Saturs

1. Tjūringa Mašīnas

1.1. Variācijas

Var būt 3 veida uzdevumi: stāvokļu, tekstuāls, vairāklenšu.

> **i Čērča-Tjūringa tēze**
>
> Viss, ko var intuitīvi saukt par „algoritmu“ vai „efektīvu procedūru“, var tikt izpildīts ar Tjūringa mašīnu. Tā nav pierādāma teorēma, bet gan skaitļošanas modeļa definīcija, kurai līdz šim nav atrasts pretpiemērs.

1.1.1. Viena lente

$(q, a) \to (q', a', d)$ – stāvoklī q redzot a, ieraksta a' un iet virzienā d ($\leftarrow$ vai $\rightarrow$).

1.1.2. Divas (vai vairākas fiksētas) lentes

$(q, a_1, a_2) \to (q', b_1, b_2, d_1, d_2)$ – a_1, b_1, d_1 pirmai lentei un a_2, b_2, d_2 otrai lentei. Svarīga atšķirība ir ka vairāklenšu TM papildus $\leftarrow$ un $\rightarrow$ virzieniem ir $\downarrow$ (stāvēšana uz vietas).*

1.1.3. Stāvēšana uz vietas

Nosimulēt stāvēšanu uz vietas jeb $d = 0$ var šādi:

- $(q, a) \to (q_{\text{new}}, a', \rightarrow)$
- $(q_{\text{new}}, a/b/c/*) \to (q_{\text{new}}, a/b/c/*, \leftarrow)$

1.1.4. Modelis, ko pamatā izmanto šajā kursā!

Šajā kursā fokusējas uz TM, kas ir:

1) Vienas lentes (skat. 1.1.1);
2) Bezgalīga vienā virzienā – pa labi;
3) Pirmais simbols ir tukšais simbols (_);
4) Pēc pirmā simbola ir ievade;
5) Pēc ievades ir bezgalīgs tukšu simbolu skaits;
6) Sāk uz pirmā ievades simbola (viens pēc pirmā tukšuma).

1.2. Risinājuma shēma

1) Izdomāt, kā aizstājot simbolus ar $*$ var pārbaudīt virknes derību.
2) Atcerēties par secību – aiz a var sekot tikai b/c, aiz b var sekot tikai c, utt.
3) Doties katrā no virzieniem var doties arī līdz galam jeb tukšumam _.
4) Vairāklenšu TM pārraksta pirmo daļu līdz # uz otras lentes un salīdzina.

Tas ir bieži sastopamais formāts, bet var gadīties arī cits.

1.3. Piemērs ($a^n b^n c^n$, kur $n \geq 0$)

Vai ieejas virknē $a^n b^n c^n$, kur $n \geq 0$

$(q_1, a) \to (q_2, *, \rightarrow)$
$(q_1, b/c) \to q_{\text{rej}}$
$(q_1, *) \to (q_1, *, \rightarrow)$
$(q_1, \_) \to q_{\text{acc}}$

$(q_2, a) \to (q_2, a, \rightarrow)$
$(q_2, b) \to (q_3, *, \rightarrow)$
$(q_2, c) \to q_{\text{rej}}$
$(q_2, *) \to (q_2, *, \rightarrow)$
$(q_2, \_) \to q_{\text{rej}}$

$(q_3, a) \to q_{\text{rej}}$
$(q_3, b) \to (q_3, b, \rightarrow)$
$(q_3, c) \to (q_4, *, \leftarrow)$
$(q_3, *) \to (q_3, *, \rightarrow)$
$(q_3, \_) \to q_{\text{rej}}$

$(q_4, a/b/c/*) \to (q_4, a/b/c/*, \leftarrow)$
$(q_4, \_) \to (q_1, \_, \rightarrow)$

1.4. Piemērs (vai virkne atkārt. pēc #)

Vai ieejas virkne $x\#x$, kur $x \in \{0, 1\}^*$

$(q_1, 0, \_) \to (q_1, 0, 0, \rightarrow, \rightarrow)$
$(q_1, 1, \_) \to (q_1, 1, 1, \rightarrow, \rightarrow)$
$(q_1, \#, \_) \to (q_2, \#, \_, \downarrow, \leftarrow)$

---

*Derīgs ar uzdevumiem, kur palīdz kopēšana/salīdzināšana.

$(q_2, \#, 1) \to (q_2, \#, 1, \downarrow, \leftarrow)$
$(q_2, \#, 0) \to (q_2, \#, 0, \downarrow, \leftarrow)$
$(q_2, \#, \_) \to (q_3, \#, \_, \to, \to)$

$(q_3, 0, 0) \to (q_3, 0, 0, \to, \to)$
$(q_3, 1, 1) \to (q_3, 1, 1 \to, \to)$
$(q_3, 0, 1) \to q_{\text{rej}}$
$(q_3, 1, 0) \to q_{\text{rej}}$
$(q_3, 0/1, \_) \to q_{\text{rej}}$
$(q_3, \_, 0/1) \to q_{\text{rej}}$
$(q_3, \_, \_) \to q_{\text{acc}}$

2. Sanumurējamība

2.1. Definīcija

- Bezgalīgas kopas A, B ir vienāda izmēra, ja ir bijekcija $(1:1$ attiecība$)$ $F : A \to B$.
- A ir sanumurējama ar atkārtojumiem, ja ir attēlojums $F : N \to A$, kas par katru $a \in A$ attēlo vismaz vienu $x \in \mathbb{N}$.

> **! Teorēma**
>
> A ir sanumurējama ar atkārtojumiem tad un tikai tad, ja A ir sanumurējama.

2.2. Sanumerātības pierādījums

- Kopa ir sanumurējama, ja tajā eksistē bijekcija starp kopas elementiem un naturāliem skaitļiem. Citos vārdos sakot, katram kopas elementam var piešķirt unikālu naturālu skaitli.
- Ja kopa ir galīga, tā ir triviāli sanumurējama, jo katram elementam var piešķirt unikālu naturālu skaitli.
- Ja kopa ir bezgalīga (ar ko bieži vien darbojamies), jāņem vērā divi varianti:
  - Ja var izveidot bijekciju starp kopu un naturāliem skaitļiem, tad jāpierāda, ka bijekcija pastāv – Var skaidri definēt funkciju, kas katram kopas elementam piešķir unikālu naturālu skaitli un parādīt, ka tā apvieno visus elementus bez dublēšanās.
  - Ja nevar atrast bijekciju starp kopu un naturāliem skaitļiem, tad jāpierāda, ka bijekcija nepastāv. Parasti tas tiek darīts, izmantojot pierādīšanas tehnikas, piemēram, diagonālināciju vai pretrunu (sk. 2.2.2, 2.2.3).
- Ja izdodas pierādīt, ka start kopu un naturāliem skaitļiem nav bijekcijas, tad kopa ir nesaskaitāma.

