Tuntuu luontevalta vaatia, että algoritmissa

- yksi operaatio saa tehdä vain äärellisen määrän työtä.

Siis erityisesti yksi operaatio saa

- lukea äärellisen määrän tietoa,
- etsiä toimintaohjeen äärellisestä sääntöjoukosta ja
- kirjoittaa äärellisen määrän tietoa.

Toisaalta ei ole mitään syytä olla sallimatta, että

- algoritmin käytössä on rajattomasti apumuistia.

Tämän esittäminen matemaattisesti johtaa suoraan ajatukseen, että matemaattiselta kannalta

algoritmi on sama asia kuin *Turingin kone*.

Tämä on (karkeasti) alkuperäinen ajatus Turingin koneen takana.

Turingin kone ei syntynyt sattumalta.

Matematiikan perusteiden tutkimuksessa 1900-luvun alussa keskeistä oli matematiikan mekanisoiminen (voiko matemaatikon korvata algoritmilla). Tätä varten esitettiin useita formalisointeja mekaaniselle laskennalle:

- Turingin kone (Turing 1936)
- Postin kone (Post 1936)
- μ-rekursiiviset funktiot (Gödel, Kleene 1936)
- λ-kalkyyli (Church 1936)

Merkittävä havainto oli, että nämä kaikki olivat *ekvivalentteja*, ts. antavat tasan *saman* vastauksen kysymykseen "Mitä ongelmia voidaan ratkaista algoritmisesti?" Tämä johti *Churchin-Turingin teesinä* tunnettuun väittämään:

> Ongelma voidaan ratkaista algoritmilla, jos ja vain jos se voidaan ratkaista Turingin koneella.

Churchin-Turingin teesi ei tietenkään ole matemaattinen väittämä. Sitä voi (kenties) pitää luonnontieteellisenä väittämänä, jonka voisi periaatteessa falsifioida rakentamalla Turingin konetta voimakkaamman laskulaitteen.

Käytännössä Churchin-Turingin teesi lähinnä sanoo, että olemme tyytyväisiä Turingin koneeseen algoritmin määritelmänä. Kuten edellä ilmeni, tälle tyytyväisyydelle on tiettyjä filosofisia perusteita. Myöskään sellaiset modernit laskennan mallit kuin kvanttitietokone eivät näyttäisi uhkaavan tätä teesiä.

Käytännön tietojenkäsittelyn kannalta tärkeä havainto on, että myös (idealisoitu) nykyaikainen tietokone on *Turing-ekvivalentti* eli pystyy ratkaisemaan tasan samat ongelmat kuin Turingin kone. Tarkastellaan tätä hieman lähemmin.

On selvää, että tietokone on ainakin yhtä laskentavoimainen kuin Turingin kone: on helppoa kirjoittaa Turingin kone -simulaattori esim. C-kielellä. Tietokoneen simuloiminen Turingin koneella vaatii hieman enemmän työtä.

Ajatellaan, että tietokonessa on k rekisteriä ja lisäksi rajoittamaton määrä hajasaantimuistia (RAM).

Simuloidaan tietokonetta $k+2$-nauhaisella Turingin koneella. Nauhoille $1,\ldots,k$ talletetaan rekisterien $1,\ldots,k$ sisällöt. Nauhalla $k+1$ esitetään tietokoneen muistin sisältö muodossa

$$\texttt{\#}a_1\texttt{\#}d_1\texttt{\#\#}a_2\texttt{\#}d_2\texttt{\#\#}\ldots\texttt{\#\#}a_n\texttt{\#}d_n\texttt{\#}\llcorner\!\lrcorner,$$

missä n on käytössä olevan muistin määrä ja d_i on muistipaikan a_i sisältö (esim. binäärikoodattuna). Nauha $k+2$ toimii apumuistina.

Aiempien esimerkkien perusteella pitäisi olla uskottavaa, että Turingin kone pystyy simuloimaan konekäskyjä kuten

- `lataa rekisterin 2 osoittaman muistipaikan sisältö rekisteriin 3`,
- `lisää rekisterin 1 sisältöön rekisterin 2 sisältö` jne.

Tosin muistiin kirjoitettaessa voidaan joutua siirtämään oikealle tai vasemmalle pitkä pätkä nauhan $k+1$ sisältöä.

