Laskennan teoria

Pekka Orponen
Helsingin yliopisto
Tietojenkäsittelytieteen laitos

Lukijalle

Tämä moniste on syntynyt Helsingin yliopiston tietojenkäsittelytieteen laitoksella vuosina 1990–1993 pitämieni luentojen pohjalta. Moniste antaa perustiedot siitä tietojenkäsittelyteorian alasta, jolle on englanninkielessä vakiintunut nimi "Theory of Computation", siis automaattien, kielioppien, laskettavuuden ja laskennan vaativuuden teoriasta.

Tämän aihepiirin käsittelyssä on valittavissa kaksi linjaa: joko pelkkä yleiskatsauksellinen teorian tulosten ja sovellusesimerkkien esittely, tai sitten käsitteiden huolellinen määrittely ja väitteiden todistaminen. Olen pitänyt oman esitykseni tavoitteena pääpiirteittäin jälkimmäistä, kuitenkin siten, että olen yrittänyt yleistajuisin selityksin ja esimerkein havainnollistaa sitä, mikä on kulloistenkin teoreettisten kehittelyjen käytännöllinen merkitys.

Joissakin paikoin olen täydellisyyden vuoksi, ja toivottavasti asiasta harrastuneiden ilahduttamiseksi, ottanut mukaan materiaalia, joka ei varsinaisesti ole kuulunut kurssin vaatimuksiin. Nämä tekstijaksot olen merkinnyt tähdellä (∗).

Helsingissä, 14. joulukuuta 1993

Pekka Orponen

Tähän monisteen toiseen painokseen olen korjannut joukon ladontavirheitä ja muita vähäisiä puutteita. Pääosin teksti on muuttumaton.

Helsingissä, 9. elokuuta 1994

Pekka Orponen

Sisältö

1 Laskennalliset ongelmat ja niiden esittäminen **1**
1.1 Laskennalliset ongelmat, merkkijonot ja kielet 1
1.2 Laskennallisten ongelmien ratkeavuus 3

2 Äärelliset automaatit ja säännölliset kielet **7**
2.1 Tilasiirtymäkaaviot ja -taulukot 7
2.2 Äärellisiin automaatteihin perustuva ohjelmointi 9
2.3 Äärellisen automaatin käsitteen formalisointi 12
2.4 Äärellisten automaattien minimointi 14
2.5 Epädeterministiset äärelliset automaatit 18
2.6 Säännölliset lausekkeet ja kielet 21
2.7 Äärelliset automaatit ja säännölliset kielet 24
2.8 Säännöllisten kielten rajoituksista 30

3 Kontekstittomat kieliopit ja kielet **33**
3.1 Kieliopit ja merkkijonojen tuottaminen 33
3.2 Säännölliset kielet ja kontekstittomat kieliopit 36
3.3 Kontekstittomien kielioppien jäsennysongelma 38
3.4 Rekursiivisesti etenevä jäsentäminen 41
3.5 Attribuuttikieliopit 47
3.6 Eräs yleinen jäsennysmenetelmä 51
3.7 Pinoautomaatit 56
3.8 $*$ Kontekstittomien kielten rajoituksista 59

4 Turingin koneet **61**
4.1 Kielten tunnistaminen Turingin koneilla 61
4.2 Turingin koneiden laajennuksia 66

5 Rajoittamattomat ja kontekstiset kieliopit **73**

6 Laskettavuusteoriaa **79**
6.1 Rekursiiviset ja rekursiivisesti lueteltavat kielet 79
6.2 Rekursiivisten ja rekursiivisesti lueteltavien kielten perusominaisuuksia 80
6.3 Turingin koneiden kooodaus ja eräs ei rekursiivisesti lueteltava kieli 82
6.4 Universaalikieli U ja universaalit Turingin koneet 83
6.5 Turingin koneiden pysähtymisongelma 86
6.6 Ratkeamattomuustuloksia Pascalilla 87

6.7 Rekursiiviset kielet ja Chomskyn kieliluokat 87
6.8 Lisää ratkeamattomia ongelmia 88
6.9 Ratkeamattomuustuloksia muilla aloilla 92
6.10 Rekursiiviset funktiot 93
6.11 ∗ Rekursiiviset palautukset ja RE-täydelliset kielet 94

7 Laskennan vaativuusteoriaa **99**
7.1 Työläät ongelmat 99
7.2 Turingin koneiden aika- ja tilavaativuus 100
7.3 Vaativuusfunktioiden kertaluokat 103
7.4 Vaativuusluokat 104
7.5 Vaativuusluokkien ominaisuuksia 105
7.6 Epädeterministiset vaativuusluokat 109
7.7 Esimerkkejä luokan NP ongelmista 113
7.8 Polynomiset palautukset ja NP-täydelliset kielet 115
7.9 Toteutuvuusongelman NP-täydellisyys 117
7.10 Muita NP-täydellisiä ongelmia 122

8 Kirjallisuutta **127**

Harjoitustehtäviä **129**

Luku 1

Laskennalliset ongelmat ja niiden esittäminen

1.1 Laskennalliset ongelmat, merkkijonot ja kielet

Tämän monisteen aiheena ovat eräät *laskennallisten ongelmien* täsmälliset esitystavat, niiden ominaisuudet ja hyväksikäyttö. Laskennallisella ongelmalla tarkoitetaan tässä yleisesti mitä tahansa tehtävää, joka voidaan mallintaa ratkaistavaksi digitaalisella tietokoneella: tällaisia ovat esimerkiksi kokonaislukujen kertolasku, kirjastokortiston aakkostaminen, yrityksen palkanlaskenta, tai yliopistollisen kurssin kurssitietojen ylläpito. Yksi laskennallisen ongelman mahdollinen esitystapa on sen ratkaiseva ohjelma — mutta monesti on tarvetta esityksille, joissa ongelma on kuvattu abstraktimmin, ottamatta kantaa ohjelmallisen toteutuksen yksityiskohtiin.

Jotta laskennallisten ongelmien ominaisuuksista ja esittämisestä voitaisiin puhua täsmällisesti, käsite on ensin formalisoitava, pyrkien kuitenkin mahdollisuuksien mukaan säilyttämään kaikki sen intuitiivisesti välttämättömät ominaisuudet. Käsitteelle seuraavassa annettava formalisointi perustuu kahteen havaintoon:

- Laskennallisissa ongelmissa voidaan yleensä erottaa potentiaalisesti ääretön joukko *tapauksia* ("syötteitä"); ongelman ratkaisu on tällöin *algoritmi*, joka liittää kuhunkin tapaukseen sen oikean *vastauksen* ("tulosteen"). Esimerkiksi kokonaislukujen kertolaskuongelman tapauksia ovat kaikki mahdolliset kokonaislukuparit, annettuun tapaukseen liittyvä vastaus on lukuparin tulo, ja ongelman ratkaisuksi käy mikä tahansa yleinen kertolaskualgoritmi.

- Koska tietokoneet ovat rajallisia laitteita, on luontevaa vaatia, että kukin yksittäinen tapaus ja sen vastaus ovat *äärellisesti esitettäviä*. Konkreettisesti voi ajatella, että tapaukset ja niiden vastaukset esitetään viime kädessä koneen keskus- tai tukimuistin tiloina, so. bittijonoina. Tapauksia voi olla potentiaalisesti ääretön määrä, aina isompien ja isompien koneiden käsiteltäviksi, mutta kukin yksittäinen ongelman tapaus pitää voida esittää jonkin äärellisen koneen ratkaistavaksi.

Tämän tarkastelun perusteella täsmennetään alustavasti, että laskennallinen ongelma on *kuvaus äärellisesti esitettävien tapausten joukosta äärellisesti esitettävien vastausten joukkoon*[1].

[1] Aivan kaikkia tietokoneiden tehtäviä ei voida formalisoida näin yksinkertaisesti. Ensinnäkin voi annet-

Seuraavaksi on tarkasteltava sitä, mitä tarkkaan ottaen tarkoitetaan ongelman tapausten ja vastausten "äärellisillä esityksillä". Edellä jo viitattiin siihen, että kaikki tietokoneen käsittelemä tieto on viime kädessä voitava koodata bittijonoiksi. Monesti on kuitenkin luontevaa sallia koodaukseen käytettävän myös muita merkkejä kuin 0 ja 1 (muut merkit voidaan tietenkin tarvittaessa edelleen esittää bittijonoina). Niinpä määritellään tässä "äärellisen esityksen" tarkoittavan jonkin aakkoston *merkkijonoa*.

Seuraavassa on lueteltu joitakin merkkijonoihin liittyviä peruskäsitteitä ja merkintöjä:

- *Aakkosto* (engl. alphabet, vocabulary) on mikä tahansa äärellinen, epätyhjä joukko alkeis*merkkejä* t. *symboleita*. Tärkeitä aakkostoja ovat esimerkiksi *binääriaakkosto* $\{0,1\}$ ja *latinalainen aakkosto* $\{\mathrm{A,B,\ldots,Z}\}$.

- *Merkkijono* (engl. string) on äärellinen järjestetty jono jonkin aakkoston merkkejä. Esimerkiksi "01001" ja "000" ovat binääriaakkoston merkkijonoja, ja "LTE" ja "XYZZY" ovat latinalaisen aakkoston merkkijonoja.

- Merkkijonon x *pituutta*, so. siihen sisältyvien merkkien määrää, merkitään $|x|$:llä. Esimerkiksi:
$$|01001| = |\mathrm{XYZZY}| = 5, \qquad |000| = |\mathrm{OTE}| = 3.$$

- Erikoistapaus minkä tahansa aakkoston merkkijonojen joukossa on *tyhjä merkkijono*, jonka paikka havaittavuuden parantamiseksi usein osoitetaan erikoismerkillä λ. Tyhjän merkkijonon pituus on tietenkin $|\lambda| = 0$.

- Merkkijonojen välinen perusoperaatio on *katenaatio* eli jonojen peräkkäin kirjoittaminen. Katenaation operaatiomerkkinä käytetään joskus selkeyden lisäämiseksi symbolia $\hat{}$. Esimerkkejä:

  (i) KALA$\hat{}$KUKKO = KALAKUKKO;

  (ii) jos $x = 00$ ja $y = 11$, niin $xy = 0011$ ja $yx = 1100$;

  (iii) kaikilla x on $x\lambda = \lambda x = x$;

  (iv) kaikilla x, y on $|xy| = |x| + |y|$.

- Aakkoston Σ kaikkien merkkijonojen joukkoa merkitään Σ^*:lla. Esimerkiksi jos $\Sigma = \{0,1\}$, niin $\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, \ldots\}$.

Laskennallisia ongelmia voidaan siis tarkastella yleisesti *kuvauksina*

$$\pi : \Sigma^* \to \Gamma^*,$$

missä Σ *ja* Γ *ovat aakkostoja*[2].

tuun ongelman tapaukseen liittyä useita kelvollisia vastauksia, jolloin ongelman esittämiseen tarvitaan funktion sijaan yleisempi *relaatio*. Tämä ei ole merkittävä laajennus. Mielenkiintoisemman formalisointihaasteen tarjoavat sellaiset "reaktiiviset" järjestelmät, joiden toiminta on jatkuvaa (käyttöjärjestelmät, prosessinohjaus, robotit). Näidenkin toimintaa voidaan tosin tarkastella jonona kuvauksenomaisia osatehtäviä, mutta toiminnan jatkuvuus tuo sen formalisointiin lisäpiirteitä.

[2] Entä jos kaikki joukon Σ^* merkkijonot eivät luontevasti vastaa tarkasteltavan "oikean" ongelman tapauksia, vaan jotkut ovat intuitiivisesti "syntaksivirheellisiä"? — Yksinkertaisin ratkaisu tässä tilanteessa on olettaa, että aakkostossa Γ on jokin erityinen "syntaksivirhemerkki" $\perp$, ja kuvata kaikki Σ^*:n virheelliset jonot merkkijonolle $\perp$.

