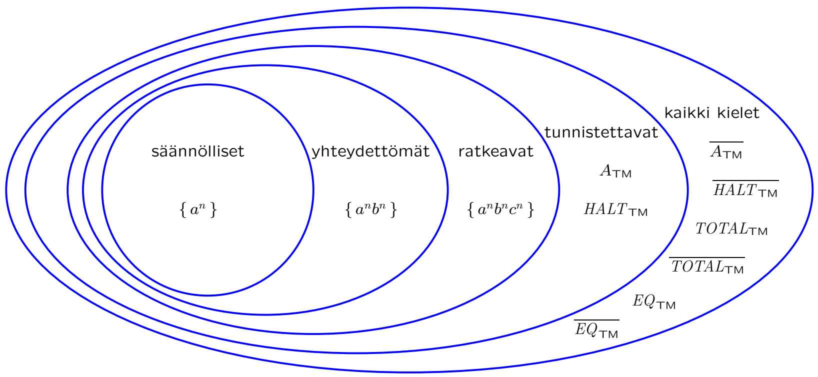
Kurssi tähän asti:



5. Laskennan vaativuus

Edellä olemme tutkineet, mitä ylipäänsä on tai ei ole *mahdollista* laskea. Nyt tarkennamme käsittelyä kysymällä, *kuinka kauan* laskenta kestää.

Käymme läpi Sipserin luvun 7 pääasiat. Tavoitteena on, että opiskelija osaa

- selittää polynomisen ja eksponentiaalisen aikavaativuuden eron
- määritellä luokat P ja NP
- selittää, mistä NP-täydellisyydessä ja P vs. NP -ongelmassa on kysymys
- tunnistaa perusesimerkkejä NP-täydellisistä ongelmista.

Aikavaativuus [Sipser luku 7.1]

Algoritmin aikavaativuuden perusajatus on tuttu esim. kurssilta *Tietorakenteet*. Täsmennämme sen nyt käyttämällä Turingin konetta laskennan mallina.

Määritelmä 5.1: Olkoon M Turingin kone, joka pysähtyy kaikilla syötteillä. Turingin koneen M *aikavaativuus* (time complexity) on funktio $f\colon \mathbb{N} \to \mathbb{N}$, missä $f(n)$ on suurin määrä laskenta-askelia, jonka M tekee millään syötteellä, jonka pituus on n. Tällöin sanomme myös, että M toimii ajassa $f(n)$. □

Siis kurssilta *Tietorakenteet* tuttuun tapaan ryhmittelemme saman pituiset syötteet yhteen ja tarkastelemme *pahimman tapauksen* aikavaativuutta. Käytämme myös *Tietorakenteista* tuttua "iso-O-merkintää": jos esim. Turingin koneen aikavaativuus on f, jolla $f(n) = O(n^2)$, sanomme lyhyesti, että koneen aikavaativuus on $O(n^2)$. Tässä siis sovitaan, että n esittää syötteen pituutta.

(Tarvittaessa kertaa iso-O-merkintä kurssikirjan sivuilta 252–254 tai *Tietorakenteiden* materiaalista.)

Esimerkki 5.2: Tarkastellaan sivun 231 Turingin konetta, joka tunnistaa kielen $\{ \mathrm{a}^m \mathrm{b}^m \mathrm{c}^m \mid m \in \mathbb{N} \}$. Pahimmassa tapauksessa kone käy jokaista syötteen a-merkkiä kohti syötteen loppuosan läpi "yliviivaten" b- ja c-merkkejä. Yksi syötteen läpikäynti vie $O(n)$ laskenta-askelta. Kaikkiaan a-merkkejä ja siis läpikäyntejä on $O(n)$ kappaletta. Lopuksi vielä tarkastetaan koko nauhan sisältö, mihin kuluu $O(n)$ askelta. Koneen aikavaativuus on siis $O(n) \cdot O(n) + O(n) = O(n^2)$. □

Määritelmä 5.3: Luokka $\mathrm{TIME}(t(n))$ koostuu kielistä, jotka voidaan tunnistaa ajassa $O(t(n))$ toimivalla Turingin koneella. □

