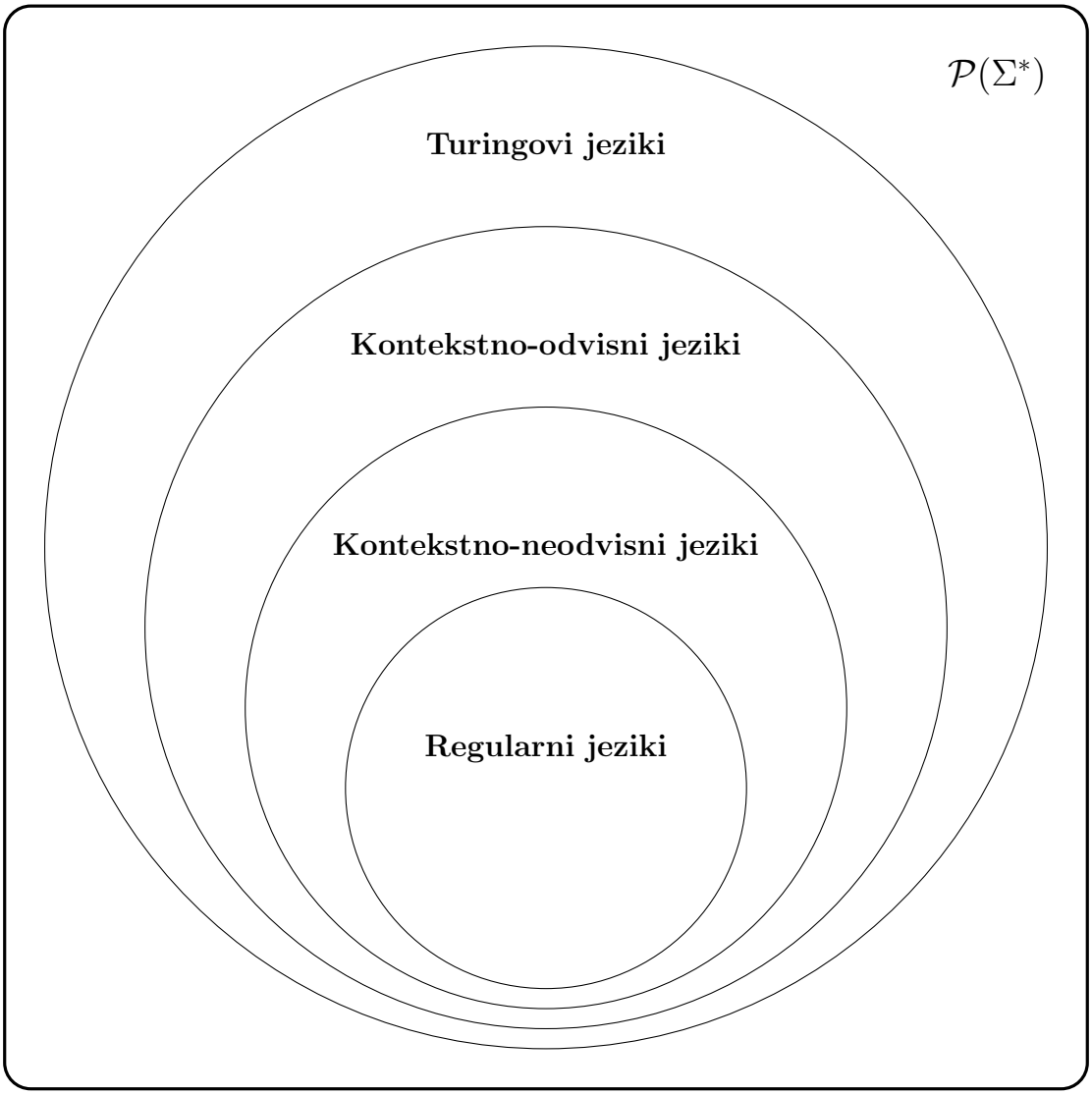
Teoretične osnove računalništva

14. marec 2011



Kazalo

Poglavje 1

Uvod

1.1 Matematične osnove

1.1.1 Teorija množic

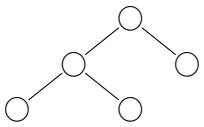
1.1.2 Dokazovanje

Dokaz s konstrukcijo

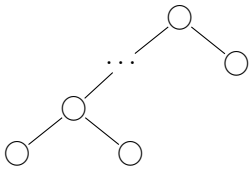
Dokaz obstoja nekega matematičnega objekta je to, da nam ga uspe sestaviti.

