

581336-0 Laskennan teoria

luennot syyslukukaudella 2003

Jyrki Kivinen

- tietojenkäsittelytieteen laudatur-kurssi, 3 ov
- pakollinen tietojenkäsittelytieteen suuntautumisvaihtoehdossa
- esitiedot käytännössä Tietorakenteet, Ohjelmoinnin ja laskennan perusmallit, joitain matematiikan kursseja
- kurssin voi hyvin aloittaa vaikka Ohj. lask. perusmallit olisi kesken

Opetusmuodot

- luennot 14.10.–3.12. ti 12-14, ke 10-12
- luennoijan vast.otot ti 11.30–12.00, ke 12.00–12.30
- harjoitukset ks. opetusohjelma
- kurssikoe pe 12.12. kello 9-13 Auditorio (huom. RIO)
- harjoitukset **pakollisia**: ratkaistava väh. 25% tehtävistä

Kurssin suorittaminen

- maksimi 60 pistettä: koe 50 p., harjoitukset 10 p.
- hyväksymisraja n. 30 p., arvosanan 3/3 raja n. 54 p.
- harjoituspisteet kun ratkaistu p % laskuharjoitustehtävistä:
 - $p < 25$: hylätty
 - $25 \leq p \leq 50$: 0 pistettä
 - $50 \leq p \leq 90$: $10 \cdot (p - 50)/40$ pistettä
 - $90 \leq p \leq 100$: 10 pistettä
- jos laskarien kertymisessä on ongelmia, selvitä **ajoissa** luennoijan kanssa

Oppimateriaali

Luentomateriaali ilmestyy kurssin kotisivulle ja luentokansioon (A412)
mutta ei ole täydellinen esitys kurssin asioista

Kurssikirja Hopcroft, Motwani, Ullman: *Introduction to Automata Theory, Languages, and Computation* (luvut 8–10; kurssikirjahyllyssä)

Oheislukemisto Orponen: Laskennan teoria (luvut 4–7 kattavat kurssin asiat; myydään laitoksen monistemyynnissä)

Muitakin kirjoja on paljon, esim. Sipser: *Introduction to the Theory of Computation* (kurssikirjahyllyssä)

Motto

Computational problems are not only things that have to be solved, they are also objects that can be worth studying.

Christos Papadimitriou

Tavoitteet

- tutustua **universaaleihin** laskennan malleihin
- * hallita Turingin koneiden peruskonstruktiot
- ymmärtää että laskennalliset ongelmat voivat olla **ratkeamattomia** tai **työläitä**
- ymmärtää **NP-täydellisyyden** merkitys (myös matemaattinen merkitys)
- * tunnistaa tyypilliset ratkeamattomat ja NP-täydelliset ongelmat
- * osata yksinkertaiset ratkeamattomuus- ja NP-täydellisyydestodistukset

Miksi?

- (tehokkaan) laskennan perusolemuksen selvittämistä
- ratkeamattomia ongelmia esiintyy logiikassa ja siihen liittyen tekoälyssä, formaalissa verifiointissa jne.
- työläitä ongelmia esiintyy kaikenlaisissa sovelluksissa (pakkaus, verkot, ...)
- johdatusta teoreettisen tietojenkäsittelytieteen käsitteistöön ja ajatteluun
- nämä asiat ovat niin keskeisiä että ne pitää tuntea pintaa syvemmältä

(Lyhyt vastaus: hauskaa ja hyödyllistä)

Sisältö

0. **Johdanto:** laskennalliset ongelmat, pysähtymisongelman ratkeamattomuus
1. **Universaaleja laskennan malleja:** Turingin koneet, rajoittamattomat kieliopit, Churchin-Turingin teesi
2. **Laskettavuusteoriaa:** rekursiiviset ja rekursiivisesti lueteltavat kielet, rekursiiviset funktiot ja palautukset, universaalit Turingin koneet, ratkeamattomuustuloksia
3. **Vaativuusteoriaa:** aika- ja tilavaativuus, epädeterministiset vaativuusluokat, polynomiset palautukset, NP-täydellisyys

0. Johdanto

Merkintöjä ja konventioita:

- Γ, Σ : äärellisiä aakkostoja; esim. $\Gamma = \{0, 1\}$, $\Sigma = \{a, b, c, d\}$.
- $|\Sigma|$: aakkoston koko; esim. $|\Sigma| = 4$.
- pienet kirjaimet a, b, c, \dots : akkosmerkkejä
- pienet kirjaimet x, y, z, u, v, w, \dots : merkkijonoja; esim. $x = ab$, $y = bac$.
- $|x|$: merkkijonon pituus; esim. $|x| = 2$.
- xy : merkkijonojen katenaatio; esim. $xy = abbac$.

- Σ^* : aakkoston Σ (äärellisten) merkkijonojen joukko
- ε : tyhjä merkkijono (merkitään usein myös λ); siis $|\varepsilon| = 0$
- esim. jos $\Sigma = \{0, 1\}$ niin $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$
- millä tahansa äärellisellä Σ joukko Σ^* on **numeroituvasti ääretön**; ts. on olemassa bijektio $f: \mathbb{N} \rightarrow \Sigma^*$
- esim. $f(0) = \varepsilon, f(1) = 0, f(2) = 1, f(3) = 00$ jne.;
leksikografinen järjestys
- **kieli** on mikä tahansa joukko merkkijonoja; esim.
 $\text{Primes} = \{x \in \{0, 1\}^* \mid x \text{ on alkuluvun binääriesitys}\}$
- siis $\text{Primes} = \{10, 11, 101, 111, 1011, \dots\} \stackrel{\Delta}{=} \{2, 3, 5, 7, 11, \dots\}$

- **laskennallinen ongelma** on mikä tahansa kuvaus $\pi: \Sigma^* \rightarrow \Gamma^*$ millä tahansa Σ, Γ
- **päätösongelma** on laskennallinen ongelma jonka arvojoukko on $\{0, 1\}$ ($\triangleq \{ \text{ei, kyllä} \}$); päätösongelma π samastetaan usein kielen $\{x \mid \pi(x) = 1\}$ kanssa
- ohjelmat ovat merkkijonoja, joten missä tahansa ohjelmointikielessä on vain numeroituva määrä mahdollisia ohjelmia
- kieliä (ja päätösongelmia) on **ylinumeroituvasti**

Johtopäätös: on olemassa **ratkeamattomia** ongelmia, joita ei voi ratkaista (esim.) Java-kielellä

Mutta

- kenties kaikki ratkeamattomat ongelmat ovat keinotekoisia ja mielenkiinnottomia, tai
- kenties jokainen ongelma voidaan ratkaista **jollain** kielellä?

Osoittautuu kuitenkin, että

- monet luonnostaan esiintyvät ongelmat ovat ratkeamattomia, ja
- ratkeamattomuuden käsite on suunnilleen sama kaikilla riittävän voimakkailla laskentaformalismeilla (\approx ohjelmointikielillä)

Pysähtymisongelman ratkeamattomuus

(Epämuodollinen johdattelleva esimerkki; yksityiskohtiin palataan.)

Väite: ei ole olemassa C-funktiota `halts(p, x)` joka

- saa syötteenä mielivaltaisen C-funktion tekstin `p` ja tälle sopivan syötteen `x`,
- palauttaa 1 jos laskenta `p(x)` pysähtyy ja
- palauttaa 0 muuten.

Huom. 1: `halts` ei siis saa millään parametreilla joutua ikuiseen silmukkaan.

Huom. 2: syntaksivirheet `p:n` tekstissä jne. kohtaan "muuten".

”Todistus” (hieman C:n syntaksia muokaten): Tehdään vastaoletus että tällainen `halts` on olemassa.

Olkoon c seuraavan ohjelman `confuse` tekstiesitys:

```
void confuse(char *p);
  int halts(char *p, char *x){
    ... /* funktion "halts" runko */
  }
  if (halts(p, p)==1) while (1);
}
```

Nyt sovelletaan funktion `halts` spesifikaatiota:

`confuse(c)` pysähtyy \Leftrightarrow `halts(c, c)==1` \Leftrightarrow `confuse(c)` jää silmukkaan;
ristiriita. ”□”

Johtopäätös: hyvinkin perustavanlaatuiset ohjelmointiin liittyvät kysymykset ovat ratkeamattomia.

Seuraavaksi tarkastellaan tämäntyyppisiä ilmiöitä ohjelmointikielten sijaan formaaleilla laskennan malleilla, erityisesti [Turingin koneilla](#).

Formaalien mallien etuja:

- semantiikka helppo määritellä formaalisti
- vältetään ohjelmointikielten hankalat erikoispiirteet
- vältetään tulosten riippuvuus ohjelmointikielestä
- saadaan yleinen matemaattinen teoria joka on täysin riippumaton käytettävissä olevista laskentalaitteista

1. Universaaleja laskennan malleja

Laskenta \approx datan käsittely annettuja sääntöjä täsmällisesti seuraamalla

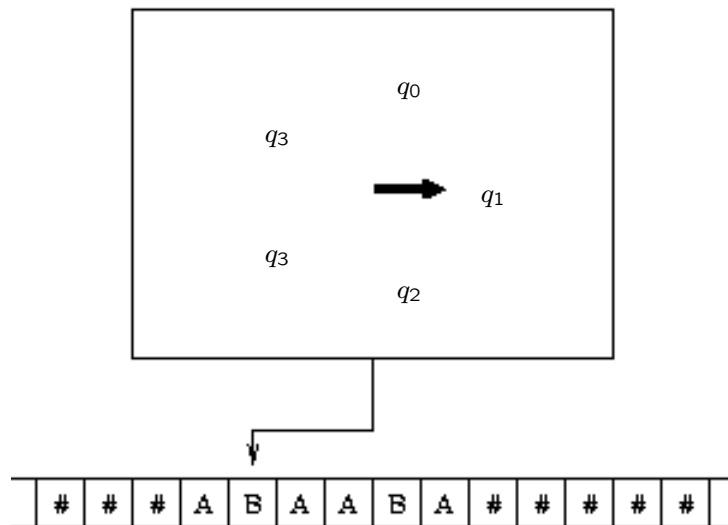
- kahden kokonaisluvun kertolasku tietokoneella, tai kynällä ja paperilla: selvästi laskentaa
- entä matemaattisten teoreemojen todistaminen aksiomista lähtien?

Pitää myös voida todeta että jotain **ei ole mahdollista** laskea, joten tarvitaan täsmällisempi määritelmä.

- määritelmän pitäisi olla riittävän yleinen että esim. tuleva tekninen edistys ei mitätöi tuloksia
- mallin pitää kuitenkin olla periaatteessa fyysisesti toteutettavissa

\Rightarrow universaalit laskennan mallit

Turingin kone (Alan Turing, 1936)



Ohjausyksikkö: kone tilassa q_1
Nauhapää osoittaa merkkiä B
Työnauha sis. merkkijonon ABAAB

Koneen siirtymäfunktio määrää

- mikä merkki kirjoitetaan nauhapään kohdalle,
- mihin suuntaan nauhapää liikkuu ja
- mikä on seuraava tila kun on annettu
- nykyinen tila ja
- nauhapään alla oleva merkki.

Motivaatio: yritetään tehdä abstrakti malli siitä, millaista laskentaa matemaatikko (tms.) voi tehdä "mekaanisesti":

- käytettävissä kynä, kumi ja rajattomasti paperia
- kerralla nähdään vain vakiokokoinen osa muistiinpanoista
- matemaatikon muisti on äärellinen

Vaikuttaa vähän erilaiselta kuin tietokoneet, mutta

- vuonna 1936 ei ollut tietokoneita
- malli osoittautuu yhtä voimakkaaksi kuin suuremmin moderneja tietokoneita esittävät mallit (lisää tuonnempana)

Muodollisemmin **Turingin kone** (Turing machine, TM) on seitsikko

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

missä

- Q on **tilajoukko** jonka on oltava **äärellinen**
- Γ on **nauha-aakkosto** ja $\Sigma \subset \Gamma$ **syöteaakkosto** (kumpikin äärellinen)
- δ on **siirtymäfunktio**
- $q_0 \in Q$ on **alkutila**
- $B \in \Gamma - \Sigma$ on **tyhjämerkki** (blank)
- $F \subseteq Q$ on **hyväksyvien** tilojen joukko

δ on osittainen funktio joukolta $Q \times \Gamma$ joukkoon $Q \times \Gamma \times \{L, R\}$.

Siirtymäfunktion arvo $\delta(q, X) = (q', Y, D)$ tarkoittaa että jos

- M on tilassa q ja
- nauhapään alla on merkki X

niin seuraavalla laskenta-askelella M

- siirtyy tilaan q' ,
- kirjoittaa nauhalle merkin Y (merkin X tilalle) ja
- siirtää nauhapäätä yhden askelen suuntaan D (L : vasen, R : oikea).

Jos $\delta(q, X)$ on määrittelemätön niin M pysähtyy.

Intuitiivisesti

- jos M pysähtyy hyväksyvään tilaan se **hyväksyy** syötemerkkijonon
- jos M pysähtyy muunlaiseen tilaan se **hylkää** syötemerkkijonon.

Turingin koneen M **hyväksymä** (tai **tunnistama**) kieli $L(M) \subseteq \Sigma^*$ on niiden merkkijonojen joukko jotka M hyväksyy. Jatkossa oletetaan että hyväksyvistä tiloista ei ole siirtymiä.

Huom. kieleen L_M eivät kuulu ne syötteen joilla M jää ikuisen silmukkaan.

Kieltä $L \subseteq \Sigma^*$ sanotaan **rekursiivisesti lueteltavaksi** jos $L = L(M)$ jollain Turingin koneella M , ja **rekursiiviseksi** jos lisäksi M pysähtyy kaikilla syötteillä. (Tästä lisää myöhemmin.)

Turingin koneen **tilannetta** (configuration) merkitään merkkijonolla vqw missä

- $q \in Q$ on koneen tila,
- $v \in \Gamma^*$ on nauhan sisältö vasemmanpuolimmaisesta ei-tyhjästä merkistä nauhapään vasemmalla puolella olevaan merkkiin ja
- $w \in \Gamma^*$ on nauhan sisältö nauhapään kohdalla olevasta merkistä oikeanpuolimmaiseen tyhjään merkkiin.

Siis alussa ollut esimerkkitalanne merkitään Aq_1BAAB .

Jos nauhapään vasemmalla puolella on vain tyhjää, niin $v = \varepsilon$ ja merkkijonon w alussa voi olla tyhjää; vastaavasti oikealla.

Koneen **alkutilanne** syötteellä $w \in \Sigma^*$ on q_0w . Siis aluksi kone on alkutilassa, syöte on nauhalla tyhjien ympäröimänä ja nauhapää syötteen alussa.

Jos siirtymäfunktion mukaan tilannetta vqw seuraa tilanne $v'q'w'$, merkitään

$$vqw \vdash_M v'q'w'.$$

Siis

- jos $\delta(q, a) = (q', b, R)$ niin $vqaw \vdash vbq'w$ kaikilla $v, w \in \Gamma^*$
- jos $\delta(q, a) = (q', b, L)$ niin $vcqaw \vdash vq'cbw$ kaikilla $c \in \Gamma, v, w \in \Gamma^*$

Jos on olemassa tilannejono $v_1q_1w_1 = vqw, v_2q_2w_2, v_3q_3w_3, \dots, v_nq_nw_n = v'q'w'$ missä $v_iq_iw_i \vdash v_{i+1}q_{i+1}w_{i+1}$, merkitään

$$vqw \vdash_M^* v'q'w'.$$

Siis

$$L(M) = \{ x \in \Sigma^* \mid q_0x \vdash_M^* vqw \text{ joillain } q \in F, v, w \in \Gamma^* \}.$$

Esimerkki

Konstruoidaan Turingin kone joka hyväksyy kielen $A = \{0^n 1^n \mid n \geq 1\}$.

Perusidea: Apumerkkeinä X ja Y . Toistetaan seuraavaa:

- vaihdetaan 0:n tilalle X
- siirrytään nauhalla oikealle kunnes tulee 1
- vaihdetaan 1:n tilalle Y
- palataan vasemmalle kunnes löytyy X
- aloitetaan seuraava iteraatio tämän X :n oikealta puolelta

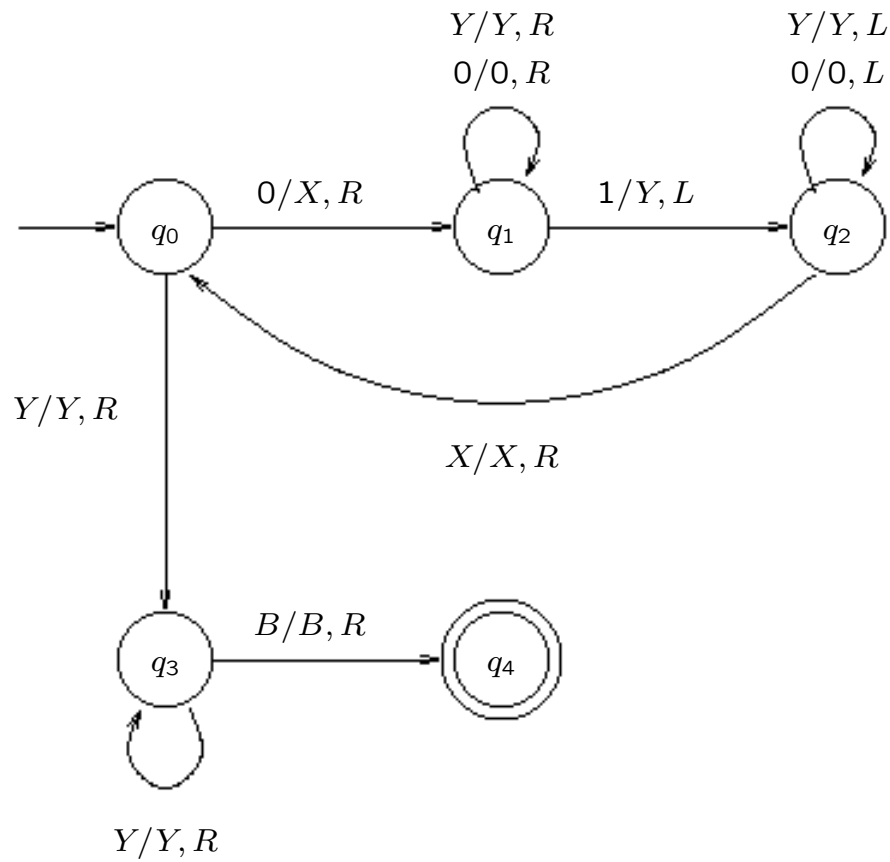
Muodollisemmin $A = L(M)$ missä

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

ja δ on oheisen taulukon mukainen.

tila	merkki				
	0	1	X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	—	—	—	—	—

Havainnollisemmin asian voi esittää [siirtymäkaaviona](#).



Kielen $\{0^n 1^n \mid n \geq 1\}$ tunnistaminen; siirtymäkaavio

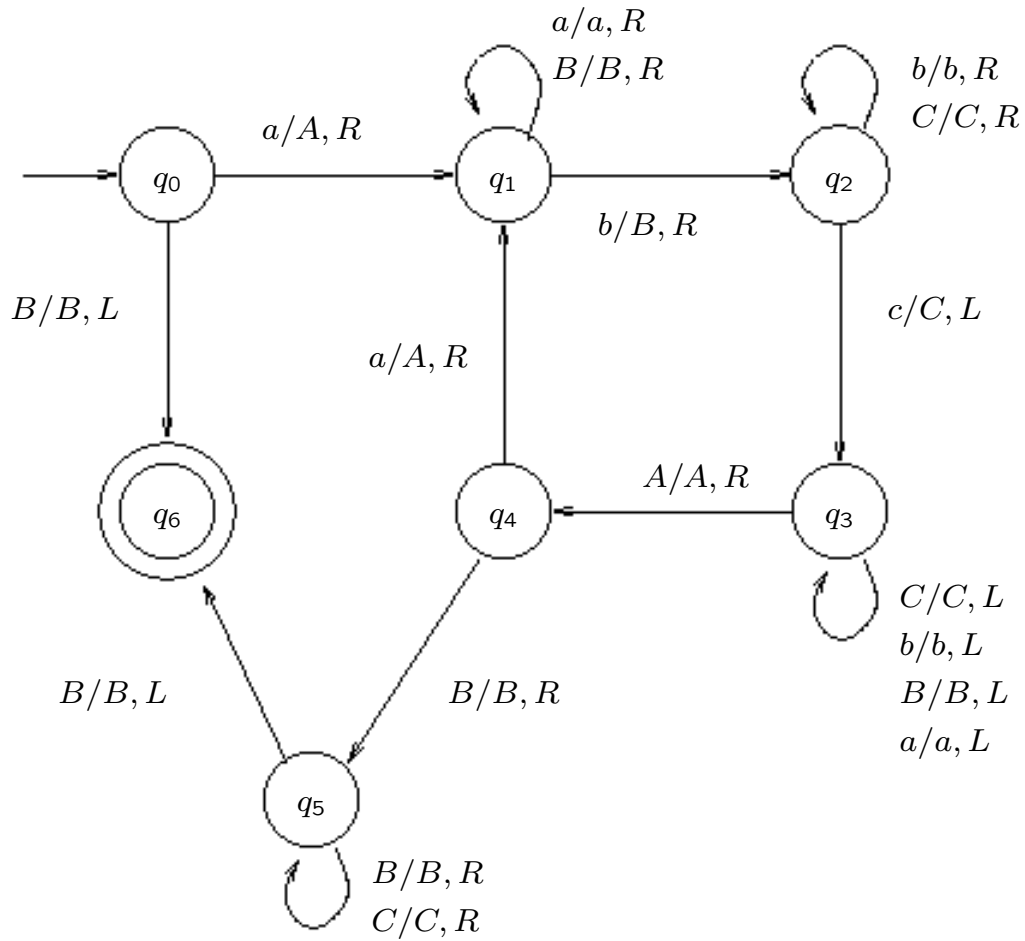
Esimerkki 2

Tunnistetaan kieli $\{ a^k b^k c^k \mid k \geq 0 \}$

Perusajatus:

- korvataan yksitellen jotkin a , b ja c merkeillä A , B ja C
- samalla tulee tarkastetuksi että a :t on ennen b :itä jne.
- kun a :t loppuvat, tarkastetaan ettei b - tai c -merkkejä jäänyt yli

Huom. kieli ei ole kontekstiton.



Kielen $\{a^n b^n c^n \mid n \geq 0\}$ tunnistaminen

Huom. Turingin koneen laskentavoima (se mitkä kielet ovat tunnistettavissa) on sama vaikka mallia muunneltaisiin paljonkin

- erillinen hylkäävä lopputila
- nauha vain toiseen suuntaan ääretön
- nauhalla useita **uria**
- useita nauhoja
- . . .

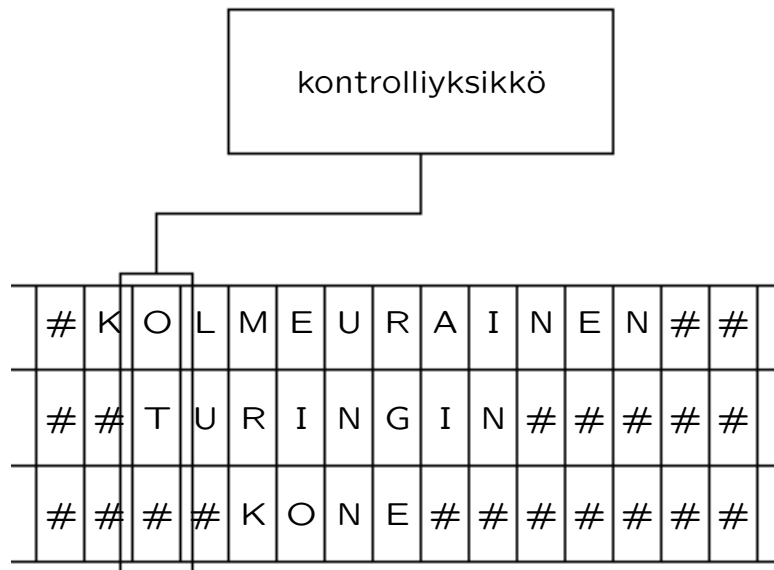
Vielä oleellisempaa on, että Turingin kone on laskentavoimaltaan sama kuin aivan muista lähtökohdista johdetut formalismit (Kleenen rekursiiviset funktiot, yleiset kieliovit, RAM-koneet, . . .).

- monimutkaisia turinginkonekonstruktioita ei tietenkään voi käytännössä esittää siirtymäkaavion tarkkuudella
- myös Turingin koneista puhuttaessa voidaan käyttää aliohjelmia ja muita vastaavia ajattelumalleja
- perusteiden ymmärtämiseksi kurssilla käytetään jonkin verran aikaa yksinkertaisten Turingin koneiden tarkkaan käsittelyyn
- samalla Turingin koneen varianttien asema selvenee.

Turingin koneen laajennuksia

Turingin koneen määritelmään voidaan tehdä erilaisia muutoksia siten että edelleen voidaan tunnistaa tasan sama luokka kieliä.

Moniuraiset Turingin koneet: nauha jakautuu k uraan (track) joilla kuitenkin on yhteinen nauhapää.



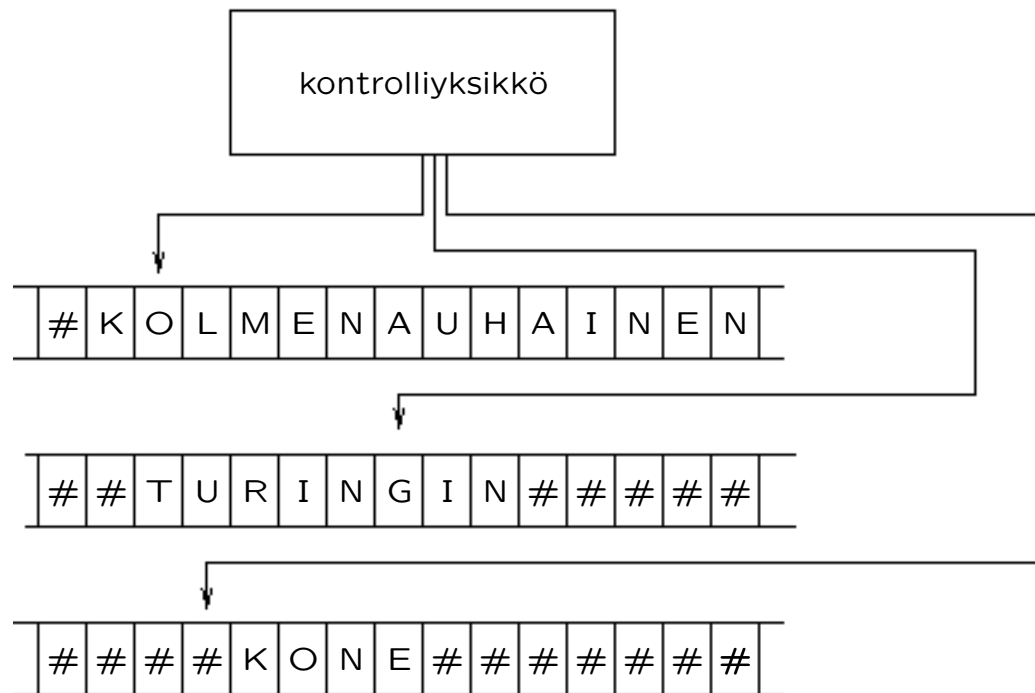
- jokaisella askelella luetaan ja kirjoitetaan kullekin uralle samalle kohdalle mutta muuten toisista urista riippumatta
- formaalisti nyt siis siirtymäfunktio on

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}$$

- alkutilanteessa syöte ensimmäisellä uralalla, muilla urilla tyhjämerkkiä
- helppo simuloida yksiuraisella koneella: vaihdetaan aakkoston Γ tilalle Γ^k , tyhjämerkiksi $(\#, \dots, \#)$ jne.

⇒ voidaan käyttää moniuraisia koneita silloin kun se tuntuu helpommalta

Moninauhaiset Turingin koneet: nyt meillä on k nauhaa joilla omat nauhapäät (voivat liikkua eri suuntiin).



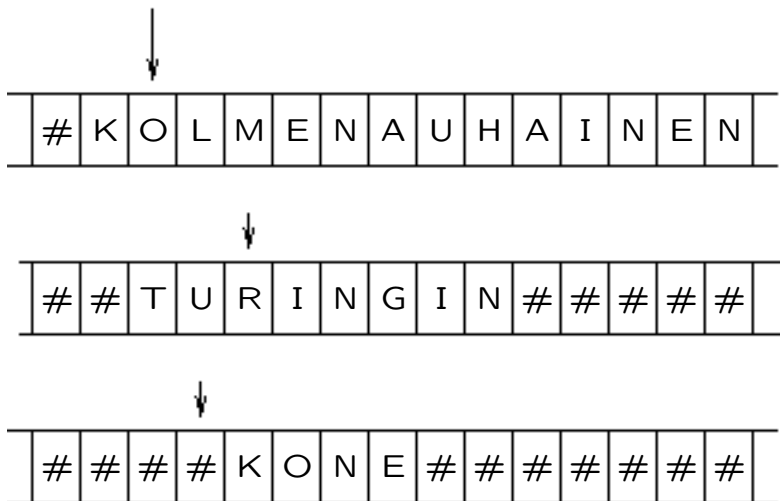
- formaalisti nyt siis siirtymäfunktio on

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$$

- alkutilanteessa syöte ensimmäisellä nauhalla, muilla nauhoilla tyhjämerkkiä

Osoitetaan että k -nauhaista konetta voi simuloida $2k$ -uraisella yksinauhaisella koneella

Idea: merkataan ylimääräisille urilla nauhapäiden sijainnit



#	K	O	L	M	E	N	A	U	H
-	-	X	-	-	-	-	-	-	-
#	#	T	U	R	I	N	G	I	N
-	-	-	-	X	-	-	-	-	-
#	#	#	K	O	N	E	#	#	#
-	-	X	-	-	-	-	-	-	-

Moninauhaisen koneen yhden askelen simuloimiseksi

- luetaan koko nauha kerran läpi ja muistetaan (äärellistilaisessä kontrollissa) mitkä merkit ovat nauhapäiden kohdalla
- valitaan siirtymä ja kirjoitettavat merkit
- luetaan koko nauha uudestaan läpi ja tehdään asiaankuuluvat muutokset

Oletetaan että syötteellä x simuloitava kone tekee $t(x)$ siirtymää

⇒ käsiteltävän nauhanosan pituus myös kork. $t(x)$ merkkiä

⇒ simuloiva kone suorittaa $O(t(x))$ askelta per simuloitava askel

⇒ simuloiva kone tekee kaikkiaan $O(t(x)^2)$ askelta

Johtopäätös: kieli voidaan tunnistaa **standardimallisella** Turinginkoneella jos ja vain jos se voidaan tunnistaa moniuraisella Turingin koneella jos ja vain jos se voidaan tunnistaa moninauhaisella Turingin koneella.

Muita samantyyppisiä variaatioita:

- syöte erillisellä read only -nauhalla
- nauhoilla alkukohta jonka vasemmalle puolelle ei saa mennä

Näissä tapauksissa on myös selvää että tunnistamiseen käytettävien laskenta-askelien määrä ei muutu "liikaa".

Epädeterministiset koneet, joita seuraavaksi käsitellään, ovat ilmaisuvoimaltaan samoja kuin deterministiset mutta laskenta-aikojen suhteen tilanne on ongelmallisempi.

