TAMPEREEN YLIOPISTO
Pro gradu -tutkielma

Kerttu Moilanen

Laskettavuudesta

Matematiikan, tilastotieteen ja filosofian laitos
Matematiikka
Toukokuu 2009

Tampereen yliopisto

Matematiikan ja tilastotieteen laitos

MOILANEN, KERTTU: Laskettavuudesta

Pro Gradu -tutkielma, 55 s.

Matematiikka

Toukokuu 2009

Tiivistelmä

Tässä tutkielmassa käsitellään laskettavuuden historiaa ja joitain perustuloksia. Laskettavuuden teoria perustuu matemaattisen logiikkaan, mutta 1930–luvulla siitä alkoi syntyä oma tieteenalansa laskennan mekanisointia mietittäessä. Eräs malli laskennan mekanisoinnista on tutkielmassakin käsiteltävä äärellinen automaatti, ja niihin liittyvät säännölliset kielet ja säännölliset lausekkeet. Äärellinen automaatti ei kuitenkaan ole tarkka algoritmi, koska sillä ei voida ratkaista kaikkia ratkaistavia ongelmia. Kielet ovat merkkijonojoukkoja, jotka voidaan tunnistaa automaatilla.

Turing-tunnistettavat ja ratkaistavat kielet liittyvät Turingin koneeseen. Turingin kone on eräs tarkan algoritmin määritelmä, eli kaikki ratkaistavat ongelmat voidaan ratkaista Turingin koneella. Kaikkia ongelmia ei kuitenkaan voida ratkaista ja ne ovat silloin ratkeamattomia. Kuuluisimpia niistä lienee universaalikielen ratkeamattomuus ja pysähtymisongelman ratkeamattomuus. Universaalikieli on kieli, joka sisältää tiedon kaikista binääriaakkoston tunnistettavista Turingin koneista. Pysähtymisongelmassa pyritään ratkaisemaan, pysähtyykö syötteenä annettu Turingin kone annetulla merkkijono syötteellä. Pysähtymisongelmasta ja yleensäkin ratkeavuudesta päästäisiin laskennanvaativuusteoriaan. Tässä tutkielmassa ei kuitenkaan käsitelty laskennanvaativuusteoriaa.

Asiasanat:laskettavuus, laskettavuuden teoria, Turingin kone, universaalikone, pysähtymisongelma, ratkeavuus

Sisältö

1 Johdanto **1**

2 Historiaa **3**
2.1 Aristoteles . . . 3
2.2 G. W. Leibniz . . . 3
2.3 George Boole . . . 5
2.4 Gottlob Frege . . . 7
2.5 Georg Cantor . . . 8
2.6 David Hilbert . . . 9
2.7 Kurt Gödel . . . 11
2.8 Alan Turing . . . 12

3 Esitietoja **14**
3.1 Peruskäsitteitä . . . 14
3.2 Ongelmanratkaisusta . . . 16
3.3 Koodaus . . . 18
3.4 Graafeista . . . 19

4 Äärellinen automaatti **21**
4.1 Deterministinen äärellinen automaatti . . . 21
4.2 Epädeterministinen äärellinen automaatti . . . 25

5 Turingin kone **28**
5.1 Moninauhainen Turingin kone . . . 35
5.2 Epädeterministinen Turingin kone . . . 36
5.3 Ratkeavuus . . . 40
5.4 Algoritmi ja Churchin-Turingin teesi . . . 41
5.5 Ratkeavia kieliä . . . 43

6 Universaali Turingin kone **46**
6.1 Turingin koneiden koodaus . . . 46
6.2 Diagonaalikieli . . . 47
6.3 Universaali Turingin kone . . . 48

7 Ratkeamattomia ongelmia **51**
7.1 Hyväksymisongelma . . . 51
7.2 Pysähtymisongelma . . . 52

8 Yhteenveto **54**

Viitteet **56**

1 Johdanto

Tässä tutkielmassa käsitellään laskettavuuden historiaa ja perustuloksia. Laskettavuuden teoria perustuu matemaattisen logiikkaan, mutta 1930–luvulla alkoi syntyä oma tieteenalansa laskennan mekanisointia mietittäessä. Eräs malli laskennan mekanisoinnista on tutkielmassakin käsiteltävä äärellinen automaatti, ja niihin liittyvät säännölliset kielet ja säännölliset lausekkeet. Äärellinen automaatti ei kuitenkaan ole tarkka algoritmi, koska sillä ei voida ratkaista kaikkia ratkaistavia ongelmia. Kielet taas ovat merkkijonojoukkoja, jotka voidaan tunnistaa automaatilla.

Turing-tunnistettavat ja ratkaistavat kielet taas liittyvät Turingin koneeseen. Turingin kone on eräs tarkan algoritmin määritelmä, eli kaikki ratkaistavat ongelmat voidaan ratkaista Turingin koneella. Kaikkia ongelmia ei voida ratkaista ja ne ovat silloin ratkeamattomia. Kuuluisimpia niistä lienee universaalikielen ratkeamattomuus ja pysähtymisongelman ratkeamattomuus. Universaalikieli on kieli, joka sisältää tiedon kaikista binääriaakkoston tunnistettavista Turingin koneista. Pysähtymisongelmassa pyritään ratkaisemaan, pysähtyykö syötteenä annettu Turingin kone annetulla merkkijono syötteellä. Pysähtymisongelmasta ja yleensäkin ratkeavuudesta päästäisiin laskennanvaativuusteoriaan. Tässä tutkielmassa ei kuitenkaan käsitelty laskennanvaativuusteoriaa.

Esitietoina tässä työssä edellytetään joitain matematiikan perusopinnoissa opittuja asioita, muun muassa joukko-opin ja logiikan perusteista. Myös joukkojen mahtavuuteen liittyvät käsitteet ja numeroituvuuden ja ylinumeroituvuuden käsitteet olisi hyvä ymmärtää. Niissä tarvittava Cantorin diagonaalimenetelmän tunteminen helpottaa esimerkiksi kappaleessa 6 käsiteltävien asioiden ymmärtämistä. Monet tietotekniikan peruskursseilla opitut asiat kuten algoritmin ja graafin käsitteiden ymmärrys helpottavat tutkielman lukemista.

Lisäksi on hyvä tietää joitain merkinnällisiä asioita tästä tutkielmasta. Tutkielmassa käytetään monissa kohdissa lukua 1 tarkoitettaessa jonkin asian olevan tosi ja lukua 0 epätoden kohdalla. Lisäksi tässä työssä käytetään lyhennettä joss, kun tarkoitetaan lausahdusta ”jos ja vain jos”. Tässä työssä ohjelmalla tarkoitetaan Turingin konetta.

Tulevissa kappaleissa tutustutaan laskettavuuden historiaan luvussa 2, laskettavuuden peruskäsitteistöön luvussa 3, erilaisiin mekaanista laskentaa kuvaaviin malleihin luvuissa 4 ja 5, ja lisäksi esitellään joitain ratkeavuuden perustuloksia luvussa 6. Mekaaninen laskenta on jo sinällään kiinnostava ongelma matemaatikoille, ja siksi matemaatikot ja loogikot ovat miettineet sitä kautta aikojen, erityisesti 1930–luvulla. Nämä pohdinnat käynnistivät kehityskulun ensimmäisten tietokoneiden rakentamiseen.

Ensimmäisen luvun lähdeteoksena on käytetty Martin Davisin kirjaa *Tietokoneen esihistoria Leibnizista Turingiin* [2]. Muissa luvuissa päälähdeteok-

sena on käytetty Michael Sipserin kirjaa *Introduction to the theory of computation* [9], lukuun ottamatta erikseen merkittyjä viittauksia, esimerkkejä ja lukua 6. Luvun 6 pääasiallisena lähdeteoksena on käytetty Hopcroftin, Motwanin ja Ullmanin teosta *Introduction to automata theory, languages, and computation* [5]. Lisäksi joissain erikseen merkityissä kohdissa tutkielmaa on käytetty lähteenä Cutlandin teosta *Computability an introduction to recursive funktion theory* [1] ja Papadimitrioun kirjaa *Computational complexity* [7]. Tutkielmassa on käytetty tukimateriaalina, suomenkielisten termien lähteenä ja joissain merkinnällisissä asioissa internetistä löytyvää materiaalia. Nämä ovat Minna-Liisa Rantaniemen raportti *Lukujärjestelmät*, Helsingin yliopiston professorin Jyrki Kivisen kurssimoniste *Laskennan mallit* [6] ja Tampereen teknillisen yliopiston professorin Tapio Elomaan kurssimoniste*Johdatus tietojenkäsittelyteoriaan* [3].

2 Historiaa

On paljon matemaatikkoja, jotka vaikuttivat matematiikan kehitykseen siten, että lopulta Turingin universaalikoneen pohjalta rakennettiin ensimmäinen tietokone. Tässä luvussa on kerrottu joistain näistä merkittävistä henkilöistä.

2.1 Aristoteles

Aristoteles loi aikanaan logiikalle pohjan, jolle matemaatikot pystyivät myöhemmin rakentamaan laskettavuudeksi kutsutun matematiikan haaran, joka oli edellytys nykyaikaisten tietokoneiden kehittämisessä. Hänen kehittelemäänsä loogista päättelyrakennetta kutsutaan *syllogismiksi*. Syllogismit perustuvat siihen, että on olemassa kaksi premissiksi kutsuttua lausetta, joista voidaan johtaa johtopäätökseksi kutsuttu lause. Premissit ja johtopäätökset pitää pystyä esittämään jonkin alla olevan neljän perusmuodon avulla.

- Jokainen X on Y
- Mikään X ei ole Y
- Jokin X on Y
- Jokin X ei ole Y

Siis aristoteelinen logiikka eli syllogistiikka on yksipaikkaisten väitelauseiden logiikkaa. Jokainen syllogismi ei kuitenkaan ole pätevä, vaan on olemassa sääntöjä, jotka ohjaavat päättelyä.

Aristoteleen oivallusta, että mikään ei voi sekä kuulua että olla kuulumatta johonkin joukkoon kutsutaan *ristiriidan laiksi*. Siis ei voi olla yhtä aikaa $A = 1$ ja $A = 0$. Hän kutsui tätä lakia kaiken filosofian perusaksioomaksi ja se voidaan kirjoittaa muotoon $x(1 - x) = 0$.

Hän tutki myös äärettömyyksiä ja nimesi potentiaaliseksi äärettömyydeksi mahtavuuden, jota nykyään kutsutaan numeroituvasti äärettömäksi. Aktuaalinen eli täydellistynyt äärettömyys taas tarkoittaa ylinumeroituvasti ääretöntä joukkoa. Aristoteles julisti aktuaalisen äärettömyyden laittomaksi, koska hänen mielestään se on jumalallinen asia, jota ihminen ei pysty käsittämään.

2.2 G. W. Leibniz

Leibniz eli 1600–luvulla matemaatikkona ja filosofina. Hän oli taustaltaan yläluokkaa, joten hänellä oli taloudelliset mahdollisuudet keskittyä matematiikan kehittelyyn. Häntä on kutsuttu yhdeksi kaikkien aikojen suurimmaksi

matemaatikoksi. Hän oli erityisen kiinnostunut loogisen päättelyn hyödyntämisestä oikeusprosessa, erityisesti hankalien tapausten yhteydessä. Leibniz teki gradunkin logiikan ja lakiasioiden yhdistämisestä. Myöhemmin hän kehitteli kaupalliseen toimintaan malleja loogisen päättelyn avulla. Ympärivuotisen kaivostoiminnan suunnittelu oli eräs merkittävä askel hänen kehitykselleen. Yhteiskunnan rakenteiden takana oleviin päättelymalleihin kohdistuvan kiinnostuksensa takia hän tuli kehitelleeksi matemaattista logiikkaa eteenpäin. Erityisen tärkeää laskettavuuden kannalta oli, että hänellä oli alusta asti halu etsiä aakkosto, jossa merkeillä olisi käsitesisältö.

Leibnizin unelma aakkostosta ja laskutoimitusten formalisoinnista, joka kattaisi kaiken inhimillisen maailman tapahtumat, tiivistyi *Universaalin karakteristikan* kehittelyyn. Universaalinen karakteristiikka tarkoittaa suoraan käännettynä koko inhimillisen maailman kattavaa merkkijärjestelmää. Järjellä ajateltuna tällaisen karakteristikan kehittely on hyvin absurdi idea, mutta Leibniz uskoi Jumalan loogisuuteen maailman suunnittelussa, ja siksi hänelle tämä unelma oli ihan järkevä. Leibniz kehitti *symbolisen logiikan* eli logiikan algebran, jota kutsui nimellä *calculus ratiocinator*.

Etsiessään aakkostoa Leibniz toimi matemaattisen merkistön kehittelijänä. Hänen mielestään sopivien symbolien löytäminen on ensi arvoisen tärkeää. Leibniz ajatteli, että yksinkertaisten merkkien ilmaisemisella voidaan selkeyttää ajatteluprosessia. Hän otti käyttöön integraali- ja differentiaalilaskennassa käytetyt merkinnät $\int$ ja d. Integraalimerkki $\int$ muodostui summaa merkitsevästä kirjaimesta S venyttämällä, ja merkki d sanasta differentia. Hän huomasi, että nämä matemaattiset operaatiot ovat keskenään käänteiset. Leibniz kehitti myös laskukaavoja näille operaatioille.

Newton teki tahollaan vastaavanlaista merkistön tutkimusta, ja siksi hänellä oli Leibnizin kanssa kiistaa plagioinnista. Kuitenkin Leibnizin merkintätavat olivat ylivoimaisia, koska teki innovatiivisia ratkaisuja, siksi hänen integraalimerkintätapansa ovat jääneet käyttöön. Leibniz otti käyttöön lisäksi merkin $\oplus$ algebrallisissa laskutoimituksissa, kun laskutoimitus oli yhteenlaskun kaltainen. Hän otti ensimmäisenä käyttöön *binäärilukujärjestelmän*, jossa merkitään kaikki merkit lukujen 0 ja 1 avulla. Binäärilukujärjestelmää käytetään tietokoneissa vielä tänäkin päivänä.

Siihen aikaan äärettömyys miellettiin jumalien asiaksi, mutta Leibniz ei tästä lannistunut vaan tutki asiaa päättymättömien sarjojen ja differentiaalien yhteydessä. Leibnizin tuloksia on Leibnizin sarja $\frac{\Pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \cdots$, joka kertoo ympyrän alan jatkuvan sarjan avulla, kun säde on $\frac{1}{2}$. Leibnizin sarjassa äärettömät ovat Aristoteleen määritelmien mukaan potentiaalisia.

Leibniz esitteli vuonna 1674 algebrallisia yhtälöitä ratkovan koneen. Tämä oli merkittävä esimerkki mekanisoidun laskemisen merkityksestä ja kone toi esille laskemisen mekaanisen puolen. Tosin tässä vaiheessa ei vielä tiedetty, että kaikkia ongelmia ei voida ratkaista laskemalla. Leibnizin laskukonetta kutsuttiin *Leibnizin pyöräksi*, koska se toimi pyörän avulla. Pascal

oli kehittänyt oman suppeamman laskukoneen vuonna 1642. Pascalin laskukoneesta parannuksena Leibnizin koneeseen mukaan tulivat kerto- ja jakolaskut yhteen- ja vähennyslaskujen lisäksi. Leibnizin pyörä oli käytössä aina 1900-luvulle asti.

Leibniz haaveili loogisesta (tietokone) kielestä jo paljon ennen muita. Vasta noin kolme sataa vuotta myöhemmin Leibnizin unelman voidaan sanoa toteutuneen siinä mittakaavassa kuin se on mahdollista.

2.3 George Boole

Voidaan sanoa, että Boole jatkoi laskettavuuden tutkimusta siitä, mihin Leibniz jäi. Boole tuli kuitenkin köyhästä perheestä, eikä ollut käynyt juurikaan kouluja. Hän syntyi vuonna 1815 neljälapsisen perheen vanhimpana lapsen. Vanhempiensa tuella Boole pääsi kuitenkin pitkälle lähinnä itse opiskellen. Hänen isänsä oli aktiivisesti mukana mekaniikkaopiston toiminnassa ja kehitteli erilaisia laitteita harrastuksenaan. Boole kulki tässä isänsä jalanjälkiä ja innostui opiston toiminnasta. Opiskeluidensa jälkeen Boole elätti vanhempiaan ja perhettään perustamalla koulun, jonka johtamisen sivussa hän kehittyi itsenäiseksi matemaatikoksi.

Leibnizin tuloksiin Boole tutustui matemaatikko Samuel Clarkin kautta, joka oli ollut mukana Newtonin ja Leibnizin plagiointikiistoissakin ja esittänyt kuuluisan Jumalan olemassaolotodistuksen. Jo 32-vuotiaana Boole julkaisi artikkelin, jossa hän esitti logiikan ja matematiikan olevan sama asia.

Boole sovelsi algebrallisia menetelmiä niin kutsuttuihin operaattoreihin. Ne ”operoivat” tavallisen algebran ilmaisuista uusia ilmaisuja. Hänellä oli erityinen kiinnostus differentiaalioperaattoreita kohtaan, koska monet maailmankaikkeuden peruslaeista ovat differentiaaliyhtälöitä. Loogisten suhteiden ilmaiseminen algebran avulla tuli Boolen pohdintaan mukaan aikalaisten de Morganin ja S.W Hamiltonin kiistasta, joka käsitteli predikaatin kvantifioimisen periaatetta.

Boole ymmärsi Aristoteleen klassista logiikkaa tutkiessaan, että on olemassa luokkia. Aristoteleen logiikka käsitteli lauseita, jotka olivat esimerkiksi ”Kaikki kivet ovat elottomia” ja ”Yksikään pöllö ei osaa virkata”. Näissä lauseissa yksilöiden muodostamat kokonaisuudet eli ”kivet” ja ”pöllöt” ovat luokkia. Boole huomasi, miten näihin sanoihin liittyvää päättelyä voidaan esittää algebran avulla, kun merkitään luokkia kirjaimilla.

Jos esimerkiksi oletetaan, että x tarkoittaa lampaita (eli lampaiden luokkaa) ja y valkoisia olioita (eli niiden luokkaa), niin silloin xy tarkoittaa valkoisia lampaita. Boole oivalsi, että koska xx tarkoittaa lampaita, jotka ovat lampaita, niin se on sama luokka kuin x eli lampaiden luokka. Voidaan kirjoittaa siis $xx = x$. Boole huomasi edellisen lausekkeen olevan kertolaskun tyyppinen, paitsi tapauksessa xx.

Boolen algebra

Tässä alakohdassa esitellään lyhyesti Boolen oivalluksia logiikan parissa tekemästään työstä, jota jo edellä hieman sivuttiin. Jos x ja y edustavat kahta eri luokkaa (eli joukkojen joukkoa), niin xy tarkoittaa niiden olioiden luokkaa, jotka kuuluvat sekä luokkaan x että luokkaan y. Nykyisin joukkoa xy merkitään luokkien x ja y leikkauksena $(x \cap y)$.

Huomataan myös, että $xx = x$ pätee silloin, kun x on jokin luokka. Boole huomasi, että $xx = x$ pitää paikkansa tavallisessa algebrassa eli luvuilla vain, kun $x = 0$ tai $x = 1$. Logiikan algebra on siis täysin sama kuin binäärilukujen algebra. Selvyyden vuoksi Boole palautti luvut 0 ja 1 takaisin luokiksi, eli 0 tarkoitti luokkaa, joka oli tyhjä joukko $\emptyset$ ja 1 tarkoittaa luokkaa, joka on tarkastelun universumi eli perusjoukko. Eli kuten normaalissa algebrassa niin myös luokkien kohdalla $0x = 0$ ja $1x = x$ (siis $\emptyset \cap x = \emptyset$ ja $U \cap x = x$, kun U sisältää kaikki alkiot).

Koska tavallisessa algebrassa on myös yhteen- ja vähennyslaskut, löytyvät nämä myös Boolen algebrassa. Seuraavassa Boolen vastineet yhteen- ja vähennyslaskuille:

- $x + y$ vastaa $x \cup y$ ja
- $x - y$ vastaa $x \setminus y$ eli tarkoittaa niitä luokan x alkioita, jotka eivät kuulu luokkaan y.

Boole huomasi myös, että kirjoittamalla xx normaalia algebrallista merkintätapaa käyttäen x^2, saadaan perusyhtälöstä $xx = x$ yhtälö $x^2 = x$. Manipuloimalla edellistä yhtälöä edelleen saadaan $x - x^2 = 0$ ja edelleen $x(1 - x) = 0$, mikä onkin Aristoteleen ristiriidan laki. Samoin Boole huomasi, että joukkoon kuuluvat (x) ja kuulumattomat $(1 - x)$ muodostavat yhdessä perusjoukon, siis $x + (1 - x) = 1$.

Boole hylkäsi laajalle levinneen käsityksen siitä, että kaikki päättely palautuisi Aristoteleen syllogismeihin. Hänen mukaansa jo yksinkertaisissa arkielämän päätelmissä mukana on sekundaarisia lauseita, joilla ilmaistaan toisten lauseiden välisiä suhteita. Sekundaarisia lauseita sisältävä päättely ei ole syllogistista. Boole huomasi, että sama algebra, joka toimi luokkien tapauksessa toimii myös tämän kaltaisessa päättelyssä. Hän merkitsi $X = 1$ silloin kun X on tosi ja $X = 0$ vastaavasti epätoden kohdalla.

