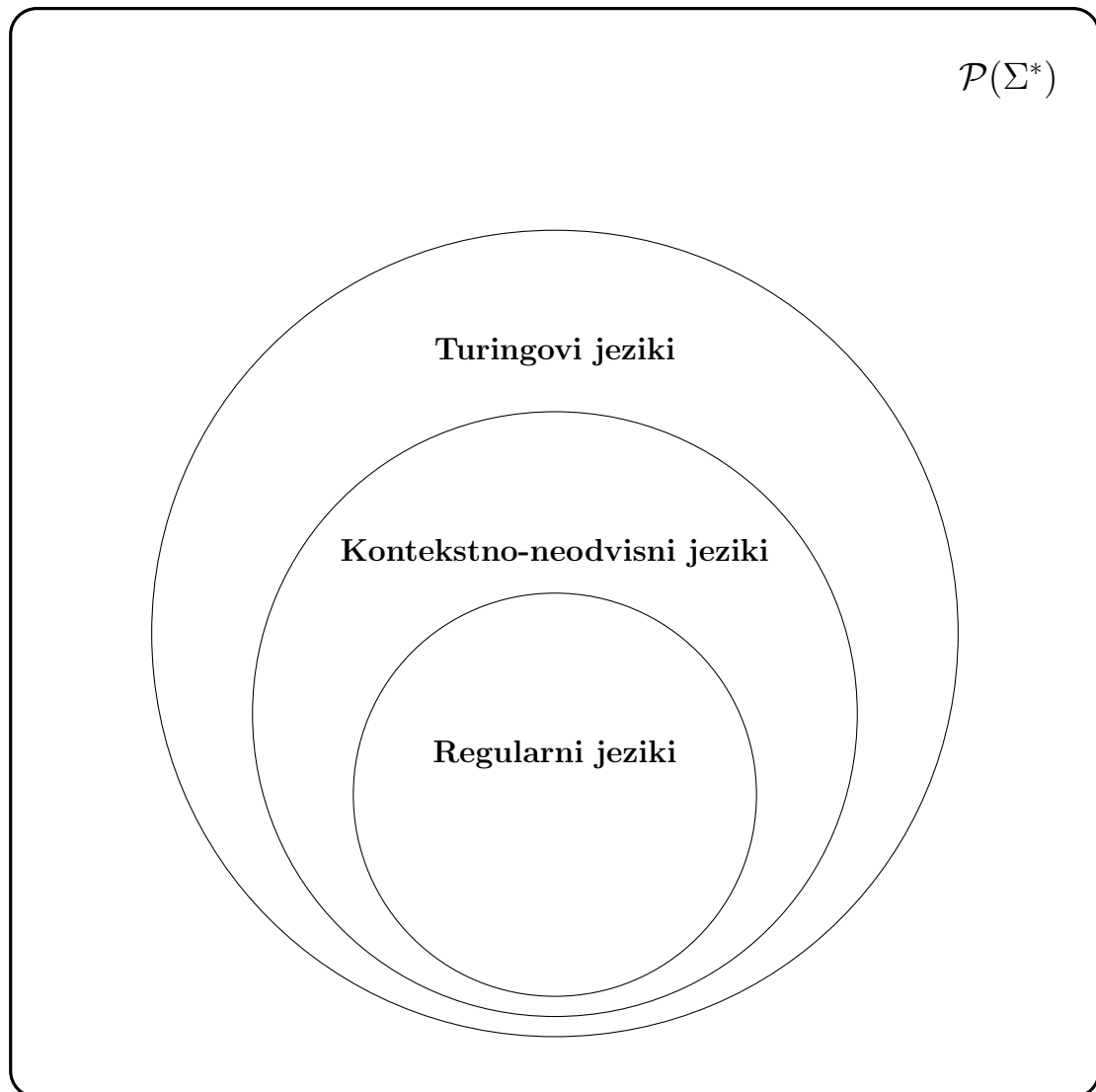


Teoretične osnove računalništva

3. april 2011



This work is licensed under a Creative Commons
Attribution-NonCommercial-ShareAlike 3.0
Unported License