Esimerkki 5.4: Edellisen esimerkin perusteella

$$\{ \mathrm{a}^m \mathrm{b}^m \mathrm{c}^m \mid m \in \mathbb{N} \} \in \mathrm{TIME}(n^2).$$

Tämän perusteella voidaan myös todeta heikommin, että

$$\{ \mathrm{a}^m \mathrm{b}^m \mathrm{c}^m \mid m \in \mathbb{N} \} \in \mathrm{TIME}(n^3)$$
$$\{ \mathrm{a}^m \mathrm{b}^m \mathrm{c}^m \mid m \in \mathbb{N} \} \in \mathrm{TIME}(2^n)$$

jne. □

Laskennan vaativuus eri malleissa

Edellä, erityisesti luokan TIME($f(n)$) määritelmässä, Turingin koneella on tarkoitettu determinististä, yksinauhaista konetta.

Aikavaativuuden määritelmä tietysti yleistyy suoraan moninauhaiselle koneelle, mutta aikavaativuusluokat ovat herkkiä sille, millaisia koneita käytetään.

Lause 5.5: [Sipser Thm. 7.8] Jos kieli voidaan tunnistaa ajassa $f(n)$ toimivalla *moninauhaisella* Turingin koneella, niin se voidaan tunnistaa ajassa $O(f(n)^2)$ toimivalla *yksinauhaisella* Turingin koneella.

Todistushahmotelma: Tämä on oleellisesti käyty läpi todistettaessa, että moninauhaisella ja yksinauhaisella koneella voidaan ratkaista samat ongelmat (sivut 244–245). Huomaa, että nauhojen lukumäärä k on vakio, ts. ei riipu syötteestä. □

Tämä yläraja $O(f(n)^2)$ ei kenties ole tiukin mahdollinen, mutta emme ole kiinnostuneet sen viilaamisesta. Ei ole mitään perustetta sanoa, että moninauhainen kone olisi tarkempi tai epätarkempi malli "oikealle algoritmille" kuin yksinauhainen. Pyrimmekin nostamaan tarkastelun hieman korkeammalle abstraktiotasolle, jotta tällaiset pienet erot malleissa eivät häiritse.

Epädeterminismi on paljon isompi muutos kuin moninauhaisuus.

Määritelmä 5.6: Olkoon N on epädeterministinen Turingin kone, jonka kaikki laskennat kaikilla syötteillä pysähtyvät. Koneen N aikavaativuus on funktio $f: \mathbb{N} \to \mathbb{N}$, missä $f(n)$ on suurin määrä laskenta-askelia, jonka N tekee missään laskennassa syötteellä, jonka pituus on n. □

Huom. Epädeterministinen aikavaativuus ei ole tarkoitettu suoraan mallintamaan minkään todellisen tietokoneen suoritusaikaa. Se on vain abstrakti matemaattinen käsite, jonka hyödyllisyys tulee kohta ilmi.

Tekninen huomio: Tarkastelemme siis epädeterministisen laskentapuun *pisintä* haaraa kullakin syötteellä. Lyhimmän haaran tarkasteleminen ei ole mielekästä, koska koneeseen voidaan helposti lisätä "turhia" lyhyitä hylkääviä laskentoja. Lyhimmän *hyväksyvän* laskennan tarkasteleminen johtaisi oleellisesti samoihin tuloksiin kuin ylläoleva määritelmä.

Lause 5.7: [Sipser Thm. 7.11] Jos kieli voidaan tunnistaa ajassa $f(n)$ toimivalla epädeterministisellä Turingin koneella, se voidaan tunnistaa ajassa $2^{O(f(n))}$ toimivalla deterministisellä yksinauhaisella Turingin koneella.

Todistushahmotelma: Sivuilla 250–254 on esitetty muunnos epädeterministisestä Turingin koneesta kolmenauhaiseksi deterministiseksi.

Laskenta käy läpi epädeterministisen laskentapuun haarat, joita on korkeintaan $b^{f(n)}$. Yhden haaran läpikäynti vie deterministisen ajan $O(f(n))$.