Epädeterministiset Turingin koneet

- analoginen epädeterministisen äärellisen ja pinoautomaatin kanssa
- yhdestä tilanteesta voi olla useita vaihtoehtoisia siirtymiä
- intuitio: ajatellaan että kone osaa ”arvata” vaihtoehdon joka johtaa lopulta hyväksyvään tilaan (jos mahdollista)
- epädeterminismi on näppärä ”ohjelmointitekniikka”

Erityyppinen variantti kuin moninauhaiset jne. koneet

- ei suoraan sovi algoritmi-käsitteen formalisoinniksi
- ratkeavien ongelmien luokka ei kuitenkaan muutu vaikka epädeterminismi sallitaan
- tilanne muuttuu oleellisesti jos puhutaan **nopeasta** ratkaisemisesta; tähän palataan kurssin loppupuoliskolla

Muodollinen määrittely: Epädeterministinen Turingin kone (Nondeterministic Turing machine, NTM) on seitsikko

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

missä

- $Q, \Sigma, \Gamma, q_0, B$ ja F kuten deterministisessä tapauksessa
- siirtymäfunktio on funktio

$$\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

missä

$$\begin{aligned} \mathcal{P}(A) &= \text{joukon } A \text{ potenssijoukko} \\ &= \{B \mid B \subseteq A\} \end{aligned}$$

Epädeterministisen koneen hyväksymä kieli

- määritellään tilanteet vqw kuten deterministisessä tapauksessa
 - samoin seuraajarelaatio \vdash_M ja sen sulkeuma \vdash_M^* , paitsi että ehdot muotoa $(q', Y, D) = \delta(q, X)$ korvataan ehdoilla $(q', Y, D) \in \delta(q, X)$
- \Rightarrow voi päteä $vqw \vdash_M v'q'w'$ nolalla, yhdellä tai useammalla $v'q'w'$ (kuitenkin kork. $2|Q||\Gamma|$)

- koneen M hyväksymä kieli on

$$L(M) = \{ x \in \Sigma^* \mid q_0x \vdash_M^* vqw \text{ joillain } q \in F, v, w \in \Gamma^* \}$$

- \Rightarrow M hyväksyy jos "sopivat" seuraajatilanteen valinnat johtaisivat hyväksyvään tilaan

Olkoon M epädeterministinen Turingin kone. Osoitetaan nyt miten kieli $L(M)$ voidaan tunnistaa deterministisellä Turingin koneella.

Perusidea koneen M simuloimiseksi annetulle syötteellä:

- tutkitaan (mahdollisesti ääretöntä) verkkoa, jonka solmuina ovat alkutilanteesta saavutettavissa olevat koneen M tilanteet
- tilanteiden vqw ja $v'q'w'$ välillä on kaari jos $vqw \vdash_M v'q'w'$
- M hyväksyy jos alkutilanteesta on polku hyväksyvään tilanteeseen
- etsitään tällainen polku leveyssuuntaisesti

Tarkemmin: Oletetaan että M on yksinauhainen. Simuloidaan deterministisellä 3-nauhaisella konella M' .

- nauha 1 on työnauha
- nauha 2 sisältää **jonon** jolla leveyshakua ohjataan
- nauhaa 3 käytetään tilanteiden ”monistamiseen” jonon jatkoksi

Jos koneen M nauha-aakkosto on Γ , otetaan uudeksi nauha-aakkostoksi

$$\Gamma' = \Gamma \cup \{*\} \cup Q.$$

Jono jossa tilanteet $v_1q_1w_1, \dots, v_nq_nw_n$ voidaan koodata nauhalle 2 muotoon

$$\dots \#\#\# * *v_1q_1w_1 * v_2q_2w_2 * \dots * v_nq_nw_n * * \#\#\# \dots$$

Simulaatio (eli polunetsintä) etenee vaiheittain.

- vaiheen 1 aluksi nauhalla 2 on syötettä x vastaava alkutilanne $**q_0x**$
- jos vaiheen k aluksi nauhalla 2 on (tyhjämerkkien lisäksi)

$$**v_1q_1w_1 * v_2q_2w_2 * \dots * v_nq_nw_n **$$

niin vaiheen k lopuksi nauhalla on

$$**v_2q_2w_2 * \dots * v_nq_nw_n * v'_1q'_1w'_1 * v'_2q'_2w'_2 * \dots * v'_p q'_p w'_p **$$

missä $v'_i q'_i w'_i$, $i = 1, \dots, p$, ovat ne tilanteet joilla $v_1q_1w_1 \vdash_M v'_i q'_i w'_i$.

- jos joskus tulee kirjoitettavaksi koneen M hyväksyvän tilan koodi, niin M' hyväksyy syötteen
- jos jono tyhjenee, niin M' hylkää syötteen

Vaiheen k toteutus suunnilleen:

- tarkista kuinka monta seuraajaa tilanteella $v_1q_1w_1$ on (huom. tällä on vakioyläraja $2|\Gamma||Q|$)
- tee tilanteesta $v_1q_1w_1$ tämän mukainen määrä kopioita nauhalle 3
- käy kopiot järjestyksessä läpi ja muuta kukin vastaamaan "oikeannumeroista" seuraajaa
- kopioi nauhalta 3 nauhalle 2

Oletetaan että koneella M on jokin hyväksyvä kork. n askelen pituinen laskenta, ja millään tilanteella ei ole yli m seuraajaa.

- M' vie ensin jonoon tilanteet yhden (koneen M) laskenta-askelen päässä alkutilanteesta, sitten kahden askelen jne.
- k askelen päässä olevan tilanteen löytämiseksi voidaan joutua käymään läpi $1 + m + m^2 + \dots + m^k$ tilannetta
- siis riittää tutkia nm^n tilannetta
- yhden koneen M tilanteen kuvaus on $O(n)$ merkkiä

⇒ selvästi M' hyväksyy jossain äärellisessä ajassa

Johtopäätös: Jos $A = L(M)$ jollain epädeterministisellä M , niin $A = L(M')$ eräällä deterministisellä M' .

Käänteinen suunta tietysti myös pätee.

Siis kieli A voidaan tunnistaa epädeterministisellä Turingin koneella jos ja vain jos se voidaan tunnistaa deterministisellä Turingin koneella.

Mutta edelläesitetystä konstruktiosta laskenta-askelia voi tulla eksponentiaalisesti lisää; tätä ongelmapiiriä käsitellään kurssin loppupuoliskolla.

Rajoittamattomat kieliopit

Ohjelmoinnin ja laskennan perusmalleista muistetaan, että kieli voidaan kuvata (esim.) kieliopilla joka tuottaa sen, tai automaatilla joka tunnistaa sen.

säännölliset lausekkeet ~ äärelliset automaatit
kontekstittomat kieliopit ~ pinoautomaatit

Nyt saadaan yksi vastaava pari lisää:

rajoittamattomat kieliopit ~ Turingin koneet

Rajoittamaton kielioppi on nelikko $G = (V, \Sigma, P, S)$ missä

- V aakkosto
- Σ päätemerkit; $N = V - \Sigma$ välikemerkit
- $P \subseteq (V^* - \{\varepsilon\}) \times V^*$ produktiot
- $S \in N$ lähtösymboli

Produktiota (α, β) merkitään yleensä $\alpha \rightarrow \beta$.

Erona kontekstittomiin kielioppeihin, että produktion vasemmalla puolella voi olla mikä tahansa epätyhjä merkkijono.

Merkkijono $\gamma \in V^*$ **johtaa suoraan** merkkijonon $\gamma' \in V^*$ jos voidaan kirjoittaa $\gamma = \alpha\omega\beta$ ja $\gamma' = \alpha\omega'\beta$ missä $\omega \rightarrow \omega' \in P$. Tällöin merkitään

$$\gamma \xRightarrow{G} \gamma'.$$

Merkkijono $\gamma \in V^*$ **johtaa** merkkijonon $\gamma' \in V^*$ jos on olemassa $\gamma_0 = \gamma, \gamma_1, \gamma_2, \dots, \gamma_n = \gamma'$ joille $\gamma_{i-1} \xRightarrow{G} \gamma_i$. Tällöin merkitään

$$\gamma \xRightarrow{G^*} \gamma'.$$

Kieliopin G **tuottama kieli** on

$$L(G) = \left\{ x \in \Sigma^* \mid S \xRightarrow{G^*} x \right\}.$$

Esimerkki: muodostetaan $G = (V, \Sigma, P, S)$ jolle

$$L(G) = \{ a^k b^k c^k \mid k \geq 0 \}.$$

(Huom. kieli $\{ a^k b^k c^k \mid k \geq 0 \}$ ei ole kontekstiton.)

Siis $\Sigma = \{ a, b, c \}$. Valitaan $N = \{ S, X, T, A, B, C \}$ ja otetaan produktiot

$S \rightarrow XT$	$CA \rightarrow AC$	$aB \rightarrow ab$
$S \rightarrow \varepsilon$	$CB \rightarrow BC$	$bB \rightarrow bb$
$T \rightarrow ABCT$	$XA \rightarrow a$	$bC \rightarrow bc$
$T \rightarrow ABC$	$aA \rightarrow aa$	$cC \rightarrow cc$
$BA \rightarrow AB$		

Siis merkkijonon $a^k b^k c^k$ tuottamiseksi

- tuotetaan $X(ABC)^k$
- järjestetään A -, B - ja C -merkit aakkosjärjestykseen; tuloksena $LA^k B^k C^k$
- korvataan isot kirjaimet pienillä vasemmalta alkaen.

Tämä osoittaa että rajoittamattomilla kielioppeilla voidaan tuottaa muitakin kuin kontekstisia kieliä.

Seuraavaksi käydään periaatetasolla läpi konstruktiot, jotka osoittavat että itse asiassa rajoittamattomilla kielioppeilla voidaan tuottaa tasan ne kielet, jotka voidaan tunnistaa Turingin koneella.

Lause: Jos kieli voidaan tuottaa rajoittamattomalla kieliopilla, niin se voidaan tunnistaa Turingin koneella.

Todistus (periaate): Olkoon G rajoittamaton kielioppi. Aiemmin esitetyn perusteella riittää muodostaa kaksinauhainen epädeterministinen Turingin kone M jolle $L(M) = L(G)$.

Nauha 1 sisältää vain kopion syötejonosta.

Nauhalle 2 tuotetaan (epädeterministisesti) lähtösymbolista tuotettavissa olevia merkkijonoja. Laskennan aluksi nauhalle 2 kirjoitetaan pelkkä lähtösymboli.

Jos jossain vaiheessa nauhojen sisällöt ovat samat, hyväksytään.

Laskenta koostuu vaiheista joissa kussakin

- viedään epädeterministisesti nauhan 2 nauhapää mielivaltaiseen paikkaan
- valitaan epädeterministisesti mielivaltainen kieliopin G produktio
- jos nauhapään kohdalta löytyy produktion vasen puoli, kirjoitetaan sen paikalle produktion oikea puoli
- verrataan nauhojen 1 ja 2 sisältöjä

Koska produktioita on äärellinen määrä, ne voidaan koodata Turingin koneen tiloihin. Tarkemmat yksityiskohdat sivuutetaan. \square

Lause: Jos kieli voidaan tunnistaa Turingin koneella, niin se voidaan tuottaa rajoittamattomalla kieliopilla.

Todistus: Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ annettu.

Idea on muodostaa kielioppi $G = (V, \Sigma, P, S)$ joka tuottaa koneen M tilanteita. Välikkeiksi otetaan siis ainakin koneen M tilojen symbolit.

Produktiot suunnitellaan siten että

$$[vqw] \xRightarrow{G} [v'q'w'] \quad \text{jos ja vain jos} \quad vqw \vdash_M v'q'w'$$

missä "[", "]" ja "'" ovat uusia välikesymboleja. Tämä on mahdollista koska $vqw \vdash_M v'q'w'$ edellyttää että vqw ja $v'q'w'$ eroavat toisistaan vain merkin q lähiympäristössä.

Merkkijonon $x \in L(M)$ tuottaminen tapahtuu kolmessa vaiheessa:

1. tuotetaan lähtösymbolista S merkkijono $x[q_0x]$.
2. muunnetaan $x[q_0x] \Rightarrow_G^* x[vq_fw]$ missä $q_f \in F$
3. siistitään $x[vq_fw] \Rightarrow_G^* x$

Varsinainen työ tapahtuu vaiheessa 2 jossa G "simuloi" konetta M . Jos $x \notin L(M)$, niin G ei pysty tuottamaan muotoa $x[vq_fw]$ olevia merkkijonoja.

Esitetään vielä konstruktion yksityiskohdat. Aakkostona on

$$V = \Gamma \cup Q \cup \{S, T, [,], X, Y\} \cup \{A_a \mid a \in \Sigma\}$$

(huom. $\Sigma \subset \Gamma$).

Produktiot jakautuvat edelläesitettyjen vaiheiden mukaisesti kolmeen osakokonaisuuteen:

Vaihe 1: alkutilanteen tuottaminen

$$\begin{array}{ll} S & \rightarrow T[q_0] \\ T & \rightarrow \varepsilon \\ T & \rightarrow aTA_a \end{array} \qquad \begin{array}{ll} A_a[q_0 & \rightarrow [q_0A_a \\ A_ab & \rightarrow bA_a \\ A_a] & \rightarrow a] \end{array}$$

(kaikilla $a, b \in \Sigma$)

Vaihe 2: siirtymien simulointi (nämä niille $a, b, c \in \Gamma$ jotka ilmenevät siirtymäfunktiosta)

Siirtymä

Vastaava produktio

$\delta(q, a) = (q', b, R)$	$qa \rightarrow bq'$
$\delta(q, a) = (q', b, L)$	$cqa \rightarrow q'cb$
$\delta(q, \#) = (q', b, R)$	$q] \rightarrow bq]$
$\delta(q, \#) = (q', b, L)$	$cq] \rightarrow qcb]$
$\delta(q, a) = (q', b, L)$	$[qa \rightarrow [q\#b$

Vaihe 3: lopputilanteen siistiminen (kaikille $a \in \Gamma, q_f \in F$)

$q_f \rightarrow XY$	$Ya \rightarrow Y$
$aX \rightarrow X$	$Y] \rightarrow \varepsilon$
$[X \rightarrow \varepsilon$	



Chomskyn hierarkia

Noam Chomskyn vuonna 1956 esittämä luokittelu kielioppeille niiden ilmaisuvoiman mukaan

tyyppi	kieli	kielioppi	tunnistaminen
0	rekurs. lueteltava	rajoittamaton	Turingin kone
1	kontekstinen	kontekstinen	lin. rajoitettu TM
2	kontekstiton	kontekstiton	pinoautomaatti
3	säännöllinen	oikealle lin.	äärellinen autom.

- tyyppi 0 on juuri esitelty (ja esitellään kohta lisää)
- tyypit 2 ja 3 kurssilla Ohjelmoinnin ja laskennan perusmallit
- kuvan täydentämiseksi käydään tässä lyhyesti läpi taso 1

Kontekstinen kielioppi: kuten rajoittamaton kielioppi mutta produktioiden muotoa rajoitetaan

- sallitaan produktiot $\alpha \rightarrow \beta$ missä $|\beta| \geq |\alpha|$
- lisäksi sallitaan produktio $S \rightarrow \varepsilon$ olettaen että S ei esiinny minkään produktio oikealla puolella

Nimi "kontekstinen" tulee siitä, että tällaiset kielet voidaan muuntaa muotoon jossa produktiot (pl. mahd. $S \rightarrow \varepsilon$) ovat tyyppiä

$$\alpha A \beta \rightarrow \alpha \omega \beta$$

missä A on välike ja $\omega \neq \varepsilon$. Siis intuitiivisesti

produktiota $A \rightarrow \omega$ saadaan käyttää vain kontekstissa $\alpha * \beta$.

Lineaarisesti rajoitettu Turingin kone on epädeterministinen yksinauhainen Turingin kone joka ei koskaan kirjoita mitään muuta niiden tyhjämerkkien päälle jotka nauhalla on alkutilanteessa. Siis kone käyttää ainoastaan syötteen pituuden verran nauhatilaa.

Lause Kieli voidaan tuottaa kontekstisella kieliopilla jos ja vain jos se voidaan tunnistaa lineaarisesti rajoitetulla Turingin koneella.

(Todistus sivuutetaan.)

Tämä on siis esimerkki laskennan mallista, jossa pelkästään syötteen koon perusteella saadaan yläraja sen vaatiman laskennan määrälle.

Universaaleilla laskennan malleilla näin **ei** ole, vaan lyhytkin syöte voi johtaa pitkään laskentaan.

Rekursiiviset funktiot (Gödel ja Kleene 1936)

Palautetaan mieliin:

- kieli A on **rekursiivisesti lueteltava** jos $A = L(M)$ jollain Turingin koneella M
- A on **rekursiivinen** jos lisäksi M pysähtyy kaikilla syötteillä
- merk. $\pi_A(x) = 1$ jos $x \in A$, $\pi_A(x) = 0$ muuten

Seuraavassa määritellään **rekursiiviset funktiot** $f: \mathbf{N} \rightarrow \mathbf{N}$ joille osoittautuu pätevän

- A on rekursiivinen kieli jos ja vain jos π_A on rekursiivinen funktio
- A on rekursiivisesti lueteltava jos ja vain jos jollain rekursiiviselle f pätee

$$A = \{ f(x) \mid x \in \mathbf{N} \}$$

(Tässä on samaistettu kokonaisluvut ja niiden esitykset binääriaakkostossa.)

Rekursiivisten funktioiden määritelmän idea on seuraava:

- tietyt alkeisfunktiot pitäisi ilman muuta osata laskea
- jos on annettu joitain perusfunktioita jotka osataan laskea, niin niistä tietyillä yksinkertaisilla operaatioilla muodostetut funktiot pitäisi myös osata laskea

Rekursiivisia funktioita ovat tasan ne jotka tämän logiikan mukaan pitäisi osata laskea.

Tässä on siis tavallaan deklarativinen määritelmä laskettavuudella, itse laskentaprosessista ei puhuta mitään. Tietysti intuitio laskentaprosesseista on vahvasti taustalla kun valitaan sopivat alkeisfunktiot ja operaatiot.

Määritellään ensin yleisemmin osittaisrekursiiviset funktiot $\mathbf{N}^k \rightarrow \mathbf{N}$.

Alkeisfunktiot

- nollafunktio $Z: \mathbf{N} \rightarrow \mathbf{N}$, $Z(x) = 0$ kaikilla x
- seuraajafunktio $S: \mathbf{N} \rightarrow \mathbf{N}$, $S(x) = x + 1$ kaikilla x
- kaikilla $n \in \mathbf{N}$ ja $1 \leq i \leq n$ projektiofunktio $U_i^n: \mathbf{N}^n \rightarrow \mathbf{N}$,
 $U_i^n(x_1, \dots, x_n) = x_i$ kaikilla $x_1, \dots, x_n \in \mathbf{N}$

Nämä ovat kaikki **totaalisia** funktioita eli määritelty kaikilla argumenttien arvoilla

Määritellään nyt operaatioita uusien funktioiden muodostamiseksi.

Sijoitus: kun on annettu $f: \mathbf{N}^n \rightarrow \mathbf{N}$ ja g_1, \dots, g_k jotka ovat funktioita $\mathbf{N}^m \rightarrow \mathbf{N}$, sijoitus tuottaa funktion

$$h: \mathbf{N}^m \rightarrow \mathbf{N}, h(\mathbf{x}) = f(g_1(\mathbf{x}), \dots, g_k(\mathbf{x}))$$

Rekursio: kun on annettu funktiot $f: \mathbf{N}^k \rightarrow \mathbf{N}$ ja $g: \mathbf{N}^{k+2} \rightarrow \mathbf{N}$, rekursio tuottaa sen funktion $h: \mathbf{N}^{k+1} \rightarrow \mathbf{N}$ jolle

$$\begin{aligned} h(\mathbf{x}, 0) &= f(\mathbf{x}) \\ h(\mathbf{x}, y + 1) &= g(\mathbf{x}, y, h(\mathbf{x}, y)) \end{aligned}$$

Jos jollain argumentilla \mathbf{x} esim. jokin $g_i(\mathbf{x})$ ei ole määritelty, niin myöskään sijoittamalla saatu $h(\mathbf{x})$ ei ole määritelty tällä \mathbf{x} jne.

Jos annetut funktiot ovat totaalisia, myös sijoittamalla ja rekursiolla syntyvät funktiot ovat.

Primitiivirekursiivisten funktioiden joukko \mathcal{PR} on pienin joukko joka

- sisältää funktiot Z , S ja U_i^n kaikilla i, n ja
- on suljettu sijoittamisen ja rekursion suhteen (ts. jos funktiot f ja g_i ovat joukossa \mathcal{PR} , myös niistä sijoituksella tai rekursiolla saadut funktiot ovat)

Siis \mathcal{PR} koostuu tasan niistä funktioista, jotka voidaan muodostaa alkeisfunktioista Z , S ja U_i^n sijoitusta ja rekursiota käyttäen.

Tästä seuraa että kaikki primitiivirekursiiviset funktiot ovat totaalisia.

Esimerkkejä

Identiteettifunktio $f(x) = x$ on sama kuin projektio U_1^1 .

Muodostetaan g yhdistämällä S ja U_3^3 , siis $g(x, y, z) = z + 1$.

Yhteenlaskufunktio $p(x, y) = x + y$ toteuttaa

$$\begin{aligned} p(x, 0) &= x \\ p(x, y + 1) &= p(x, y) + 1 \end{aligned}$$

joten se saadaan rekursiolla ylläolevista f ja g .

Kertolaskufunktio $m(x, y) = xy$ puolestaan toteuttaa

$$\begin{aligned} m(x, 0) &= 0 \\ m(x, y + 1) &= m(x, y) + x \end{aligned}$$

joten se saadaan edelleen rekursiolla yhteenlaskufunktion avulla.

- kaikki tavalliset aritmeettiset perusfunktiot nähdään helposti primitiivirekursiivisiksi
- yleisemmin primitiivirekursiivisia ovat täsmälleen ne funktiot, jotka voidaan laskea käyttäen vain **for**-silmukoita joissa iteraatioiden määrä pitää kiinnittää ennen silmukan suorituksen alkua
- on kuitenkin funktioita joiden laskemiseen tarvitaan yleisempää **while**-silmukkaa
- esim. voidaan osoittaa että **Ackermannin funktio**

$$\begin{aligned}\psi(0, y) &= y + 1 \\ \psi(x + 1, 0) &= \psi(x, 1) \\ \psi(x + 1, y + 1) &= \psi(x, \psi(x + 1, y))\end{aligned}$$

kasvaa nopeammin kuin mikään primitiivirekursiivinen funktio

⇒ universaaliin laskennan malliin tarvitaan jokin voimakkaampi mekanismi

Minimointi: kun on annettu funktio $f: \mathbf{N}^{k+1} \rightarrow \mathbf{N}$, minimointi tuottaa funktion $g: \mathbf{N} \rightarrow \mathbf{N}$ jolle

1. jos jollain y kaikki arvot $f(\mathbf{x}, 0), \dots, f(\mathbf{x}, y)$ ovat määriteltyjä ja $f(\mathbf{x}, y) = 0$, niin $g(\mathbf{x})$ on pienin tällainen y
 2. muuten $g(\mathbf{x})$ ei ole määritelty
- arvoa $y = g(\mathbf{x})$ voi tietysti etsiä laskemalla järjestyksessä $f(\mathbf{x}, 0)$, $f(\mathbf{x}, 2)$, $f(\mathbf{x}, 3)$, \dots ja pysähtymällä kun tulee nolla
 - etukäteen ei kuitenkaan ole mitään arviota, kuinka pitkälle joudutaan laskemaan

Osittaisrekursiivisten funktioiden joukko \mathcal{PR} on pienin joukko joka

- sisältää funktiot Z , S ja U_i^n kaikilla i, n ja
- on suljettu sijoittamisen, rekursion ja minimoinnin suhteen

Rekursiivisia ovat ne osittaisrekursiiviset funktiot jotka ovat totaaleja.

Esim. Ackermannin funktio ja kaikki primitiivirekursiiviset funktiot ovat rekursiivisia.

Kuten alussa todettiin, voidaan osoittaa että

- A on rekursiivinen kieli jos ja vain jos π_A on rekursiivinen funktio
- A on rekursiivisesti lueteltava jos ja vain jos jollain rekursiiviselle f pätee

$$A = \{ f(x) \mid x \in \mathbf{N} \}$$

- lisäksi A on rekursiivisesti lueteltava jos ja vain jos π_A on osittaisrekursiivinen

Random Access Machine (RAM)

- abstraktin tietokoneen konekieliohjelma
- koneessa rajoittamaton määrä rekistereitä jotka voivat sisältää mielivaltaisen suuren kokonaisluvun
- merkitään rekisterin j sisältöä r_j , $j = 0, 1, 2, \dots$
- rekisteri 0 **akku**; lisäksi käskyosoitin κ
- syötteenä luvut i_1, i_2, i_3, \dots
- tuloste on rekisterin 0 sisältö laskennan pysähtymishetkellä

RAMin käskykanta

käsky	merkitys
READ j	$r_0 := i_j$
READ $*j$	$r_0 := i_{r_j}$
STORE j	$r_j := r_0$
STORE $*j$	$r_{r_j} := r_0$
LOAD x	$r_0 := \text{val}(x)$
ADD x	$r_0 := r_0 + \text{val}(x)$
SUB x	$r_0 := r_0 - \text{val}(x)$

käsky	merkitys
HALF	$r_0 := \lfloor r_0/2 \rfloor$
JUMP j	$\kappa := j$
JPOS j	jos $r_0 > 0$ niin $\kappa := j$
JZERO j	jos $r_0 = 0$ niin $\kappa := j$
JNEG j	jos $r_0 < 0$ niin $\kappa := j$
HALT	laskenta pysähtyy

- r_j on rekisterin numero j sisältämä kokonaisluku
- x voi olla jokin vaihtoehdoista $\%j$, j tai $*j$ missä $j \in \mathbb{N}$.
- $\text{val}(\%j) = j$, $\text{val}(j) = r_j$ ja $\text{val}(*j) = r_{r_j}$

Samastetaan taas luonnolliset luvut binääriesitystensä kanssa ja tarkastellaan karakteristisia funktioita π_A missä $\pi_A(x) = 1$ jos $x \in A$ ja $\pi_A(x) = 0$ muuten.

Lause: Kieli A on rekursiivinen jos ja vain jos funktio π_A voidaan laskea RAMilla.

Todistushahmotelma: "Vain jos" -suunta on helppo: pitää osata simuloita Turingin konetta RAMilla, mikä on suoraviivainen ohjelmointiharjoitus.

"Jos" -suuntaa varten kontruoidaan annetulle RAMille sitä simuloiva 7-nauhainen Turingin kone. Seuraavassa esitetään vain nauhojen sisältö, yksityiskohdat sivuutetaan.

Nauha 1: syöteluvut sopivasti koodattuna

Nauha 2: rekisterien sisältö koodattuna jonoiksi $b(j) : b(r_j)$ missä $b(\cdot)$ tarkoittaa binääriesitystä; näiden jonojen välillä voi olla mielivaltaisen määrä tyhjämerkkejä

Nauha 3: käskyosoitin κ

Nauha 4: se indeksi j jota vastaavaa r_j ollaan etsimässä nauhalta 2

Nauhat 5–7: työnauhoja aritmetiikkaan jne.

” □ ”

Churchin-Turingin teesi

- intuitiivista laskettavuuden käsitettä yritetty mallintaa useista eri lähtökohdista
 - automaatit (Turing)
 - kieliopit
 - funktioluokkien sulkeumaominaisuudet (Gödel, Kleene)
 - elektronisen tietokoneen idealisointi (RAM)
 - lukuisia muita joita ei tässä ole mainittu.
 - kaikki johtivat kuitenkin samaan laskettavuuden käsitteeseen.
- ⇒ **Teesi:** Turing koneet (eli RAM, eli rekursiiviset funktiot, ...) on oikea matemaattinen malli sille, mitä on mahdollista laskea mekaanisesti
- ei matemaattinen väittämä
 - periaatteessa falsifioitavissa esittelemällä laskulaite joka laskee ei-rekursiivisiä funktioita

2. Laskettavuusteoriaa

Palautetaan mieliin terminologiaa:

- Kieli $L \subseteq \Sigma^*$ on **rekursiivisesti lueteltava** jos $A = L(M)$ jollain Turingin koneella M .
- Turingin kone on **totaalinen** jos se pysähtyy kaikilla syötteillä. (Joissain lähteissä käytetään termiä "algoritmi" spesifisti tällaisista koneista.)
- Kieli $L \subseteq \Sigma^*$ on **rekursiivinen** jos $A = L(M)$ jollain **totaalisella** Turingin koneella M .

Vastaavasti

- Päätösongelma $\pi: \Sigma^* \rightarrow \{0, 1\}$ on **osittain ratkeava** jos vastaava kieli
$$A_\pi = \{x \in \Sigma^* \mid \pi(x) = 1\}$$
on rekursiivisesti lueteltava.
- Päätösongelma π on **ratkeava** jos A_π on rekursiivinen.

Termien selityksiä

- "rekursiivinen" tulee siitä että tämä kieliluokka historiallisesti vakiintui Kleenen ja Gödelin rekursiokonstruktion kautta
- "lueteltava" tulee siitä että A on rekursiivisesti lueteltava jos ja vain jos on olemassa "algoritmi" joka "luettelee" joukon A alkiot (ja vain ne)
 - jos $x \in A$ niin x esiintyy luettelossa jonkin äärellisen ajan kuluttua ja asia on selvä
 - jos $x \notin A$ niin x ei tule koskaan esiintymään luettelossa, mutta tätähän ei voi tietää pelkästään katsomalla jotain luettelon äärellistä alkuosaa

Luetteloimisidea esitetään myöhemmin yksityiskohtaisemmin.

Formaalin logiikan todistuvuusongelma

(Motivoiva esimerkki yleisellä tasolla; yksityiskohdat ks. esim. Matemaattinen logiikka)

Annettu: ensimmäisen kertaluvun predikaattilogiikan kaava ϕ

Kysymys: onko kaavalle ϕ olemassa todistus predikaattilogiikan aksioomista

- voidaan "helposti" luetella ensin kaavat joilla on yhden askelen mittainen todistus, sitten ne joilla on kahden askelen mittainen todistus jne. joten
- siis todistuksen omaavien kaavojen joukko on rekursiivisesti lueteltava, ja todistuvuusongelma osittain ratkeava
- kuitenkin osoittautuu että todistuvuusongelma ei ole ratkeava

Pysähtymisongelma

Intuitiivisesti on kysymys seuraavasta ongelmasta:

Annettu: Turingin kone M , merkkijono x

Kysymys: pysähtyykö kone M syötteellä x

Tavoitteena on osoittaa tämä ongelma ratkeamattomaksi.

- ensin pitää tietysti esittää ongelma formaalisti jonkin aakkoston päätösongelmana (tai kielenä)
- erityisesti pitää sopia parin Turingin koneen M ja parin (M, x) esityksestä merkkijonona

Turingin koneiden koodaus

Rajoitetaan yksinauhaisiin koneisiin ja syöteaakkostoon $\Sigma = \{0, 1\}$. Lisäksi oletetaan että hyväksyviä tiloja on tasan yksi ja se ei ole alkutila.

Numeroidaan aakkoston $\{0, 1\}$ merkkijonot siten, että merkkijonon w numero on $1w$ binääriluvuksi tulkittuna. Olkoon w_i merkkijono numero i ; siis $w_1 = \varepsilon$, $w_2 = 0$, $w_3 = 1$, $w_4 = 00$, $w_5 = 01$ jne.

Oletetaan nyt $M = (Q, \{0, 1\}, \Gamma, \delta, q_1, \#, F)$ missä

- $|Q| = k$, $Q = \{q_1, \dots, q_k\}$ ja $F = \{q_2\}$
- $|\Gamma| = m \geq 3$, $\Gamma = \{X_1, \dots, X_m\}$, $X_1 = 0$, $X_2 = 1$, $X_3 = \#$
- suunnat on numeroitu $L = D_1$ ja $R = D_2$

Nyt kaikki muu paitsi δ on numeroitu.

- koodataan yksittäinen siirtymä $\delta(q_i, X_j) = (q_k, X_l, D_m)$ jonoksi $0^i 10^j 10^k 10^l 10^m$
- huom. $i, j, k, l, m \geq 1$ joten tässä ei koskaan tule kahta ykköstä peräkkäin
- merkkijono $C_1 11 C_2 11 \dots C_{n-1} 11 C_n$ on koodi koneelle jossa on n siirtymää joiden koodit ovat C_1, \dots, C_n

Huomaa että

- samalla koneella on tyypillisesti useita eri koodeja
- yksi merkkijono ei kuitenkaan voi koodata useita eri koneita
- jotkin merkkijonot eivät ole minkään koneen koodeja

Olkoon M_{triv} jokin kiinteä Turingin kone joka hylkää kaikki syötteen.