Tärkeä laskennallisten ongelmien aliluokka ovat *päätösongelmat* (engl. decision problems), joissa kunkin ongelman tapauksen vastaus on yksinkertaisesti "kyllä" tai "ei", so. formaalisti ongelma on muotoa $\pi : \Sigma^* \to \{0,1\}$. Esimerkiksi päätösongelma "onko annettu luku alkuluku?" voidaan esittää aakkoston $\Sigma = \{0,1,2,\ldots,9\}$ kuvauksena

$$\pi : \Sigma^* \to \{0,1\}, \qquad \pi(x) = \begin{cases} 1, & \text{jos } x \text{ on alkuluku;} \\ 0, & \text{jos } x \text{ ei ole alkuluku.} \end{cases}$$

Jokaista päätösongelmaa $\pi : \Sigma^* \to \{0,1\}$ vastaa merkkijonojoukko

$$A_\pi = \{x \in \Sigma^* \mid \pi(x) = 1\},$$

so. niiden ongelman tapausten joukko, joihin vastaus on "kyllä", ja kääntäen jokaista merkkijonojoukkoa $A \subseteq \Sigma^*$ vastaa päätösongelma

$$\pi_A : \Sigma^* \to \{0,1\}, \qquad \pi_A(x) = \begin{cases} 1, & \text{jos } x \in A; \\ 0, & \text{jos } x \notin A. \end{cases}$$

Mielivaltaista merkkijonojoukkoa $A \subseteq \Sigma^*$ sanotaan aakkoston Σ *(formaaliksi) kieleksi* (engl. formal language), ja siihen liittyvää päätösongelmaa π_A sanotaan kielen A *tunnistusongelmaksi* (engl. recognition problem). Kielen ja sen tunnistusongelman vastaavuuden kautta voidaan formaalit kielet ja päätösongelmat samaistaa toisiinsa.

1.2 Laskennallisten ongelmien ratkeavuus

Sanotaan, että jollakin ohjelmointikielellä (esim. Pascal) kirjoitettu ohjelma P *ratkaisee* laskennallisen ongelman π, jos kullakin syötteellä x ohjelma P laskee ja tulostaa arvon $\pi(x)$. On luonnollista kysyä, voidaanko kaikki mahdolliset laskennalliset ongelmat ratkaista Pascal-ohjelmilla. Tämän kysymyksen selvittämiseksi tarkastellaan ensin kysymystä miten paljon laskennallisia ongelmia kaikkiaan on olemassa. Seuraavat joukkojen mahtavuuden käsitteet oletetaan pääpiirteissään ennestään tutuiksi.

Määritelmä 1.1 Ääretön joukko X on *numeroituva*, jos on olemassa bijektio $f : \mathbb{N} \to X$. Ääretön joukko, joka ei ole numeroituva on *ylinumeroituva.* Sanotaan yksinkertaisuuden vuoksi, että myös äärelliset joukot ovat numeroituvia.

Intuitiivisesti sanoen ääretön joukko X on numeroituva, jos sen alkiot voidaan järjestää ja indeksoida luonnollisilla luvuilla: $X = \{x_0, x_1, x_2, \ldots\}$.

Lause 1.1 *Minkä tahansa aakkoston Σ merkkijonojen joukko Σ^* on numeroituvasti ääretön.*

Todistus. Muodostetaan bijektio $f : \mathbb{N} \to \Sigma^*$ seuraavasti. Olkoon $\Sigma = \{a_1, a_2, \ldots, a_n\}$. Kiinnitetään Σ:n merkeille jokin "aakkosjärjestys"; olkoon se $a_1 < a_2 < \cdots < a_n$.

Joukon Σ^* merkkijonot voidaan nyt luetella valitun aakkosjärjestyksen suhteen *kanonisessa järjestyksessä* (engl. canonical t. lexicographic order) seuraavasti:

(i) ensin luetellaan 0:n mittaiset merkkijonot ($= \lambda$), sitten 1:n mittaiset ($= a_1, a_2, \ldots, a_n$), sitten 2:n mittaiset jne.;

(ii) kunkin pituusryhmän sisällä merkkijonot luetellaan aakkosjärjestyksessä.

Bijektio f on siis:

$$\begin{array}{rcl}
0 & \mapsto & \lambda \\
1 & \mapsto & a_1 \\
2 & \mapsto & a_2 \\
\vdots & & \vdots \\
n & \mapsto & a_n \\
n+1 & \mapsto & a_1 a_1 \\
n+2 & \mapsto & a_1 a_2 \\
\vdots & & \vdots \\
2n & \mapsto & a_1 a_n \\
2n+1 & \mapsto & a_2 a_1 \\
\vdots & & \vdots \\
3n & \mapsto & a_2 a_n \\
\vdots & & \vdots \\
n^2+n & \mapsto & a_n a_n \\
n^2+n+1 & \mapsto & a_1 a_1 a_1 \\
n^2+n+2 & \mapsto & a_1 a_1 a_2 \\
\vdots & & \vdots
\end{array} \qquad \square$$

Syntaktiselta kannalta myös millä tahansa ohjelmointikielellä kirjoitetut ohjelmat ovat vain kielen perusaakkoston (esim. Pascalissa ASCII-merkistön) merkkijonoja. Lauseen 1.1 mukaan minkä tahansa aakkoston merkkijonojen joukko on numeroituvasti ääretön, ja on helppo todeta, että numeroituvan joukon osajoukot ovat myös numeroituvia. Siten millä tahansa ohjelmointikielellä mahdollisten ohjelmien joukko on numeroituva.

Seuraavan lauseen mukaan kuitenkin kaikkien laskennallisten ongelmien joukko on ylinumeroituva. Laskennallisia ongelmia on siis tietyssä mielessä "enemmän" kuin niiden mahdollisia ratkaisuja, ja siksi *millään ohjelmointikielellä ei voida laatia ratkaisuja kaikille laskennallisille ongelmille.*

Lause 1.2 *Minkä tahansa aakkoston Σ päätösongelmien joukko (ja siis myös yleisemmin laskennallisten ongelmien joukko) on ylinumeroituva.*

Todistus (ns. Cantorin diagonaaliargumentti). Merkitään kaikkien Σ:n päätösongelmien kokoelmaa Π:llä:

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

Oletetaan, että Π olisi numeroituva, so. että olisi olemassa kaikki Π:n ongelmat kattava numerointi:

$$\Pi = \{\pi_0, \pi_1, \pi_2, \ldots\}.$$

Olkoot Σ^*:n merkkijonot edellisen todistuksen kanonisessa järjestyksessä lueteltuina $x_0, x_1, x_2, \ldots$. Muodostetaan uusi päätösongelma $\hat{\pi}$ seuraavasti:

$$\hat{\pi} : \Sigma^* \to \{0,1\}, \qquad \hat{\pi}(x) = \begin{cases} 1, & \text{jos } \pi_i(x_i) = 0; \\ 0, & \text{jos } \pi_i(x_i) = 1. \end{cases}$$

Koska $\hat{\pi} \in \Pi$ ja Π:n numerointi oletettiin kattavaksi, pitäisi olla $\hat{\pi} = \pi_k$ jollakin $k \in \mathbb{N}$. Mutta tällöin olisi $\hat{\pi}$:n määritelmän mukaan

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

Saadun ristiriidan takia oletus, että joukko Π on numeroituva, ei voi pitää paikkaansa.

Kuvallisesti todistuksen idea voidaan esittää seuraavasti: jos muodostetaan taulukko ongelmista $\pi_0, \pi_1, \pi_2, \ldots$ ja merkkijonoista $x_0, x_1, x_2, \ldots$, niin ongelma $\hat{\pi}$ poikkeaa kustakin ongelmasta π_i taulukon "diagonaalilla":

$\hat{\pi}$ $\searrow$	π_0	π_1	π_2	π_3	$\cdots$
x_0	$\not{1}$ 0	0	0	1	$\cdots$
x_1	0	$\not{0}$ 1	0	0	$\cdots$
x_2	1	1	$\not{0}$ 1	1	$\cdots$
x_3	0	0	0	$\not{1}$ 0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

□

Lauseiden 1.1 ja 1.2 mukaan siis kaikista laskennallisista ongelmista voidaan esimerkiksi Pascal-ohjelmilla ratkaista vain häviävän pieni osa (nimittäin ylinumeroituvan joukon numeroituva osajoukko). Tätä hämmentävää tulosta kohtaan voidaan yrittää esittää kaksi vastaväitettä:

1. Entä ratkeavuus kaikilla ohjelmointikielillä yhteensä? Ehkä kukin laskennallinen ohjelma voidaan ratkaista *jollakin* ohjelmointikielellä: jos ei Pascalilla, niin C:llä, LISPillä, Prologilla, tai jollakin olioperustaisella kielellä.

   Itse asiassa osoittautuu, että kaikki "riittävän vahvat" ohjelmointikielet määrittävät täsmälleen saman ratkeavien ongelmien luokan. Tätä ns. Churchin–Turingin teesiä käsitellään lisää monisteen luvussa 6; tulos perustuu siihen, että millä tahansa riittävän vahvalla ohjelmointikielellä voidaan kirjoittaa kääntäjä (oik. tulkkiohjelma) mille tahansa toiselle ohjelmointikielelle. Siten useimmat laskennalliset ongelmat ovat *absoluuttisesti ratkeamattomia.*

2. Ehkä edellä on vain määritelty "laskennallisen ongelman" käsite liian yleisesti; ehkä kaikki intuitiivisesti *mielenkiintoiset* ongelmat ovat kuitenkin ratkeavia?

   Tämäkään ei valitettavasti pidä paikkaansa. Todistettavasti ratkeamattomiin ongelmiin törmätään esimerkiksi ohjelmointikielten teoriassa ja tekoälytutkimuksessa jatkuvasti. Tunnetuin esimerkki konkreettisesta ratkemattomasta ongelmasta on ns. *pysähtymisongelma* (engl. halting problem). Tämän ongelman tapauksen muodostavat annettu ohjelma P ja sen syöte x; tehtävänä on ratkaista, pysähtyykö P:n laskenta syötteellä x, vai

jääkö se ikuiseen silmukkaan. Monisteen luvussa 6 annetaan työkaluja tämän tapaisten yksittäisten ongelmien todistamiseen ratkeamattomiksi ja esitetään lisää esimerkkejä tästä merkillisestä ilmiöstä[3].

Liite: Vakiintuneita merkintätapoja

Vaikka matemaattisille käsitteille käytetyt merkinnät ovatkin periaatteessa vapaasti valittavissa, on esityksen ymmärrettävyyden parantamiseksi tapana pitäytyä tietyissä käytännöissä. Aakkostoihin ja merkkijonoihin liittyville käsitteille ovat seuraavat merkintätavat vakiintuneet:

- Aakkostot: Σ, Γ, ...(isoja kreikkalaisia kirjaimia).
  Esimerkki: binääriaakkosto $\Sigma = \{0, 1\}$.
- Aakkoston koko (tai yleisemmin joukon mahtavuus): $|\Sigma|$.
- Alkeismerkit: a, b, c, ...(pieniä alkupään latinalaisia kirjaimia).
  Esimerkki: olkoon $\Sigma = \{a_1, \ldots, a_n\}$ aakkosto; tällöin $|\Sigma| = n$.
- Merkkijonot: u, v, w, x, y, ...(pieniä loppupään latinalaisia kirjaimia).
- Merkkijonojen katenaatio: $x\hat{}y$ tai vain xy.
- Merkkijonon pituus: $|x|$. *Esimerkkejä:*
  (i) $|abc| = 3$;
  (ii) olkoon $x = a_1 \ldots a_m$, $y = b_1 \ldots b_n$; tällöin $|xy| = m + n$.
- Tyhjä merkkijono: λ.
- Merkkijono, jossa on n kappaletta merkkiä a: a^n. *Esimerkkejä:*
  (i) $a^n = \underbrace{aa\ldots a}_{n \text{ kpl}}$;
  (ii) $|a^i b^j c^k| = i + j + k$.
- Merkkijonon x toisto k kertaa: x^k. *Esimerkkejä:*
  (i) $(ab)^2 = abab$;
  (ii) $|x^k| = k|x|$.
- Aakkoston Σ kaikkien merkkijonojen joukko: Σ^*.
  Esimerkki: $\{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$.

[3] Todettakoon, että pysähtymisongelma voidaan ratkaista "osittain" yksinkertaisesti suorittamalla annettu P syötteellä x, jolloin pysähtyminen voidaan havaita; mutta mikään äärellisessä ajassa toimiva algoritmi ei pysty kaikilla P ja x selvittämään, milloin P:n laskenta x:llä *ei* pysähdy. Kaikkia ongelmia ei voida ratkaista edes tässä mielessä osittain. Esimerkki vaikeammasta ongelmasta on ns. *totaalisuusongelma* (engl. total halting problem): pysähtyykö annetun ohjelman P laskenta *kaikilla* syötteillä x? — Siten on sitä jo ensimmäisellä ohjelmointikurssilla korostettavaa kelvollisen ohjelman perusominaisuutta, että ohjelma ei saa joutua ikuiseen silmukkaan, täysin mahdoton tarkastaa automaattisesti.