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 $|\mathbb{R}| = |[0,1)|$

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 •

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 (niz dolžine 1)
- Σ - abeceda (končna neprazna množica znakov)
- w - niz ali beseda (poljubno končno zaporedje znakov $w_1 w_2 \dots w_n$)
- $|w|$ - dolžina niza
- ε - prazen niz, $|w| = 0$
- Σ^* - vsi možni nizi abecede

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

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

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 3.1), 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 Ohranjanje regularnosti jezikov

Regularnost jezika že po definiciji ohranjajo operacije:

- $L_1 \cup L_2$ - unija
- $L_1 \cdot L_2$ - stik
- L^* - iteracija

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

- $L_1 \cap L_2$ - presek
  Iz avtomatov za L_1 in L_2 zgradimo t.i. produktni avtomat:

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

  Namesto stanj dobimo pare stanj in moramo preveriti v kateri par pridemo, če gledamo oba stara avtomata, končna pa so tista stanja, ki so končna v obeh starih avtomatih.

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

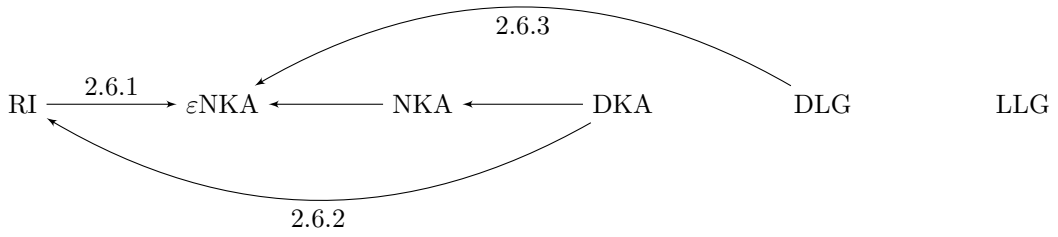
- L^R - obrat
  Obrnemo vse povezave, ustvarimo novo začetno stanje, ki gre po ε v stara končna, staro začetno stanje pa postane edino končno stanje.

Regularnost ohranjajo tudi vse operacije, ki so sestavljene iz zgoraj naštetih:

- $L_1 \setminus L_2 = L_1 \cap \overline{L}_2$ - razlika
- $\overline{L} = \Sigma^* \setminus L$ - komplement
- $L_1 \veebar L_2 = (L_1 \cup L_2) \setminus (L_1 \cap L_2)$ - ekskluzivni ali

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:



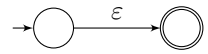
2.6.1 Regularni izraz → Nedeterministični končni avtomat z ε-prehodi

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

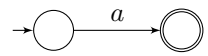
Osnovni izrazi:

- $r = \emptyset$

- $r = \varepsilon$

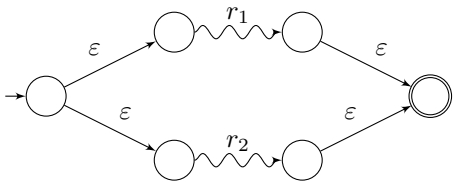


- $r = a$

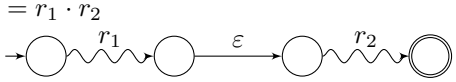


Sestavljeni izrazi:

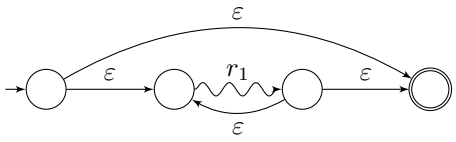
- $r = r_1 + r_2$



- $r = r_1 \cdot r_2$



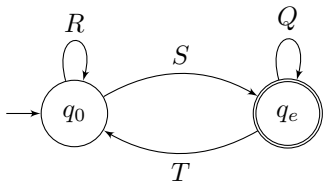
- $r = r_1^*$



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,

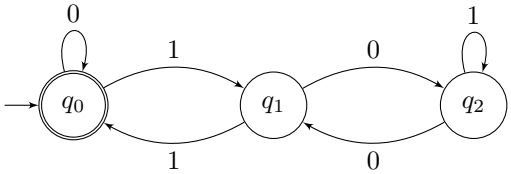


ki jih prepišemo v en sam regularni izraz oblike:

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

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.

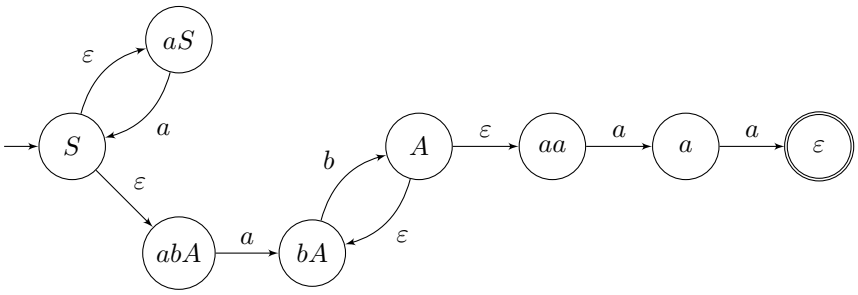
2.6.3 Desno-linearna gramatika → Nedeterministični končni avtomat z ε-prehodi

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 4.3)

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.

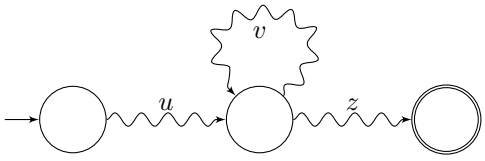
Def.: Za vsak regularni jezik obstaja neka konstanta n, taka, da lahko vsako besedo w iz jezika, daljšo od n, razbijemo na tri dele:

$$w = u\ v\ z$$

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

Linearni jeziki

3.1 Linearne gramatike

Def.: Linearna gramatika je gramatika, ki ima na desni strani produkcij največ en vmesni simbol. Definirana kot četvorček $G = \langle N, T, P, S \rangle$, kjer so:

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

Posebna primera sta levo in desno-linearne gramatike, ki opisujeta regularne jezike (glej 2.3)

Primeri:

Primer 1: Sestavi linearno gramatiko, ki sprejme jezik $L = \{a^n b^n \mid n > 0\}$.

$$S \to aSb \mid ab$$

Primer 2: Sestavi linearno gramatiko, ki sprejme jezik $L = \{10^n 10^n 1 \mid n \geq 2\}$.

$$S \to 100A001$$

$$A \to 0A0 \mid 1$$

Primer 3: Sestavi linearno gramatiko, ki sprejme jezik $L = \{ww^R \mid w \in \{0,1\}\}$.

$$S \to 1S1 \mid 0S0 \mid \varepsilon$$

Poglavje 4

Kontekstno-neodvisni jeziki

4.1 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.

4.1.1 Chomskyeva normalna oblika

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

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

4.1.2 Greibachina normalna oblika

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

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

4.2 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

4.2.1 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

4.2.2 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.

4.2.3 Jezik skladovnega avtomata

4.3 Dokazovanje kontekstne-neodvisnosti jezika

4.3.1 Lema o napihovanju za kontekstno-neodvisne jezike

4.3.2 Ogdenova lema za kontekstno-neodvisne jezike

Poglavje 5

Kontekstno-odvisni jeziki

5.1 Kontekstno-odvisne gramatike

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

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

Pri tem je:

$$P \subset [\alpha_1 A \alpha_2 \to \alpha_1 \gamma \alpha_2]$$

$$A \in N, \quad \alpha_1, \alpha_2 \in (N \cup T)^*, \quad \gamma \in (N \cup T)^+$$

Torej, niz z vsaj enim vmesnim simbolom preslikamo v nek drug niz. Pri tem je omejitev, da končnih simbolov ne smemo spreminjati.

