DSPACE

?: Kako *,,gosta"* je hierarhija DSPACE

Za koliko moramo povečati prostorsko omejitev $S(n)$, da dobimo nov, večji razred?
Pomembne so *,,lepe"* funkcije, s katerimi opisujemo prostorsko omejitev. Če je $S(n)$ *,,lepa"*, je dovolj, da neka nova funkcija $S(n)'$ narašča le *,,neznatno hitreje"* od $S(n)$, pa že velja, da je DSPACE(S(n)) drugačen od DSPACE($S(n)'$).

Def.: Funkcija je prostorsko predstavljiva (konstruktibilna), kadar obstaja Turingov stroj M s prostorsko omejitvijo $S(n)$, tak da za vsako dolžino $n \in \mathbb{N}$ obstaja vhod w, $|w| = n$, pri katerem M porabi natanko $S(n)$ celic.
Množica prostorsko predstavljivih funkcij je bogata in vsebuje: $\log n, n, 2^n, n! \ldots$

Def.: Funkcija je popolnoma prostorsko predstavljiva, kadar obstaja Turingov stroj M s prostorsko omejitvijo $S(n)$, tak da za vsako dolžino $n \in \mathbb{N}$ pri vseh vhodih w, $|w| = n$, M porabi natanko $S(n)$ celic.

Vsaka prostorsko predstavljiva funkcija $S(n) > n$, je hkrati popolnoma prostorsko predstavljiva.

Def.: Funkcija je *,,lepa"*, če je popolnoma prostorsko predstavljiva.

Def.: Funkcija raste *,,neznatno hitreje"*, če velja $\inf(\frac{S(n)}{S(n)'}) = 0$.

Izrek: Če sta $S_1(n) \geq \log_2 n$, $S_2(n) \geq \log_2 n$ in $\inf \frac{S_1(n)}{S_2(n)} = 0$ in je $S_2(n)$ popolnoma prostorsko predstavljiva, potem obstaja jezik $L \in \text{DSPACE}(S_2(n)) \wedge L \notin \text{DSPACE}(S_1(n))$.
Torej obstaja neskončna hierarhija razredov prostorske zahtevnosti, ki je *,,precej gosta"* (infimum).

5.3.2 Hierarhija razredov DTIME

?: Kako *,,gosta"* je hierarhija DTIME

Izkaže se, da je nekoliko redkejša od prostorske.

Def.: Funkcija $T(n)$ je časovno predstavljiva, če obstaja Turingov stroj M s časovno omejitvijo $T(n)$ in to tak, da za vsako dolžino $n \in \mathbb{N}$, obstaja vhod w, $|w| = n$, pri katerem M naredi natanko $T(n)$ korakov.

Def.: Funkcija $T(n)$ je popolnoma časovno predstavljiva, če obstaja Turingov stroj M s časovno omejitvijo $T(n)$ in to tak, da za vsako dolžino $n \in \mathbb{N}$, za vse vhode w, $|w| = n$ velja, da M naredi natanko $T(n)$ korakov.

Izrek: Če je $T_2(n)$ popolnoma časovno predstavljiva in je $\inf \frac{T_1(n)\log(T_1(n))}{T_2(n)} = 0$, potem obstaja jezik L, za katerega velja:

$$L \in \text{DTIME}(T_2(n)) \wedge L \notin \text{DTIME}(T_1(n))$$

Primer: $T_1(n) = 2^n$
$T_2(n) = n^2 2^n$
$\inf(\frac{T_1(n)\log T_1(n)}{T_2(n)}) = inf(\frac{1}{n}) = 0$

$\text{DTIME}(2^n) \subset \text{DTIME}(n^2 2^n)$

Primer: $T_1(n) = 2^n$
$T_2(n) = n2^n$
$\inf(\frac{T_1(n)\log T_1(n)}{T_2(n)}) = 1$

V resnici $\text{DTIME}(2^n) \subset \text{DTIME}(n^2 2^n)$ velja, ampak s tem izrekom tega ni moč dokazati. (Da pa se dokazati z lemo o pomiku)

5.3.3 Hierarhija razredov NSPACE in NTIME

Lema: Lema o pomiku.
Naj bodo $S_1(n)$, $S_2(n)$ in $f(n)$ popolnoma prostorsko predstavljive in $S_2(n) \geq n$, ter $f(n) \geq n$.