Turingin koneiden esittäminen

Olemme edellä esittäneet Turingin koneita eri tarkkuustasoilla:

formaali kuvaus: tarkka tilakaavio

toteutustaso: selitetään nauhojen käyttö ja muut pääideat, mutta ei puututa koneen yksittäisiin tiloihin jne.

korkea taso: esitetään algoritmi sellaisenaan (esim. pseudokoodina) viittaamatta erityisesti Turingin kone -malliin.

Tässä vaiheessa pitäisi olla uskottavaa, että mikä tahansa algoritmi osataan kyllä tarvittaessa "koodata Turingin koneen konekielelle". Kuten edellä on todettu, tämä on motivaatio sille, että Turingin koneet ylipäänsä ovat kiinnostava laskennan malli.

Jatkossa siis tyydymme yleensä esittämään Turingin koneet melko korkealla tasolla.

4. Ratkeavuus

Tarkastelemme formaaleihin kieliin liittyviä algoritmeja käyttäen Turingin konetta algoritmin täsmällisenä määritelmänä.

Tämän luvun jälkeen opiskelija

1. osaa kuvailla *universaaliin Turingin koneeseen* liittyvät peruskäsitteet
2. tuntee *universaalikielen ratkeamattomuuteen* liittyvät peruskonstruktiot.

Yleisemmällä tasolla tavoittena on ymmärtää, millaiset ongelmat ovat ratkeamattomia ja mitä tämä tarkoittaa.

Pysähtymisongelman ratkeavuus [Sipser luku 4.2]

Osoitamme keskeisen tuloksen, että

- jotkin Turing-tunnistettavat kielet ovat ratkeamattomia ja
- jotkin kielet eivät ole edes Turing-tunnistettavia.

Lisäksi toteamme, että ratkeamattomat ongelmat eivät välttämättä ole mitenkään "omituisia", vaan myös monet luonnolliset ja käytännössä kiinnostavat ongelmat ovat ratkeamattomia.

Perusesimerkki ratkeamattomasta ongelmasta on (sopivasti formuloitu) pysähtymisongelma

Annettu: (esim. C-kielinen) ohjelma P, syöte w
Kysymys: pysähtyykö P syötteellä w.

Jos tunnetaan hieman joukko-oppia, niin ei-tunnistettavien kielten *olemassaolo* voidaan todeta seuraavantapaisella päättelyllä:

- Turingin koneiden joukko on *numeroituva*, ts. Turingin koneet voidaan asettaa yksi-yhteen-vastaavuuteen luonnollisten lukujen kanssa.
- Kaikkien formaalien kielten joukko $\mathcal{P}(\Sigma^*)$ (missä tahansa kiinteässä aakkostossa Σ) on *ylinumeroituva*; itse asiassa formaalit kielet voidaan asettaa yksi-yhteen-vastaavuuteen *reaalilukujen* kanssa.
- Siis Turingin koneita on "vähemmän" kuin formaaleja kieliä. Koska jokainen Turingin kone tunnistaa täsmälleen yhden kielen, kaikille kielille ei mitenkään voi olla omaa tunnistavaa Turingin konettaan.

Emme rupea täsmentämään tätä päättelyä. Johdamme tarkemman tuloksen, joka antaa *konkreettisia esimerkkejä* ratkeamattomista (ja ei-tunnistettavista) kielistä.

Kuten edellä todettiin, ongelma

Annettu: C-kielinen ohjelma P, syöte w
Kysymys: pysähtyykö P syötteellä w

on tyyppiesimerkki käytännön ohjelmointiongelmasta, jolle ei ole ratkaisualgoritmia. Tämän perustelemiseksi osoitamme, että vastaava Turingin koneita koskeva ongelma

Annettu: Turingin kone M, syöte w
Kysymys: pysähtyykö M syötteellä w

ei ole Turing-ratkeava. Koska Turingin koneet ovat yhtä ilmaisuvoimaisia kuin C-ohjelmat, kyseessä on oleellisesti sama ongelma, mutta Turingin koneita on huomattavasti yksikertaisempi käsitellä.

Ensin kuitenkin on täsmennettävä, mitä tarkoitamme sillä, että Turingin kone saa syötteenä toisen Turingin koneen. Koska Turingin koneen syötteet ovat merkkijonoja, meidän pitää *koodata* Turingin koneet merkkijonoiksi sopivassa aakkostossa.

Koodaus voidaan tehdä monella tavalla eivätkä yksityiskohdat oikeastaan ole oleellisia, mutta asian konkretisoimiseksi käymme läpi yhden mahdollisuuden.