- Kun $X = 1$ ja $Y = 1$ (eli $X \wedge Y$), niin $XY = 1$.
- Kun $X = 0$ tai $Y = 0$ (eli $X \vee Y$), niin $XY = 0$
- Kun pätee ”jos $X = 1$ niin $Y = 0$”(eli $X \rightarrow Y$ tai $\neg X \vee Y$), saadaan tulos $X(1 - Y) = 0$.

Boolen algebra koostuu siis operaatioista $\wedge, \vee$ ja $\neg$, ja sillä on neutraalialkiot (0 ja 1). Operaatiot vastaavat joukko-operaatiota $\cup, \cap$ ja komplementti ja neutraalialkiot joukoille ovat perusjoukko ja tyhjäjoukko.

Boolen suurin tietokoneiden historiaan liittyvä saavutus oli tajuta, että loogista päättelyä voidaan käsitellä matemaattisesti. Voidaan ajatella, että Boolen logiikka vastasi Leibnizin yleiskalkyylia (calculus ratocinator) seuraavasti: Boolen $xx = x$ vastasi Leibnizin merkintää $A \oplus A = A$. Leibnizin ajatus, että säännöt johdettaisiin pienestä määrästä aksioomia, oli kuitenkin enemmän kuin mitä George Boole saavutti. Hän kuoli 49-vuotiaana keuhkokuumeeseen.

2.4 Gottlob Frege

Frege syntyi 1848 Saksassa ja kuoli 1925, siis ennen kuin Hitler nousi valtaan. Hänen vanhempansa pitivät tyttökoulua. Jo nuorena Fregen tavoitteena oli luoda logiikan järjestelmä, joka kattaisi matematiikassa kaiken deduktiivisen eli loogisesti sitovan päättelyn. Fregen logiikasta on tullut yliopistoissa opetettava peruskurssien logiikka.

Frege oivalsi, että esimerkiksi lause "kaikki hevoset ovat nisäkkäitä" saadaan muotoon $\forall x$(jos x on hevonen, niin x on nisäkäs) ja lause "jotkut hevoset ovat puhdasrotuisia" saadaan muotoon $\exists x$(x on hevonen ja x on puhdasrotuinen). Jos käytämme merkintää $h(x)$ predikaattiin "olla hevonen" ja vastaavasti "olla nisäkäs" $n(x)$ ja puhdasrotuinen $p(x)$, saamme lauseet $\forall x(h(x) \rightarrow n(x))$ ja $\exists x(h(x) \wedge p(x))$. Edellä mainittujen yksipaikkaisten predikaattien lisäksi Frege otti käyttöön myös kaksipaikkaiset predikaatit.

Frege teki kirjasen Begriffsschrift, joka on suoraan käännettynä *käsitekirjoitus*. Sitä on kuvailtu lukuteorian mukaisena puhtaan ajattelun kaavakielenä, ja on sanottu myös, että se on "Ehkä kaikkein aikojen tärkein logiikan alalta kirjoitettu teos". Käsitekirjoitus kuvasi keinotekoisen kielen, jolla oli täsmällinen kielioppi. Sen tärkein päättelysääntö oli seuraava: Oletetaan, että $\diamond$ ja Δ ovat mielivaltaisia käsitekirjoituksen lauseita. Silloin jos $\diamond$ ja $(\diamond \rightarrow \Delta)$ ovat totta, niin myös Δ on totta.

Fregen logiikka oli suuri edistysaskel Boolen logiikkaan verrattuna. Se ei kuitenkaan vastannut Leibnizin unelmaa siitä, että oikea vastaus päättelyihin saataisiin varmasti, kun vaan päästäisiin laskemaan. Fregen logiikassa virheiden tullessa ei voida päätellä, onko virhe premisseissä vai päättelyketjussa.

Myöhemmin Russell romutti suuren osan Fregen elämäntyöstä huomaamalla, että Fregen työ perustuu väärille oletuksille. Tätä Russellin huomiota kutsutaan *Russellin paradoksiksi*. Virhe oli ajatus, että kaikki matematiikka perustuu logiikkaan tai olisi palautettavissa logiikkaan, eli niin kutsuttu logisismi. Paradoksin voi esittää konkreettisena esimerkkinä: oletetaan, että kylän parturi ajaa parran niiltä ja vain niiltä kylän miehiltä, jotka eivät aja omaa partaansa. Ajaako hän tällöin oman partansa? Jos parturi ajaa

oman partansa, hän ei aja omaa partaansa ja kääntäen. Ongelmia tuli esille erityisesti luonnollisten lukujen $\mathbb{N}$ palauttamisessa.

Russellin paradoksi todisti Fregen ja Cantorin naiivin joukko-opin sisäisesti ristiriitaiseksi, ja teki Fregen katkeraksi mieheksi. Vasta Kurt Gödel todisti, ettei matematiikka voi olla koskaan täysin sisäisesti ristiriidatonta. Vaikka Frege pääsikin hyvin pitkälle, Leibnizin unelmasta jäi pois muun muassa tieteen totuuksien sisältävyys. Vuonna 1936 todistettiin pysähtymisongelman avulla, ettei yleistä todistuksen moitteettomuutta testaavaa keinoa ole. Frege kuului seuraavan esittelemämme matemaatikko Cantorin tukijoihin, vaikka Cantoria kritisoitiin yleisesti.

2.5 Georg Cantor

Cantor eli vuodesta 1845 vuoteen 1918. Hän oli alun perin pietarilainen, mutta sittemmin Saksassa vaikuttanut matemaatikko. Häntä sanotaan joukko-opin luojaksi, mutta hän oivalsi myös, että luonnolliset luvut voi hahmottaa sarjan tuotteina, 1, $1+1=2$, $2+1=1+1+1=3$, $\ldots$, $99+1=100$, $\ldots$.

Aristoteles kutsui edellistä kehitelmää potentiaaliseksi äärettömyydeksi. Aktuaalinen äärettömyys oli edelleen matemaatikko piireissä laiton. Kuitenkin Boolen aikoina raja-arvomenetelmän tullessa matematiikkaan potentiaalisen äärettömyyden käsittelyyn tuli työkaluja, ja sitä pystyttiin paremmin lähestyä.

Cantorin opettajista Eduard Heine kehotti Cantoria tutkimaan eräitä päättymättömiin sarjoihin liittyviä ongelmia. Cantor keskittyi erityisesti trigonometrisiin sarjoihin. Hän halusi todistaa, että kaksi eri trigonometristä sarjaa lähestyvät samaa raja-arvoa vain erikoistapauksissa. Todistukseen hän tarvitsi äärettömien joukkojen täydellistyneitä joukkoja. Leibnizin tavoin Cantorkin uhmasi aktuaalisen äärettömyyden laittomuutta. Hän otti tehtäväkseen aktuaalista äärettömyyttä koskevan laajan ja perinpohjaisen teorian luomisen. Myöhemmin Cantor kehitteli joukko-oppia itsenäisenä kohteena.

Leibniz ajallaan totesi, että äärettömien joukon alkioiden lukumäärästä puhuminen ei ole mielekästä. Hän esitti tutkimuksissaan, että luonnollisten lukujen joukko ja parillisten luonnollisten lukujen joukko ovat yhtä suuria. Cantor jatkoi tästä, että joko asia on näin tai eräillä äärettömillä joukoilla on yhtä monta alkiota kuin jollakin niiden osajoukoista. Cantor ryhtyi kehittämään jälkimmäistä ajatusta ja loi samalla pohjaa äärettömien joukkojen lukuteorialle. Hän tutki rationaalilukujen joukon vastaavuutta luonnollisiin lukuihin, koska se vaikutti suuremmalta ja loi diagonaalimenetelmänsä.

Cantor lähetti 28-vuotiaana Richard Dedekindille kirjeen, jossa oli merkittävä lausahdus: ” Reaalilukujen joukkoa ja luonnollisten lukujen joukkoa ei voida rinnastaa toisiinsa yksikäsitteisesti, joten äärettömiä joukkoja on ainakin kahta eri kokoa.”. Todistus perustui siihen, että reaaliluvut eivät tarkoita samaa kuin algebralliset luvut, vaan reaalilukuihin kuuluu enemmän, nimittäin transsendentaaliset luvut. Esimerkiksi luvut Π ja $2^{\sqrt{2}}$ ovat transsenden-

taalisia lukuja, mutta luku $\sqrt{2}$ ei ole transsendentaalinen vaan algebrallinen, koska se toteuttaa yhtälön $x^2 = 2$.

Cantor alkoi pohtia, että lukuja käytetään kahdessa eri merkityksessä. Nimittäin järjestys- eli ordinaalilukuina (1.,2.,3., ...) ja perus- eli kardinaalilukuina (1, 2, 3, ...). Erityisesti Cantor keskittyi käsittelemään äärettömiä peruslukuja. Hän oivalsi, että rinnastaakseen kaksi lukujoukkoa toisiinsa niillä pitää olla sama kardinaaliluku. Jos joukon M kardinaalia merkitään $\check{M}$ ja joukon N kardinaalia $\check{N}$, niin voi olla $\check{M} > \check{N}$ (esimerkiksi $M = \{1, 2, 3\}$ ja $N = \{\Delta, \Phi\}$).

Cantor kutsui äärettömien joukkojen kardinaaleja transfiniittisiksi. Luonnollisten lukujen joukon $\mathbb{N}$ kardinaalia hän merkitsi $\aleph_0$ (alef nolla), joka tulee hepreankielen aakkoston ensimmäisestä merkistä. Matematiikkaan on vakiintunut numeroituvasti ääretömän käsite, joka tarkoittaa kardinaalia $\aleph_0$ ja reaalilukujen kardinaalia, eli ylinumeroituvasti äärettömyyttä, Cantor merkitsi kirjaimella C (tulee kontinuumi eli jatkumo sanasta). Cantor väitti, että C on seuraava koko koon $\aleph_0$ jälkeen eli $\aleph_0 < C$ ja kardinaalien $\aleph_0$ ja C välissä ei ole muita äärettömien joukkojen kokoja. Tätä väitettä kutsutaan kontinuumihypoteesiksi. Kurt Gödel ratkaisi väitteen jollain tavalla vuonna 1938 ja Paul Cohen vuonna 1963.

Cantorilla oli vaikea elämä mielenterveyshäiriöidensä takia. Silti hän vei äärettömien joukkojen tutkimusta huomattavasti eteenpäin ja kehitti vielä nykyäänkin kaikille matematiikan opiskelijoille tutun diagonaalimenetelmän. Hän teki paljon tutkimusta matematiikan alueella, jota ei oltu aiemmin tutkittu. Siksi Cantor joutui haastamaan itseään eri tavalla kuin yleensä ja luottamaan omaan oivallukseensa. Matemaatikko David Hilbert on sanonut Cantorin transfiniittisistä luvuista (eli äärettömien joukkojen kardinaaleista), että tämä on ”matematiikan ymmärryksen ihastuttavin kukkanen ja ylipäänsä yksi ihmisälyn puhtaasti järkiperäisen ajattelun hienoimmista saavutuksista” ja myös ”kukaan ei karkota meitä Cantorin luomasta paratiisista”.

2.6 David Hilbert

Hilbert syntyi vuonna 1862 Saksassa ja vaikutti Göttingenin yliopistossa. Hän työskenteli muun muassa Gaussin, Riemannin, Dirictlet'n ja Kleinin kanssa ja syventyi matematiikan perusteiden tutkimiseen. Laskettavuuden parissa hän teki suuren saavutuksen palauttamalla euklidisen geometrian ristiriidattomuuden lukuteorian ristiriidattomuuteen. Cantorin tapana oli tarkastella logiikkaa ulkopuolelta ja hänen sanotaan sanoneen, että ”mitään ei voi perustella kaavioista näkemiseen, vaan matemaattinen todistus on oltava pelkkää logiikkaa”.

Hilbertin nimi on tunnettu myös hänen laatimastaan 23:en matemaattisen ongelman listastaan, jossa ongelmat vaikuttivat sen ajan välineillä täysin ylivoimaisilta. Listassa ensimmäinen ongelma oli kontinuumihypoteesi ja toinen ongelma oli reaalilukujen teorian aksioomien ristiriidattomuuden to-

distaminen. Hilbertin listan kymmenes ongelma oli keksiä algoritmi, jolla voidaan selvittää, onko polynomilla kokonaislukujuuria. Sellaista algoritmia ei ole, joka voisi ongelman ratkaista. Tähän palataan vielä teoriaosuudessa kappaleessa 5.4.

Vuonna 1902 matematiikan perusteisiin liittyvät ongelmat alkoivat muodostaa kriisiä matematiikan alalla. Silloin myös Russellin paradoksi tuli esiin, ja Hilbert ratkaisi Gordanin algebrallisiin invariantteihin liittyvän otaksuman, jonka mukaan jokaisen yksittäisen algebrallisen ilmaisun invarianttien joukosta löytyisi aina äärellinen määrä ratkaisevia invariantteja (invariantit ovat muunnoksissa muuttumattomina säilyviä asioita). Tämä todistus avasi uuden oven tulevan ajan matematiikalle, koska siitä kävi ilmi abstractin ajattelun voima. Tämän todistuksen jälkeisen tuotteliaan kauden jälkeen Hilbertin ura alkoi hiipua, vaikka hän kehitteli vielä paljon luomaansa matematiikan osa-aluetta, metamatematiikkaa, eteenpäin.

Russell työskenteli tiiviisti kehittääkseen symbolisen logiikan järjestelmän. Järjestelmässä koneelle annettaisiin aksioomia ja siitä tulisi ulos valmiita teoreemoja. Russell yritti paradoksinsa jälkeen vielä elvyttää Fregen ohjelmaa Whiteheadin kanssa tekemänsä Principia mathematican avulla. Tässä teoksessa ristiriitaisuuksilta vältyttiin harkittujen päällekkäisyyksien avulla, esimerkiksi sekoittamalla arkikieltä ja formaalia kieltä.

Vuonna 1920 Hilbert ryhtyi tutkimaan Ackermannin ja Neumannin kanssa reaalilukujen ristiriidattomuuden kimppuun. He aloittivat tutkimuksen Russellin ja Whiteheadin Principia mathematicasta. Ryhmä oivalsi, että on olemassa ulkoinen ja sisäinen näkemys symboliseen kieleen. Haluttiin todistaa, että tässä kielessä ei voida johtaa yhtään selvästi ristiriitaista lauseparia, kuten $1 = 0$.

Hilbertin ensimmäisen kertaluvun logiikasta eli peruslogiikasta löytyi kaksi ongelmaa. Nimittäin ensimmäisen kertaluvun logiikan täydellisyys (eli kaikkien kaavojen johtaminen kaavakokoelmasta) ja Hilbertin ratkaisuongelma. Hilbertin ratkaisuongelman sisältö on, että totuus saataisiin mille tahansa kaavalle laskemalla äärellisestä määrästä päättelysääntöjä. Hilbert ymmärsi logiikan ulkopuolelta tarkastelun tärkeyden.

Vuonna 1928 Hilbert otti esiin ongelman, joka koski Fregen sääntöjen soveltamista Peano-aritmetiikan aksiomatisoinnissa. Nuori matemaatikko Kurt Gödel huomasi ajatuksen sisältävän ristiriitaisuuden ja tuhosi tällä tiedollaan myöhemmin Hilbertin ohjelman. Vielä vuotta myöhemmin Hilbert esitti puheessaan, ettei ratkeamattomia ongelmia ole, ja lausui kuuluisat sanat: ”Meidän on tiedettävä. Me tulemme tietämään.”. Tämän puheen jälkeisessä kahvipöytäkeskustelussa Gödel esitti eriävän näkökantansa, josta esimerkiksi Neumann ymmärsi heti, että kaikki tällä saralla on ohi. Ristiriitojen jälkeen matematiikan perusteiden tutkimuksessa alkoi uusi aikakausi.

2.7 Kurt Gödel

Kurt Gödel syntyi vuonna 1906 Itävalta-Unkarin keisarikunnassa ja eli vuoteen 1978. Hän vaikutti suuresti aksiomaattisen joukko-opin kehittämiseen ja hänen tärkein opettajansa oli Frege. Hän todisti epätäydellisyysteoreeman ja ettei matematiikka voi olla sisäisesti täysin ristiriidatonta. Gödel tajusi, että ristiriidattomuus on itse todistumaton. Alan Turing käytti Gödelin tulosta todistaessaan pysähtymisongelman ratkeamattomuuden.

Väitöskirjassaan hän todisti, että Hilbertin looginen säännöstö on täydellinen. Siis deduktiivinen eli loogisesti sitova päättely etenee premisseistä johtopäätökseen. Fregen-Russellin-Hilbertin symbolisessa logiikassa premissit ja johtopäätökset esitetään loogisena kaavana eli merkkijonona.

Gödelin väitöskirjan sisältö oli ollut jo kauan esillä, mutta ristiriidat äärettömyyksistä olivat estäneet näkemästä sitä. Hilbert luuli saavansa todistuksen äärellisiin lukuihin rajattuna, mutta Gödelin täydellisyyslausetta ei voi todistaa ilman äärettömien olemassaoloa ja käyttämistä. Gödel huomasi, että on olemassa lauseita, joiden totuus voitiin tunnistaa järjestelmän ulkopuolelta, mutta ei sisäpuolelta. Yleisesti matemaatikoiden mielestä matemaattisen totuuden takeeksi kelpasi vain todistettavuus. Gödelin mukaan matemaattisen totuuden käsite on olemassa, mutta sen ala ulottui laajemmalle kuin on mahdollista todistaa missään annetussa formaalisessa järjestelmässä Peano aritmetiikasta Principia mathematicaan.

Gödelin ratkaiseva todistus oli, että luonnollisen luvun ominaisuus ”olla Principia mathematicassa todistuvan lauseen koodi” on itse ilmaistavissa Principia mathematicassa. Siis:

- Lause U väittää, että jokin tietty lause ei ole todistuva Principia mathematicassa.
- Kyseinen lause ei ole mikään muu lause kuin lause U itse.
- Näin ollen lause U siis väittää, että ”U ei ole todistuva Principia mathematicassa”.

Gödelin epätäydellisyyslause sanoo, että ”U on tosi, mutta ei todistuva Principia mathematicassa” (Myöskään $\neg U$ ei ole todistuva). Johtopäätös on nyt: *jos Principia mathematica on ristiriidaton, myös lause U on* ja siis Principia mathematican ristiriidattomuuden todistamiseen tarvitaan muutakin kuin vain Principia mathematica.

Gödel kehitti erityisen kielen, jonka avulla tarvittavia operaatioita voidaan kehittää vaihe vaiheelta. Kukin vaihe koostui operaatiolle annetusta numeerisesta määritelmästä, jota kutsutaan Gödelin luvuksi tai Gödelin koodiksi, noudattaen tiettyä operaatiota Principia mathematicassa. Gödel käytti kiinalaisena jäännöslauseena tunnettua teoreemaa ja osoitti sen avulla, että kaikki hänen erikoiskielessään olevat lauseet ovat ilmaistavissa myös luonnollisten lukujen aritmetiikassa.

Gödelin tulos, että ristiriidattomuus itse on todistumaton aiheutti Hilbertin ohjelman loppua. Tämän tapauksen takia Neumann jätti logiikan lähes kokonaan. Myöhemmin Neumannista tuli Gödelin ystävä, ja hän on jopa kutsunut Gödeliä suurimmaksi loogikoksi Aristoteleen jälkeen. Myös Einstein oli Gödelin ystävä.

Vuonna 1937 Gödelin ajoittainen masennus alkoi jälleen helpottaa, ja hän sai pitkästä aikaa läpimurron, juurikin Cantorin kontinuumihypoteesin parissa ($\aleph_0 < C$). Hän osoitti, että Russellin ja Whiteheadin Principia mathematicassa tai muissakaan joukko-opin aksioomiin perustuvissa järjestelmissä ei voida kumota kontinuumihypoteesia. Siis kontinuumihypoteesi on ratkeamaton. Kontinuumihypoteesille ei ole vieläkään ratkaisua.

Myöhemmin urallaan Gödel paneutui filosofiaan ja tekoälytutkimukseen. Gödelin epätäydellisyyslause piti siis sisällään ajatuksen, että jokaisella matemaattisella järjestelmällä on itsensä ylittäviä matemaattisia ongelmia, joten ratkaisu vaatii laajempaa järjestelmää, jolla taas on omat ongelmansa.

2.8 Alan Turing

Turing oli esitellyistä henkilöistä ehkä merkittävin matemaatikko tietokoneiden kehityksessä. Hän eli vuodesta 1912 vuoteen 1954 ja kehitti mekaanisen laskennan systeemin, jota nykyisin kutsutaan Turingin koneeksi. Turingin seksuaalisen suuntautumisen aiheuttama traaginen elämänkohtalo kietoutui merkittävästi koko maailman politiikkaan. Häntä on tituleerattu joskus jopa lausahduksella ”Mies joka ratkaisi toisen maailmansodan”. Turing nimittäin työskenteli toisessa maailmansodassa saksalaisten Enigma salakoodausjärjestelmän purkamisen kimpussa. Hän myös suunnitteli tietokoneen mallin ja oli rakentamassa ensimmäisiä tietokoneita.