Če $\text{NSPACE}(S_1(n)) \subset \text{NSPACE}(S_2(n))$,
Potem $\text{NSPACE}(S_1(f(n))) \subset \text{NSPACE}(S_2(f(n)))$.

Lema analogno velja tudi za časovno zahtevnost in za deterministične razrede.

5.4 Relacija med prostorsko in časovno zahtevnostjo

?: Kakšna je relacija med razredi različnih vrst zahtevnosti

Kakšen je odnos med determinističnim časom in prostorom?

Izrek: $\text{DTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$,
Torej, kar lahko rešimo v času $O(f(n))$, lahko rešimo tudi na prostoru največ $O(f(n))$.

Izrek: $L \in \text{DSPACE}(f(n)) \wedge f(n) \geq \log_2 n \Rightarrow \exists c(L) : L \in \text{DTIME}(c^{f(n)})$.
Torej, kar lahko rešimo na prostoru $O(f(n))$, lahko rešimo tudi v času največ $O(c^{f(n)})$.

Kakšen je odnos med determinističnim in nedeterminističnim časom?

Izrek: $L \in \text{NTIME}(f(n)) \Rightarrow \exists c(L) : L \in \text{DTIME}(c^{f(n)})$,
Torej, kar lahko rešimo nedeterministično v času $O(f(n))$, lahko tudi deterministično rešimo največ v času $O(c^{f(n)})$.

Največ koliko prostora pa nas stane simulacija nedeternimizna?

Izrek: (Savitch): $\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f(n)^2)$, če je $f(n) \geq \log_2 \wedge f(n)$ popolnoma prostorsko predstavljiva.
Torej, kar lahko rešimo na nedeterminističen način na prostoru $O(f(n))$, lahko na determinističen način rešimo na prostoru največ $O(f(n)^2)$.

Dokaz: a): Trivialno.

b): Naj ima Turingov stroj M_1:
s stanj, t tračnih simbolov, in prostorsko omejitev $f(n)$
Največ $(n+2)sf(n)t^{f(n)}$ trenutnih opisov.
ker je $f(n) \geq \log_2(n), \exists c : \forall n \geq 1 : c^{f(n)} \geq$ št. opisov.

Konstanto c dobimo tako, da logaritmiramo:
$f(n)\log c \geq \log(n+2) + \log s + \log f(n) + f(n)\log t$
$\log c \geq \frac{\log(n+2)+\log s+\log f(n)}{f(n)} + \log t$
Ker je $f(n) \geq \log_2 n$, prvi trije členi monotono padajo, zato lahko zamemo za c:
$\log c \geq \frac{\log 3+\log s+\log f(1)}{f(1)} + \log t$

Sestavimo Turingov stroj M_2 s tremi trakovi.
Na prvih dveh bo simuliral stroj M_1
Na tretjem šteje TO pri simulaciji; od 1. do kvečjemu $c^{f(n)}$

Stroj M_2 sprejme vhod, če simulirani M_1 sprejme vhod.
Stroj M_2 zavrne vhod, če simulirani M_1 zavrne vhod ali pa števec TO $= c^{f(n)}$.

c): Naj ima nedeterministični Turingov stroj M_1:
k trakov, s stanj in t tračnih sibolov.
stroj M_1 ima največ $s(f(n)+1)^k t^{kf(n)}$ trenutnih opisov po $f(n)$ korakih. $\exists d : d^{f(n)} \geq$ št. opisov., za $\forall n \geq 1$
Za d lahko vzamemo $d = s(t+1)^{3k}$

Sestavimo Turingov stroj M_2, ki:
Sestavi seznam vseh TO, ki bi jih lahko dosegel M_1 v $f(n)$ korakih.
V seznamu je največ $d^{f(n)}$ trenutnih opisov.
M_2 jih gotovo lahko zapiše v času, ki je kvadrat dolžine vseh trenutnih opisov.
Za vse to porabi največ $c^{f(n)}$
Stroj M_2 sprejme vhod natanko takrat, kadar med izpisanimi trenutnimi opisi obstaja nek končni trenutni opis.