Kolmenauhaisen koneen aikavaativuus on siis $O(f(n)b^{f(n)}) = 2^{O(f(n))}$. Yksinauhaiseen koneeseen siirtyminen antaa vaativuuden $O((2^{O(f(n))})^2) = O(2^{2 \cdot O(f(n))}) = 2^{O(f(n))}$. □

Huomaa, että $2^{f(n)}$ on yleensä paljon suurempi kuin moninauhaisen koneen yhteydessä esiintynyt $f(n)^2$. Ryhdymme nyt tutkimaan tätä eroa.

Polynominen aikavaativuus [Sipser luku 7.2]

Määritelmä 5.8: Turingin koneen M aikavaativuus on *polynominen*, jos se on $O(n^k)$ jollain $k \in \mathbb{N}$. Tällöin sanomme, että M toimii polynomisessa ajassa.

Kieli kuuluu luokkaan P, jos se voidaan tunnistaa yksinauhaisella deterministisellä polynomisessa ajassa toimivalla Turingin koneella. □

Lauseen 5.6 nojalla yksinauhainen ja moninauhainen Turingin kone ovat *polynomisesti ekvivalentteja* laskennan malleja: luokan P sisältö ei muuttuisi, vaikka sen määritelmässä käytettäisiin moninauhaisia koneita. Sama pätee myös monille muille Turingin koneen muunnelmille ja esim. RAM-koneelle (s. 260).

Sen sijaan **ei** tiedetä, että epädeterministiset ja deterministiset Turingin koneet olisivat polynomisesti ekvivalentteja (ja yleisesti epäillään, että eivät ole). Palaamme tähän pian.

Polynominen aikavaativuus on siis tekniseltä kannalta sopivan tasoinen aikavaativuusluokka: se ei ole herkkä pienille tai vähän suuremmillekaan muutoksille laskennan mallissa.

Polynominen aikavaativuus on myös sovellusten kannalta keskeinen ominaisuus: yleisesti ottaen ajassa n^k toimivat algoritmit skaalatuvat suurille syötteille, ajassa 2^n toimivat eivät skaalaudu. Tähän pitää suhtautua tietyin varauksin:

- esim. aikavaativuus $O(n^{100})$ ei varmasti skaalaudu hyvin
- pahimman tapauksen analyysi ei aina ole tarkoituksenmukaista
- pientenkin syötteiden ratkaiseminen voi olla tärkeää.

Erityisesti luokkaa P tarkastelemalla **emme** halua väittää, että esim. aikavaativuuksien $O(n^2)$ ja $O(n^3)$ välinen ero ei olisi tärkeä; päinvastoin se on hyvinkin tärkeä. Olemme vain valmiit hetkeksi unohtamaan tuon eron saadaksemme paremman kuvan vielä tärkeämmästä erosta polynomisten aikavaativuuksien $O(n^k)$ ja *eksponentiaalisten* aikavaativuuksien $O(2^{n^k})$ välillä.

Esimerkki 5.9: [Sipser Thm. 7.14] Kurssilta *Tietorakenteet* tiedämme, että ongelma

Annettu: verkko $G = (V, E)$, solmut $s \in V$ ja $t \in V$
Kysymys: onko verkossa G polku $s \rightsquigarrow t$

voidaan ratkaista ajassa $O(|V| + |E|)$ esim. leveyssuuntaisella etsinnällä.

Liittääksemme tämän tuloksen nyt esillä olevaan käsitteistöön ja erityisesti luokkaan P meidän on ensin määriteltävä ongelma formaalina kielenä. Olkoon siis $PATH = \{ \langle G, s, t \rangle \mid \text{verkossa } G \text{ on polku } s \rightsquigarrow t \}$, missä $\langle G, s, t \rangle$ on mikä tahansa järkevä tapa koodata verkko ja sen kaksi solmua jonkin aakkoston merkkijonoksi.

"Järkevyys" pitää tässä sisällään erityisesti sen, että koodin pituuden tulee olla polynominen verkon solmujen lukumäärän suhteen. Esim. vierusmatriisia käyttämällä saadaan helposti koodinpituus $O(|V|^3)$.