Aiemmin laskukoneiden suunnittelua oli tehnyt Charles Babbage 1800–luvulla. Hän esitteli vuonna 1834 ”Analyyttisen koneen”, joka toimi täysin mekaanisesti hammaspyöriä käyttäen. Vuonna 1930 kyseisiä koneita alettiin rakentaa sähköreleiden avulla. Muutenkin vuosina 1930–40 laskukoneiden ja mekaanisen laskennan parissa tehtiin paljon tutkimusta, johon myös Turing tempautui mukaan. Yliopistouran alkupuolella Turing kiinnostui John von Neumannin kvanttimekaniikan kirjasta ja normaalijakaumasta. Tärkeä merkkipaalu oli, kun Turing osallistui 1935 Neumannin matematiikan perusteita käsittelevälle luentokurssille, ja tutustui siellä Gödelin epätäydellisyyslauseeseen ja Hilbertin ratkaisuongelmaan. Vuonna 1935 Turing muotoili nykyisen laskemista koskevan peruskäsityksen pohtiessaan Hilbertin matemaattisen logiikan ongelmia.

Turingin idea Turingin koneen kehittelyssä lähti liikkeelle ajatuksesta, että algoritmi on luettelo sääntöjä, joita joku henkilö voi noudattaa täsmällisesti ja mekaanisesti. Turing keskittyi tässä ideassaan algoritmia suorittavan henkilön toimintaan. Turing oivalsi, että prosessi voidaan yksinkertaistaa muutamaan perustoimintoon ja korvasi henkilön koneella, joka suoriu-

tui näistä perustoiminnoista. Sitten hän pystyi todistamaan, että mikään tällainen kone ei pysty määrittämään, seuraako jokin annettu johtopäätös annetuista premisseistä Fregen sääntöjen perusteella, ja päätyi siihen, että ratkaisuongelmalla ei ole algoritmia. Turing näitä pohdintoja tehdessään tuli luoneeksi tietokoneen matemaattisen mallin. Tietokoneen englanninkielinen nimikin tulee sanasta laskija (computer).

Laskettavuuden ja laskukoneiden tiimoilta oli vilkasta keskustelua 1930–luvulla. Princetonin yliopistoon, jossa Turingkin toimi, oli silloin keskittynyt niin paljon matemaattista lahjakkuutta, että se ohitti Göttingenin maailman matemaattisena keskuksena. Turing kehitteli ajatusta laskennan mekanisoinnista eteenpäin ja viimein vuonna 1936 julkaisi matemaattisen mallin koneesta, jota nykyään kutsutaan Turingin koneeksi.

Myös muita ekvivalentteja laskennan malleja luotiin vuonna 1936 Turingin koneen lisäksi. Muun muassa Alonzo Church esitteli efektiivisen kalkuloitavuuden juuri ennen Turingin koneen julkaisua. Siksi Turing lisäsi raporttiinsa yhteyden Churchin tulokseen. Samoin Emil Post kehitteli Postin sääntöjärjestelmän. Gödelin ja Kleenen tulos oli nimeltään μ-rekursiiviset funktiot. Nämä kaikki systeemit on osoitettu ekvivalenteiksi, siis jokainen niistä kelpaa algoritmin täsmälliseksi määritelmäksi niin kutsutun Churchin-Turingin teesin nojalla.

Vuonna 1939 syttyi toinen maailmansota. Ennen sotaa Turing valmisti binäärilukuja kertovan koneen sähkömekaanisista releistä. Sodan aikana Turing oli Britannian armeijalla töissä purkamassa saksalaisten sukellusveneiden Enigma salakoodausjärjestelmää yhdessä 12 tuhannen muun työntekijän kanssa. Koodin purkutyössä Turingilla oli apuna kehittämänsä kone Bombe, joka teki loogisia päätelmiä kunnes jäljellä oli vain muutamia vaihtoehtoja, jotka testattiin käsin. Universaalien Turingin koneiden kehittely jatkui Turingin muiden töiden rinnalla sodan aikana ja sen jälkeen. Sodan loppuun mennessä Turing oli saanut paljon tietoa elektroniputkivirtapiireistä ja universaalikoneen rakennus oli enää vain rahasta kiinni. Ensimmäinen tietokone saatiin vihdoin rakennettua ja tietokoneiden kehittely eteni tämän jälkeen hyvin nopeasti. Jo vuonna 1950 tietokone voitti ensimmäisen kerran ihmisen shakissa ja ilmeisesti vuonna 1954 tietokoneohjelma tuotti ensimmäisen kerran matemaattisen todistuksen.

Kehityksessä pitemmälle menneitä tietokoneita oli muun muassa Colossus ja ENIAC. Colossuksessa oli 1500 elektroniputkea, kun taas ENIAC:ssa oli jo 18 000 elektroniputkea. Vuonna 1945 rakennettiin binääristä EDVAC konetta, joka meni vieläkin pidemmälle, ja jota Neumann kutsui Turingin universaalikoneen ruumiillistumaksi. Seuraavaksi alkavassa teoriaosiossa päästään vähän tarkemmin käsittelemään Turingin koneita ja universaalikoneita kappaleissa 5 ja 6. Viisikymmentäluvun alussa yhteiskunta vaikeutti Turingin elämää hänen homoseksuaalisuuteensa takia ja hän kuoli oman käden kautta vuonna 1954.

3 Esitietoja

Ennen Turingin koneeseen tutustumista tarkastellaan joitain suppeampia laskettavuuteen liittyviä malleja kuten äärellisiä automaatteja ja säännöllisiä kieliä. Kuitenkin ennen sitä määritellään joitain peruskäsitteitä, joita tullaan tarvitsemaan jatkossa.

3.1 Peruskäsitteitä

Määritelmä 3.1. *Aakkosto* on mikä tahansa äärellinen joukko merkkejä.

Esimerkki 3.1. $\sum_1 = \{0, 1\}$ on binääriaakkosto, $\sum_2 = \{a, b, c, \ldots, z\}$ on latinalainen aakkosto ja aakkosto $\sum_3 = \{\}$ on tyhjä aakkosto, eli siihen ei kuulu mitään merkkiä. Myös $\sum_4 = \{\langle ACCEPT\rangle, \langle REJECT\rangle\}$ on aakkosto, mutta harvemmin käytetään aakkostoja, joissa on alkiot ovat useamman merkin pituisia. Tämä aakkosto voitaisiin pelkistää aakkostoksi $\sum_1 = \{0, 1\}$, kun tiedetään merkin 0 tarkoittavan $\langle ACCEPT\rangle$ ja merkin 1 tarkoittavan $\langle REJECT\rangle$.

Kuten edellä esimerkeissäkin aakkostoa merkitään yleensä kirjaimella $\sum$ tai Γ. Aakkoston merkit voivat olla myös muita symboleja kuin yksittäisiä merkkejä, kuten viimeisessä esimerkissä. Kuitenkin mekaanisen laskettavuuden malleissa ja tietokoneissa laskentaa helpottaa, jos aakkoston symbolit ovat yksittäisiä merkkejä. Aakkoston $\sum$ kaikkien merkkijonojen joukkoa merkitään $\sum^*$.

Esimerkki 3.2. Aakkoston $\sum_1$ kaikkien merkkijonojen joukko leksikograafisessa järjestyksessä on $\sum_1^* = \{1, 0\}^* = \{\epsilon, 0, 1, 00, 01, 10, 000, 001, \ldots\}$.

Yleensä automaateissa ja koneissa käytetään aakkostona binääriaakkostoa, kuten oikeat tietokoneetkin käyttävät. Näin tullaan tekemään myös tässä työssä. Kuitenkin jos aakkostona ei ole binääriaakkosto, kannattaa muistaa, että kaikki aakkostot ovat muutettavissa binäärimuotoon niin kutsutun koodauksen avulla. Seuraava esimerkki käsittelee koodausta.

Esimerkki 3.3. Tässä esimerkissä on eräs esimerkki binääriluvuiksi koodauksesta. Tässä kone tunnistaa merkkijonon alun ykkösestä. Nollien lukumäärä kertoo, mikä merkki on kyseessä. On olemassa myös muita binäärikoodauksia latinalaiselle aakkostolle, jotka useimmiten vievät vähemmän muistia. Tässä aakkosto olisi $\Sigma_4 = \{1,10,100,1000,\ldots,10^{24}\}$. Koodaukseen palataan vielä myöhemmin luvussa 3.3.

binääri	latinalainen
1	a
10	b
100	c
1000	d
10000	e
$\vdots$	$\vdots$
$1\underbrace{0\ldots0}_{24}$	z

Yleensä tietokoneille tai niitä mallintaville matemaattisille malleille annetaan syötteenä merkkijonoja kuten "Tulosta nimi!", ja ne myös käsittelevät merkkijonoja koneen toiminnassa tiedon siirtämisen keinona. Täsmällisesti *merkkijono* on äärellinen järjestetty jono aakkoston merkkejä. Tyhjää merkkijonoa merkitään merkillä ϵ ja joskus myös λ. Merkkijonoa kutsutaan joskus lauseeksi tai sanaksi. Merkintä $|\alpha|$ tarkoittaa joukon alkioiden lukumäärää, eli tässä aakkoston tai merkkijonon α merkkien lukumäärää.

Esimerkki 3.4. Eräs aakkoston $\sum_1 = \{0, 1\}$ merkkijono on $w_1 = 0001101$. Toisaalta eräs aakkoston $\sum_2$ merkkijono on $w_2 = assyyria$ ja aakkoston $\Sigma_4 = \{1, 10, 100, 1000, \ldots, 10^{24}\}$ merkkijono 110 vastaa latinalaisen aakkoston merkkijonoa ab.

Esimerkki 3.5. Aakkosto $\sum_1$ sisältää kaksi merkkiä eli $|\sum_1| = 2$, merkkijono w_1 on seitsemän merkin mittainen eli $|w_1| = |0001101| = 7$ ja merkkijonolle ϵ pätee $|\epsilon| = 0$.

Määritelmä 3.2. Merkkijonojen $a = a_1a_1 \cdots a_n$ ja $b = b_1b_1 \cdots b_m$ *liitos eli katenaatio* on $a \cdot b = ab = a_1a_1 \cdots a_nb_1b_1 \cdots b_m$.

Merkkijonon x katenaatiota itsensä kanssa eli operaatiota xx merkitään x^2. Samoin jos katenaatio tehdään useamman kerran peräkkäin, esimerkiksi v kertaa tehtäessä x^v. Kuten aiemmin aakkoston Σ kaikkien merkkijonojen joukkoa merkittiin symbolilla Σ^*, niin samaa tähtioperaatiota voidaan käyttää myös merkkijonolle tai merkille. Merkkijonon a *tähtioperaatio* a^* tarkoittaa kaikkia merkkijonoja $x \in \{\epsilon, a, aa, aaa, \ldots\}$.

Määritelmä 3.3. Merkkijonon $w = w_1w_2 \cdots w_n$ *käänteismerkkijono* on merkkijono $w^R = w_n\ w_{n-1} \cdots w_1$, jossa w_1, w_2, $\ldots$ $w_n \in \Sigma$.

Esimerkiksi merkkijonojen $v = 000$ ja $w = 1111$ liitos on $vw = 0001111$, ja merkkijonon $vw = 0001111$ käänteismerkkijono on $(vw)^R = 1111000$. Palindromi on merkkijono, joka on itsensä käänteismerkkijono, esimerkiksi "saippuakauppias" (palinromille $w = w^R$).

Merkkijonon osajono on nimensä mukaan osa merkkijonosta. Vasemmanpuoleinen osajono on merkkijonon vasemmassa reunassa ja oikeanpuoleinen

vastaavasti oikeassa reunassa. Oletetaan $w_1, w_2, \ldots w_n \in \Sigma$, silloin merkkijonon $w = w_1 w_2 \cdots w_n$ *vasemmanpuoleinen osajono* on esimerkiksi merkkijono $v = w_1 w_2$. Osajonoa sanotaan toisinaan myös osamääräksi. Merkkijono "as" on merkkijonojen "kassi" ja "tassu" osajono. Tyhjä merkkijono on kaikkien merkkijonojen osajono.

Yleensä merkkijonot on järjestetty joukossaan Σ^* leksikograafiseen järjestykseen. Kun merkkijonot ovat leksikografisessa järjestyksessä, ne ovat pituuden mukaan kasvavassa järjestyksessä ja samanpituiset aakkosjärjestyksessä. Aakkosjärjestyksessä tarkoittaa kyseisen aakkoston mukaisessa järjestyksessä.

Määritelmä 3.4. Joukko A on aakkoston Σ *kieli*, joss $A \subseteq \Sigma^*$. Eli mikä tahansa joukko annetun aakkoston merkkijonoja on eräs aakkoston *kieli*.

Esimerkki 3.6. Joukko $A = \{\epsilon, 0, 1, 01, 10\}$ on aakkoston $\Sigma = \{0, 1\}$ kieli. Joukko $B = \{0, 1\}$ on myös aakkoston $\Sigma = \{0, 1\}$ kieli. Joukko $C = \{a, cd\}$ on aakkoston $\Gamma = \{a, b, c, d, e, f, g, h, i, j, k\}$ kieli.

Kielten joukkoa kutsutaan toisinaan kieliperheeksi. Esimerkiksi eräs kieliperhe on joukko $D = \{A, B\}$, mutta myös joukko $E = \{B, C\}$ on eräs kieliperhe (kaksialkioisten kielten perhe).

Kielet ja kieliperheet voivat olla myös äärettömiä. Kun äärellisessä kielessä voidaan luetella kieleen kuuluvat merkkijonot eli sanat, niin äärettömissä kielissä kieleen kuuluminen perustellaan ominaisuuksien perusteella. Samoin on kieliperheiden kanssa. Eräs ääretön kieli on kaikkien aakkoston merkkijonojen joukko Σ^* ja ääretön kieliperhe voisi koostua kielistä, joiden merkkijonojen pituus on kaksi merkkiä.

3.2 Ongelmanratkaisusta

Predikaatti on n-paikkainen relaatio. Predikaateista voi joskus muotoilla kuvauksen, jonka voi ratkaista jollain algoritmilla. Kuvauksen tuloksesta nähdään, milloin predikaatti on tosi ja milloin epätosi. Tällaista kuvausta sanotaan päätösongelmaksi.

Määritelmä 3.5. Predikaatin $M(x)$ *päätösongelma eli karakteristinen funktio* π on kuvaus $\pi : \sum^* \to \{0, 1\}$, kun $x \in \Sigma^*$, missä

$$\pi = \begin{cases} 1 & \text{, kun } M(x) \text{ on tosi} \\ 0 & \text{, kun } M(x) \text{ on epätosi.} \end{cases}$$

Päätösongelman erityistapauksia ovat niin kutsutut tunnistusongelmat. Niissä predikaattina on "Kuuluko merkkijono johonkin tiettyyn kieleen".

Määritelmä 3.6. Jokaista kieltä A vastaa kielen *tunnistusongelma* $\pi_A(x)$, joka on päätösongelma $\pi_A : \sum^* \to \{0, 1\}$

$$\pi_A(x) = \begin{cases} 1 & \text{, jos } x \in A; \\ 0 & \text{, jos } x \notin A. \end{cases}$$

Siis päätösongelma tunnistaa, mitkä merkkijonot kuuluvat kieleen A.

Määritelmä 3.7. Predikaatti on *ratkeava*, jos sen karakteristinen funktio on laskettava eli ratkaistavissa tarkalla mekaanisella algoritmilla.

Esimerkki 3.7. Voidaan esittää predikaatti "suurempi kuin" $S = (a, b)$, joka kuvaa ongelmaa, päteekö $a \leq b$. Jos näin on, predikaatti on tosi, jos taas muuttujat eivät ole suuruusjärjestyksessä pienemmästä suurempaan, predikaatti on epätosi. Tämän predikaatin karakteristinen funktio on

$$\pi(a, b) = \begin{cases} 1 & \text{kun } S(a, b) \text{ pätee eli on tosi} \\ 0 & \text{kun } S(a, b) \text{ ei päde eli on epätosi} \end{cases}$$

Sanotaan, että predikaatti S on ratkeava, jos sen ratkaisuongelma voidaan ratkaista jollain mekaanisella algoritmilla, eli ratkaisuongelma on laskettava eli ratkeava. Tässä tapauksessa voidaan algoritmiksi ottaa seuraava algoritmi, joka voitaisiin esittää myös myöhemmin esitettävillä täsmällisillä laskennan malleilla kuten Turingin koneella.

1. Piirrä paperille a kappaletta viivoja.
2. Piirrä paperille toiseen kohtaan b kappaletta viivoja.
3. Toista seuraavaa, kunnes toisesta tai molemmista kohdista ovat viivat loppuneet.
   - Yliviivaa yksi a kohdan viiva.
   - Yliviivaa yksi b kohdan viiva.
4. Jos lopussa a kohdassa on jäljellä viivoja, väite ei päde, joten vastaa 0. Muussa tapauksessa väite pätee, ja siis vastaus on 1.

Lause 3.8. *Jokaisen aakkoston Σ päätösongelmien joukko on ylinumeroituva.*

Todistus. (Ks. [3, s. 15-16])
Tämän lauseen todistus perustuu Cantorin diagonaalimenetelmään. Todistuksessa saadaan muotoiltua päätösongelma, joka eroaa jokaisesta päätösongelmien joukon alkiosta diagonaalilla. Merkitään kaikkia aakkoston Σ päätösongelmien joukkoa

$$\Pi = \{\pi \mid \pi \text{ on kuvaus } \Sigma^* \to \{0, 1\}\}.$$

Oletetaan, että Π on numeroituva. Siis on olemassa joukon Π kaikki alkiot läpi käyvä numerointi $\Pi = \{\pi_0, \pi_1, \pi_2, \ldots\}$. Olkoot joukon Σ^* merkkijonot lueteltuina leksikografisessa järjestyksessä $x_0, x_1, x_2 \ldots$. Muodostetaan uusi päätösongelma $\tilde{\pi} : \Sigma^* \to \{0, 1\}$

$$\tilde{\pi}(x_i) = \begin{cases} 1, & \text{jos } \pi_i(x_i) = 0 \\ 0, & \text{jos } \pi_i(x_i) = 1 \end{cases}$$

Koska Π on kaikkien aakkoston Σ päätösongelmien numerointi, niin $\tilde{\pi} \in \Pi$. Siis $\tilde{\pi} = \pi_k$, jollakin $k \in \mathbb{N}$. Tällöin

$$\tilde{\pi}(x_k) = \begin{cases} 1, & \text{jos } \pi_k(x_k) = \tilde{\pi}_k(x_k) = 0; \\ 0, & \text{jos } \pi_k(x_k) = \tilde{\pi}_k(x_k) = 1. \end{cases}$$

Koska tässä on ristiriita, niin alussa tehty oletus joukon Π numeroituvuudesta ei pidä paikkaansa. On siis todistettu, että Π on ylinumeroituva. □

Ohjelma

Ohjelma on tarkka algoritmi, jossa on äärellinen jono käskyjä. (Ks. [1, s. 16]) Myöhemmin käsiteltävä Turingin kone on yhdenpitävä ohjelman käsitteen kanssa.

Ohjelman P yhteydessä merkitään

- $P(x)$, jos ohjelma ajetaan syötteellä x.
- $P(x) \downarrow$, jos laskenta päättyy syötteellä x.
- $P(x) \uparrow$, jos laskenta jatkuu ikuisesti eli jää luuppiin syötteellä x.
- $P(x) \downarrow b$, jos laskenta loppuu loppuarvoon b syötteellä x.

Määritelmä 3.8. Ohjelma $P(x)$ ratkaisee karakteristisen ongelman π, jos kaikilla syötteillä x ohjelma P tulostaa arvon $\pi(x)$.

Määritelmä 3.9. Funktio f on *ratkeava*, jos on olemassa ohjelma P, joka ratkaisee funktion f, siis jos $P(x) = f(x)$ kaikilla $x \in Dom f$

3.3 Koodaus

Laskettavuutta tutkittaessa eri laskentamalleilla niissä kaikissa voidaan käyttää mitä tahansa aakkostoa. Voidaan käyttää esimerkiksi latinalaista aakkostoa tai aakkostoa $\Sigma = \{\alpha, \beta, \chi, \delta, \epsilon, \phi, \varphi, \gamma, \eta, \iota\}$ tai $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Maailmassa merkkejä on kuitenkin niin paljon, että kaikki tietokoneet tai ohjelmat eivät voi kaikkia merkkejä tunnistaa. Siksi aakkostot koodataan yleensä binäärimuotoon Leibnizin jalanjälkiä seuraten. Silloin koneen pitää tunnistaa vain merkit 0, 1 ja tyhjän ruudun symboli, joka tämän tutkielman tapauksessa on $\sqcup$.

Erilaisia koodauksia on hyvin paljon. Alla on eräs esimerkki luonnollisten lukujen koodauksesta binäärilukumuotoon.