Kazalo

1	Uvod	3
1.1	Matematične osnove	3
1.1.1	Teorija množic	3
1.1.2	Dokazovanje	3
1.2	Osnove teorije jezikov	4
1.2.1	Uporabljene oznake	5
1.2.2	Operacije nad jeziki	5
2	Regularni jeziki	6
2.1	Regularni izrazi	6
2.2	Končni avtomati	7
2.2.1	Nedeterministični končni avtomati z ε -prehodi	7
2.2.2	Nedeterministični končni avtomati	7
2.2.3	Deterministični končni avtomat	7
2.2.4	Jeziki končnih avtomatov	7
2.3	Levo in desno-linearne gramatike	8
2.3.1	Produkcije	8
2.3.2	Relacija izpeljave \Rightarrow	8
2.3.3	Jezik gramatik	8
2.4	Jezik regularnih jezikov	8
2.5	Ohranjanje regularnosti jezikov	9
2.6	Prevedbe med modeli regularnih jezikov	9
2.6.1	Regularni izraz \rightarrow Nedeterministični končni avtomat z ε -prehodi	9
2.6.2	Končni avtomat \rightarrow Regularni izraz	10
2.6.3	Desno-linearne gramatika \rightarrow Nedeterministični končni avtomat z ε -prehodi	10
2.7	Dokazovanje regularnosti jezika	11
2.7.1	Lema o napihovanju za regularne jezike	11
2.7.2	Izrek Myhill-Nerode	12
2.7.3	Minimizacija končnih avtomatov	12
3	Kontekstno-neodvisni jeziki	13
3.1	Kontekstno-neodvisne gramatike	13
3.1.1	Chomskyeva normalna oblika	13
3.1.2	Greibachina normalna oblika	13
3.2	Skladovni avtomati	13
3.2.1	Trenutni opis	14
3.2.2	Relacija \vdash	14
3.2.3	Jezik skladovnega avtomata	14
3.3	Dokazovanje kontekstne-neodvisnosti jezika	14
3.3.1	Lema o napihovanju za kontekstno-neodvisne jezike	14
3.3.2	Ogdenova lema za kontekstno-neodvisne jezike	14

4	Turingovi jeziki	15
4.1	Zgodovina	15
4.2	Turingovi stroji	16
4.2.1	Trenutni opis	16
4.2.2	Relacija \vdash	17
4.3	Jezik Turingovega stroja	17
4.3.1	Ugotavljanje pripadnosti besed Turingovemu jeziku	17
4.3.2	Turingov stroj kot računalnik funkcij	18
4.4	Razširitve in alternative Turingovemu stroju	19
4.4.1	Nadzorna enota kot pomnilnik	19
4.4.2	Večsledni trak	20
4.4.3	Prestavljanje vsebine traku	20
4.4.4	Podprogrami	21
4.4.5	Turingov stroj z dvosmernim trakom	21
4.4.6	Večtračni Turingov stroj	22
4.4.7	Nedeterministični Turingov stroj	23
4.4.8	Večdimenzionalni Turingov stroj	25
4.4.9	Rekurzivne funkcije	25
4.4.10	λ -račun	26
4.4.11	Postov Stroj	26
4.4.12	Algoritmi Markova	26
4.5	Church-Turingova teza	27
4.6	Univerzalni Turingov stroj	28
4.6.1	Pravi stroji, ki so univerzalni	29
4.7	Reševanje računskih problemov	29
4.7.1	Jezik odločitvenih problemov	29
4.7.2	Neodločljivi problemi	30
4.7.3	Primeri nerešljivih problemov	30
4.7.4	Dokazovanje neodločljivosti problemov	31

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:

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

- Definiramo preslikavo:

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

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: $n \in \mathbb{N} \implies n + 1 \in \mathbb{N}$

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:
 $P = \{2, 3, 5, \dots, p\}$, 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 :
 $q = 2 * 3 * 5 * \dots * p + 1$
- 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 $a = 2k$:

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

- 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

$$w = a_1 a_2 \dots a_n$$

$$x = b_1 b_2 \dots b_m$$

$$wx = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$$

Stik

$$A = \{w_1, w_2, \dots, w_n\}$$

$$B = \{x_1, x_2, \dots, x_m\}$$

$$A \cdot B = \{w_i x_j \mid w_i \in A \wedge x_j \in B\}$$

Potenciranje

$$A^0 = \{\varepsilon\}$$

$$A^k = A \cdot A \cdot \dots \cdot A = \bigcirc_{i=1}^k A$$

Iteracija

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

Obrat

$$w = a_1 a_2 \dots a_{n-1} a_n$$

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

Poglavje 2

Regularni jeziki

2.1 Regularni izrazi

Def.: Imamo tri osnovne izraze:

- \emptyset je opisuje prazen jezik $L(\emptyset) = \{\}$
- ε opisuje jezik $L(\varepsilon) = \{\varepsilon\}$
- \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 $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
- $(r_1 r_2)$ 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 mešanica 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?