Luku 2

Äärelliset automaatit ja säännölliset kielet

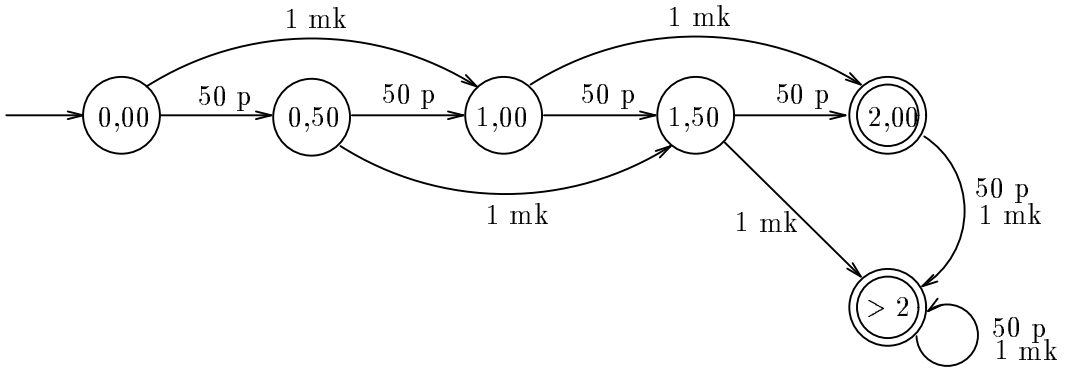
2.1 Tilasiirtymäkaaviot ja -taulukot

Tässä toisessa luvussa siirrytään yleisestä laskennallisten ongelmien tarkastelusta tutkimaan eräitä luonnollisia ongelmien kuvausformalismeja: sitä, miten näitä voidaan käyttää ongelmien ratkaisun perustana, ja mitä rajoituksia niillä on.

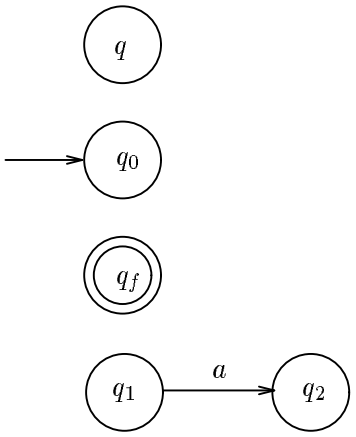
Yksinkertaisimpia laskentajärjestelmiä ovat sellaiset, joilla on vain äärellisen monta mahdollista tilaa. Tällaisen järjestelmän toiminta voidaan kuvata *äärellisenä automaattina* (engl. finite automaton). Äärellisillä automaateilla puolestaan on useita vaihtoehtoisia esitystapoja, joista *tilasiirtymäkaaviot* (engl. state transition diagrams) lienee havainnollisin.

Esimerkiksi kuvassa 2.1 on esitetty yksinkertaisen, kahden markan hintaista kahvia tarjoavan kahviautomaatin toimintaa kuvaava tilasiirtymäkaavio. Kaavioesityksessä käytettyjä merkintöjä on selostettu kuvassa 2.2. Automaatti ottaa vastaan syötteenä jonon 50 pennin ja yhden markan rahoja, ja "hyväksyy" syötejonon, jos siihen sisältyvien rahojen summa on vähintään kaksi markkaa. Automaatti ei anna vaihtorahaa ja tarjoilee vain yhdenlaista kahvia.

Päätösongelmanäkökulmasta voidaan ajatella, että kuvan 2.1 automaatti ratkaisee ongelman "riittävätkö annetut rahat kahvin ostamiseen?" Äärellisiä automaatteja voidaan yleensä-



Kuva 2.1: Yksinkertaisen kahviautomaatin tilasiirtymäkaavio.



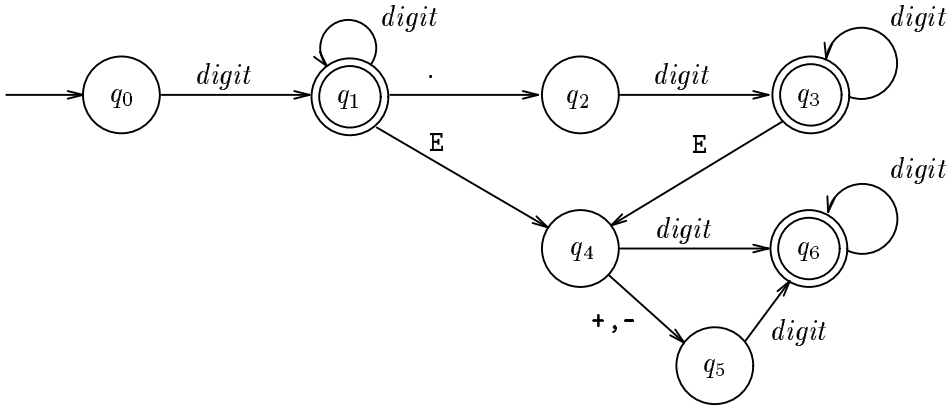
Automaatin tila nimeltä q

Alkutila

Lopputila: automaatti "hyväksyy" syötejonon, joss se jonon loppuessa on tällaisessa tilassa

Syötemerkin a aikaansaama siirtymä tilasta q_1 tilaan q_2

Kuva 2.2: Tilasiirtymäkaavioiden merkinnät.

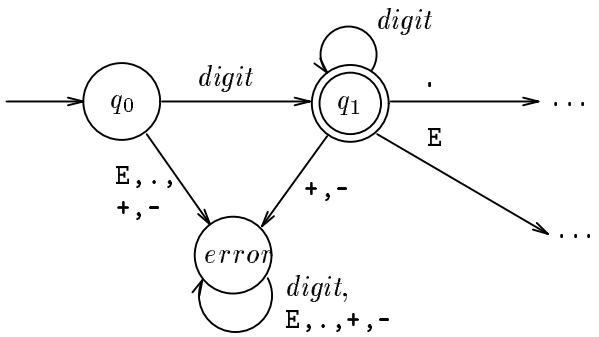


Kuva 2.3: Etumerkittömät reaaliluvut tunnistava automaatti.

kin käyttää yksinkertaisten päätösongelmien ratkaisujen mallintamiseen. Automaattimallista on muitakin kuin 0/1-arvoisten funktioiden kuvaamiseen tarkoitettuja versioita (ns. Moore- ja Mealy-automaatit), mutta niitä ei käsitellä tässä monisteessa.

Toisena esimerkkinä tarkastellaan kuvassa 2.3 esitettyä automaattia, joka tutkii, onko syötteenä annettu merkkijono Pascalin syntaksin mukainen etumerkitön reaaliluku. Automaatin esityksessä on käytetty lyhennettä *digit* merkkijoukolle $\{\texttt{0}, \texttt{1}, \ldots, \texttt{9}\}$.

Äärellinen automaatti voidaan esittää myös *tilasiirtymätaulukkona* (engl. state transition table), joka kuvaa automaatin uuden tilan vanhan tilan ja syötemerkin funktiona. Esimerkiksi



Kuva 2.4: Äärellisen automaatin varustaminen virhetilalla.

kuvan 2.3 automaatin tilasiirtymätaulukkoesitys olisi:

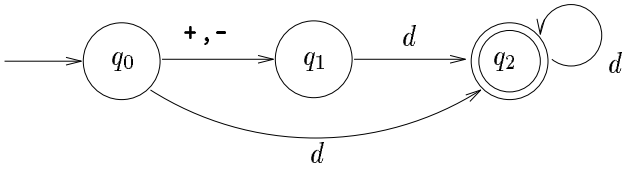
		digit	.	E	+	−
→	q_0	q_1				
←	q_1	q_1	q_2	q_4		
	q_2	q_3				
←	q_3	q_3		q_4		
	q_4	q_6			q_5	q_5
	q_5	q_6				
←	q_6	q_6				

Tilasiirtymätaulukon tyhjät paikat, tai vastaavasti tilasiirtymäkaavion "puuttuvat" kaaret, kuvaavat automaatin virhetilanteita. Jos automaatti ohjautuu tällaiseen paikkaan, syötejono ei kuulu automaatin hyväksymään joukkoon. Muodollisesti voidaan ajatella automaatissa olevan erityinen virhetila *error*, jota ei vain kaavion selkeyden vuoksi merkitä näkyviin. Esimerkiksi kuvan 2.3 automaatin täydellinen kaavioesitys olisi tällöin kuvan 2.4 mukainen, ja taulukkoesitys seuraavanlainen:

		digit	.	E	+	−
→	q_0	q_1	*error*	*error*	*error*	*error*
←	q_1	q_1	q_2	q_4	*error*	*error*
⋮	⋮	⋮	⋮	⋮	⋮	⋮
←	q_6	q_6	*error*	*error*	*error*	*error*
	error	*error*	*error*	*error*	*error*	*error*

2.2 Äärellisiin automaatteihin perustuva ohjelmointi

Annetun äärellisen automaatin pohjalta on helppo laatia automaatin toimintaa vastaava ohjelma. Esimerkiksi kuvan 2.3 automaatin perusteella voitaisiin ohjelmoida seuraavanlainen testi sille, onko syötteenä annettu merkkijono Pascal-kielen syntaksin mukainen reaaliluvun esitys:



Kuva 2.5: Etumerkilliset kokonaisluvut tunnistava automaatti.

```
digits := ['0',...,'9'];
q := 0;
c := getchar;                    {Funktio getchar palauttaa syötevirran
                                    seuraavan merkin.}
while c <> eof do
begin
    case q of
    0:  if c in digits then q := 1
        else q := 99;
    1:  if c in digits then q := 1
        else if c = '.' then q := 2
        else if c = 'E' then q := 4
        else q := 99;
    ...
    99:
    end case;
    c := getchar
end while;
if q = 1 or q = 3 or q = 6 then writeln('SYÖTE ON REAALILUKU')
else writeln('SYÖTE EI OLE REAALILUKU').
```

Äärellisten automaattien pohjalta laadittuihin ohjelmiin voidaan liittää "syntaktisen" merkkijonon oikeellisuuden testaamisen rinnalle myös "semanttisia" toimintoja. Tarkastellaan esimerkkinä kuvassa 2.5 esitettyä, Pascal-kielen syntaksin mukaisia etumerkillisiä kokonaislukuja tunnistavaa äärellistä automaattia. (Kuvassa on käytetty lyhennysmerkintää d merkkijoukolle $\{0, 1, \ldots, 9\}$.)

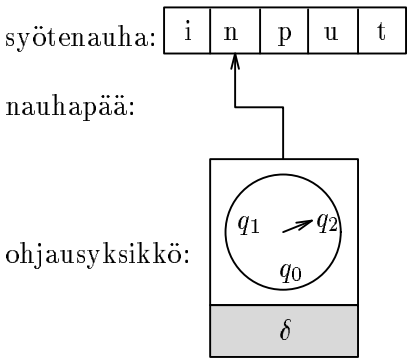
Tämä automaatti voidaan toteuttaa ohjelmana edellisen esimerkin tapaan:

```
digits := ['0',...,'9'];
q := 0;
c := getchar;
while c <> eof do
begin
    case q of
    0:  if c = '+' or c = '-' then q := 1
        else if c in digits then q := 2
        else q := 99;
    1:  if c in digits then q := 2
        else q := 99;
    2:  if c in digits then q := 2
        else q := 99;
```

```
        99:
        end case;
        c := getchar
    end while;
    if q = 2 then {OK}
    else writeln('VIRHEELLINEN LUKU').
```

Toteutukseen voidaan nyt helposti liittää esimerkiksi syötteenä saadun luvun arvon laskevat operaatiot (merkitty seuraavassa "→"-merkillä):

```
    digits := ['0',...,'9'];
→   sgn := 1; val := 0;                {sgn = etumerkki, val = luvun itseisarvo}
    q := 0;
    c := getchar;
    while c <> eof do
    begin
        case q of
        0:  if c = '+' then q := 1
            else if c = '-' then
→           begin sgn := -1; q := 1 end
            else if c in digits then
            begin
→               val := ord(c) - ord('0');
                q := 2
            end
            else q := 99;
        1:  if c in digits then
            begin
→               val := ord(c) - ord('0');
                q := 2
            end
            else q := 99;
        2:  if c in digits then
            begin
→               val := 10 * val + (ord(c) - ord('0'));
                q := 2
            end
            else q := 99;
        99:
        end case;
        c := getchar;
    end while;
→   if q = 2 then writeln('LUVUN ARVO ON ', sgn * val)
    else writeln('VIRHEELLINEN LUKU').
```



Kuva 2.6: Äärellinen automaatti.