Primeri:

Primer 1: Sestavi kontekstno-odvisno gramatiko, ki sprejme jezik $L = \{a^n b^n c^n \mid n > 0\}$.

$$\begin{aligned} S &\to aSBC \\ S &\to aBC \\ CB &\to HB \\ HB &\to HC \\ HC &\to BC \\ aB &\to ab \\ bB &\to bb \\ bC &\to bc \\ cC &\to cc \end{aligned}$$

S kompleksnejšo kontekstno-odvisno gramatiko sprejmemo tudi jezik $\{a^n b^n c^n d^n \mid n > 0\}$

5.1.1 Kurodova normalna oblika

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

$$\begin{aligned} AB &\to CD \\ A &\to BC \\ A &\to B \\ A &\to a \end{aligned}$$

$$a \in T, \quad A, B, C, D \in N$$

Poglavje 6

Turingovi jeziki

6.1 Zgodovina

Leta 1900 je Nemški matematik David Hilbert objavil seznam triidvajsetih 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 odprtih 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 - •

Nekateri zgodnji poskusi formalizacije pojma algoritma so:

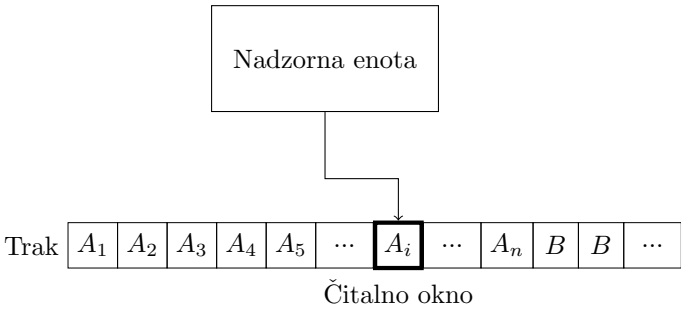
- Rekurzivne funkcije (Kurt Gödel, Stephen Kleene)
- Splošne rekurzivne funkcije (Jacques Herbrand, Kurt Gödel)
- Algoritmi Markova (Andrey Markov, ml.),
- Produkcijski sistem (Emil Post),
- Lambda račun (Alonso Church, 1936)
- Turingov stroj (Alan Turing, 1936)

6.2 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:

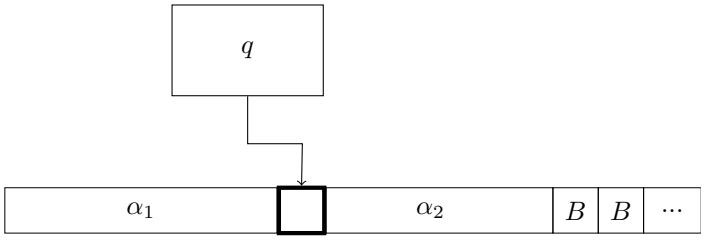
- Q končna množica stanj
- Σ končna množica vhodnih simbolov, $Q \cap \Sigma = \emptyset$
- Γ končna množica tračnih simbolov, $\Sigma \subset \Gamma$
- δ funkcija prehodov: $Q \times \Gamma \to Q \times \Gamma \times \{L, D\}$, kjer L in D označujeta premik levo ali desno
- q_0 začetno stanje, $q_0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

6.2.1 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

6.2.2 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 $x_0 \vdash x_1 \wedge x_1 \vdash x_2 \wedge \dots \wedge x_{k-1} \vdash x_k$

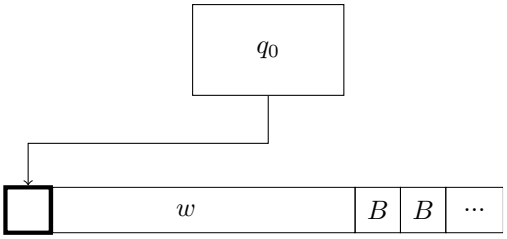
Torej, trenutni opis v sledi iz u, v k korakih Turingovega stroja.