Nyt $PATH \in$ P. Tämän todistamiseksi toteamme, että kurssilta *Tietorakenteet* tutun leveyssuuntaisen etsinnän tekemät $O(|V| + |E|)$ laskenta-askelta voidaan selvästi kukin toteuttaa syötteen koon suhteen polynomisessa ajassa Turingin koneella. □

Lause 5.10: Jos kieli L on yhteydetön, niin $L \in$ P.

Todistus: Jos L on yhteydetön, sillä on Chomskyn normaalimuodossa oleva kielioppi. Voimme siis annetulla w ratkaista, päteekö $w \in L$, käyttämällä CYK-algoritmia (s. 178). Algoritmin suoritus vaatii $O(n^3)$ iteraatiota. Kun algoritmi muunnetaan Turingin koneeksi, esim. taulukon $table$ indeksointi ei enää toimi vakioajassa. Kuitenkin on selvää, että Turingin koneella iteraation kukin yksittäinen askel voidaan suorittaa ajassa $O(n^k)$ jollain (melko pienellä) $k \in \mathbb{N}$. Saamme siis kielelle L Turingin koneen, jonka aikavaativuus on $O(n^{k+3})$. □

Yleisemmin voimme todeta, että ongelma kuuluu luokkaan P, jos se voidaan esittää formaalina kielenä ja sillä on ratkaisualgoritmi, joka toimii ajassa $O(n^k)$ normaalien (esim. kurssilla *Tietorakenteet* esitettyjen) laskuperusteiden mukaan.

Epädeterministinen polynominen aikavaativuus

[Sipser luku 7.3]

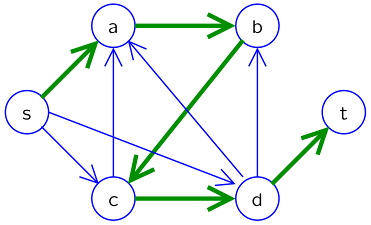
Määritelmä 5.11: Kieli kuuluu luokkaan NP (nondeterministic polynomial time), jos se voidaan tunnistaa *epädeterministisellä* Turingin koneella, jonka aikavaativuus on $O(n^k)$ jollain $k \in \mathbb{N}$. □

Lauseen 5.7 mukaan jos $A \in$ NP, niin A voidaan ratkaista deterministisessä ajassa $2^{O(n^k)}$ jollain $k \in \mathbb{N}$. Mitään oleellisesti tämän tarkempaa ei pystytä nykyisillä tiedoilla päättelemään. Luokka NP on kuitenkin käytännössä tärkeä, koska se sisältää monia tärkeitä ongelmia, joiden vaativuutta ei tiedetä sen tarkemmin. Tämäkin tieto antaa vihjeitä siitä, millaisilla tekniikoilla ongelmaa kannattaa tai ei kannata yrittää ratkaista.

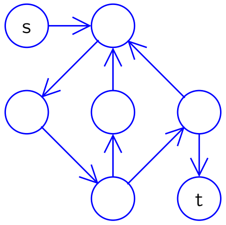
Esimerkki 5.12: Verkon *Hamiltonin polku* on polku, joka vierailee jokaisessa solmussa tasan kerran. Tarkastelemme versiota, jossa verkko on suunnattu ja polun alku- ja loppusolmu annettu:

$$HAMPATH = \{\, \langle G, s, t \rangle \mid \text{suunnatussa verkossa } G \text{ on Hamiltonin polku } s \rightsquigarrow t \,\}.$$

$HAMPATH$ on perusesimerkki ongelmasta, joka kuuluu luokkaan NP, mutta jonka kuuluminen luokkaan P on avoin ongelma.