2.2.1. Algoritmiska sanumurētība (par sanumerātību)

- Kopa A ir sanumurējama, ja $A = \{x_1, x_2, ...\}$
- Kopa A ir algoritmiski sanumurējama, ja ir Tjūringa mašīna, kas izdod virkni $x_1, x_2, ...,$ kurai $A = \{x_1, x_2, ...\}$

Divu lenšu TM, kur viena ir klasiska darba lente (DL) un otra ir izvada lente (IL) (tikai rakstīšanai).

2.2.1.1. Piemērs $\left(a^k b^k \,\middle|\, k \geq 0\right)$ alg. sanum.

Pamatot, ka kopa $\left\{a^k b^k \,\middle|\, k \geq 0\right\}$ ir algoritmiski sanumurējama.

1) Uzraksta uz izejas lentes tukšu vārdu.
2) Uzraksta vienu a uz darba lentes.
3) Atkārto:
   a) Uz ieejas lentes uzrakstām tikpat a, cik bija uz DL
   b) Uz izejas lentes uzrakstām tikpat b, cik a bija uz DL
   c) Uz izejas lentes uzrakstām _;
   d) Uz darba lentes pierakstām klāt vienu a.
4) Izejas lente $= \varepsilon, ab, aabb, aaabbb, ...$

2.2.1.2. Piemērs (atkārt. pēc # ir sanum.?)

Pamatot, ka kopa $\{x\#x \,|\, x \in \{a, b\}^*\}$ ir algoritmiski sanumurējama.

1) Uz darba lentes iet cauri visiem x (šeit būtu jāparāda/jāpaskaidro, kā to izdara).
2) Katram x uz izejas lentes uzraksta $x\#x$.
3) Uz izejas lentes uzraksta $\#\_$.
4) Uz darba lentes uzraksta a.
5) Atkārto:
   a) Pārraksta darba lentes saturu x uz izejas lenti;
   b) Uzraksta # uz IL, vēlreiz pārraksta x uz IL, uzrakstām _ uz IL.
   c) Uz DL nomaina x pret nākošo vārdu.

2.2.2. Diagonālinācija (pret sanumurējāmību)

Ir saraksts (pieņem, ka ir iegūts saraksts) vai uzskaitījums ar visiem kopas elementiem un tiek konstruēts jauns elements, kas neatrodas šajā sarakstā. Tas demonstrē, ka kopa ir nesanumurējama, jo vienmēr var izveidot jaunu elementu, kas nav iekļauts uzskaitē (piemēram, reālos skaitļos).

2.2.3. Pretruna (pret sanumurējāmību)

Pieņem, ka kopa ir saskaitāma un tad rodas pretruna. To var izdarīt, parādot, ka kaut kādas kopas īpašības vai kardinalitāte ir pretrunā ar sanumurējamības pieņēmumu.

2.3. Piemērs ($\mathbb{Z}$ sanumurētība)

Vai visu veselo skaitļu kopa $\mathbb{Z} = \{..., -1, 0, 1, ...\}$ ir sanumurējama? Vai ir $F : \{F(1), F(2), ..., F(n), ...\} = \mathbb{Z}$?

$F(n) = \frac{n}{2}$, ja pāra un $-\frac{n-1}{2}$ ja nepāra.

Sākumā tiek iegūta nulle.

Iterējot par n:

1) Funkcijas vērtības skaitļa modulis palielinās par 1, tiek iegūts pozitīvais skaitlis;
2) Tiek iegūts negatīvais skaitlis;

$F(1) = 0, F(2) = 1, F(3) = -1, F(4) = 2, F(5) = -2...$

Injekcija: ja $F(n_1) = F(n_2)$ no formulas seko, ka $n_1 = n_2$.

Surjekcija: katram $z \in \mathbb{Z}$ eksistē $n \in \mathbb{N}$, ka $F(n) = z$.

3. Redukcijas

3.1. Definīcija

- $A \leq B$, ja ir ar Tjūringa mašīnu izrēķināms pārveidojums
  - R: (ieejas dati A) $\rightarrow$ (ieejas dati B),
  - $B(R(x)) = A(x)$.

Ja $A \leq B$ – ja var atrisināt B, tad var atrisināt A. A ir reducējuma par B.

Ja $A \leq B \wedge A \geq B$, tad A un B ir ekvivalentas.

3.2. Piemērs (var vai nevar reducēt)

? Uzdevums

Dota problēma $\text{HALTING}_2(M, x, y) = 1$, kur Tjūringa mašīna M apstājas vismaz uz vienas no ievadēm x vai y. Pierādīt, ka to var vai nevar reducēt uz HALTING, tas ir parādīt (HALTING 2 $\leq$ HALTING).

Lai pierādītu, ka problēmu $\text{HALTING}_2(M, x, y)$ var noreducēt uz HALTING, mums jāparāda, ka varam konstruēt Tjūringa mašīnu, kas atrisina HALTING_2, izmantojot HALTING kā atrisinātu problēmu (jeb kā apakšprogrammu).

Pieņemsim, ka mums ir Tjūringa mašīna H, kas atrisina HALTING problēmu. Konstruēsim jaunu Tjūringa mašīnu H_2, kas atrisina HALTING_2 problēmu:

Tjūringa mašīna H_2 darbojas sekojoši:

- Doti ievades dati M, x un y.
- Palaiž H ar ievaddatiem (M, x).
- Ja H akceptē (M, x), apstājas un akceptē.
- Ja H noraida (M, x), palaiž H ar ievaddatiem (M, y).
- Ja H akceptē (M, y), apstājas un akceptē.
- Ja H noraida (M, y), apstājas un noraida.