Määritellään kaikille $w \in \{0, 1\}^*$ Turingin kone M_w seuraavasti:

- jos w on jonkin koneen M koodi, niin M_w on tämä M
- muuten $M_w = M_{\text{triv}}$

Aputuloksena pysähtymisongelman ratkeamattomuustodistuksessa osoitetaan, että "diagonaalikieli"

$$L_d = \{ w \in \{0, 1\}^* \mid w \notin L(M_w) \}$$

ei ole edes rekursiivisesti lueteltava.

Lause: Kieli

$$L_d = \{ w \in \{0, 1\}^* \mid w \notin L(M_w) \}$$

ei ole rekursiivisesti lueteltava.

Todistus: Tehdään vastaoletus että $L_d = L(M)$ jollain M .

Olkoon w jokin koneen M koodi; siis $L_d = L(M_w)$. Nyt

$$w \in L_d \Leftrightarrow w \notin L(M_w) \Leftrightarrow w \notin L_d$$

missä on ensin käytetty kielen L_d määritelmää ja sitten koodin w valintaperustetta; ristiriita. □

Rekursiivisuuden perusominaisuuksia

Ennen kuin ruvetaan todistamaan pysähtymisongelman ratkeamattomuutta, on hyödyllistä tarkastella rekursiivisten kielten luokan joitain perusominaisuuksia.

Oletetaan jatkossa että Turingin koneissa on

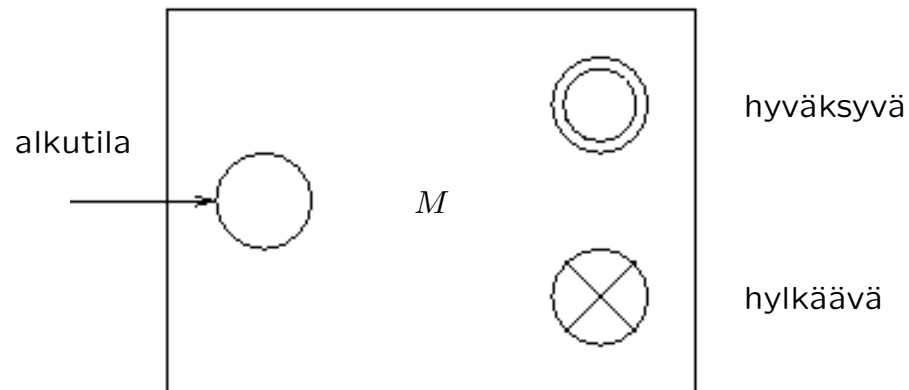
- tasan yksi hyväksyvä tila, ja tästä ei mitään siirtymiä
- tasan yksi sellainen ei-hyväksyvä tila, josta ei ole mitään siirtymiä (hylkäävä lopputila)
- muuten kaikki siirtymät määriteltyjä

On helppo muuntaa mikä tahansa Turingin kone tällaiseen muotoon.

Lause: Olkoot $A, B \subseteq \Sigma^*$ rekursiivisia. Nyt myös $\overline{A} = \Sigma^* - A$, $A \cup B$ ja $A \cap B$ ovat rekursiivisia.

Todistus: Olkoon $A = L(M_A)$ ja $B = L(M_B)$ missä M_A ja M_B ovat totaalisia.

Esitetään Turingin koneet kaavamaisesti:

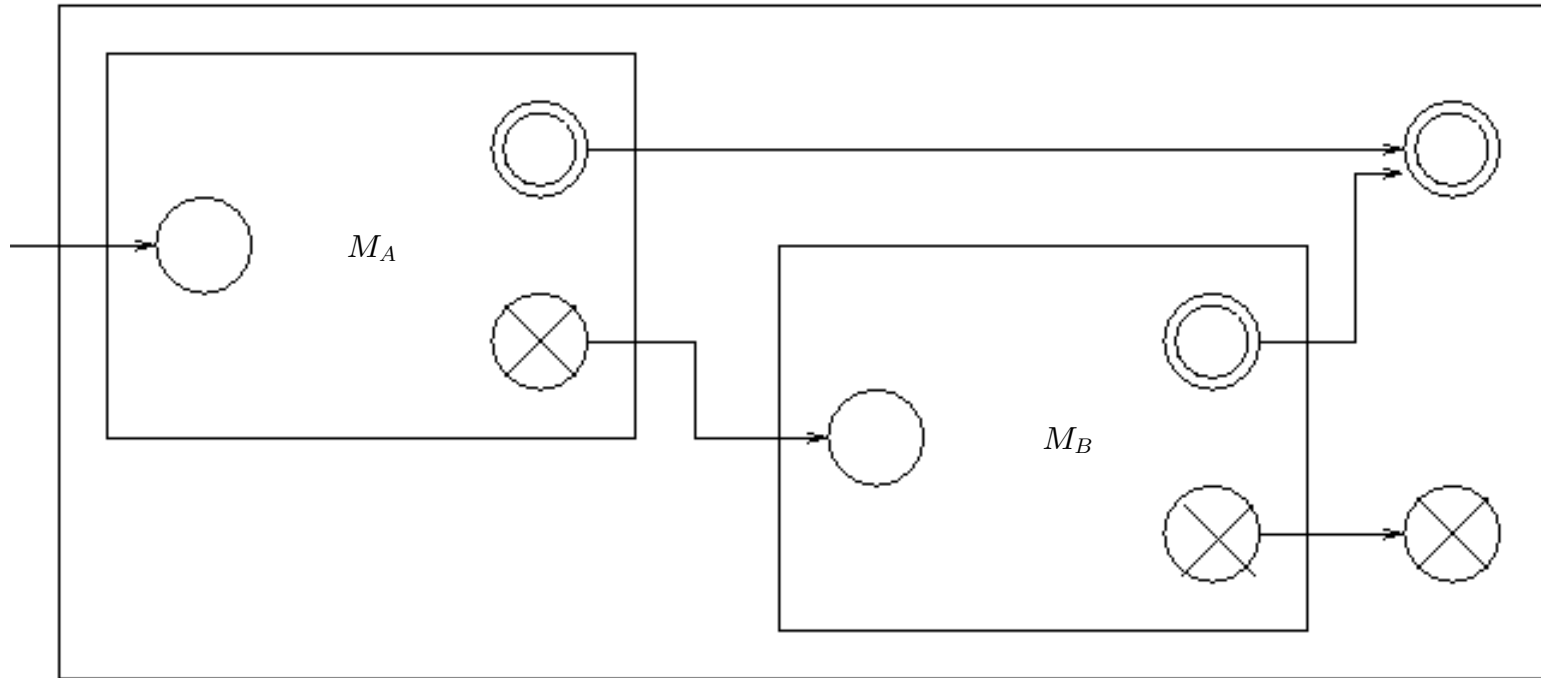


- Kielen \overline{A} hyväksyvä kone saadaan vaihtamalla koneen M_A hyväksyvä ja hylkäävä lopputila keskenään
- Kielen $A \cup B$ hyväksymiseksi simuloidaan ensin konetta M_A .
 - jos M_A hyväksyy, niin hyväksytään
 - jos M_A hylkää, niin simuloidaan konetta M_B jonka ratkaisu jää voimaan

(kuva seuraavalla sivulla)

- Tapaus $A \cap B$ seuraa koska $A \cap B = \overline{\overline{A} \cup \overline{B}}$





Kielen $A \cup B$ tunnistaminen koneiden M_A ja M_B avulla

Lause: Olkoot $A, B \subseteq \Sigma^*$ rekursiivisesti lueteltavia. Nyt myös $A \cup B$ ja $A \cap B$ ovat rekursiivisesti lueteltavia.

Todistus: Harjoitustehtävä. □

Sen sijaan yleisesti **ei** päde että rekursiivisesti lueteltavan kielen A komplementti \bar{A} olisi rekursiivisesti lueteltava.

Sen sijaan pätee

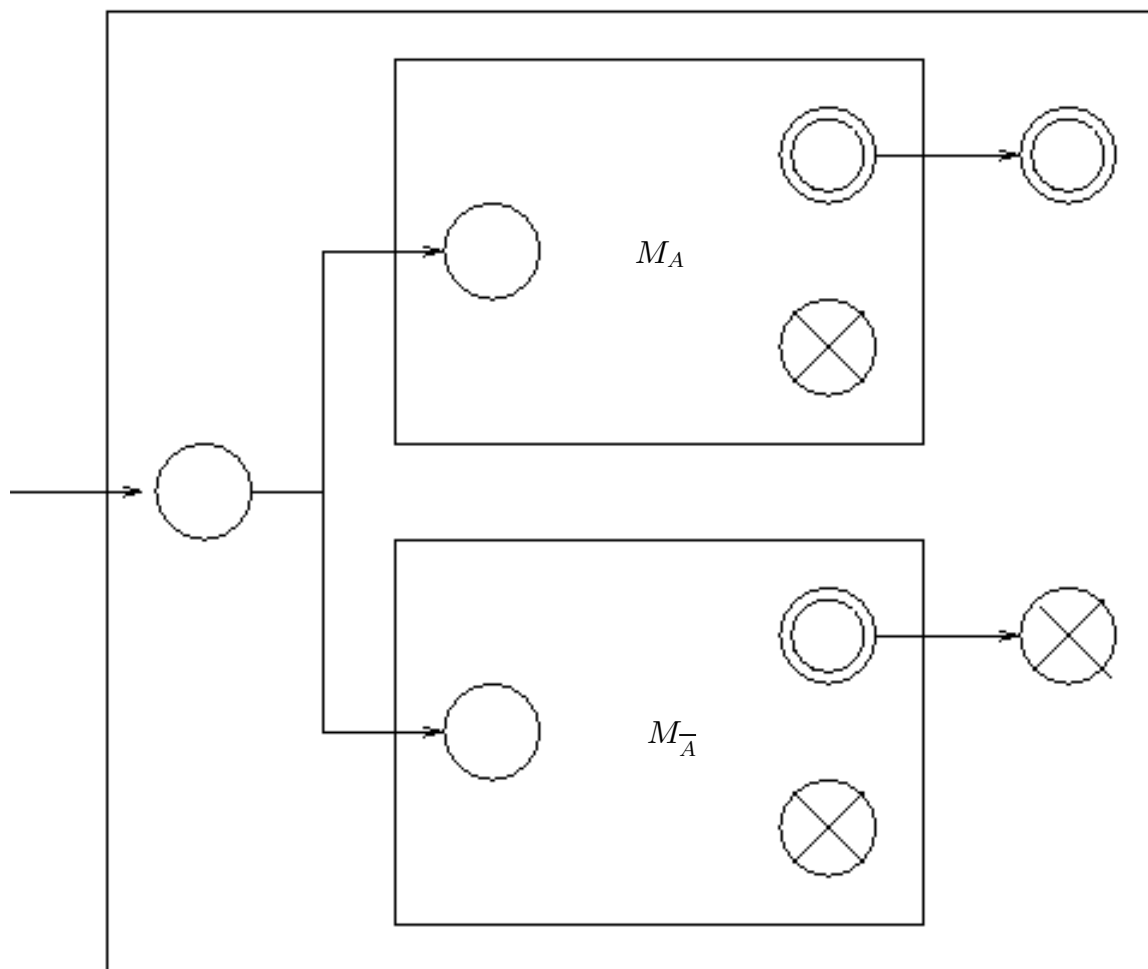
Lause: Kieli A on rekursiivinen jos ja vain jos sekä A että \bar{A} ovat rekursiivisesti lueteltavia.

Todistus: Jos A rekursiivinen, niin \bar{A} on rekursiivinen, joten suunta vasemmalta oikealle on selvä.

- olkoot M_A ja $M_{\bar{A}}$ koneet jotka tunnistavat kielet A ja \bar{A}
- muodostetaan kaksinauhainen kone M joka simuloi konetta M_A ykkösnauhalla ja konetta $M_{\bar{A}}$ kakkosnauhalla
- jos ykkössimulaatio hyväksyy, hyväksytään
- jos kakkossimulaatio hyväksyy, hylätään
- koska jokaisella x joko $x \in L(M_A)$ tai $x \in L(M_{\bar{A}})$, aina tasan yksi simulaatioista hyväksyy

(kuva seuraavalla sivulla)





Kielen A tunnistaminen totaalisella koneella jos A ja \bar{A} ovat rekursiivisesti lueteltavia

Formalistisemmin tulos voidaan esittää seuraavasti:

- olkoon RE rekursiivisesti lueteltavien kielten joukko:

$$RE = \{ L(M) \mid M \text{ mielivaltainen Turingin kone} \}$$

- olkoon co-RE rekursiivisesti lueteltavien kielten komplementtien joukko:

$$\text{co-RE} = \{ A \mid \bar{A} \in RE \}$$

- olkoon REC rekursiivisten kielten joukko:

$$REC = \{ L(M) \mid M \text{ totaalinen} \}$$

Edelläesitetyn mukaan

$$REC = RE \cap \text{co-RE}.$$

Huomaa yhteys "luettelemisen" ajatukseen:

- oletetaan, että jokin "algoritmi" M_A osaa luetella kielen A ; samoin $M_{\bar{A}}$ kielelle \bar{A}
- halutaan tietää päteekö $x \in A$
- listataan rinnakkain joukkoja A ja \bar{A}
- koska joko $x \in A$ tai $x \in \bar{A}$, niin jonkin äärellisen ajan kuluttua x esiintyy jommassa kummassa listassa ja vastaus tiedetään
- myös \bar{A} -lista tarvitaan jotta voidaan taata pysähtyminen myös kielteisessä tapauksessa

Universaalikieli

Seuraava askel kohti pysähtymisongelman ratkeamattomuutta on **universaalikieli** L_u joka on muutenkin hyvin tärkeä:

$$L_u = \{ w111x \in \{0,1\}^* \mid x \in L(M_w) \}.$$

Huom.

- mikään Turingin koneen koodi ei sisällä merkkijonoa 111
- siis merkkijono $z \in \{0,1\}^*$ voidaan esittää korkeintaan yhdellä tavalla muodossa $z = w111x$ siten, että w on validi Turingin koneen koodi
- jos w ei ole validi koodi, edellä sovitun mukaan $x \notin L(M_w)$ kaikilla x

Osoitamme nyt että universaalikieli on rekursiivisesti lueteltava, mutta ei rekursiivinen.

Kielen L_u tunnistavaa Turingin konetta U sanotaan **universaaliksi Turingin koneeksi**.

(Universaalikone ei tietenkään ole yksikäsitteinen.)

Lause: Universaalikieli L_u on rekursiivisesti lueteltava.

Todistus: Muodostetaan nelinauhainen U jolle $L_u = L(U)$.

Nauhojen käyttö syötteellä $z = w111x$ missä w on koneen M koodi:

Nauha 1 sisältää syötteen z ja siis erityisesti koneen M siirtymäfunktion koodin w

Nauha 2 simuloi koneen M nauhan sisältöä käyttäen samaa koodausta kuin siirtymäfunktiossa; siis esim. pätkää $\dots X_3 X_1 X_4 \dots$ esittäisi $\dots 100010100001 \dots$

Nauha 3 simuloi koneen M tilaa; tila q_i koodataan 0^i

Nauha 4 on työtilaa

Koneen U laskenta:

Aluksi tarkista onko syöte muotoa $w11x$ jollain validilla koodilla w . Jos ei ole, niin hylkää. Muuten rupea simuloimaan koneta $M = M_w$ syötteellä x .

Kussakin askeleessa

- olkoon nauhalla 3 jono $\dots \#0^i\# \dots$ ja nauhalla 2 nauhapäästä alkaen jono 0^j1
- etsi koneen M kuvauksesta nauhalla 1 kohta $\dots 110^i10^j \dots$; jos ei löydy, niin hylkää
- olkoon nauhalta 1 löytynyt jono $\dots 110^i10^j10^k10^l10^m11 \dots$
- vaihda nauhan 3 sisällöksi 0^k ja nauhalle 2 nauhapäästä alkaen 0^l ; siirrä nauhan 2 loppuosuutta tarpeen mukaan
- siirrä nauhan 2 nauhapää seuraavaan ykköseen vasemmalla (jos $m = 1$) tai oikealla (jos $m = 2$).
- jos q_k on hyväksyvä tila (siis $k = 2$), hyväksy



Lause: Universaalikieli L_u ei ole rekursiivinen.

Todistus: Tehdään vastaoletus että $L_u = L(M)$ jollain totaalisella M . Muodostetaan totaalinen M' jolle $L(M') = L_d$, missä L_d on aiemmin ei-rekursiiviseksi osoitettu diagonaalikieli; ristiriita.

Kone M' toimii syötteellä w seuraavasti:

- jos w ei ole validi koodi, hyväksy
- muuten muunna nauhan sisällöksi $w111w$
- simuloi koneen M laskentaa; oletuksen mukaan tämä johtaa joko hylkäävään tai hyväksyvään lopputilaan
- jos M hyväksyy, niin hylkää; jos M hylkää, niin hyväksy

Nyt

$$w \in L(M') \Leftrightarrow (w \text{ ei validi koodi tai } w111w \notin L_u) \Leftrightarrow w \notin L(M_w) \Leftrightarrow w \in L_d.$$

□

Pysähtymisongelma

Voimme nyt formuloida pysähtymisongelman kieleksi

$$H = \{ w111x \in \{0,1\}^* \mid w \text{ validi koodi ja } M_w \text{ pysähtyy syötteellä } x \}.$$

Lause: Kieli H on rekursiivisesti lueteltava.

Todistus: Universaalikoneen U konstruktiota on helppo muuttaa siten että se hyväksyy jos ja vain jos simuloitavan koneen laskenta pysähtyy. \square

Lause: Kieli H ei ole rekursiivinen.

Todistus: Tehdään vastaoletus että $H = L(M)$ jollain totaalilla M .

Tästä saadaan helposti sellainen totaali M' , että $L(M') = H$ ja hyväksyessään syötteen x kone M' jättää laskennan lopuksi nauhalle alkuperäisen syötteen tyhjämerkkien ympäröimänä.

Tästä saadaan totaalinen kone M'' joka tunnistaa universaalikielen L_u seuraavasti:

- tarkista että syöte on muotoa $w111x$ missä w on validi koodi; jos ei ole, niin hylkää
- simuloi sitten konetta M' ; jos hylkäsi niin hylkää
- jos M' hyväksyi, käsittele sama syöte universaalikoneella U jonka hyväksyminen tai hylkääminen jää voimaan

Universaalikoneen konstruktiosta nähdään, että U pysähtyy syötteellä $w111x$ jos ja vain jos M_w pysähtyy syötteellä x .

Siis M'' on totaali ja tunnistaa saman kielen kuin U ; ristiriita □

Lisää pysähtymisaiheisia ongelmia

Lause: "Pysähtymättömyysongelma" \tilde{H} missä

$$\tilde{H} = \{w111x \mid w \text{ validi koodi, } M_w \text{ ei pysähdy syötteellä } x\}$$

ei ole rekursiivisesti lueteltava.

Todistus: Pysähtymisongelman komplementti \overline{H} voidaan esittää muodossa $\overline{H} = \tilde{H} \cup E$ missä

$$E = \{z \in \{0, 1\}^n \mid z \text{ ei ole } w111x \text{ millään validilla koodilla } w\}.$$

- selvästi E rekursiivinen
- siis jos \tilde{H} olisi rekursiivisesti lueteltava, myös $\overline{H} = \tilde{H} \cup E$ olisi
- H on rekursiivisesti lueteltava, joten jos \overline{H} myös olisi, niin H olisi rekursiivinen; ristiriita.
- siis \tilde{H} ei voi olla rekursiivisesti lueteltava □

Ohjelmointikielen pysähtymisongelma

Turingin koneet ovat analogisia jollain ohjelmointikielellä kirjoitettujen ohjelmien kanssa:

Turingin kone -formalismi	~	jonkin ohjelmointikieli
Turingin kone	~	tällä kielellä kirjoitettu ohjelma
Turingin koneen koodi	~	ohjelma käännettynä konekielelle
universaali Turingin kone	~	konekielen tulkki

- joitain ohjelmointikielten ratkeamattomuustuloksia voidaan kätevästi osoittaa suoraan (kuten johdannossa C-kielen pysähtymisongelma)
- voidaan myös käyttää hyväksi Turingin koneiden ja minkä tahansa yleisohjelmointikielen samaa ilmaisuvoimaa ja siirtää tulokset Turingin koneista ohjelmointikieliin
- "sama ilmaisuvoima" tarkoittaa tässä, että mikä tahansa ongelma voidaan ratkaista Turingin koneella jos ja vain jos se voidaan ratkaista kyseisellä ohjelmointikielellä

Esimerkki: C-kielen pysähtymisongelma

- rajoitutaan tarkastelemaan C-kielen funktioita jotka saavat parametrinä yhden merkkijonon ja palauttavat 0 tai 1 elleivät jää silmukkaan
- sanotaan että tällainen funktio f hyväksyy merkkijonon x jos $f(x)$ palauttaa 1, ja on **totaalinen** jos $f(x)$ pysähtyy kaikilla x
- ollaan valmiit uskomaan, että kieli A voidaan tunnistaa (totaalisella) Turingin koneella jos ja vain jos se voidaan tunnistaa tällaisella (totaalisella) C-funktiolla
- tämän uskomuksen tarkka perustelemine vaatisi tietysti C-kielen semantiikan tarkkaa läpikäymistä, mutta intuitiivisesti se on "selvästi" totta (vrt. RAM-malli)
- Halutaan osoittaa Turingin koneiden pysähtymisongelmalle analoginen tulos: mikään totaalinen C-funktio ei tunnista kieltä

$$H_C = \{ w * x \mid w \text{ on C-funktio joka syötteellä } x \text{ pysähtyy} \}$$

Argumentti menee pääpiirtessään seuraavasti:

- koska C on yhtä ilmaisuvoimainen kuin Turingin koneet, voidaan Turingin koneiden universaalikieli L_u tunnistaa jollain (ei-totaalisella) C-funktiolla **univT**
- siis kysymykset Turingin koneen M_w toiminnasta syötteellä x palautuvat kysymyksiksi C-funktion **univT** toiminnasta syötteellä $w111x$.
- erityisesti jos jokin C-funktio **halts** ratkaisisi C-funktioiden pysähtymisongelman, niin tätä kautta se ratkaisisi myös Turingin koneiden pysähtymisongelman
- koska toisaalta Turingin koneet ovat yhtä ilmaisuvoimaisia kuin C-funktiot, funktiota **halts** voitaisiin simuloida Turingin koneella
- näin saataisiin ratkaistuksi Turingin koneiden pysähtymisongelma; ristiriita

Esitetään sama hieman yksityiskohtaisemmin. Vastaoletus siis on, että jokin totaalinen C-funktio **haltsC** ratkaisee C-kielen pysähtymisongelman.

- on olemassa C-funktio f_u joka simuloi universaalia Turingin konetta U
- siis f_u hyväksyy merkkijonon $w111x$ jos ja vain jos $x \in L(M_w)$
- lisäksi f_u syötteellä $w111x$ pysähtyy jos ja vain jos M_w syötteellä x pysähtyy
- olkoon **haltsT** C-funktio joka syötteellä z tekee kutsun **haltsC**(p^*z) missä p on funktion f_u teksti
- siis **haltsT** on totaalinen ja **haltsT**($w111x$) = 1 jos ja vain jos $x \in L(M_w)$
- on olemassa totaalinen Turingin kone joka simuloi funktiota **haltsT** ja siis ratkaisee Turingin koneiden pysähtymisongelman; **ristiriita**

Rekursiiviset palautukset

Yleiskäyttöinen työkalu ratkeamattomuusongelmien todistamiseen

Idea: Määritellään laskennallisten ongelmien relaatio $A \leq B$, "ongelma A voidaan palauttaa ongelmaan B "

Intuitiivinen tulkinta: Kun $A \leq B$ niin

- B on ainakin yhtä vaikea kuin A
- ongelma A voidaan ratkaista ongelman B avulla

Tyypillinen käytötapa: Halutaan osoittaa ongelma B "vaikeaksi". Osoitetaan $A \leq B$ jollain vaikeaksi tunnetulla A .

Palautuksen käsite voidaan määritellä useilla idealtaan samansuuntaisilla mutta ei-yhtäpitävillä tavoilla. Tässä tarkasteltava palautustyyppi perustuu rekursiivisiin funktioihin.

Rekursiiviset funktiot

Olkoon f osittaisfunktio joukosta Σ^* joukkoon Γ^* . Siis sallitaan että $f(x)$ on määrittelemätön joillain $x \in \Sigma^*$.

Turingin kone M laskee osittaisfunktion f , jos syötteellä x

- mikäli $f(x)$ on määritelty, niin M pysähtyy, ja tällöin sen nauhalla on $f(x)$ (ja tyhjämerkkejä)
- muuten M ei pysähdy

Tällöin merkitään $f = f_M$.

Osittaisfunktio $f: \Sigma^* \rightarrow \Gamma^*$ on osittaisrekursiivinen jos jokin Turingin kone laskee sen.

Jos funktio on lisäksi totaalinen (eli määritelty koko joukossa Σ^*), se on rekursiivinen.

Jatkossa tarvitaan lähinnä totaalisia funktioita (ja yleensä "funktio" tarkoittaa totaalista funktioita). Osittaisrekursiiviset funktiot ovat kuitenkin tärkeitä laskennan teoriassa.

Olkoot $A \subseteq \Sigma^*$ ja $B \subseteq \Gamma^*$.

Funktio $f: \Sigma^* \rightarrow \Gamma^*$ on **rekursiivinen palautus** kielestä A kieleen B (eli kielen A rekursiivinen palautus kieleen B) jos

- f on rekursiivinen ja
- kaikilla $x \in \Sigma^*$ pätee $x \in A$ jos ja vain jos $f(x) \in B$.

Tällöin merkitään $f: A \leq_m B$.

Kieli A **palautuu rekursiivisesti** kieleen B jos on olemassa rekursiivinen palautus kielestä A kieleen B . Tällöin merkitään $A \leq_m B$.

Lause: Olkoon $A \leq_m B$ ja B rekursiivinen (rekursiivisesti lueteltava). Tällöin A on rekursiivinen (vast. rekursiivisesti lueteltava).

Todistus: Helppo. □

Oletetaan $f: A \leq_m B$.

Idea: jos ongelmalla " $x \in B$?" on ratkaisualgoritmi, niin saamme heti algoritmin myös ongelmalla " $x \in A$?":

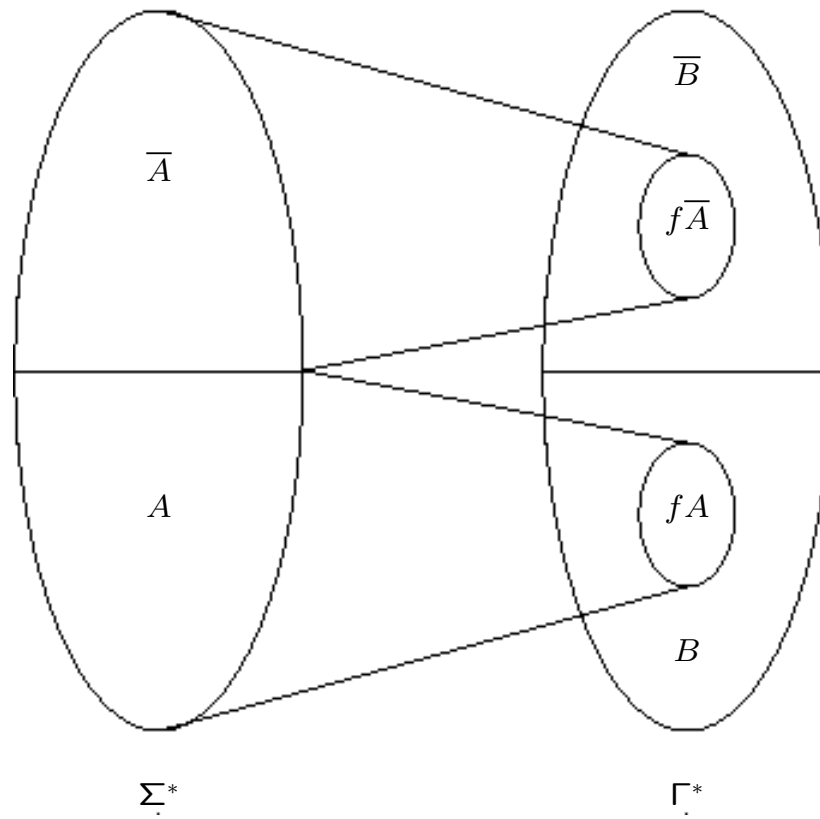
1. laske $y = f(x)$
2. ratkaise " $y \in B$?"

Mutta: Tässä määritelty palautuksen käsite \leq_m ei suinkaan ole ainoa mahdollinen. Voitaisiin esim. sallia laskennan aikana useampi kuin yksi muotoa " $y \in B$?" -tyyppinen kysymys.

Siis $A \leq_m B$ on vahvempi väite kuin pelkästään että " A voidaan ratkaista B :n avulla".

On olemassa muita palautuksen käsitteitä joita ei kuitenkaan tällä kurssilla käsitellä.

(Alaindeksi m tulee termistä many-one, koska tässä sallitaan sama arvo $f(x) = f(y)$ eri alkiuille $x \neq y$.)



Periaatekuva palautuksesta $f: A \leq_m B$

Epätyhjyysongelma

Esimerkkinä palautustekniikasta, ja johdatuksena yleisempään Ricen lauseeseen, tarkastellaan kieltä

$$L_{ne} = \{w \in \{0, 1\}^* \mid L(M_w) \neq \emptyset\}.$$

Ongelmana on siis päättää annetusta Turingin koneesta, hyväksyykö se ylipäänsä mitään merkkijonoja.

Lause: Kieli L_{ne} on rekursiivisesti lueteltava mutta ei rekursiivinen.

Todistus Kiinnostavampi puoli väitteestä on ei-rekursiivisuus, mutta tarkastetaan kuitenkin ensin rekursiivinen lueteltavuus. Tämä käy helpoimmin konstruoimalla epädeterministinen Turingin kone M jolle $L_{ne} = L(M)$.

Koneen M laskenta etenee seuraavasti:

1. Kirjoita nauhalle syötteen w jatkoksi 111 ja epädeterministisesti generoitu mielivaltainen $x \in \{0, 1\}^*$.
2. Palaa nauhan alkuun ja simuloi universaalikonetta U syötteellä $w111x$.

Nyt

$$\begin{aligned}w \in L_{ne} &\Leftrightarrow x \in L(M_w) \quad \text{jollain } x \in \{0, 1\}^* \\ &\Leftrightarrow w111x \in L(U) \quad \text{jollain } x \in \{0, 1\}^* \\ &\Leftrightarrow w \in L(M).\end{aligned}$$

□/2

Ei-rekursiivisuus osoitetaan palautuksella universaalikiielestä L_u .

- siis muodostetaan funktio f jolle $f(z) \in L_{ne}$ jos ja vain jos $z \in L_u$
- lisäksi funktio f on rekursiivinen.
- nyt $f: L_u \leq_m L_{ne}$, joten jos L_{ne} olisi rekursiivinen niin myös L_u olisi; ristiriita.

Funktion f pitää siis annetulla syötteellä $z = w111x$ muodostaa Turingin kone M (tai oikeastaan sen koodi) siten, että

- jos $x \notin L(M_w)$ niin M ei hyväksy yhtään merkkijonoa
- jos $x \in L(M_w)$ niin M hyväksyy ainakin jonkin merkkijonon
- tässä itse asiassa käy niin että jos $x \in L(M_w)$ niin M hyväksyy kaikki merkkijonot

Tarkastellaan jotain kiinteää $z = w111x$. Syötteellä y kone M toimii seuraavasti:

- Pyyhi syöte y pois nauhalta; nauha on nyt tyhjä.
- Kopioi nauhalle merkkijono x ; palauta nauhapää alkuun.
- Simuloi konetta M_w .

Konstruktion perusteella on ilmeistä että

- jos $w111x \in L_u$ niin $L(M) = \Sigma^*$
- jos $w111x \notin L_u$ niin $L(M) = \emptyset$

Siis jos on olemassa rekursiivinen funktio joka syötteellä z tuottaa koodin v siten, että kone M_v toimii kuten M yllä, todistus on valmis.

Tarkastellaan nyt tarkemmin, millaisia tiloja ja siirtymiä yllä kuvatussa koneessa $M = M_v$ pitäisi olla. Olkoon $z = w111x$ missä $x = x_1 \dots x_n$.