6.3 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 $L = L(M)$.

6.3.1 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
- $X0^{n-1}1^n$ - 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:

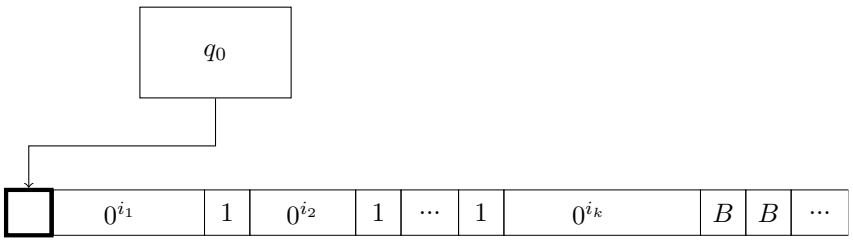
	0	1	B	X	Y
x_0	$\langle q_1, X, D\rangle$	–	–	$\langle q_3, Y, D\rangle$	–
x_1	$\langle q_1, 0, D\rangle$	$\langle q_2, Y, L\rangle$	–	$\langle q_1, Y, D\rangle$	–
x_2	$\langle q_2, 0, D\rangle$	–	$\langle q_0, X, D\rangle$	$\langle q_2, Y, L\rangle$	–
x_3	–	–	–	$\langle q_3, Y, D\rangle$	$\langle q_4, B, D\rangle$
x_4	–	–	–	–	–

Izvajanje stroja s trenutnimi opisi:

$$q_0 0011 \vdash X q_1 011 \vdash X0 q_1 11 \vdash X q_2 0Y1 \vdash \dots$$

6.3.2 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

6.3.3 Lažja konstrukcija Turingovih strojev

Obstaja nekaj tehnik, ki poenostavijo in pohitrijo sestavljanje Turingovih strojev.

Nadzorna enota kot pomnilnik

Vsako stanje stroja, je sestavljeno iz dveh delov – stanja avtomata, ter 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 $\delta(\langle q_1,1\rangle, 1)$

Večsledni trak

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

- - slika večslednega stroja

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

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:
    $\delta(\langle q_1, A_{n-1}, A_n\rangle, B) = \langle\langle q_1, A_n, B\rangle, A_{n-1}, D\rangle$
    $\delta(\langle q_1, A_n, B\rangle, B) = \langle\langle q_2, B, B\rangle, A_n, L\rangle$

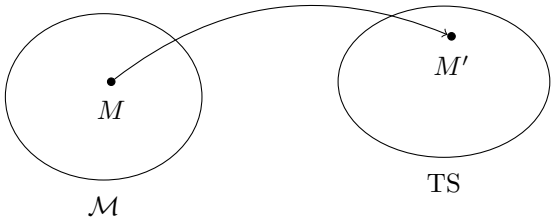
Podprogrami

-

6.4 Moč Turingovega stroja

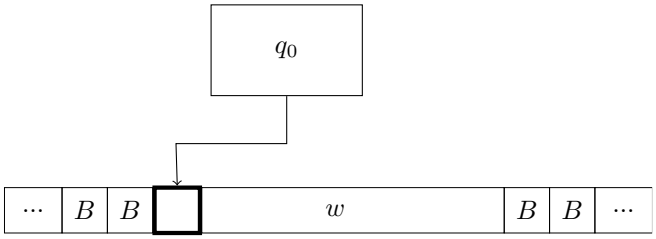
V moč Turingovega stroja se bomo prepričali na dva načina – tako, da bomo pokazali, da so različne razširitve Turingovega stroja enako močne kot osnovni model, ter tako, da bomo pokazali, da je tudi nekaj precej drugačnih modelov računanja enako močnih. Nato pa bomo pogledali še nekaj stvari, ki jih Turingov stroj kljub vsemu ne more izračunati.

Postopek dokazovanja: Recimo, da je $\mathcal{M}$ razred modelov, za katerega želimo dokazati, da so ekvivalentni TS, moramo poiskati sistematičen, končen postopek S, ki poljubnemu stroju $M \in \mathcal{M}$ priredi nek TS M', ki je sposoben simulirati M.