Konstruējot H_2 šādā veidā, mēs simulējam H darbību uz abām ievadēm x un y:

- Ja H akceptē (M, x) vai (M, y), H_2 akceptēs un apstāsies.
- Ja H noraida gan (M, x), gan (M, y), H_2 noraidīs un apstāsies.

Tālākais teksts nav obligāts risinājumā.

Redukcijas analīze:

- Ja $\text{HALTING}_2(M, x, y) = 1$, tas nozīmē, ka Tjūringa mašīna M apstājas vismaz uz vienas no ievadēm x vai y. Šajā gadījumā H_2 arī apstāsies un akceptēs, jo tā veiksmīgi simulē H uz abām ievadēm un akceptē, ja H akceptē kādu no tām. Tādējādi HALTING_2 tiek reducēta uz HALTING.
- Ja $\text{HALTING}_2(M, x, y) = 0$, tas nozīmē, ka Tjūringa mašīna M neapstājas ne uz x, ne uz y. Šajā gadījumā HALTING_2 arī neapstāsies un noraidīs, jo tā simulē H uz abām ievadēm un noraida, ja H norada abas.

Tādējādi HALTING_2 tiek reducēta uz HALTING.

4. Neatrisināmas (non-decidable) problēmas

4.1. Definīcija

Neatrisināma problēma ir problēma ir problēma, kurai neeksistē TM, kas atrisinātu šo problēmu.

4.2. Funkcionālās īpašības

- $F(M)$, M – Tjūringa mašīnas programma;
- F var definēt caur M atbildēm uz dažādām ieejas virknēm x.

Piemēram:

- $\text{ONE}(M) = 1$, ja $\exists x : M(x) = 1$
- $\text{INFINITE}(M) = 1$, ja $\exists^{\infty} x : M(x) = 1$

Citiem vārdiem, ja $\forall x : M_1(x) = M_2(x)$, tad $F(M_1) = F(M_2)$.

4.3. Nefunkcionālās īpašības

- $A(M) = 1$, ja $\exists x : M$ apstājas pēc ≤ 17 soļiem.
- $B(M) = 1$, ja Tjūringa mašīnā M ir stāvoklis q, kurš netiek sasniegts nevienai ieejas virknei x.

Papildus Tjūringa mašīnas funkcionālam aprakstam, mums ir papildus informācija par mašīnu, par tās struktūru utt.

4.4. Raisa teorēma

> **Raisa teorēma**
>
> Ja F nav triviāla (ir $M : F(M) = 0$ un $M' : F(M') = 1$), tad F – neatrisināma.

Teorēma „pasaka“ mums priekšā, ka jebkuru netriviālu **funkcionālu** īpašību nevar atrisināt.

4.5. Uzskaitījums

- $\text{HALTING}(M\#x) = 1$, ja M apstājas, ja ieejas virkne $= x$.
- $\text{ACCEPTING}(M\#x) = 1$, ja M uz ieejas virknes izdod atbildi 1.
- $\text{EQUAL-ANSWERS}(M\#x\#y) = 1$, ja M uz ieejas virknēm x un y izdod vienādas atbildes.
- $\text{ONE}(M) = 1$, ja eksistē ieejas virkne x, uz kuras M izdod atbildi 1.
- $\text{INFINITE}(M) = 1$, ja eksistē bezgalīgi daudzas ieejas virknes x, uz kurām M izdod atbildi 1.
- $\text{EQUIV}(M1, M2) = 1$, ja $M_1(x) = M_2(x)$
- $\text{PCP}(S_1, S_2) = 1$ (Post-correspondance problem), ja divām galībām kopām $A = [a_1, a_2, ..., a_n]$ un $B = [b_1, b_2, ..., b_n]$ var izvēlēties indeksu secības $a_{i1}a_{i2}...a_{ik} = b_{i1}b_{i2}...b_{ik}$ (tā lai konkatenācijas būtu vienādas); domino kauliņi ar augšu un apakšu, ko saliekot kopā, globālai augšai un apakšai jāsakrīt.
- $\text{MORTAL-MATRIX}(S) = 1$, vai no matricām $M_1, M_2, ..., M_n$ var izveidot secību (ar atkārtojumiem), ka matricu reizinājums būtu 0.

> **Info**
>
> Ja var atrisināt ACCEPTING, tad var atrisināt arī HALTING.

> **Info**
>
> Ja var atrisināt EQUAL-ANSWERS / ONE / INFINITE / EQUIV, tad var atrisināt arī ACCEPTING.

4.6. Pierādījums no redukcijas (Soļi)

1) Skaidri definē problēmu, kuru vēlas pierādīt kā NP – norādot, kas ir derīga ievade un kādus rezultātus programmai paredzēts izvadīt.
2) Pieņem, ka eksistē algoritms vai TM, kas spēj atrisināt problēmu visiem iespējamiem ievaddatiem. Šī pieņēmuma mērķis ir rādīt pretrunu.
3) Definē citu problēmu, kas var tikt noreducēta līdz sākotnējai problēmai. Tas nozīmē, ka, ja var atrisināt sākotnējo problēmu, var atrisināt arī saistīto problēmu.
4) Izveido transformācijas vai redukcijas algoritmu, kas ņem saistītās problēmas instanci un pārveido to par sākotnējās problēmas instanci. Šai redukcijai vajadzētu saglabāt atbildi uz saistīto problēmu.
5) Pierāda, ka, ja sākotnējai problēmai ir risinājums, tad arī saistītajai problēmai ir risinājums. Šajā solī parasti ir jāpierāda, ka redukcijas algoritms ir pareizs un pareizi pārveido instances.
6) Pierāda, ka, ja saistītajai problēmai ir risinājums, tad arī sākotnējai problēmai ir risinājums. Šis solis parasti ietver pierādījumu, ka redukcijas algoritmu var atgriezt atpakaļ, lai iegūtu risinājumu sākotnējai problēmai.
7) Apvieno iepriekšējos soļus, lai parādītu, ka, ja sākotnējai problēmai ir risinājums, tad arī saistītajai problēmai ir risinājums. Šeit jārodas pretrunai.