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

2.2 Končni avtomati

2.2.1 Nedeterministični končni avtomati z ε -prehodi

Def.: ε 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, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$
- 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 možna stanja do katerih pridemo iz A z določenim znakom abecede in z vsemi ε prehodi, naprimer $\{A_1, A_2, \dots, A_n\}$. Tukaj je množica stanj $\{A_1, A_2, \dots, A_n\}$ 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 $\delta : Q \times \Sigma \rightarrow 2^Q$
- 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.

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

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, $\delta : Q \times \Sigma \rightarrow Q$
- q_0 - začetno stanje
- F - množica končnih stanj

2.2.4 Jeziki končnih avtomatov

Def.: Jezik ε NKA 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-linearne gramatikah, s produkcijami slikamo nek vmesni simbol v niz, ki ima lahko vmesni simbol le na skrajno levi pri levo-linearne, oz. le na skrajno desni pri desno-linearne:

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

2.3.2 Relacija izpeljave \Rightarrow

Def.: Relacija izpeljave pri levo in desno-linearne 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-linearne pa le na skrajno desni strani, torej:

- $[A \rightarrow B\beta]$ - pri levo-linearne gramatikah
- $[A \rightarrow \beta B]$ - pri desno-linearne 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 splošno relacijo izpeljave \Rightarrow^* .

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

2.3.3 Jezik gramatik

2.4 Jezik regularnih jezikov

Def.: Jezik ki ga opisuje poljubni regularni izraz, končni avtomat, levo ali desno-linearne 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:

$$M_{L_1} = \{Q_1, \Sigma, \delta_1, q_{1_0}, F_1\}$$

$$M_{L_2} = \{Q_2, \Sigma, \delta_2, q_{2_0}, F_2\}$$

$$M_{L_1} * M_{L_2} = \{Q_1 \times Q_2, \Sigma, \delta_*, \langle q_{1_0}, q_{2_0} \rangle, F_1 \times F_2\}$$

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

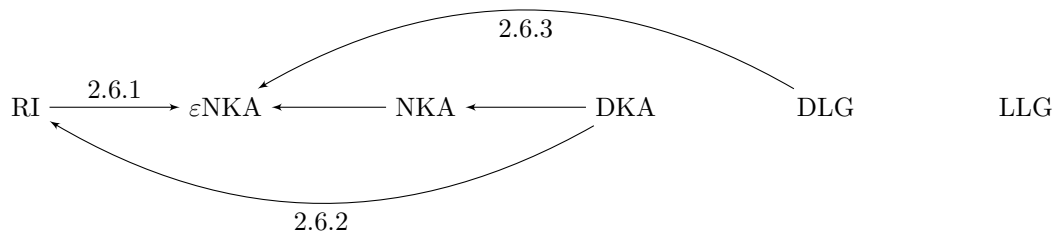
Obrnemo vse povezave, ustvarimo novo začetno stanje, ki gre po ε v stara končna, staro začetno stanje pa postane edino končno stanje.

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
- $L_1 \underline{\vee} L_2 = (L_1 \cup L_2) \setminus (L_1 \cap L_2)$ - ekskluzivni ali

2.6 Prevedbe med modeli regularnih jezikov

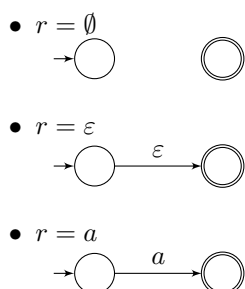
Regularni izrazi, regularne gramatike in končni avtomati so enako močni modeli in je mogoče pretvarjati med njimi. V tem odseku bomo predstavili naslednje prevedbe:



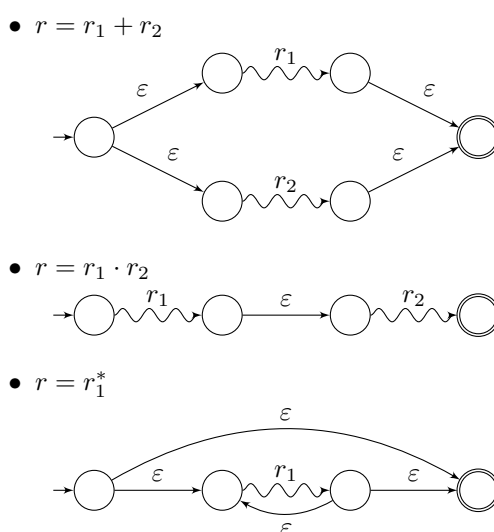
2.6.1 Regularni izraz \rightarrow Nedeterministični končni avtomat z ε -prehodi

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

Osnovni izrazi:



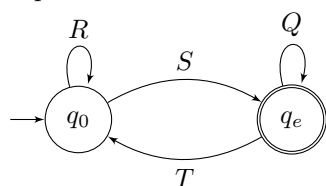
Sestavljeni izrazi:



2.6.2 Končni avtomat \rightarrow Regularni izraz

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

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

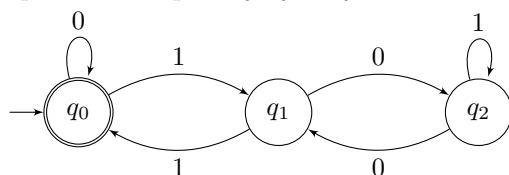


ki jih prepišemo v en sam regularni izraz oblike:

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

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 \rightarrow Nedeterministični končni avtomat z ε -prehodi

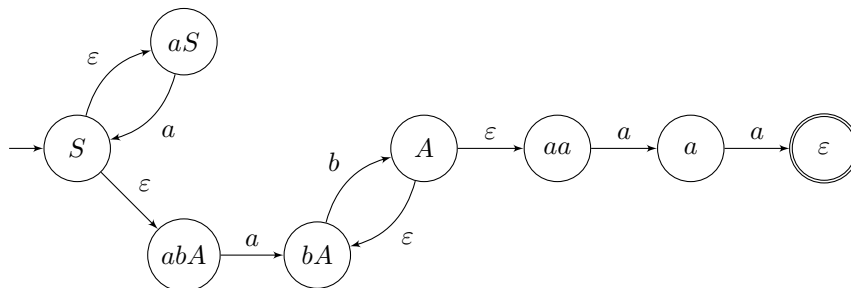
Vhodno stanje avtomata je začetni simbol gramatike, nato pa stanja označujemo glede na končne in vmesne simbole, ki jih moramo še porabiti. Produkcije predstavljajo ε prehode v avtomatu, preostali prehodi avtomata pa so črke, ki jih generiramo.

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

$$S \rightarrow abA \mid aS$$

$$A \rightarrow aa \mid bA$$

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

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

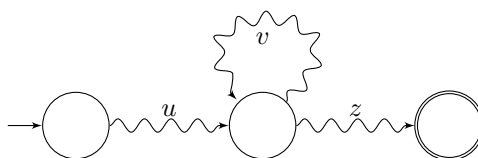
Def.: Za vsak regularni jezik obstaja neka konstanta n , taka, da lahko vsako besedo w iz jezika, daljšo od n , razbijemo na tri dele:

$$w = u v z$$

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:

$$A \rightarrow a$$

$$A \rightarrow BC$$

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

3.1.2 Greibachina normalna oblika

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

$$A \rightarrow a\gamma$$

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

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, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$

- 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 $P(x_1, x_2, \dots, x_n) = 0$, 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:

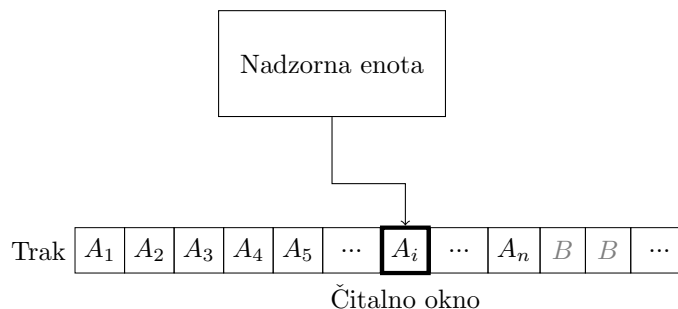
- Rekurzivne funkcije (Kurt Gödel, Stephen Kleene)
- Splošne rekurzivne funkcije (Jacques Herbrand, Kurt Gödel)
- Algoritmi Markova (Andrey Markov, ml.),
- Produkcijski sistem (Emil Post),
- Lambda račun (Alonso Church, 1936)
- Turingov stroj (Alan Turing, 1936)

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:

- Nadzorna enota (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: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, D\}$,
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, $F \subseteq Q$

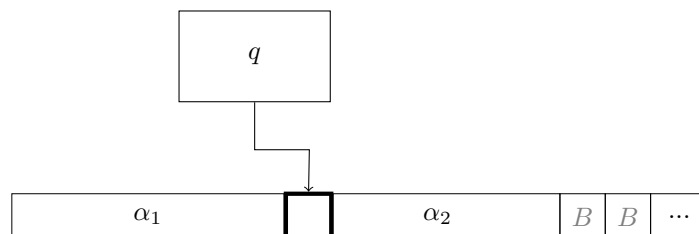
Stroj deluje tako, da v vsakem koraku opravi naslednje:

- 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 $L = \{0^n 1^n \mid n \geq 1\}$

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	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:

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

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:

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

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 $f^{(k)} : \mathbb{N}_+^k \rightarrow \mathbb{N}$, 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 ,

Primeri:

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

Skica stroja, ki računa $m + n$:

- $0^m 10^m$ - 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?