2.3 Äärellisen automaatin käsitteen formalisointi

Jotta äärellisen automaatin käsitteen tarjoamat mahdollisuudet voitaisiin täysin hyödyntää ja sen rajoitukset saataisiin selville, käsite täytyy formalisoida. Formalisointi perustuu seuraavaan mekanistiseen malliin automaatista ja sen toiminnasta (ks. kuva 2.6): äärellinen automaatti M koostuu äärellistilaisesta *ohjausyksiköstä*, jonka toimintaa säätelee automaatin *siirtymäfunktio* δ, sekä merkkipaikkoihin jaetusta *syötenauhasta* ja nämä yhdistävästä *nauhapäästä*, joka kullakin hetkellä osoittaa yhtä syötenauhan merkkiä[1].

Automaatti käynnistetään erityisessä *alkutilassa* q_0, siten että tarkasteltava syöte on kirjoitettuna syötenauhalle ja nauhapää osoittaa sen ensimmäistä merkkiä.

Yhdessä toiminta-askelessa automaatti lukee nauhapään kohdalla olevan syötemerkin, päättää ohjausyksikön tilan ja luetun merkin perusteella siirtymäfunktion mukaisesti ohjausyksikön uudesta tilasta, ja siirtää nauhapäätä yhden merkin eteenpäin.

Automaatti pysähtyy, kun viimeinen syötemerkki on käsitelty. Jos ohjausyksikön tila tällöin kuuluu erityiseen *(hyväksyvien) lopputilojen* joukkoon, automaatti *hyväksyy* syötteen, muuten *hylkää* sen.

Automaatin *tunnistama kieli* on sen hyväksymien merkkijonojen joukko.

Täsmällisesti, mekanistisia rinnastuksia käyttämättä, nämä käsitteet voidaan muotoilla seuraavasti:

Määritelmä 2.1 *Äärellinen automaatti* (engl. finite automaton) on viisikko

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

missä

- Q on automaatin *tilojen* (engl. states) äärellinen joukko;
- Σ on automaatin *syöteaakkosto* (engl. input alphabet);
- $\delta : Q \times \Sigma \to Q$ on automaatin *siirtymäfunktio* (engl. transition function);
- $q_0 \in Q$ on automaatin *alkutila* (engl. initial state);
- $F \subseteq Q$ on automaatin *(hyväksyvien) lopputilojen* (engl. accepting final states) joukko.

[1] Äärellisen automaatin ydin on sen "ohjelma", siirtymäfunktio δ. Edellä esitetyt tilasiirtymäkaaviot ja -taulukot ovat juuri tämän siirtymäfunktion vaihtoehtoisia esitystapoja.

Esimerkiksi kuvassa 2.3 esitetyn etumerkittömiä reaalilukuja tunnistavan automaatin formaali esitys olisi:

$$M = (\{q_0, \ldots, q_6, error\}, \{\texttt{0},\texttt{1},\ldots,\texttt{9},\texttt{.},\texttt{E},\texttt{+},\texttt{-}\}, \delta, q_0, \{q_1, q_3, q_6\}),$$

missä δ on kuten taulukossa sivulla 9 on esitetty; esimerkiksi

$$\delta(q_0, \texttt{0}) = \delta(q_0, \texttt{1}) = \cdots = \delta(q_0, \texttt{9}) = q_1,$$

$$\delta(q_0, \texttt{.}) = error, \quad \delta(q_1, \texttt{.}) = q_2, \quad \delta(q_1, \texttt{E}) = q_4 \quad \text{jne.}$$

Automaatin *tilanne* (engl. configuration) on pari $(q, w) \in Q \times \Sigma^*$; erityisesti automaatin *alkutilanne syötteellä* x on pari (q_0, x). Tilanteen (q, w) intuitiivinen tulkinta on, että q on automaatin tila ja w on syötemerkkijonon jäljellä oleva, so. nauhapäästä oikealle sijaitseva osa.

Tilanne (q, w) *johtaa suoraan* tilanteeseen (q', w'), merkitään

$$(q, w) \underset{M}{\vdash} (q', w'),$$

jos on $w = aw'$ $(a \in \Sigma)$ ja $q' = \delta(q, a)$. Tällöin sanotaan myös, että tilanne (q', w') on tilanteen (q, w) *välitön seuraaja* (engl. immediate successor). Relaation $\underset{M}{\vdash}$ intuitiivinen tulkinta on, että automaatti ollessaan tilassa q ja lukiessaan nauhalla olevan merkkijonon $w = aw'$ ensimmäisen merkin a siirtyy tilaan q' ja siirtää nauhapäätä yhden askelen eteenpäin, jolloin nauhalle jää merkkijono w'. Jos automaatti M on yhteydestä selvä, relaatiota voidaan merkitä yksinkertaisesti

$$(q, w) \vdash (q', w').$$

Tilanne (q, w) *johtaa tilanteeseen* (q', w'), tai tilanne (q', w') on tilanteen (q, w) *seuraaja* (engl. successor), merkitään

$$(q, w) \underset{M}{\vdash^*} (q', w'),$$

jos on olemassa välitilannejono (q_0, w_0), (q_1, w_1), $\ldots$, (q_n, w_n), $n \geq 0$, siten että

$$(q, w) = (q_0, w_0) \underset{M}{\vdash} (q_1, w_1) \underset{M}{\vdash} \cdots \underset{M}{\vdash} (q_n, w_n) = (q', w').$$

Erikoistapauksena $n = 0$ saadaan $(q, w) \underset{M}{\vdash^*} (q, w)$ millä tahansa tilanteella (q, w). Jälleen, jos automaatti M on yhteydestä selvä, merkitään yksinkertaisesti

$$(q, w) \vdash^* (q', w').$$

Automaatti M *hyväksyy* (engl. accepts) merkkijonon $x \in \Sigma^*$, jos on voimassa

$$(q_0, x) \underset{M}{\vdash^*} (q_f, \lambda) \qquad \text{jollakin } q_f \in F;$$

muuten M *hylkää* (engl. rejects) x:n. Toisin sanoen: automaatti hyväksyy x:n, jos sen alkutilanne syötteellä x johtaa, syötteen loppuessa, johonkin hyväksyvään lopputilanteeseen.

Automaatin M *tunnistama kieli* (engl. language recognized by M) määritellään:

$$L(M) = \{x \in \Sigma^* \mid (q_0, x) \underset{M}{\vdash^*} (q_f, \lambda) \quad \text{jollakin } q_f \in F\}.$$

Esimerkkinä tarkastellaan merkkijonon "`0.25E2`" käsittelyä kuvan 2.3 mukaisella reaalilukuautomaatilla:

$$\begin{array}{lllll} (q_0, \texttt{0.25E2}) & \vdash & (q_1, \texttt{.25E2}) & \vdash & (q_2, \texttt{25E2}) \\ & \vdash & (q_3, \texttt{5E2}) & \vdash & (q_3, \texttt{E2}) \\ & \vdash & (q_4, \texttt{2}) & \vdash & (q_6, \lambda). \end{array}$$

Koska $q_6 \in F = \{q_1, q_3, q_6\}$, on siis $\texttt{0.25E2} \in L(M)$.

2.4 Äärellisten automaattien minimointi

Annetun äärellisen automaatin kanssa ekvivalentin (so. saman kielen tunnistavan), mutta tilamäärältään minimaalisen automaatin muodostaminen on sekä käytännössä että teoreettiselta kannalta erittäin tärkeä tehtävä.

Tehtävä voidaan ratkaista seuraavassa esitettävällä tehokkaalla menetelmällä. Menetelmän perusideana on pyrkiä samaistamaan keskenään sellaiset syötteenä annetun automaatin tilat, joista lähtien automaatti toimii täsmälleen samoin kaikilla merkkijonoilla.

Täsmällisemmin sanoen: olkoon

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

jokin äärellinen automaatti. Merkintöjen yksinkertaistamiseksi laajennetaan automaatin siirtymäfunktio yksittäisistä syötemerkeistä merkkijonoihin: jos $q \in Q$, $x \in \Sigma^*$, merkitään

$$\delta^*(q, x) = \text{ se } q' \in Q, \text{ jolla } (q, x) \underset{M}{\vdash^*} (q', \lambda).$$

Automaatin M tilat q ja q' ovat *ekvivalentit*, merkitään

$$q \equiv q',$$

jos kaikilla $x \in \Sigma^*$ on

$$\delta^*(q, x) \in F \quad \text{jos ja vain jos} \quad \delta^*(q', x) \in F;$$

toisin sanoen, jos automaatti q:sta ja q':sta lähtien hyväksyy täsmälleen samat merkkijonot.

Määritellään myös lievempi k-ekvivalenssiehto: tilat q ja q' ovat *k-ekvivalentit*, merkitään

$$q \overset{k}{\equiv} q',$$

jos kaikilla $x \in \Sigma^*$, $|x| \leq k$, on

$$\delta^*(q, x) \in F \quad \text{jos ja vain jos} \quad \delta^*(q', x) \in F;$$

toisin sanoen, jos mikään enintään k:n pituinen merkkijono ei pysty erottamaan tiloja toisistaan.

Ilmeisesti on:

$$\begin{array}{lllll} \text{(i)} & q \overset{0}{\equiv} q' & \text{joss} & \text{sekä } q \text{ että } q' \text{ ovat lopputiloja} & \\ & & & \text{tai kumpikaan ei ole; ja} & (2.1) \\ \text{(ii)} & q \equiv q' & \text{joss} & q \overset{k}{\equiv} q' \text{ kaikilla } k = 0, 1, 2, \ldots & \end{array}$$

Seuraava minimointialgoritmi perustuu syötteenä annetun automaatin tilojen k-ekvivalenssiluokkien hienontamiseen $(k+1)$-ekvivalenssiluokiksi kunnes saavutetaan täysi ekvivalenssi.

Algoritmi 2.1 (Äärellisen automaatin minimointi)

Syöte: Äärellinen automaatti $M = (Q, \Sigma, \delta, q_0, F)$.

Tulos: M:n kanssa ekvivalentti äärellinen automaatti $\widehat{M}$, jossa on minimimäärä tiloja.

Menetelmä:

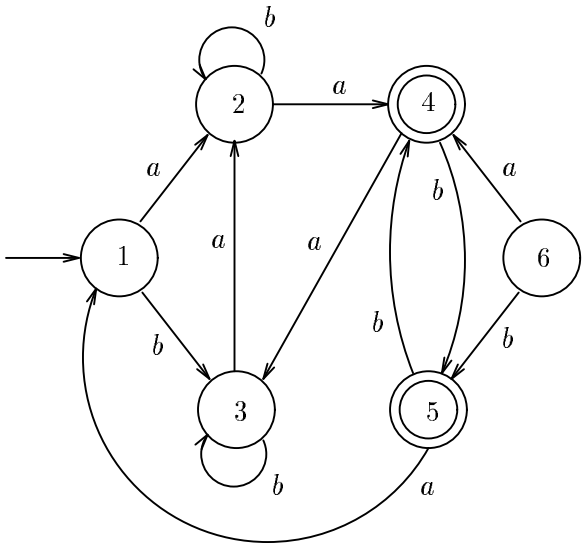
1. [Turhien tilojen poisto.] Poista M:stä kaikki tilat, joita ei voida saavuttaa tilasta q_0 millään syötemerkkijonolla.
2. [0-ekvivalenssi.] Osita M:n jäljelle jääneet tilat kahteen luokkaan: ei-lopputiloihin ja lopputiloihin.
3. [k-ekvivalenssi $\rightarrow$ $(k+1)$-ekvivalenssi.] Tarkastele M:n tilasiirtymien käyttäytymistä muodostetun osituksen suhteen: jos tilasiirtymät ovat täysin yhteensopivia osituksen kanssa, so. jos samaan luokkaan kuuluvista tiloista siirrytään samoilla merkeillä aina samanluokkaisiin tiloihin, niin algoritmi päättyy ja minimiautomaatin $\widehat{M}$ tiloiksi tulevat muodostuneet M:n tilojen *luokat*. $\widehat{M}$:n siirtymäfunktio saadaan M:n siirtymäfunktiosta, joka oletuksen mukaan on yhteensopiva syntyneen luokituksen kanssa. $\widehat{M}$:n alku- ja lopputilat määräytyvät samoin M:n alku- ja lopputilojen perusteella.