4.7. Piemēri (prob. ir neatr)

4.7.1. HALTING $\leq$ ACCEPTING

- Teorēma: Ja var atrisināt ACCEPTING, tad var atrisināt arī HALTING
- Pierādījums: Attēlojums $R : M \to M'$ ar īpašību HALTING$(M\#x) =$ ACCEPTING$(M'\#x)$.
- Tad HALTING$(M\#x)$ var atrisināt sekojoši:
- izrēķina $M' = R(M)$, izrēķinām, vai ACCEPTING$(M'\#x)$.
- M' – programma, ko iegūst no M, visur aizstājot q_{rej} ar q_{acc}.
- Ja M akceptē/noraida, tad M' akceptē (izdos 1).
- Ja M neapstājas, M' arī neapstājas.

4.7.2. ACCEPTING $\leq$ EQUAL-ANSWERS

- Zināms: ACCEPTING nav atrisināma
- Pierādām: Ja var atrisināt EQUAL-ANSWERS, tad var atrisināt arī ACCEPTING.
- Secinām: EQUAL-ANSWERS nevar atrisināt.
- Dots: M, x
- Jādefinē: M', y : EQUAL-ANSWERS$(M'\#x\#y) =$ ACCEPTING$(M\#x)$.
- y – virkne no viena simbola s, kas nav vārdā x.
- Ja M' redz simbolu s, M' akceptē, citādi darbina M.
- $M'(\text{«}s\text{»}) = 1$, $M'(x) = M(x)$, ja x nesatur s.
- EQUAL-ANSWERS$(M'\#x\#\text{«}s\text{»}) =$ ACCEPTING$(M\#x)$.

4.7.3. ACCEPTING $\leq$ ONE

- Pierādām: Ja var atrisināt ONE, tad var atrisināt arī ACCEPTING.
- Dots: M, x

- Jādefinē: M' : ONE(M') = ACCEPTING($M\#x$).
- M': nodzēš no lentes ieejas virkni y, uzraksta x, palaiž M programmu.
- Jebkurai $y, M'(y) = M(x)$.

4.7.4. ACCEPTING $\leq$ INFINITE

- Pierādām: Ja var atrisināt INFINITE, tad var atrisināt arī ACCEPTING.
- Dots: M, x
- Jādefinē: M' : INFINITE(M') = ACCEPTING($M\#x$).
- Jebkurai $y, M'(y) = M(x)$.
- Ja $M(x) = 1$, tad $M'(y) = 1$ jebkurai y.

4.7.5. ACCEPTING $\leq$ EQUIV

- Pierādām: Ja var atrisināt EQUIV, tad var atrisināt arī ACCEPTING.
- Dots: M, x
- Jādefinē: M_1, M_2 : EQUIV(M_1, M_2) = ACCEPTING($M\#x$).
- Ja M akceptē x, M_1, M_2 jābūt ekvivalentām.
- Ja M neakceptē x, M_1, M_2 nav jābūt ekvivalentām.
- M_1: nodzēš ieejas virkni y, uzraksta x, darbina M.
- M_2: uzreiz pāriet uz akceptējoši stāvokli.
- Ja ACCEPTING($M\#x$) = 1, tad $M_1(y) = 1$ visiem y.

Tā kā $M_2(y) = 1$, tad EQUIV(M_1, M_2) = 1.

4.7.6. HALTING – pierādījums no pretējā

- Pieņemsism, ka ir TM M_H, kas risina HALTING.
- Definē M' šādā veidā:
  - Ieejas dati: x.
  - Atrod x atbilstošo Tjūringa mašīnas programmu M.
  - Ja $M_{H(M\#x)} = 1$, tad caur universālo TM darbina M uz x, izdod pretējo atbildi.
  - Ja $M_{H(M\#x)} = 0$, tad izdod 1.

Jebkuram $x : M'(x) \neq M(x)$, kur M – TM, kas atbilst x.

4.7.7. ACCEPTING – pierādījums no pretējā

- Pieņem, ka ACCEPTING ir atrisināma.
- $M(x)$:
  - Atrod ieejas virknei x atbilstošo M_i.
  - Ja ACCEPTING($M_i\#x$) = 1, M izdod 0.
  - Ja ACCEPTING($M_i\#x$) = 0, M izdod 1.

Jebkurai M_i, būs x, kuram $M(x) \neq M_{i(x)}$.

4.7.8. $A(M) = 1$, ja M – TM programma un, darbinot M uz tukšas ieejas virknes, tā apstājas un izdod 1

Pieņemsim, ka eksistē algoritms D problēmai $A(M)$. D ir algoritms, kas spēj noteikt, vai Tjūringa mašīna M, apstājas un atgriež 1 ar tukšu ievades virkni.

Tagad konstruēsim jaunu TM, N:

1) Palaiž M ar tukšu ievades virkni.
2) Ja M apstājas un atgriež 1, N apstājas un atgriež 1.
3) Ja M apstājas, bet neatgriež 1, N ieiet bezgalīgā ciklā.
4) Ja M neapstājas, N apstājas un atgriež 0.

Dodot algoritmam D ievadi N notiks sekojošais:

- Ja D atgriež 1, tas nozīmē, ka M apstājas un atgriež 1 ar tukšu ievades virkni (atbilstoši $A(M)$ definīcijai). Šajā gadījumā N apstāsies un atgriezīs 1, un D būs devis pareizu atbildi.
- Ja D atgriež 0, tas nozīmē, ka vai nu M neapstājas, vai M apstājas, bet neatgriež 1 ar tukšu ievades virkni. Pirmajā gadījumā N apstāsies un atgriezīs 0, kas atbilst D dotajai atbildei. Tomēr otrajā gadījumā N ieies bezgalīgā ciklā, un D būs devis nepareizu atbildi.

Tā kā D uz N sniedz nepareizu atbildi vienā no gadījumiem, mēs varam secināt, ka problēma $A(M)$ ir NP. Tāpēc neeksistē algoritms, kas spētu noteikt, vai dotā Tjūringa mašīna apstājas un atgriež 1 ar tukšu ievades virkni visām iespējamām Tjūringa mašīnām.

! Kuka-Levina teorēma

SAT problēma (Boolean satisfiability problem) ir NP-pilna.

i Info

SAT problēma: dots Būla algebras izteikums, vai ir iespējams piešķirt mainīgajiem vērtības (patiess/aplams) tā, lai viss izteikums būtu patiess?