$\mathbb{N}$	binääri
00	0000
01	0001
02	0010
03	0011
04	0100
$\vdots$	$\vdots$

Yleinen sääntö tässä luonnollisen luvun muuttamiseksi binäärimuotoon tapahtuu siten, että luonnollinen luku

$$y_{n-1}y_{n-2}\cdots y_0 \text{ muutetaan binääriluvuksi kaavalla } \sum_{i=0}^{n-1}(y_i)_2 \cdot 1010_2^i,$$

jossa alaindeksi kertoo, missä kantaluvussa luku on. (Ks. [8, s. 3])

Tietokoneet käyttävät binärilukujärjestelmän rinnalla heksadesimaali eli 16-lukujärjestelmää ja oktaali eli 8-lukujärjestelmää. Tämän tutkielman kaikissa esimerkeissä koodauksia ei ole tehty ja esimerkeissä on sekaisin esimerkiksi sekä binärilukuina että normaaleilla kymmenlukujärjestelmän luvuilla tehtyjä tehtäviä. Hyvä on kuitenkin muistaa, että toisen järjestelmän luku voidaan aina muuttaa toiseksi yksinkertaisella tavalla.

Myös kirjaimia tai matemaattisia merkkejä voidaan koodata koneella luettavaan muotoon. Esimerkiksi lausekkeen $(\forall x)((\exists y)L(x,y) \rightarrow H(x))$, joka voisi tarkoittaa "jokainen joka opettaa jotakuta saa harmaita hiuksia", saa muutettua koneella tutkittavaan muotoon (Ks. [2, s.]), kun muutetaan merkit vastaamaan luonnollisia lukuja seuraavalla tavalla :

$\mathbb{N}$	0	1	2	3	4	5	6	7	8	9
merkki	,	L	H	$\rightarrow$	$\forall$	$\exists$	x	y	(	)

Siis lauseke $(\forall x)((\exists y)L(x,y) \rightarrow H(x))$ voidaan saada muotoon 846988579186079328699. Tietysti mikä tahansa muukin yksikäsitteinen koodaus toimisi. Samoin voidaan koodata esimerkiksi lauseke $\int(x^4 - x^{-5})$ ja voidaan suunnitella Turingin kone, joka osaa laskea tämän lausekkeen arvon näillä symboleilla ja tietyillä alkuehdoilla eli syötteellä. Universaalikoneiden yhteydessä palataan itse Turingin koneiden koodaamiseen kappaleessa 6.1.

3.4 Graafeista

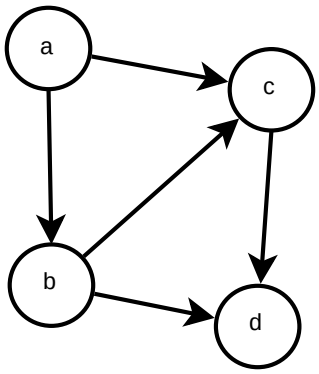
Seuraavassa esitettävää graafia käytetään usein apuna ongelmanratkaisussa algoritmin toimintaa kuvaamaan. Graafi on kuva, jossa on ympyröillä merkittyjä tiloja, ja kaarilla merkittyjä tilasiirtymiä tiloista toiseen. Jos kaaret on kuvattu nuolilla, niiden suunnalla on merkitystä. Graafin polku tarkoittaa kaarien peräkkäistä seuraamista solmusta toiseen.

Määritelmä 3.10. (Ks. [9, s. 10]) *Graafi* eli verkko $G = (V, E)$ on äärellinen joukko solmuja V ja kaaria E, missä kaaret ovat solmupareja. Joukko E on siis $E \subseteq \{(a, b) \mid a, b \in V\}$. [7]

Yleensä graafissa kaarien joukon E parien (a, b) järjestyksellä ei ole väliä, vaan solmut voidaan merkitä myös järjestyksessä (b, a). Jos järjestyksellä on merkitystä, graafia sanotaan suunnatuksi graafiksi.

Määritelmä 3.11. *Suunnatun graafin* $G = (V, E)$ kaarten joukossa E pätee, että $(a, b) \neq (b, a)$ kaikilla $a, b \in V$. Siis suunnatussa graafissa kaarten joukko E koostuu järjestetyistä pareista $\langle a, b \rangle$.

Esimerkki 3.9. Seuraava graafi on suunnattu. Siinä $F = (V,\ E)$, jossa $V = \{a,\ b,\ c,\ d\}$ ja $E = \{(a, b),\ (a, c),\ (b, c),\ (b, d),\ (c, d)\}$



Kuva 1: Suunnattu graafi F

4 Äärellinen automaatti

Tässä tutkielmassa ensimmäiseksi tutkittava malli laskennan mekanisoinnista on äärellinen automaatti. Nimitys tulee siitä, että automaatilla on äärellisen monta mahdollista tilaa. Se on yksinkertaisempi laskentalaite kuin myöhemmin esiteltävä Turingin kone. Erityisesti äärellinen automaatti on hyvä malli laitteille, joilla on pieni määrä muistia. Kieliä, jotka äärellinen automaatti tunnistaa kutsutaan säännöllisiksi kieliksi. Myöhemmin käytetään äärellisen automaatin sijaan nimitystä automaatti tarkoitettaessa samaa asiaa. Äärellisestä automaatista on kaksi eri muotoa, deterministinen ja epädeterministinen. Niistä käsitellään ensiksi deterministinen ja vasta myöhemmin epädeterministinen.

4.1 Deterministinen äärellinen automaatti

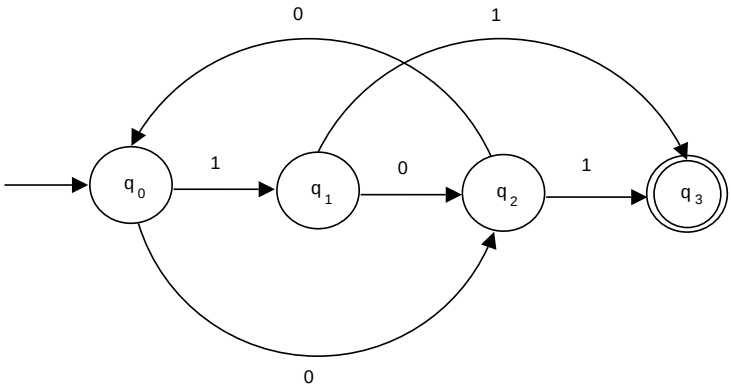
Määritelmä 4.1. (Ks. [9, s. 35]) Deterministinen äärellinen automaatti on viisikko
$(Q, \Sigma, \delta, q_0, F)$, jossa

1. Q on äärellinen *tilajoukko*, sen alkioita sanotaan *tiloiksi*,
2. Σ on äärellinen *aakkosto*,
3. $\delta : Q \times \Sigma \to Q$ on *siirtymäfunktio*,
4. $q_0 \in Q$ on *alkutila* ja
5. $F \subseteq Q$ *hyväksyvien lopputilojen* joukko.

Tilat ovat koodeja, joista kone tietää, mikä siirtymäfunktio otetaan käyttöön. Siirtymäfunktio δ kertoo, millä tavalla automaatti toimii. Esimerkiksi jos tiloilla q_1 ja q_2 ja symbolilla $a \in \Sigma$ pätee $\delta(q_1, a) = q_2$, niin silloin oltaessa tilassa q_1 ja seuraavana merkkinä a, siirrytään tilaan q_2. Äärellinen automaatti käy syötteen läpi kerran merkki merkiltä ja siirtyy jokaisen vaiheen jälkeen seuraavaan merkkiin ja tilaan, kunnes koko syöte on käyty läpi. Jos automaatti on lopussa hvyäksyvässä lopputilassa, automaatti hyväksyy syötteen. Jos taas lopputilana ei ole hyväksyvä lopputila, automaatti hylkää syötteen.

Automaatin toimintaa voidaan kuvata niin kutsutulla tilakaaviolla, joka on graafi tiloista ja niiden järjestyksistä. Siinä siirtymäfunktiot on merkitty nuolilla, tilat ympyröillä. Alkutila ja lopputilat ovat erotettavissa siten, että alkutilaan tulee nuoli graafin ulkopuolelta ja hyväksyvä lopputila on merkitty kaksinkertaisella ympyrällä. Lisäksi jos kuvataan mallia, jossa on hylkääviä lopputiloja, niin ne merkitään ympyröillä, joissa on rasti.

Esimerkki 4.1. Kuvassa 2 on esimerkki tilakaaviosta. Siinä alkutila on q_0 ja hyväksyvien lopputilojen joukkoon kuuluu tässä tapauksessa vain yksi tila, q_3. Siirtymäfunktion arvot näkyvät myös taulukossa.



Kuva 2: Tilakaavio

δ	0	1
q_0	q_2	q_1
q_1	q_2	q_3
q_2	q_0	q_3

Määritelmä 4.2. (Ks. [9, s. 40])Automaatti M *hyväksyy* merkkijonon $w = w_1 w_2 \cdots w_n$, jos tilojen joukosta Q on muodostettavissa tilojenjono eli laskentahistoria $s_0, s_1, \ldots, s_n$, jossa

- $s_0 = q_0$,
- $\delta(s_i, w_{i+1}) = s_{i+1}$, kun $i = 0, \ldots, n-1$ ja
- $s_n \in F$.

Määritelmä 4.3. (Ks. [9, s. 40]) Automaatin M *tunnistama kieli* A on kaikkien niiden merkkijonojen joukko, jotka M hyväksyy. Siis $A = \{w \in \Sigma^* \mid M$ on automaatti ja M hyväksyy merkkijonon $w\}$. Merkitään $A = L(M)$.

Huom! Voidaan samaistaa kielen ja sitä vastaavan päätösongelman tunnistettavuus toisiinsa, kuten myöhemmin tehdään ratkeavuutta tutkittaessa.

Kaksi automaattia *vastaavat* toisiaan eli ovat ekvivalentteja, joss ne tunnistavat saman kielen. Äärellinen automaatti on minimaalinen eli normaalimuodossa, jos se on ekvivalenttien automaattien joukossa tilamäärältään pienin.

Määritelmä 4.4. (Ks. [9, s. 40]) Kieli A on *säännöllinen*, jos jokin äärellinen automaatti tunnistaa sen. Siis jos $A = L(M)$ jollain automaatilla M, niin A on säännöllinen kieli.

Lause 4.2. *Jos A ja B ovat säännöllisiä kieliä, niin myös $A \cup B$ on säännöllinen.*

Todistus. (Ks. [9, s. 46]) Olkoot A ja B säännöllisiä kieliä. Silloin on olemassa äärelliset automaatit, jotka tunnistavat nämä kielet. Olkoot $A = L(M_1)$ ja $B = L(M_2)$ äärellisillä automaateilla $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ ja $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$. Tässä todistuksessa tulisi muodostaa äärellinen automaatti $M = (Q, \Sigma, \delta, q, F)$, jolle pätee $L(M) = A \cup B$.

Uuden automaatin M tulisi hyväksyä syöte w, jos jompikumpi automaateista M_1 tai M_2 hyväksyy syötteen. Automaatteja ei voida suorittaa peräkkäin, koska äärellinen automaatti käsittelee kunkin merkin syötteestä vain yhden kerran, siis jokainen merkki pitää tutkia heti sekä automaatilla M_1 että M_2.

Merkitään automaatin M_1 tilajoukon kokoa $|Q_1| = m$ ja automaatin M_2 tilajoukon kokoa $|Q_2| = n$. Automaatin M tilajoukossa täytyy ottaa molempien automaattien tilajoukot mukaan ja tässä todistuksessa luodaan uusi tilajoukko Q, mitä voidaan havainnollistaa alla olevan taulukon avulla. Automaatin M tilajoukoksi valitaan $Q_1 \times Q_2$ taulukko, jossa rivi esittää automaatin M_1 tilaa ja sarake vastaavasti automaatin M_2 tilaa. Samoin myös siirtymäfunktio δ_1 kertoo, mille riville siirrytään seuraavaksi ja δ_2 mihin sarakkeeseen.

Merkitään tilojen joukkoa $Q_1 = \{q_0^1, q_1^1, q_2^1, \ldots, q_{m-2}^1, q_{m-1}^1\}$ automaatille M_1 ja $Q_2 = \{q_0^2, q_1^2, q_2^2, \ldots, q_{n-2}^2, q_{n-1}^2\}$ automaatille M_2. Siis taulukossa automaatin M_1 aloitustilaa merkitään $q_0^1 = q_1$ ja automaatin M_2 aloitustilaa merkitään $q_0^2 = q_2$

$q_0 = (q_1, q_2)$	(q_0^1, q_1^2)	(q_0^1, q_2^2)	$\cdots$	(q_0^1, q_{n-1}^2)
(q_1^1, q_0^2)	(q_1^1, q_1^2)	(q_1^1, q_2^2)	$\cdots$	(q_1^1, q_{n-1}^2)
(q_2^1, q_0^2)	(q_2^1, q_1^2)	(q_2^1, q_2^2)	$\cdots$	(q_2^1, q_{n-1}^2)
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
(q_{m-1}^1, q_0^2)	(q_{m-1}^1, q_1^2)	(q_{m-1}^1, q_2^2)	$\cdots$	(q_{m-1}^1, q_{n-1}^2)

Täsmällisesti äärellinen automaatti on $M = (Q, \Sigma, \Gamma, q_0, F)$, missä

- Tilojen joukko $Q = Q_1 \times Q_2 = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ ja } r_2 \in Q_2\}$.
- Aakkosto Σ on alkuperäinen eli sama kuin automaateilla M_1 ja M_2. Jos automaateilla olisi eri aakkostot Σ_1 ja Σ_2, niin automaatin M aakkosto olisi $\Sigma = \Sigma_1 \cup \Sigma_2$.
- Siirtymäfunktio muodostuu seuraavasti: Kun r on yhden taulukon solu eli $r = (r_1, r_2) \in Q_1 \times Q_2$ ja $a \in \Sigma$, niin $\delta(r, a) = (s_1, s_2)$, missä s_1 ja s_2 ovat alkuperäisten automaattien M_1 ja M_2 siirtymäfunktiot $s_1 = \delta_1(r_1, a)$ ja $s_2 = \delta_2(r_2, a)$.
- Aloitustila automaatilla M on $q_0 = (q_1, q_2)$.
- Lopetustilojen joukko F koostuu niistä soluista, joissa jompikumpi automaatti M_1 tai M_2 ovat omissa lopetustiloissaan eli $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ tai } r_2 \in F_2\}$.

Siis tarkasteltaessa mielivaltaista merkkijonoa $w = w_1 \cdots w_h$ on olemassa tietty automaatin M_1 laskentaa esittävä tilojenjono $(r_0, \ldots, r_h) \in Q_1^{h+1}$ ja vastaavasti $(s_0, \ldots, s_h) \in Q_2^{h+1}$, missä

- $r_0 = q_1$ ja $s_0 = q_2$ ja
- $r_{i+1} = \delta(r_i, w_{i+1})$ ja $s_{i+1} = \delta(s_i, w_{i+1})$, $i = 0, \ldots, n-1$.

Siis valitsemalla $p_i = (r_i, s_i)$ saadaan automaatin M laskentaa kuvaava tilojenjono, jolla

- $p_0 = q_0$ ja
- $p_{i+1} = \delta(p_i, w_{i+1})$, kun $i = 0, \ldots, n-1$ ja
- $p_n = \delta(p_{n-1}, w_n) \in F$ ja siis $p_n = (r_n, s_n) \in F$

Viimeisestä kohdasta seuraa, että $r_n \in F_1$ tai $s_n \in F_2$ siis M_1 hyväksyy merkkijonon w tai M_2 hyväksyy merkkijonon w. Tällöin saadaan haluttu tulos: $L(M) = A \cup B$. □

Lause 4.3. *Jos A on säännöllinen kieli, niin myös $\overline{A} = \Sigma^* - A$ on säännöllinen.*

Todistus. (Ks. [5, s. 133]) Olkoon $A = L(M_A)$, jollain automaatilla $M_A = (Q, \Sigma, \delta, q_0, F)$. Silloin $\overline{A} = L(\overline{M}_A)$, missä $\overline{M}_A$ on automaatti $\overline{M}_A = (Q, \Sigma, \delta, q_0, Q - F)$. Siis $\overline{M}_A$ on samanlainen kuin M_A, mutta automaatin $\overline{M}_A$ hyväksymistilat ovat automaatin M_A tilat paitsi hyväksymistilat ja päinvastoin. Silloin
$w \in L(\overline{M}_A)$, ja siis automaatin $\overline{M}_A$ viimeiseksi tilaksi jää ei hyväksyvä tila. Tämä taas pätee, joss $w \notin L(A)$. □

Lause 4.4. *Jos joukot A ja B ovat säännöllisiä kieliä, niin myös $A \cap B$ on säännöllinen.*

Todistus. Seuraa suoraan edellisistä lauseista, koska voidaan de Morganin säännöllä kirjoittaa $A \cap B = \overline{\overline{A} \cup \overline{B}}$. □

Lause 4.5. *Jos joukot A ja B ovat säännöllisiä kieliä, niin myös $A - B$ on säännöllinen.*

Todistus. (Ks. [5, s. 137]) Seuraa suoraan edellisistä lauseista, koska voidaan muotoilla $A - B = A \cap \overline{B}$ □

4.2 Epädeterministinen äärellinen automaatti

Kuten kappaleen alussa mainittiin äärellisestä automaatista on kaksi eri muotoa, epädeterministinen ja deterministinen. Deterministisellä äärellisellä automaatilla on olemassa vain yksi mahdollinen tilojen jono eli laskentahistoria tietyn merkkijonon käsittelyssä, kun taas epädeterministisellä mahdollisia tilojen jonoja on useita. Epädeterministisyyden salliminen helpottaa monissa todistuksissa, koska jokaista epädeterminististä automaattia vastaa jokin normaalimuodossa oleva äärellinen automaatti, siksi säännölliset kielet voidaan tunnistaa myös epädeterministisellä äärellisellä automaatilla (Ks. [9, s. 55-56]). Siksi tulokset voidaan yleistää deterministisillekin äärelliselle automaatille. Epädeterministinen äärellinen automaatti esitellään kappaleessa 4.2.

Seuraavat todistukset ovat käytännöllisempiä epädeterministisellä äärellisellä automaatilla, joten seuraavassa sen määritelmä. Epädeterministinen automaatti eroaa deterministisestä siten, että deterministisen automaatin siirtymäfunktion pitää olla funktio, mutta epädeterministisellä automaatilla siirtymä on kuvaus joukolle mahdollisia arvoja. Epädeterministisessä automaatissa myös niin kutsutut ϵ-siirtymät ovat mahdollisia. Niissä ei siirrytä merkkijonolla eteenpäin vaan tila vain muuttuu. Epädeterministisen automaatin sanotaan tunnistavan merkkijonon, jos löytyy yksikin tilasiirtymien reitti, mitä seuraamalla tunnistus onnistuu. Seuraavassa $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ ja potenssijoukon määritelnähän oli $\mathcal{P}(A) = \{B \mid B \subseteq A\}$.

Määritelmä 4.5. (Ks. [9, s. 53]) Epädeterministinen äärellinen automaatti on viisikko $(Q, \Sigma, \delta, q_0, F)$, missä

1. Q on äärellinen *tilajoukko*,
2. Σ on äärellinen *aakkosto*,
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ on *siirtymäkuvaus*, jossa myös ϵ-siirtymät ovat mahdollisia,

4. $q_0 \in Q$ on *alkutila* ja

5. $F \subseteq Q$ *hyväksyvien lopputilojen* joukko.

On todistettu, esimerkiksi lähteessä [9], että jokaiselle epädeterministiselle äärelliselle automaatille löytyy ekvivalentti deterministinen äärellinen automaatti. Siksi säännöllisiä kieliä voidaan tunnistaa myös epädeterministisillä äärellisillä automaateilla, kuten seuraavassa.

Lause 4.6. *Jos A ja B ovat säännöllisiä kieliä, niin myös $A \circ B$ on säännöllinen kieli.*

Todistus. (Ks. [9, s. 60]) Olkoot kielet A ja B säännöllisiä kieliä. Silloin on olemassa epädeterministiset automaatit $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$ ja $N = (Q_2, \Sigma, \delta_2, q_2, F_2)$, joille $L(M) = A$ ja $L(N) = B$. Laaditaan automaatti $K = (Q, \Sigma, \delta, q_1, F_2)$ kielen $A \circ B$ tunnistamiseksi. Automaatissa K ensiksi käydään läpi automaatin M toiminta ja sitten automaatin N toiminta.

1. Automaatin K tilajoukko on $Q = Q_1 \cup Q_2$,

2. alkutila on q_1,

3. hyväksyvien lopputilojen joukko on F_2 ja

4. siirtymät määräytyvät seuraavasti, kun kaikilla $q \in Q$ kaikki $a \in \Sigma_\epsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \text{ ja } q \notin F_1; \\ \delta_1(q,a), & q \in F_1 \text{ ja } a \neq \epsilon; \\ \delta_1(q,a) \cup \{q_2\}, & q \in F_1 \text{ ja } a = \epsilon; \\ \delta_2(q,a), & q \in Q_2. \end{cases}$$

□

Lause 4.7. *Jos A on säännöllinen kieli, niin myös A^* on säännöllinen kieli, missä $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ ja } x_i \in A \text{ kaikilla } i\}$.*

Todistus. (Ks. [9, s. 62]) Olkoon A säännöllinen kieli. Silloin on olemassa epädeterministinen äärellinen automaatti $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$, jolle $L(M) = A$. Laaditaan automaatti $K = (Q, \Sigma, \delta, q_0, F)$ kielen A^* tunnistamiseksi.