Verkossa on Hamiltonin polku (s,a,b,c,d,t)



Verkossa ei ole Hamiltonin polkua

Kieli $HAMPATH$ voidaan tunnistaa epädeterministisellä kaksinauhaisella Turingin koneella, joka ensin arvaa solmujonon $p_1, \ldots, p_{|V|}$ ja sitten tarkistaa, syntyikö Hamiltonin polku:

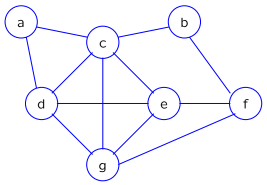
1. Etsi syötteestä $\langle G, s, t \rangle$ verkon G solmujen lukumäärä $|V|$.
2. Kirjoita kakkosnauhalle jono $p_1, \ldots, p_{|V|}$, jossa on $|V|$ lukua. Kukin luku p_i valitaan epädeterministisesti joukosta $\{\, 1, \ldots, |V| \,\}$.
3. Jos p_1 ei ole solmun s järjestysnumero, niin hylkää.
   Jos $p_{|V|}$ ei ole solmun t järjestysnumero, niin hylkää.
   Jos $p_i = p_j$ joillakin $i \neq j$, niin hylkää.
4. Kaikilla $i = 1, \ldots, |V| - 1$, tarkista, onko verkossa G kaari solmusta numero p_i solmuun numero p_{i+1}. Jos ei ole, niin hylkää.
5. Hyväksy.

Tässä siis on ajateltu, että verkon G koodaus samalla kiinnittää jonkin numeroinnin sen solmuille. Kohtien 3–5 tarkastusten yksityiskohdat riippuvat koodauksen yksityiskohdista, mutta millä tahansa järkevällä koodauksella tarkastukset voidaan selvästi tehdä polynomisessa ajassa.

Edellisen perusteella $HAMPATH \in$ NP.

Sen sijaan ei tiedetä, päteekö $HAMPATH \in$ P; melko yleisesti arvellaan, että *ei* päde. Koska mahdollisia polkuja $s \rightsquigarrow t$ voi olla verkon koon $|V|$ suhteen eksponentiaalinen määrä, mikään raakaan voimaan perustuva etsintä ei takaa polynomista aikavaativuutta. Lauseen 5.7 perusteella (tai analysoimalla jotain raakaan voimaan perustuvaa algoritmia) tiedämme kuitenkin, että $HAMPATH \in \mathrm{TIME}(2^{n^k})$ jollain $k \in \mathbb{N}$. □

Esimerkki 5.13: Suuntaamattoman verkon *klikki* on solmujoukko, jonka jokaista solmuparia yhdistää kaari. Allaolevassa verkossa on neljän solmun klikki $\{\, \mathrm{c, d, e, g} \,\}$ ja useita pienempiä klikkejä, esim. $\{\, \mathrm{c, d, e} \,\}$; $\{\, \mathrm{a, c, d} \,\}$; $\{\, \mathrm{b, f} \,\}$ ja $\{\, \mathrm{b} \,\}$.



Perusongelma on löytää verkosta mahdollisimman suuri klikki. Määrittelemme vastaavan formaalin kielen:

$$CLIQUE = \{\, \langle G, k \rangle \mid \text{suuntaamattomassa verkossa } G \text{ on } k \text{ solmua käsittävä klikki} \,\}.$$

$CLIQUE$ on vaativuudeltaan hyvin samansukuinen ongelma kuin $HAMPATH$.

Kielelle $CLIQUE$ saadaan samantapaisella arvaa ja testaa -algoritmi kuin Hamiltonin poluille:

1. Etsi syötteestä $\langle G, k \rangle$ haluttu klikin koko k.
2. Valitse epädeterministisesti solmujoukko $C \subseteq V$, jossa on k solmua.
3. Tarkista kaikki parit $(u, v) \in C \times C$. Jos kaikilla näistä $(u, v) \in E$, niin hyväksy. Muuten hylkää.

Johtopäätökset ovat kuten aiemmin:

- $CLIQUE \in \mathsf{NP}$
- $CLIQUE \in \mathsf{TIME}(2^{n^k})$ jollain $k \in \mathbb{N}$
- ei tiedetä, kuuluuko $CLIQUE$ luokkaan P. □

Esimerkki 5.14: Toteutuvuusongelmassa (satisfiability) on annettu propositiologiikan kaava, joka siis koostuu totuusarvoisista muuttujista $x_1, x_2, x_3, \ldots$ ja loogisista operaattoreista $\neg$ (negaatio "ei"), $\wedge$ (konjunktio "ja") sekä $\vee$ (disjunktio "tai"). Kaava on *toteutuva*, jos sen muuttujille voidaan valita sellaiset totuusarvot, että kaava tulee todeksi. Esim. kaava