Šī teorēma bija pirmā, kas pierādīja kādas problēmas NP-pilnību. Pēc tās, lai pierādītu, ka cita problēma L ir NP-pilna, pietiek parādīt, ka $L \in \text{NP}$ un ka $\text{SAT} \underset{p}{<} L$ (vai jebkura cita zināma NP-pilna problēma).

4.8. Daļēja atrisināmība

- A – daļēji atrisināma, ja ir Tjūringa mašīna T:
  - Ja $A(x) = 1$, tad $T(x) = 1$.
  - Ja $A(x) = 0$, tad $T(x) = 0$ vai $T(x)$ neapstājas.

! Teorēma

A – daļēji atrisināma tad un tikai tad, ja A – algoritmiski sanumurējama.

Cits nosaukums daļējai atrisināmībai ir atpazīstamība (angl. recognizability/ recognizable).

4.8.1. Piemērs (daļēji neatrisināma)

Problēma A, kurai neviena no A, $\overline{A}$ nav daļēji atrisināma?

- $\text{EQUIV}(M_1, M_2) = 1$, ja $\forall x : M_1(x) = M_2(x)$.
- $\overline{\text{EQUIV}}(M_1, M_2) = 1$, ja $\exists x : M_1(x) \neq M_2(x)$.

5. Nekustīgo punktu teorija

5.1. Nekustīgā punkta teorēma

Lai φ_x ir daļēji definēta funkcija, ko aprēķina Tjūringa mašīna ar programmu x. Lai $F : \Sigma^* \to \Sigma^*$ ir jebkurš visur definēts aprēķināms pārveidojums uz programmām.

Tad: eksistē$(x) : \varphi_{\{F(x)\}} = \varphi_x$

- φ_x: funkcija, ko aprēķina programma x.
- F: jebkura visur definēta aprēķināma funkcija virknēm (programmām).
- Σ^*: visu galīgo virkņu kopa pār alfabētu Σ.

5.2. NPT pielietošanas piemērs

„Pierādīt, ka eksistē programma, kas pieņem tikai savu aprakstu.“

Definējam skaitļojamu $F(x)$ ar ievadi – aprakstu x, kas pārveido to uz programmu, kas darbojas sekojoši:

1) Iegūst mašīnas ievadi y;
2) Programmai ir pieejams programmas apraksts x;
3) Iet cauri simboliem no x un y līdz nonāk līdz tukšumam:
   a) Ja simboli sakrīt un simboli nav tukšumi, iet tālāk.
   b) Ja abi simboli ir tukšumi, akceptē.
   c) Ja simboli nesakrīt, apstājas un neakceptē.

$F(x)$ ir vienmēr izskaitļojama, jo ja mums ir pieejama x programmas apraksts, ir iespējams uzbūvēt programmu, kas salīdzina ar brīvi izvēlamu ievadi y.

Pēc nekustīgā punkta teorēmas eksistē x, ka

$$\varphi_{F(x)} = \varphi_x.$$

1) Tātad eksistē tāds x, ka x ir funkcionāli ekvivalenta $F(x)$.

2) Tā kā funkcijas $F(x)$ rezultāts ir iepriekš aprakstītā programma, x sakrīt ar uzdevumā prasīto uzvedību.

QED.

6. Sarežģītības teorija

6.1. Lielais O un mazais o

Notācija, kas tiek izmantota, lai raksturotu **funkciju** sarežģītību asimptotiski.

6.1.1. Lielais-O (formālā definīcija)

$f(n) \in O(g(n))$, ja:

$\exists C > 0, \exists n_0 > 0 : (\forall n \geq n_0 : f(n) \leq c \cdot g(n))$

Tas nozīmē, ka funkcija $f(n)$ asimptotiski nepārsniedz konstanti c reizinātu $g(n)$.

6.1.2. Mazais-o (formālā definīcija)

$f(n) \in o(g(n))$, ja:

$$\lim_{i \to \infty} \frac{f(n)}{g(n)} = 0$$

Tas nozīmē, ka funkcija $f(n)$ kļūst nenozīmīga attiecībā pret $g(n)$, n tiecoties uz bezgalību.

6.1.3. $f(n) \in O(g(n))$ pamatojuma triks

Ja ir pierādījums, ka $f(n) \in o(g(n))$, tad automātiski var secināt, ka $f(n) \in O(g(n))$. **Tikai pozitīvajā gadījumā!** Jo mazais o ir stingrāka prasība par lielo O.

6.1.4. Pamatojuma soļi

- Ja funkcija pielīdzināta lielajam O:
  1) Salīdzina funkcijas augstāko pakāpi ar doto O pakāpi.
  2) Ja funkcijas pakāpe ir lielāka, tad vienādojums būs patiess, jo funkcija aug straujāk.
  3) Korektam risinājumam jāpamato kāpēc definīcijas nevienādība ir patiesa visiem $n \geq n_0$ un iespējams jāparāda piemēra c.
- Ja funkcija pielīdzināta mazajam o:
  1) Jāievieto dotais robežā $\lim_{x \to \infty} \frac{f(x)}{g(x)}$;
  2) Rezultāts ir 0, patiess, citādi – nepatiess.

6.1.5. Piemērs (lielais-O)

$$2n^4 + 6n^2 + 17 \stackrel{?}{=} O(n^4)$$

Izteiksme ir patiesa, tā kā kreisās puses izteiksmes augstākā pakāpe jeb kārta ir 4 un iekš O tā arī ir 4.

6.1.6. Piemērs (lielais-O)

$$2n^4 + 6n^2 + 17 \stackrel{?}{=} O(n^3)$$

Izteiksme ir aplama, jo kreisajā pusē augstākā pakāpe ir 4, kamēr labajā ir norādīta 3, un 4 pakāpes izteiksmi nevar izpildīt $O(n^3)$.

6.1.7. Piemērs (lielais-O)

$$n^3 + 17n + 4 \stackrel{?}{\in} O(n^3)$$

Jā, $n^3 + 17n + 4 \leq n^3 + 17n^3 + 4n^3 = 22n^3$.