1. Automaatin K tilojenjoukko on $Q = \{q_0\} \cup Q_1$,

2. uusi alkutila on q_0,

3. hyväksyvien lopputilojen joukko on $F = \{q_0\} \cup F_1$ ja

4. siirtymät millätahansa tilalla $q \in Q$ ja $a \in \Sigma_{epsilon}$ määräytyvät seuraavasti:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \text{ ja } q \notin F_1; \\ \delta_1(q,a), & q \in F_1 \text{ ja } a \neq \epsilon; \\ \delta_1(q,a)\{q_1\}, & q \in F_1 \text{ ja } a = \epsilon; \\ \{q_1\}, & q = q_0 \text{ ja } a = \epsilon; \\ \emptyset, & q = q_0 \text{ ja } a \neq \epsilon. \end{cases}$$

Tässä todistuksessa siis luodaan uusi automaatin M ulkopuolinen aloitustila q_0, jolla pääsee automaatin K toimintaan ϵ-siirtymällä. Sitten automaatti K toimii simuloiden M, kunnes automaatin ajaminen loppuu hyväksyvään lopputilaan. □

Epädeterministinen äärellinen automaatti luo sillan säännöllisten kielten ja säännöllisten lausekkeiden välille. Säännöllisillä lausekkeilla voidaan esittää tasan ne kielet, jotka voidaan tunnistaa äärellisellä automaatilla, eli säännölliset kielet(kts. [9]). Säänööliset lausekkeet ovat tärkeitä joissain tietoteknisissä sovelluksissa, esimerkiksi sanoja etsivillä hakukoneilla, joten siksi niistä on hyödyllistä tietää jotain.

Aritmeettisen lausekkeen arvo on jokin luku, esimerkiksi lausekkeen $23-7$ arvo on 16. Säännöölisen lausekkeen arvona on aina jokin kieli. Jos R on säännöllinen lauseke, sen esittämää kieltä merkitään $L(R)$. Idea on siis esittää uusia kieliä operaatioiden $\cup, \circ$ ja * avulla. (Ks. [9, s. 63])

Määritelmä 4.6. (Ks. [9, s. 64]) Aakkoston Σ lauseke R on säännöllinen lauseke, jos R on

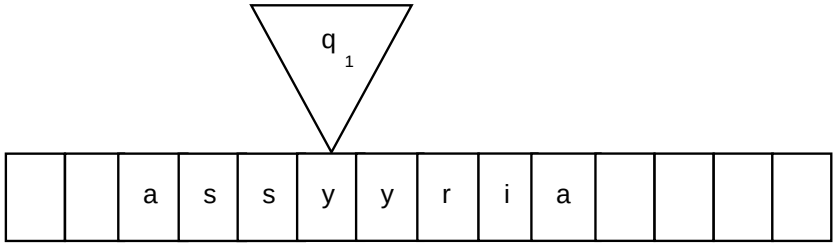
1. a kaikilla $a \in \Sigma$ ja $L(a) = \{a\}$,
2. $\emptyset$ ja $L(\emptyset) = \emptyset$,
3. ϵ ja $L(\epsilon) = \{\epsilon\}$,
4. $(R_1 \cup R_2)$, jossa R_1 ja R_2 ovat säännöllisiä lausekkeita, ja $L(R_1 \cup R_2) = L((R_1) \cup L((R_2)$,
5. $(R_1 \circ R_2)$, jossa R_1 ja R_2 ovat säännöllisiä lausekkeita, ja $L(R_1 \circ R_2) = L((R_1) \circ L((R_2)$,
6. $(R_1)^*$, jossa R_1 on säännöllinen lauseke, ja $L(R_1^*) = L(R_1)^*$ ja

Esimerkki 4.8. (Ks. [6, s. 49]) Säännöllisen lausekkeen $a(b \cup c)^*$ esittämä kieli on
$L(a(b \cup c)^*) = \{a\} \circ (\{b, c\})^* = \{a, ab, ac, abb, abc, acc, abbb, \ldots\}$. Siis kieleen kuulu ne merkkijonot, joiden alussa on a ja sen jälkeen merkkejä b tai c mielivaltaisessa järjestyksessä nollasta äärettömään kappaletta.

5 Turingin kone

Seuraavassa tutkitaan äärellisiä automaatteja tehokkaampaa mallia, nimittäin Alan Turingin vuoden 1936 ehdotusta, jota nykyisin kutsutaan Turingin koneeksi. Se on samantyyppinen kuin äärellinen automaatti, mutta rajattomalla muistillaan Turingin kone on hyvä malli yleistietokoneesta. Turingin kone voi suorittaa kaikki laskennalliset tehtävät, jotka oikeakin tietokone voi. Se ei kuitenkaan voi ratkaista kaikkia ongelmia, kuten ei voi tietokonekaan. Nämä ongelmat ovat laskennallisesti ratkeamattomia, esimerkkejä ratkeamattomista ongelmista löytyy luvusta 7.

Turingin koneen malli käyttää ääretöntä nauhaa ja sillä on muistia siirtymäfunktioita varten. Siinä on lisäksi lukija, joka voi lukea ja kirjoittaa merkkejä, ja liikkua nauhalla oikealle tai vasemmalle tai pysyä paikoillaan. Lukijaa esittää kuvassa kolmion kärki.



Kuva 3: Turingin koneen malli

Turingin koneen määritelmän ydin on siirtymäfunktio δ, koska se kertoo koneen toiminnan kussakin tilanteessa, eli on koneen "ohjelma". Turingin koneessa siirtymäfunktio δ on muotoa $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}$. Tämä tarkoittaa, että kun kone on tietyssä tilassa q ja lukija on ylittänyt nauhan ruudun, jossa on merkki a, ja jos $\delta(q,a) = (r,b,X)$, niin kone kirjoittaa merkin b merkin a tilalle ja menee tämän jälkeen tilaan r. Kolmas osa $X \in \{L, R, S\}$ kertoo sen, liikkuuko lukija oikealle, vasemmalla vai pysyykö paikallaan. Monesti näkee merkinnöille L, R ja S käytettävän myös visuaalisempia merkintöjä $\leftarrow$, $\rightarrow$ ja $-$. Syöte on alussa nauhalla ja lukija on syötteen ensimmäisen merkin kohdalla. Niissä ruuduissa, joissa ei ole syötettä on tyhjänruudunmerkki $\sqcup$.

On olemassa Turingin koneen malleja, joissa lukija ei voi pysy paikallaan. Sellaiset koneet voidaan kuitenkin muokata paikallaan pysyväksi, kun luodaan ohjelman siihen kohtaan ylimääräinen tila, jossa tehdään tarvittava edestakainen nauhalla siirtyminen. Jossain Turingin koneissa nauhalle kirjoittaminen ja liikkuminen ovat vaihtoehtoisia toimintoja, mutta silloin kirjoittaminen ja siirtyminen ovat peräkkäisissä siirtymäfunktioissa, kun tässä mallissa ne ovat samassa. Tästä eteenpäin tässä tutkielmassa oletetaan kuitenkin, että kone voi olla paikallaan sekä siirtyä että kirjoittaa samassa käskyssä.

Määritelmä 5.1. (Ks. [9, s. 128]) Turingin kone on seitsikko (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}), missä Q, Σ ja Γ ovat äärellisiä joukkoja ja

1. Q on *tilojen* joukko,
2. Σ on *syöteaakkosto*, jolle $\sqcup \notin \Sigma$,
3. Γ on *nauha-aakkosto*, jossa $\sqcup \in \Gamma$ ja $\Sigma \subseteq \Gamma$,
4. $\delta : Q' \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}$ on *siirtymäfunktio*, missä $Q' = Q \setminus \{q_{acc}, q_{rej}\}$
5. $q_0 \in Q$ on *alkutila*,
6. $q_{acc} \in Q$ on *hyväksymistila* eli lopputila, jossa syöte hyväksytään ja
7. $q_{rej} \in Q$ on *hylkäämistila*, jossa $q_{acc} \neq q_{rej}$.

Turingin kone $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ laskee seuraavasti. Aluksi M vastaanottaa syötteen $w = w_1 w_2 \dots w_n \ni \Sigma^*$ eli merkkijono w täyttää nauhalta n ruutua ja loput ruudut jäävät tyhjiksi, eli niissä on tyhjänruudunmerkki $\sqcup$. Lukija aloittaa nauhan merkkien lukemisen syötteen vasemmasta reunasta. Pitää kuitenkin huomata, että Σ ei sisällä tyhjän ruudun symbolia, joten ensimmäinen tyhjä ruutu merkitsee syötteen loppumista. Koska tutkielmassa käsitellään Turingin konetta matemaattisesti, voidaan olettaa nauhan olevan äärettömän pitkä kumpaankin suuntaan, joten nauhan reunan yli siirtymistä ei tarvitse ottaa huomioon.

Turingin koneen laskiessa tapahtumat muuttuvat tilan, nauhan sisällön ja lukijan sijainnin mukaan. Tätä kolmen asian kokonaisuutta kutsutaan Turingin koneen *tilanteeksi*. Tiettyä tilannetta merkitsemme seuraavasti. Nykyinen tila on q ja nauhalla on merkkijono $u = w_1 w_2 w_3 w_4 w_5 w_6$, jossa $w_k \in \Sigma$. Merkkijonon molemmilla puolilla on tyhjänruudunmerkkejä $\sqcup$. Jos lukija on merkin w_1 kohdalla, niin tilannetta merkitään qu tai yksityiskohtaisemmin $qw_1w_2w_3w_4w_5w_6$. Jos lukija on merkin w_4 kohdalla, merkitään $w_1w_2w_3\ q\ w_4w_5w_6$.

Määritelmä 5.2. (Ks. [9, s. 129]) Oletetaan, että a, b ja c kuuluvat nauhaaakkostoon Γ ja u ja v ovat joukon Γ^* eräät merkkijonot ja q_i ja q_j ovat tiloja. Tässä tapauksessa uaq_ibv ja uq_jacv ovat kaksi tilannetta. Tällöin sanotaan, että uaq_ibv *johtaa suoraan* tilanteeseen uq_jacv ja sitä merkitään

$$uaq_ibv \vdash uq_jacv \text{ tai } C_k \vdash C_l. \text{ (Ks. [5, s. 320])}$$

Tässä siirtymäfunktiolle pätee $\delta(q_i, b) = (q_j, c, L)$. Oikealle siirryttäessä merkitään vastaavasti. Kun kone M aloittaa toimintansa, laskentaprosessi etenee siirtymäfunktion sääntöjen mukaan. Laskeminen jatkuu mahdolliseen hyväksymis- tai hylkäämistilaan q_{acc} tai q_{rej} saakka, jossa se sitten pysähtyy.

Jos kumpaakaan tilannetta ei synny, laskenta jatkuu loputtomiin eli kone jää luuppiin. Seuraava esimerkki on laskennallisen ongelman Turingin kone eli vastaus on jokin lukuarvo. Samalla systeemillä voitaisiin rakentaa Turingin kone jollekin päätösongelmalle, jossa vastaus on joko 1 tai 0. Hyväksi esimerkiksi sopisi aiemminkin käsitelty ”suurempi kuin” esimerkki 3.7.

Esimerkki 5.1. Turingin kone $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, jossa $Q = \{q_0, q_1, q_2, \ldots, q_n, q_{acc}, q_{rej}\}$, $\Sigma = \{0, 1\}$ ja siirtymäfunktio δ on oheisen taulukon mukainen, ratkaisee seuraavan ongelman.

Jos halutaan laskea funktio $f(x) = x + 3$, kun $x \in \mathbb{N}$, voidaan tehdä se Turingin koneella. Tässä $\Sigma = \{0, 1\}$ ja nauhalla on ykkösten jono, joka vastaa lukua x. Ykkösiä on yksi enemmän kuin luku itse, jotta voidaan erottaa 0 tyhjästä syötteestä ja laskea $f(0) = 0 + 3$. Luvun x koodaus merkkijonoksi on merkin 1 katenaatio itsensä kanssa $x + 1$ kertaa eli 1^{x+1}. Koodaus siis on $\mathbb{N} \to \{0, 1\}^* : h(x) = \underbrace{1 \ldots 1}_{x+1 kpl}$.

Seuraavassa Turingin koneen käskyjen jono, joka tuottaa halutun ratkaisun. Lukija on alussa mahdollisen syötteen ensimmäisen merkin kohdalla.

$\delta(q_0, 1) = (q_0, 1, R)$
$\delta(q_0, \sqcup) = (q_1, 1, R)$
$\delta(q_1, \sqcup) = (q_2, 1, S)$

Ja lopussa nauhalla on siis $x + 3$ kappaletta merkkejä 1 eli katenaatio 1^{x+3}. Tässä vaiheessa aiempaa $x+1$ koodausta ei enää tarvita, koska luku on aina suurempi kuin 0. Turingin kone M_1 siis ratkaisee funktion $f(x)$, koska ohjelma $M_1(x) \downarrow (x + 3)$, siis $M_1(x) = f(x)$, kun $x \in Dom(f)$.

Turingin koneen M aloitustilanne syötteellä w on $q_0 w$, josta nähdään, että kone on alussa tilassa q_0 ja koneen lukupää on syötteen vasemmassa reunassa. Hyväksymis- tai hylkäämistilaan tullessaan Turingin kone pysähtyy.

Turingin kone M hyväksyy syötteen w, jos on sellainen tilanteiden C_1, C_2, …, C_k jono eli laskentahistoria, jossa

1. C_1 on koneen M aloitustilanne syötteellä w,
2. jokaisella i tilanne C_i johtaa tilanteeseen C_{i+1} ja
3. C_k on hyväksymistilanne.

Määritelmä 5.3. *Hyväksyvä laskentahistoria* on Turingin koneen tilanteiden jono, joka päättyy hyväksymistilaan ja *hylkäävä laskentahistoria* on tilanteiden jono, joka päättyy hylkäämistilaan.

Äärellisillä automaateilla syötteen hyväksyminen vaatii vastaavasti laskentahistorian, joka alkaa aloitustilasta ja päättyy hyväksymistilaan. Turingin koneen tapauksessa lukija voi liikkua eri suuntiin ja syötteessä olevat merkit vaikuttavat liikkeeseen ja tilojen jonoon, joten siksi pelkkä tilojen

tarkastelu ei riitä. Äärellisillä automaateilla merkkijonojen joukkoa, jotka automaatti hyväksyy, kutsutaan automaatin kieleksi, samalla tavoin kutsutaan myös Turingin koneiden hyväksyvien merkkijonojen joukkoa Turingin koneen kieleksi.

Määritelmä 5.4. Merkkijonojen kokoelma A, jotka Turingin kone M hyväksyy, sanotaan *Turingin koneen M kieleksi*, merkitään sitä $A = L(M)$. Siis Kieli A on $L(M) = \{w \in \Sigma^* \mid M \text{ hyväksyy merkkijonon } w\}$.

Määritelmä 5.5. Sanotaan että kieli on *tunnistettava*, jos jokin Turingin kone hyväksyy sen. Siis kieli A on tunnistettava, jos on olemassa Turingin kone M, jolle $A = L(M)$.

Tunnistettaville kielille täytyy siis löytyä Turingin kone, joka hyväksyy kaikki kielen merkkijonot ja vain ne. Tunnistettavia kieliä kutsutaan toisinaan Turing-tunnistettavien kielten ja osittain ratkeavien kielten lisäksi myös rekursiivisesti numeroituviksi tai RE-kieliksi. Nimitys tulee siitä, että Turingin kone, jota kutsutaan luettelijaksi, tai enumerator, pystyy tulostamaan kaikki tunnistettavan kielen merkkijonot. Luettelija toimii tavallisen Turingin koneen tapaan, mutta syöte on aina tyhjä merkkijono ja aika ajoin kone voi tulostaa aakkoston Σ^* merkkijonon. Luettelijoihin emme tässä työssä tämän enempää paneudu.

Esimerkki 5.2. Tarkastellaan kielen $A = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ tunnistamista. Tässä esimerkissä merkitseminen tarkoittaa merkin k muuttamista merkiksi k', kaikilla $k \in \{a, b, c\}$. Siis merkitsemätön on k ja merkitty on k'. Turingin kone, joka tunnistaa kyseisen kielen on seuraava.

Kone *M*

Merkkijono syötteellä w toimitaan seuraavasti:

1. Kelataan oikealle, jotta löytyy merkitsemätön merkki a,
   - jos löytyy, merkitään se ja siirrytään oikealle etsimään merkitsemättömiä merkkejä b.
   - jos ei löydy enää merkkiä a, mennään viimeiseen vaiheeseen, jossa tarkistetaan, onko kaikki mahdolliset syötteen merkit merkitty.
2. Kelataan oikealle, jotta löytyy merkitsemätön merkki b,
   - jos löytyy, merkitään se ja siirrytään oikealle etsimään merkitsemättömiä merkkejä c.
   - jos ei löydy enää, syötteessä oli ainakin yksi a enemmän kuin merkkejä b. Hylätään siis syöte.
3. Kelataan oikealle, jotta löytyy merkitsemätön merkki c,

- jos löytyy, merkitään se ja kelataan syötteen alkuun ja aloitetaan uudestaan syötteen tarkistus alusta.
- jos ei löydy enää, syötteessä oli ainakin yksi a ja b enemmän kuin merkkejä c. Hylätään siis syöte.

4. Viimeisessä vaiheessa kelataan syötteen alkuun, ja samalla jos matkalla huomataan merkitsemättömiä kirjaimia, hylätään. Kun sitten ollaan alussa käydään syöte merkki merkiltä läpi loppuun asti,
   - jos löytyy merkitsemättömiä kirjaimia, hylätään
   - jos kaikki kirjaimet on merkitty, niitä on yhtä monta, joten hyväksytään.

Täsmällisesti Turingin koneen siirtymäfunktiot ovat:

Ensimmäinen vaihe
$\delta(q_0, a') = (q_0, a', R)$
$\delta(q_0, a) = (q_1, a', S)$
$\delta(q_0, \sqcup) = (q_4, \sqcup, S)$
$\delta(q_0, b) = (q_4, b, S)$
$\delta(q_0, b') = (q_4, b', S)$
$\delta(q_0, c) = (q_4, c, S)$
$\delta(q_0, c') = (q_4, c', S)$
Toinen vaihe:
$\delta(q_1, b) = (q_1, b', R)$
$\delta(q_1, b') = (q_2, b', S)$
$\delta(q_1, \sqcup) = (q_{rej}, \sqcup, S)$
$\delta(q_1, c) = (q_{rej}, c, S)$
$\delta(q_1, c') = (q_{rej}, c', S)$
Kolmas vaihe:
$\delta(q_2, c') = (q_2, c', R)$
$\delta(q_2, c) = (q_3, c', S)$
$\delta(q_2, \sqcup) = (q_{rej}, \sqcup, S)$
Kelataan alkuun:
$\delta(q_3, 1) = (q_0, \sqcup, S)$
$\delta(q_3, a) = (q_3, a, L)$
$\delta(q_3, a') = (q_3, a', L)$
$\delta(q_3, b) = (q_3, b, L)$
$\delta(q_3, b') = (q_3, b', L)$
$\delta(q_3, c) = (q_3, c, L)$
$\delta(q_3, c') = (q_3, c', L)$
Viimeinen vaihe:
$\delta(q_4, a) = (q_{rej}, a, L)$
$\delta(q_4, a') = (q_4, a', L)$
$\delta(q_4, b) = (q_{rej}, b, L)$
$\delta(q_4, b') = (q_4, b', L)$
$\delta(q_4, c) = (q_{rej}, c, L)$
$\delta(q_4, c') = (q_4, c', L)$
$\delta(q_4, \sqcup) = (q_5 \sqcup, S)$
Tarkistus:
$\delta(q_5, a) = (q_{rej}, a, R)$
$\delta(q_5, a') = (q_3, a', R)$
$\delta(q_5, b) = (q_{rej}, b, R)$
$\delta(q_5, b') = (q_3, b', R)$
$\delta(q_5, c) = (q_{rej}, c, R)$
$\delta(q_5, c') = (q_3, c', R)$
Hyväksyminen
$\delta(q_5, \sqcup) = (q_{acc}, \sqcup, S)$

Esimerkki 5.3. (Ks. [9, s. 131])

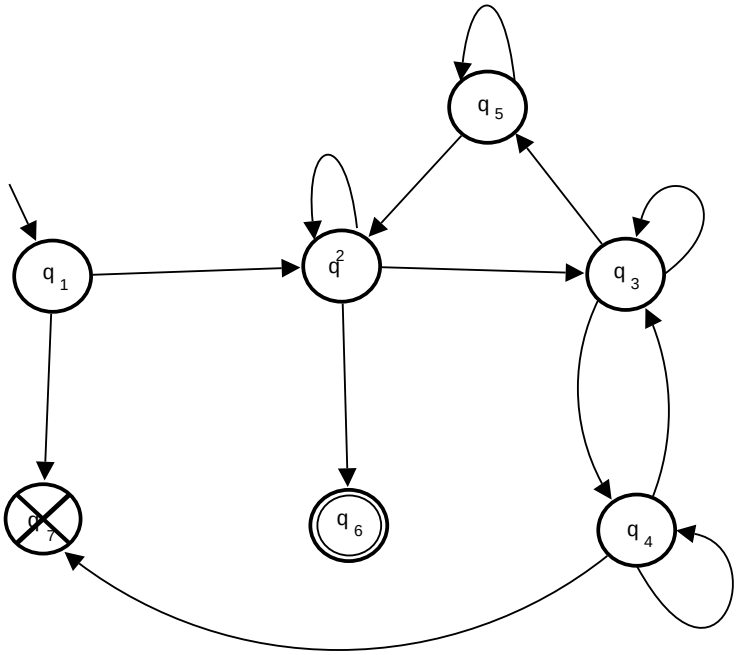
Määritellään Turingin kone M_1, joka tunnistaa kielen, joka koostuu kaikista $w \in \{0\}^*$ merkkijonoista, joiden pituus on kahden potenssi. Kieli on siis $A = \{0^{2^n} \mid n \geq 0\}$.