5.4.1 Izrek o vrzeli

Hierarhiji DSPACE in DTIME sta precej *,,gosti"* in že majhno povečanje virov omogoči reševanje novih problemov.
Izreka s katerima smo to dokazovali, sta zahtevala, da so funkcije časovno oz. prostorsko predstavljive, tu pa bomo pokazali, da se začno dogajati čudne stvari, če tega ne upoštevamo.

Priprava na izrek:
Izberemo poljubno totalno izračunljivo funkcijo $g(n)$, ki raste vsaj linearno.
Izberemo poljubno totalno izračunljivo funkcijo $S(n)$.
Dobimo: $S(n) \leq g(S(n))$ za $\forall n$. Torej $g(S(n))$ narašča hitreje od $S(n)$
DSPACE$(S(n)) \subseteq$ DSPACE$(g(S(n)))$
Ampak predvidevamo, da velja le $\subset$ (in ne $\subseteq$), saj $g(n)$ narašča dovolj hitro, da bo gotovo inf $\frac{S(n)}{g(S(n))} = 0$, kar zahteva tudi to, da je $g(S(n))$ prostorsko predstavljiva.

Toda: Izrek zahteva tudi, da je $g(S(n))$ prostorsko predstavljiva, mi pa tega nismo zahtevali v (neki)
To nam izniči pričakovanje da velja $\subset$ in ne $\subseteq$. Lahko se zgodi, da povečanje (celo bistveno) prostorske omejiteve iz $S(n)$ na $g(S(n))$ ne pomaga.
Vrzel med $S(n)$ in $g(S(n))$ nam ne omogoči nam rešiti nobenega novega problema.

Primer: $g(n) = 2^n$ $S(n) = n^2$

Izrek: (Borodin): Za $\forall$ totalno izračunljivo funkcijo $g(n)$, kjer $g(n) \geq n$ za $\forall n$,
$\exists$ totalno izračunljiva funkcija $S(n)$: DSPACE$(S(n))$ = DSPACE$(g(S(n))$.

5.4.2 Izrek o uniji

DSPACE$(p(n))$, kjer je $p(n) = n, n^2, n^3, \ldots$, so različni razredi, vsebovani v DSPACE$(n^i) \subset$ DSPACE(n^{i+1}).
DSPACE$(n^i) \subset$ DSPACE(n^{i+1}) Podobno velja tudi za DTIME, NSPACE in NTIME.

Ali $\exists S(n) : \mathrm{DSPACE}(S(n))$, ki vsebuje vse $\mathrm{DSPACE}(p(n))$
Ali $\exists S(n) : \mathrm{DSPACE}(S(n)) = \bigcup \mathrm{DSPACE}(p(n))$

Kakšen bi moral biti $S(n)$: Skoraj povsod narašča hitreje od vsakega polinoma
Naraščati mora dovolj počasi, da ne bi zajeli nadpolinomskih funkcij. ($n^{\log n} = \cdots = 2^{c \log^2 n}$)

Izrek: Naj bo $\mathcal{F} = \{f_i(n) \mid i \in \mathbb{N}$ in je f_i izračunljiva $\}$ Turingova množica.
Tedaj obstaja izračunljiva funkcija $S(n)$, da je:

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

Uporaba: $f_i(n) = n^i$
Obstaja $S(n)$, da je:

$$\mathrm{DSPACE}(f(n)) = \bigcup_{i \geq 1} \mathrm{DSPACE}(n^i)$$

Izrek o uniji velja tudi za DTIME, NSPACE in NTIME.

Posledica:

$$\mathrm{P} \overset{def}{=} \bigcup_{i \geq 1} \mathrm{DTIME}(n^i) \qquad \mathrm{PSPACE} \overset{def}{=} \bigcup_{i \geq 1} \mathrm{DSPACE}(n^i)$$
$$\mathrm{NP} \overset{def}{=} \bigcup_{i \geq 1} \mathrm{NTIME}(n^i) \qquad \mathrm{NPSPACE} \overset{def}{=} \bigcup_{i \geq 1} \mathrm{NSPACE}(n^i)$$

V kakšni relaciji pa so ti razredi zahtevnosti?