6.1.8. Piemērs (lielais-O)

$$n^4 + 17n + 4 \stackrel{?}{\in} O(n^3)$$

Nē $n^4 + 17n + 4 > n^4 = n \cdot n^3$

6.1.9. Piemērs (mazais-o)

$$n \log^4 n \stackrel{?}{=} o(n^{1.5})$$

Ir zināms, ka mazajā O notācijai, ja $\lim_{x \to \infty} \frac{f(x)}{g(x)}$, kur $f(x)$ ir funkcija un $g(x)$ ir o, tad vienādība izpildās. Ievietojot vērtības

$$\lim_{n \to \infty} \frac{n \log^4 n}{n^{1.5}} = 0$$

Tātad vienādojums ir patiess.

6.1.10. Piemērs (mazais-o)

$$2^n n^2 \stackrel{?}{=} o(n^3)$$

Pēc tās pašas aprakstītās īpašības, kā 6.1.9, sanāktu

$$\lim_{n \to \infty} \frac{2^n n^2}{3^n}$$

un tā kā 3^n aug ātrāk kā 2^n, šī robeža būs 0 un sākotnējais vienādojums būs patiess.

6.2. Sarežģītības klases

6.2.1. Laiks (TIME)

TIME($f(n)$) – problēmas L, kurām eksistē Tjūringa mašīna M, kas pareizi risina L un izmanto $O(f(n))$ soļus.

6.2.1.1. TM darbības laiks

1) Identificēt galvenās operācijas vai soļus, ko Tjūringa mašīna veic katrai ievadei.
   a) Tas varētu ietvert simbolu lasīšanu no lentes, simbolu rakstīšanu lentē, lentes galviņas pārvietošanu, aprēķinu veikšanu vai lēmumu pieņemšanu, pamatojoties uz pašreizējo stāvokli un ievades simbolu.
2) Noteikt sliktākā gadījuma scenāriju.
   a) Analizēt ievadi vai ievades secību, kas prasītu maksimālo soļu skaitu, lai Tjūringa mašīna pabeigtu savu aprēķinu.
   b) Apsvērt ievades, kas maksimizē iterāciju skaitu vai liek mašīnai izpētīt visas iespējamās aprēķinu zaru versijas.
3) Izteikt darbības laiku atkarībā no ievades izmēra.
   a) Definēt funkciju, kas attēlo Tjūringa mašīnas veikto soļu vai pāreju skaitu kā funkciju no ievad izmēra.
   b) Piemēram, ja ievades izmērs ir n, darbības laika funkciju varētu apzīmēt kā $f(n)$.
4) Vienkāršot darbības laika funkciju un izsakot to, izmantojot lielā O notāciju.
   a) Lielā O notācija nodrošina augšējo robežu darbības laika funkcijas pieauguma ātrumam, palielinoties ievaddatu izmēram.
   b) Noņemt konstantes faktorus un zemākas kārtas locekļus no darbības laika funkcijas, lai koncentrētos uz dominējošo locekli, kas atspoguļo pieauguma ātrumu.
5) Izteikt vienkāršoto darbības laika funkciju, izmantojot atbilstošo lielā O notāciju, piemēram, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$ utt.

6.2.1.2. Programmas darbības laiks

Īstām programmām rindiņas var izpildīties atšķirīgā laikā – tas ir atkarīgs apakšprogrammu izsaukumi, procesora operācijām. Taču šajā tas tiek abstrahēts un katra koda rindiņa tiek uzskatīta par vienu soli.

Lai atrastu koda izpildes laiku:

1) Novērtē katras rindiņas „globālo laiku“ – cik tā izpildās ņemot vērā visus ciklus, izmantojot summu funkcijas.
2) Novērtē kādos gadījumos tā izpildās, ja vispār izpildās.
3) Izvērtē to kāds gadījums (ievade) būtu vissliktākā un aprēķina tam funkciju no n (skat. 9.5).
4) Novērtē šīs funkcijas klasi izmantojot lielā-O notāciju.

6.2.1.2.1. Piemērs ($|a| \stackrel{?}{=} |b|$)

Vai ieejas virknē ir vienāds skaits a un b?

1) Virzās no kreisās puses uz labo, aizstājot vienu a un vienu b ar x;
2) Ja neatrod nedz a, nedz b, akceptē;
3) Ja neatrod vienu no a vai b, noraida;
4) Ja atrod gan a, gan b, virzās atpakaļ uz pirmo simbolu un atkārto.

Kopējais soļu skaits:

- Ne vairāk kā $(\frac{n}{2} + 1)2n = n^2 + 2n$ soļi.
- Ja n nav ļoti mazs, n^2 būs vairāk nekā $2n$ un soļu skaits $O(n^2)$.

6.2.2. Telpa/Vieta (SPACE)

6.2.2.1. SPACE Definīcija (Precīzs modelis)

- Ieejas lente - $x_1, ..., x_n$, var tikai lasīt.
- Darba lente – sākumā tukša, var arī rakstīt.
- $S(x_1, ..., x_N)$ – šūnu skaits, kas tiek apmeklētas uz darba lentes.
- $S(N) = \max S(x_1, ..., x_N)$.

$$\text{SPACE}(f(N)) =$$

$$= \{L \mid L \text{ var atrisināt ar Tjūringa}$$

$$\text{mašīnu, kurai } S(N) \leq Cf(N)\}.$$

6.2.2.2. NSPACE Definīcija

$$\text{NSPACE}(f(N)) =$$

$$= \{L \mid L \text{ ir determinēta } M, \text{visiem } x, L(x) = M(x),$$

$$\text{un } M \text{ izmanto } \leq cf(N) \text{ šūnas uz darba lentes}\}.$$

> **! Savča teorēma**
>
> $$\text{NSPACE}(f(N)) \subseteq \text{SPACE}\ (f^2(N))$$

6.2.2.3. LOGSPACE Definīcija

$$\text{LOGSPACE} = \text{SPACE}\ (\log N).$$

$$\text{LOGSPACE} \subseteq U_c\ \text{TIME}\left(c^{\log N}\right) =$$

$$U_c\ \text{TIME}\ (N^c) = P$$

Labs mentālais modelis, lai pierādītu, ka algoritms pieder LOGSPACE – ja var iztikt ar $O(1)$ mainīgo daudzumu, kur katrs mainīgais ir no 0 līdz N vai noteikts fiksētu vērtību skaits.