> **Kone** M_1
>
> Merkkijono syötteellä w toimitaan seuraavasti:
>
> 1. Kelataan nauha vasemmalta oikealle kaikkien nollien yli, pyyhitään pois joka toinen 0.
> 2. Jos ensimmäisessä tilassa löydetään vain yksi 0, hyväksytään.
> 3. Jos ensimmäisessä tilassa löytyy enemmän kuin yksi 0 ja nollien lukumäärä on pariton, hylkää.
> 4. Palautetaan lukija syötteen vasempaan reunaan.
> 5. Mennään ensimmäiseen vaiheeseen.

Täsmällinen määritelmä koneelle M_1:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$, missä q_1 on aloitustila, $q_6 = q_{acc}$ hyväksymistila ja $q_7 = q_{rej}$ hylkäämistila.
- $\Sigma = \{0\}$
- $\Gamma = \{0, 1, \sqcup\}$
- Siirtymäfunktiot saadaan oheisesta taulukosta ja ohessa myös tilakaavion hahmotelma kuvassa 4.

$\delta(q_n, a)$	(q_m, b, L)
$\delta(q_1, 0)$	$(q_2, \sqcup, R)$
$\delta(q_1, \sqcup)$	$(q_7, \sqcup, R)$
$\delta(q_1, x)$	(q_7, x, R)
$\delta(q_2, x)$	(q_2, x, R)
$\delta(q_2, 0)$	(q_3, x, R)
$\delta(q_2, \sqcup)$	$(q_6, \sqcup, R)$
$\delta(q_3, x)$	(q_3, x, R)
$\delta(q_3, 0)$	$(q_4, 0, R)$
$\delta(q_3, \sqcup)$	$(q_5, \sqcup, L)$
$\delta(q_4, x)$	(q_4, x, R)
$\delta(q_4, 0)$	(q_3, x, R)
$\delta(q_4, \sqcup)$	$(q_7, \sqcup, R)$
$\delta(q_5, x)$	(q_5, x, L)
$\delta(q_5, 0)$	$(q_5, 0, L)$
$\delta(q_5, \sqcup)$	$(q_2, \sqcup, R)$



Kuva 4: Esimerkin 5.3 tilakaavio

Esimerkki 5.4. (Ks. [9, s. 134])

Seuraava Turingin kone M_2 tunnistaa kielen $C = \{a^i b^j c^k \mid i \cdot j = k \text{ ja } i, j, k \geq 1\}$. Tässäkin esimerkissä merkitseminen tarkoittaa merkin k muuttamista merkiksi k', kaikilla $k \in \{a, b, c\}$. Siis merkitsemätön on merkki k ja merkitty on merkki k'.

Kone M_2

Merkkijonosyötteellä w toimitaan seuraavasti:

1. Tarkistetaan syöte vasemmalta oikealle, onko se muotoa $a^i b^j c^k$, jos ei ole, hylätään.
2. Palautetaan lukija syötteen vasempaan reunaan.
3. Yliviivataan vasemmanpuoleisin a ja siirrytään seuraavaan b merkkiin. Kelataan sitten edes takaisin b ja c osien väliä ja merkitään aina yksi b ja yksi c. Kun kaikki b merkit on merkitty siirrytään kohtaan 4. Jos taas c merkit on merkitty ennen b merkkejä, c merkkejä oli liian vähän, siis hylätään syöte.
4. Poistetaan merkinnät kaikista b merkeistä. Jos a merkkejä on jäljellä, palataan kohtaan 3. Jos ei ole, tarkistetaan, onko kaikki c merkit merkitty. Jos on hyväksytään, muuten hylätään.

Selvästi edellinen algoritmi voidaan esittää Turingin koneen avulla.

5.1 Moninauhainen Turingin kone

Moninauhainen Turingin kone on Turingin kone usealla nauhalla. Jokaisella nauhalla on oma lukija, joka voi lukea nauhaa tai kirjoittaa sille. Alussa syöte on nauhalla 1 ja muut nauhat ovat täynnä tyhjiä ruutuja $\sqcup$. Siirtymäfunktio on muuttunut siten, että se ohjaa lukemista, kirjoittamista ja siirtymistä kaikilla nauhoilla samanaikaisesti. Täsmällisesti se on muotoa $\delta : Q' \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$, missä k on nauhojen lukumäärä ja Q' ei sisällä hyväksymis- eikä hylkäämistilaa.

Esimerkiksi siirtymäfunktio $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, X_1, \ldots, X_k)$, missä $X_r \in \{L, R, S\}$ kaikilla r, tarkoittaa, että jos kone on tilassa q_i ja lukija on lukenut merkit $a_1, \ldots, a_k$ nauhoilta (merkin a_1 nauhalta 1 ja niin edelleen), niin kone menee tilaan q_j ja kirjoittaa merkit $b_1, \ldots, b_k$ nauhoille ja liikuttaa jokaista lukijaa oman ohjeensa mukaan.

Moninauhainen Turingin kone vaikuttaa tehokkaammalta kuin tavallinen Turingin kone, mutta itse asiassa tavallisella Turingin koneella voidaan toteuttaa kaikki ohjelmat, jotka voidaan toteuttaa moninauhaisellakin koneella. Moninauhainen Turingin kone voidaan muuttaa yksinauhaiseksi Turingin koneeksi esimerkiksi siten, että erillisten nauhojen sisällöt kirjoitetaan samalle nauhalle ja ”lukijoiden” kohdat merkitään täplällä. Moninauhaisen Turingin koneen siirtymäfunktio jaetaan peräkkäisiksi siirtymäfunktioiksi, joita toteutetaan jokaisessa nauhan osiossa. Kun kaikkien osioiden toiminnot on tehty täplän siirtoa myöten, aloitetaan seuraavan moninauhaisen siirtymäfunktion toteuttaminen tavallisten siirtymäfunktioiden avulla (Ks. [9, s. 137]).

Lause 5.5. *Kieli on tunnistettava, joss jokin moninauhainen Turingin kone tunnistaa sen.*

Todistus. (Ks. [9, s. 138]) Koska moninauhainen Turingin kone voidaan esittää yksinauhaisena Turingin koneena, väite seuraa tunnistettavuuden määritelmästä. □

5.2 Epädeterministinen Turingin kone

Epädeterminismi tarkoittaa, että siirtymäfunktio antaa joukon mahdollisia seuraavia tiloja eikä yhtä tiettyä tilaa kuten deterministisillä Turingin koneilla. Siis siirtymäfuktio voi saada useita eri arvoja samoilla muuttujan arvoilla. Tässä käytetään Turingin koneen mallia, jossa on nauhalla siirtymisen lisäksi paikallaan pysyminen, joka siis voitaisiin toteuttaa myös peräkkäisillä oikealle ja vasemmalle siirtymisillä ja mahdollisesti tarvittavalla uudella tilalla.

Määritelmä 5.6. Epädeterministinen Turingin kone on Turingin kone, jonka siirtymäfunktio on muotoa $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$. Tässä siirtymäfunktio ei ole funktio.

Siis epädeterministinen Turingin kone hyväksyy syötteen, jos jokin mahdollinen laskenta johtaa hyväksymistilaan. Mahdollinen laskenta tarkoittaa vastaavaa kuin epädeterministisellä äärellisellä automaatilla, mutta nyt siirtymäfunktion arvo on kolmikko, johon kuuluu tila merkki ja siirtymiskäsky. Epädeterministisen Turingin koneen siirtymäfunktio ei ole siis yksikäsitteinen tiettyä tilaa ja nauhan merkkiä kohtaan, vaan samalla tilalla ja nauhan symbolilla on useita mahdollisia siirtymäfunktioita. On huomattava myös, että kun $A = L(N)$, missä N on epädeterministinen Turingin kone, niin

- jos $w \in A$, niin ainakin yksi laskenta päättyy hyväksymistilaan ja muut laskennat voivat päätyä silmukkaan tai hylkäämistilaan;
- jos taas $w \notin A$, niin mikään laskenta ei pääty hyväksymistilaan vaan silmukkaan tai hylkäämistilaan.

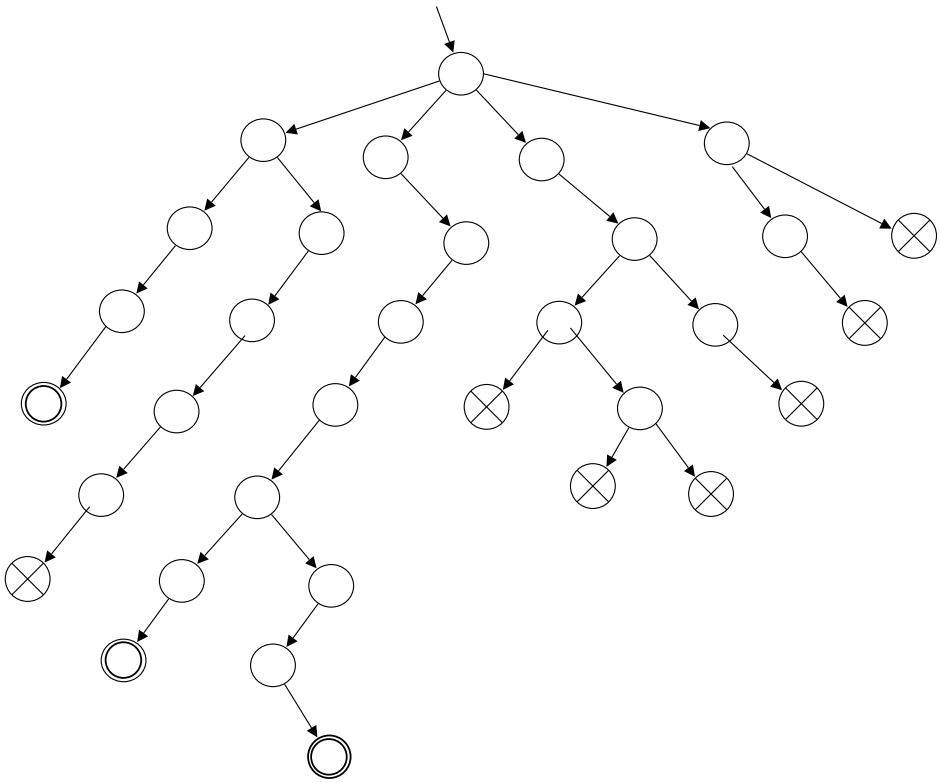
Epädeterministisen Turingin koneen tilakaavio voidaan esittää puumallin avulla, jossa alkutila on keskellä ylhäällä ja mahdolliset tilojen jonot näkyy puun haaroina ja haarat päättyvät joko hylkäävään tilaan (ruksilliset ympyrät) tai hyväksyvään tilaan (kaksoisreunalliset ympyrät). Kuvassa 5 on puurakenteesta esimerkki, siitä tosin puuttuu kaikki myöhemmästä tilasta aiempaan tilaan tilasiirtymät.

Esimerkki 5.6. Kieli $B = \{ww \mid w \in \{0,1\}^*\}$ voidaan tunnistaa epädeterministisellä Turingin koneella S, joka toimii seuraavalla tavalla:

> **Kone** S
>
> Syötteellä ww, jossa w on merkkijono, toimitaan seuraavasti:
>
> 1. Laitetaan ensimmäinen merkki muistiin ja pyyhitään alkuperäinen pois.
> 2. Valitaan epädeterministisesti jokin merkki syötteestä. Jos se ei ole sama kuin ensimmäinen merkki, hylätään, muussa tapauksessa yliviivataan merkki.

3. Palataan syötteen alkuun.
4. Laitetaan muistiin syötteen seuraava merkki ja pyyhitään alkuperäinen pois.
5. Selataan nauhaa oikealle yliviivattujen symbolien yli. Jos niiden jälkeen vastaan tulee ⊔-merkki mennään viimeiseen kohtaan. Jos taas merkki ei ole sama kuin muistissa oleva, hylätään, muuten yliviivaa ja palataan takaisin kohtaan 3.
6. Jos kaikki syötteen merkit ovat yliviivattuja, hyväksytään syöte, muuten hylätään.

Koska epädeterministinen Turingin kone hyväksyy syötteen, jos jokin mahdollinen laskenta johtaa hyväksymistilaan, jossain vaiheessa oikea keskikohta löytyy ja kone pääsee loppuun. Riittää siis arvata ja tarkistaa. Kieli voitaisiin tunnistaa myös deterministisellä koneella, mutta siinä on etsittävä ensiksi syötteen keskikohta ja sen jälkeen vasta vertailtava. Syötteen keskikohdan voisi etsiä esimerkiksi muuttamalla yksitellen syötteen ensimmäisen ja viimeisen merkin ja sitten kun ollaan keskikohdassa voidaan tutkia olivatko alkuperäiset samat. Esimerkiksi alussa merkit 0 ja 1 voidaan muuttaa merkeiksi a ja b ja lopusta päin merkeiksi A ja B.



Kuva 5: Puumalli

Syötteestä

Turingin koneen syöte on aina merkkijono. Jos halutaan antaa syötteenä joku muu kuin merkkijono, joudutaan se muuttamaan jollain tavalla merkkijonomuotoon. Merkintätapa objektin O koodaamiselle merkkijono muotoon on $\langle O \rangle$. Jos on useita objekteja $O_1, O_2, \ldots, O_k$ ne kirjoitetaan koodiksi yksikäsitteisesti $\langle O_1, O_2, \ldots, O_k \rangle$. Koodauksen voi tehdä monella erilaisella tavalla. Tavan valinnalla ei ole väliä.

Esimerkki 5.7. (Ks. [9, s. 145-147]) Olkoon A kaikkien suuntaamattomien yhtenäisten graafien kielten joukko. Graafi on yhtenäinen, jos jokaisesta solmusta pääsee mihin tahansa toiseen solmuun, ainakin yhtä polkua pitkin. Polku tarkoitti yhtenäisten kaarien reittiä jostain solmusta johonkin toiseen. Kirjoitamme

$$\langle A \rangle = \{\langle G \rangle \mid G \text{ on yhdistetty suuntaamaton graafi}\}.$$

Seuraavassa on määritelmä Turingin koneelle M, joka ratkaisee kielen A. Tässä kuten aiemminkin merkitseminen tarkoittaa merkin k muuttamista merkiksi k', kaikilla solmuilla k.

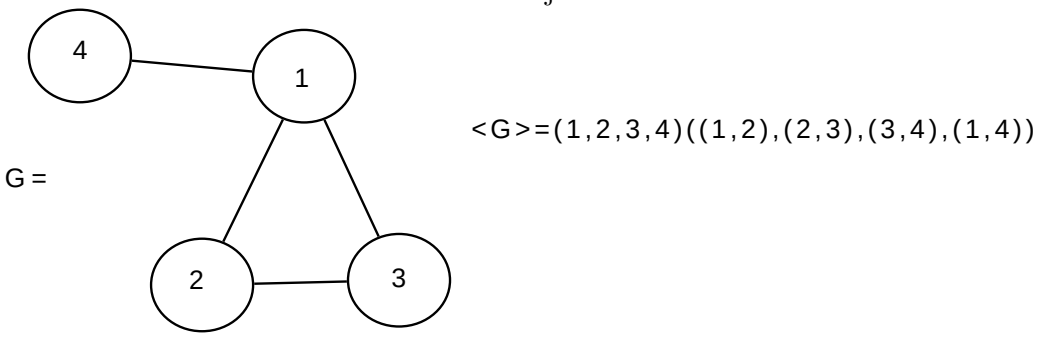
Kun kone M vastaanottaa syötteen $\langle G \rangle$, se ensiksi tarkistaa, onko syöte varmasti jokin graafi. Tehdessään näin, M käy yksitellen läpi koko nauhan. Kone tarkistaa, että syöte koostuu kahdesta listasta ja listat ovat oikeissa muodoissa. Ensimmäisen listan pitäisi olla erillisiä desimaalilukuja ja toisen desimaalilukupareja. Lisäksi kone M tarkistaa, että solmujen lista ei sisällä toistoja ja, että jokainen kaarilistassa oleva solmu on myös solmujen listalla.

Jos syöte w läpäisee nämä tarkistukset, se on jonkin graafin G koodaus. Tämä varmennus lopettaa syötteen tutkimisen ja M menee vaiheeseen 1.

> **Kone** *M*
>
> Syötteellä $\langle G \rangle$, joka on graafin G koodaus merkkijonoksi, toimitaan seuraavasti:
>
> 1. Valitaan graafin G ensimmäinen solmu ja merkitään se.
> 2. Toistetaan seuraavaa niin kauan, ettei ole enää jäljellä merkitsemättömiä solmuja.
> 3. Merkitään jokainen solmu joukossa G, joka on samassa kaaressa merkityn kanssa. Siis jokaiselle merkitylle tehdään sama.
> 4. Tarkistetaan kaikki joukon G solmut, määrittääksesi onko ne kaikki merkitty. Jos on, *hyväksytään*, jos ei *hylätään*.

Ensinnä tulee ymmärtää, kuinka $\langle G \rangle$ koodaa graafin G merkkijonoksi. Tarkastellaan koodaamista joukon G solmujen listana, jota seuraa sen kaarien

Kuva 6: Graafi G ja sen koodaus



lista. Jokainen solmu on luonnollinen luku ja jokainen kaari on lukupari, jossa luvut vastaavat solmuja kaaren kummassakin päässä. Seuraava kuvio kuvaa graafia ja sen koodausta.

5.3 Ratkeavuus

Tässä kappaleessa otetaan selvää Turingin koneen tehokkuudesta ratkaistaessa ongelmia. Käynnistettäessä jokin Turingin kone, se voi johtaa hyväksymis- tai hylkäämistilaan tai ikuiseen silmukkaan eli luuppiin. Luupissa kone ei pysähdy koskaan. Turing kone M voi jättää hyväksymättä syötteen hylkäämällä eli menemällä q_{rej} tilaan tai jatkamalla koneen toimintaa ikuisesti. Yleensä pidetään parempana koneita, jotka seisahtuvat kaikilla syötteillä eivätkä mene luuppiin. Näitä koneita kutsutaan *ratkaisijoiksi*, koska ne tekevät aina hyväksyvät tai hylkäävät syötteen. Jos ratkaisija tunnistaa, mitkä merkkijonot kuuluvat kieleen ja mitkä eivät, niin ratkaisija *ratkaisee* kyseisen kielen. Ratkaisijoita kutsutaan myös totaalisiksi Turingin koneiksi, koska kaikilla syötteillä laskenta pysähtyy. Seuraavassa ratkeavuuden käsite täsmällisesti määriteltynä.

Määritelmä 5.7. Jos Turingin kone M pysähtyy kaikilla mahdollisilla syötteillä, M on *ratkaisija*.

Määritelmä 5.8. Jos Turingin kone M on ratkaisija ja kieli $A = L(M)$, niin Turingin kone M *ratkaisee* kielen A. Lisäksi sanotaan, että kieli A on *ratkeava*, jos sillä on ainakin yksi ratkaisija.

Päätösongelmaa sanotaan osittain ratkeavaksi kielen ollessa tunnistettava, ja kun kieli on ratkeava, niin vastaavaa päätösongelmaa sanotaan ratkeavaksi. Ratkeavista kielistä käytetään usein myös nimitystä rekursiiviset kielet tai Turing-ratkeavat kielet.

Lause 5.8. *Kaikki ratkeavat kielet ovat tunnistettavia. Eli jos kieli A on ratkeava, se on myös tunnistettava.*

Todistus. Selvästi nähdään, että kone M joka ratkaisee kielen pystyy myös määrittämään, mitkä merkkijonot kuuluvat kieleen ”kyllä” tai 1 tulosten perusteella. □

Lause 5.9. *Jos kielet A ja B ovat ratkeavia, myös $\overline{A}$, $A \cup B$ ja $A \cap B$ ovat ratkeavia.*

Todistus. 1. ”$\overline{A}$ on ratkeava”(Ks. [7, s. 61]).

Koska kieli A on ratkeava, niin on olemassa Turingin kone M, joka ratkaisee kielen A. Merkintä $M(x)$ tarkoitti, koneen suorittamista merkkijono syötteellä x.

$$M(x) = \begin{cases} 1 & , \text{jos } x \in A \\ 0, & , \text{jos } x \notin A \end{cases}$$

Helposti voidaan muotoilla Turingin kone $\overline{M}$, joka ratkaisee kielen $\overline{A}$, vaihtamalla lukujen 0 ja 1 paikkaa.

$$\overline{M}(x) = \begin{cases} 0 & \text{, jos } x \in A \text{ eli } x \notin \overline{A} \\ 1 & \text{, jos } x \notin A \text{ eli } x \in \overline{A} \end{cases}$$

2. "$A \cup B$ on ratkeava."