- pohjana toimii kone M_w
- lisätään alkutilaksi uusi tila \hat{q}_0 , jossa kone kirjoittaa nauhalle tyhjää niin kauan kuin ei-tyhjämerkkejä riittää. Kun tulee vastaan tyhjä, siirrytään uuteen tilaan \hat{q}_1 .
- lisätään tilat \hat{q}_i , $i = 1, \dots, n$, missä tilassa \hat{q}_i , $1 \leq i \leq n - 1$, kone kirjoittaa merkin x_i , siirtyy tilaan \hat{q}_{i+1} ja siirtää nauhapäätä oikealla.
- tilasta \hat{q}_n siirrytään uuteen tilaan \hat{q}_{n+1} jossa kone siirtää nauhapäätä takaisin vasemmalle kunnes taas löytyy tyhjämerkki
- tilasta \hat{q}_{n+1} siirrytään alkuperäisen koneen M_w alkutilaan
- siirtymät koneen M_w alkuperäisten tilojen välillä säilyvät ennallaan, samoin lopputilat

- selvästi yllä kuvattu $M = M_v$ toimii kuten väitetään
- palautus on nyt funktio f joka laskee syötteen $z = w111x$ perusteella koodin v yllä kuvatulle koneelle
- on intuitiivisesti selvää että tämä f voidaan laskea Turingin koneella.

Siis $f: L_u \leq_m L_{ne}$; kieli L_{ne} ei ole rekursiivinen.



Lause: *Tyhjyysoongelma* ei ole osittain ratkeava; ts. kieli

$$L_e = \{w \in \{0, 1\}^* \mid L(M_w) = \emptyset\}$$

ei ole rekursiivisesti lueteltava.

Todistus: Aiemmin esitetyn mukaan jos A ja \overline{A} ovat rekursiivisesti lueteltavia, niin A on rekursiivinen.

Koska $L_e = \overline{L_{ne}}$ ja L_{ne} juuri todistettiin ei-rekursiiviseksi mutta rekursiivisesti lueteltavaksi, L_e ei voi olla rekursiivisesti lueteltava. \square

Esimerkkinä yksinkertaisesta palautuksesta tarkastellaan kysymystä, onko kahden rekursiivisesti lueteltavan kielen leikkaus tyhjä.

Lause: Kieli

$$L_{ep} = \{v111w \mid v, w \in \{0, 1\}^*, L(M_v) \cap L(M_w) = \emptyset\}$$

ei ole rekursiivisesti lueteltava.

Todistus: Palautus kielestä L_e . Siis muodostetaan rekursiivinen funktio

$f: \{0, 1\}^* \rightarrow \{0, 1\}^*$, jolle pätee

$$f(v) \in L_{ep} \Leftrightarrow v \in L_e.$$

Olkoon w_{all} jokin sellainen koodi, että $L(M_{w_{\text{all}}}) = \{0, 1\}^*$.

Palautusfunktio f määritellään

$$f(v) = v111w_{\text{all}}.$$

Nyt jos v on validi koodi, pätee $L(M_v) = L(M_v) \cap \{0, 1\}^* = L(M_v) \cap L(M_{w_{\text{all}}})$ ja siis

$$v \in L_e \Leftrightarrow L(M_v) = \emptyset \Leftrightarrow L(M_v) \cap L(M_{w_{\text{all}}}) = \emptyset \Leftrightarrow f(v) \in L_{ep}.$$

Jos v ei ole validi koodi, sopimuksen mukaan $L(M_v) = \emptyset$, ja koska tällöin ei voi olla mitään valideja koodeja v', w' joille $v111w_{\text{all}} = v'111w'$, myös $f(v) \in L_{ep}$.

Funktio f voidaan helposti laskea Turingin koneella:

1. Siirrä nauhapää syötemerkkijonon loppuun.
2. Kirjoita nauhalle merkkijono $111w_{\text{all}}$.
3. Palauta nauhapää syötteen alkuun.

Siis on muodostettu rekursiivinen palautus $f: L_e \leq_m L_{ep}$. Jos L_{ep} olisi rekursiivisesti lueteltava, myös L_e olisi; ristiriita. □

Semanttisten ongelmien ratkeamattomuus

Intuitiivisesti semanttinen ongelma on Turingin konetta koskeva kysymys, jonka vastaus riippuu vain koneen hyväksymästä kielestä. Siis jos $L(M) = L(M')$, vastauksen on oltava sama koneille M ja M' .

Muodollisemmin **semanttinen ominaisuus** \mathcal{S} on kokoelma rekursiivisesti lueteltavia aakkoston $\{0, 1\}$ kieliä. Toisin sanoen $\mathcal{S} \subseteq \text{RE}$, missä siis

$$\text{RE} = \{ L(M) \subseteq \{0, 1\}^* \mid M \text{ Turingin kone syöteaakkostolla } \{0, 1\} \}$$

Ominaisuus \mathcal{S} on **ratkeava** jos joukko

$$L_{\mathcal{S}} = \{ w \in \{0, 1\}^n \mid L(M_w) \in \mathcal{S} \}$$

on rekursiivinen.

Esimerkki: Valitaan $\mathcal{S} = \{\emptyset\}$, siis pelkän tyhjän kielen sisältävä luokka. Nyt $L_{\mathcal{S}} = L_e$, edellä esitetty tyhjiysongelma, joten \mathcal{S} ei ole ratkeava.

Semanttinen ominaisuus \mathcal{S} on **triviaali** jos joko $\mathcal{S} = \emptyset$ tai $\mathcal{S} = \text{RE}$. Siis ominaisuus on ei-triviaali jos on olemassa kaksi kieltä $L, L' \in \text{RE}$ joille $L \in \mathcal{S}$ ja $L' \notin \mathcal{S}$.

Ricen lause: Kaikki ei-triviaalit semanttiset ominaisuudet ovat ratkeamattomia.

Siis esim. seuraavat ongelmat ovat ratkeamattomia:

- hyväksyykö M tasan k merkkijonoa (millä tahansa k)
- hyväksyykö M äärettömän monta merkkijonoa
- onko koneen M hyväksymä kieli säännöllinen
- onko koneen M hyväksymä kieli kontekstiton

(kaikissa ongelmissa siis syötteenä koneen M koodi)

Ricen lauseen todistus: Olkoon \mathcal{S} ei-triviaali. Jatkossa esitetään todistus tapaukselle $\emptyset \notin \mathcal{S}$. Jos $\emptyset \in \mathcal{S}$, voidaan samaa todistusta soveltaa ominaisuuteen $\overline{\mathcal{S}} = \text{RE} - \mathcal{S}$ joka on ei-triviaali eikä sisällä kieltä \emptyset .

Todistus perustuu palautukseen universaalikiielestä L_u . Siis muodostetaan rekursiivinen $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ jolle $f(v) \in L_{\mathcal{S}}$ jos ja vain jos $v \in L_u$.

Toisin sanoen on oltava $L(M_{f(v)}) \in \mathcal{S}$ jos ja vain jos $v = w111x$ missä $x \in L(M_w)$.

Kun v ei ole muotoa $v = w111x$ millekään koodille w , määritellään $f(v) = w_0$ missä w_0 on jokin kiinteä koodi jolle $L(M_{w_0}) \notin \mathcal{S}$.

Tarkastellaan nyt miten $f(v)$ määritellään kun $v = w111x$ missä w on validi koodi jollekin koneelle M .

Nyt on siis annettuna $M = M_w$ ja x , ja pitää konstruoida $M' = M_v$ jolle $L(M') \in \mathcal{S}$ jos ja vain jos $x \in L(M)$. (Ja sitten asetetaan $f(w111x) = v$.)

Olkoon M_L jokin kiinteä kone jolle $L(M_L) \in \mathcal{S}$. Kone M' konstruoidaan parin (M, x) perusteella siten, että

- jos $x \in L(M)$ niin $L(M') = L(M_L)$
- jos $x \notin L(M)$ niin $L(M') = \emptyset$

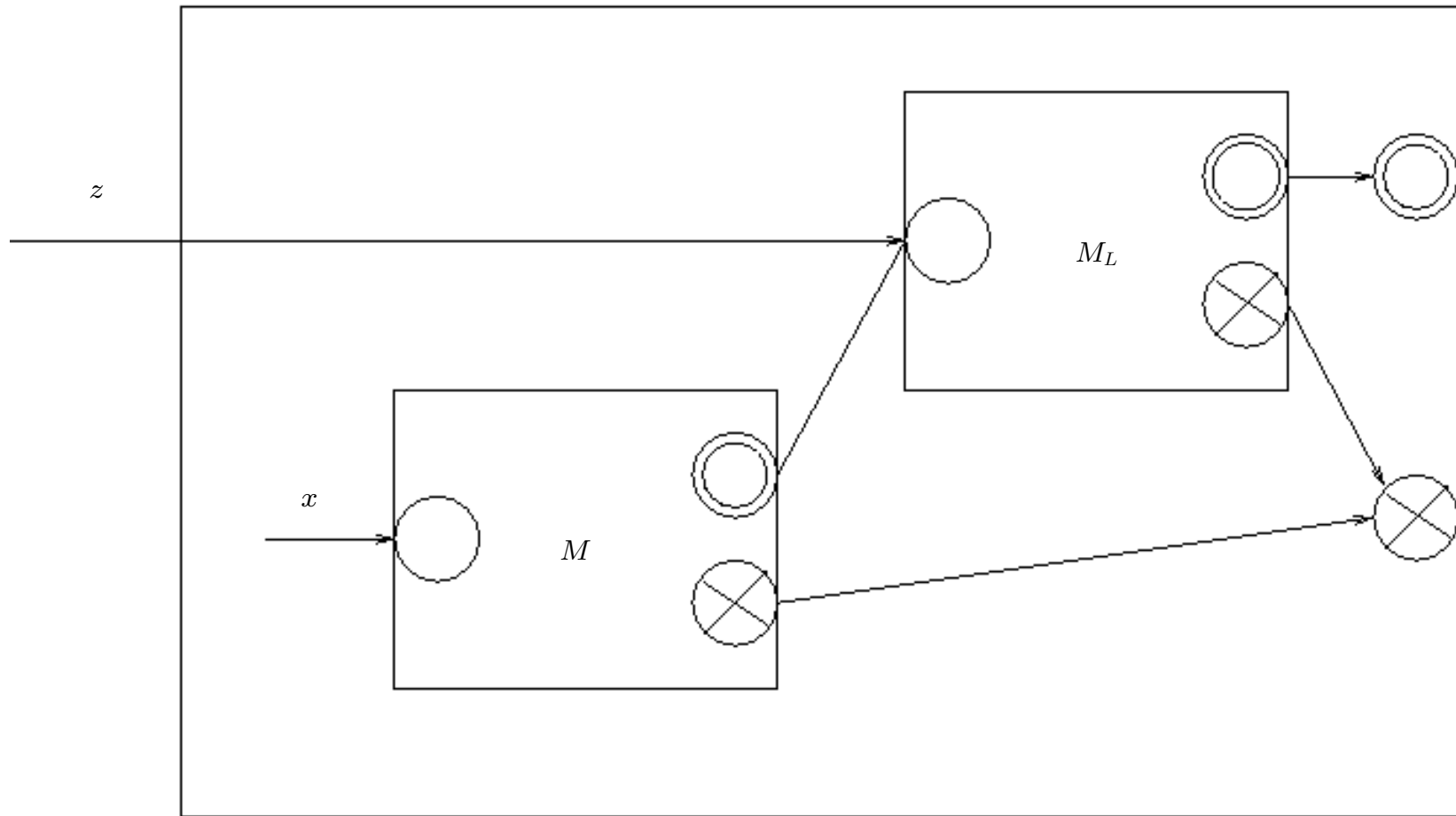
mikä toteuttaa vaatimukset.

Kone M' on kaksinauhainen ja toimii syötteellä z seuraavasti:

1. Kopio merkkijono x kakkosnauhalle.
2. Kakkosnauhaa käyttäen simuloi koneen M laskenta syötteellä x
3. Jos simulointi johti hylkäämiseen, hylkää z . (Tässä vaiheessa syötettä z ei ole vilkaistukaan!)
4. Jos simulointi johti hyväksymiseen, simuloi koneen M_L laskentaa varsinaisella syötteellä z ; hyväksy jos M_L hyväksyy.

Nyt $L(M') = L(M_L)$ jos $x \in L(M)$; muuten $L(M') = \emptyset$.

Selvästi merkkijonosta $v = w111x$ missä w on koneen M indeksi voidaan laskea $f(v)$ joka on kuvatun koneen M' indeksi. Huomaa että x tulee koodatuksi koneen M' sisään eikä ole sen syöte. □



Koneen M' konstruktio Ricen lauseessa

Muita ratkeamattomuustuloksia

Ricen lause voidaan ilmeisellä tavalla siirtää Turingin koneista koskemaan esim. C-kielisiä ohjelmia. Turingin koneiden semanttiset ominaisuudet vastaavat C-ohjelmien syöte-tuloste-relaation ominaisuuksia.

On siis esim. ratkeamatonta

- pysähtyykö ohjelma millään syötteellä
- pysähtyykö ohjelma kaikilla syötteillä
- tulostaako ohjelma "1" millään syötteellä
- jne.

Turingin koneille on ratkeamatonta, käykö kone annetussa tilassa $q \in Q$ millään syötteellä. Vastaavasti on ohjelmille on ratkeamatonta

- suoritetaanko ohjelman riviä numero k millään syötteellä

Matematiikan ratkeamattomuustuloksia

Klassinen tulos (Church, Gödel 1930-luvulla) sanoo että seuraava ongelma on ratkeamaton:

Annettu: luonnollisten lukujen aritmetiikkaa koskeva 1. kertaluvun logiikan kaava ϕ

Kysymys: onko ϕ tosi

Tässä ei mennä tarkemmin siihen, miten kaavat ja niiden totuus muodollisesti määritellään. Huomataan kuitenkin, että jos rajoitutaan kaavoihin ilman kertolaskua, ongelma on ratkeava.

Yllättäen Matijasevitsh 1970 osoitti paljon vahvemmin, että jo seuraava paljon rajoitetumpi ongelma on ratkeamaton:

Annettu: kokonaislukukertoiminen n muuttujan polynomi $p(x_1, \dots, x_n)$

Kysymys: onko olemassa $(y_1, \dots, y_n) \in \mathbf{Z}^n$ joilla $p(y_1, \dots, y_n) = 0$

Itse asiassa Matijasevitsh osoitti vielä vahvemmin että mille tahansa osittain ratkeavalle luonnollisten lukujen ongelmalle π on olemassa kokonaislukukertoiminen polynomi p_π jolle

$$\pi(x) = 1 \quad \text{jos ja vain jos} \quad \exists y_1, \dots, y_n : p_\pi(x, y_1, \dots, y_n) = 0.$$

Formaalien kielten ratkeamattomia ongelmia

Olkoot G_1 ja G_2 kontekstittomia kielioppeja ja R säännöllinen lauseke. Tällöin esim. seuraavat ongelmat ovat ratkeamattomia:

- onko G_1 moniselitteinen
- onko $L(G_1) \cap L(G_2) = \emptyset$
- onko $L(G_1) = L(G_2)$
- onko $L(G_1) = L(R)$
- onko $L(G_1) = T^*$ jollain aakkostolla T

Näiden ratkeamattomuustulosten todistukset sivuutetaan.

Todetaan kuitenkin että ne kaikki voidaan tehdä palutuksella [Postin vastaavuusongelmasta](#) joka seuraavaksi käydään läpi tarkemmin.

Postin vastaavuusongelma

(Post Correspondence Problem, E. Post 1946)

Edellä on todistettu ratkeamattomuustuloksia vain melko suoraan Turingin koneisiin liittyville ongelmille. Tarkastellaan nyt seuraavaa merkkijonojen järjestelemiseen liittyvää ongelmaa:

Annettu: Jono merkkijonopareja $((w_1, x_1), \dots, (w_k, x_k))$ missä $k \in \mathbf{N}$ on mielivaltainen ja $w_i, x_i \in \Sigma^*$

Kysymys: onko olemassa indeksijono (i_1, \dots, i_m) jolle

$$w_{i_1}w_{i_2} \dots w_{i_m} = x_{i_1}x_{i_2} \dots x_{i_m}$$

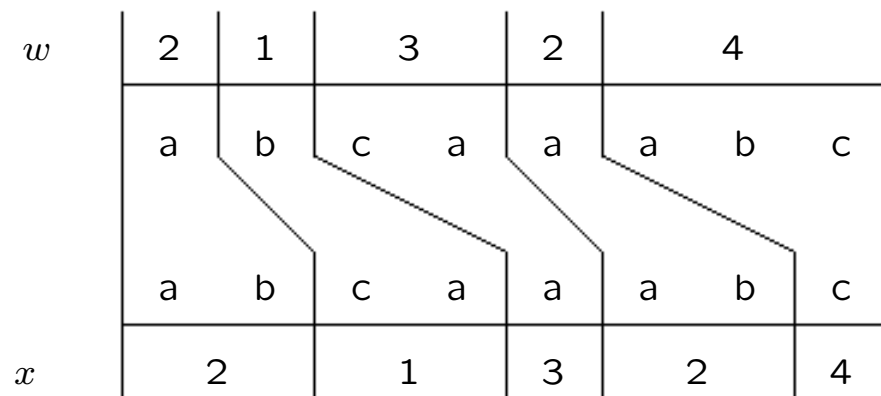
Esimerkki: Tapauksella

$$((w_1, x_1), (w_2, x_2), (w_3, x_3), (w_4, x_4)) = ((b, ca), (a, ab), (ca, a), (abc, c))$$

vastaus on **kyllä**, mikä nähdään indeksijonolla $(2, 1, 3, 2, 4)$. Nimittäin

$$w_2w_1w_3w_2w_4 = abcaaabc = x_2x_1x_3x_2x_4.$$

Edellisen kalvon esimerkkivastaavuus kuvallisesti:



Tekninen huomio: Jotta voisimme esittää tämän ongelman formaalina kielenä, meidän pitää sopia miten merkkijonopari, ja sellaisten muodostama jono, koodataan yhdeksi merkkijonoksi.

Eräs ilmeinen ratkaisu on laajentaa aakkostoa merkeillä "(", " , ", " ja ")". Monissa yhteyksissä on kuitenkin kätevää, jos pareja ja jonoja voidaan koodata aakkostoa laajentamatta.

Kun $w = x_1x_2 \dots x_n$ missä $x_i \in \Sigma$, määritellään $\hat{w} = x_1x_1x_2x_2 \dots x_nx_n$; siis kaikki merkit tuplataan.

Kun $|\Sigma| \geq 2$, määritellään jononmuodostusfunktio $\langle \cdot \rangle$ seuraavasti:

$$\langle w_1, w_2, w_3, \dots, w_{n-1}, w_n \rangle = \hat{w}_1ab\hat{w}_2ab\hat{w}_3ab \dots \hat{w}_{n-1}ab\hat{w}_n$$

missä a ja b ovat mitkä tahansa kaksi aakkoston Σ merkkiä.

Esim. $\langle 101, 01, 10 \rangle = 110011010011011100$ missä on valittu $a = 0, b = 1$.

Nyt $\langle \cdot \rangle$ on helppo laskea ja kuvaa kaikki aakkoston Σ merkkijonojen äärelliset jonot aakkoston Σ merkkijonoiksi. Kääntäen jos $v = \langle w_1, \dots, w_k \rangle$ on annettu, niin komponentit w_i ovat yksikäsitteiset ja helppo löytää.

Olkoon nyt $|\Sigma| \geq 2$. Määritellään formaali kieli PCP seuraavasti:

PCP koostuu merkkijonoista $\langle w_1, x_1, \dots, w_k, x_k \rangle$ joille on olemassa indeksijono i_1, \dots, i_m missä

$$w_{i_1} \dots w_{i_m} = x_{i_1} \dots x_{i_m}.$$

Seuraavassa tarvitaan myös modifioitua PCP-kieltä, jossa vaaditaan että sovituksen pitää alkaa parilla (w_1, x_1) :

MPCP koostuu merkkijonoista $\langle w_1, x_1, \dots, w_k, x_k \rangle$ joille on olemassa indeksijono i_2, \dots, i_m missä

$$w_1 w_{i_2} \dots w_{i_m} = x_1 x_{i_2} \dots x_{i_m}.$$

Seuraavassa todistetaan

Lause: Kieli MPCP ei ole rekursiivinen.

Tästä päästään alkuperäiseen ongelmaan PCP seuraavan aputuloksen nojalla:

Lemma: $\text{MPCP} \leq_m \text{PCP}$.

Todistus: Harjoitustehtävä. \square

Tuttujen periaatteiden mukaisesti nyt seuraa

Korollaari: Kieli PCP ei ole rekursiivinen.

Todistus: Jos PCP olisi rekursiivinen, niin edellisen lemmän seurauksena myös MPCP olisi, vastoin edellistä lausetta. \square

Kielen MPCP ratkeamattomuus osoitetaan konstruoimalla rekursiivinen palautus $f: L_u \leq_m \text{MPCP}$ universaalikielestä L_u . Käydään läpi palautuksen konstruktio pääpiirteissään.

Pitää siis määritellä f siten, että $f(v) \in \text{MPCP}$ jos ja vain jos $v \in L_u$. Mielenkiintoinen tapaus on se, jossa $v = w111x$ missä w on koodi jollekin Turingin koneelle M . Seuraavassa esitellään ne parit (w_i, x_i) joille $f(v) = \langle w_1, x_1, \dots, w_n, x_n \rangle$.

MPCP-tapauksen kieleksi tulee

$$\Gamma \cup Q \cup \{ \# \}$$

missä Γ on koneen M nauha-aakkosto, Q sen tilojen joukko ja $\#$ symboli joka ei kuulu kumpaankaan.

MPCP-tapauksen **osaratkaisu** on indeksijono (i_2, \dots, i_n) missä joko merkkijono $w_1 w_{i_2} \dots w_{i_n}$ on merkkijonon $x_1 x_{i_2} \dots x_{i_n}$ alkuosa, tai päinvastoin.

Siis osaratkaisuja ovat erityisesti ne indeksijonot jotka voidaan täydentää varsinaisiksi ratkaisuiksi. Osaratkaisulta ei kuitenkaan edellytetä että tällainen täydennys olisi olemassa.

Esimerkki: tyhjä indeksijono ($n = 1$) on aina osaratkaisu (edellyttäen että w_1 on merkkijonon x_1 alkuosa tai kääntäen).

Esimerkki: Tapauksella

$$((w_1, x_1), (w_2, x_2), (w_3, x_3), (w_4, x_4)) = ((a, ab), (b, ca), (ca, a), (abc, c))$$

indeksijono $(2, 3, 1)$ on osaratkaisu, sillä

$$\begin{aligned} w_1 w_2 w_3 w_1 &= abcaa \\ x_1 x_2 x_3 x_1 &= abcaaab \end{aligned}$$

Seuraavaksi ruvetaan konstruoida itse palautusta $L_u \leq_m$ MPCP.

Koneelle M ja syötteelle x konstruoidaan nyt MPCP-tapaus jonka osaratkaisut vastaavat koneen M laskentoja syötteellä x :

- olkoon

$$q_0x \vdash_M y_1q'_1z_1 \vdash_M \dots \vdash_M y_pq'_pz_p$$

missä q_0 on koneen M alkutila

- esitetään laskenta kielen $\Gamma \cup Q \cup \{ \# \}$ merkkijonoina

$$\#q_0x\#y_1q'_1z_1\#y_2q'_2z_2\#\dots\#y_pq'_pz_p\#$$

- nyt MPCP-tapaukselle on kaikilla $t = 0, 1, \dots, p - 1$ osaratkaisu (i_2, \dots, i_n) missä

$$w_1w_{i_2}\dots w_{i_n} = \#q_0x\#y_1q'_1z_1\#y_2q'_2z_2\#\dots\#y_tq'_tz_t\#$$

$$x_1x_{i_2}\dots x_{i_n} = \#q_0x\#y_1q'_1z_1\#y_2q'_2z_2\#\dots\#y_tq'_tz_t\#y_{t+1}q'_{t+1}z_{t+1}\#$$

MPCP-tapaukseen tulee kaikki seuraavanlaiset parit (w, x) , missä q on mielivaltainen tila, q_F hyväksyvä tila, X ja Y mielivaltaisia nauha-aakkosia ja B tyhjämerkki (siis $B \neq \#$)

- | | | |
|-----|---|---------------------------------------|
| (A) | $(\#, \#q_0x\#)$ | tulee alkupariksi (w_1, x_1) |
| (B) | (a, a) | kaikilla $a \in \Gamma \cup \{ \# \}$ |
| (C) | (qX, Yp) | jos $\delta(q, X) = (p, Y, R)$ |
| (C) | (ZqX, pZY) kaikilla $Z \in \Gamma$ | jos $\delta(q, X) = (p, Y, L)$ |
| (D) | $(q\#, Yp\#)$ | jos $\delta(q, B) = (p, Y, R)$ |
| (D) | $(Zq\#, pZY\#)$ kaikilla $Z \in \Gamma$ | jos $\delta(q, B) = (p, Y, L)$ |
| (E) | (Xq_F, q_F) ja $(q_F X, q_F)$ | kaikilla $q_F \in F, X \in \Gamma$ |
| (F) | $(q_F \#\#, \#)$ | kaikilla $q_F \in F$ |

Idea on että (A)-pari esittää laskennan alun ja (E)- ja (F)-tyyppiset parit lopun. (C)-parit esittävät laskennan siirtymiä normaalitilanteessa ja (D)-parit nauhan "reunoilla". (B)-parit kopioivat ne nauhan osat joita siirtymä ei muuta.

Tarkastellaan esimerkkinä Turingin konetta

$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_1\})$ missä
 $\delta(q_0, 0) = (q_2, 1, R)$, $\delta(q_2, 0) = (q_1, 0, R)$, $\delta(q_2, 1) = (q_3, 0, R)$ ja
 $\delta(q_3, B) = (q_2, 1, L)$; muita siirtymiä ei ole.

Kone M hyväksyy syötteen 01 seuraavasti:

$$q_0 0 1 \vdash 1 q_2 1 \vdash 1 0 q_3 \vdash 1 q_2 0 1 \vdash 1 0 q_1 1.$$

MPCP-tapaukseen tulee esitetyn laisia pareja seuraavasti:

- (A): $(\#, \#q_0 0 1 \#)$
- (B): $(0, 0), (1, 1), (B, B), (\#, \#)$
- (C): $(q_0 0, 1 q_2), (q_2 0, 0 q_1), (q_2 1, 0 q_3),$
 $(0 q_3 B, q_2 0 1), (1 q_3 B, q_2 1 1), (B q_3 B, q_2 B 1)$
- (D): $(0 q_3 \#, q_2 0 1 \#), (1 q_3 \#, q_2 1 1 \#), (B q_3 \#, q_2 B 1 \#)$
- (E): $(0 q_1, q_1), (1 q_1, q_1), (B q_1, q_1), (q_1 0, q_1), (q_1 1, q_1), (q_1 B, q_1)$
- (F): $(q_1 \# \#, \#)$

Katsotaan nyt miten MPCP-ratkaisu syntyy ylläolevaa laskentaa simuloiden.

Merkitään $W = w_1w_{i_2}\dots w_{i_k}$ ja $X = x_1x_{i_2}\dots x_{i_k}$ missä (i_2, \dots, i_k) on tähän mennessä kasattu osaratkaisu.

Aluksi pitää siis ottaa pari $(\#, \#q_001\#)$:

W: $\#$
X: $\#q_001\#$

Nyt W -merkkijonon perään pitää saada liitetyksi $q_001\#$, joten on valittava parit $(q_00, 1q_2)$, $(1, 1)$ ja $(\#, \#)$:

W: $\#q_001\#$
X: $\#q_001\#1q_21\#$

X -merkkijonon loppuun siis ilmestyi $1q_21\#$, mikä pitää saada W -merkkijonoon; pitää valita parit $(1, 1)$, $(q_21, 0q_3)$ ja $(\#, \#)$:

W: $\#q_001\#1q_21\#$
X: $\#q_001\#1q_21\#10q_3\#$

Nyt ollaan syötteen lopussa, ja parin $(1, 1)$ kanssa pari $(0q_3\#, q_201\#)$ sopii jonon jatkoksi:

W: $\#q_001\#1q_21\#10q_3\#$
X: $\#q_001\#1q_21\#10q_3\#1q_201\#$

Seuraavaksi pitää ottaa parit $(1, 1)$, $(q_20, 0q_1)$, $(1, 1)$ ja $(\#, \#)$:

W: $\#q_001\#1q_21\#10q_3\#1q_201\#$
X: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#$

On päästy lopputilaan q_1 , ja voidaan "syödä" yksi merkki valitsemalla esim. parit $(1, 1)$, $(0q_1, q_1)$, $(1, 1)$ ja $(\#, \#)$:

W: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#$
X: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#$

Jatketaan pareilla $(1q_1, q_1)$, $(1, 1)$ ja $(\#, \#)$

W: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#$

X: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#q_11\#$

ja sitten (q_11, q_1) ja $(\#, \#)$:

W: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#q_11\#$

X: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#q_11\#q_1\#$

Nyt ratkaisu tulee valmiiksi lisäämällä pari $(q_1\#\#, \#)$:

W: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#q_11\#q_1\#\#$

X: $\#q_001\#1q_21\#10q_3\#1q_201\#10q_11\#1q_11\#q_11\#q_1\#\#$

Yleisemmin

- jos M hyväksyy syötteen x , niin hyväksyvää laskentaa vastaa MPCP-tapauksen ratkaisu edelläesitettyyn tapaan
- on myös selvää että ainoa mahdollisuus saada osittainen MPCP-ratkaisu on laskennan simuloiminen, ja ainoa tapa saada ratkaisu valmiiksi on päätyä laskennassa hyväksyvään tilaan
- siis M hyväksyy merkkijonon x jos ja vain jos parista (M, x) konstruoidulla MPCP-tapauksella on ratkaisu
- koska MPCP-tapauksen konstruoiva funktio ilmeisesti on rekursiivinen, olemme osoittaneet $L_u \leq_m$ MPCP

Siis MPCP ei ole rekursiivinen.



Esimerkkinä kieliteoreettisista ongelmista todistetaan seuraava:

Lause Seuraava ongelma on ratkeamaton:

Annettu: kontekstittomat kieliopit G_1 ja G_2

Kysymys: onko $L(G_1) \cap L(G_2) \neq \emptyset$

Todistus: Osoitetaan, että mistä tahansa PCP-tapauksesta $(w_1, x_1), \dots, (w_k, x_k)$ voidaan muodostaa sellaiset kontekstittomat kieliopit G_1 ja G_2 , että $L(G_1) \cap L(G_2) \neq \emptyset$ jos ja vain jos kyseisellä PCP-tapauksella on ratkaisu. Koska PCP on ratkeamaton, tämä todistaa väitteen.

Kielioppien G_1 ja G_2 pääteakkostoksi tulee $\Sigma \cup \{a_1, \dots, a_k\}$, missä Σ on PCP-tapauksen aakkosto ($w_i, x_i \in \Sigma^*$), ja a_i missä $i = 1, \dots, k$ on uusi symboli ("nimi" parille (w_i, x_i)).

Määritellään kielet L_A ja L_B seuraavasti:

$$\begin{aligned} L_A &= \{ w_{i_1} w_{i_2} \dots w_{i_n} a_{i_1} a_{i_2} \dots a_{i_n} \mid i_j \in \{ 1, \dots, k \} \} \\ L_B &= \{ x_{i_1} x_{i_2} \dots x_{i_n} a_{i_1} a_{i_2} \dots a_{i_n} \mid i_j \in \{ 1, \dots, k \} \} \end{aligned}$$

- jos (i_1, \dots, i_n) on ratkaisu PCP-tapaukselle, niin $w_{i_1} w_{i_2} \dots w_{i_n} = x_{i_1} x_{i_2} \dots x_{i_n}$, joten $w_{i_1} w_{i_2} \dots w_{i_n} a_{i_1} a_{i_2} \dots a_{i_n} \in L_A \cap L_B$.
- jos $z \in L_A \cap L_B$, niin z on muotoa $v a_{i_1} a_{i_2} \dots a_{i_n}$ missä $v \in \Sigma^*$. Lisäksi

$$z \in L_A \Rightarrow v = w_{i_1} w_{i_2} \dots w_{i_n}$$

$$z \in L_B \Rightarrow v = x_{i_1} x_{i_2} \dots x_{i_n}$$

joten $w_{i_1} w_{i_2} \dots w_{i_n} = v = x_{i_1} x_{i_2} \dots x_{i_n}$ ja (i_1, \dots, i_n) on ratkaisu PCP-tapaukselle.