6.2.3. Laika-Telpas sakarības

> **! Teorēma**
>
> Ja $f(n) \geq \log N$, tad
>
> $$\text{TIME}(f(N)) \subseteq \text{SPACE}(f(N)) \subseteq$$
>
> $$\subseteq \bigcup_c \text{TIME}\ (c^{f(N)})$$

Laiks $O(f(N)) \rightarrow$ atmiņa $O(f(N))$.

6.2.4. Asimptotiskas augšanas hierarhija

Sekojošas funkcijas pieaugums pie $x \rightarrow \infty$:

$\log(x) \ll x \ll x \cdot \log(x) \ll x^k \ll a^x \ll x! \ll x^x$

- x: mainīgais (parasti n).
- k: jebkurš vesels pozitīvs skaitlis ($k \in \mathbb{N}$).
- a: reāla konstante lielāka par 1 ($a > 1$).

Šo hierarhiju var izmantot intuīcijai par to vai funkcija pieder klasei sarežģītības klasei, bet kā pamatojums tas nederētu.

x^ε ir izņemts laukā, lai nejauktu galvu

Source; Mathematics for Computer Science, 2018, Eric Lehman, Google Inc.

7. Klase P

7.1. Definīcija

Klase P ir problēmu kopa, ko var atrisināt ar deterministisku Tjūringa mašīnu polinomiālā laikā.

- $P = \bigcup_k \text{TIME}(n^k)$

Citiem vārdiem: problēma pieder P, ja eksistē deterministiska Tjūringa mašīna, kas to atrisina $O(n^k)$ soļos, kādai konstantei k.

Klase P tiek uzskatīta par praktiski atrisināmo problēmu klasi. Visi saprātīgie deterministiskie skaitļošanas modeļi ir polinomiāli ekvivalenti (vienu var simulēt ar otru polinomiālā laikā).

7.2. Piemērs (PATH)

- Dots grafs G un divas virsotnes u, v.
- Jautājums: vai eksistē ceļš no u uz v?
- Rupjais-spēks: pārbaudīt visus ceļus – eksponenciāls laiks.
- Efektīvs algoritms: meklēšana plašumā (breadth-first search); laika sarežģītība: $O(|V| + |E|)$.

7.3. Piemērs (RELPRIME)

- Doti skaitļi x, y (binārā kodējumā).
- Jautājums: vai skaitļi ir savstarpēji pirmskaitļi?
- Efektīvs algoritms: Eiklīda algoritms (izmantojot mod); laika sarežģītība: $O(\log n)$ (jo katrā iterācijā skaitļi būtiski samazinās).

8. Klase NP

8.1. NP problēmas

NP (nederminēti-polinomiālas) problēmas ir problēmas (2 ekvivalentas definīcijas):

1) $L \in \text{NP}$, ja eksistē pārbaudes algoritms – $O(n^c)$ laika Tjūringa mašīna M:
   a) Ja $L(x) = 1$, tad eksistē y: $M(x, y) = 1$.
   b) Ja $L(x) = 0$, tad visiem y: $M(x, y) = 0$.

c) *Informācija y var saturēt brīvi definētu informāciju.*

d) *Pārbaudes algoritmā tas ir risinājums, ko tas pārbauda.*

2) NP = problēmas L, ko var atrisināt ar nedeterminētu mašīnu $O(n^c)$ laikā.

Ekvivalence ir pierādīta ar abpusēju pārveidojumu no pārbaudītāja uz nedet. TM un atpakaļ.

8.2. NP-pilnas probēmas un to redukcijas

8.2.1. Polinomiāla redukcija ($\leq_{\text{poly}}$)

- $A \leq B$ – A var atrisināt, noreducējot to uz B.
- $A \underset{\text{poly}}{\leq} B$, ja ir $O(n^c)$ laika algoritms (Tjūringa mašīna) P:
  - M pārveido A ieejas datus par B ieejas datiem;
  - $A(x) = B(M(x))$.

8.2.2. NP-pilnīgums

- A – NP-pilna, ja:
  - $A \in$ NP;
  - Ja $B \in$ NP, tad $B \leq_{\text{poly}} A$.

8.2.3. 3-SAT problēma

Vai formulai 3-CNF formā var piemeklēt katra mainīgā vērtības tā, lai formula būtu 1 (patiess).

8.2.4. CIRCUIT-SAT problēma

Dota funkcija $F(x_1, ..., x_n)$, kas sastāv no vārtiem (AND, OR, NOT).

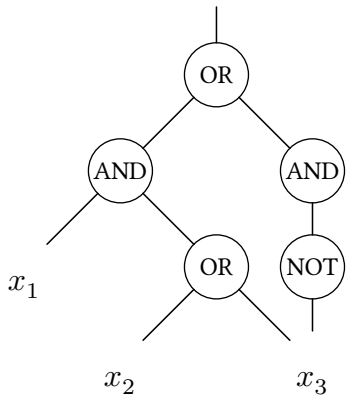

Figure 8.1: CIRCUIT-SAT visual

Vai var atrast mainīgo vērtības tā lai gala izvade būtu 1 (patiess).

8.2.5. CLIQUE problēma

$\exists C \subseteq V : (|C| = k) \wedge (\forall (u, v \in C) : (u, v) \in E)$

Vārdiski. Vai eksistē virsotņu kopa S lielumā k, kurā katra virsotne ir savienota ar katru otro no kopas S.

8.2.6. IND-SET problēma

$\exists S \subseteq V : (|S| = k) \wedge (\forall (u, v \in S) : (u, v) \notin E)$

Vārdiski. Vai grafā $G = (V, E)$ eksistē virsotņu kopa S lielumā k, kurā katra no virsotnēm nav savienota ar nevienu citu virsotni no šīs kopas.

8.2.7. LIN-INEQ problēma

Vai dotā lineāru nevienādību sistēma ar bināriem mainīgajiem ir atrisināma.

8.2.8. CIRCUIT-SAT $\leq_{\text{p}}$ 3-SAT

- Katram starprezultātam (kas nav pirmajā ievadē, i.e., x_1, x_2, ..., x_n) ievieš jaunus mainīgos y_i.
- Katriem vārtiem formulē atbilstošas izteiksmes.