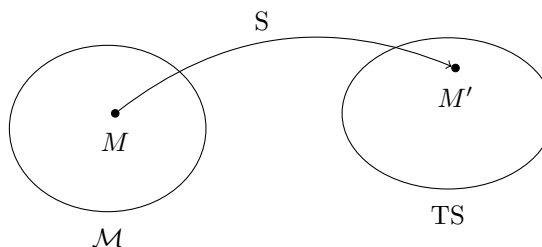
Skica stroja, ki računa $m * n$:

- $0^m 10^n$ - vhodna beseda
- $0^m 10^n 1$ - premakni se na konec in zapiši 1 (ločnica za rezultat)
- $B0^{m-1}10^n 1$ - premakni se na začetek in izbriši 0
- $B0^{m-1}10^m 10^n$ - prekopiraj n ničel za ločnico (in ničle)
- $B^m 10^m 10^{m*n}$ - 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 in alternative Turingovemu stroju

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, 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:
 - $\delta(\langle q_0, B \rangle, 0) = \langle \langle q_1, 0 \rangle, 0, D \rangle$
 - $\delta(\langle q_0, B \rangle, 1) = \langle \langle q_1, 1 \rangle, 1, D \rangle$

- 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.



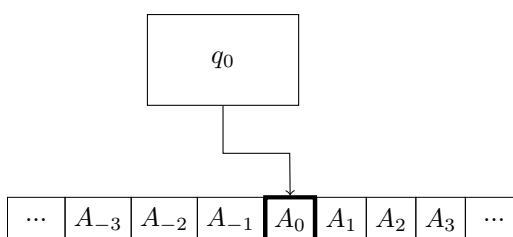
Stroj je definiran skoraj enako kot osnovni Turingov stroj, le funkcija prehodov δ je enostavnejša, saj ni treba skrbeti, kaj se zgodi, če zadanemo levi rob, kot pri običajnem Turingovem stroju.

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

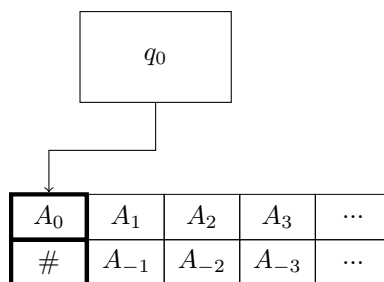
Dokaz: Stroj se lahko vede kot da je omejen na levi. Na začetku izvajanja se premaknemo levo, zapišemo poseben znak, ki nam pomeni konec traku. Nato se premaknemo desno in stroj normalno izvajamo.

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 :

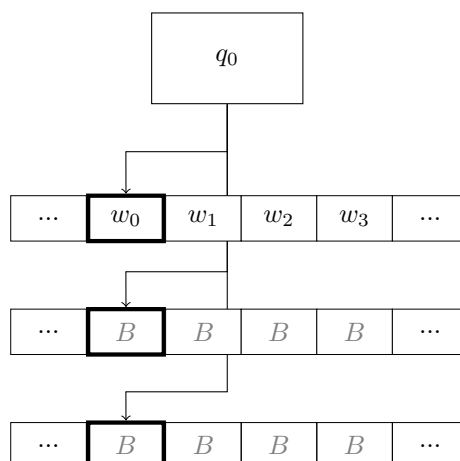


Stroj M' deluje tako:

- naenkrat 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 strani. Poleg tega ima vsak trak svoje okno, ki se lahko neodvisno od ostalih premika ob vsakem koraku. Spet imamo na začetku vhodno besedo na prvem traku in je okno prvega traku na prvem znaku vhodne besede, na ostali trakovi pa so prazni.



Def.: Korak stroja δ opišemo kot:

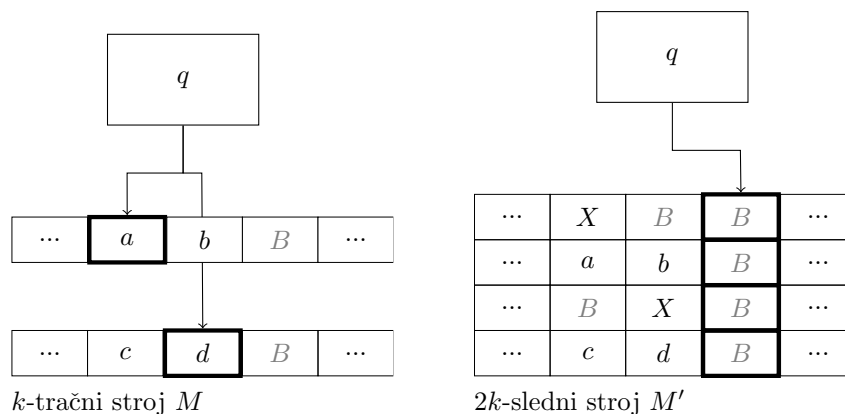
$$\delta = Q \times \Gamma^k \rightarrow Q \times (\Gamma \times \{L, D, -\})^k$$

Torej, na vsakem koraku dobimo iz trenutnega stanja, ter tračnega simbola 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): 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 .