Trditev: $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NSPACE}$.

Dokaz: $\mathrm{PSPACE} = \mathrm{NSPACE}$: **($\Rightarrow$):** $\mathrm{PSPACE} \subseteq \mathrm{NSPACE}$ – Trivialno
($\Leftarrow$): $\mathrm{PSPACE} \subseteq \mathrm{NSPACE}$
$\mathrm{NSPACE} \overset{def}{=} \bigcup_{i \geq 1} \mathrm{NSPACE}(n^i) \overset{Savitch}{\subseteq} \bigcup_{i \geq 1} \mathrm{DSPACE}(n^{2i}) \subseteq \mathrm{PSPACE}$

$\mathrm{NP} \subseteq \mathrm{PSPACE}$: $L \in \mathrm{NP} \overset{def}{\Rightarrow} \exists k : L \in \mathrm{NTIME}(n^k) \overset{izrek}{\Rightarrow} L \in \mathrm{NSPACE}(n^k) \overset{Savitch}{\Rightarrow}$
$\overset{Savitch}{\Rightarrow} L \in \mathrm{DSPACE}(n^{2k}) \overset{def}{\Rightarrow} L \in \mathrm{PSPACE}$

Če je prostor polinomsko omejen, nedeterminizem ne poveča moči ($\mathrm{PSPACE} = \mathrm{NSPACE}$).
Pri času pa odgovor na to vprašanje ni znan, vprašanje pa poznamo kot $\mathrm{P} \overset{?}{=} \mathrm{NP}$.

Trditev: $\mathrm{DSPACE}(\log n) \subseteq \mathrm{P}$

Dokaz: $L \in \mathrm{DSPACE}(\log n) \overset{izrek}{\Rightarrow} \exists c_L : L \in \mathrm{DTIME}(c_L^{\log n}) \overset{(\log n = k_L \log_{c_L} n)}{\Longrightarrow} L \in \mathrm{DTIME}(c_L^{k_L \log_{c_L} n}) =$
$\mathrm{DTIME}(n^{k_L}) \overset{def}{\Rightarrow} L \in ?$

Trditev: $\mathrm{DSPACE}(\log n) \subsetneq \mathrm{PSPACE}$

Dokaz: Funkciji $\log n$ in n sta obe prostorsko predstavljivi in $\inf\limits_{n \to \infty} \dfrac{\log n}{n} = 0$.
Zato po izreku: $\mathrm{DSPACE}(\log n) \subsetneq \mathrm{DSPACE}(n) \subsetneq \mathrm{PSPACE}$

Posledica: Vsaj eno od naslednjih vsebovanj($\subseteq$) mora biti strogo(torej: $\subsetneq$):
$\mathrm{DSPACE}(\log n) \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NSPACE}$.

Vprašanje P $\stackrel{?}{=}$ NP : •
Zakaj nas zanima? ... gotovo zmoremo deterministično rešiti v eksponentnem času... ... Če je $L \in$ NP$\Rightarrow \exists k : L \in$NTIME$(n^k) \Rightarrow \exists : c_L : L \in$DTIME$(c_L^{n^k})$. ... V praksi to ni zanimivo. Ali vsaj enega takega problema res ne moremo rešiti v polinomskem času? Ali ... res ni močnejši od deterministočnega polinomsko omejenega Turingovega stroja.

Ali konstrukcija problema v NP res ni težja od preverjanja rešitve?

Strategija reševanja P $\stackrel{?}{=}$ NP :
Domnevamo P $\subsetneq$ NP in skušamo dokazati.
Poišči najtežje probleme v NP in poskusi za enega izmed njih dokazati, da ni v P.
• slika P v NP
Nek problem iz NP je med najtežjimi, kadar znamo vsak drug problem iz NP rešiti s prevedbo na tega.
Torej, kadar lahko reševanje prvega problema nadomestimo z reševanjem drugega problema.
Splošno o prevedbah:

Def.: Jezik L je **R-prevedljiv** na jezik L_0, kadar obstaja deterministični Turingov stroj M z lastnostjo R, ki za vsak vhod x vrne izhod $M(x)$. da velja $x \in L \Leftrightarrow M(x) \in L_0$.
• slika S pomočjo M vprašanje $x \in^? L$ nadomestimo z vprašanjem $M(x) \in^? L_0$

Def.: Jezik L_0 pravimo, da je **C-težek** glede na prevedbo ($\stackrel{R}{\leq}$), če za vsak $L \in \mathcal{C} : L \stackrel{R}{\leq} L_0$
• slika
Jezik L_0 ni nujno v $\mathcal{C}$, razen v določenih primerih.