   Merkitään M_A ja M_B koneita, jotka tunnistavat kielet A ja B. Koneet voidaan yhdistää siten, että ensiksi tutkitaan, kuuluuko syöte joukkoon A koneella M_A, jos kuuluu, hyväksytään. Jos taas ei kuulu, tutkitaan kyseinen syöte koneella M_B, eli tutkitaan, onko merkkijono x kielen B alkio. Jos on, hyväksytään ja, jos ei ole hylätään lopullisesti. Käytännössä tämä tarkoittaa sitä, että koneen M_A hylkäämistila muutetaan koneen M_B aloitustilaksi. Tässä on tietenkin huomioitava, ettei koneilla ole muita samannimisiä tiloja.

3. "$A \cap B$ on ratkeava."

   Koska kaava $A \cap B$ voidaan kirjoittaa muotoon $A \cap B = \overline{\overline{A} \cup \overline{B}}$, todistus seuraa edellisistä kohdista.

□

Lause 5.10. *Kieli $A \in \Sigma^*$ on ratkeava, joss A ja $\overline{A}$ ovat tunnistettavia.*

Todistus. (Ks. [7, s. 61])

"⇒"Oletetaan, että A on ratkeava. Edellisestä lauseesta saadaan, että myös $\overline{A}$ on ratkeava. Koska kaikki ratkeavat kielet ovat tunnistettavia, ovat molemmat $\overline{A}$ ja A tunnistettavia.

"⇐" Oletetaan, että kielet $\overline{A}$ ja A ovat tunnistettavia. Merkitään M_A ja $M_{\overline{A}}$ koneita, jotka tunnistavat ne. Koska kaikilla merkkijonoilla $x \in \Sigma^*$ pätee, että $x \in A$ tai $x \in \overline{A}$, niin kaikilla merkkijonoilla x jompi kumpi koneista M_A ja $M_{\overline{A}}$ pysähtyy. Koneet voidaan kytkeä rinnakkain toimimaan samanaikaisesti, jolloin saadaan aikaiseksi kielelle A ratkaisija.

Kone voitaisiin toteuttaa myös moninauhaisella Turingin koneella, mutta on toteutettavissa myös yksinauhaisella koneella, kuten muutkin moninauhaiset Turingin koneet. Käytännössä kone voidaan toteuttaa esimerkiksi siten, että nauhalle kopioidaan toiseenkin kohtaan syöte, ja koneiden käskyjonoja käydään vuorotellen läpi omiin syötejonoihinsa ja muistiin laitetaan aina kohta johon jäätiin. Kyseisestä kohdasta jatketaan, kun tullaan takaisin toisen koneen syötteestä. Tällöin kone M_A hyväksyessä syöte hyväksytään ja koneen $M_{\overline{A}}$ hyväksyessä, syöte hylätään. □

5.4 Algoritmi ja Churchin-Turingin teesi

Kuten alussa historia kappaleessa 2 todettiin 1900 matemaatikko David Hilbert esitti kuuluisan puheensa Pariisin kongressissa. Hän listasi 23 matemaattista ongelmaa haasteena tulevalle vuosisadalle. Kymmenes näistä ongelmista käsitteli algoritmeja. Ongelma oli keksiä algoritmi, jolla voidaan selvittää,

onko polynomilla kokonaislukujuuria. Hän ei käyttänyt sanaa algoritmi, mutta nykyisin hänen esittämäänsä asiaa kutsutaan algoritmiksi. Algoritmi tarkoittaakin yksiselitteisesti kuvattua jonoa käskyjä, jotka voidaan toteuttaa mekaanisesti.

Tähän Hilbertin kysymykseen algoritmeista vastaukseksi syntyi muun muassa Churchin λ- kalkyyli ja Turingin kone. Vasta vuonna 1970 Yuri Matijasevitsi todisti täsmällisesti, että ei ole algoritmia, jolla voitaisiin selvittää, onko polynomilla kokonaislukujuuria. Churchin λ- kalkyyli ja Turingin kone on todistettu yhtä pitäviksi. Myös muita yhtäpitäviä algoritmin määritteleviä malleja on; esimerkiksi Postin sääntöjärjestelmät (Post 1936) ja μ-rekursiiviset funktiot (Gödel, Kleene 1936). Kaikki nämä mallit ovat ekvivalentteja, joten ne antavat täysin saman listan ongelmista, jotka voidaan ratkaista algoritmisesti. Tätä yhteyttä epämuodollisen algoritmin käsitteen ja tarkan laskettavuuden määritelmän välillä kutsutaan Churchin-Turingin teesiksi. Seuraavassa teesissä Turingin koneen tilalla siis voisi olla mikä tahansa edellisistä ekvivalenteista malleista.

Lause 5.11. *(Churchin-Turingin teesi)(Vertaa [9, s. 143]) Intuitiivinen algoritmin käsite on yhden pitävä Turingin koneen algoritmin kanssa, eli mikä tahansa mekaanisesti ratkeava ongelma voidaan ratkaista Turingin koneella.*

Alussa totesimme predikaatin ratkeavuuden yhteydessä, että funktion ratkeavuus on kiinni ratkaisevasta ohjelmasta eli tarkan algoritmin määritelmästä, nyt Turingin kone ollaan Churchin-Turingin teesin avulla todettu tällaiseksi algoritmin määritelmäksi ja voidaan määritellä seuraavassa tarkasti. (Äärellinen automaatti ei ole tarkan algoritmin määritelmä, eikä näin liity Churchin-Turingin teesiin millään tavalla.) Turingin kone R vastaa ohjelman käsitettä P. Merkintä $R(x)$ tarkoittaa, että kone ajetaan syötteellä x. Ohjelman merkinnät voi palauttaa mieleen kappaleesta 3.

Määritelmä 5.9. Oletetaan että f on funktio $f : \Sigma^* \to \Sigma^*$ ja olkoon M aakkoston Σ Turingin kone. Silloin sanotaan, että M laskee funktion f, jos kaikilla merkkijonoilla $x \in \Sigma^*$ pätee $M(x) = f(x)$, missä merkintä $M(x)$ tarkoittaa tulostett, kun kone M ajetaan syötteellä x. Jos tällainen Turingin kone on olemassa, funktiota f sanotaan ratkeavaksi funktioksi.

Siis nauhalla pitää olla laskennan lopussa tulos $f(x)$. Yksinkertaistettuna, funktio joka voidaan ratkaista jollain algoritmilla on *laskettava*. Edellisessä algoritmin määritelmän pohjana käytämme Church-Turingin teesin nojalla Turingin konetta.

Churchin-Turingin teesin sisältö siis on, että ratkeavuus tarkoittaa samaa, kuin että ongelma voidaan ratkaista jollain algoritmisella menettelyllä, kuten Turingin koneella. Vastaisuudessa käytämme Turingin konetta tarkan algoritmin määritelmänä. Turingin konetta voidaan kuitenkin muuttaa paljonkin laskentavoiman säilyessä. Aiemmin on mainittu esimerkkinä, että Turingin

kone, jossa ei ole mahdollista pysyä paikallaan, ja Turingin kone, jossa lukija voi pysyä paikallaan nauhalla, voivat suorittaa vastaavat laskennat. Laskentavoima säilyy myös, vaikka nauha olisi ääretön vain toiseen suuntaan. Samoin, kun nauhalla on useita uria, eli rinnakkaisia merkkejä, jotka lukija lukee samanaikaisesti. Laskentavoima säilyy myös, vaikka koneessa olisi useita nauhoja joilla jokaisella oma lukija (tällä koneella on vain yksi tilojen joukko). Turingin koneen kyky laskea ja ratkaista ongelmia siis säilyy näissä eri variaatioissa, mikä tarkoittaa sitä, että nämä kaikki mallit on ilmaistavissa aiemmin esittelemämme tavallisen Turingin koneen muodossa.

5.5 Ratkeavia kieliä

Seuraavassa esitellään joitain esimerkkejä kielistä, jotka ovat ratkeavia. Aloitetaan ongelmilla, jotka koskevat äärellisiä automaatteja, ja merkitään äärellistä automaattia lyhenteellä DFA. Siis luodaan algoritmeja, joilla voidaan testata, hyväksyykö äärellinen automaatti merkkijonon, onko äärellisen automaatin kieli tyhjä ja tunnistavatko kaksi äärellistä automaattia saman kielen eli vastaavatko ne toisiaan. Lauseissa käytetään päätösongelmia vastaavia kieli edustamaan kyseistä ongelmaa, koska jos kieli on ratkeava, on myös kyseinen ongelmakin ratkeava.

Hyväksymisongelma äärelliselle automaatille, ”hyväksyykö äärellinen automaatti B syötteen w” (eli päteekö $w \in L(B)$), vastaa ongelmaa, kuuluuko syötteeksi koodattu automaatin ja merkkijonon kombinaatio $\langle R, w\rangle$ kieleen A_{DFA}. Seuraavassa lauseessa osoitamme, että A_{DFA} on ratkeava, josta seuraa että hyväksymisongelmakin on ratkeava.

Lause 5.12. *Kieli $A_{DFA} = \{\langle B, w\rangle \mid B$ on DFA, joka hyväksyy merkkijono syötteen $w\}$ on ratkeava.*

Todistuksen idea on hyvin yksinkertainen. Tarvitsee ainoastaan luoda Turingin kone M, joka ratkaisee kielen A_{DFA}. Sitä ennen Turingin kone tarkistaa, onko syötteen alkuosa äärellinen automaatti, eli onko se yhtenäinen graafi, kuten esimerkissä 5.7.

> **Kone M**
>
> Syötteellä $\langle B, w\rangle$, missä B on äärellinen automaatti merkkijonoksi koodattuna ja w on merkkijono, toimitaan seuraavasti:
>
> 1. Tarkistetaan, onko syöte äärellinen automaatti ja merkkijono. Jos ei ole, hylätään.
> 2. Simuloidaan automaattia B syötteellä w.
> 3. Jos simulointi päättyy hyväksyttyyn tilaan, *hyväksytään*. Jos simulointi päättyy hylkäävään tilaan, *hylätään*.”

Todistus. (Ks. [9, s. 152-153]) Edellä esiteltiin todistuksen ideaa. Ensiksi tarkastellaan syötettä $\langle B, w \rangle$. Se on äärellisen automaatin B ja merkkijonon w yhdistelmä. Eräs automaatin B esitysmuoto on esimerkiksi sen viiden osan lista ($Q, \sum, \delta, q_0$ ja F). Kun kone M vastaanottaa syötteen, M ensiksi tarkistaa, onko B todella äärellinen automaatti (siis sen koodaus) ja w merkkijono. Jos ei ole, M hylkää syötteen.

Sitten M suorittaa simulaation. Se merkitsee muistiin automaatin B nykyisen tilan ja sen nykyisen paikan syötteessä w, kirjoittamalla nämä nauhalle. Alussa automaatin B tila on q_0 ja sen paikka on merkkijonon w vasemman puoleisimman merkin kohdalla. Tila ja paikka on päivitetty tietyn siirtymäfunktion δ mukaan. Kun M lopettaa prosessin merkkijonon w viimeisessä merkissä, M hyväksyy syötteen, jos automaatti B on hyväksymistilassa. Vastaavasti M hylkää syötteen, jos B on hylkäämistilassa. □

Edellisessä lauseessa mietittiin, hyväksyykö äärellinen automaatti tiettyä merkkijonoa. Seuraavassa todistuksessa testataan, hyväksyykö äärellinen automaatti mitään merkkijonoa.

Lause 5.13. *$E_{DFA} = \{\langle A \rangle \mid A$ on DFA ja $L(A) = \emptyset\}$ on ratkeava kieli.*

Todistus. (Ks. [9, s. 154-155])

Äärellinen automaatti hyväksyy jonkin merkkijonon, joss on olemassa hyväksyvä laskentahistoria eli polku tilakaavion aloitustilasta hyväksymistilaan. Muodostetaan Turingin kone T, joka tarkistaa tämän automaatin koodatusta tilakaaviosta. Jos ei ole hyväksyvää laskentahistoriaa, niin hyväksytään. Sitä ennen kone tarkistaa syötteen oikeellisuuden eli sen, onko syöte tosiaan äärellinen automaatti eli yhdistetty graafi.

> **Kone T**
>
> Syötteellä $\langle A \rangle$, missä A on äärellinen automaatti merkkijonoksi koodattuna, toimitaan seuraavasti:
>
> 1. Tarkistetaan, onko syöte äärellisen automaatin koodi. Jos ei ole, hylätään.
> 2. Merkitään äärellisen automaatin A aloitustila.
> 3. Toistetaan seuraavaa, kunnes ei ole enää yhtään uutta tilaa:
>   - Merkitään kaikki tilat, joihin on siirtymä merkatusta tilasta.
> 4. Jos hyväksymistila ei ole merkitty, hyväksytään, muussa tapauksessa hylätään”

□

Seuraavassa lauseessa tutkitaan, voivatko kaksi äärellistä automaattia ratkaista saman kielen.

Lause 5.14. *$EQ_{DFA} = \{\langle A, B\rangle \mid A$ ja B ovat äärellisiä automaatteja ja $L(A) = L(B)\}$ on ratkeava kieli.*

Todistus. (Ks. [9, s. 155]) Tässä todistuksessa käytämme edellistä lausetta 5.13. Ja muodostamme uuden äärellisen automaatin C automaateista A ja B, missä C hyväksyy vain ne merkkijonot, jotka A tai B hyväksyy, mutta C ei hyväksy molempien hyväksymiä. Siten jos A ja B hyväksyy saman kielen, automaatti C ei hyväksy mitään. Automaatin C kieli on

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Tätä kaavaa kutsutaan kielten $L(A)$ ja $L(B)$ symmetriseksi erotukseksi. Tässä tapauksessa $L(C) = \emptyset$, joss $L(A) = L(B)$. Voidaan muodostaa automaatti C automaateista A ja B, koska tiedetään, että säännöllisten kielten joukko on suljettu leikkauksen, unionin ja negaation suhteen. Siis automaatti C muodostetaan suoraviivaisesti lauseiden 4.2 - 4.4 muodostetuista äärellisistä automaateista. Koska on muotoiltu automaatti C, voidaan käyttää edellistä tyhjyyslausetta 5.13 testaukseen, onko $L(C) = \emptyset$. Jos se on, $L(A) = L(B)$. Edellisten lauseiden tavoin syötteen oikeellisuus tarkistetaan alussa.

Kone F

Syötteellä $\langle A, B\rangle$, missä A ja B ovat äärellisiä automaatteja merkkijonoksi koodattuna, toimitaan seuraavasti:

1. Tarkistetaan, onko syötteen A ja B oikeita äärellisiä automaatteja, jos vähintään toinen ei ole hylkää.
2. Muodostetaan äärellinen automaatti C lauseiden 4.2 - 4.4 äärellisiä automaateja yhdistellen.
3. Ajetaan Turingin kone T lauseen 5.13 todistuksesta syötteellä $\langle C\rangle$.
4. Jos T hyväksyy, hyväksytään. Jos T hylkää, hylätään."

□

6 Universaali Turingin kone

Tässä kappaleessa käsitellään Universaalia Turingin konetta, mutta sitä ennen esitellään Turingin koneiden koodaus, diagonaalikieli ja universaalikieli. Näiden asioiden ymmärtäminen auttaa universaalikoneiden ymmärtämisessä ja myöhemmin ratkeamattomuutta tutkittaessa. Tämän kappaleen pohjana on käytetty lähdettä [5]. Nyt käsitellään kuitenkin Turingin koneiden koodausta.

6.1 Turingin koneiden koodaus

(Ks. [5, s. 369-370]) Edellisessä kappaleessa äärellisen automaatin ratkeavuusongelmissa koodattiin automaatit tietyllä tavalla Turingin koneen käsittelyä varten. Seuraavassa koodataan Turingin kone ratkeavuusongelmiaan varten. Jokainen Turingin kone, joka on muotoa $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, missä $\Sigma = \{0, 1\}$, voidaan esittää binäärijonoja esimerkiksi seuraavasti.

1. $Q = \{q_0, q_1, \ldots q_n\}$, missä $q_{n-1} = q_{acc}$ ja $q_n = q_{rej}$
2. $\Gamma = \{a_0, a_1, \ldots a_m\}$, missä $a_0 = 0$, $a_1 = 1$ ja $a_2 = \sqcup$.
3. Merkitään myös $L = X_0$, $R = X_1$ ja $S = X_2$
4. Siirtymäfunktio on muotoa $\delta(q_i, a_j) = (q_r, a_s, X_t)$, jonka koodi on

$$c_{ij} = 0^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1}$$

.

Tällöin koko Turingin koneen koodi voisi olla esimerkiksi

$$c_M = 111c_{00}11c_{01}11 \ldots 11c_{0m}11c_{10}11 \ldots 11c_{(n-2)0}11 \ldots 11c_{(n-2)m}111$$

Siis koodin alussa on tilasta nolla olevat siirrot, sitten tilasta 1 ja niin edelleen, tilat erotettuina ykkösillä. Kone toimii siten, että kun kone huomaa luvun 1 se ”tietää”, että nyt alkaa jokin uusi asia. Peräkkäisten ykkösten määrästä voidaan päätellä alkaako jonkin tilan, merkin, siirtymisen vaiko koko Turingin koneen koodi. Näin ollen jokaisella binääriaakkoston jonkin kielen tunnistavalla Turingin koneella M on siis yksikäsitteinen binäärikoodi c_M jota merkitään joskus $\langle M \rangle$. Jokaiseen c muotoiseen binäärijonoon voidaan samoin liittää yksikäsitteisesti jokin Turingin kone.

Kuitenkaan kaikki binäärijonot eivät ole Turingin koneen koodauksia, esimerkiksi lukujono 10 ei ole. Tällaiset ”virheelliset” binäärijonot liitetään yksinkertaiseen kaikki syötteet hylkäävään Turingin koneeseen M_{triv}. Samoin kuin aiemmin graafin ja automaattien kohdalla koneen piti ensin tarkistaa, vastaako syöte graafia tai automaattia.

$$M_c = \begin{cases} M, & \text{jos } c \text{ on koneen } M \text{ koodi eli } c = c_M, \\ M_{triv}, & \text{muuten.} \end{cases}$$

Tällä tavalla koodattuna kaikki aakkoston $0, 1$ Turingin koneet voidaan luetella leksikograafisessa järjestyksessä, M_ϵ, M_0, M_1, M_{00}, M_{01}, M_{10}, ... ja samoin koneiden tunnistamat kielet voidaan luetella $L(M_\epsilon)$, $L(M_0)$, $L(M_1)$, $L(M_{00})$, $L(M_{01})$, ... Kielet voivat olla luettelossa useammankin kerran.

6.2 Diagonaalikieli

Nyt voidaan todistaa diagonaalimenetelmää ja koodausta apuna käyttäen seuraava lause, joka pitää sisällään teoreettisen ongelman "Hylkääkö annetun koodin c_M esittämä Turingin kone syötteen c_M ?".

Lause 6.1. *Diagonaalikieli $D = \{c_M \in \{0,1\}^* \mid c_M \notin L(M_c)\}$ ei ole tunnistettava}.*

Todistus. (Ks. [5, s. 372]) Oletetaan, että kieli D on tunnistettava jollakin Turingin koneella M, siis $D = L(M)$. Olkoon c_M koneen M binäärikoodi. Siis $D = L(M_{c_M})$ (eli D on kieli, jonka tunnistaa kone M, jonka koodi on c_M).

Mutta tällöin on $c_M \in D \Leftrightarrow c_M \notin L(M_{c_M}) = D$, mikä on ristiriita. Siis kielen D tunnistavaa Turingin konetta ei ole, ja oletus, että kieli D olisi tunnistettava, on väärä. □

Visuaalisesti todistuksen voi ymmärtää seuraavan äärettömän taulukon avulla. Tässä taulukossa on lueteltuna kaikkien aakkoston $\{0,1\}$ Turingin koneiden $M_\epsilon, M_0, \ldots$ kielet koneiden koodin mukaan leksikografisessa järjestyksessä $L(M_\epsilon), L(M_0), \ldots$. Siis kielet löytyvät sarakkeista, ja riveiltä löytyvät koneiden koodit leksikografisessa järjestyksessä $c_{M_\epsilon}, c_{M_0,\ldots}$, jota merkitään lyhyemmin $c_\epsilon, c_0, \ldots$.

Taulukko koostuu soluista, joissa

$$T(i,j) = \begin{cases} 1, & \text{jos } c_j \in L(M_i) \\ 0, & \text{jos } c_j \notin L(M_i) \end{cases}$$

D	$L(M_\epsilon)$	$L(M_0)$	$L(M_1)$	$L(M_{00})$	$\cdots$
c_ϵ	$\underline{1}$	0	0	0	$\cdots$
c_0	0	$\underline{0}$	1	0	$\cdots$
c_1	0	1	$\underline{1}$	0	$\cdots$
c_{00}	1	0	1	$\underline{1}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Nyt kieli D on $D = \{c_d \in \{0,1\}^* \mid c_d \notin L(M_d)\}$.

Kieleen kuuluvat siis ne taulukon koodit c, jotka diagonaalilla ovat nollia, eli eivät kuulu koodiaan vastaavan Turingin koneen kieleen. Ne koodit, jotka kuuluvat omaa koodiaan vastaavan koneen kieleen, eli ovat diagonaalilla ykkösiä, eivät kuulu kieleen D.