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

on toteutuva, kuten nähdään vaikkapa valitsemalla $x_1 =$ False, $x_2 =$ True, $x_3 =$ False ja $x_4 =$ True. Sen sijaan kaava

$$(x_1 \vee (\neg x_2)) \wedge (x_2 \vee (\neg x_1)) \wedge (x_1 \vee x_2) \wedge ((\neg x_1) \vee (\neg x_2))$$

ei ole toteutuva, sillä mitkään muuttujien x_1 ja x_2 arvot eivät samanaikaisesti toteuta kaikkia neljää ehtoa $x_1 \vee (\neg x_2)$, $x_2 \vee (\neg x_1)$, $x_1 \vee x_2$ ja $((\neg x_1) \vee (\neg x_2))$. Toteutuvuusongelmaa vastaava formaali kieli

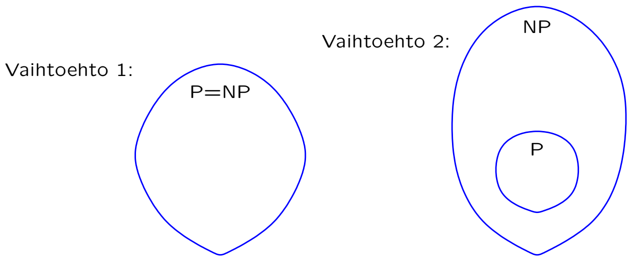
$$SAT = \{ \langle \varphi \rangle \mid \text{propositiologiikan kaava } \varphi \text{ on toteutuva} \}$$

kuuluu luokkaan NP, mikä taas nähdään arvaa ja testaa -algoritmilla. □

P vs. NP -ongelma [Sipser luku 7.4]

Koska selvästi $\mathsf{P} \subseteq \mathsf{NP}$, niin jompi kumpi seuraavista pätee:

- $\mathsf{P} = \mathsf{NP}$
- luokka P on luokan NP aito osajoukko.



Kysymys, päteekö $\mathsf{P} = \mathsf{NP}$, on ollut tietojenkäsittelyteorian keskeinen avoin ongelma 1970-luvun alusta lähtien. Se on yksi Millennium Prize -ongelmista, matematiikan suurista avoimista ongelmista, joiden ratkaisusta on luvattu miljoonan dollarin palkinto.

Jos pätee $\mathsf{P} = \mathsf{NP}$, niin Hamiltonin polut, klikkiongelma ja toteutuvuusongelma, sekä satoja muita tunnettuja luokan NP ongelmia, voidaan ratkaista polynomisessa ajassa. Koska lukuisista yrityksistä huolimatta millekään näistä ongelmista ei ole keksitty polynomisessa ajassa toimivaa algoritmia, arvellaan yleisesti, että pätee $\mathsf{P} \neq \mathsf{NP}$, mutta kenelläkään ei ole hyvää ideaa, miten tämä voitaisiin *todistaa*.

Monilla ongelmilla, mukaan lukien $HAMPATH$, $CLIQUE$ ja SAT, on vieläkin vahvempi kytkös P vs. NP -ongelmaan. Ensimmäisenä tämä kytkös osoitettiin SAT-ongelmalle.

Lause 5.15 (Cook-Levin): [Sipser Thm. 7.27] $SAT \in \mathrm{P}$, jos ja vain jos $\mathrm{P} = \mathrm{NP}$. □

Tämä lause siis sanoo, että tasan toinen seuraavista pätee:

- $SAT \in \mathrm{P}$ ja $\mathrm{P} = \mathrm{NP}$
- $SAT \notin \mathrm{P}$ ja $\mathrm{P} \neq \mathrm{NP}$.

"Jossittelu" lauseessa ei tarkoita, että esim. väittämällä $SAT \in \mathrm{P}$ voisi olla useita totuusarvoja. Joko se on tosi tai se on epätosi, emme vain (ainakaan toistaiseksi) tiedä, kumpi.