- siis PCP-tapauksella on ratkaisu jos ja vain jos $L_A \cap L_B \neq \emptyset$.

- pitää vielä osoittaa että $L_A = L(G_1)$ ja $L_B = L(G_2)$ kontekstittomilla kielioppeilla G_1 ja G_2 jotka voidaan laskea kun PCP-tapaus on annettu
- kieliopissa G_1 ainoa välike A
- kieliopissa G_1 yht. $2k$ produktiota $A \rightarrow w_i A a_i$ ja $A \rightarrow w_i a_i$, $i = 1, \dots, k$
- kieliopissa G_2 ainoa välike B
- kieliopissa G_2 yht. $2k$ produktiota $B \rightarrow x_i B a_i$ ja $B \rightarrow x_i a_i$, $i = 1, \dots, k$
- selvästi nämä kieliopit täyttävät halutut ehdot



Luokan RE rakenne

Seuraavan lauseen intuitiivinen tulkinta on, että universaalikieli L_u on "ainakin yhtä vaikea" kuin mikä tahansa rekursiivisesti lueteltava kieli.

Lause: Jos $A \in \text{RE}$ niin $A \leq_m L_u$.

Todistus: Olkoon $A \in \text{RE}$; siis $A = L(M)$ jollain M . Olkoon w koneen M koodi. Määritellään

$$f(x) = w111x \quad \text{kaikilla } x.$$

Selvästi f on rekursiivinen. Koska $x \in A$ jos ja vain jos $f(x) \in L_u$, funktio f on haluttu palautus $A \leq_m L_u$. □

Koska lisäksi L_u itse on rekursiivisesti lueteltava, sanomme että L_u on "maksimaalisen vaikea" rekursiivisesti lueteltavien kielten joukossa.

Seuraavaksi formalisoimme tämän käyttäen **RE-täydellisyyden** käsitettä.

Määritelmä Jos \mathcal{C} on kokoelma aakkoston Σ kieliä, sanomme että kieli $A \subseteq \Sigma^*$ on **\mathcal{C} -täydellinen** rekursiivisten palautusten suhteen jos

1. $A \in \mathcal{C}$ ja
2. $B \leq_m A$ kaikilla $B \in \mathcal{C}$.

Yleensä on selvää millaisista palautuksista puhutaan, jolloin sanotaan yksinkertaisesti että A on \mathcal{C} -täydellinen.

Siis \mathcal{C} -täydelliset ongelmat ovat luokan \mathcal{C} maksimaalisen vaikeita ongelmia.

Esimerkki: L_u on RE-täydellinen.

Palautuksilla on transitiivisuusominaisuus:

Lause: Jos $A \leq_m B$ ja $B \leq_m C$ niin $A \leq_m C$.

Todistus: HT. □

Korollaari: Jos $A \in \mathcal{C}$ ja $B \leq_m A$ jollain \mathcal{C} -täydellisellä B , niin A on \mathcal{C} -täydellinen.

Todistus: Jos $C \in \mathcal{C}$, niin $C \leq_m B$ (koska B on \mathcal{C} -täydellinen) joten $C \leq_m A$ (transitiivisuus). □

Esimerkki: Kun todistimme että epätyhjyysongelma L_{ne} ei ole rekursiivinen, teimme sen muodostamalla palautuksen $L_u \leq_m L_{ne}$. Koska lisäksi $L_{ne} \in \text{RE}$, niin L_{ne} on RE-täydellinen.

Lisää esimerkkejä: Ricen lauseen todistuksessa osoitettiin, että $L_u \leq_m \text{codes}(\mathcal{S})$ millä tahansa semanttisella ongelmalla \mathcal{S} . Siis jos $\text{codes}(\mathcal{S})$ ylipäänsä on rekursiivisesti lueteltava, se on RE-täydellinen.

Toisaalta jos A on rekursiivinen, niin $A \leq_m B$ mille tahansa B kunhan $B \neq \emptyset$ ja $B \neq \Sigma^*$. Siis rekursiiviset kielet ovat "maksimaalisen helppoja".

Yleisesti kaikista "luonnollisista" rekursiivisesti lueteltavista kielistä osoittautuu, että ne ovat joko rekursiivisia tai RE-täydellisiä. On kuitenkin mahdollista konstruoida esimerkkejä RE-kielistä, joiden vaikeus on näiden kahden ääripään välillä:

Lause Luokassa RE – REC on kieliä jotka eivät ole RE-täydellisiä.

Vastaavasti voidaan puhua luokan

$$\text{co-RE} = \{ \bar{A} \mid A \in \text{RE} \}$$

täydellisistä kielistä.

Lause: Kieli A on RE-täydellinen jos ja vain jos \bar{A} on co-RE-täydellinen.

Todistus: Määritelmän mukaan $A \in \text{RE}$ jos ja vain jos $\bar{A} \in \text{co-RE}$.

Jos f on palautus $B \leq_m A$, niin se on samalla palautus $\bar{B} \leq_m \bar{A}$. □

Kielet A ja B ovat rekursiivisesti isomorfiset jos $f: A \leq_m B$ jollain bijektiolla $f: \Sigma^* \rightarrow \Sigma^*$.

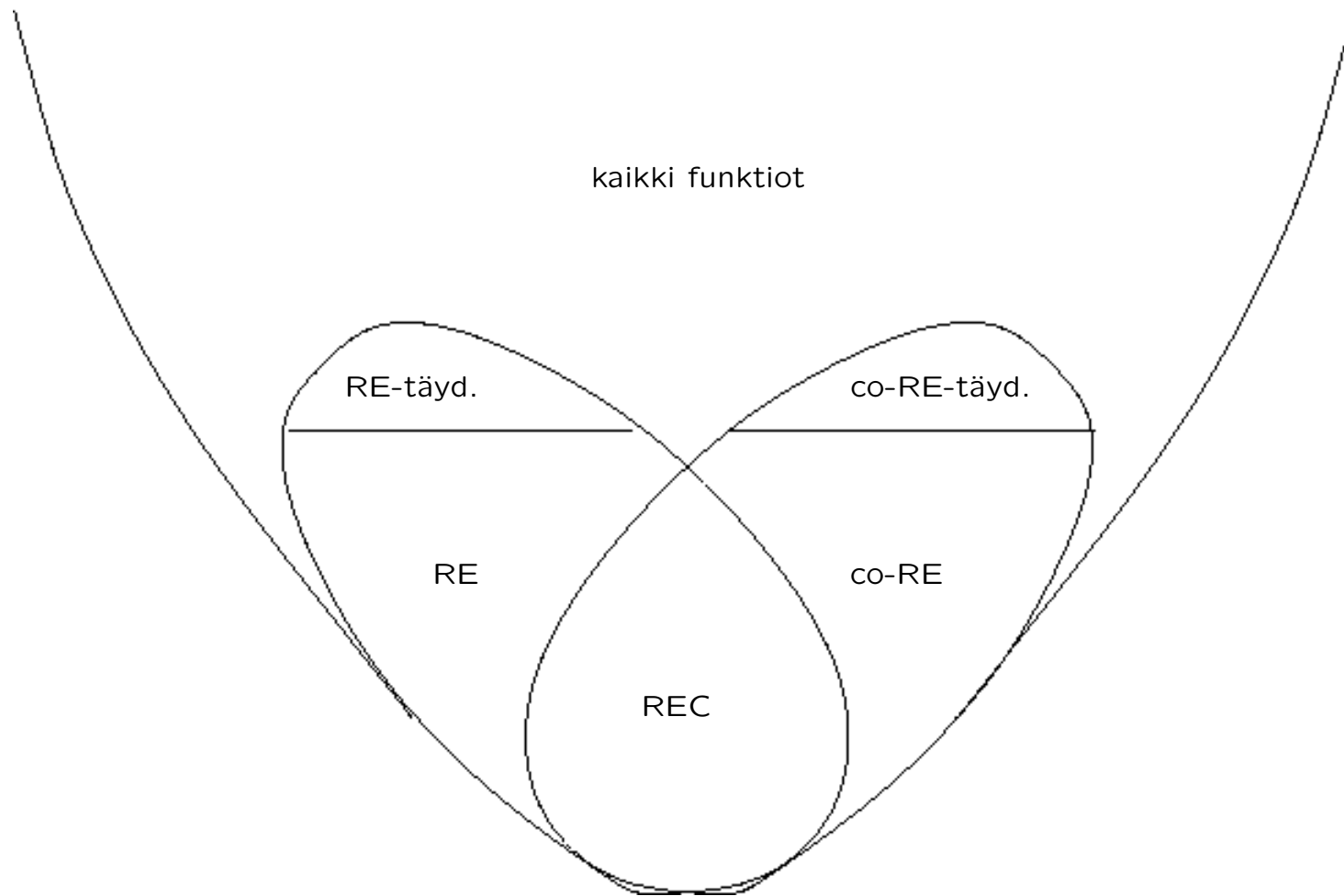
Lause: Kaikki RE-täydelliset kielet ovat keskenään rekursiivisesti isomorfisia.

Siis RE-täydelliset kielet ovat kaikki "samoja" modulo merkkijonojen rekursiivinen uudelleenjärjestäminen.

Laskennallisten ongelmien vaativuusluokat jatkuvat (pitkälle) RE- ja co-RE-täydellisten kielten ulkopuolelle. Esim. totaalisuusongelma

$$L_{tot} = \{ w \in \{0, 1\}^n \mid M_w \text{ pysähtyy kaikilla syötteillä} \}$$

jää luokan $RE \cup co-RE$ ulkopuolelle.



Funktioiden laskettavuusluokkia kuvannollisesti

Yhteenveto laskettavuudesta

- monet tärkeät ongelmat ovat ratkeamattomia

kieli	REC?	RE?	co-RE?
universaalikieli L_u	ei	on	ei
pysähtymisongelma H	ei	on	ei
epätyhjiysongelma L_{ne}	ei	on	ei
tyhjiysongelma L_e	ei	ei	on
totaalisuusongelma L_{tot}	ei	ei	ei

- ratkeamattomat ongelmat matematiikassa ja formaaleissa kielissä
- universaalien laskennan mallien ekvivalessin perusteella tulokset siirtyvät koskemaan RAM-mallia, ohjelmointikieliä, ...
- jos hyväksymme Churchin-Turingin teesin, tämä kertoo siitä mitä missään realistisessa mallissa on mahdollista laskea
- **Perustekniikka:** Osoitetaan $A \notin \text{REC}$ todistamalla $B \leq_m A$ jollain tunnetulla $B \notin \text{REC}$; vastaavasti $A \notin \text{RE}$

3. Laskennan vaativuusteoriaa

- tähän asti puhuttu siitä, mitä on mahdollista laskea ”äärellisessä ajassa”
- siirrytään tarkastelemaan laskemista ”kohtuullisessa ajassa”
- vaihtoehtoisesti voidaan laskenta-ajan sijaan tarkastella tarvittavan työmuistin määrää
- yleisesti ottaen teoreettiset tarkastelut muuttuvat teknisesti vaativammiksi
- tulosten intuitiivisen tulkinnan kanssa syytä olla varovainen (mikä annetussa sovelluksessa on ”kohtuullista”; kuinka yleinen ongelma oikeasti halutaan ratkaista jne.)

Tarkastellaan standardimallisen (yksinauhaisen, deterministisen) Turingin koneen M syötteellä x alkutilanteesta q_0x alkavaa laskentaa

$$q_0x \vdash_M v_1q_1w_1 \vdash_M \dots \vdash_M v_nq_nw_n \vdash_M \dots$$

- laskenta **pysähtyy** jos jollain k tilanteesta $v_kq_kw_k$ ei ole siirtymää; tällöin laskennan **pituus** on k mitä merkitään

$$\text{time}_M(x) = k.$$

- jos laskenta ei pysähdy, merkitään $\text{time}_M(x) = \infty$
- vastaavasti määritellään

$$\text{space}_M(x) = \max \{ |v_k| + |w_k| \mid k = 1, 2, 3, \dots \}$$

- $\text{time}_M: \Sigma^* \rightarrow \mathbf{N} \cup \{ \infty \}$ ja $\text{space}_M: \Sigma^* \rightarrow \mathbf{N} \cup \{ \infty \}$ ovat koneen M **aika-** ja **tilavaativuusfunktio**

Intuitiivinen vastaavuus tietokoneohjelmiin

aikavaativuus: kuinka monta konekielikäskyä ajon aikana suoritetaan

tilavaativuus: kuinka monta tavua muistia ajon aikana varataan

Liian yksityiskohtaisia mittareita ollakseen hyödyllisiä:

- vaikea saada selville tarkasti
- vaikka tunnettaisiin tarkasti, lausekkeet tyypillisesti liian monimutkaisia ollakseen valaisevia
- saman ongelman ratkaisujen M ja N vertaileminen vaikeaa, koska yleensä aina joillain x, y on $\text{time}_M(x) < \text{time}_N(x)$ ja $\text{time}_M(y) > \text{time}_N(y)$

Tämän takia

- tarkastellaan **pahimman tapauksen** aika- ja tila vaativuutta
- tarkastellaan vaativuusfunktioita **kertaluokan** tarkkuudella

Pahimman tapauksen aika- ja tilavaativuus

Kun $n \in \mathbb{N}$, määritellään

$$\begin{aligned}\text{time}_M(n) &= \max \{ \text{time}_M(x) \mid |x| = n \} \\ \text{space}_M(n) &= \max \{ \text{space}_M(x) \mid |x| = n \}\end{aligned}$$

- yleensä kuvaa kohtuullisesti toimintaa "realistisilla" syötteillä (varmasti paremmin kuin "parhaan tapauksen" vaativuus)

Voidaan tarkastella myös keskimääräistä tapausta, esim.

$$\text{time}_M^{\text{avg}}(n) = \sum_{x \in \Sigma^n} p_n(x) \text{time}_M(x)$$

missä p_n , $n \in \mathbb{N}$, on jokin todennäköisyysjakauma joukossa Σ^n .

- yleensä vaikea analysoida edes helpoilla p_n
- helpot p_n usein epärealistisia
- usein vaikea edes tietää millainen p_n olisi realistinen

Funktioden kertaluokat

Olkoon $f, g: \mathbf{N} \rightarrow \mathbf{R}_+$ funktioita.

- f on **kertaluokkaa** $O(g)$, merk. $f = O(g)$, jos joillain $c > 0$, $m \in \mathbf{N}$ pätee

$$f(n) \leq cg(n) \quad \text{aina kun } n \geq m$$

- f on **samaa kertaluokkaa** kuin g , merk. $f = \Theta(g)$, jos joillain $a, b > 0$, $m \in \mathbf{N}$ pätee

$$ag(n) \leq f(n) \leq bg(n) \quad \text{aina kun } n \geq m$$

- f on **alempaa kertaluokkaa** kuin g , merk. $f = o(g)$, jos kaikilla $c > 0$ on olemassa $m \in \mathbf{N}$ jolle

$$f(n) < cg(n) \quad \text{aina kun } n \geq m$$

- g on **alaraja** funktiolle f , merk. $f = \Omega(g)$, jos jollain $c > 0$ pätee

$$f(n) \geq cg(n) \quad \text{äärettömän monella } n$$

Motivaatio: miksi vaativuusfunktioiden kertaluokkia

- rajataan tarkastelusta pois pienillä syötteillä mahdollisesti esiintyvät epäsäännöllisyydet
- ei välitetä vakiokertoimista ($cf(n)$ samaa suuruusluokkaa kuin $f(n)$ kaikilla $c > 0$) koska ne kuitenkin täysin riippuvat mallin yksityiskohdista (koneen "käskykanta")

Yleensä käytetään epätäsmällisiä ilmauksia, kuten esim. " $f(n) = O(n^2)$ " tai " f on kertaluokkaa n^2 " merkityksessä " $f = O(g)$ missä $g(n) = n^2$ kaikilla n "

Kertaluokkien perusominaisuuksia

- $\log_a(n) = \Theta(\log_b(n))$ kaikilla $a, b > 1$ (joten kertaluokista puhuttaessa merkitään usein vain $\log n$ missä kantaluovun valinta on yhdentekevä)
- $n^a = o(n^b)$ jos $a < b$
- $\log n = o(n^a)$ kaikilla $a > 0$
- $n^a = o(2^{bn})$ kaikilla $a, b > 0$
- $cf(n) = \Theta(f(n))$ kaikilla $c > 0$
- $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$
- jos p on asteen k polynomi niin $p(n) = \Theta(n^k)$

(Todistus: HT; oleellisesti kurssilla Diff. int. I.1 tms.)

Vaativuusluokat

Kieli A voidaan tunnistaa ajassa t (tilassa s) jos $A = L(M)$ missä $\text{time}_M(n) \leq t(n)$ (vast. $\text{space}_M(n) \leq s(n)$) kaikilla n .

Funktioiden t ja s määräämät (deterministiset) aika- ja tilavaativuusluokat ovat

$$\begin{aligned} \text{DTIME}(t) &= \{ A \subseteq \Sigma^* \mid A \text{ voidaan tunnistaa ajassa } t \} \\ \text{DSPACE}(s) &= \{ A \subseteq \Sigma^* \mid A \text{ voidaan tunnistaa tilassa } s \} \end{aligned}$$

- yksinkertaisuuden vuoksi sallitaan moninauhainen M ; (aika- ja tilavaativuus määritellään moninauhaiselle koneelle ilmeisellä tavalla)
- kuitenkin oleellista että M deterministinen (mistä kirjain "D")
- epädeterministisiin vaativuusluokkiin palataan kohta

Tärkeitä vaativuusluokkia

Määritellään polynominen ja eksponentiaalinen aika- ja tilavaativuusluokka:

$$\begin{aligned}P &= \bigcup \{ \text{DTIME}(t) \mid t \text{ polynomi} \} \\ \text{PSPACE} &= \bigcup \{ \text{DSPACE}(s) \mid s \text{ polynomi} \} \\ E &= \bigcup \{ \text{DTIME}(2^{n^k}) \mid k = 0, 1, 2, \dots \} \\ \text{ESPACE} &= \bigcup \{ \text{DSPACE}(2^{n^k}) \mid k = 0, 1, 2, \dots \}\end{aligned}$$

Siis erityisesti

$$P = \{ L(M) \mid \text{time}_M(n) = O(n^k) \text{ jollain } k \}.$$

Tila- ja aikavaativuus eri laskennan malleissa

- jos $A = L(M)$ moninauhaisella Turingin koneella M , niin $A = L(N)$ missä N on yksinauhainen, $\text{time}_N(n) = O(\text{time}_M(n)^2)$ ja $\text{space}_N(n) = O(\text{space}_M(n))$ (vrt. aiemmin esitetty simulaatio)
- jos kieli voidaan tunnistaa RAM-mallissa (tai jossain muussa "nykyaikaisen tietokoneen" abstraktissa mallissa) ajassa t ja tilassa s , niin se voidaan tunnistaa Turingin koneella ajassa $O(t(n)^2)$ ja tilassa $O(s(n))$.
- yleisemmin aiemmin esitetyissä universaaleissa laskennan malleissa aika- ja tilavaativuudet (kun ne määritellään järkevästi) ovat **polynomisessa suhteessa**
- ts. jos jokin ongelma ratkeaa jossain mallissa ajassa $t(n)$ niin missä tahansa toisessa mallissa se ratkeaa ajassa $O(t(n)^k)$ jollain k
- siis erityisesti luokat P, E, PSPACE ja ESPACE ovat riippumattomia valitusta laskennan mallista

Usein ajatellaan, että polynomisessa ajassa ratkeavat ongelmat ovat ”käytännössä ratkeavia”. Mahdollista kritiikkiä:

- Jos vaativuus on esim. luokkaa n^{100} niin ongelma ei varmaan ole käytännössä ratkeava.
Kommentti: harvoja poikkeuksia lukuunottamatta ei tunneta käytännön esimerkkejä joissa aikavaativuus olisi polynominen mutta korkeaa astetta
- Käytännössä syötteen koolla on jokin äärellinen yläraja, joten ei-polynomisuus ei ole ongelma
Kommentti: näin voi olla jos yläraja on riittävän pieni; tyypilliset ei-polynomiset aikavaativuusfunktiot kuitenkin kasvavat nopeasti jo melko pienillä n .
- Pahin tapaus ei kerro koko totuutta.
Kommentti: voi hyvin pitää paikkansa, mutta esim. keskimääräisen tapauksen analyysi vaikeaa ja usein edellyttää kyseenalaisia oletuksia.

Johtopäätös: yleensä ”polynomisuus = tehokkuus” mutta mallin rajoituksia pitää harkita tapauskohtaisesti

Esimerkkejä polynomisista ja ei-polynomisista ongelmista

Ennen yleisempiä teoriatarkasteluja katsotaan joitain tyypillisiä esimerkkejä ongelmista ja niiden vaativuudesta

- kaikki nämä ongelmat ratkeavia (nähdään helposti)
- pääkysymys: mitkä niistä ratkeavat polynomisessa ajassa
- osoittautuu, että tämä on yllättävän vaikea kysymys

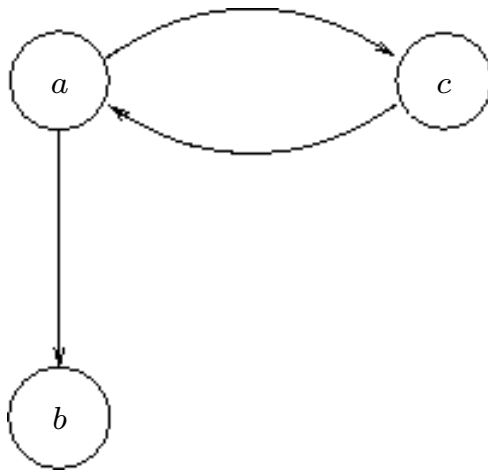
Kuten ratkeavuustarkasteluissa, meidän pitää tehdä joitain teknisiä esitysmuoto-oletuksia.

- luokat P, PSPACE jne. määritelty luokkina formaaleja kieliä
- ts. formaalisti ongelmien syötteen ovat merkkijonoja
- käytännön ongelmassa syötteen ovat verkkoja, loogisia kaavoja, joukkoja jne.

⇒ täytyy sopia syötteiden esittämisestä merkkijonoina

Esimerkki: suunnattu verkko

Kurssilla Tietorakenteet opittuun tapaan suunnattu verkko $G = (V, E)$ voidaan esittää esim. vierusmatriisina tai vieruslistana.



Vierusmatriisi

A	1	2	3
1	0	1	1
2	0	0	0
3	1	0	0

Vieruslista

$((2, 3), (), (1))$

Alilista i sisältää ne j joilla $(v_i, v_j) \in E$

$a = v_1, b = v_2, c = v_3$
 $A(i, j) = 1$ joss $(v_i, v_j) \in E$

$V = \{a, b, c\}$

$E = \{(a, b), (a, c), (c, a)\}$

Vierusmatriisi tai vieruslista voidaan esittää merkkijonona sillä tavalla kuin tällaisia rakenteita koodataan esim. ASCII-jonoiksi normaalissa ohjelmoinnissa.

- kun sanomme että jokin esim. verkkoja koskeva päätösongelma ratkeaa polynomisessa ajassa, tarkoitamme että vastaava merkkijonoesityksistä koostuva formaali kieli kuuluu luokkaan P
 - jatkossa ei kiinnitetä huomiota esitystavan yksityiskohtiin, vaan oletetaan se tehdyksi jollain "järkevällä" tavalla
 - lopputulos on, että muodollinen määritelmä on sama kuin esim. kurssilta Tietorakenteet tuttu polynominen aikavaativuus
 - pitää kuitenkin muistaa, että aikavaativuus määritellään syötteen pituuden suhteen
- ⇒ "järkevä" pitää sisällään sen, että syötteen koko ei kasva liikaa
- **Huom.** tärkeä erikoistapaus: luonnollisen luvun n koodin pituus on $O(\log n)$ (esim. binääriesitys $\lfloor \log_2 n \rfloor + 1$ bittiä)

Esimerkkejä verkko-ongelmista

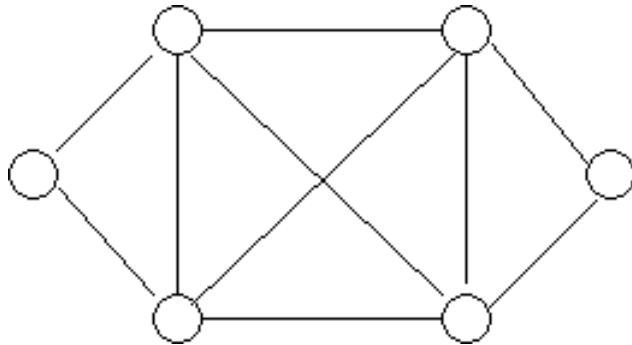
- vierusmatriisiesityksen koko on $O(n^2)$, vieruslistaesityksen koko $O(n + m) = O(n^2)$ missä $n = |V|$ ja $m = |E|$ (huom. $m = O(n^2)$)
- ⇒ verkon tapauksessa oletetaan, että syötteen pituus on polynominen verkon solmujen lukumäärän n suhteen
- siis ongelmaa vastaavan formaalin kielen kuuluminen luokkaan P tarkoittaa samaa kuin että ongelma ratkeaa solmujen lukumäärän n suhteen polynomisessa ajassa

Kurssilla Tietorakenteet opitun perustella esim. seuraavat ongelmat ratkeavat polynomisessa ajassa:

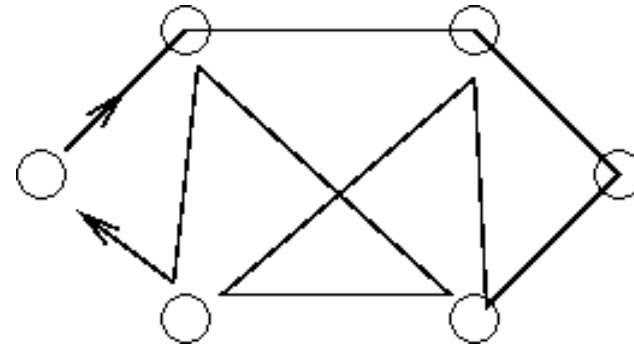
- onko suunnatussa (tai suuntaamattomassa) verkossa G polku solmusta u solmuun v (syvyysuuntainen etsintä $O(n + m)$)
- onko painotetussa verkossa G solmusta u solmuun v polku, jonka kustannus on korkeintaan c (Dijkstran algoritmi $O(n \log n + m)$)

Huom. nämä on tässä muotoiltu päätösongelmina; palaamme pian kysymykseen etsintä- ja optimointiongelmistä

Eulerin kehä



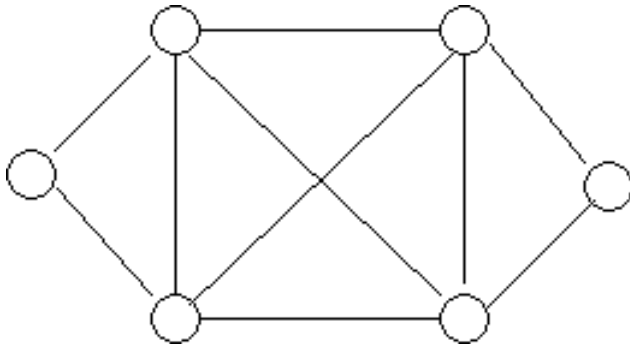
Verkko



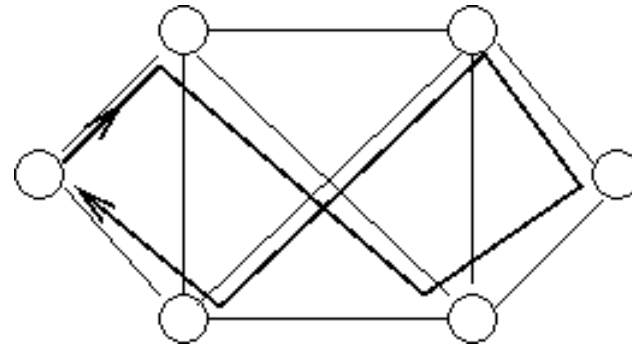
Eräs Eulerin kehä

- Eulerin kehä on polku, joka käyttää verkon jokaista kaarta tasan kerran ja palaa lähtöpisteeseensä
- Kysymys "Onko annetussa suuntaamattomassa verkossa Eulerin kehä?" ratkeaa polynomisessa ajassa
- perustuu havaintoon, että Eulerin kehä on olemassa, jos ja vain jos verkko on yhtenäinen ja jokaisen solmun aste on parillinen (Euler 1736)

Hamiltonin kehä (Hamiltonian Circuit, HC)



Verkko



Eräs Hamiltonin kehä

- Hamiltonin kehä on polku, joka käy jokaisessa solmussa tasan kerran ja palaa lähtöpisteeseensä
- Ongelmalle ”Onko annetussa suuntaamattomassa verkossa Hamiltonin kehä?” ei tunneta polynomisessa ajassa toimivaa algoritmia
- Pidetään luultavana, ettei tällaista algoritmia ole olemassa, eli että $HC \notin P$

Hamiltonin kehä -ongelmalle on seuraava naiivi ratkaisualgoritmi:

Olkoon $G = (V, E)$ missä $V = \{v_1, \dots, v_n\}$.

Toista kaikilla jonon $(v_1, v_2, v_3, \dots, v_n)$ permutaatioilla $(u_1, u_2, u_3, \dots, u_n)$:

Jos $(u_j, u_{j+1}) \in E$ kun $j = 1, \dots, n - 1$ ja $(u_n, u_1) \in E$

niin palauta "kyllä"

Palauta "ei".

- toimii selvästi polynomisessa tilassa; siis $HC \in PSPACE$
- tarkistettavia permutaatioita $n!$ kappaletta, yhden tarkistus selvästi polynomisessa ajassa
- $(n/2)^{n/2} \leq n! \leq n^n$
- Siis $HC \in E$, mutta tämän algoritmin aikavaativuus ei polynominen
- HC on NP-täydellinen ongelma, mistä lisää pian

Päätösongelmaan HC liittyy myös ilmeinen etsintäongelma:

Syöte: suuntaamaton verkko $G = (V, E)$

Tuloste: jokin verkon G Hamiltonin kehä $H \subseteq E$ jos olemassa; "ei ole" muuten

- ol. funktio $HC(G)$ ratkaisee Hamiltonin kehä -ongelman, ts. palauttaa tosi joss verkossa G on Hamiltonin kehä
- seuraava algoritmi ratkaisee vastaavan etsintäongelman:

```
/* Olkoon  $E = \{e_1, \dots, e_m\}$ . */  
 $H := E$ ;  
for  $i := 1$  to  $m$  do  
    if  $HC(V, H - \{e_i\})$  then  $H := H - \{e_i\}$ ;  
return  $H$ .
```

- erityisesti jos $HC \in P$ niin etsintäongelma ratkeaa polynomisessa ajassa
- tosin luultavasti $HC \notin P$ jolloin etsintäongelmakaan ei tietysti ratkea polynomisessa ajassa

Alkulukutestaus

- luonnollinen luku $n \geq 2$ on **alkuluku** jos luvun n ainoat tekijät ovat 1 ja n
- $\text{PRIMES} = \{ n \mid n \text{ alkuluku} \}$
- ongelmalle PRIMES on äskettäin löydetty polynomisessa ajassa toimiva ratkaisualgoritmi ("PRIMES is in P", M. Agrawal, N. Kayal ja N. Saxena, 2002)
- aikavaativuuspolynomi on korkeaa astetta, käytännössä aiemmat satunnaisuuteen perustuvat algoritmit saattavat olla parempia
- huom. ongelma on testata luku n sen **pituuden** $O(\log n)$ suhteen polynomisessa ajassa; itse luvun n suhteen polynominen aika saavutetaan triviaalisti (mikä ei auta jos n on esim. 1000-bittinen)
- tässä tapauksessa etsintäongelma näyttäisi olevan vaikeampi kuin päätösongelma: edelleenkin ei osata tehokkaasti jakaa lukua alkutekijöihin
- tehokas algoritmi tekijöihin jaolle murtaisi monet salausmenetelmät

Yleistetty tammipeli

Annettu: mielivaltainen pelitilanne $n \times n$ tammilaudalla

Kysymys: voiko musta pelaaja pakottaa voiton itselleen

Ongelma ratkeaa eksponentiaalisessa ajassa ja polynomisessa tilassa tietyillä tasapelisäännöillä; ratkeavuus polynomisesta ajassa avoin (veikkaus: ei)

Yleistetty go-peli

Annettu: mielivaltainen pelitilanne $n \times n$ go-laudalla

Kysymys: voiko musta pelaaja pakottaa voiton itselleen

Ongelma ratkeaa eksponentiaalisessa mutta ei polynomisessa ajassa; ratkeavuus polynomisessa tilassa avoin

Epädeterministiset vaativuusluokat

Äskeisten esimerkkien jälkeen palataan nyt teorian kehittelyyn.