6.4.1 Turingov stroj z dvosmernim trakom

Imamo Turingov stroj, ki ima trak neomejen v obe smeri. Vhodna beseda se vpiše nekam na trak, okno se prestavi na prvi znak besede.



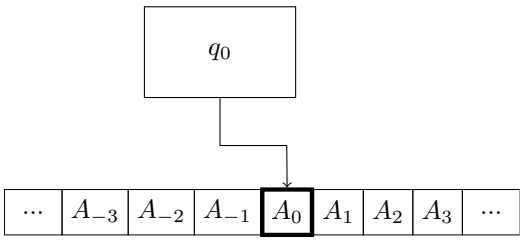
Stroj je definiran skoraj enako kot običajni Turingov stroj, le funkcija prehodov δ je enostavnejša, saj ni treba skrbeti, kaj se zgodi, če zadanemo levi rob, kot pri običajnem Turingovemu stroju.

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

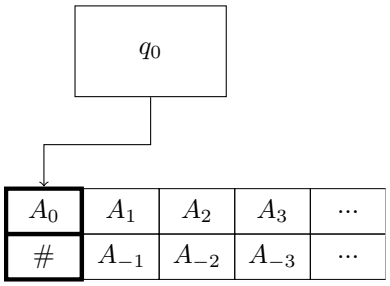
Dokaz: Lahko se vede kot da je omejen na levi. • - to smo delali na vajah

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

Dokaz: Imamo poljuben Turingov stroj z dvosmernim trakom:



Priredimo mu dvosledni Turingov stroj M':

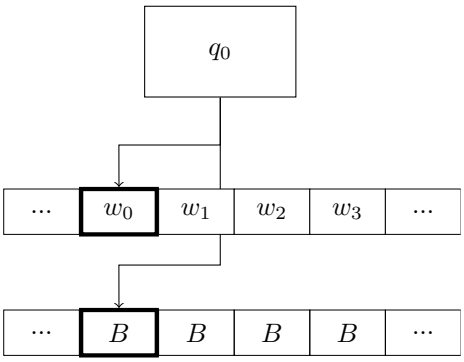


Stroj M' deluje tako:

- 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

Večtračni Turingov stroj

Večtračni Turingov stroj ima več trakov, ki so neomejeni v obe strani, ter vsak trak ima svoje okno, ki se neodvisno premika ob vsakem koraku. Na začetku imamo po dogovoru vhodno besedo na prvem traku in je okno prvega traku na prvem znaku vhodne besede.



Def.: Korak stroja δ opišemo kot:

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

Torej, na vsakem koraku dobimo iz trenutnega stanje glave, ter tračnega simbola iz vsakega traku, neko novo stanje, ter za vsak trak neodvisno nov simbol in premik.

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

Dokaz: **($\Rightarrow$):** Preprosto delamo vse le na prvem traku

($\Leftarrow$): Vzamemo k-tračni Turingov stroj M in $2k$-sledni v obe strani neskončni M'.

od tu naprej je neurejeni del zapiskov.. halp!

- dve sledi za vsak trak.. na prvi oznka X, ki pove ... na spodnji je ... stroja M
- - slika z dvemi trakovi in en večsledni

... , ki vsebuje stanje prostor za k ... znakov števec je omejen za štetje 0..k

M' simuliramo en korak stroja M tako ...pod njim in zmanjša števec za 1 ko števec pade na nič, se začne pomikati v levo pri tem pomikanju v levo, vsakokrat, ko najde X .. pod njim .... (kot bi zamenjal M v svojem ...)

poveča števec za 1 ko števec naraste na k, se pomikanje v levo konča, nadzorna enota predie v novo stanje

Torej en korak stroja M simulira M' s končno dolgim sprehodom iz desne pa v levo?

enako sta močna

Izrek: za L obstaja večtrani TS ntk, obstaja za L običajni TS.