Stroj M' torej hrani trenutni položaj na trakovih s 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 naredil stroj M

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

4.4.7 Nedeterministični Turingov stroj

Za δ sedaj velja: $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, D, -\}}$

Torej ima pri nekem stanju in tračnem znaku tak TS več kot le eno trojko iz $Q \times \Gamma \times \{L, D, -\}$. Toda teh trojk je 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 to ni možno, se bo stroj ustavil ali pa bo mletel v neskončnost.

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

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

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

Trditev: Nedeterministični Turingov stroj ni močnejši od osnovnega modela.

Dokaz: Simulacija NTS z osnovnim:

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

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

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

Natančneje:

- na drugi sledi sestavi novo, naslednje navodilo
- prepiše vhodno besedo s prvega na tretjo sled
- na tretji sledi oponaša stroj M , kot da bi ta izbral svoje poteze po tekočem navodilu.

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 ša se ustavi v končnem stanju

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

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

od tu naprej je neurejeni del zapiskov.. halp!

4.4.8 Večdimenzionalni Turingov stroj

... $k_L=2$ 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

4.4.9 Rekurzivne funkcije

Definiral je funkcije: ničelna, naslednik, projekcija. Dodal je pravilo 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 funkcije, 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
- $\pi(n_1, n_2, \dots, n_k) = n_i \forall n..$ - 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 $f(n_1, n_2, \dots, n_k)$ po def $g(h_1(n_1, n_2, \dots, n_k), \dots, h_m(n_1, n_2, \dots, n_k))$ sestavljena s kompozicijo funkcij 2. Če sta dani funkciji $g : N^k \leftarrow N$ in $h : N^{k+1} \leftarrow n$, potem je f definirana z:

$$f(n_1, n_2, \dots, n_k, 0) \text{ po def } g(n_1, n_2, \dots, n_k)$$

$$f(n_1, n_2, \dots, n_k, m+1) \text{ po def } h(n_1, n_2, \dots, n_k, m), f(n_1, n_2, \dots, n_k, m)), \text{ } zam \geq 0$$

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, \dots, n_k obstaja n ..neki.. m , da je $g(n_1, n_2, \dots, n_k, m) = 0$, potem je funkcija:

$$f(n_1, n_2, \dots, n_k) \text{ po def } \mu x g(n_1, \dots, n_k, x)$$

sestavljena z μ -operacijo iz funkcije g .

Konstrukcija rekurzivne funkcije f je končno zaporedje f_1, f_2, \dots, 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 konstrukcija 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, \dots, 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 $f(g(\dots), g_k(\dots))$ 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 $f(n_1, n_2, \dots, n_k)$ 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 imenuje splošno rekurzivna

Povzetek: Algoritem po Herbrand-Gödlu (HG) je $\varepsilon(f)$.

Računanje po HG je izračun vrednosti funkcije $f(n_1, \dots, n_k)$ iz $\varepsilon(f)$ z uporabo pravil 1,2.

Funkcija je izračunljiva po HG, če je splošno rekurzivna.

4.4.10 λ -račun

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

Kako je funkcija opisana? Začetni λ -term

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

To preoblikovanje dosežemo z uporabo t.i. redukcij. Ta

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

$t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_K$ $f, n_1, n_2, \dots, n_k \dots f(n_1, \dots, n_k)$

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.

4.4.11 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.

4.4.12 Algoritmi Markova

Imamo abecedo Σ , končno zaporedje produkcij: $x_1 \leftarrow y_1 \ x_2 \leftarrow y_2 \ \dots \ x_n \leftarrow y_n \ x, y \in \Sigma^*$

Produkcija 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

4.5 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: „algoritem” \iff algoritem po Turingu (ali po katerem izmed ostalih ekvivalentnih modelov)

Kaj je doprinos Church-Turingove teze? algoritem \bullet - krog... alg, kot so ga opisali (in okrog napisani $T_i =_i P_i =_i GK_i =_i HG_i =_i M$)

”algoritem \ll — $_i$ program TS

računanje \bullet - krog... računanje, kot so ga opisali...