- halutaan määritellä epädeterministisen Turingin koneen aika- ja tilavaativuus
- epädeterministinen aikavaativuus hyödyllinen käsite ongelmien analysoimisessa; ei realistinen mitta ongelman ratkaisujalle ”oikeilla” tietokoneilla
- yksittäisen laskennan aika- ja tilavaativuus voidaan määritellä kuten deterministisessä tapauksessa
- epädeterministisellä koneella N voi syötteellä x olla useita laskentoja, joten määritellään

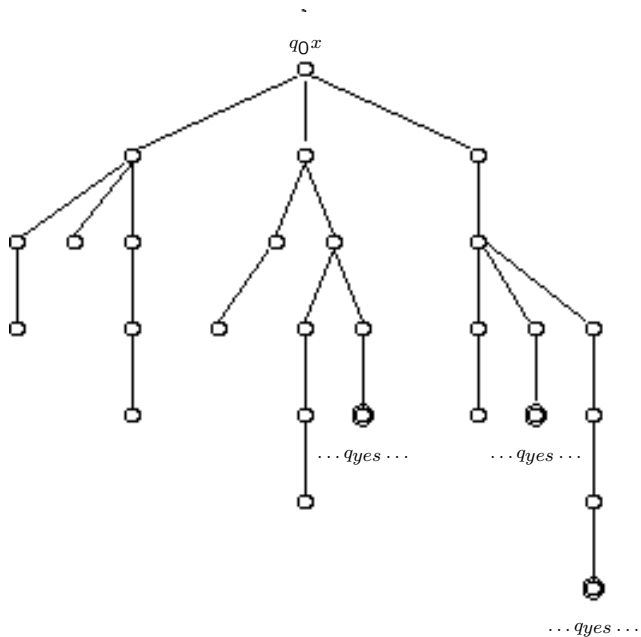
$$\begin{aligned} \text{time}_N(x) &= \text{pisimmän koneen } N \text{ laskennan pituus syötteellä } x \\ \text{space}_N(x) &= \text{eniten tilaa vievän koneen } N \text{ laskennan} \\ &\quad \text{tilavaativuus syötteellä } x \end{aligned}$$

- pahimman tapauksen aikavaativuus $\text{time}_N(n)$ jne. kuten ennen
- "pisin laskenta" ja "eniten tilaa" voivat vaikuttaa oudoilta valinnoilta, mutta tällä ei itse asiassa ole suurta merkitystä:

Lause Jos $A = L(N)$ missä N on yksinauhainen epädeterministinen Turingin kone ja jokaisella $x \in A$ koneen N lyhimmän hyväksyvän laskennan pituus on $O(|x|^k)$, niin $A = L(N')$ missä N' on yksinauhainen epädeterministinen Turingin kone ja $\text{time}_{N'}(x) = O(|x|^{2k})$.

Todistushahmotelma Muodostetaan ensin kaksinauhainen N'' , joka aluksi kirjoittaa kakkosnauhalle $O(|x|^k)$ nollaa ja sen jälkeen simuloi konetta N käyttäen kakkosnauhaa käskylaskurina (vrt. harj. 4. teht. 5.) Tämä kaksinauhainen kone toimii ajassa $O(n^k)$, joten sitä voidaan simuloida yksinauhaisella ajassa $O(n^{2k})$. □

Epädeterministisen Turingin koneen N laskentaa syötteellä x on usein hyödyllistä ajatella **laskentapuuna**



- solmuina laskennan mahdolliset tilanteet
- juurena alkutilanne
- lehtinä tilanteet joista ei siirtymää, erityisesti hyväksyvät tilanteet
- solmun vqw jälkeläisiä ne $v'q'w'$ joilla $vqw \vdash_N v'q'w'$
- N hyväksyy syötteen x joss puussa ainakin yksi hyväksyvä lehti

- jos $\text{time}_N(x) = t$, laskentapuussa kork. d^t solmua missä d suurin määrä seuraajia millään yksittäisellä tilanteella
- triviaali yläraja $d \leq 2|\Gamma||Q|$

Edellinen lause: jos lyhimmän hyväksyvän polun pituudelle tunnetaan yläraja, laskennasta voidaan karsia kaikki tätä pitemmät haarat

- tunnistettu kieli pysyy samana
- karsinnan toteuttamisesta tulee korkeintaan neliöllinen lisäaikaraste

Joitain epädeterministisiä vaativuusluokkia

Kieli A voidaan tunnistaa epädeterministisesti ajassa t (tilassa s) jos $A = L(N)$ epädet. Turingin koneella N jolla $\text{time}_N(n) \leq t(n)$ (vast. $\text{space}_N(n) \leq s(n)$) kaikilla n .

Analogisesti deterministisen tapauksen kanssa määritellään

$$\begin{aligned}\text{NTIME}(t) &= \{ A \subseteq \Sigma^* \mid A \text{ voidaan tunnistaa epädet. ajassa } t \} \\ \text{NSPACE}(s) &= \{ A \subseteq \Sigma^* \mid A \text{ voidaan tunnistaa epädet. tilassa } s \}\end{aligned}$$

ja

$$\begin{aligned}\text{NP} &= \bigcup \{ \text{NTIME}(t) \mid t \text{ polynomi} \} \\ \text{NPSPACE} &= \bigcup \{ \text{NSPACE}(s) \mid s \text{ polynomi} \} \\ \text{NE} &= \bigcup \{ \text{NTIME}(2^{n^k}) \mid k = 0, 1, 2, \dots \} \\ \text{NESPACE} &= \bigcup \{ \text{NSPACE}(2^{n^k}) \mid k = 0, 1, 2, \dots \}\end{aligned}$$

Aiemmin esitettyyn tapaan myös epädet. koneilla moninauhaista konetta N voidaan simuloida yksinauhaisella ajassa $O(\text{time}_N(n)^2)$ ja tilassa $O(\text{space}_N(n))$, joten näiden luokkien kannalta on sama käytetäänkö moni- vai yksinauhaisia koneita.

Osoitetaan esimerkkinä että $HC \in NP$, eli muodostetaan polynomisessa ajassa toimiva epädeterministinen Turingin kone N em. Hamiltonin kehä-ongelmalle.

Olkoon syöte $G = (V, E)$, missä $V = \{v_1, \dots, v_n\}$ ja $E = \{(v_{i_1}, v_{j_1}), \dots, (v_{i_m}, v_{j_m})\}$.

Konkreettisuuden vuoksi oletetaan tämä esitetyksi aakkoston $\{0, 1, -\}$ merkkijonona

$$\hat{n} - -\hat{i}_1 - \hat{j}_1 - -\hat{i}_2 - \hat{j}_2 - - \dots - -\hat{i}_m - \hat{j}_m - -$$

missä \hat{i} on luvut i binääriesitys. Tämä esitys on helppo muodostaa vieruslistasta tai -matriisista.

Esim. jos $V = \{a, b, c, d\}$ ja $E = \{(a, b), (a, c), (b, c), (c, d)\}$ niin koodataan $a \mapsto 1$, $b \mapsto 10$, $c \mapsto 11$ ja $d \mapsto 100$ ja esitykseksi tulee

$$100 - -1 - 10 - -1 - 11 - -10 - 11 - -11 - 100 - -$$

Kone N on kaksinauhainen ja toimii seuraavasti:

1. Kirjoita kakkosnauhalle epädeterministisesti jokin merkkijono $v \in \{0, 1, -\}^*$.
2. Olkoon $v = -w_1 - w_2 - \dots - w_n-$ missä $w_i \in \{0, 1\}^*$ ja $n = |V|$; jos v ei ole tätä muotoa niin hylkää.
3. Jos jollain $i = 1, \dots, n$ koodi $-\hat{i}-$ esiintyy merkkijonossa v kaksi kertaa tai ei esiinny lainkaan, hylkää.
4. Jos jollain $i = 1, \dots, n - 1$ syöte ei sisällä merkkijonoa $- - w_i - w_{i+1} - -$ eikä $- - w_{i+1} - w_i - -$ niin hylkää (huom. suuntaamaton verkko, jompi kumpi suunta riittää)
5. Jos syöte ei sisällä merkkijonoa $- - w_1 - w_n - -$ eikä $- - w_n - w_1 - -$, niin hylkää.
6. Muuten hyväksy.

Selvästi N toimii polynomisessa ajassa ja tunnistaa kielen HC, joten $HC \in NP$.

Jatkossa esitämme epädeterministiset Turingin koneet vähemmän yksityiskohtaisena pseudokoodina.

Esim. Hamiltonin kehä -ongelman epädeterministinen ratkaisu polynomisessa ajassa:

1. Lue $G = (V, E)$; olkoon $n = |V|$ ja $V = \{v_1, \dots, v_n\}$.
2. Arvaa epädeterministisesti jono $(i_1, \dots, i_n) \in \{1, \dots, n\}^n$.
3. Jos
 - jokainen luku $j = 1, \dots, n$ esiintyy jonossa (i_1, \dots, i_n) tasan kerran ja
 - kaikilla $j = 1, \dots, n$ pätee $(v_{i_j}, v_{i_{j+1}}) \in E$ tai $(v_{i_{j+1}}, v_{i_j}) \in E$ ja
 - $(v_{i_1}, v_{i_n}) \in E$ tai $(v_{i_n}, v_{i_1}) \in E$niin hyväksy.
4. Muuten hylkää.

Käytännössä tämäkin on turhan yksityiskohtaista, kohta 3 voidaan vain sanoa ” Jos solmujono $(v_{i_1}, v_{i_2}, \dots, v_{i_n})$ on Hamiltonin kehä, niin hyväksy.”

Propositiologiikan toteutuvuusongelma (Satisfiability, SAT)

Otetaan vielä esimerkkinä yksi tärkeä luokan NP ongelma.

Propositiologiikan kaava muodostuu loogisista konnektiiveista \vee , \wedge ja \neg , sulkumerkeistä (ja), vakiosymboleista 0 ja 1 sekä muuttujasymboleista x_i , $i = 1, 2, 3, \dots$, tuttuun tapaan; esim. $x_4 \vee \neg(x_7 \wedge x_1)$.

Kaavat voidaan ilmeisellä tavalla esittää aakkoston $\{\vee, \wedge, \neg, (,), 0, 1, x\}$ merkkijonoina; esim. $x100 \vee \neg(x111 \wedge x1)$.

Kaavan totuus määritellään tuttuun tapaan; 0 = epätosi ja 1 = tosi.

Jos $\phi(x_1, \dots, x_n)$ on muuttujia x_1, \dots, x_n sisältävä kaava ja $(v_1, \dots, v_n) \in \{0, 1\}^n$, niin $\phi(v_1, \dots, v_n)$ on kaavan arvo kun muuttujille sijoitetaan arvot $x_i := v_i$; esim. kun $\phi(x_1, x_2, x_3) = (x_3 \wedge \neg x_1) \vee \neg x_2$ niin $\phi(0, 1, 0) = (0 \wedge \neg 0) \vee \neg 1 = 0$.

Kaava $\phi(x_1, \dots, x_n)$ on **toteutuva** jos $\phi(v_1, \dots, v_n) = 1$ jollain $(v_1, \dots, v_n) \in \{0, 1\}^n$.

Määritellään nyt propositiologiikan toteutuvuusongelma

$$\text{SAT} = \{ \phi \mid \phi \text{ on toteutuva} \}.$$

Toteutuvuusongelma voidaan helposti ratkaista epädeterministisesti polynomisessa ajassa:

1. Lue $\phi(x_1, \dots, x_n)$.
2. Arvaa epädeterministisesti $(v_1, \dots, v_n) \in \{0, 1\}^n$.
3. Jos $\phi(v_1, \dots, v_n) = 1$ niin hyväksy; muuten hylkää.

Siis $\text{SAT} \in \text{NP}$.

Luokan NP ongelmien yleinen luonne

Edelliset esimerkit luokan NP ongelmista ratkesivat epädeterministisesti tyyliin

1. Lue syöte x .
2. Arvaa y , jonka pituus riippuu syötteen x koosta.
3. Jos $R(x, y)$ niin hyväksy; muuten hylkää.

Tässä $R(x, y)$ on jokin ominaisuus joka voidaan ratkaista deterministisesti polynomisessa ajassa. Siis kaikki epädeterminismi on alun arvausvaiheessa.

Itse asiassa muunlaista epädeterminismiä ei tarvita luokan NP kielten tunnistamiseen.

Lause: Kieli A kuuluu luokkaan NP jos ja vain jos on olemassa polynomi q ja $R \in P$ joille

$$x \in A \Leftrightarrow \exists y \in \Sigma^*: |y| \leq q(n) \wedge \langle x, y \rangle \in R.$$

Todistus: Helppo, mutta sivuutetaan. □

Tulkinta: Olkoon $A \in NP$ ja merkinnät kuten yllä.

- jos $x \in A$, on olemassa polynomisen kokoinen y joka on **todiste** tälle asianhaaralla
- on olemassa polynomisessa ajassa toimiva **tarkistaja** (nim. kielen R hyväksyvä kone), joka annetulla syötteellä x ja väitetyllä todisteella y tarkistaa, onko todiste kelvallinen
- esim. SAT-ongelman tapauksessa kaavaa $\phi(x_1, \dots, x_n)$ vastaava todiste on mikä tahansa $(v_1, \dots, v_n) \in \{0, 1\}^n$ jolla $\phi(v_1, \dots, v_n) = 1$; tarkistaja sijoittaa annetut arvot annettuun kaavaan ja tarkistaa, tuliko tosi
- ongelman " $x \in A$?" tekee vaikeaksi se, että potentiaalisia todistajia on eksponentiaalinen määrä

P = NP -ongelma

- laskettavuusteoria: mikä voidaan tunnistaa epädeterministisellä Turingin koneella, voidaan tunnistaa myös deterministisellä Turingin koneella
- vaativuusteoria: koskeeko sama tunnistamista **polynomisessa ajassa**; ts. päteekö $P = NP$?
- Kysymys formuloitu 1971 (Cook), **yhä avoin!**
- intuitiivisesti $P = NP$ olisi hyvin yllättävä tulos, koska em. tulkinnan mukaan se tarkoittaisi, että todistaja voidaan löytää eksponentiaalisesta etsintäavaruudesta polynomisessa ajassa

$P = NP$ pätee jos ja vain jos jokin **NP-täydellinen** ongelma (esim. HC, SAT) ratkeaa polynomisessa ajassa; siirrytään tarkastelemaan näitä.

Polynomiset palautukset ja NP-täydellisyys

Polynomisen palautuksen idea on sama kuin rekursiivisen palautuksen, paitsi että liikutaan polynomisen aikavaativuuden maailmassa.

Funktio f voidaan laskea polynomisessa ajassa, jos jokin Turingin kone M laskee sen ja $\text{time}_M(n) = O(n^k)$ jollain k .

Funktio $f: \Sigma^* \rightarrow \Gamma^*$ palauttaa polynomisesti kielen $A \subseteq \Sigma^*$ kieleen $B \subseteq \Gamma^*$, merkitään $f: A \leq_m^p B$, jos f on laskettavissa polynomisessa ajassa ja

$$x \in A \Leftrightarrow f(x) \in B \quad \text{kaikilla } x \in \Sigma^*.$$

Jos tällainen f on olemassa, sanotaan että kieli A palautuu kieleen B ja merkitään $A \leq_m^p B$.

Lemma: Olkoon \mathcal{S} jokin vaativuusluokista P, NP, PSPACE, E, NE. Jos $A \leq_m^p B$ ja $B \in \mathcal{S}$, niin $A \in \mathcal{S}$.

Todistus: Helppo. □

Olkoon $\mathcal{C} \subseteq \mathcal{P}(\Sigma^*)$ luokka aakkoston Σ kieliä.

Kieli $A \subseteq \Sigma^*$ on **\mathcal{C} -täydellinen** polynomisten palautusten suhteen jos

1. $A \in \mathcal{C}$ ja
2. $B \leq_m^p A$ kaikilla $B \in \mathcal{C}$

Kieli $A \subseteq \Sigma^*$ on **\mathcal{C} -kova** polynomisten palautusten suhteen jos $B \leq_m^p A$ kaikilla $B \in \mathcal{C}$.

Siis \mathcal{C} -kova ongelma on **ainakin yhtä vaikea** kuin mikä tahansa luokan \mathcal{C} ongelma. Luokan \mathcal{C} **maksimaalisen vaikeat** ongelmat ovat \mathcal{C} -täydellisiä.

Kiinnostavia luokkia \mathcal{C} ovat tässä erityisesti NP, mutta myös PSPACE, E, ESPACE jne.

Kaikilla vaativuusluokilla \mathcal{C} **ei** ole täydellisiä ongelmia (mutta kaikilla tähän mennessä esitellyillä itse asiassa on).

Yleensä puhutaan vain "NP-täydellisyydestä" jne. ja yhteydestä selviää että puhutaan polynomisista palautuksista (**mutta** muunkinlaisia palautuksia esiintyy vaativuusteoriassa).

Täydellisten joukkojen perusominaisuuksia

Lemma: Jos $A \leq_m^p B$ ja $B \leq_m^p C$ niin $A \leq_m^p C$.

Todistus: HT. \square

Korollaari Jos $A \leq_m^p B$ ja A on \mathcal{C} -kova niin B on \mathcal{C} -kova.

Erityisesti B osoitetaan NP-kovaksi palauttamalla jokin tunnettu NP-kova A siihen. Jos lisäksi $B \in \text{NP}$, niin B on tällöin NP-täydellinen.

Korollaari Jos A on \mathcal{C} -kova ja $A \in \text{P}$, niin $\mathcal{C} \subseteq \text{P}$.

Erityisesti jos jollakin NP-täydellisellä ongelmalla on polynomisessa ajassa toimiva ratkaisu, niin $\text{P} = \text{NP}$ (mitä siis ei pidetä luultavana).

Esimerkkejä NP-täydellisistä ongelmista

Toteutuvuus (Satisfiability, SAT)

Annettu: propositiologiikan kaava ϕ

Kysymys: onko ϕ toteutuva

Hamiltonin kehä (Hamiltonian Circuit, HC)

Annettu: suuntaamaton verkko G

Kysymys: onko verkossa G Hamiltonin kehä

Kauppamatkustajan ongelma (Travelling Salesman Problem, TSP)

Annettu: suuntaamaton verkko G , painot $c(u, v) \in \mathbb{N}$ kaikille kaarille $(u, v) \in E$, luonnollinen luku k (Siis tarkemmin syöte on sopivasti koodattu kolmikko $\langle G, c, k \rangle$.)

Kysymys: onko verkossa G Hamiltonin kehä jonka kokonaispaino on kork. k

Lisää NP-täydellisiä ongelmia

Solmupeite (Vertex Cover, VC)

Annettu: suuntaamaton verkko $G = (V, E)$, luonnollinen luku k

Kysymys: onko verkossa G kork. k solmua sisältävä **solmupeite**, ts. joukko $U \subseteq V$ joka kaikista kaarista $(u, v) \in E$ sisältää ainakin toisen päätepisteistä u ja v .

Riippumaton joukko (Independent Set, IS)

Annettu: suuntaamaton verkko $G = (V, E)$, luonnollinen luku k

Kysymys: onko verkossa G **riippumaton joukko** jossa väh. k solmua, ts. sellainen $U \subseteq V$ että $(u, v) \notin E$ kaikilla $u, v \in U$

Klikki (Clique, CLIQUE)

Annettu: suuntaamaton verkko $G = (V, E)$, luonnollinen luku k

Kysymys: onko verkossa G **klikki** jossa väh. k solmua, ts. sellainen $U \subseteq V$ että $(u, v) \in E$ kaikilla $u, v \in U$

Vielä lisää NP-täydellisiä ongelmia

Verkonväritys (Graph Coloring, GC)

Annettu: suuntaamaton verkko G , luonnollinen luku k

Kysymys: voidaanko verkon G solmut värittää k värillä niin että minkään kaaren päätepisteet eivät ole saman väriset

Ositus (Partition, PARTITION)

Annettu: jono (a_1, \dots, a_n) luonnollisia lukuja

Kysymys: onko olemassa $A \subseteq \{1, \dots, n\}$ jolla $\sum_{i \in A} a_i = \sum_{i \notin A} a_i$

Repunpakkaus (Knapsack, KNAPSACK)

Annettu: äärellinen joukko U ("esineet") ja jokaiselle $u \in U$ "arvo" $v(u) \in \mathbf{N}$ ja "paino" $w(u) \in \mathbf{N}$; kokonaisluvut V ja W

Kysymys: voidaanko valita sellainen joukko $X \subseteq U$ esineitä, että $\sum_{x \in X} w(x) \leq W$ ja $\sum_{x \in X} v(x) \geq V$

Toiminta kun kohdataan NP-täydellinen ongelma

1. Havaitaan että käsillä oleva ongelma A "näyttää NP-täydelliseltä"
2. Etsitään kirjallisuudesta "samantyyppinen" NP-täydelliseksi tunnettu ongelma B ; peruslähde Garey & Johnson: Computers and Intractability (1979)
3. Osoitetaan $B \leq_m^p A$
4. **Mietitään mitä sitten tehdään**

Huom. yleensä tällaisissa tilanteissa varsinainen ongelma on jotain NP-täydellistä päätösongelmaa vastaava **optimointiongelma**, esim.

opt-TSP: annettu painotettu verkko, määrättävä pienin Hamiltonin kehän paino

Myös tämän tyyppisiä ongelmia kutsutaan usein NP-koviksi.

Miten käytännössä ratkaistaan NP-kova ongelma

- Luultavasti ei kannata etsiä algoritmia joka aina toimii polynomisessa ajassa ja antaa tarkalleen oikean ratkaisun.
- Ehkä voidaan tehdä helpottavia lisäoletuksia siitä, millaisia syötteitä todella esiintyy.
- Jos tapaukset ovat riittävän pieniä, ehkä huolellisesti viritetty ei-polynominen algoritmi on riittävän nopea (branch-and-bound).
- Ehkä on olemassa tehokas approksimointialgoritmi joka todistettavasti löytää melkein optimaalisen ratkaisun.
- Ehkä joku heuristinen algoritmi, mahdollisesti yhdistettynä satunnaisuuteen, antaa riittävän hyviä ratkaisuja (lokaali haku, simuloitu jäähditys).

Esimerkkejä NP-täydellisyystodistuksista

Oletetaan tunnetuksi että IS on NP-täydellinen (tämä tullaan osoittamaan jatkossa)

Lause: VC on NP-täydellinen.

Todistus: Helposti nähdään $VC \in NP$ (harj.teht.). Osoitetaan $IS \leq_m^p VC$.

Olkoon $G = (V, E)$ missä $|V| = n$, ja $U \subseteq V$. Nyt

U on riippumaton joukko \Leftrightarrow millään $(u, v) \in E$ ei päde $u \in U$ ja $v \in U$
 \Leftrightarrow kaikilla $(u, v) \in E$ pätee $u \in V - U$ tai $v \in V - U$
 $\Leftrightarrow V - U$ on solmupeite.

Siis verkossa G on kokoa k oleva riippumaton joukko jos ja vain jos siinä on kokoa $n - k$ oleva solmupeite.

Olkoon f funktio joka kuvaa parin $\langle G, k \rangle$ pariaksi $f(\langle G, k \rangle) = \langle G, n - k \rangle$. Siis $\langle G, k \rangle \in IS$ joss $f(\langle G, k \rangle) \in VC$. Selvästi f on laskettavissa pol. ajassa, joten $f: IS \leq_m^p VC$. \square

Toinen esimerkki: Oletetaan tunnetuksi, että HC on NP-täydellinen (kuten jatkossa todistetaan).

Lause: TSP on NP-täydellinen.

Todistus: Helposti nähdään että $\text{TSP} \in \text{NP}$. Osoitetaan $\text{HC} \leq_m^p \text{TSP}$.

Olkoon $G = (V, E)$ HC-tapaus. Muodostetaan TSP-tapaus $f(G) = \langle G, c, n \rangle$ missä $c(u, v) = 1$ kaikilla $(u, v) \in E$, ja $n = |V|$. Siis verkko ennallaan ja kaikkien kaarten paino 1.

Selvästi verkossa G on Hamiltonin kehä joss painotetussa verkossa $\langle G, c \rangle$ on Hamiltonin kehä jonka paino on korkeintaan n . Siis $G \in \text{HC}$ jos ja vain jos $f(G) \in \text{TSP}$.

Koska $f: G \mapsto \langle G, c, n \rangle$ voidaan laskea polynomisessa ajassa, $\text{HC} \leq_m^p \text{TSP}$. \square

Mutta että tämä todella olisi validia, meidän pitää ensin osoittaa IS ja HC NP-täydellisiksi jollain muulla tavalla!

Toteutuvuusongelman NP-täydellisyys

Että päästään osoittamaan ongelmia NP-täydellisiksi palautuksilla, pitää ensin osoittaa jokin ongelma NP-täydelliseksi suoraan määritelmästä.

- valitaan täksi perusongelmaksi toteutuvuusongelma SAT
- tiedetään jo että $\text{SAT} \in \text{NP}$
- olkoon $A \in \text{NP}$; olkoon $A = L(M)$ missä $\text{time}_M(n) = p(n)$
- muodostetaan kaava $\phi_{M,w}$ s.e. $\phi_{M,w}$ on toteutuva jos ja vain jos koneella M on hyväksyvä laskenta syötteellä w
- kaavaan $\phi_{M,w}$ tulee $O(p(|w|))^2$ muuttujaa; intuitiiviset tulkinnat tyyppiä " $y_{ijc} = 1$ jos hetkellä i nauhapaikassa j on merkki c "

Lause Toteutuvuusongelma SAT on NP-täydellinen.

Todistus Tiedetään jo, että $\text{SAT} \in \text{NP}$. Pitää vielä osoittaa $A \leq_m^p \text{SAT}$ kaikilla $A \in \text{NP}$.

Olkoon $A = L(M)$ missä $\text{time}_M(n) \leq p(n)$, p polynomi ja M yksinauhainen. Muodostetaan palautus $f_M: A \leq_m^p \text{SAT}$; siis

- f_M laskettavissa polynomisessa ajassa ja
- $w \in A$ jos ja vain jos $f_M(w) \in \text{SAT}$

$f_M(w)$ on propositiologiikan kaava joka kuvaa mahdollisia koneen M laskentoja syötteellä w

- muuttujat ovat tyyppiä "hetkellä t kone on tilassa q ja nauhapään paikka on j " tai "hetkellä t nauhapositiossa j on merkki c "
- kaava $f_M(w)$ kuvaa, mitkä ajanhetkeä $t + 1$ vastaavat muuttujien arvot ovat sallittuja kun ajanhetken t arvot tunnetaan

Määritellään ensin apumuuttujat $X_{t,j}$, arvojoukkona $Q \cup \Gamma$, joiden avulla varsinaisten totuusarvomuttujien y_{tjz} merkitys voidaan selittää.

- numeroidaan nauhapositiot; aluksi nauhapää positiossa 1
- nauhapää voi enimmillään liikkua $p(n)$ kertaa vasemmalle tai $p(n)$ kertaa oikealle; siis nauhapää pysyy positioiden $-p(n)$ ja $p(n) + 1$ välillä
- aluksi syöte positioissa $1, \dots, n$

Määritellään muuttujat $X_{t,j} \in Q \cup \Gamma$, $j = -p(n) - 1, \dots, p(n) + 1$, kuvaamaan laskentaa hetkellä t :

- ol. koneen tilanne aqb , nauhapään sijainti m
- siis kone tilassa q , nauhapään vas. puolella $a = a_1 \dots a_k \in \Gamma^*$ ja nauhapäästä oikealle $b = b_1 \dots b_l \in \Gamma^*$
- koodataan tila ja nauhapään sijainti: $X_{t,m-1} = q$
- koodataan nauhan vasen osa: $X_{t,m-k-2+j} = a_j$, $j = 1, \dots, k$
- koodataan nauhan oikea osa: $X_{t,m+j-1} = b_j$, $j = 1, \dots, l$

Esimerkki Oletetaan q_1 alkutila, q_2 hyväksyvä tila, $\Sigma = \{a, b, c\}$,
 $\Gamma = \Sigma \cup \{ \# \}$ missä $\#$ on tyhjämerkki, $w = aab$ ja siis $n = 3$, $p(3) = 4$.

Olkoon koneessa (ainakin) siirtymät $\delta(q_0, a) = (q_3b, L)$, $\delta(q_3, \#) = (q_3, c, R)$,
 $\delta(q_3, b) = (q_3, b, R)$ ja $\delta(q_3, a) = (q_2, c, R)$. Syötteellä aab on siis olemassa
 hyväksyvä laskenta

$$q_1aab \vdash_M q_3\#bab \vdash_M cq_3bab \vdash_M cbq_3ab \vdash_M cbcq_2b$$

Vastaavat muuttujien $X_{t,j}$ arvot, $t = 0, \dots, 4$, $j = -5, \dots, 5$:

$X_{t,j}$	-5	-4	-3	-2	-1	0	1	2	3	4	5
0	#	#	#	#	#	q_0	a	a	b	#	#
1	#	#	#	#	q_3	#	b	a	b	#	#
2	#	#	#	#	c	q_3	b	a	b	#	#
3	#	#	#	#	c	b	q_3	a	b	#	#
4	#	#	#	#	c	b	c	q_2	b	#	#

Oleellinen havainto: arvo $X_{t,j}$ määräytyy arvoista $X_{t-1,j-1}$, $X_{t-1,j}$ ja $X_{t-1,j+1}$.

Varsinaisiksi totuusarvomuuttujiksi muodostettavaan propositiologiikan kaavaan $f_M(w)$ tulee nyt

$$y_{tjz}, \quad t = 0, \dots, p(n), \quad j = -p(n) - 1, \dots, p(n) + 1, \quad z \in Q \cup \Gamma,$$

siis kaikkiaan $(p(n) + 1)(2p(n) + 3)(|Q| + |\Gamma|)$ muuttujaa, joiden intuitiivinen tulkinta on

$$y_{tjz} = 1 \quad \text{joss} \quad X_{tj} = z.$$

Nyt $f_M(w)$ tulee olemaan muuttujia y_{tjz} sisältävä kaava, joka saa arvon 1 tasan sellaisilla muuttujien y_{tjz} arvoilla jotka esittävät koneen M hyväksyvää laskentaa syötteellä w .

Korkealla tasolla kaava tulee olemaan muotoa $A \wedge B \wedge C \wedge D$ missä osakaavoilla A , B , C ja D on seuraavat tulkinnat:

A : jokaisella ajanhetkellä koneen tila, nauhapään paikka ja jokaisen nauhaposition sisältö on yksikäsitteisesti määritelty;

B : rivi $X_{0,*}$ esittää laskennan alkutilannetta

C : kukin rivi $X_{t-1,*}$ seuraa riviä $X_{t,*}$ koneen M siirtymäfunktion mukaisesti

D : jollain j pätee $X_{p(n),j} = q$ missä q on hyväksyvä tila.

Kaava A: tilanteen yksikäsitteisyys

Merkitään indeksijoukkoja $J = \{-p(n) - 1, \dots, p(n) + 1\}$, $T = \{0, \dots, p(n)\}$ ja $Z = Q \cup \Gamma$.

Kaava $A_{t,j,z}$, $t \in T$, $j \in J$, $z \in Z$, on $|Q| + |\Gamma|$ osan konjunktio joka sanoo että $X_{t,j}$ saa arvon z eikä mitään muuta arvoa:

$$A_{t,j,z} = y_{t,j,z} \wedge \left(\bigwedge_{r \in Q \cup \Gamma, r \neq z} \neg y_{t,j,r} \right).$$

Kaava A'_t sanoo, että $X_{t,j}$ saa kaikilla j jonkin yksikäsitteisen arvon:

$$A'_t = \bigwedge_{j \in J} \bigvee_{z \in Z} A_{t,j,z}.$$

Kaavan A'_t koko on $(2p(n) + 3)(|Q| + |\Gamma|)^2$ kun yksinkertaisuuden vuoksi lasketaan muuttujasymbolin kooksi 1 eikä lasketa loogisia konnektiiveja.

