Teoretične osnove računalništva

Zapiski predavanj 2010/2011

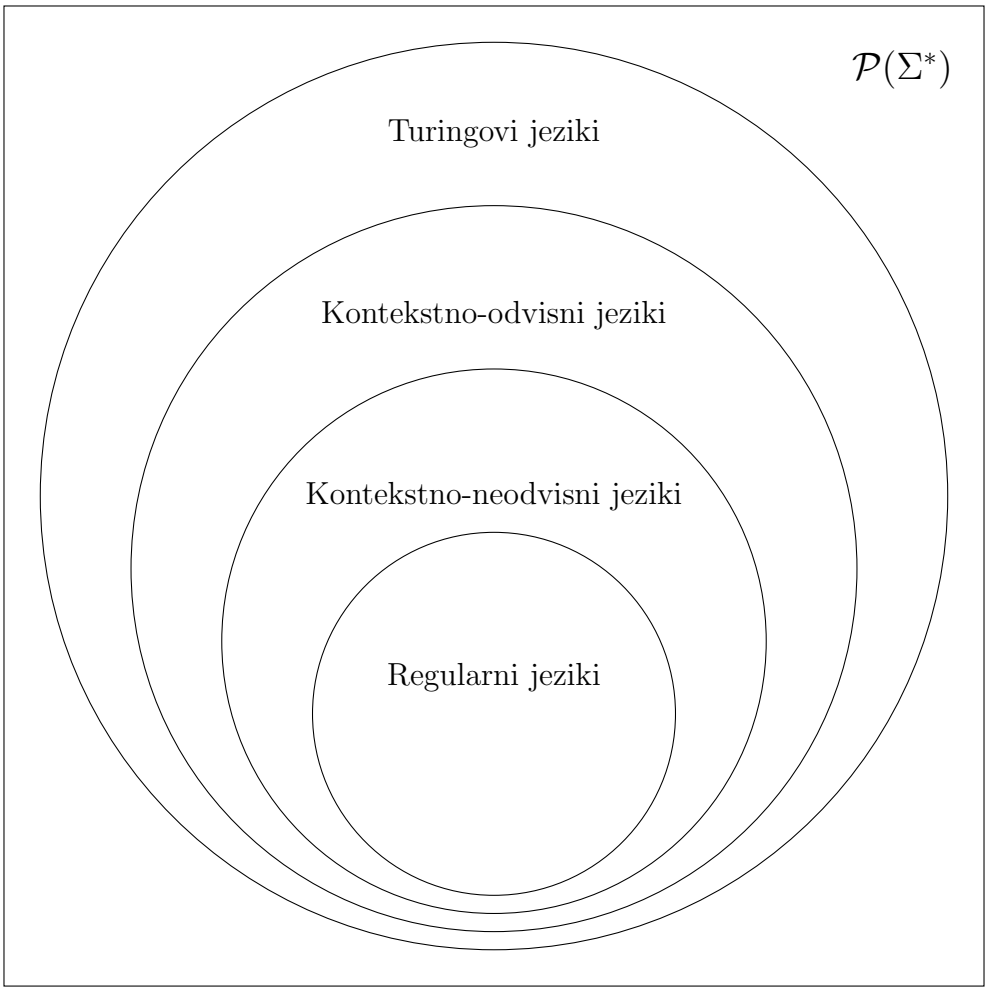
2. marec 2011

Kazalo

Poglavje 1

Uvod

1.1 Chomskyeva hiearhija jezikov



1.2 Matematične osnove

1.2.1 Dokazovanje

Dokaz s konstrukcijo

Dokaz obstoja nekega matematičnega objekta je to, da nam objekt uspe skonstruirati.

Primeri:

Primer 1: Za vsak $n > 4$, obstaja dvojiško drevo, ki ima natanko 3 liste.
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:
  - $|\mathbb{R}| \leq |[0,1)|$,
  - $|\mathbb{R}| \geq |[0,1)|$, ker velja $[0,1) \subset \mathbb{R}$
- 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[1]

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 $\frac{a}{b}$.