”računanje \ll — $_i$ delovanje TS

izračunljiva funkcija \bullet - krog... izr. funkcija, kot so jo opisali...

”izračunljiva $f \ll$ — $_i$ f izračunljiva s TS

Težava: ...

Težave s totalnimi funkcijami

$f : \mathbb{N} \rightarrow \mathbb{N}$

?: Kako na splošno dokazati, da je neka funkcija res totalna?

V najslabšem primeru, bi bilo treba preizkusiti za vsak $x \in A$, ali je funkcija pri tem x definirana (pišemo: $f(x) \uparrow$)

Če je A neskončen, ta način ni finiten.

= $_i$: Pojem izračunljivosti funkcije se opira na nek algoritmično težko določljiv pojem totalnosti

Po CT tezi je izračunljiva ... rekurziva funkcija

- moč $N^k : N$ je c (moč realnih števil)
- rekurzivnih funkcije je \aleph_0 (števno mnogo)
- Vsako definira program TS, programov pa je števno mnogo.
- ? ali je med vsemi funkcijami ($\mathbb{N}^{\mathbb{N}}$) možno najti tako, ki je izračunljiva, pa ni rekurzivna.
- DA – dokaz z diagonalizacijo:

Trditev: Obstaja izračunljiva funkcija, ki ni rekurzivna.

Dokaz: Definicije funkcij so končna zaporedja.

uredimo zaporedja po dolžini, enako dolga pa leksikografsko

\Rightarrow lahko govorimo o prvem, drugem, ..., o n -tem programu TS. Zato lahko tudi prvi, drugi, ... n -ti rekurzivni funkciji.

označimo n -to rekurzivno funkcijo s f_n

definirajmo

$g(n, a_1, \dots, a_k)$ po def. $f_n(a_1, a_2, \dots, a_k) + 1$, kjer $a_i \in \mathbb{N}$

funkcija je izračunljiva

algoritem je

....

ali je g tudi rekurzivna

ali obstaja program TS, ki je izračuna.

predpostavka: g je rekrivna

Tedaj obstaja nek naravni m , da je $f_m = g$.

Poglejmo vrednost $g(m, m, \dots, m)$

iz (*) sledi $g(m, m, \dots, m) = f_m(m, m, \dots, m)$

iz (**) sledi $f_m(m, m, \dots, m) = g(m, m, \dots, m)$

iz tega sledi $f_m(m, m, \dots, m) = f_m(m, m, \dots, m) + 1$

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 CT 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 $f_m(mmmm) \uparrow$.

Sklep: Funkcija je "izračunljiva" – če f računa nek TS tam, kejr je definirana

4.6 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. Kdaj je to koristno?

Kodiranje Turingovih strojev

Kako poljuben Turingov stroj zakodirati z abecedo $\{0, 1\}$? Zadošča da zakodiramo program δ Turingovega stroja. Naj bo $T = \langle Q, \Sigma, \Gamma, \delta, q_1, B_1, q_f \rangle$ poljuben stroj. Če je $\delta(q_1, a_j) = \langle q_k, a_l, S_m \rangle$ ukaz programa δ , ga zakodiramo kot:

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

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

$$\langle T \rangle = 111K_111K_211 \dots 11K_r111$$

Na $\langle T \rangle$ lahko gledamo kot da je dvojiški zapis nekega naravnega števila in to je indeks Turingovega stroja T .

Nekatera naravna števila niso indeksi TS, zato se dogovorimo: če naravno število nima oblike $*$, rečemo, da je indeks praznega Turingovega stroja (njegova δ je povsod nedefinirana – takoj se ustavi in ne sprejme nobene besede) Posledica: vsako naravno število, je indeks natanko enega Turingovega stroja. Obratno ne velja. Isti Turingov stroj ima več indeksov (dodajamo nepotrebne in nesmiselne ukaze, pa je.)

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

Dokaz: Stroj si zamislimo v grobem in intuitivno zapišemo njegov program. Skličemo se na CT tezo, ki nam zagotovi obstoj nekega konkretnega TS (ki opravlja ta algoritem)

Zamisel stroja U:

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 U(intuitivo)

- 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 $\delta(q_i, a) = \dots$
- Če je ne najdemo se U ustavi (tak T bi se ustavil)
- Denimo, da najdena koda opisuje ukaz $\delta(q_i, a) = \langle q, b, S \rangle \dots$ 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 - vhodni($\langle T \rangle, w$), delovni($www—a—www$), pomožni($\langle q_i \rangle, \langle q_f \rangle$)

To je bil intuitiven opis algoritma programa stroja U - algoritma. Po CT tezi lahko sestavimo pravi TS z vsemi podrobnostmi, ki izvajajo delo opisanega stroja.