Def.: Jezik L_0 pravimo, da je **C-poln** glede na prevedbo ($\stackrel{R}{\leq}$), če je $L_0 \in \mathcal{C}$ in hkrati C-težek.

Vprašanje P $\stackrel{?}{=}$ NP in prevedbe:
Za reševanje vprašanja P $\stackrel{?}{=}$ NP vzamemo:
$\mathcal{C}$ = NP
$\mathcal{R}$ = polinomsko časovno omejen deterministični Turingov stroj (oz. logaritemsko prostorsko omejen stroj)

$\stackrel{R}{\leq}$ – Polinomska časovna prevedba $\stackrel{p}{\leq}$
(Logaritemska prostorska prevedba $\stackrel{\log}{\leq}$)

Def.: Jezik L je polinomsko prevedljiv na jezik L_0 ($L \stackrel{p}{\leq} L_0$), če obstaja polinomsko časovno omejen Turingov stroj M, ki za vsak vhod x vrne izhod $M(x)$, za katerega je $x \in L \Leftrightarrow M(x) \in L_0$
Opomba: Stroj M vprašanje $x \in^? L$ nadomesti z vprašanjem $p(x) \in L_0$ v polinomsko omejenem času.

Zakaj smo izbrali ravno tako lastnost R?

Zato ker omogoča da: besedo $M(x)$ sestavimo v polinomskem času $p(|x|)$ (časovna omejitev stroja M)
zato je $|M(x)| \leq p(|x|)$

Lema: Naj bo jezik $L \stackrel{p}{\leq} L_0$

Tedaj velja: Če je $L_0 \in$ P $\Rightarrow L \in$ P Če je $L_0 \in$ NP $\Rightarrow L \in$ NP

Kaj pa druga možnost, torej $\stackrel{\log}{\leq}$.

Def.: **Pretvornik** z logaritemsko prostorsko omejitvijo je deterministični Turingov stroj, ki:
ima en vhodni trak
ima en izhodni trak
ima en delovni trak
in se vedno ustavi.

• slika stroja

Jezik L je logaritemsko prostorsko prevedljiv na L_0 ($L \overset{\log}{\leq} L_0$), če obstaja logaritemsko omejen pretvornik M, ki za vsak vhod x vrne izhod $M(x)$, tak da je $x \in L \Leftrightarrow M(x) \in L_0$.

Lema: Naj bo jezik $L \overset{\log}{\leq} L_0$.

Tedaj velja: $L_0 \in \mathrm{P} \Rightarrow L \in \mathrm{P}$

5.5 Dokazovanje NP-polnosti

Trditev: Kompozitum dveh polinomsko zahtevnih prevedb je polinomsko zahtevna prevedba.
Prevedba je torej tranzitivna:

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

• slika tranzitivnost

To odpira možnosti za dokazovanje NP-polnosti in težkosti, jezika L_1 tako, da nanj prevedemo nek NP-poln/težek problem L_0.
• slika • slika
Kako dokažemo NP-polnost prvega NP-polnega problema (torej, če noben drug ni znan) in ali taki problemi sploh obstajajo?

5.5.1 Problem izpolnjivosti Boolovih izrazov

Def.: **Boolove oz. logične izraze** sestavljajo:

- Spremenljivke – $x_1, x_2, \ldots, x_n$
- Operatorji – $\vee, \wedge, \not\ldots$

Boolov izraz je **izpolnjiv**, če lahko vrednosti spremenljivk priredimo tako, da izraz lahko dobi vrednost 1. Možnih prireditev je 2^m, kjer je m število spremenljivk, ki nastopajo v izrazu.

Primer: $x \wedge \overline{x}$ je primer neizpolnjivega izraza.