Piemērs AND vārtiem. Nosaucam ievades kā x, y un izvadi kā z: $z = x \wedge y$

x	y	z	$z = x \wedge z$?
0	0	0	jā
0	0	1	nē
0	1	0	jā
0	1	1	nē
1	0	0	jā
1	0	1	nē
1	1	0	nē
1	1	1	jā

Izveidojam pretrunas katrai rindai. Tas ir, konjunkciju katrai rindai ar „nē“.

Piemēram, 2. rindai $(0, 0, 1)$: $x \vee y \vee \neg z$.

Tad uzbūvējam konjunkciju vārtiem:

$$(x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z) \wedge$$

$$\wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$

Veido konjunkciju no visiem vārtiem shēmā. Tā kā 3-SAT sagaida 3 mainīgos katrā iekavā. Tiem,

kas satur 1 vai 2 (identitātes vārti un not vārti attiecīgi), pārveido tos par 3-CNF konjunkciju pievienojot jaunu mainīgo, kas vienā formulā ir pozitīvs un otrā – negācija.

$$(x \vee \neg b) = (x \vee \neg b \vee a) \wedge (x \vee \neg b \vee \neg a)$$

Analogiski iekavām ar vienu elementu. Rezultātā ir 3-CNF formula, ko var izmantot ar 3-SAT algoritmu.

8.2.9. 3-SAT $\leq_{\text{p}}$ IND-SET

Katrai iekavai no formulas veido 3 virsotnes (grafa komponenti), kas apzīmē mainīgo (ar NOT, ja ir negācija). Katra virsotne (literālis) no komponentes ir savā starpā savienota ar pārējām. Starp pretrunīgiem literāliem starp komponentēm pievieno šķautni.

Piemērs formulai $(x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z)$:

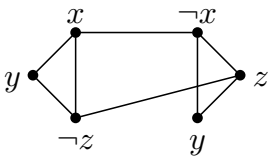

Figure 8.2: 3-SAT → INDSET visual

Tagad varam pielietot IND-SET(G, m) ar izveidoto grafu un m kā iekavu skaitu oriģinālajā formulā.

8.2.10. IND-SET $\leq_{\text{p}}$ CLIQUE

- Veido grafa papildinājumu (komplementu) $G' = (V, E')$:

$$E' := \{(u, v) \in V \times V \,|\, (u, v) \notin E\}$$

Vārdiski. Jauns grafs G, kurā ir visas virsotnes no V, bet visas šķautnes, kas ir G nav G' un pretēji – visas šķautnes kā nav G ir G'.

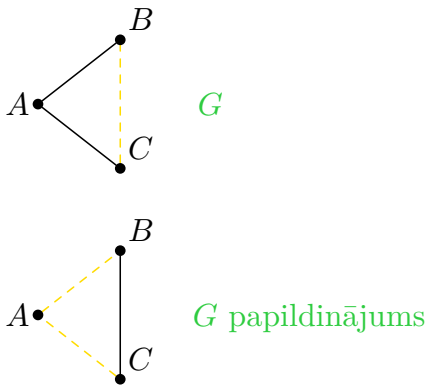

Figure 8.3: Papildgrafa piemērs

Ir spēkā sakarība INDSET(G, k) = CLIQUE(G', k).

9. Extras

9.1. Logaritmu īpašības

Property	Definition	Example
Product	$\log_b mn = \log_b m + \log_b n$	$\log_3 9 = \log_3 9 + \log_3 x$
Quotient	$\log_b \frac{m}{n} = \log_b m - \log_b n$	$\log_{\frac{1}{4}} \frac{4}{5} = \log_{\frac{1}{4}} 4 - \log_{\frac{1}{4}} 5$
Power	$\log_b m^p = p \cdot \log_b m$	$\log_2 8^x = x \cdot \log_2 8$
Equality	If $\log_b m = \log_b n$, then $m = n$	$\log_8(3x-4) = \log_8(5x+2)$ so, $3x - 4 = 5x + 2$
Pow. to log	$a^{\log_a(x)} = x$	$2^{\log_2(x)} = x$

9.2. Atvasinājumu tabula

Funkcija	Atvasinājums†	Piezīmes
x^n	nx^{n-1}	
e^x	e^x	
a^x	$a^x \ln(a)$	$a > 0$
$\ln(x)$	$\frac{1}{x}$	
$\log_a(x)$	$\frac{1}{x \ln(a)}$	
$\frac{1}{x}$	$-\frac{1}{x^2}$	
$\frac{1}{x^n}$	$-\frac{n}{x^{n+1}}$	
$\sqrt{x}$	$\frac{1}{2\sqrt{x}}$	
$\frac{1}{\sqrt{x}}$	$-\frac{1}{2x^{\frac{3}{2}}}$	

†Ja $x = g(x)$ (kompleksa funkcija), tad pie atvasinājuma piereizina $g(x)'$.

9.3. Atvasinājumu īpašības

Rule Name	Function	Derivative
Summa	$f(x) + g(x)$	$f'(x) + g'(x)$
Starpība	$f(x) - g(x)$	$f'(x) - g'(x)$
Reizinājums	$f(x) \cdot g(x)$	$f'(x) \cdot g(x) + f(x) \cdot g'(x)$

9.4. Kāpinājumu īpašības

Rule Name	Formula
Reizinājums	$a^m \cdot a^n = a^{m+n}$
Dalījums	$\frac{a^m}{a^n} = a^{m-n}$
Pakāpes pakāpe	$(a^m)^n = a^{m \cdot n}$
Reizinājuma pakāpe	$(a \cdot b)^m = a^m \cdot b^m$
Dalījuma pakāpe	$\left(\frac{a}{b}\right)^m = \frac{a^m}{b^m}$
0-pakāpe	$a^0 = 1$
Negatīva pakāpe	$a^{-m} = \frac{1}{a^m}$
Saikne ar sakni	$a^{\frac{m}{n}} = \sqrt[n]{a^m}$

9.5. Noderīgas izteiksmes laika analīzē

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

$$r > 1 : \sum_{i=0}^{n} a \cdot r^i = a \cdot \frac{r^{n+1} - 1}{r - 1}$$

$$r < 1 : \sum_{i=0}^{\infty} a \cdot r^i = \frac{a}{1 - r}$$

$$\sum_{i=1}^{n} \log i = \log(n!) \approx n \log n - n + O(\log n)$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$