   Jos taas osituksen luokat sisältävät keskenään eri lailla käyttäytyviä tiloja, hienonna ositusta edelleen jakamalla kunkin luokan sisällä erityyppiset tilat eri luokkiin. Palaa suorittamaan askel 3 uudestaan; muista erityisesti toistaa tilasiirtymätarkastelu uuden osituksen suhteen.

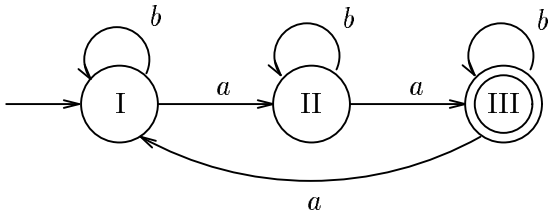
On melko helppo osoittaa, että askelen 3 $(k+1)$:nnen suorituskerran $(k = 0, 1, \ldots)$ alussa kaksi tilaa kuuluu samaan muodostetun osituksen luokkaan, jos ja vain jos ne ovat k-ekvivalentteja. Tästä seuraa edelleen, että algoritmin suorituksen päättyessä, kun ositus ei enää hienone, muodostuneet tilaluokat ovat täsmälleen M:n tilojen $\equiv$-ekvivalenssiluokat (vrt. ominaisuus (2.1.ii)). Algoritmin suoritus päättyy välttämättä aina, sillä kullakin askelen 3 suorituskerralla, viimeistä lukuunottamatta, vähintään yksi tilaluokka ositetaan pienemmäksi.

Esimerkkinä algoritmin toiminnasta tarkastellaan kuvan 2.7 automaatin minimointia. Ensimmäisessä vaiheessa automaatista poistetaan tila 6, johon ei päästä millään merkkijonolla. Toisessa vaiheessa ositetaan automaatin tilat 1–5 ei-lopputiloihin (luokka I) ja lopputiloihin (luokka II), ja tarkastetaan siirtymien käyttäytyminen osituksen suhteen:

			a	b
I:	$\rightarrow$	1	2,I	3,I
		2	4,II	2,I
		3	2,I	3,I
II:	$\leftarrow$	4	3,I	5,II
	$\leftarrow$	5	1,I	4,II



Kuva 2.7: Redundantti äärellinen automaatti M.



Kuva 2.8: Minimiautomaatti $\widehat{M}$.

Luokassa I on nyt kahdentyyppisiä tiloja ($\{1,3\}$ ja $\{2\}$), joten ositusta täytyy hienontaa ja tarkastaa siirtymät uuden osituksen suhteen:

			a	b
I :	$\rightarrow$	1	2, II	3, I
		3	2, II	3, I
II :		2	4, III	2, II
III :	$\leftarrow$	4	3, I	5, III
	$\leftarrow$	5	1, I	4, III

Nyt kunkin luokan sisältämät tilat ovat keskenään samanlaisia, joten minimointialgoritmi päättyy; saatu minimiautomaatti on esitetty tilasiirtymäkaaviona kuvassa 2.8.

Todistetaan vielä Algoritmin 2.1 oikeellisuutta koskeva keskeinen tulos:

Lause 2.1 *Algoritmi 2.1 muodostaa annetun äärellisen automaatin M kanssa ekvivalentin äärellisen automaatin $\widehat{M}$, jossa on minimimäärä tiloja. Tämä automaatti on tilojen nimeämistä paitsi yksikäsitteinen.*

∗ *Todistus.* Sivuutetaan suoraviivainen ekvivalenssitarkastelu ja keskitytään muodostetun automaatin minimaalisuuteen ja yksikäsitteisyyteen.

Olkoon siis algoritmin tuottama automaatti

$$\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{F}),$$

ja olkoon

$$\widetilde{M} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{F})$$

toinen M:n kanssa ekvivalentti automaatti, jolla $|\tilde{Q}| \leq |\hat{Q}|$. Automaatin $\widehat{M}$ minimaalisuus ja (rakenteellinen) yksikäsitteisyys tulee todistetuksi, jos voidaan muodostaa tilojen vastaavuuskuvaus

$$f : \tilde{Q} \to \hat{Q},$$

jolla on se ominaisuus, että kaikilla $\tilde{q} \in \tilde{Q}$, $a \in \Sigma$ on

$$f(\tilde{\delta}(\tilde{q}, a)) = \hat{\delta}(f(\tilde{q}), a). \tag{2.2}$$

Tällainen kuvaus on välttämättä surjektio, koska $\widehat{M}$:ssa ei ole saavuttamattomia tiloja[2], ja koska se on surjektio ja $|\tilde{Q}| \leq |\hat{Q}|$, se on myös injektio; siis bijektio ja isomorfismi.

Muodostetaan kuvaus f yksinkertaisesti seuraavasti: olkoon $\tilde{q} \in \tilde{Q}$ jokin automaatin $\widetilde{M}$ tila ja $x \in \Sigma^*$ jokin merkkijono, jolla $\tilde{q} = \tilde{\delta}^*(\tilde{q}_0, x)$. (Voidaan olettaa, että myöskään automaatissa $\widetilde{M}$ ei ole saavuttamattomia tiloja.) Asetetaan

$$f(\tilde{q}) = \hat{\delta}^*(\hat{q}_0, x);$$

"jäljitellään" siis tilan $\tilde{q}$ saavuttamista $\widehat{M}$:ssa.

On osoitettava, että kuvaus f on hyvin määritelty, so. että eri merkkijonot tilan $\tilde{q}$ saavuttamiseen $\widetilde{M}$:ssa eivät johda eri tiloihin $\widehat{M}$:ssa. Tehdään vastaoletus, että olisi $x, y \in \Sigma^*$, $x \neq y$, joilla

$$\tilde{\delta}^*(\tilde{q}_0, x) = \tilde{\delta}^*(\tilde{q}_0, y),$$

mutta

$$\hat{\delta}^*(\hat{q}_0, x) \neq \hat{\delta}^*(\hat{q}_0, y). \tag{2.3}$$

Koska $\widehat{M}$:n tilat vastaavat alkuperäisen automaatin M tilojen $\equiv$-ekvivalenssiluokkia, ehto (2.3) merkitsee että M:n tilat

$$q_x = \delta^*(q_0, x) \text{ ja } q_y = \delta^*(q_0, y)$$

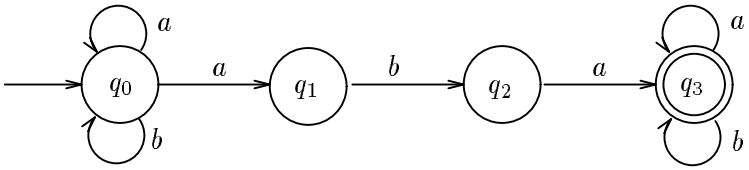
eivät ole ekvivalentteja, so. että on olemassa merkkijono $z \in \Sigma^*$, jolla

$$\delta^*(q_x, z) \in F \text{ ja } \delta^*(q_y, z) \notin F$$

(tai toisinpäin). Siten automaatti M hyväksyy merkkijonon xz, jos ja vain jos se hylkää merkkijonon yz.

Mutta koska merkkijonot x ja y johtavat automaatin $\widetilde{M}$ samaan tilaan, se hyväksyy merkkijonon xz, jos ja vain jos se hyväksyy merkkijonon yz. Siis $\widetilde{M}$ ei olekaan ekvivalentti

[2]Kuvauksen f surjektiivisuus seuraa tästä, koska jos tila $\hat{q} \in \hat{Q}$ voidaan saavuttaa tilasta $\hat{q}_0$ merkkijonolla x ja merkitään $\tilde{q} = \tilde{\delta}^*(\tilde{q}_0, x)$, niin on $\hat{q} = f(\tilde{q})$.



Kuva 2.9: Yksinkertainen epädeterministinen automaatti.

M:n kanssa. Saadusta ristiriidasta seuraa, että vastaoletus on väärä, ja kuvaus f on hyvin määritelty.

Lopuksi on helppo tarkastaa, että kuvaus f täyttää isomorfiaehdon (2.2). Olkoon nimittäin $\tilde{q} = \tilde{\delta}^*(\tilde{q}_0, x)$, jolloin siis $f(\tilde{q}) = \hat{\delta}^*(\hat{q}_0, x)$. Tällöin on

$$\hat{\delta}(f(\tilde{q}), a) = \hat{\delta}(\hat{\delta}^*(\hat{q}_0, x), a) = \hat{\delta}^*(\hat{q}_0, xa) = f(\tilde{\delta}^*(\tilde{q}_0, xa)) = f(\tilde{\delta}(\tilde{\delta}^*(\tilde{q}_0, x), a)) = f(\tilde{\delta}(\tilde{q}, a)).$$

□

2.5 Epädeterministiset äärelliset automaatit

Tarkastellaan seuraavaa, esimerkiksi tekstinkäsittelyjärjestelmien toteutuksessa esiintyvää tehtävää: on laadittava automaatti, joka tutkii esiintyykö syötteenä annetussa, aakkoston $\{a, b\}$ merkkijonossa osajonoa aba. Ensimmäinen ratkaisuyritys tähän tehtävään voisi olla kuvan 2.9 tapainen. Tässä muuten luontevassa automaatissa on kuitenkin tilasta q_0 *kaksi siirtymää merkillä* a (tiloihin q_0 ja q_1) — automaatti on *epädeterministinen*.

Tällainen epädeterminismi ei ole edellä esitetyn automaattiformalismin mukaan sallittua, eikä päällisin puolin katsoen oikein järkevääkään: miten automaatti voisi "tietää", kumpaa vaihtoehtoista siirtymää kulloinkin lähteä seuraamaan? Esimerkki antaa kuitenkin viitteen siitä, että vaikka epädeterministisiä automaatteja ei voikaan sellaisinaan toteuttaa ohjelmallisesti, ne ovat joissakin tilanteissa kätevä *kuvausformalismi*.

Seuraavassa esitettävät lähemmät tarkastelut vahvistavat tämän: paitsi että ovat sellaisenaankin hyödyllinen käsite, epädeterministiset automaatit auttavat rakentamaan tärkeän yhteyden tavallisten determinististen äärellisten automaattien ja ns. *säännöllisten lausekkeiden* välille (luku 2.7).

Epädeterministiset automaatit ovat siis muuten samanlaisia kuin deterministisetkin, mutta niiden siirtymäfunktio δ ei liitä automaatin vanhan tilan ja syötemerkin muodostamiin pareihin yksikäsitteistä uutta tilaa, vaan *joukon* mahdollisia seuraavia tiloja. Epädeterministinen automaatti hyväksyy syötteensä, jos *jokin* mahdollisten tilojen jono johtaa hyväksyvään lopputilaan.

Esimerkiksi kuvan 2.9 automaatti hyväksyy syötejonon $aaba$, koska sen on mahdollista edetä seuraavasti:

$$(q_0, aaba) \vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_2, a) \vdash (q_3, \lambda).$$

Automaatin olisi tosin mahdollista päätyä myös hylkäävään tilaan:

$$(q_0, aaba) \vdash (q_0, aba) \vdash (q_0, ba) \vdash (q_0, a) \vdash (q_0, \lambda),$$

mutta tästä mahdollisuudesta ei tarvitse välittää — voidaan ajatella, että automaatti osaa "ennustaa" ja valita aina parhaan mahdollisen vaihtoehdon.

Täsmällisesti ottaen asetetaan seuraavat määritelmät: merkitään joukon A potenssijoukkoa $\mathcal{P}(A)$:lla; siis

$$\mathcal{P}(A) = \{B \mid B \subseteq A\}.$$

Määritelmä 2.2 *Epädeterministinen äärellinen automaatti* (engl. nondeterministic finite automaton) on viisikko

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

missä

- Q on äärellinen *tilojen* joukko;
- Σ on *syöteaakkosto*;
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ (huom.!) on automaatin (joukkoarvoinen) *siirtymäfunktio*;
- $q_0 \in Q$ on *alkutila*;
- $F \subseteq Q$ on *(hyväksyvien) lopputilojen* joukko.