Izrek: Problem izpolnjivosti je NP-poln odločitveni problem. Problemu pripada jezik L_{SAT}, ki vsebuje kode vseh izpolnjivih Boolovih izrazov.

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

• slika

Trditev: $L_{SAT} \in \mathrm{NP}$

Dokaz: $w \in L_{SAT} \Rightarrow \exists$ nedeterministični Turingov stroj T, tak:

- v vhodni besedi poišče spremenljivke
- ugane rešitev (priredi vrednosti)
- preveri rešitev v polinomskem času

Trditev: L_{SAT} je NP-težek

Dokaz: Dokazati moramo:

$$\forall L, L \in \text{NP} : L \overset{\log}{\leq} L_{SAT}$$

Raje govorimo o Turingovih strojih:

$$\forall T, L(T) \in \text{NP} : L(T) \overset{\log}{\leq} L_{SAT}$$

Sedaj gledamo nedeterministične Turingove stroje:

$$\forall \text{polinomsko časovno omejen nedeterministični stroj } T : L(T) \overset{\log}{\leq} L_{SAT}$$

Po definiciji $\overset{\log}{\leq}$:

$\forall$polinomsko časovno omejen nedeterministični stroj T : $\exists$logaritemsko prostorsko omejen nedeterministični stroj $M : x \in$

Cook: $\exists$ polinomsko omejen izračun:
$\#\beta_0, \#\beta_1, \ldots, \#\beta_Z$
v katerem nedeterministični stroj T sprejme x.

Šele na podagi tega .. izračuna ....
Za vsak poly... obstaja log ... obstaja poly izračun $\forall T \exists M \exists$ polinomsko omejen izračun v katerem nedeterministični stroj T sprejme x. $\Leftrightarrow \exists$ prireditev vrednosti spremenljvikam Boolovega izraza $M(x)$, da ima ta vrednost 1.

Kako na podlagi trenutnih opisov izračuna sestavimo Boolov izraz, ki bo izpolnjiv natanko tedaj, ko je izračun sprejemajoč.

1. Izračun nedeterminističnega Turingovega stroja T pri vhodu x je zaporedje trenutnih opisov $\#\beta_0, \#\beta_1, \ldots \#\beta_Z$
   Pišemo $n = |x|$, ker je T polinomsko omejen, $z \leq p(n)$ za nek polinom.
2. Poenoti se zapis izračunov (zato, da bo splošna in enostavnejša obravnava izračunov)
   Izračun stroja T nad vhodom x tako opisuje neka beseda $\#\beta_0, \#\beta_1, \ldots, \#\beta_{p(n)}$, ki je dolga točno $(p(n)+1)^2$ simbolov.
3. Uvedemo množico Boolovih spremenljivk $c_{i,s}$, kjer je $i \in 0..pn$, $s \in \Gamma \cup \{\#\}$
4. iz spremenljivk $c_{i,s}$ bomo sestavili ... izraz $M(x)$ z lastnostjo: $M(x)$ je resničen pri neki prireditvi vredosti spremenljivk $c_{i,s}$.Ko spremenljivke $c_{i,s}$ z vrednostjo 1 opisujejo ravno izračun v katerem T, sprejme x.
5. Izraz $M(x)$ so konjunkcija štirih logičnih izrazov $M(x) = A \wedge B \wedge C \wedge D$, kjer:
   - A – spremenljivke $c_{i,s}$ z vrednostjo 1 opisuje besedo $\#\beta_0, \#\beta_1, \ldots \#\beta_{p(n)}$
   - B – β_0 opisuje začetno stanje q_0 in vhodno besedo x stroja T
   - C – $\beta_{p(n)}$ opisuje neko končno stanje stroja T
   - D – $\forall \beta_i$ sledi iz β_{i-1} z eno potezo stroja T
6. $A = \ldots$
   $B = \ldots$
   $C = \ldots$
   $D = \ldots$

7. Trditev: Izračun $\#\beta_0, \#\beta_1, \ldots \#\beta_{p(n)}$ sprejme $x \Leftrightarrow M(x)$ je izpolnjiv.
8. Dokazati je treba, da pretovrnik M za konstrukcijo izraza $M(x)$ potrebuje kvečjemu $O(\log |x|)$ prostora.