[1] http://en.wikipedia.org/wiki/Hilbert_curve

- 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.

Poglavje 2

Regularni jeziki

2.1 Uvod

Oznake

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

Operacije

- Stik
  - Stik nizov:

$$\begin{aligned} w &= w_1 w_2 \dots w_n \\ x &= x_1 x_2 \dots x_m \\ wx &= w_1 w_2 \dots w_n x_1 x_2 \dots x_m \end{aligned}$$

  - Stik množic:

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

- Potenciranje

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

- Iteracija

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

Regularni jezik

Def.: Regularni jezik L nad abecedo Σ je poljubna podmnožica Σ^*

$$L \subseteq \Sigma^*$$

Primeri:

Primer 1: Prazen jezik: $L_1 = \{\}$
Primer 2: Jezik, ki vsebuje ε (ni prazen): $L_2 = \{\varepsilon\}$
Primer 3: Jezik, ki vsebuje nize "a, aa, ab": $L_3 = \{a, aa, ab\}$

2.2 Regularni izrazi

Def.: Osnovni izrazi:

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

Def.: 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?

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

2.2.1 Jezik regularnih izrazov

Def.: Jezik ki ga opisuje poljubni regularni izraz, je regularni jezik.

Regularni jeziki ne vsebujejo informacije o prejšnjih znakih in se z njimi ne da opisati poljubnega jezika. Kasneje bomo za dokazovanje tega uporabljali lemo o napihovanju za regularne jezike (glej 2.6.1).

Primeri:

Primer 1: $\{\}$ je regularni jezik
Primer 2: $\{0^n1^n \mid n \geqslant 0\}$ <u>ni</u> regularni jezik

2.3 Končni avtomati

2.3.1 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 simbolom 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 $P(Q)$

2.3.2 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

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

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

Definirajmo še posplošeno funkcijo prehodov $\hat{\delta}$, ki nam pove, do katerega stanja pridemo po nekem nizu.

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

2.3.3 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

2.3.4 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.

2.3.5 Levo in desno-regularne gramatike

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

- V - množica spremenljivk oz. vmesnih simbolov, $V \subseteq \Sigma$
- T - množica znakov oz. končnih simbolov, $T \subset \Sigma$
- P - množica produkcij, $[\alpha_1 \to \alpha_2]$
- S - začetni simbol, $S \in V$

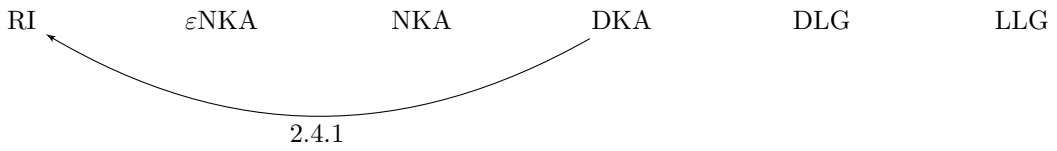
Pri tem pa regularne gramatike ločimo na levo in desno-regularne.

- Pri levih so produkcije $P \subset V \times ((V \cup \{\varepsilon\}) \cdot T^*)$
- Pri desnih so produkcije $P \subset V \times (T^* \cdot (V \cup \{\varepsilon\}))$

To pomeni, da imamo pri levo-regularnih gramatikah vmesne simbole lahko le na skrajni levi, pri desno-regularnih pa le na desni.

2.4 Prevedba med izvedbami regularnih jezikov

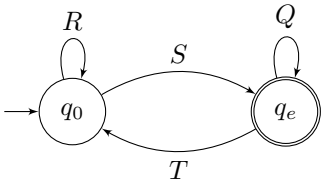
Regularni izrazi, regularne gramatike in končni avtomati so vsi enako močni in je mogoče pretvarjati med njimi. V tem odseku bomo predstavili naslednje prevedbe:



2.4.1 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,

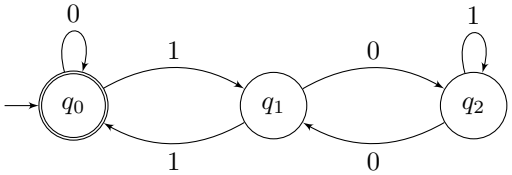


ki jih prepišemo v en sam regularni izraz oblike:

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

Primeri:

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



Regularni izraz dobimo po postopku iz 2.4.1:

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

2.5 Ohranjanje regularnosti jezikov

Regularnost jezika 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 oz. reverz
  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 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-regularne gramatike
- Dokažemo da ni regularen:
  - Z uporabo leme o napihovanju za regularne jezike (glej 2.6.1)
  - Pokažemo, da jezik ne spada niti v nek širši razred jezikov:
    - Dokažemo, da ni kontekstno-neodvisen (glej 3.3)

2.6.1 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:

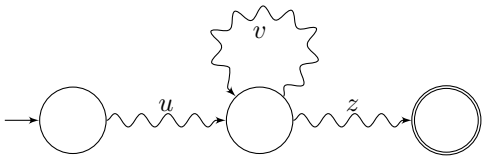
$$w = u\ v\ z$$

Pri čemer velja:

- $|uv| \leq n$
- $|v| > 0$

- $uv^iz \in L,\ \forall i \geq 0$ (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 $|v| = 0$.

Poglavje 3

Kontekstno-neodvisni jeziki

3.1 Kontekstno-neodvisne gramatike

3.2 Skladovni avtomati

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

- Q - končna množica stanj
- Σ - vhodna abeceda
- Γ - skladovna abeceda
- δ - funkcija prehodov, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$
- q_0 - začetno stanje, $q_0 \in Q$
- Z_0 - začetni skladovni simbol, $Z_0 \in \Gamma$
- F - množica končnih stanj

3.2.1 Trenutni opis

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

3.2.2 Relacija $\vdash$

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

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

3.2.3 Jezik skladovnega avtomata

3.3 Dokazovanje da je jezik kontekstno-neodvisen

3.3.1 Lema o napihovanju za kontekstno-neodvisne jezike

3.3.2 Ogdenova lema za kontekstno-neodvisne jezike

Poglavje 4

Kontekstno-odvisni jeziki

Prepoznajo jezik $L = \{a^n b^n c^n | n > 0\}$, imajo gramatike s produkcijami oblike $AaB \to \alpha_1 a \beta_2$.

Poglavje 5

Turingovi jeziki

5.1 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:

- GK (Kurt Gödel, Stephen Kleene)
- HG (Jacques Herbrand, Kurt Gödel)
- Produkcijski sistem (Emil Post),
- Lambda račun (Alonso Church, 1936)
- Turingov stroj (Alan Turing, 1936)

5.2 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

5.2.1 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.

Iz α_1 in α_2, 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

5.2.2 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: $x_1 \dots x_{i-2}x_{i-1}\ q\ x_i \dots x_n \vdash x_1 \dots\ x_{i-2}p\ x_{i-1}\ Y\ x_{i+1} \dots x_n$

5.2.3 Tranzitivna ovojnica $\vdash^*$ relacije $\vdash$

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.

5.3 Jezik Turingovega stroja

Def.: Jezik Turingovega stroja je definiran kot:

$$L(M) = \{w \mid w \in \Sigma^* \wedge q_0 w \vdash^* w_1\ q\ w_2 \wedge w_1, w_2 \in \Gamma^* \wedge q \in F\}$$

Z besedami to pomeni, da je $L(M)$ množica besed $w \in \Sigma^*$, ki če jih damo na vhod stroju M, povzročijo, da se stroj M v končno mnogo korakih znajde v končnem stanju.

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

5.3.1 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.

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:

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

5.3.2 Turingov stroj kot računalnik funkcij

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

$$0^{i_1}10^{i_2}1\ldots10^{i_k}$$

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 lahko izračunal neko funkcijo

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

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: $f^{(1)}, f^{(2)}, \ldots f^{(k)}$.

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

5.3.3 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: $\Gamma = \Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_k$.

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.

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