Esimerkiksi kuvan 2.9 automaatin siirtymäfunktio voitaisiin esittää seuraavana taulukkona:

		a	b
$\rightarrow$	q_0	$\{q_0, q_1\}$	$\{q_0\}$
	q_1	$\emptyset$	$\{q_2\}$
	q_2	$\{q_3\}$	$\emptyset$
$\leftarrow$	q_3	$\{q_3\}$	$\{q_3\}$

Taulukosta voidaan lukea, että esimerkiksi $\delta(q_0, a) = \{q_0, q_1\}$ ja $\delta(q_1, a) = \emptyset$. (Epädeterministisissä automaateissa voidaan virhetilanteen ilmaisemiseen käyttää erityisen virhetilan sijaan tyhjää seuraajatilajoukkoa.)

Muut epädeterministisiin automaatteihin liittyvät määritelmät ovat yhtä poikkeusta lukuunottamatta samat kuin deterministisilläkin automaateilla. Poikkeuksen muodostaa suoran johtamisen, tai tilanteen välittömän seuraajan määritelmä, jossa sallitaan useita seuraajavaihtoehtoja: epädeterministisen automaatin tilanne (q, w) voi *johtaa suoraan* tilanteeseen (q', w'), merkitään

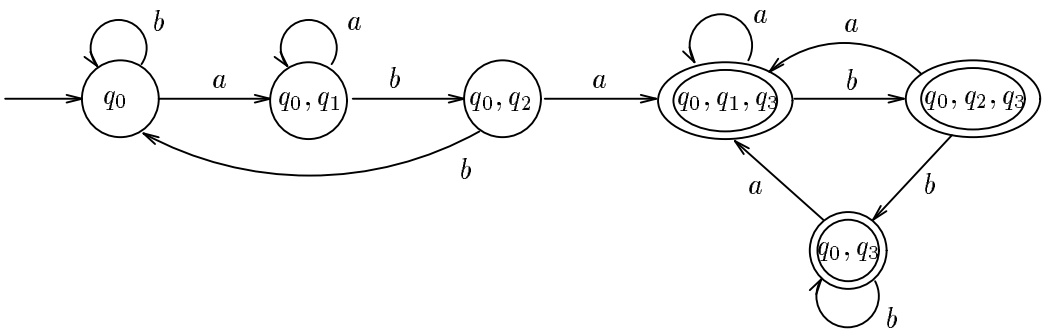
$$(q, w) \underset{M}{\vdash} (q', w'),$$

jos on $w = aw'$ $(a \in \Sigma)$ ja $q' \in \delta(q, a)$ (huom.!); tällöin sanotaan myös, että tilanne (q', w') on tilanteen (q, w) mahdollinen *välitön seuraaja*.

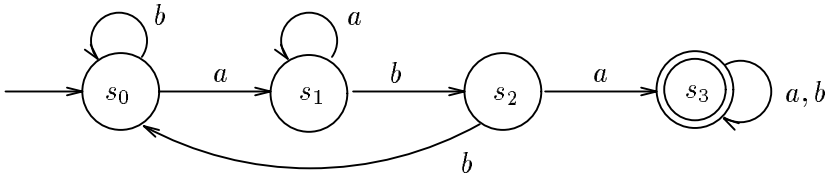
Useamman askelen mittaiset tilannejohdot, merkkijonojen hyväksyminen ja hylkääminen ym. käsitteet määritellään aivan samoin sanoin kuin aiemminkin, mutta koska perustava yhden askelen johdon määritelmä nyt on toinen, niiden sisältö muovautuu hieman erilaiseksi.

Koska deterministiset äärelliset automaatit ovat epädeterminististen erikoistapaus, on selvää, että kaikki edellisillä tunnistettavat kielet voidaan tunnistaa myös jälkimmäisillä. Tärkeä ja yllättävä tulos on kuitenkin, että myös käänteinen väite pätee: *deterministiset ja epädeterministiset äärelliset automaatit ovat yhtä vahvoja.*

Lause 2.2 *Olkoon $A = L(M)$ jonkin epädeterministisen äärellisen automaatin M tunnistama kieli. Tällöin on olemassa myös deterministinen äärellinen automaatti $\widehat{M}$, jolla $A = L(\widehat{M})$.*



Kuva 2.10: Determinisoitu automaatti $\widehat{M}$.



Kuva 2.11: Minimoitu ja uudelleennimetty deterministinen automaatti.

Todistus. Olkoon $A = L(M)$, $M = (Q, \Sigma, \delta, q_0, F)$. Todistuksen ideana on laatia deterministinen äärellinen automaatti $\widehat{M}$, joka simuloi M:n toimintaa kaikissa sen kullakin hetkellä mahdollisissa tiloissa *rinnakkain*.

Formaalisti tämä toteutetaan siten, että automaatin $\widehat{M}$ tilat vastaavat M:n tilojen *joukkoja*:

$$\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \widehat{F}),$$

missä

$$\begin{aligned} \hat{Q} &= \mathcal{P}(Q) = \{S \mid S \subseteq Q\}, \\ \hat{q}_0 &= \{q_0\}, \\ \widehat{F} &= \{S \subseteq Q \mid S \text{ sisältää jonkin } q_f \in F\}, \\ \hat{\delta}(S, a) &= \bigcup_{q \in S} \delta(q, a). \end{aligned}$$

Esimerkki. Ennen todistuskonstruktion oikeellisuuden tarkastamista tarkastellaan esimerkkinä kuvan 2.9 automaatin M determinisointia.

Muodollisesti M:n deterministisen vastinautomaatin $\widehat{M}$ tiloja olisivat kaikki osajoukot $S \subseteq \{q_0, q_1, q_2, q_3\}$, mutta useimpia näistä ei ole mahdollista saavuttaa alkutilasta millään syötejonolla, eikä yksinkertaisuuden vuoksi myöskään tapana kirjoittaa näkyviin. Automaatin $\widehat{M}$ saavutettavat tilat ja niiden väliset siirtymät on esitetty kuvassa 2.10. Minimoimalla automaatti luvun 2.4 menetelmällä voidaan todeta, että sen kolme lopputilaa voidaan yhdistää yhdeksi. Nimeämällä vielä minimiautomaatin tilat uudelleen saadaan kuvassa 2.11 esitetty deterministinen automaatti.

∗ *Todistus jatkuu.* Tarkastetaan, että edellä esitetyllä tavalla epädeterministisestä automaatista M muodostettu deterministinen automaatti $\widehat{M}$ todella on ekvivalentti M:n kanssa,

so. että $L(\widehat{M}) = L(M)$. (Jatkossa tyydytään yleensä esittämään todistuksista vain peruskonstruktio ja jättämään tämänkaltaiset oikeellisuustarkastukset lukijalle.)

Palautetaan mieliin, että määritelmien mukaan on

$$x \in L(M) \text{ joss } (q_0, x) \underset{M}{\vdash^*} (q_f, \lambda) \text{ jollakin } q_f \in F$$

ja

$$x \in L(\widehat{M}) \text{ joss } (\{q_0\}, x) \underset{\widehat{M}}{\vdash^*} (S, \lambda) \text{ ja } S \text{ sisältää jonkin } q_f \in F.$$

Siten kielten ekvivalenssi seuraa, kun todistetaan kaikilla $x \in \Sigma^*$ ja $q \in Q$ väite:

$$(q_0, x) \underset{M}{\vdash^*} (q, \lambda) \text{ joss } (\{q_0\}, x) \underset{\widehat{M}}{\vdash^*} (S, \lambda) \text{ ja } q \in S. \tag{2.4}$$

Väite voidaan todistaa induktiolla merkkijonon x pituuden suhteen:

(i) Tapaus $|x| = 0$*:* $(q_0, \lambda) \underset{M}{\vdash^*} (q, \lambda)$ joss $q = q_0$; samoin $(\{q_0\}, \lambda) \underset{\widehat{M}}{\vdash^*} (S, \lambda)$ joss $S = \{q_0\}$.

(ii) Induktioaskel: Olkoon $x = ya$; oletetaan, että väite 2.4 pätee y:lle. Tällöin:

$$\begin{aligned}
&(q_0, x) = (q_0, ya) \underset{M}{\vdash^*} (q, \lambda) \text{ joss} \\
&\exists q' \in Q \text{ s.e. } (q_0, ya) \underset{M}{\vdash^*} (q', a) \text{ ja } (q', a) \underset{M}{\vdash} (q, \lambda) \text{ joss} \\
&\exists q' \in Q \text{ s.e. } (q_0, y) \underset{M}{\vdash^*} (q', \lambda) \text{ ja } (q', a) \underset{M}{\vdash} (q, \lambda) \text{ joss } \text{ (ind.ol.)} \\
&\exists q' \in Q \text{ s.e. } (\{q_0\}, y) \underset{\widehat{M}}{\vdash^*} (S', \lambda) \text{ ja } q' \in S' \text{ ja } q \in \delta(q', a) \text{ joss} \\
&(\{q_0\}, y) \underset{\widehat{M}}{\vdash^*} (S', \lambda) \text{ ja } \exists q' \in S' \text{ s.e. } q \in \delta(q', a) \text{ joss} \\
&(\{q_0\}, y) \underset{\widehat{M}}{\vdash^*} (S', \lambda) \text{ ja } q \in \textstyle\bigcup_{q' \in S'} \delta(q', a) = \hat{\delta}(S', a) \text{ joss} \\
&(\{q_0\}, ya) \underset{\widehat{M}}{\vdash^*} (S', a) \text{ ja } q \in \hat{\delta}(S', a) = S \text{ joss} \\
&(\{q_0\}, ya) \underset{\widehat{M}}{\vdash^*} (S', a) \text{ ja } (S', a) \underset{\widehat{M}}{\vdash} (S, \lambda) \text{ ja } q \in S \text{ joss} \\
&(\{q_0\}, x) = (\{q_0\}, ya) \underset{\widehat{M}}{\vdash^*} (S, \lambda) \text{ ja } q \in S. \quad \square
\end{aligned}$$

2.6 Säännölliset lausekkeet ja kielet

Seuraavassa esitetään edeltävistä automaattimalleista päällisin puolin huomattavasti poikkeava tapa kuvata yksinkertaisia kieliä. Näitä ns. *säännöllisiä lausekkeita* (engl. regular expressions) käytetään esimerkiksi UNIX-käyttöjärjestelmän komentokielessä kuvaamaan toiminnon (grep, sed tms.) kohdistumista tietyt ominaisuudet omaaviin merkkijonoihin.

Määritellään ensin joitakin perusoperaatioita kielten yhdistelemiseen. Olkoot A ja B aakkoston Σ kieliä. Tällöin:

(i) A:n ja B:n *yhdiste* (engl. union) on kieli

$$A \cup B = \{x \in \Sigma^* \mid x \in A \text{ tai } x \in B\};$$

(ii) A:n ja B:n *tulo* (engl. product) on kieli

$$AB = \{xy \in \Sigma^* \mid x \in A,\ y \in B\};$$

(iii) A:n *potenssit* (engl. powers) A^k, $k \geq 0$, määritellään iteratiivisesti:

$$\begin{cases} A^0 &= \{\lambda\}, \\ A^k &= AA^{k-1} = \{x_1 \ldots x_k \mid x_i \in A \quad \forall i = 1, \ldots, k\} \qquad (k \geq 1); \end{cases}$$

(iv) A:n *sulkeuma* (engl. closure) on kieli

$$A^* = \bigcup_{k \geq 0} A^k = \{x_1 \ldots x_k \mid k \geq 0,\ x_i \in A \quad \forall i = 1, \ldots, k\}.$$

Esimerkiksi jos $A = \{aa, b\}$ ja $B = \{ab\}$, niin

$$\begin{aligned} A \cup B &= \{aa, b, ab\}, \\ AB &= \{aaab, bab\}, \\ BA &= \{abaa, abb\}, \\ A^2 &= \{aaaa, aab, baa, bb\}, \\ A^* &= \{\lambda, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, \ldots\}. \end{aligned}$$

Määritelmä 2.3 Aakkoston Σ *säännölliset lausekkeet* määritellään induktiivisesti seuraavilla säännöillä:

(i) $\boldsymbol{\emptyset}$ ja $\boldsymbol{\lambda}$ ovat Σ:n säännöllisiä lausekkeita;

(ii) $\boldsymbol{a}$ on Σ:n säännöllinen lauseke kaikilla $a \in \Sigma$;

(iii) jos r ja s ovat Σ:n säännöllisiä lausekkeita, niin $(r \cup s)$, (rs) ja r^* ovat Σ:n säännöllisiä lausekkeita;

(iv) muita Σ:n säännöllisiä lausekkeita ei ole.