Jos siis löytäisimme polynomisessa ajassa toimivan algoritmin SAT-ongelmalle, niin pätisi $\mathrm{P} = \mathrm{NP}$ ja kaikki muutkin luokan NP ongelmat voitaisiin ratkaista polynomisessa ajassa. Toisaalta, jos pystymme osoittamaan, että $\mathrm{P} \neq \mathrm{NP}$, se tarkoittaa, että SAT-ongelmalle ei ole olemassa polynomisessa ajassa toimivaa algoritmia.

Emme esitä Cookin-Levinin lauseen todistusta, mutta selvitämme hieman siihen keskeisesti liittyvää NP-*täydellisyyden* käsitettä.

Tarvitsemme ensin pari teknistä apumääritelmää.

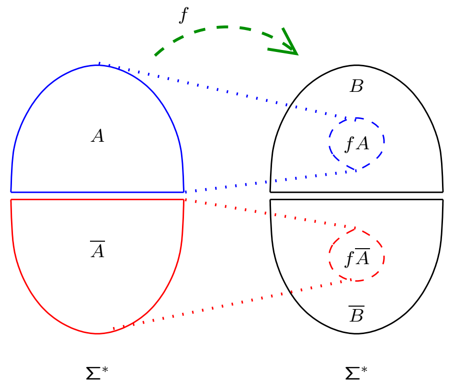
Määritelmä 5.16: Turingin kone M *laskee* funktion $f\colon \Sigma^* \to \Sigma^*$, jos millä tahansa syötteellä w kone M pysähtyy tilanteeseen, jossa nauhan sisältö on $f(w)$.

Funktio f on laskettavissa *polynomisessa ajassa*, jos jokin polynomisessa ajassa toimiva Turingin kone laskee sen. □

Määritelmä 5.17: Olkoot A ja B aakkoston Σ formaaleja kieliä. Sanomme, että kieli A *palautuu polynomisesti* kieleen B, ja kirjoitamme $A \leq_P B$, jos jollain polynomisessa ajassa laskettavalla funktiolla f on ominaisuus

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

Tällöin f on *polynominen palautus* kielestä A kieleen B. □



Kielen A polynominen palautus f kieleen B.

Edellisten määritelmien keskeinen sovellus on seuraava:

Lause 5.18: [Sipser Thm. 7.31] Jos $A \leq_P B$ ja $B \in \mathrm{P}$, niin $A \in \mathrm{P}$.

Todistus: Olkoon f funktio, jolla $w \in A \Leftrightarrow f(w) \in B$, kuten palautuksen määritelmässä. Nyt kieli A voidaan tunnistaa seuraavalla algoritmilla:

1. Syötteellä w laske ensin $z = f(w)$.
2. Jos $z \in B$, niin hyväksy. Jos $z \notin B$, niin hylkää.

Funktiota f koskevasta oletuksesta seuraa, että algoritmi hyväksyy, jos $w \in A$, ja hylkää, jos $w \notin A$.

Jos f on polynominen palautus, eli lisäksi laskettavissa polynomisessa ajassa, niin algoritmin askel 1 voidaan tehdä polynomisessa ajassa.

Jos lisäksi $B \in \mathrm{P}$, niin myös askel 2 voidaan tehdä polynomisessa ajassa. □

Määritelmä 5.19: Kieli B on NP-*täydellinen* (NP-complete), jos $B \in$ NP ja kaikilla $A \in$ NP pätee $A \leq_P B$. □

Edellä mainitut ongelmat $HAMPATH$, $CLIQUE$ ja SAT voidaan kaikki osoittaa NP-täydellisiksi.

Lause 5.20: [Sipser Thm. 7.35] Jos B on NP-täydellinen ja $B \in$ P, niin P = NP.

Todistus: Seuraa suoraan määritelmästä ja lauseesta 5.18. □

Intuitiivisesti NP-täydelliset ongelmat ovat luokan NP vaikeimpia mahdollisia ongelmia: Jos yhdellekin sellaiselle löytyisi polynomisessa ajassa toimiva algoritmi, tästä seuraisi heti P = NP. Kääntäen jos P $\neq$ NP, niin erityisesti NP-täydelliset ongelmat jäävät luokan P ulkopuolelle.