Kaava $A_{t,j,Q}$ sanoo että $X_{t,j}$ on tilasymboli:

$$A_{t,j,Q} = \bigvee_{z \in Q} A_{t,j,z}.$$

Kaavan $A_{t,j,Q}$ koko on siis $|Q|(|Q| + |\Gamma|)$. Kaava A_t'' sanoo, että $X_{t,j}$ on tilasymboli tasan yhdellä j :

$$A_t'' = \bigvee_{j \in J} \left(A_{t,j,Q} \wedge \bigwedge_{j' \in J, j' \neq j} \neg A_{t,j',Q} \right);$$

kaavan A_t'' koko on $(2p(n) + 3)^2 |Q|(|Q| + |\Gamma|)$. Lopullinen kaava A on

$$A = \bigwedge_{t \in T} (A_t' \wedge A_t''),$$

koko $(p(n) + 1)((2p(n) + 3)(|Q| + |\Gamma|)^2 + (2p(n) + 3)^2 |Q|(|Q| + |\Gamma|)) = O(p(n)^3)$.

Kaava B: alkutilanne

Oikeaa alkutilannetta vastaava rivi $X_{0,*}$ koodataan kaavaksi

$$B = \left(\bigwedge_{j=-p(n)-1, \dots, -1} y_{0,j,\#} \right) \wedge y_{0,0,q_0} \wedge \left(\bigwedge_{j=1, \dots, n} y_{0,j,w_j} \right) \wedge \left(\bigwedge_{j=n+1, \dots, p(n)+1} y_{0,j,\#} \right)$$

missä $\#$ on tyhjämerkki ja syöte on $w = w_1 \dots w_n$; kaavan koko $O(p(n))$.

Kaava D: hyväksyminen

Oletetaan kone M muunnetuksi siten, että jokaisesta hyväksyvästä tilasta on siirtymä itseensä kaikilla merkeillä. Tämä selvästi ei muuta sitä, onko annetulla syötteellä kork. $p(n)$ askelen mittaisia hyväksyviä laskentoja.

Riittää siis tarkastaa, että hetkellä $p(n)$ ollaan hyväksyvässä tilassa:

$$D = \bigvee_{q \in F} \bigvee_{j \in J} y_{p(n),j,q}$$

missä F on hyväksyvien tilojen joukko; kaavan koko $O(p(n))$

Kaava C: laskennan oikea eteneminen; todistuksen ydin on tässä

Huomataan, että koko laskennan, eli $(p(n) + 1) \times (2p(n) + 3)$ -taulukon $(X_{t,j})$, oikeellisuuden tarkastamiseksi riittää tarkastaa kaikki (vakiokokoiset!) 2×3 -laatikot

$X_{t-1,j-1}$	$X_{t-1,j}$	$X_{t-1,j+1}$
$X_{t,j-1}$	$X_{t,j}$	$X_{t,j+1}$

- jos $X_{t-1,j-1}$, $X_{t-1,j}$ ja $X_{t-1,j+1}$ ovat kaikki nauhasymboleita, on oltava $X_{t,j} = X_{t-1,j}$ (nauhan sisältö voi muuttua vain nauhapään kohdalla)
- jos $X_{t-1,j}$ on tilasymboli, niin nauhapään alla on merkki $X_{t-1,j+1}$ ja siirtymäfunktion arvo $\delta(X_{t-1,j}, X_{t-1,j+1})$ antaa uudet arvot $X_{t,j-1}$, $X_{t,j}$ ja $X_{t,j+1}$

Esitetään ensin eri mahdollisuudet kaavioina. Seuraavassa A, B, C jne. esittävät nauhasymboleja ja q, q' tilasymboleja.

Nauhapää muualla: laatikon keskikohta ei muutu; reunoista ei voi sanoa koska nauhapää saattaa olla juuri laatikon vieressä

A	B	C
$X_{t,j-1}$	B	$X_{t,j+1}$

Nauhapää liikkuu oikealle: tässä $\delta(q, B) = (q', C, R)$

A	q	B
A	C	q'

Nauhapää liikkuu vasemmalle: tässä $\delta(q, B) = (q', C, L)$

A	q	B
q'	A	C

Reunat on helpoin käsitellä erikoistapauksina: $X_{t,-p(n)-1}$ ja $X_{t,p(n)+1}$ ovat aina tyhjämerkkejä, koska nauhapää ei voi ehtiä niin kauas.

Reunat hetkellä t käsitellään kaavalla

$$\widehat{C}_t = y_{t,-p(n)-1,\#} \wedge y_{t,p(n)+1,\#}.$$

Esitetään nyt em. kolme päätapausta kaavoina $C_{t,j}$, $C'_{t,j}$ ja $C''_{t,j}$.

Nauhapää muualla: Tämä on helpoin tapaus. Kaava $C_{t,j}$ sanoo, että $X_{t-1,j-1}$ ja $X_{t-1,j+1}$ ovat nauhasymboleja, ja $X_{t-1,j}$ ja $X_{t,j}$ ovat sama nauhasymboli:

$$C_{t,j} = \left(\bigvee_{z \in \Gamma} y_{t-1,j-1,z} \right) \wedge \left(\bigvee_{z \in \Gamma} y_{t-1,j+1,z} \right) \wedge \left(\bigvee_{z \in \Gamma} (y_{t-1,j,z} \wedge y_{t,j,z}) \right)$$

Nauhapää siirtyy oikealle: Kaava $C'_{t,j}$ sanoo, että hetkellä t nauhapään sijainti nauhalla on $j + 1$. Lisäksi, kun $q = X_{t-1,j}$ on koneen tila ja $B = X_{t-1,j+1}$ nauhapään alla oleva merkki, on valittu jokin sallittu siirtymä $(q', C, R) \in \delta(q, B)$. Merkki A nauhapään vasemmalla puolella pysyy ennallaan:

$$C'_{t,j} = \bigvee_{q \in Q} \bigvee_{A, B \in \Gamma} \left(y_{t-1,j-1,A} \wedge y_{t,j-1,A} \wedge y_{t-1,j,q} \wedge y_{t-1,j+1,B} \right. \\ \left. \wedge \left(\bigvee_{(q',C,R) \in \delta(q,B)} (y_{t,j,C} \wedge y_{t,j+1,q'}) \right) \right)$$

Nauhapää siirtyy vasemmalle: Kaava $C''_{t,j}$ muodostetaan analogisesti kaavan $C'_{t,j}$ kanssa.

Laskennan oikean etenemisen esittää nyt kaava

$$C = \bigwedge_{t=1, \dots, p(n)} \left(\hat{C}_t \wedge \left(\bigwedge_{j=-p(n), \dots, p(n)} (C_{t,j} \vee C'_{t,j} \vee C''_{t,j}) \right) \right).$$

Koska kukin $C'_{t,j}$ jne. on jotain (koneesta M riippuvaa) vakiokokoa, kaavan C koko on $O(p(n)^2)$.

Edellä esitetystä seuraa, että konstruoitu kaava $f_M(w) = A \wedge B \wedge C \wedge D$ toteutuu joillain muuttujien $y_{t,j,z}$ arvoilla jos ja vain jos koneella M on syötteellä w hyväksyvä laskenta jonka pituus on kork. $p(|w|)$.

Lisäksi f_M voidaan selvästi laskea polynomisessa ajassa. (Huomaa, että M on kiinteä; laskettaessa $f_M(w)$ ainoat syötteestä w riippuvat asiat ovat kaava A eli alkutilanne ja $p(n)$ eli kaavojen pituus.)

Siis $f_M: A \leq_m^p \text{SAT}$



SAT-ongelman rajoitetut muodot

- olemme juuri osoittaneet että SAT on NP-täydellinen
- perusidea on nyt osoittaa joukolle kiinnostavia ongelmia $A \in \text{NP}$ että $\text{SAT} \leq_m^p A$, jolloin kyseiset A myös ovat NP-täydellisiä
- tässä $A = \text{IS}, \text{VC}, \text{HC}, \text{TSP}, \dots$
- **tekninen ongelma:** hankalaa käsitellä mielivaltaisia propositiologiikan kaavoja
- **ratkaisu:** määritellään luokka syntaktisesti "yksinkertaisia" kaavoja, jotka kuitenkin ovat riittävän "vaikeita" että SAT on NP-täydellinen myös näihin "yksinkertaisiin" kaavoihin rajoitettuna
- tarkemmin: määritellään SAT-ongelman erikoistapaus **CSAT**, ja edelleen tämän erikoistapaus **3SAT**, joille

$$\text{SAT} \leq_m^p \text{CSAT} \leq_m^p \text{3SAT} \leq_m^p \text{SAT}$$

ja siis $\text{SAT} \leq_m^p A$ jos ja vain jos $\text{3SAT} \leq_m^p A$

Propositiologiikan kaavojen luokittelua

(Jatkossa "kaava" tarkoittaa propositiologiikan kaavaa.)

- **Literaali** on muuttuja tai muuttujan negaatio; esim. x_7 tai $\neg x_3$
- **Klausuuli** (engl. clause) on literaalien disjunktio; esim. $x_3 \vee \neg x_2 \vee x_4$ tai x_1
- kaava on **konjunkttiivisessa normaalimuodossa** (conjunctive normal form, CNF) jos se on klausuulien konjunktio; esim.
 $(x_3 \vee \neg x_2 \vee x_4) \wedge (x_7 \vee \neg x_3) \wedge x_1$
- CNF-kaava on **k -konjunkttiivisessa normaalimuodossa** (k -CNF) jos jokaisessa klausuulissa on tasan k literaalia; esim.
 $(x_3 \vee \neg x_2) \wedge (x_7 \vee \neg x_3) \wedge (x_1 \vee x_4)$ on 2-CNF-kaava

Määritellään nyt SAT-ongelman erikoistapaukset CSAT ja k SAT, $k = 1, 2, 3, \dots$:

$$\begin{aligned} \text{CSAT} &= \{ \phi \mid \phi \text{ on toteutuva CNF-kaava} \} \\ k\text{SAT} &= \{ \phi \mid \phi \text{ on toteutuva } k\text{-CNF-kaava} \} \end{aligned}$$

Kuten pian nähdään, jokaiselle kaavalle $\phi(x_1, \dots, x_n)$ on olemassa CNF-kaava $\psi(x_1, \dots, x_n)$ jolle $\phi(v_1, \dots, v_n) = \psi(v_1, \dots, v_n)$ kaikilla $(v_1, \dots, v_n) \in \{0, 1\}^n$.

Todetaan ensin että mikä tahansa kaava voidaan muuntaa muotoon, jossa **negaatiot kohdistuvat suoraan muuttujiin**, ts. ei esiinny muotoa $\neg(A \wedge B)$ tai $\neg(A \vee B)$ olevia osakaavoja.

Tämä perustuu de Morganin lakeihin

$$\neg(A \wedge B) \Leftrightarrow \neg(A) \vee \neg(B)$$

$$\neg(A \vee B) \Leftrightarrow \neg(A) \wedge \neg(B)$$

ja kaksinkertaisen negaation lakiin

$$\neg\neg A \Leftrightarrow A.$$

Esimerkki negaatioiden "painamisesta" literaaleihin

Lähtökohtana kaava

$$\neg(x_2 \wedge (\neg(x_1 \wedge \neg x_3) \vee (x_3 \wedge x_4))).$$

Saadaan

$$\begin{aligned} & \neg(x_2 \wedge (\neg(x_1 \wedge \neg x_3) \vee (x_3 \wedge x_4))) \\ \Leftrightarrow & \neg x_2 \vee \neg(\neg(x_1 \wedge \neg x_3) \vee (x_3 \wedge x_4)) \\ \Leftrightarrow & \neg x_2 \vee \neg\neg(x_1 \wedge \neg x_3) \wedge \neg(x_3 \wedge x_4) \\ \Leftrightarrow & \neg x_2 \vee (x_1 \wedge \neg x_3) \wedge (\neg x_3 \vee \neg x_4). \end{aligned}$$

- jotta saataisiin CNF-kaavoja pitää lisäksi painaa disjunktiot konjunktoiden sisäpuolelle
- tämä onnistuu osittelulakien avulla:

$$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$$

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$$

- **Ongelma:** kaavan koko voi kasvaa eksponentiaalisesti; esim. kaavan $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$ CNF-esitys on

$$\begin{aligned} & (x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee x_3 \vee x_6) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_1 \vee x_4 \vee x_6) \\ & \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee x_6) \wedge (x_2 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_6). \end{aligned}$$

- **Ratkaisu:** muodostetaan ei-CNF-kaavasta $\phi(x_1, \dots, x_n)$ CNF-kaava $\psi(x_1, \dots, x_n, y_1, \dots, y_m)$ missä y_1, \dots, y_m ovat uusia muuttujia ja $\phi(v_1, \dots, v_n) = 1$ jos ja vain jos $\psi(v_1, \dots, v_n, v'_1, \dots, v'_m)$ jollain $(v'_1, \dots, v'_m) \in \{0, 1\}^m$

Seuraava rekursiivinen funktio $\text{CNF}(\phi)$ palauttaa CNF-kaavan ψ joka on toteutuva joss ϕ on, **olettaen** että kaavassa ϕ negaatiot kohdistuvat suoraan muuttujiin.

$\text{CNF}(\phi)$: kolme tapausta kaavan ϕ muodon mukaan.

1. ϕ on literaali: palauta ϕ sellaisenaan.
2. $\phi = \phi_1 \wedge \phi_2$: laske $\psi_1 = \text{CNF}(\phi_1)$ ja $\psi_2 = \text{CNF}(\phi_2)$; palauta $\psi = \psi_1 \wedge \psi_2$.
3. $\phi = \phi_1 \vee \phi_2$: laske $\psi_1 = \text{CNF}(\phi_1)$ ja $\psi_2 = \text{CNF}(\phi_2)$; olkoon

$$\begin{aligned}\psi_1 &= g_1 \wedge g_2 \wedge \dots \wedge g_p \\ \psi_2 &= h_1 \wedge h_2 \wedge \dots \wedge h_q\end{aligned}$$

missä g_i ja h_j ovat klausuuleja. Ota käyttöön uusi muuttujasymboli y ja palauta kaava

$$(y \vee g_1) \wedge \dots \wedge (y \vee g_p) \wedge (\neg y \vee h_1) \wedge \dots \wedge (\neg y \vee h_q).$$

Esimerkki

Lasketaan $\text{CNF}(\phi)$ kun

$$\phi(x_1, x_2, x_3, x_4, x_5, x_6) = ((x_1 \wedge x_2) \vee (x_3 \wedge x_4)) \vee (x_5 \wedge x_6).$$

Yksinkertaisuuden vuoksi tehdään laskenta alhaalta ylöspäin.

Selvästi $\text{CNF}(x_1 \wedge x_2) = x_1 \wedge x_2$ ja $\text{CNF}(x_3 \wedge x_4) = x_3 \wedge x_4$. Siis

$$\text{CNF}((x_1 \wedge x_2) \vee (x_3 \wedge x_4)) = (y \vee x_1) \wedge (y \vee x_2) \wedge (\neg y \vee x_3) \wedge (\neg y \vee x_4)$$

missä y on uusi muuttuja. Edelleen $\text{CNF}(x_5 \wedge x_6) = x_5 \wedge x_6$ joten ottamalla käyttöön taas uusi muuttuja z saadaan $\text{CNF}(\phi) = \psi$ missä

$$\begin{aligned} \psi(x_1, x_2, x_3, x_4, x_5, x_6, y, z) = & (z \vee y \vee x_1) \wedge (z \vee y \vee x_2) \wedge (z \vee \neg y \vee x_3) \\ & \wedge (z \vee \neg y \vee x_4) \wedge (\neg z \vee x_5) \wedge (\neg z \vee x_6). \end{aligned}$$

Nyt esim. $\phi(0, 0, 1, 1, 0, 0) = 1$, mitä vastaten $\psi(0, 0, 1, 1, 0, 0, y, z) = 1$ kun valitaan $y = 1$ ja $z = 0$.

Lause: Olkoon $\phi(x_1, \dots, x_n)$ kaava jossa negaatiot kohdistuvat suoraan muuttujiin, ja $\text{CNF}(\phi(x_1, \dots, x_n)) = \psi(x_1, \dots, x_n, y_1, \dots, y_m)$. Nyt ψ on CNF-kaava, ja $\phi(v_1, \dots, v_n) = 1$ jos ja vain jos $\psi(x_1, \dots, x_n, v'_1, \dots, v'_m) = 1$ joillain v'_i .

Todistus: Induktio kaavan ϕ sisältämien disjunktoiden ja konjunktoiden lukumäärän suhteen.

Perustapaus: ei konjunktioita eikä disjunktioita. Siis ϕ on literaali x_i tai $\neg x_j$ jotka sellaisenaan ovat CNF-kaavoja. Väite selvästi pätee.

Induktioaskel: Oletetaan, että väite pätee kun konjunktioita ja disjunktioita on kork. n ; olkoon niitä nyt $n + 1$.

On ilmeistä että palautettavat kaavat ovat CNF-kaavoja; mielenkiintoinen puoli on niiden toteutuvuus.

Tapaus 1: $\phi(x_1, \dots, x_n) = \phi_1(x_1, \dots, x_n) \wedge \phi_2(x_1, \dots, x_n)$. Merk.

$\text{CNF}(\phi_1(x_1, \dots, x_n)) = \psi_1(x_1, \dots, x_n, y'_1, \dots, y'_r)$ ja

$\text{CNF}(\phi_2(x_1, \dots, x_n)) = \psi_2(x_1, \dots, x_n, y''_1, \dots, y''_s)$. Siis $\text{CNF}(\phi) = \psi$ missä

$$\begin{aligned} \psi(x_1, \dots, x_n, y'_1, \dots, y'_r, y''_1, \dots, y''_s) \\ = \psi_1(x_1, \dots, x_n, y'_1, \dots, y'_r) \wedge \psi_2(x_1, \dots, x_n, y''_1, \dots, y''_s). \end{aligned}$$

Konstruktioista seuraa että muuttujajoukot $\{y'_1, \dots, y'_r\}$ ja $\{y''_1, \dots, y''_s\}$ ovat erilliset, joten

$$\begin{aligned} \phi(v_1, \dots, v_n) = 1 &\Leftrightarrow \phi_1(v_1, \dots, v_n) = 1 \quad \text{ja} \quad \phi_2(v_1, \dots, v_n) = 1 \\ &\Leftrightarrow \psi_1(v_1, \dots, v_n, y'_1, \dots, y'_r) = 1 \text{ joillain } y'_i \text{ ja} \\ &\quad \psi_2(v_1, \dots, v_n, y''_1, \dots, y''_s) = 1 \text{ joillain } y''_i \text{ (ind.ol.)} \\ &\Leftrightarrow \psi(v_1, \dots, v_n, y'_1, \dots, y'_r, y''_1, \dots, y''_s) = 1 \\ &\quad \text{joillain } y_i, y'_i \text{ (koska muuttujajoukot erilliset).} \end{aligned}$$

Tapaus 2: $\phi = \phi_1 \vee \phi_2$ (jätetään nyt selvyyden vuoksi muuttujat merkitsemättä; ne voidaan käsitellä kuten tapauksessa 1). Merk.

$$\begin{aligned}\text{CNF}(\phi_1) &= g_1 \wedge g_2 \wedge \dots \wedge g_p \\ \text{CNF}(\phi_2) &= h_1 \wedge h_2 \wedge \dots \wedge h_q\end{aligned}$$

Siis $\text{CNF}(\phi) = \psi$ missä

$$\psi(y) = (y \vee g_1) \wedge \dots \wedge (y \vee g_p) \wedge (\neg y \vee h_1) \wedge \dots \wedge (\neg y \vee h_q).$$

Jos $\phi = 1$ niin $\phi_1 = 1$ tai $\phi_2 = 1$. Jos $\phi_1 = 1$, niin $g_i = 1$ kaikilla i , joten $\psi(0) = 1$. Vastaavasti jos $\phi_2 = 1$, niin $\psi(1) = 1$. Kummassakin tapauksessa $\psi(y)$ on siis toteutuva.

Kääntäen jos $\psi(y)$ on toteutuva, niin $\psi(0) = 1$ tai $\psi(1) = 1$, jolloin $\phi_1 = 1$ tai $\phi_2 = 1$. □

Olkoon f seuraava funktio, joka saa argumenttina mielivaltaisen kaavan ϕ ja palauttaa CNF-kaavan ψ :

1. Muunna de Morganin ja kaksinkertaisen negaation lakeja käyttäen ϕ yhtäpitäväksi kaavaksi $\tilde{\phi}$ jossa negaatiot kohdistuvat suoraan muuttujiin.
2. Palauta $\psi = \text{CNF}(\tilde{\phi})$.

Vaihe 1 toimii selvästi polynomisessa ajassa. Myös $\text{CNF}(\tilde{\phi})$ voidaan laskea polynomisessa ajassa (ks. alla) joten f voidaan laskea polynomisessa ajassa.

Olemme myös nähneet että ϕ on toteutuva jos ja vain jos ψ on, eli $\phi \in \text{SAT}$ joss $f(\phi) \in \text{CSAT}$. Siis $\text{SAT} \leq_m^p \text{CSAT}$.

Koska SAT on NP-täydellinen ja $\text{CSAT} \in \text{NP}$, seuraa erityisesti

Korollaari CSAT on NP-täydellinen.

Funktion CNF aikavaativuusanalyysi

Olkoon $T(n)$ kutsun $\text{CNF}(\tilde{\phi})$ aikavaativuus kun kaavassa $\tilde{\phi}$ on n konjunktio- ja disjunktiosymbolia. Huomaa että kaavan kokonaispituus on $\Theta(n)$.

Selvästi

$$T(n) \leq a \quad \text{jollain vakiolla } a.$$

Tapauksessa $n > 1$ voidaan kirjoittaa

$$T(n) \leq T(i) + T(n - i - 1) + bn$$

missä $T(i)$ ja $T(n - i - 1)$ ovat rekursiivisten kutsujen $\text{CNF}(\tilde{\phi}_1)$ ja $\text{CNF}(\tilde{\phi}_2)$ suoritusajat ja bn kaikkeen muuhun kuluva aika (b vakio). (Huomaa että yhtä lukuunottamatta kaikki kaavan $\tilde{\phi}$ symbolit menevät joko kaavoihin $\tilde{\phi}_1$ ja $\tilde{\phi}_2$.)

Väite: $T(n) \leq (a + b)n^2 + a$.

Induktiotodistus: Tapaus $n = 0$ selvä; ol. väite pätee kun $n < m$.

Huomaa että funktion $x \mapsto x^2 + (m - 1 - x)^2$ kuvaaja on ylöspäin aukeava paraabeli, joten funktio saavuttaa suurimman arvonsa välin päätepisteessä.

Siis

$$\begin{aligned} T(m) &\leq \max_{0 \leq i \leq m-1} (T(i) + T(m - i - 1)) + bm \\ &\leq (a + b) \max_{0 \leq i \leq m-1} (i^2 + (m - i - 1)^2) + 2a + bm \\ &= (a + b)(m - 1)^2 + 2a + bm \\ &= (a + b)m^2 + m(b - 2(a + b)) + 3a + b \\ &\leq (a + b)m^2 + a \end{aligned}$$

kun $m \geq 1$.



3SAT-ongelman NP-täydellisyys

- erotukseksi yleisestä CNF-esityksestä, kaikilla kaavoilla ei ole 3-CNF-esitystä; esim. $x_1 \vee x_2 \vee x_3 \vee x_4$
- esitämme muunnoksen, jolla polynomisessa ajassa mielivaltaisesta CNF-kaavasta ϕ tuotetaan 3-CNF-kaava ψ joka on toteutuva joss ϕ on

Tämän muunnoksen olemassaolosta seuraa siis

Lause: $\text{CSAT} \leq_m^p \text{3SAT}$.

Koska CSAT on NP-täydellinen ja $\text{3SAT} \in \text{NP}$, saadaan

Korollaari: 3SAT on NP-täydellinen.

Huom. SAT-ongelma rajoitettuna 2-CNF-kaavoihin ratkeaa polynomisessa ajassa.

Todistus sille että $\text{CSAT} \leq_m^p \text{3SAT}$

Olkoon $F(x_1, \dots, x_n)$ klausuuli (siis literaalien disjunktio).

Muodostamme 3-CNF-kaavan $\tilde{F}(x_1, \dots, x_n, y_1, \dots, y_m)$ missä y_i :t ovat uusia muuttujia ja $F(v_1, \dots, v_n) = 1$ jos ja vain jos $\tilde{F}(v_1, \dots, v_n, v'_1, \dots, v'_m) = 1$ joillain $(v'_1, \dots, v'_m) \in \{0, 1\}^m$.

Tästä seuraa yleisemmin, että kun ϕ on CNF-kaava

$$\phi = \bigwedge_{j=1}^k F_j$$

niin voidaan muodostaa 3-CNF-kaavojen konjunktio, eli edelleen 3-CNF-kaava,

$$\psi = \bigwedge_{j=1}^k \tilde{F}_j$$

missä $\phi(v_1, \dots, v_n) = 1$ joss $\psi(v_1, \dots, v_n, v'_1, \dots, v'_l) = 1$ jollain $(v'_1, \dots, v'_l) \in \{0, 1\}^l$, joten erityisesti ϕ on toteutuva joss ψ on toteutuva.

Huom. eri kaavoihin \tilde{F}_j lisättävien muuttujien pitää olla erillisiä.

Olkoon F klausuuli. Neljä tapausta sen mukaan, kuinka monta literaalia z_i klausuuli F sisältää:

1. $F = z_1$ missä $z_1 = x_i$ tai $z_1 = \neg x_i$ jollain i . Uudet muuttujat u ja v ;

$$\tilde{F} = (z \vee u \vee v) \wedge (z \vee u \vee \neg v) \wedge (z \vee \neg u \vee v) \wedge (z \vee \neg u \vee \neg v).$$

Koska uudet muuttujat u ja v on otettu kaikilla merkkikombinaatioilla, ainoa tapa saada $\tilde{F} = 1$ on valita $z_1 = 1$ jolloin $F = 1$.

2. $F = z_1 \vee z_2$. Uusi muuttuja w ;

$$\tilde{F} = (z_1 \vee z_2 \vee w) \wedge (z_1 \vee z_2 \vee \neg w).$$

Kuten edellisessä kohdassa \tilde{F} voidaan toteuttaa vain tekemällä F todeksi.

3. $F = z_1 \vee z_2 \vee z_3$: valmiiksi 3-CNF; $\tilde{F} = F$.

4. $F = z_1 \vee z_2 \vee \dots \vee z_m$ missä $m \geq 4$. Uudet muuttujat y_1, \dots, y_{m-3} ;

$$\tilde{F} = (z_1 \vee z_2 \vee y_1) \wedge (z_3 \vee \neg y_1 \vee y_2) \wedge (z_4 \vee \neg y_2 \vee y_3) \wedge \dots \wedge (z_{m-2} \vee \neg y_{m-4} \vee y_{m-3}) \wedge (z_{m-1} \vee z_m \vee \neg y_{m-3}).$$

- Jos $F = 1$, niin $z_i = 1$ jollain i , joten $\tilde{F} = 1$ kun valitaan $y_j = 1$ kun $j < i - 1$ ja $y_j = 0$ muuten.

- Jos $F = 0$ ja siis $z_i = 0$ kaikilla i , niin $\tilde{F} = 0$ miten tahansa y_j :t valitaankin.

Jos nimittäin yritetään saada $\tilde{F} = 1$, pitää 1. klausuulin ja ehdon $z_1 = z_2 = 0$ takia valita $y_1 = 1$.

Kun $y_1 = 1$ ja $z_3 = 0$, toisen klausuulin takia pitää ottaa $y_2 = 1$.

Näin päädytään lopulta valitsemaan $y_{m-3} = 1$, mutta tällöin viimeinen klausuuli jää toteutumatta.

Siis $\phi \in \text{CSAT}$ jos ja vain jos $\psi \in \text{3SAT}$. Selvästi muunnos $\phi \mapsto \psi$ voidaan laskea polynomisessa ajassa. □

NP-täydellisyys tähän mennessä:

- jos jollakin NP-täydellisellä ongelmalla on polynominen ratkaisualgoritmi, niin $P = NP$
- tätä ei pidetä luultavana, koska se tarkoittaisi että eksponentiaalisen suuria hakuavaruuksia voitaisiin etsiä polynomisessa ajassa
- suoraan määritelmästä todistettiin, että SAT on NP-täydellinen
- suoraviivaisilla palautuksilla osoitettiin, että jo SAT-ongelman rajoitetut erikoistapaukset CSAT ja 3SAT ovat NP-täydellisiä

Jatko:

- osoitetaan $3SAT \leq_m^p IS$ ja $3SAT \leq_m^p HC$
- siis IS ja HC NP-täydellisiä
- aiemmin osoitettu $IS \leq_m^p VC$ ja $HC \leq_m^p TSP$ joten myös VC ja TSP NP-täydellisiä

Lause: Riippumaton joukko -ongelma on NP-täydellinen

Todistus: Tarkastellaan siis joukkoa

$$IS = \{ \langle G, k \rangle \mid G \text{ sisältää } k\text{-solmuisen riippumattoman joukon} \}.$$

Selvästi $IS \in NP$. Muodostetaan palautus $f: 3SAT \leq_m^p IS$. Koska 3SAT on NP-täydellinen, väite seuraa.

Siis $f(x) = \langle G, k \rangle$ missä $\langle G, k \rangle \in IS$ jos ja vain jos x on toteutuva 3-CNF-kaava.

Erityisesti jos x ylipäänsä ei ole 3-CNF-kaava, pitää olla $f(x) \notin IS$. Tällaisissa tapauksissa valitaan esim. $f(x) = \langle (V, E), n + 1 \rangle$ missä $|V| = n$. Keskitytään jatkossa tapaukseen jossa x todella on 3-CNF-kaava.

Olkoon siis ϕ 3-CNF-kaava, jossa m klausuulia:

$$\phi = \bigwedge_{i=1}^m (z_{i,1} \vee z_{i,2} \vee z_{i,3})$$

missä kukin $z_{i,r}$ on muotoa x_j tai $\neg x_j$.

Nyt $f(\phi) = \langle (V, E), k \rangle$ missä

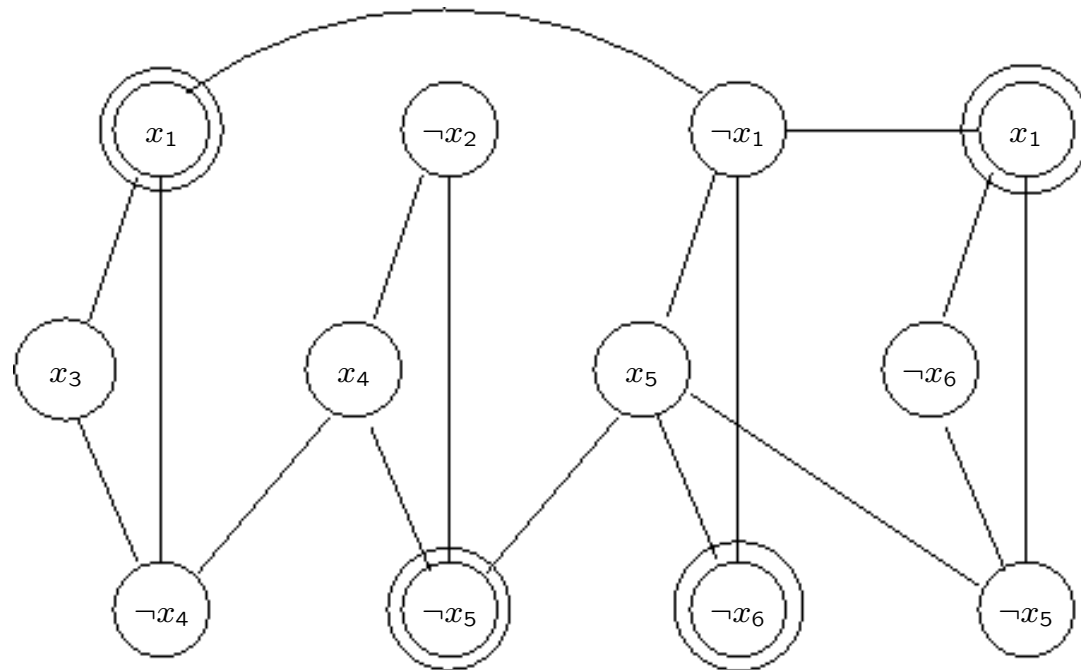
- $k = m$
- $|V| = 3m$ ja $V = \{v_{i,r} \mid i \in \{1, \dots, m\}, r \in \{1, 2, 3\}\}$
- $(v_{i,r}, v_{i,s}) \in E$ kaikilla $i \in \{1, \dots, m\}$, $r, s \in \{1, 2, 3\}$, $r \neq s$
- kun $i \neq j$, niin $(v_{i,r}, v_{j,s}) \in E$ jos $z_{i,r} = \neg z_{j,s}$ tai $\neg z_{i,r} = z_{j,s}$

Selvästi f laskettavissa polynomisessa ajassa.