Kukin Σ:n säännöllinen lauseke r *kuvaa* tietyn kielen $L(r)$, joka määritellään:

(i) $L(\boldsymbol{\emptyset}) = \emptyset$;

(ii) $L(\boldsymbol{\lambda}) = \{\lambda\}$;

(iii) $L(\boldsymbol{a}) = \{a\}$ kaikilla $a \in \Sigma$;

(iv) $L((r \cup s)) = L(r) \cup L(s)$;

(v) $L((rs)) = L(r)L(s)$;

(vi) $L(r^*) = (L(r))^*$.

Esimerkiksi seuraavat ovat aakkoston $\{a, b\}$ säännöllisiä lausekkeita:

$$r_1 = ((\boldsymbol{ab})\boldsymbol{b}), \quad r_2 = (\boldsymbol{ab})^*, \quad r_3 = (\boldsymbol{ab}^*), \quad r_4 = (\boldsymbol{a}(\boldsymbol{b} \cup (\boldsymbol{bb})))^*.$$

Lausekkeiden kuvaamat kielet ovat:

$$\begin{aligned}
L(r_1) &= (\{a\}\{b\})\{b\} = \{ab\}\{b\} = \{abb\};\\
L(r_2) &= \{ab\}^* = \{\lambda, ab, abab, ababab, \ldots\} = \{(ab)^i \mid i \geq 0\};\\
L(r_3) &= \{a\}(\{b\})^* = \{a, ab, abb, abbb, \ldots\} = \{ab^i \mid i \geq 0\};\\
L(r_4) &= (\{a\}\{b, bb\})^* = \{ab, abb\}^* = \{\lambda, ab, abb, abab, ababb, \ldots\}\\
&= \{x \in \{a,b\}^* \mid \text{ kutakin } a\text{-kirjainta } x\text{:ssä seuraa 1 tai 2 } b\text{-kirjainta }\}.
\end{aligned}$$

Sulkumerkkien vähentämiseksi sovitaan operaattoreiden prioriteettisäännöiksi, että sulkeumaoperaattori (*) sitoo vahvemmin kuin tulo, joka puolestaan sitoo vahvemmin kuin yhdiste. Lisäksi huomataan, että yhdiste- ja tulo-operaatiot ovat assosiatiivisia, so.

$$L(((r \cup s) \cup t)) = L((r \cup (s \cup t))), \qquad L(((rs)t)) = L((r(st))),$$

joten peräkkäisiä yhdisteitä ja tuloja ei tarvitse suluttaa.

Tapana on myös kirjoittaa lausekkeet tavanomaisilla kirjasimilla, mikäli sekaannuksen vaaraa kuvattujen kielten merkkijonoihin ei ole.

Edellä olevat esimerkkilausekkeet kirjoitettaisiin siis näiden sääntöjen mukaan yksinkertaisemmin:

$$r_1 = abb, \quad r_2 = (ab)^*, \quad r_3 = ab^*, \quad r_4 = (a(b \cup bb))^*.$$

Vielä yksi esimerkki: Pascalin etumerkittömät reaaliluvut, joita tarkasteltiin jo luvussa 2.1, voidaan kuvata lausekkeella

$$number = dd^*(.dd^* \cup \lambda)(\mathtt{E}(+ \cup - \cup \lambda)dd^* \cup \lambda)^3,$$

missä d on lyhennysmerkintä lausekkeelle

$$d = (\mathtt{0} \cup \mathtt{1} \cup \mathtt{2} \cup \mathtt{3} \cup \mathtt{4} \cup \mathtt{5} \cup \mathtt{6} \cup \mathtt{7} \cup \mathtt{8} \cup \mathtt{9}).$$

Määritelmä 2.4 Kieli on *säännöllinen* (engl. regular), jos se voidaan kuvata säännöllisellä lausekkeella.

Säännöllisten lausekkeiden sieventäminen

Annettu säännöllinen kieli voidaan yleensä kuvata säännöllisenä lausekkeena useilla eri tavoilla. Esimerkiksi säännölliset lausekkeet $a^*b^* \cup (a \cup b)^*ba(a \cup b)^*$, $(a^*b^*)^*$ ja $(a \cup b)^*$ kuvaavat kaikki samaa kieltä. Lausekkeiden sieventämisen tavoitteena on löytää annetun kielen vaihtoehtoisista kuvaustavoista jossakin mielessä yksinkertaisin: esimerkiksi edellä kolmantena esitetty lauseke lienee vaihtoehdoista selkein. Sieventämisen tavoite ei tosin aina ole hyvin määritelty, sillä eri esitystapoihin voidaan päätyä myös vain tarkastelemalla kuvattavaa kieltä hieman eri tavoin: jos tavoitteena on kuvata vaikkapa binääriaakkoston kaikki merkkijonot, jotka sisältävät vähintään yhden ykkösen, voidaan kuvauksessa nostaa esiin jonon ensimmäinen, viimeinen, tai ylipäänsä vain jokin ykkönen. Eri tarkastelukulmat johtavat tässä tapauksessa lausekkeisiin $0^*1(0 \cup 1)^*$, $(0 \cup 1)^*10^*$ ja $(0 \cup 1)^*1(0 \cup 1)^*$, joista mikään ei liene muita oleellisesti sievempi.

[3] Usein esiintyvää lausekemuotoa rr^* merkitään joskus lyhyemmin r^+:lla. Vastaavasti kielestä A johdettua kieltä AA^* merkitään A^+:lla ja sanotaan A:n *aidoksi sulkeumaksi* (engl. proper closure). Esimerkin lauseke voitaisiin siis kirjoittaa myös: $d^+(.d^+ \cup \lambda)(\mathtt{E}(+ \cup - \cup \lambda)d^+ \cup \lambda)$.

Sanotaan, että säännölliset lausekkeet r ja s ovat *ekvivalentit* ja merkitään $r = s$, jos $L(r) = L(s)$. (Korrektimpi, mutta kömpelömpi merkintä olisi $r \equiv s$.) Säännöllisten lausekkeiden ekvivalenssin testaaminen on epätriviaali ongelma, joka voidaan kuitenkin periaatteessa ratkaista mekaanisesti seuraavassa luvussa esitettävää säännöllisten lausekkeiden ja äärellisten automaattien vastaavuutta hyväksi käyttäen. Yksinkertaisissa tapauksissa ekvivalenssi on silti helpompi todeta suoraan tunnettujen ekvivalenssisääntöjen avulla tai lausekkeiden kuvaamia kieliä tarkastellen.

Seuraavassa on joitakin yksinkertaisia säännöllisten lausekkeiden ekvivalenssisääntöjä:

$$\begin{aligned}
r \cup (s \cup t) &= (r \cup s) \cup t \\
r(st) &= (rs)t \\
r \cup s &= s \cup r \\
r(s \cup t) &= rs \cup rt \\
(r \cup s)t &= rt \cup st \\
\emptyset^* &= \lambda \\
\emptyset r &= \emptyset \\
\lambda r &= r \\
r^* &= r^* r \cup \lambda \\
r^* &= (r \cup \lambda)^*.
\end{aligned}$$

Itse asiassa voidaan osoittaa, että mikä tahansa voimassa oleva säännöllisten lausekkeiden ekvivalenssi voidaan johtaa näistä laskulaeista, kun niihin vielä lisätään päättelysääntö: jos $r = rs \cup t$, niin $r = ts^*$, edellyttäen että $\lambda \notin L(s)$.

Kahden lausekkeen ekvivalenssin toteamiseksi kannattaa usein päätellä erikseen kummankin kuvaaman kielen sisältyminen toiseen. Merkitään, jälleen hieman epätarkasti, että $r \subseteq s$, jos $L(r) \subseteq L(s)$; tällöin on voimassa $r = s$, jos ja vain jos $r \subseteq s$ ja $s \subseteq r$.

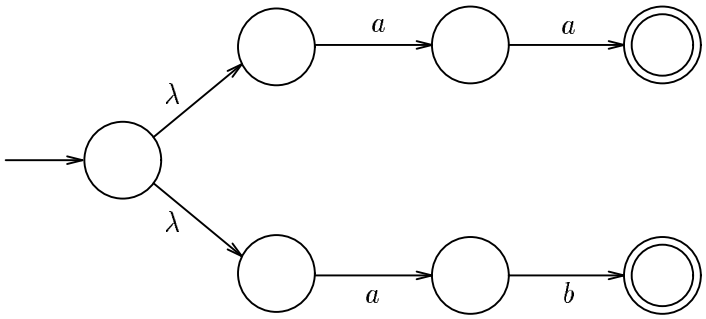
Esimerkiksi kappaleen alussa mainittujen ekvivalenssien toteamiseksi on ensinnäkin selvää, että $(a^*b^*)^* \subseteq (a \cup b)^*$ ja $a^*b^* \cup (a \cup b)^*ba(a \cup b)^* \subseteq (a \cup b)^*$, koska lauseke $(a \cup b)^*$ kuvaa *kaikkia* aakkoston $\{a, b\}$ merkkijonoja. Toisaalta, koska selvästi on $(a \cup b) \subseteq a^*b^*$, on myös $(a \cup b)^* \subseteq (a^*b^*)^*$. Vain hieman mutkikkaampi tarkastelu tarvitaan sen toteamiseen, että jos aakkoston $\{a, b\}$ mielivaltainen merkkijono ei ole muotoa a^*b^*, niin se sisältää osajonon ba, ja on siis muotoa $(a \cup b)^*ba(a \cup b)^*$; siten on myös $(a \cup b)^* \subseteq a^*b^* \cup (a \cup b)^*ba(a \cup b)^*$.

2.7 Äärelliset automaatit ja säännölliset kielet

Tässä kappaleessa todistetaan säännöllisten kielten teorian perustulos: *kieli on säännöllinen, jos ja vain jos se voidaan tunnistaa äärellisellä automaatilla.* Väitteen molempien suuntien todistukset sisältävät tärkeitä konstruktioita, joten ne esitetään erikseen.

Lause 2.3 *Jokainen säännöllinen kieli voidaan tunnistaa äärellisellä automaatilla.*

Todistus. Väitteen todistusta varten joudutaan tarkastelemaan uutta äärellisten automaattien mallin laajennusta, nimittäin epädeterminististisiä äärellisiä automaatteja, joissa sallitaan λ-*siirtymiä* — siirtymiä, joissa automaatti ei lue yhtään syötemerkkiä. Esimerkiksi kuvassa 2.12 on esitetty λ-automaatti, joka tunnistaa kielen $\{aa, ab\}$.



Kuva 2.12: Kielen $\{aa, ab\}$ tunnistava λ-automaatti.

Formaalisti λ-automaatti voidaan määritellä viisikkona $M = (Q, \Sigma, \delta, q_0, F)$, missä siirtymäfunktio δ on kuvaus

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q).$$

Automaattimalliin liittyvät muut määritelmät ovat samat kuin tavallisilla epädeterministisillä äärellisillä automaateilla, paitsi suoran tilannejohdon määritelmä: λ-automaattien tapauksessa relaatio

$$(q, w) \underset{M}{\vdash} (q', w')$$

on voimassa, jos on

(i) $w = aw'$ $(a \in \Sigma)$ ja $q' \in \delta(q, a)$; tai

(ii) $w = w'$ ja $q' \in \delta(q, \lambda)$.

Lemma 2.4 *Olkoon $A = L(M)$ jollakin λ-automaatilla M. Tällöin on olemassa myös λ-siirtymätön epädeterministinen automaatti $\widehat{M}$, jolla $A = L(\widehat{M})$.*

Todistus. Olkoon $M = (Q, \Sigma, \delta, q_0, F)$ jokin λ-automaatti. Automaatti $\widehat{M}$ toimii muuten aivan samoin kuin M, mutta se simuloi kunkin askelensa yhteydessä myös kaikki M:n mahdolliset λ-siirtymät.