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

To je univerzalni Turingov stroj.

4.6.1 Pravi stroji, ki so univerzalni

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

4.7 Reševanje računskih problemov

Vrste računskih problemov lahko delimo na:

- Odločitvene - odgovorijo na dano vprašanje z DA ali NE
- Iskalne - sprašujejo po elementu množice, ki ima neko lastnost
- Preštevalne - dajo število elementov v množici, ki imajo neko lastnost
- Naštevne - generirajo vse elemente v množici, ki imajo neko lastnost

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

4.7.1 Jezik odločitvenih problemov

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

S P označujemo odločitveni problem (npr. $P = \langle \text{„Ali je število } n \text{ praštevilo?“} \rangle$).

Ko v problem P vstavimo namesto n nek konkretni podatek dobimo nalogo p . (npr. $\langle \text{„Ali je 19871 praštevilo?“} \rangle$). Vsaka naloga ima odgovor DA ali NE, rečemo tudi, da je naloga pozitivna ali negativna. V opisu naloge so vedno konkretni podatki. Prek kodirne funkcije $\langle \rangle$ dobi vsaka naloga p neko kodo $\langle p \rangle \in \Sigma^*$, ki jo razume nek Turingov stroj. Naj bo $\langle \rangle : P \rightarrow \Sigma^*$. Zahtevamo, da je $\langle \rangle$ injektivna ($p \neq p' \Rightarrow \langle p \rangle \neq \langle p' \rangle$). Naj bo jezik problema P definiran kot:

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

• pravokotnik P s squigly črto po sredi... pozitivne/negativne naloge in pravokotnik Σ^* s podmnožico $\mathcal{L}(P)$ in puščica iz pozitivnih v $\mathcal{L}(P)$ prek kodirne funkcije $\langle \rangle$

Torej naloga $p \in P$ je pozitivna, $\Leftrightarrow \langle p \rangle \in \mathcal{L}(P)$. Računanje odgovora na nalogo p lahko zamenjamo z ugotavljanjem, ali je beseda $\langle p \rangle$ v jeziku $\mathcal{L}(P)$.

Def.: Naj bo P odločitveni problem. Pravimo, da je: P je odločljiv, kadar je jezik problema $\mathcal{L}(P)$ rekurziven. P je neodločljiv, kadar jezik problema $\mathcal{L}(P)$ ni rekurziven. P je poldločljiv, kadar je jezik problema $\mathcal{L}(P)$ Turingov(rekurzivno prešteven).

4.7.2 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: $\mathcal{K}_0 = \{\langle T, w \rangle \mid T \text{ se pri } w \text{ ustavi}\}$ Nekaj časa nas bo zanimal jezik $\mathcal{K} \subseteq \mathcal{K}_0$, ki ga dobimo iz \mathcal{K}_0 , če za w vzamemo kar kodo stroja, torej $\langle T, T \rangle$. $\mathcal{K} = \{\langle T, T \rangle \mid T \text{ se pri } \langle T \rangle \text{ ustavi.}\}$ 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 razpozna. $\exists TS : R_{\mathcal{K}}$, ki za poljuben T odgovori $R(\langle T, T \rangle)$

kjer je $R(\langle T, T \rangle) = \text{DA}$, če se T ustavi nad $\langle T \rangle$... NE, če se ne ustavi. Sestavimo nov Turingov stroj N s pomočjo domnevnega $R_{\mathcal{K}}$: \bullet 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 še enkrat 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: \bullet $\langle N \rangle$ 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: \mathcal{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 odločitveni problem, za katerega ni Turingovega stroja, ki bi ga vedno rešil. Turingov stroj je po CT-tezi algoritem, torej za nekatere probleme ni algoritma, ki bi ga rešil.

4.7.3 Primeri nerešljivih problemov

- \bullet 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 ustavitve 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 vseh vseh?“

Razpoznavanje jezikov:

„Ali Turingov stroj T razpozna rekurziven jezik?“,
 „Ali Turingov stroj T razpozna 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

$\varphi : A \rightarrow B$ 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 $p(x_1, x_2, \dots, x_n)$ celoštevilsko rešitev?“

Dana je množica matrik $\mathcal{M} = \{M_1, M_2, \dots, M_n\}$. Vse matrike so reda $n \times n$ in imajo celoštevilске 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 $T = \{ \bullet \text{ križ-kraž kvadrati z različno pobarvanimi četrtinami} \}$. 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 ?“

4.7.4 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.