Yksinkertaisuuden vuoksi oletetaan, että kaikkien jatkossa esiintyvien Turingin koneiden syöteaakkosto sisältää ainakin merkit 0, ..., 9 ja #. Kun on annettu Turingin kone $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, niin

- Numeroidaan tilat: $Q = \{ q_1, \dots, q_n \}$, missä $n = |Q|$.
- Valitaan tilan q_i koodiksi indeksin i tavallinen kymmenjärjestelmäesitys ja merkitään tätä koodia $\langle q_i \rangle$. Siis $\langle q_i \rangle$ on aakkoston $\{ 0, \dots, 9 \}$ merkkijono.

Numeroidaan ja koodataan samalla tavalla myös aakkostot Σ ja Γ ja siirtymäsuunnat L ja R. Siirtymäfunktiolle saadaan koodi $\langle \delta \rangle$ luettelemalla kaikki siirtymät $\delta(q, a) = (q', b, D)$ muodossa

$$\texttt{\#} \langle q \rangle \texttt{\#} \langle a \rangle \texttt{\#} \langle q' \rangle \texttt{\#} \langle b \rangle \texttt{\#} \langle D \rangle \texttt{\#}.$$

Koko koneen M koodi on nyt aakkoston $\{ 0, \dots, 9, \texttt{\#} \}$ merkkijono

$$\langle M \rangle = \texttt{\#} \langle \delta \rangle \texttt{\#\#} \langle q_0 \rangle \texttt{\#} \langle q_{\text{accept}} \rangle \texttt{\#} \langle q_{\text{reject}} \rangle \texttt{\#}.$$

Sovitaan lisäksi, että Turingin kone M yhdessä merkkijonon $w = w_1 \dots w_n$ kanssa koodataan muotoon $\langle M, w \rangle = \langle M \rangle \texttt{\#} \langle w_1 \rangle \texttt{\#} \dots \texttt{\#} \langle w_n \rangle$. Voimme nyt määritellä formaalin kielen

$$A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ on Turingin kone joka hyväksyy } w\text{:n} \},$$

jota kutsutaan *Turingin koneen hyväksymisongelmaksi* ja myös *universaalikieleksi*. Se siis sisältää yhteen kieleen pakattuna tiedon kaikkien Turingin koneiden tunnistamista kielistä.

Lause 4.1: [Sipser s. 176] Kieli A_{TM} on Turing-tunnistettava.

Todistus: Kieli A_{TM} voidaan tunnistaa Turingin koneella U, joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Tarkasta, että syöte todella on muotoa $\langle M, w \rangle$; jos ei ole, niin *hylkää*.
2. Simuloi konetta M syötteellä w.
3. Jos M menee tilaan q_{accept}, niin *hyväksy*.
   Jos M menee tilaan q_{reject}, niin *hylkää*.

Simulaation yksityiskohdat eivät ole oleellisia, mutta hahmottelemme erään mahdollisuuden lyhyesti seuraavalla sivulla.

Kun syötteenä on saatu koodi

$$\langle M, w \rangle = \texttt{\#} \langle \delta \rangle \texttt{\#\#} \langle q_0 \rangle \texttt{\#} \langle q_{\text{accept}} \rangle \texttt{\#} \langle q_{\text{reject}} \rangle \texttt{\#\#} \langle w_1 \rangle \texttt{\#} \dots \texttt{\#} \langle w_n \rangle,$$

voidaan simulointi hoitaa esim. kaksinauhaisella Turingin koneella:

nauha 1 sisältää syötteen, ja sen sisältöä ei muuteta
nauha 2 koodaa koneen M tilanteen $u_1 \dots u_n q v_1 \dots v_m$ muodossa

$$\texttt{\#} \langle u_1 \rangle \texttt{\#} \dots \texttt{\#} \langle u_n \rangle \texttt{\#\#} \langle q \rangle \texttt{\#} \langle v_1 \rangle \texttt{\#} \dots \texttt{\#} \langle v_m \rangle \texttt{\#}.$$

Yhden laskenta-askelen simuloimiseksi tapahtuu seuraavasti:

- Etsitään ensin nauhalta 2 merkkiparin "##" kohdalta koneen M tila q ja nauhapään alla oleva merkki v_1.
- Nauhalta 1 etsitään tätä paria (q, v_1) vastaava siirtymä $\delta(q, v_1) = (q', b, D)$.
- Päivitetään tilannetta nauhalla 2 vastaavasti. (Koska koodien pituudet vaihtelevat, tässä voidaan joutua siirtämään nauhan loppuosaa vasemmalle tai oikealle.)
- Uutta tilaa verrataan nauhaan 1, ja jos se on q_{accept} tai q_{reject}, niin simulaatio pysähtyy.