Esimerkki: Muodostetaan $G = f(\phi)$ kun

$$\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_5 \vee \neg x_6) \wedge (x_1 \vee \neg x_6 \vee \neg x_5).$$

Rengastetuista solmuista muodostuu kokoa 4 oleva solmupeite; vastaa kaavan toteuttavia arvoja $x_1 = 1$, $x_5 = 0$, $x_6 = 0$, muut muuttujat mielivaltaisia



Nähdään että ϕ toteutuva \Leftrightarrow verkossa G on m solmun riippumaton joukko U :

\Rightarrow : Ol. ϕ toteutuva. Valitaan muuttujien x_i arvot siten, että jokaisessa kaavan ϕ klausuulissa i ainakin yksi literaali $z_{i,r}$ saa arvon 1. Valitaan joukkoon U vastaavat $v_{i,r}$.

Nyt U sisältää tasan yhden solmun jokaisesta "kolmiosta".

Lisäksi valitut solmut vastaavat "tosia" literaaleja, joten minkään kahden eri "kolmiosta" valitun solmun välillä ei ole kaarta.

Siis U on riippumaton ja $|U| = m$.

\Leftarrow : Ol. m solmun joukko U riippumaton. Nyt U sisältää tasan yhden solmun joka "kolmiosta".

Koska minkään eri "kolmioista" valittujen solmujen välillä ei ole kaarta, voidaan ilman ristiriitaa asettaa "todeksi" valittuja solmuja vastaavat literaalit.

Jokaiseen klausuuliin tulee ainakin yksi "tosi" literaali; ϕ toteutuva. □

Tarkastellaan seuraavaksi **suunnattu Hamiltonin kehä** -ongelmaa (Directed Hamiltonian Circuit, DHC):

Annettu: suunnattu verkko G

Kysymys: onko verkossa G suunnattu polku joka käy jokaisessa solmussa tasan kerran ja palaa lähtösolmuunsa

Siis eroksi alkuperäiseen Hamiltonin kehä -ongelmaan (HC) verkon kaarilla on suunta, ja kehän pitää näitä suuntia noudattaa.

Lause: DHC on NP-täydellinen.

Koska $HC \in NP$ ja harjoitustehtävänä osoitetaan $DHC \leq_m^p HC$, saadaan

Korollaari: HC on NP-täydellinen.

Todistus: Kuten suuntaamattomassa tapauksessa nähdään helposti että $DHC \in NP$. Muodostetaan palautus $f: 3SAT \leq_m^p DHC$ mistä väite nyt seuraa.

Olkoon annettu n -muuttujainen 3-CNF-kaava ϕ jossa m klausuulia

$$\phi = \bigwedge_{j=1}^m (z_{j,1} \vee z_{j,2} \vee z_{j,3}).$$

Suunnattu verkko $f(\phi) = G = (V, E)$ muodostuu kahdenlaisista osaverkoista eli **laitteista** (gadget):

- jokaista muuttujaa x_i kohti laite A_i , $i = 1, \dots, n$
- laite A_i voidaan käydä läpi kahdella eri tavalla jotka koodaavat valintaa $x_i = 0$ tai $x_i = 1$
- jokaista klausuulia $\beta_j = z_{j,1} \vee z_{j,2} \vee z_{j,3}$ kohti laite B_j , $j = 1, \dots, m$
- laitteen B_j läpikäynti voidaan lomittaa laitteen A_i läpikäyntiin joss muuttuja x_i "tekee todeksi" klausuulin β_j

Laite A_i : laitteessa on $2m + 4$ solmua a_i , d_i sekä $b_{i,j}$ ja $c_{i,j}$, $j = 0, \dots, m$.

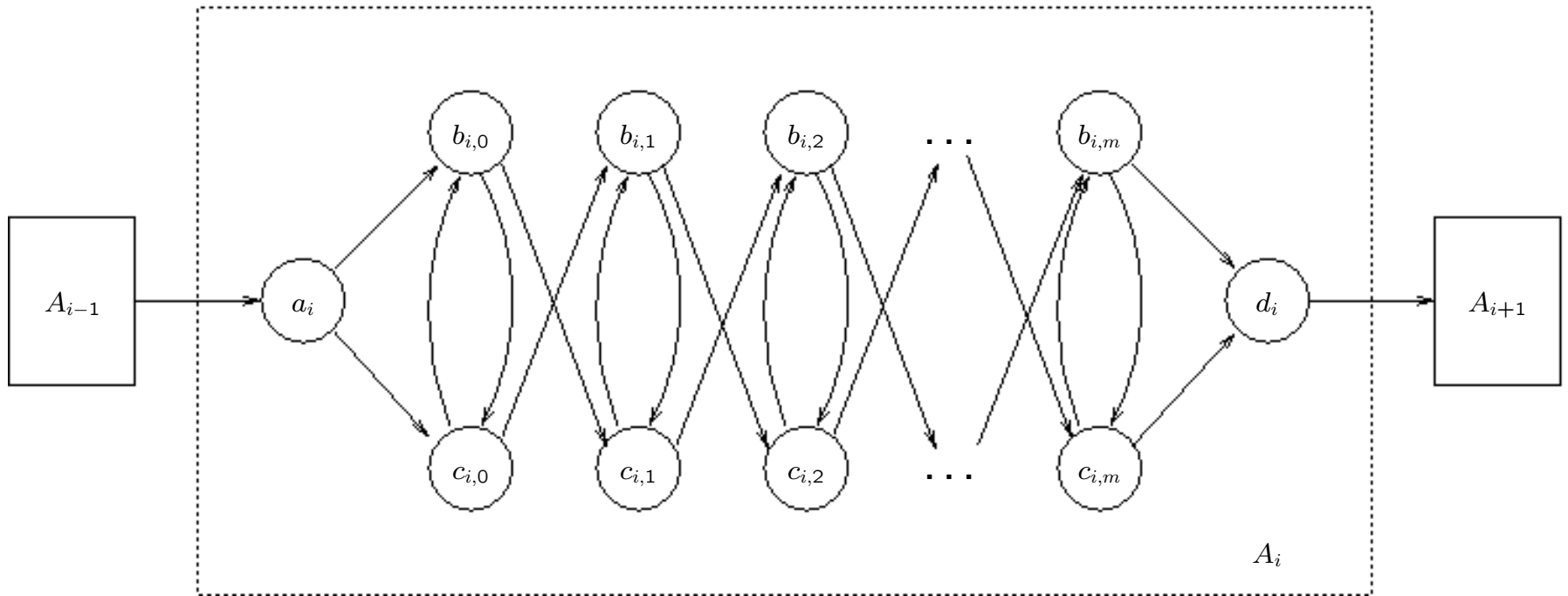
Laitteen sisäiset kaaret:

- solmusta a_i kaari solmuihin $b_{i,0}$ ja $c_{i,0}$
- solmuista $b_{i,m}$ ja $c_{i,m}$ kaari solmuun d_i
- solmusta $b_{i,j}$ kaari solmuun $c_{i,j}$ ja solmusta $c_{i,j}$ kaari solmuun $b_{i,j}$,
 $j = 0, \dots, m$
- solmusta $b_{i,j-1}$ kaari solmuun $c_{i,j}$ ja solmusta $c_{i,j-1}$ kaari solmuun $b_{i,j}$,
 $j = 1, \dots, m$

Havaitaan että laitteen A_i solmut voidaan käydä läpi kahdessa järjestyksessä:

- järjestys $a_i - c_{i,0} - b_{i,0} - c_{i,1} - b_{i,1} - \dots - c_{i,m} - b_{i,m} - d_i$ vastaa valintaa $x_i = 0$
- järjestys $a_i - b_{i,0} - c_{i,0} - b_{i,1} - c_{i,1} - \dots - b_{i,m} - c_{i,m} - d_i$ vastaa valintaa $x_i = 1$

Eri laitteita yhdistävät kaaret esitellään pian.



Laite A_i . Kuvasta puuttuu kaaret laitteisiin B_j .

Laite B_j : laitteessa 6 solmua $r_{j,1}, r_{j,2}, r_{j,3}, s_{j,1}, s_{j,2}, s_{j,3}$.

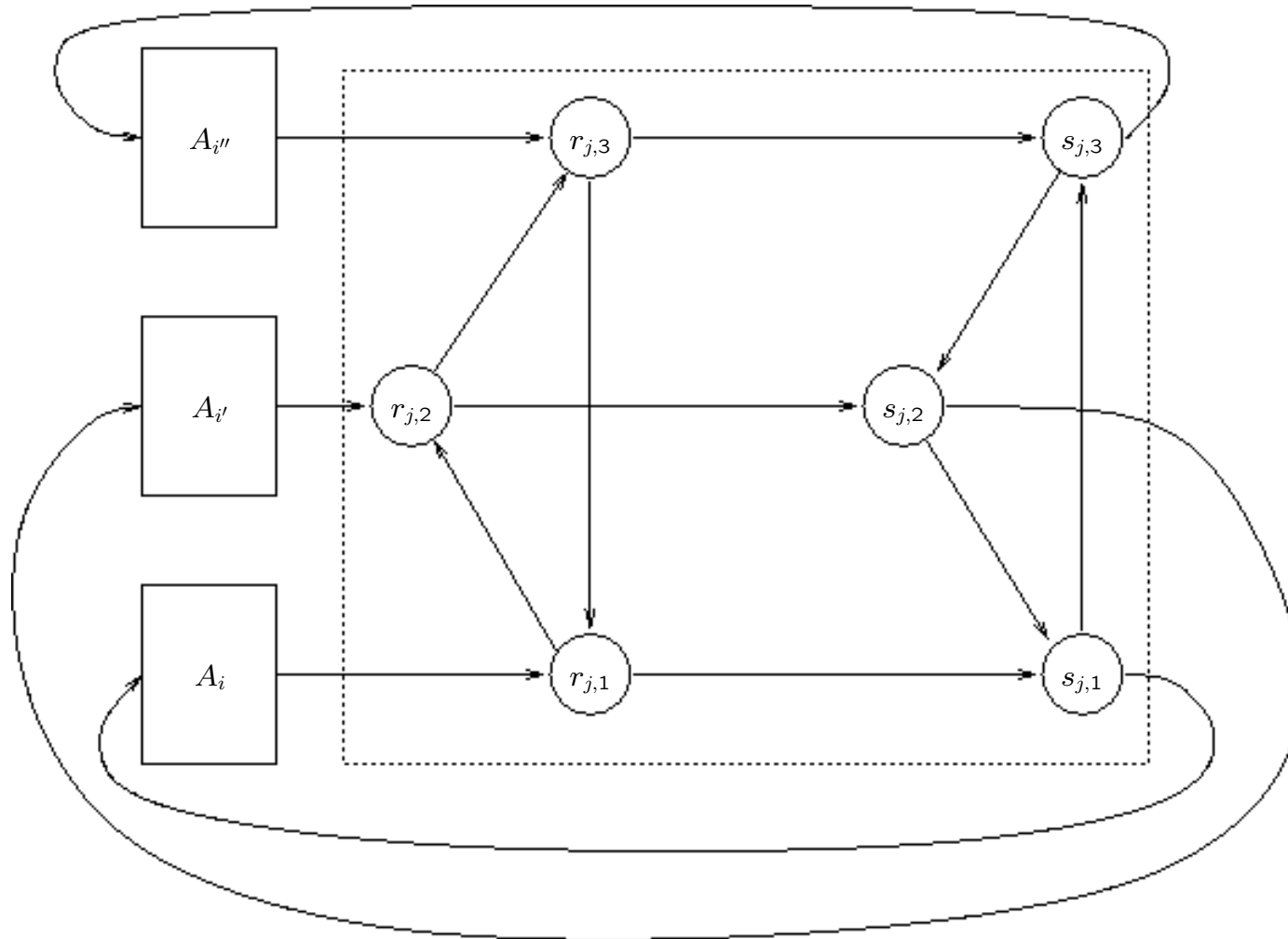
Laitteen sisäiset kaaret:

- solmut $r_{j,k}$ sisältävä sykli: kaaret $(r_{j,1}, r_{j,2}), (r_{j,2}, r_{j,3})$ ja $(r_{j,3}, r_{j,1})$
- solmut $s_{j,k}$ sisältävä sykli: kaaret $(s_{j,1}, s_{j,3}), (s_{j,3}, s_{j,2})$ ja $(s_{j,2}, s_{j,1})$ (huom. suunta)
- syklejä kytkevät kaaret $(r_{j,1}, s_{j,1}), (r_{j,2}, s_{j,2})$ ja $(r_{j,3}, s_{j,3})$

Havaitaan että jos Hamiltonin kehä tulee laitteeseen B_j solmun $r_{j,k}$ kautta niin sen pitää vastaavasti poistua solmun $s_{j,k}$ kautta.

(Kehä voi vieraillla laitteessa B_j useita kertoja, mutta jokaisen vierailun on noudatettava tätä sääntöä.)

Muuten osa solmuista jäisi eristyksiin eikä enää pääsisi mukaan kehään.



Laite B_j . Tässä literaalien β_j muuttujat ovat x_i , $x_{i'}$ ja $x_{i''}$.

Laitteet A_i yhdistetään yhdeksi isoksi sykliksi:

- solmusta d_i kaari solmuun a_{i+1} , $i = 1, \dots, n - 1$
- solmusta d_n kaari solmuun a_1

Laite A_i yhdistetään niihin laitteisiin B_j joilla klausuuli β_j sisältää literaalin x_i tai $\neg x_i$:

- jos $z_{j,k} = x_i$ niin lisätään kaaret $(c_{i,j-1}, r_{j,k})$ ja $(s_{j,k}, b_{i,j})$
- jos $z_{j,k} = \neg x_i$ niin lisätään kaaret $(b_{i,j-1}, r_{j,k})$ ja $(s_{j,k}, c_{i,j})$

Havainto:

Jos klausuuli β_j sisältää literaalin x_i (vast. $\neg x_i$)

ja laitteen A_i läpikäyntijärjestys vastaa valintaa $x_i = 1$ (vast. $x_i = 0$)

niin laitteen B_j läpikäynti voidaan sijoittaa laitteen A_i läpikäynnin lomaan.

Verkkoon $G = f(\phi)$ ei tule muita kaaria kuin edellä luetellut.

Selvästi f voidaan laskea polynomisessa ajassa. Seuraavasta väitteestä seuraa että $f: 3SAT \leq_m^p DHC$.

Väite: ϕ on toteutuva \Leftrightarrow verkossa G on Hamiltonin kehä

\Rightarrow : Olkoon $\phi = 1$ kun $(x_1, \dots, x_n) = (v_i, \dots, v_n) \in \{0, 1\}^n$.

Siis jokaisella klausuulilla

$$\beta_j = z_{j,1} \vee z_{j,2} \vee z_{j,3}$$

ainakin yksi literaali $z_{j,k}$ on tosi kun $(x_1, \dots, x_n) = (v_i, \dots, v_n)$. Valitaan jokin indeksi $k(j) \in \{1, 2, 3\}$ s.e. jollain i joko $z_{j,k(j)} = x_i$ ja $v_i = 1$, tai $z_{j,k(j)} = \neg x_i$ ja $v_i = 0$,

Siis klausuulin numero j toteutuminen on siinä olevan literaalin numero $k(j)$ vastuulla.

Verkkoon G voidaan muodostaa Hamiltonin kehä seuraavasti:

1. käy laite A_i läpi järjestyksessä

- $a_i - c_{i,0} - b_{i,0} - \dots - c_{i,m} - b_{i,m} - d_i$ jos $v_i = 0$
- $a_i - b_{i,0} - c_{i,0} - \dots - b_{i,m} - c_{i,m} - d_i$ jos $v_i = 1$

2. yhdistä laitteiden A_i läpikäynnit järjestyksessä $A_1 - A_2 - \dots - A_n - A_1$

3. kaikilla $j = 1, \dots, m$:

- jos $z_{j,k(j)} = \neg x_i$ (jolloin $v_i = 0$), korvaa laitteen A_i läpikäynnin kaari $(b_{i,j-1}, c_{i,j})$ polulla $(b_{i,j-1}, r_{j,k(j)}, \dots, s_{j,k(j)}, c_{i,j})$ missä $r_{j,k(j)}, \dots, s_{j,k(j)}$ on laitteen B_j läpikäynti
- jos $z_{j,k(j)} = x_i$ (jolloin $v_i = 1$), korvaa laitteen A_i läpikäynnin kaari $(c_{i,j-1}, b_{i,j})$ polulla $(c_{i,j-1}, r_{j,k(j)}, \dots, s_{j,k(j)}, b_{i,j})$ missä $r_{j,k(j)}, \dots, s_{j,k(j)}$ on laitteen B_j läpikäynti

\Leftarrow : Olkoon verkossa G Hamiltonin kehä.

Jos kehä sisältää kaaren $(a_i, b_{i,0})$ valitaan $v_i = 1$.

Jos kehä sisältää kaaren $(a_i, c_{i,0})$ valitaan $v_i = 0$.

Selvästi tasan yksi näistä pätee. Väitetään että $\phi = 1$ jos $x_i = v_i$ kaikilla i .

Jos kehä tulee laitteeseen B_j solmun $r_{j,k}$ kautta, se poistuu solmun $s_{j,k}$ kautta; muuten osa laitteen solmuista leikkautuisi pois kehältä.

Siis laitteen B_j kautta voidaan siirtyä

- solmusta $c_{i,j-1}$ solmuun $b_{i,j}$ jos β_j sisältää literaalin x_i ja
- solmusta $b_{i,j-1}$ solmuun $c_{i,j}$ jos β_j sisältää literaalin $\neg x_i$.

Siis jos $v_i = 1$, laite A_i käydään läpi järjestyksessä

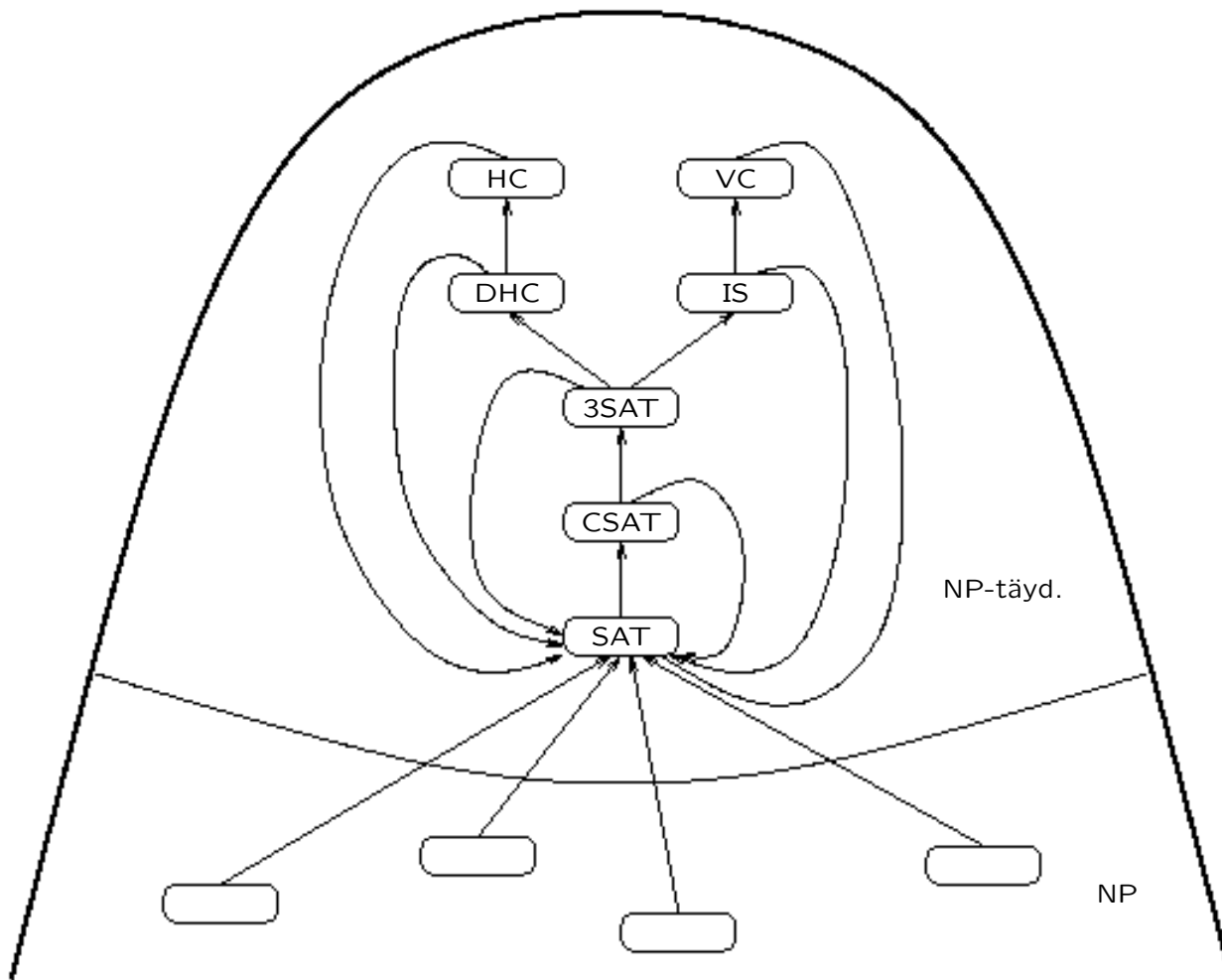
$$a_i - b_{i,0} - c_{i,0} \sim b_{i,1} - c_{i,1} \sim \dots \sim b_{i,m} - c_{i,m} - d_i$$

missä $c_{i,j-1} \sim b_{i,j}$ tarkoittaa siirtymistä solmusta $c_{i,j-1}$ solmuun $b_{i,j}$ joko suoraan tai laitteen B_j kautta. Jos $v_i = 0$, saadaan vastaavasti

$$a_i - c_{i,0} - b_{i,0} \sim c_{i,1} - b_{i,1} \sim \dots \sim c_{i,m} - b_{i,m} - d_i.$$

Jos laitteessa B_j käydään laitteen A_i läpikäynnin välissä ja $v_i = 1$, niin laitteeseen B_j mennään kaarta $(c_{i,j-1}, r_{j,k})$ jolloin $z_{j,k} = x_i$. Vastaavasti jos $v_i = 0$ niin $z_{j,k} = \neg x_i$. Siis $z_{j,k} = 1$ kun valitaan $x_i = v_i$.

Jokaisessa laitteessa B_j käydään ainakin kerran, joten jokaisessa klausuulissa β_j on ainakin yksi literaali $z_{j,k}$ joka saa arvon 1 kun $x_i = v_i$ kaikilla i . \square



Yhteenveto todistetuista polynomisista palautuksista
 Nuoli $A \rightarrow B$ tarkoittaa $A \leq_m^p B$.
 Huom. **kaikki** NP-ongelmat on palautettu SAT-ongelmaan.

Muita vaativuusluokkia

Käydään lyhyesti läpi tärkeimpiä vaativuusluokkiin liittyviä tuloksia.

Monet tunnetuista tuloksista ovat vaikeita todistaa, ja monet kysymykset ovat vielä auki.

Lause (Ladner 1975): Jos $P \neq NP$ niin luokassa $NP - P$ on muitakin kuin NP -täydellisiä kieliä.

Siis ei ole mahdollista, että luokka NP jakaantuisi toisaalta polynomisesti ratkeaviin ja toisaalta NP -täydellisiin ongelmiin: joko nämä ongelmaluokat yhtyvät tai niiden välissäkin on jotain.

Eräs ehdokas "välissä olevaksi" ongelmaksi on [verkkoisomorfia](#):

Annettu: verkot $G_1 = (V_1, E_1)$ ja $G_2 = (V_2, E_2)$

Kysymys: onko olemassa bijektio $f: V_1 \rightarrow V_2$ jolla $(u, v) \in E_1$ joss
 $((f(u), f(v)) \in E_2$

Luokan NP ongelmien komplementit muodostavat luokan

$$\text{co-NP} = \{ \bar{A} \mid A \in \text{NP} \}.$$

Luokalla co-NP on omat täydelliset ongelmansa.

Lause: A on co-NP-täydellinen jos ja vain jos \bar{A} on NP-täydellinen.

Todistus: Helppo. \square

Jos $P = \text{NP}$ niin tietysti myös $P = \text{co-NP}$.

Joka tapauksessa selvästi $P \subseteq \text{NP} \cap \text{co-NP}$.

Nykytietämyksen mukaan seuraavat vaihtoehdot ovat kaikki mahdollisia:

- $P = \text{NP} = \text{co-NP}$
- $P \neq \text{NP}$ mutta $\text{NP} = \text{co-NP}$
- $P = \text{NP} \cap \text{co-NP}$ mutta $P \neq \text{NP}$ ja $P \neq \text{co-NP}$
- $P \neq \text{NP} \cap \text{co-NP}$ ja $\text{NP} \neq \text{co-NP}$

Näistä viimeistä pidetään ehkä luultavimpana.

Koska "P = NP?" ja NP = co-NP?" ovat klassisia avoimia ongelmia, on hieman yllättävää että vastaavat tilavaativuusluokkien ongelmat ovat "helppoja":

Lause (Savitch 1970) Jos s on tilakonstruoituva ja $s(n) \geq n$ kaikilla n niin

$$\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2).$$

(Tilakonstruoituvuus on tekninen ehto jonka kaikki kiinnostavat tilavaativuusfunktiot, erityisesti polynomit, toteuttavat.)

Korollaari: $\text{PSPACE} = \text{NPSPACE} = \text{co-NPSPACE}$

Tässä on siis merkitty $\text{co-}\mathcal{C} = \{ \overline{A} \mid A \in \mathcal{C} \}$ kuten aiemminkin.

Komplementoinnin suhteen on saatu vielä tarkempikin tulos:

Lause (Immerman 1988, Szelepcsényi 1987) Jos $s(n) \geq \log n$ kaikilla n niin

$$\text{NSPACE}(s(n)) = \text{co-NSPACE}(s(n)).$$

Seuraava lause suhteuttaa hyvin karkealla tasolla aika- ja tilavaativuusluokat.

Lause: Kaikilla s pätee

$$\text{DTIME}(t(n)) \subseteq \text{DSPACE}(t(n)) \subseteq \bigcup_{c \geq 1} \text{DTIME}(c^{t(n)}).$$

Todistushahmotelma:

- ajassa $t(n)$ ei ehdi kuluttaa yli $t(n)$ paikkaa muistia
- jos koneen M tilavaativuus on $s(n)$ ja laskenta on kestänyt yli $|Q||\Gamma|^{s(n)}$ askelta, jokin tilanne on toistunut, joten kone on ikuisessa silmukassa \square

Korollaari: $P \subseteq NP \subseteq PSPACE \subseteq E \subseteq ESPACE$.

Ylläolevien sisältyvyyksien aitoudesta tiedetään hyvin vähän:

Lause: $P \neq E$ ja $PSPACE \neq ESPACE$

Toisistaan poikkeavia aika- ja tilavaativuusluokkia on ääretön määrä.

Määritelmä Funktio t on **aikakonstruoituva** jos syötteellä 1^n luvun $t(n)$ binääriesitys voidaan laskea ajassa $O(t(n))$.

Kaikki "järkevät" vähintään $n \log n$ olevat funktiot ovat aikakonstruoituvia, esim. $n \log n$, $\lfloor n^{3/2} \rfloor$, n^5 , 2^n .

Lause: Jos t on aikakonstruoituva niin on olemassa ongelma A joka voidaan ratkaista ajassa $O(t(n))$ mutta ei ajassa $o(t(n)/\log n)$.

Määritelmä Funktio s on **tilakonstruoituva** jos syötteellä 1^n luvun $s(n)$ binääriesitys voidaan laskea tilassa $O(s(n))$.

Kaikki "järkevät" vähintään $\log n$ olevat funktiot ovat tilakonstruoituvia, esim. $\log n$, $\lfloor n^{3/2} \rfloor$, n^5 , 2^n .

Lause: Jos s on tilakonstruoituva niin on olemassa ongelma A joka voidaan ratkaista tilassa $O(s(n))$ mutta ei tilassa $o(s(n))$.

Yhteenveto

Laskennan mallit

- Turingin kone on **universaali** laskennan malli: sen on tarkoitus kuvata kaikkia periaatteessa mahdollisia ”mekaanisia” laskentoja
 - alkuperäinen motivaatio logiikasta, ei tietokoneista
 - laskettavuus Turingin koneella ei riipu mallin yksityiskohdista (erityisesti deterministisyydestä)
 - vastaavia malleja on muitakin: rekursiiviset funktiot, yleiset kieliopit, Random Access Machine; ohjelmointikielet
 - kaikki nämä määrittelevät saman käsitteen ”laskettavuus”
- ⇒ Churchin-Turingin teesi: Turingin koneet ovat oikea malli mekaaniselle laskennalle

Laskettavuusteoria

- peruskysymys: mitä voidaan laskea äärellisessä ajassa
- rekursiivisuus (ratkeavuus) ja rekursiivinen lueteltavuus (osittainratkeavuus)
- tärkeä tekninen yksityiskohta: Turingin koneen esitys merkkijonona, universaali Turingin kone
- **universaalikieli** L_u : rekursiivisesti lueteltava, ei rekursiivinen
- muita ei-ratkeavia ongelmia: pysähtymisongelma, epätyyjyysongelma, tyhjiysongelma; Postin vastaavuusongelma
- **Ricen lause**: semanttiset ominaisuudet ratkeamattomia

- rekursiivinen palautus $A \leq_m B$: ongelma A ainakin "yhtä ratkeava" kuin B
- tärkeä tekniikka: A todistetaan ratkeamattomaksi osoittamalla $B \leq_m A$ missä B on jokin tunnettu ratkeamaton ongelma
- ongelma A on RE-täydellinen jos A on rekursiivisesti lueteltava ja $B \leq_m A$ kaikilla rekursiivisesti lueteltavilla B
- RE-täydelliset ongelmat ovat "maksimaalisen vaikeita" osittain ratkeavia ongelmia
- esim. universaalikieli, pysähtymisongelma, epätyhjyysongelma

Vaativuusteoria

- peruskysymys: mitä voidaan laskea polynomisessa ajassa
- kertaluokat, aika- ja tilavaativuudet; vaativuusluokat, erityisesti P
- epädeterministiset vaativuusluokat, erityisesti NP
- avoin ongelma: onko $P = NP$, ts. vaikuttaako epädeterminismi siihen mikä on laskettavissa polynomisessa ajassa
- yleinen uskomus: $P \neq NP$

- **polynominen palautus** $A \leq_m^p B$: ongelma A ainakin yhtä helppo kuin ongelma B
- A on NP-täydellinen jos $A \in \text{NP}$ ja $B \leq_m^p A$ kaikilla $B \in \text{NP}$
- jos A on NP-täydellinen ja A ratkeaa deterministisesti polynomisessa ajassa, niin $\text{P} = \text{NP}$
- jos A on NP-täydellinen, on siis aihetta uskoa että A ei ratkea deterministisesti polynomisessa ajassa
- ongelman A osoittaminen NP-täydelliseksi:
 1. osoita $A \in \text{NP}$ (yleensä helppoa)
 2. osoita $B \leq_m^p A$ jollain NP-täydelliseksi tunnetulla B