Cookin-Levinin lause seuraa nyt seuraavasta tuloksesta:

Lause 5.21: [Sipser Thm. 7.37] SAT on NP-täydellinen.

Todistuksen perusidea: Olkoon $A \in$ NP. Siis $A = L(N)$ jollain polynomisessa ajassa toimivalla epädeterministisellä Turingin koneella N. Koneen N rakenteen perusteella muodostetaan polynomisessa ajassa laskettava funktio f_N. Funktion arvo $f_N(w)$ on propositiokaava, joka simuloi koneen N laskentaa syötteellä w. Jos koneella on hyväksyvä laskenta, niin $f_N(w)$ on toteutuva, muuten ei. Kaavan $f_N(w)$ muodostamisen yksityiskohdat ovat jossain määrin teknisiä. □

Kun SAT on todistettu NP-täydelliseksi, voidaan tämän avulla todistaa (hieman) kevyemmällä tekniikalla monien muiden ongelmien NP-täydellisyys. NP-täydellisiä ongelmia tunnetaan satoja (ellei tuhansia). Ne liittyvät verkkoihin, logiikkaan, pakkausongelmiin ja lukuisiin muihin yleisiin ja erityisiin sovelluksiin.

NP-täydelliset ongelmat ovat tehokkaan laskettavuuden rajamailla, joten niiden algoritmiikka on haastavaa. Asiaa käsitellään mm. kursseilla *Algoritmien suunnittelu ja analyysi*, *Diskreetti optimointi*.

Lopuksi

Kurssin sisältöä voidaan tarkastella kahdesta näkökulmasta:

1. Perustiedot formaaleista kielistä ja niiden tunnistamisesta; esim.

- kielen määritteleminen äärellisen automaatin, säännöllisen lausekkeen tai yhteydettömän kieliopin avulla,
- em. formalismien väliset yhteydet,
- Turingin kone yleisenä algoritmin mallina ja
- laskennallisen "vaikeuden" perustulokset (ratkeamattomuus, NP-täydellisyys)

2. Johdatus tietojenkäsittelyteoriaan ja sen metodiikkaan; erityisesti

- matematiikan soveltaminen laskennan mallintamiseen ja
- miten väitteet perustellaan täsmällisesti.

Käydään lyhyesti läpi kurssin sisältöä tältä kannalta.

Säännölliset kielet

Käytännön ohjelmoinnissa hyödyllistä:

- tilasiirtymäkone laskennan mallina
- säännölliset lausekkeet ja äärelliset automaatit

Teoreettisia ajatusmalleja:

- epädeterministinen laskenta
- mallien väliset konversiot (esim. NFA → DFA)
- laskulaitteen ja kuvausformalismin ekvivalenssi (DFA vs. säännöllinen lauseke)
- luokan sulkeumaominaisuudet
- mahdottomuustodistukset (pumppauslemma)

Yhteydettömät kielet

Käytännön ohjelmoinnissa hyödyllistä:

- kielen kuvaaminen kieliopilla
- jäsentämisen peruskäsitteet, erityisesti jäsennyspuu

Teoreettisia ajatusmalleja: samat kuviot kuin säännöllisillä kielillä, teknisesti haastavammassa tilanteessa

Algoritmisia tekniikoita:

- iteratiiviset algoritmit (nollautuvat muuttujat jne.)
- CYK-algoritmi ja taulukointi

Laskettavuus ja laskennan vaativuus

Tärkeitä peruskäsitteitä:

- Churchin-Turingin teesi; universaali Turingin kone
- ratkeamattomat ongelmat
- polynominen aikavaativuus
- NP-täydelliset ongelmat

Todistustekniikoita yms.:

- ratkeamattomuustodistukset
- laskennan aikavaativuuden mallintaminen
- polynomisesti ekvivalentit laskennan mallit (esim. yksi- vs. moninauhainen Turingin kone)
- epädeterministinen aikavaativuus; polynominen palautus.

Muista antaa kurssipalaute!

— Loppu —