Nedeterministični Turingov stroj

$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,D,N\}}$ .. .. toda le končno mnogo

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 je to vodi. ...: ...

Def.: NTS sprejme besedo, kadar obstaja konno zaporedje kononononooooooooooo!!! korakov, ki se konča v končnem stanju

NTS zmore vse kar zmore osnovni TS Dokaz: δ osnovnega TS je le poseben primer delte NTS NTS Ni močnejši Dokaz: simulacija NTS z osnovnim: naj ima njegov program delta v vsaki množici delta q a kvečjemu r možnih potez(oz. elementov delta? množice) Stroju M priredimo trisledni osnovni TS M' na prvi sledi ima M' vhodno besedo M na drugi sledi M' generira/izpisuje navodila eno za drugim. navodila so besed nad 1,2...r v leksikografskem redu. npr r=3 ... 1,2,3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, 112, na tretji sledi simulira stroj M, kot da bi M sam izbral svoje poteze, skladno s tekočim navodilom. ajprov?

Natančneje: M' na 2. sledi sestavi novo, naslednje navodilo prepiše vhodno besedo s prvega na tretji sled (prej lahko tretha sked zbrupe(tretjo sled zbriše), če je kaj ostalo od prej) na tretji sledi oponaa stroj M, kot da bi ta izbiral svoje poteze po tekočem navodilu.

pri tem: če M' pride do konca navodila, in je tedaj v končnem stanju, besedo sprejmemo. sicer če ne pride do konca navodila, ali pa se ustavi v nekončnem stanju, potem pojdi v (natančneje prva vrstica)

Trditev: če M sprejme besedo, jo sprejme tudi M' Dokaz:

izrek: za L obstaja nts, ntk za L obstaja osnovni ts.

Večdimenzionalni Turingov stroj

... k¿=2 premikamo se lahko v 2k smeri ... ...

Primer: k=3

- - 3D trak s trollhands puščico na celico

Turingov stroj z več okni

- - slika z večimi okni

Rekurzivne funkcije

Definiral je funkcije: ničelna, naslednik, projekcija. Dodal je pravili sestavljanja, kako iz začetnih in že sestavljenih dobimo novo: kompozicija, primitivna rekurzija. Konstrukcija take funkcije opisuje tudi mehanični postopek za izračun vrednosti funkcije, to pa je tudi kandidat za formalni opis pojma algoritma.

Ampak... Ackermannova funkcija ni primitivno rekurzivna in narašča hitreje od vsake funkije, ki pripada primitivno rekuzivnim.

Kleene leta 1936 doda šeeno pravilo sestavljanja, μ-operacija.

Def.: Totalne š.t. funkcije $f : N^k \leftarrow N$, ki se dajo konstruirati iz treh začetnih funkcij s končno mnogo uporabami treh pravil sestavljanja se imenujejo rekurzivne funkcije.

Razred rekurzivnih funkcij ...

- $\zeta(n) = 0 \forall n$ - ničelna
- $\sigma(n) = n + 1 \forall n$ - naslednik
- $\pi(n_1, n_2, \ldots, n_k) = n_i \forall n..$ - projekcija

Pravila sestavljanja 1. Če so dane funkcije $g : N^m \leftarrow N$ in $n_h : N^k \leftarrow N$, kjer i 1-m, potem je funkcija $f(n_1, n_2, \ldots, n_k)$ po def $g(h_1(n_1, n_2, \ldots, n_k), \ldots, h_m(n_1, n_2, \ldots, n_k))$ sestavljena s kompozicijo funkcij 2. Če sta dani funkciji $g : N^k \leftarrow N$ in $h : N^{k+1} \leftarrow n$, potem je f definiriana z:

$f(n_1, n_2, \ldots, n_k, 0)$ po def $g(n_1, n_2, \ldots, n_k)$

$f(n_1, n_2, \ldots, n_k, m+1)$ po def $h(n_1, n_2, \ldots, n_k, m), f(n_1, n_2, \ldots, n_k, m)), zam \geq 0$

sestavljeno s .... in g in h 3. Če je funkcija $g : N^{k+1} \leftarrow N$ taka, da za vsako k-terico naravnih števil $n_1, n_2, \ldots, n_k$ obstaja n ..neki.. m, da je $g(n_1, n_2, \ldots, n_k, m) = 0$, potem je funkcija:

$f(n_1, n_2, \ldots, n_k) pode f \mu x g(n_1, \ldots, n_k, x)$

sestavljena z μ-operacijo iz funkcije g.