□

Edellä esitetyn kaltaista konetta U sanotaan *universaalikoneeksi*. Sille pätee

$$\begin{aligned} U \text{ hyväksyy syötteen } \langle M, w \rangle &\Leftrightarrow M \text{ hyväksyy syötteen } w \\ U \text{ hylkää syötteen } \langle M, w \rangle &\Leftrightarrow M \text{ hylkää syötteen } w \\ U \text{ jää silmukkaan syötteellä } \langle M, w \rangle &\Leftrightarrow M \text{ jää silmukkaan syötteellä } w. \end{aligned}$$

Koska U voi jäädä silmukkaan, tämä ei osoita kieltä A_{TM} ratkeavaksi. Itse asiassa hyväksymisongelma onkin ratkeamaton, kuten kohta näemme.

Kuvitellaan, että haluamme tosiaan rakentaa Turingin koneita formaalien kielten tunnistamiseksi. Universaalikoneen takia meidän ei tarvitse rakentaa erikseen konetta M jokaiselle kielelle $L(M)$.

Riittää rakentaa yksi universaalikone U. Mikä tahansa kieli $L(M)$ voidaan tunnistaa universaalikoneella U antamalla sille *ohjelmaksi* koneen M koodi $\langle M \rangle$. Siis universaali Turingin kone on malli ohjelmoitavalle yleiskäyttöiselle tietokoneelle.

Todistamme nyt, että Turingin koneen hyväksymisongelma ei ole ratkeava. Keskeisenä aputuloksena todetaan ensin [Sipser s. 181–183]

Lemma 4.2: Diagonaalikieli

$$D = \{ \langle M \rangle \mid \langle M \rangle \notin L(M) \}$$

ei ole Turing-tunnistettava.

Huom. Tarkastelemme siis, mitä tapahtuu, kun jokin kone M saa syötteenä oman koodinsa $\langle M \rangle$. Tämä saattaa tuntua oudolta, mutta siinä ei ole mitään sen kummempaa kuin siinä, että voimme antaa C-kielellä kirjoitetulle ohjelmalle syötetiedostoksi tekstitiedoston, jossa sattuu olemaan kyseisen ohjelman lähdekoodi.

Todistus: Halutaan siis osoittaa, että kieli

$$D = \{ \langle M \rangle \mid \langle M \rangle \notin L(M) \}$$

ei ole Turing-tunnistettava.

Tehdään vastaoletus, että D on Turing-tunnistettava. Tällöin $D = L(M)$ jollain M. Nyt saadaan ristiriita tarkastelemalla merkkijonoa $\langle M \rangle$:

$$\begin{aligned} \langle M \rangle \in L(M) &\Leftrightarrow \langle M \rangle \in D && M\text{:n valinta} \\ &\Leftrightarrow \langle M \rangle \notin L(M) && D\text{:n määritelmä.} \end{aligned}$$

□

Jokainen kone M joko hyväksyy oman koodinsa $\langle M \rangle$ tai ei hyväksy. Kumpikin vaihtoehto on väärin kielen D tunnistavalle koneelle.

Nyt saamme haluamamme keskeisen ratkeamattomuustuloksen:

Lause 4.3: [Sipser Thm. 4.11] Kieli A_{TM} ei ole ratkeava.

Todistus: Tehdään vastaoletus, että jokin ratkaisija R tunnistaa kielen A_{TM}. Diagonaalikieli $D = \{ \langle M \rangle \mid \langle M \rangle \notin L(M) \}$ voidaan esittää muodossa

$$D = \{ \langle M \rangle \mid \langle M, \langle M \rangle \rangle \notin A_{\mathsf{TM}} \}.$$

Käyttämällä apuna konetta R kieli D voidaan tunnistaa (ja peräti ratkaista) Turingin koneella, joka syötteellä w toimii seuraavasti:

1. Jos w ei ole $\langle M \rangle$ millään M, niin *hylkää*.
2. Muodosta $\langle M, \langle M \rangle \rangle$, missä $w = \langle M \rangle$.
3. Simuloi konetta R syötteellä $\langle M, \langle M \rangle \rangle$. Jos R hyväksyi, niin *hylkää*. Jos R hylkäsi, niin *hyväksy*.