Jokainen sarake taulukossa kertoo, mitkä koodit kuuluvat kyseiseen kieleen. Siis voidaan luoda jono kieleen D kuuluvista koodeista diagonaalia pitkin kulkemalla. Siis koodien ja kielten vastaavuudet voidaan esittää jonona $r = d(\epsilon), d(0), \ldots$, joka kertoo mitkä koodit kuuluvat joukkoon D. Edellä olevan taulukon tapauksessa $r = 0, 1, 0, 0, \ldots$.

$$d(i) = \begin{cases} 0, & \text{jos } c_i \in L(M_i) \\ 1, & \text{jos } c_i \notin L(M_i) \end{cases}$$

Siis jono r ei ole taulukon sarake, koska se on diagonaalilla eri kuin jokaisen sarakkeen jono. Koska kieli D ei ole mikään kielistä $L(M_\epsilon)$, $L(M_0)$, $L(M_1)$ ... , se ei ole mikään tunnistettava kieli $L(M_i)$.

Siis oletus kielen D tunnistavasta koneesta ja siis kielen tunnistettavuudesta on väärä ja kieli D ei ole tunnistettava.

6.3 Universaali Turingin kone

Edelliset asiat koodaus ja diagonaalikieli johtavat kohti seuraavaa aihetta, jota kutsumme universaaliksi Turingin koneeksi. Universaalit Turingin koneet tarkoittavat Turingin koneita, jotka pystyvät toistamaan minkä tahansa Turingin koneen toiminnan saatuaan kyseisen Turingin koneen koodin. Seuraavassa täsmällisemmin universaalikielestä ja universaaleista Turingin koneista.

Määritelmä 6.1. Aakkoston $\Sigma = \{0, 1\}$ *universaalikieli* U on $U = \{c_M w \mid w \in L(M)\}$. Kieli U sisältää tiedon kaikista aakkoston $\Sigma = \{0, 1\}$ tunnistettavista kielistä.

Siis jos kieli $A \subseteq \{0, 1\}^*$ on tunnistettava ja $A = L(M)$, missä M on kielen tunnistava Turingin kone, niin silloin $A = \{w \in \{0, 1\}^* \mid c_M w \in U\}$. Edellä $U = \{c_M w \mid w \in L(M)\}$ on Turingin koneen ohjelma c_M ja koneen M hyväksymät merkkijonot peräkkäin.

On huomattava, että c_M ja w on koodauksen takia helposti erotettavissa toisistaan. Voitaisiin erotella koodit myös niin, että alussa olisi merkki, joka kertoisi koodin c_M pituuden tai jollain muulla tavalla.

Määritelmä 6.2. Kielen U tunnistavia Turingin koneita sanotaan *universaaleiksi Turingin koneiksi.*

Lause 6.2. *Kieli U on tunnistettava.*

Todistus. (Ks. [5, s. 378]) Muodostetaan nelinauhainen kone T, joka tunnistaa kielen U eli $U = L(T)$. Muistetaan että Turingin koneen M koodi on muotoa

$$c_M = 111c_{00}11c_{01}11\ldots11c_{0m}11c_{10}11\ldots11c_{(n-2)0}11\ldots11c_{(n-2)m}111$$

Missä $c_{ij} = 0^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1}$ kuvaa siirtymäfunktiota $\delta(q_i, a_j) = (q_r, a_s, X_t)$, $q_{n-1} = q_{acc}$, $q_n = q_{rej}$, ja nauha-aakkoston merkkien lukumäärä on m.

Kun syöte on muotoa $z = c_M w$, missä c_M on koneen M koodi ja w on koneen M syöte (merkit koodattu kuten koodissakin a_j vastaa jonoa 0^{j+1}), niin koneen T nauhoja käytetään seuraavalla tavalla. Nauhalla 1 on syöte z kokonaisuudessaan ja siis erityisesti koneen M siirtymäfunktioiden koodi c_{ij}. Nauhalla 2 on koneen M nauhan sisältö, jossa nauha-aakkoston symbolit on koodattuina nollajonoiksi aiempaa koodausta käyttäen 0^{j+1}. Nämä on erotuttu toisistaan ykkösillä. Nauhalla 3 on koneen M tila koodattuna vastaavasti nollajonona ja nauha 4 on koneen T työnauha eli siellä suoritetaan laskennat.

Koneen T laskenta alkaa siitä, että aluksi tarkistetaan, onko syötteen koodi c_M toimivan Turingin koneen koodi. Jos ei ole, niin T hylkää syötteen. Muuten kone T alkaa simuloimaan konetta $M = M_{c_M}$ syötteellä w.

Alussa nauhalla 1 on siis koko syöte $c_M w$, nauhalla 2 on merkkijono $w(\in \{0,1\}^*)$, nauhalla 3 on koneen M ensimmäinen tila q_0 ja nauhalla 4 ei vielä mitään. Näiden ulkopuolisilla alueilla nauhoilla on tyhjä merkkejä $\sqcup$.

Jokaisessa askeleessa, kun nauhalla 3 on jono 0^{i+1} ja nauhalla 2 ollaan jonon 0^{j+1} alussa

1. Kone T etsii koneen M kuvauksesta nauhalta 1 kohdan $110^{i+1}10^{j+1}1$ (siirtymäfunktion $\delta(q_i, a_j)$), jos tällaista ei löydy, hylätään.

2. Jos nauhalta 1 löytyy jono $110^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1}$, kone vaihtaa nauhalle 3 tilaksi merkkijonon 0^{r+1} ja nauhalle 2 lukijasta lähtien merkin uutta merkkiä vastaavan 0^{s+1} (nauhalla 2 olevan merkkijonon loppupäätä siirretään tarpeen mukaan, jos uusi "merkki" eli nollajono vie enemmän tilaa kuin aiempi) ja nauhan 2 lukija palaa tämän kirjoitetun nollarivin alkuun.

3. Tämän jälkeen siirrytään nauhalla 2 seuraavaan ykköseen joko vasemmalla (jos $t = 0$), oikealle (jos $t = 1$) tai pysytään paikallaan ($t = 2$).

4. Kierroksen lopussa tarkistetaan, onko nauhalla 3 koneen M hyväksymistila q_{n-1}. (Tässä tapauksessa hylkäämistila aiheuttaa loputtoman siirtymisen oikealle, joten ei haittaa). Jos kone M on hyväksymistilassa, kone T hyväksyy, mutta jos ei ole, aloitetaan kierros alusta.

□

Lause 6.3. *Kieli U ei ole ratkeava.*

Todistus. (Ks. [5, s. 380-381])

Todistus alkaa vastaoletuksesta, että kieli U on ratkeava. Silloin myös $\overline{U}$ on ratkeava lauseen 5.9 mukaan. Siis jos saadaan muodostettua kielen $\overline{U}$ tunnistava Turingin kone M, niin voitaisiin muodostaa diagonaalikielen tunnistava Turingin kone. Ja koska tiedetään, että diagonaalikieli ei ole tunnistettava, niin ristiriidasta seuraa, ettei universaalikieli voi olla ratkeava. Seuraavassa todistus täsmällisesti.

Oletetaan nyt, että $L(M) = \overline{U}$. Voidaan muodostaa Turingin koneesta M Turingin koneen M', joka hyväksyy diagonaalikielen D. Alussa kone M' tarkistaa, onko syöte jonkin Turingin koneen koodi. Jos ei ole, hyväksytään, ja jos on mennään seuraaviin askeliin.

> **Kone M'**
>
> Syötteellä c_M, missä c_M on Turingin kone M merkkijonoksi koodattuna, toimitaan seuraavasti:
>
> 1. Saadessaan syötteen c_M kone M' muuttaa syötteen muotoon $c_M 111 c_M$.
> 2. Simuloidaan koneen M toimintaa uudella syötteellä. Jos kone M hyväksyy, hylätään, ja jos M hylkää, niin hyväksytään. Jos c_M on c_{M_i} koodauksessa, niin määritetään, hyväksyykö M_i syötteen c_{M_i}. Siis M hyväksyy kielen $\overline{U}$, joss M_i ei hyväksy syötettä w_i; eli $w_i \in D$.

Siis kone M' hyväksyy syötteen c_M, joss $w \in D$. Siten M' ei voi olla olemassa, koska diagonaalikieli D ei ole tunnistettava. Päättely voidaan kirjoittaa ketjuna

> $w \in L(M') \Leftrightarrow w$ ei validi koodi tai $c_M 111 c_M \notin U \Leftrightarrow c_M \notin L(M_{c_M}) \Leftrightarrow w \in D$.

□

Lause 6.4. *Kieli $\overline{U} = \{c_M w \mid w \notin L(M)\}$ ei ole tunnistettava kieli.*

Todistus. Tämä lause seuraa yksinkertaisesti edellisistä lauseista ja lauseesta 5.10. Jos $\overline{U}$ olisi tunnistettava, niin myös kieli U olisi ratkeava, mikä on mahdotonta. □

7 Ratkeamattomia ongelmia

Kuten aiemmin huomattiin universaalikielen yhteydessä, että kaikki kielet eivät ole ratkeavia. Nyt aletaan käsitellä muitakin ratkeamattomia ongelmia. Monissa ratkeamattomuustodistuksissa käytetään apuna palautusta. Se tarkoittaa sitä, että jos tiedetään jonkin kielen, esimerkiksi universaalikielen tai diagonaalikielen, olevan ratkeamaton, niin toisen kielen tai ongelman ratkeamattomuus voidaan todistaa yhdistämällä todistus tähän aiempaan tulokseen. Edellä universaalikielen ratkeamattomuustodistuksessa käytettiin palautusta diagonaalikielen tunnistamattomuuteen. Yleisesti sanomme, että ongelma A voidaan palauttaa ongelmaan B, jos ongelmalle B voidaan muodostaa ratkaisualgoritmi ongelman A mitä tahansa algoritmia käyttäen. Tässä ongelman A algoritmi on ikään kuin aliohjelmana ongelman B ratkaisussa.

Huom! Seuraavissa lauseissa käytetään merkintää $\langle M, w\rangle$ tarkoitettaessa merkkijonosyötettä $c_M w$.

7.1 Hyväksymisongelma

Seuraavassa lauseessa käsitellään ongelmaa, ”hyväksyykö Turingin kone annetun merkkijonosyötteen”. Sitä voidaan merkitä kielellä $A_{TM} = \{\langle M, w\rangle \mid M \text{ on Turingin kone ja } M \text{ hyväksyy syötteen } w\}$. Kuitenkin vaikka vastaava kieli äärellisille automaateille A_{DFA} oli ratkeava, niin A_{TM} ei ole. Todistetaan kuitenkin ensin, että se on tunnistettava.

Lause 7.1. *$A_{TM} = \{\langle M, w\rangle \mid$ Turingin kone M hyväksyy merkkijonon $w\}$ on tunnistettava.*

Todistus. (Ks.[6, s. 201]) Kieli A_{TM} voidaan tunnistaa universaalikoneella U.

Kone U

Syötteellä $\langle M, w\rangle$, missä on Turingin kone M merkkijonoksi koodattuna ja merkkijono w, toimitaan seuraavasti:

1. Simuloidaan konetta M syötteellä w.
2. Jos M hyväksyy, hyväksytään. Jos M hylkää, hylätään.

□

Lause 7.2. *$A_{TM} = \{\langle M, w\rangle \mid$ Turingin kone M hyväksyy merkkijonon $w\}$ ei ole ratkeava.*

Todistus. (Ks. [9, s. 159-160, 165-166], [6, s. 208-212]) Todistus tehdään vastaoletuksen kautta. Vastaoletus tässä tapauksessa on, että jokin ratkaisija R ratkaisee kielen A_{TM}. Siis syötteellä $\langle M, w\rangle$ kone R hyväksyy, jos $w \in L(M)$ ja muuten hylkää.

Nyt diagonaalikieli voidaan tunnistaa seuraavalla koneella A.

Kone A

Syötteellä w, missä w merkkijono, toimitaan seuraavasti:

1. Jos w ei ole c_i millään i, niin hylätään.
2. Muodostetaan $\langle M_i, w\rangle$, missä i on indeksi, jolla $w = c_i$.
3. Simuloidaan konetta R syötteellä $\langle M, w\rangle$, jos R hyväksyy, hylkää. Ja jos R hylkää, hyväksy.

Koska D ei ole tunnistettava, syntyy ristiriita ja siis vastaoletus on väärä. Tulos on siis, että kieli $A_{TM} = \{\langle M, w\rangle \,|\text{Turingin kone } M \text{ hyväksyy merkkijonon } w\}$ ei ole ratkeava. □

Lause 7.3. *$\overline{A_{TM}}$ ei ole tunnistettava.*

Todistus. Seuraa suoraan edellisistä lauseista ja lauseesta 5.10. Koska jos $\overline{A_{TM}}$ olisi tunnistettava ja koska A_{TM} on tunnistettava, niin kieli A_{TM} olisi ratkeava lauseen 5.10 perusteella. □

7.2 Pysähtymisongelma

Pysähtymisongelma on eräs hyväksymisongelman muunnelma, koska jos kone hyväksyy, se myös pysähtyy. Pysähtymisongelmassa on siis lisänä ne tapaukset, joissa kone pysähtyy hylkäämistilaan.

Lause 7.4. *Pysähtymisongelma $HALT = \{\langle M, w\rangle \mid M$ pysähtyy syötteellä $w\}$ on tunnistettava.*

Todistus. Universaalikoneesta U on kohtuullisen vaivatonta muokata kone, joka simuloi koneen M laskentaa syötteellä w ja hyväksyy, joss simuloitu laskenta pysähtyy. Siis muokattu kone hyväksyy aina, kun kone M pysähtyy. □

Lause 7.5. *Pysähtymisongelma eli kieli $HALT = \{\langle M, w\rangle \mid M$ pysähtyy syötteellä $w\}$ ei ole ratkeava.*

Todistus. (ks.[9, s. 160,165], [7, s. 59-60])

Koska $\overline{HALT} = \{\langle M, w\rangle \mid M$ ei pysähdy syötteellä $w\}$ ei voi koskaan olla tunnistettava, koska koneen M laskenta ei pysähdy, niin siksi kieli $HALT$ ei voi olla ratkeava lauseen 5.10 perusteella.

Toisaalta todistus voidaan tehdä palauttamalla ongelma kielen A_{TM} ratkeamattomuuteen. Siis tehdään vastaoletus, että $HALT_{TM}$ on ratkeava. Kieli A_{TM} voitaisiin ratkaista seuraavalla Turingin koneella E.

Kone *E*

Syötteellä $\langle M, w \rangle$, missä on Turingin kone M merkkijonoksi koodattuna ja merkkijono w, toimitaan seuraavasti:

1. Jos $\langle M, w \rangle \notin HALT_{TM}$, niin hylätään.
2. Simuloidaan konetta M syötteellä w. Jos M hyväksyy, hyväksytään. Jos M hylkää syötteen w, hylätään.

Vastaoletuksen perusteella ensimmäinen kohta voidaan ratkaista. Kohdassa 2 tiedetään, että laskenta pysähtyy aina. Mutta kieli A_{TM} ei ole ratkeava. Siis syntyy ristiriita ja vastaoletus on väärä ja kieli $HALT_{TM}$ ei ole ratkeava. □

8 Yhteenveto

Tässä työssä käsiteltiin joitain laskettavuuden perusasioita. Alussa kerrottiin laskettavuuden historiasta ja siihen vaikuttaneista matemaatikoista, joista tärkeimpiä lienee Leibniz, Cantor, Gödel ja Turing. Erityisesti 1930–luvulla keskustelu laskennan mekanisoinnista kävi kuumana matemaatikoiden kesken, ja toisen maailmansodan aikaan alettiin tosissaan luoda fyysisiä tietokoneita.

Laskettavuuden teoriaan kuuluu paljon käsitteistöä, joita matematiikan opinnoissa ei aina käsitellä. Tutkielmassa selitettiin muun muassa, mitä ovat aakkostot, kielet, kieliperheet ja ratkeavuus. Laskennan mekanisoinnin malleista tutustuttiin äärellisiin automaatteihin ja Turingin koneisiin. Äärelliset automaatit eivät vastaa tarkan algoritmin käsitettä, mutta ovat hyviä malleja laitteille, joilla ei ole paljoa muistia. Turingin koneet taas ovat tarkkoja algoritmeja, siis jokainen ongelma, joka voidaan ylipäätään ratkaista, voidaan ratkaista Turingin koneella.

Tutkielman lopussa esitettiin joitain ratkeavuus ja ratkeamattomuus todistuksia Turingin konetta käyttäen. Tärkeimpiä niistä lienee pysähtymisongelman ratkeamattomuuden todistus. Siinä todistettiin pysähtymisongelman ratkeamattomuus, eli ongelman ”pysähtyykö syötteenä annettu Turingin kone annetulla merkkijono syötteellä” ratkeamattomuus. Tällä on merkitystä muun muassa tietokoneohjelmia suunniteltaessa. Käytännössä tämä tulee esiin, kun tietokoneessa on jokin laskeminen käynnissä ja eteneminen tuntuu kestävän, niin ennen pysähtymistä ei voida sanoa pysähtyykö kone vai jääkö se ikuiseen luuppiin. Yleensä näissä tapauksissa ohjelmissa on aikakatkaisu, jolloin laskeminen keskeytyy, jos pysähtymistä ei ole tapahtunut johonkin aikaan mennessä. Yleensä algoritmit pyritään tekemään niin tehokkaiksi, että pysähtyminen tapahtuisi mahdollisimman nopeasti.

Laskettavuuden teoriassa voidaan luoda automaatteja, Turingin koneita tai muita algoritmeja, aivan kuin käytössä olisi mielivaltaisen paljon resursseja, esimerkiksi tilaa, aikaa ja muistia. Näin toimimme myös tässä tutkielmassa. Kuitenkin fyysiset laitteet ovat rajallisia ja tietokoneissa on rajallinen määrä muistia. Käytännön sovelluksia tehtäessä algoritmien tulisi olla tehokkaita ja käyttää mahdollisimman vähän resursseja. Laskennanvaativuusteoria on laskettavuudenteorian osa, jossa pohditaan millaisille ongelmille on olemassa tehokas algoritmi. Yleisimmin tämä tarkoittaa että laskenta-ajan pitäisi skaalautua kohtuullisesti, esimerkiksi polynomisesti, syötteen koon suhteen.

Tietokoneet ja niiden matemaattinen pohja, laskettavuuden teoria, koskettaa lähes kaikkia nykyajan ihmisiä. Kuitenkin harvemmat ihmiset sitä tietävät tai tulevat ajatelleeksi, vaikka tietokoneiden käyttöön liittyvistä ongelmista yleisesti puhutaankin. Tämä tutkielma oli pintaraapaisu laskettavuuden teoriaan. Tutkielmasta jäi pois paljon kiinnostavia asioita lasketta-

vuuden teorian kehityksestä ja historiasta ja hyvin kiehtova alue, jota kutsutaan laskennanvaativuusteoriaksi. Jos lukijaa kiinnostaa tietää vielä lisää aiheesta, niin lähdeteoksista on hyvä aloittaa ja erityisesti kirjoista [9] ja [5]. Lisäksi ensimmäisten tietokoneiden suunnittelusta ja kehittelystä löytyy paljon Andrew Hodgesin kirjasta Alan Turing arvoitus [4]. Tietokoneen esihistorian tunteminen herättää kunnioitusta tietokoneen kehittäneitä matemaatikkojen työtä ja oivalluksia kohtaan, joka usein unohtuu tietokoneiden ollessa nykyään täysin itsestäänselvyyksiä.

Viitteet

[1] N.J. Cutland, *Computability An introduction to recursive funktion theory.* Cambridge University Press, 1980

[2] M. Davis, *Tietokoneen esihistoria Leibnizista Turingiin* (The Universal Computer. The Road from Leibniz to Turing), Dark Oy, Vantaa, 2003

[3] T. Elomaa, J. Kujala, *Johdatus tietojenkäsittelyteoriaan.* Tampereen Teknillinen yliopisto, Tampere, 2006, (viitattu 21.5.2009), http://www.cs.tut.fi/ẽlomaa/opetus/iTCS06-1.pdf

[4] A. Hodges, *Alan Turing arvoitus* (Alan Turing enigma), Hakapaino, Helsinki, 2000

[5] J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to automata theory, languages, and computation*, 2nd ed., Pearson Education International/Addison Wesley, Upper Saddle River, 2003.

[6] J. Kivinen, *Laskennan mallit.* Helsingin yliopisto, 2006, (viitattu 21.5.2009) http://www.cs.helsinki.fi/u/jkivinen/opetus/lama/s06/s001-241.pdf

[7] C. Papadimitriou, *Computational Complexity*, 2nd ed., Addison-Wesley Publishing Company, USA, 1995

[8] M-L. Rantaniemi, *Lukujärjestelmät.* Jyväskylän yliopisto, Jyväskylä 2005, (viitattu 21.5.2009), http://www.math.jyu.fi/matcl/zvanhat/kurssit/matematiikan_osa-alueita/Esitelmat/Lukujarjestelmat.pdf

[9] M. Sipser, *Introduction to the Theory of Computation*, PWS Publishing company 1997