Konstrukcija rekurzivne funkcije f je končno zaporedje $f_1, f_2, \ldots, f_L$, kjer je $f_L = f$ in je vsaka funkcija f_i vmes, bodisi začetna, bodisi sestavljena z enim izmed treh pravil iz predhodnic v tem zaporedju

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

Splošne rekurzivne funkcije

Herbrand je študiral, kako poljubno ŠT funkcijo definirati s sistemom enačb.

f je neznana funkcija, $g_1, g_2, \ldots g_m$ pa znane f-je in g-je poljubno vstavljamo kot argumente v druge, nato pa nekatere dobljene izraze izenačimo, potem pa če ima dobljeni sistem natanko eno rešitev za funkcijo f, potem je f rekurzivna.Gödel je dodal dve dodatni zahtevi... f se sme na levi strani enačb sme pojaviti v obliki $f(g(...), g_k(...)$ f naj bo povsod na N^k definirana (totalna). Če se jo da zapisati s takim sistemom, je zapisana s standardnim sistemom. Tedaj je tak sistem za f z oznako $\varepsilon(f)$.

Kakšna so pravila za računanje vrednosti $f(n_1, n_2, \ldots, n_k)$ iz $\varepsilon(f)$. Pravili sta samo 2. 1. v enačbi lahko vse pokave iste spremenljivke zamenjamo z istim naravnim številom??? 2. v enačbi lahko pojave funkcije zamenjamo z njeno vrednostjo

Godel trdi: funkcija f za katero obstaja $\varepsilon(f)$, se imenjue splošno rekurzivna

Povzetek: Algoritem po Herbrant-Gödlu(HG) je $\varepsilon(f)$.
Računanje po HG je izračun vrednosti funkcije $f(n_1, \ldots, n_k)$ iz $\varepsilon(f)$ z uporabo pravil 1,2.
Funkcija je izračunljiva po HG, če je splošno rekurzivna.

λ račun

Imamo vhodni izraz, ki opisuje neko funkcijo f in argumente $n_1, \ldots, n_k$.

Kako je funkcija opisana? Začetni λ-term

Cilj: preoblikovati začetni λ-term v končni λ-term, tak, ki bo opisoval ravno vrednost $f(n_1, \ldots, n_k)$

To preoblikovanje dosežemo z uporabo t.i. redukcij. Ta
bodisi preimenuje spremenljivko v λ-termu, (α) bodisi uporabi neko funkcijo nad njenimi argumenti.(β)

$t0 -> t1 -> ... -> tK \ f, n1, n2, ...nk....f(n_1, \ldots, n_k)$

Def.: funkcija, ki jo je možno predstaviti in računati v tem lambda računu, taka funkcija 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.

Postov Stroj

Model je podoben Turingovemu stroju, z naslenjimi spremembami:

- s traku le bere znake
- Uporablja vrsto znakov

Korak: prebere iz celice pod oknom znak in iz začetka vrste znak. na podlagi teh dveh znakov in stanja bo premaknil okno, nov znak dal na konec vrste in prešel 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.

Algoritmi Markova

Imamo abecedo Σ, končno zaporedje produkcij: $x_1 \leftarrow y_1\ x_2 \leftarrow y_2\ ...\ x_n \leftarrow y_n\ x, y \in \Sigma^*$
Produkija preoblikuje besedo tako da v tej 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 s prvo (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

6.4.2 Church-Turingova teza

Church je postavil domnevo... "algoritem"debela leftright puščica algoritem po Churchu
Turing je postavil domnevo: "algoritem"debela leftright puščica algoritem po Turingu
Church-Turingova teza: algoritem intuitivno debela leftright puščica algoritem po Turingu