Mutta D ei ole Turing-tunnistettava; ristiriita. □

Kieleen A_{TM} liittyy läheisesti *Turingin koneen pysähtymisongelma*

$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ on Turingin kone joka pysähtyy syötteellä } w.\}$$

Korollaari 4.4: [Sipser Thm. 5.1] Kieli $HALT_{\mathsf{TM}}$ on ratkeamaton.

Todistus: Tehdään vastaoletus, että $HALT_{\mathsf{TM}}$ olisi ratkeava. Kieli A_{TM} voitaisiin ratkaista Turingin koneella, joka syötteellä $\langle M, w\rangle$ toimii seuraavasti:

1. Jos $\langle M, w\rangle \notin HALT_{\mathsf{TM}}$, niin *hylkää*.
2. Simuloi konetta M syötteellä w. Jos M hyväksyi, niin *hyväksy*. Jos M hylkäsi, niin *hylkää*.

Vastaoletuksen nojalla kohdan 1 ehto voidaan ratkaista, ja kohtaan 2 mennessä tiedetään, että simulaatio ei jää silmukkaan. Mutta A_{TM} tiedetään ratkeamattomaksi; ristiriita. □

Churchin-Turingin teesin ja erilaisten laskentaformalismien ekvivalenssin nojalla tulkitsemme tämän niin, että ei ole olemassa mitään algoritmia, joka ratkaisisi annetusta tietokoneohjelmasta, pysähtyykö se annetulla syötteellä.

Tietysti yksittäisestä ohjelmasta voi olla helppoakin nähdä, millä syötteillä se pysähtyy. Tulos kertoo, että mille tahansa yleisen pysähtymisongelman ratkaisemiseen tähtäävälle algoritmille voidaan esittää syöte, jolla algoritmi ei toimi (vaan antaa väärän vastauksen tai menee silmukkaan).

Todetaan vielä, että "hyväksymättömyysongelma" $\overline{A_{\mathsf{TM}}}$ ei ole edes Turing-tunnistettava. Todistetaan ensin aputulos

Lause 4.5: [Sipser Thm. 4.22] Jos A ja $\overline{A}$ ovat tunnistettavia, niin ne ovat ratkeavia.

Todistus: Oletetaan, että Turingin koneet M_1 ja M_2 tunnistavat kielet A ja $\overline{A}$. Kieli A voidaan ratkaista koneella, joka syötteellä w toimii seuraavasti:

1. Simuloi koneita M_1 ja M_2 syötteellä w rinnakkain, kumpaakin aina yksi askel kerrallaan, kunnes toinen pysähtyy.
2. Jos M_1 hyväksyy tai M_2 hylkää, niin *hyväksy*.
3. Jos M_1 hylkää tai M_2 hyväksyy, niin *hylkää*.

Koska w kuuluu jompaan kumpaan kielistä A ja $\overline{A}$, ainakin toinen simulaatio varmasti pysähtyy. □

Korollaari 4.6: [Sipser Cor. 4.23] Kieli $\overline{A_{\mathsf{TM}}}$ ei ole Turing-tunnistettava.

Todistus: Tiedämme, että A_{TM} on Turing-tunnistettava. Jos myös $\overline{A_{\mathsf{TM}}}$ olisi Turing-tunnistettava, niin A_{TM} olisi ratkeava. □

On myös runsaasti esimerkkejä, joissa sen enempää kieli A kuin sen komplementti $\overline{A}$ eivät ole tunnistettavia. Sivun 260 numeroituvuusargumentin perusteella itse asiassa tällaisia kieliä on ylinumeroituvasti.

Konkreettisia esimerkkejäkin on melko helppo löytää, esim. Turingin koneiden totaalisuusongelma

$$TOTAL_{\mathsf{TM}} = \{\langle M\rangle \mid \text{Turingin kone } M \text{ pysähtyy kaikilla syötteillä}\}$$

ja Turingin koneiden ekvivalenssiongelma

$$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid \text{Turingin koneet } M_1 \text{ ja } M_2 \text{ tunnistavat saman kielen}\}.$$

Emme tässä rupea kehittämään ratkeamattomuuden teoriaa tämän pidemmälle.