Formaalisti määritellään:

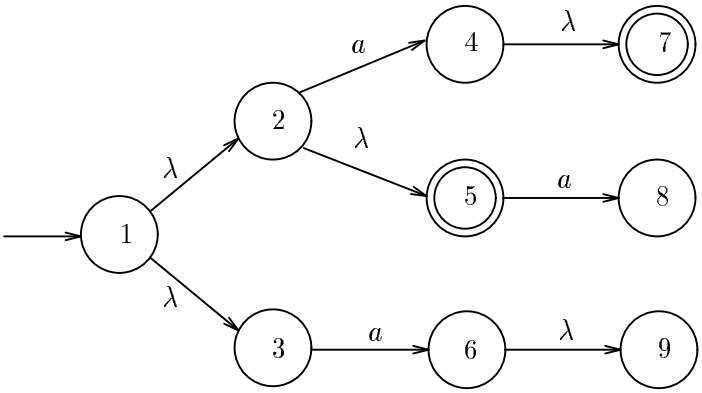
$$\widehat{M} = (Q, \Sigma, \hat{\delta}, q_0, \widehat{F}),$$

missä

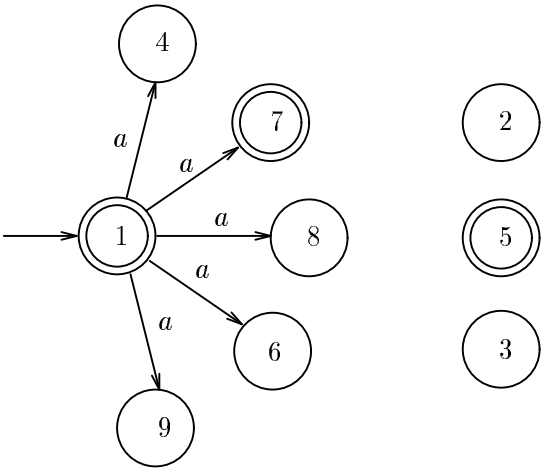
$$\begin{aligned} \hat{\delta}(q, a) &= \{q' \in Q \mid (q, a) \underset{M}{\vdash^*} (q', \lambda)\}; \\ \widehat{F} &= \begin{cases} F \cup \{q_0\}, & \text{jos } (q_0, \lambda) \underset{M}{\vdash^*} (q_f, \lambda) \text{ jollakin } q_f \in F; \\ F, & \text{muuten.} \end{cases} \end{aligned}$$

Esimerkkinä konstruktiosta on kuvassa 2.13 esitetty eräs λ-automaatti M, ja kuvassa 2.14 vastaava epädeterminstinen automaatti $\widehat{M}$, josta λ-siirtymät on poistettu. □ (Lemma 2.4)

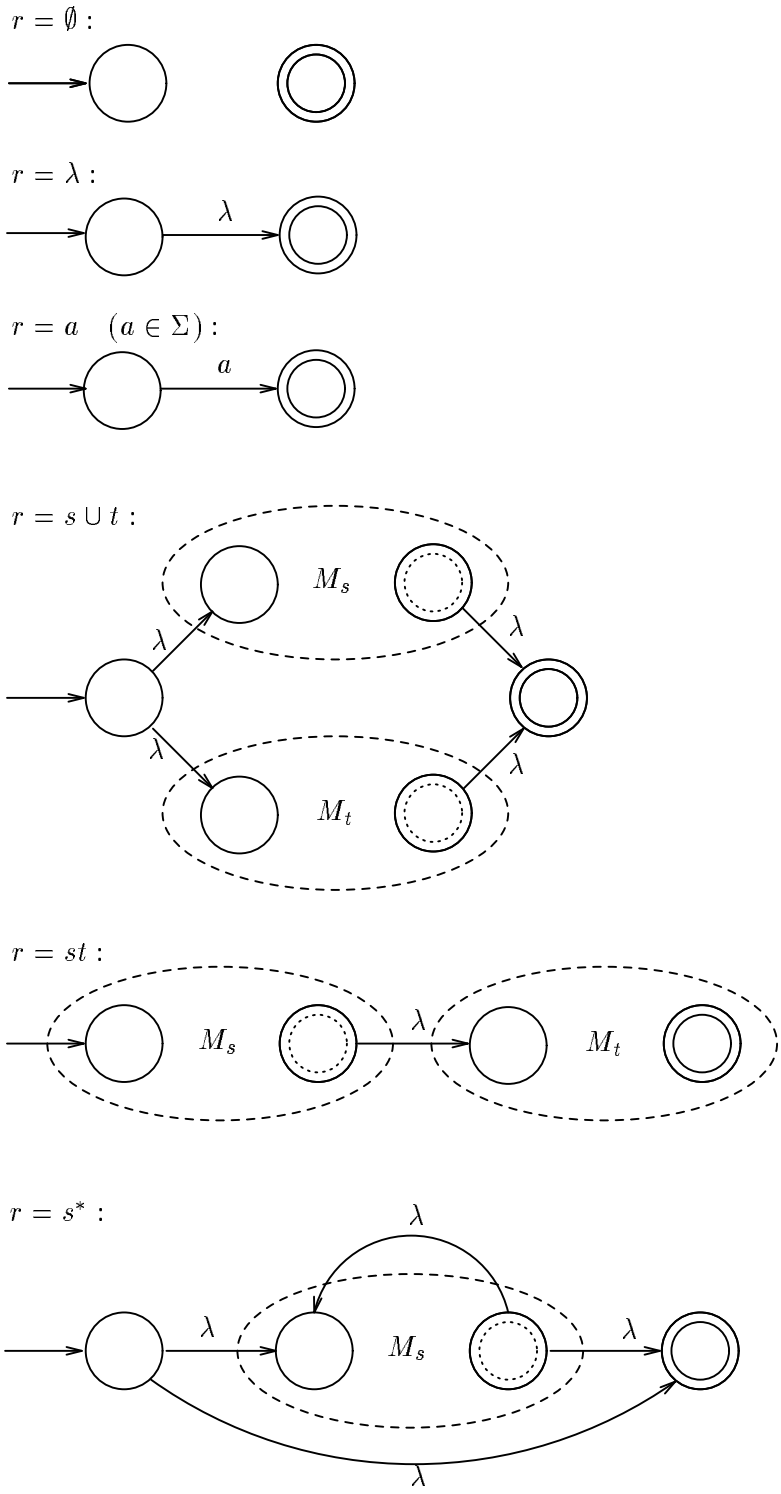
Palataan sitten varsinaisen lauseen todistukseen. Kuvassa 2.15 on esitetty induktiivinen konstruktio, jonka avulla voidaan mielivaltaisen säännöllisen lausekkeen r rakennetta seuraten muodostaa λ-automaatti M_r, jolla $L(M_r) = L(r)$. Tästä automaatista voidaan sitten



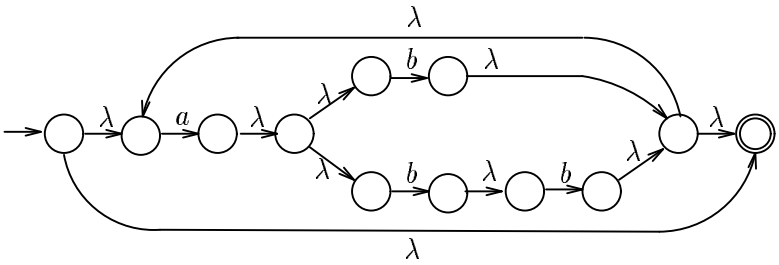
Kuva 2.13: λ-automaatti M.



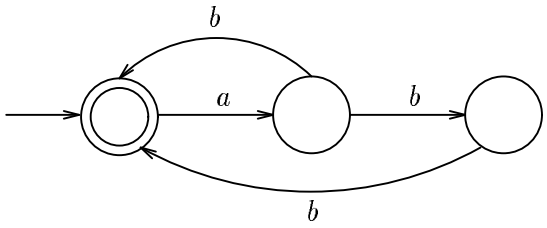
Kuva 2.14: Epädeterministinen automaatti $\widehat{M}$.



Kuva 2.15: Lauseketta r vastaavan λ-automaatin M_r muodostaminen.



Kuva 2.16: Lauseketta $r = (a(b \cup bb))^*$ vastaava λ-automaatti.



Kuva 2.17: Lauseketta $r = (a(b \cup bb))^*$ vastaava λ-siirtymätön automaatti.

poistaa λ-siirtymät lemmassa 2.4 esitetyllä tavalla, ja tarvittaessa voidaan näin syntyvä epädeterministinen automaatti edelleen determinisoida lauseen 2.2 konstruktiolla. Kuvan 2.15 konstruktiosta on syytä huomata, että siinä muodostettavissa λ-automaateissa on aina yksikäsitteiset alku- ja lopputila.

Esimerkiksi lausekkeesta $r = (a(b \cup bb))^*$ näiden sääntöjen mukaan muodostettu λ-automaatti on esitetty kuvassa 2.16. Automaatti on selvästi hyvin redundantti; λ-siirtymien poistaminen, automaatin determinisointi ja minimointi jätetään lukijalle[4]. □

Kuvan 2.15 säännöissä on tavoiteltu ennen muuta mahdollisimman yksinkertaista ja mekaanista todistuskonstruktiota. Käsin automaatteja muodostettaessa ei sääntöjä useinkaan kannata seurata aivan tunnollisesti; esimerkiksi lausekkeesta $r = (a(b \cup bb))^*$ on helppo muodostaa suoraan kuvan 2.17 yksinkertainen λ-siirtymätön automaatti.

Lause 2.5 *Jokainen äärellisellä automaatilla tunnistettava kieli on säännöllinen.*

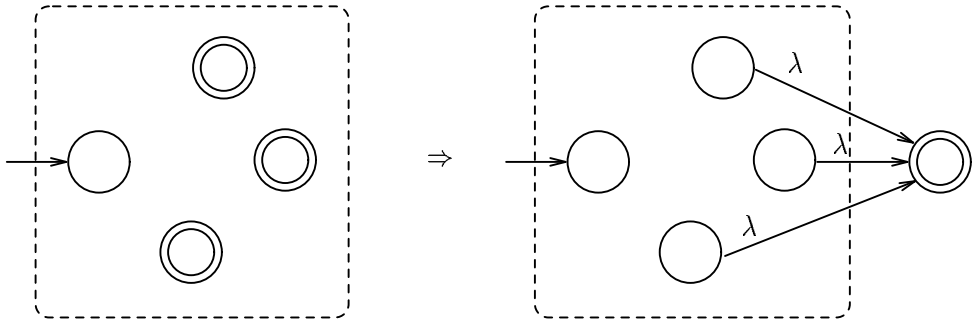
Todistus. Määritellään vielä yksi äärellisten automaattien laajennus: *lausekeautomaatissa* voidaan siirtymien ehtoina käyttää mielivaltaisia säännöllisiä lausekkeita.

Vaikka tämä yleistys ei ole käsitteellisesti juuri sen vaikeampi kuin λ-automaatitkaan, sen täsmällinen formulointi on hieman hankalampaa. Pääpiirteissään formulointi kuitenkin sujuu vanhaan tapaan.

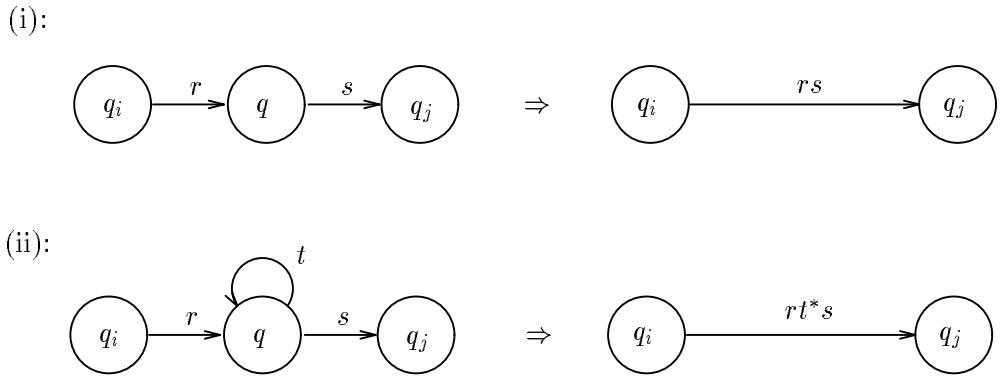
Merkitään aakkoston Σ säännöllisten lausekkeiden joukkoa RE_Σ:lla. Lausekeautomaatti voidaan tällöin määritellä viisikkona $M = (Q, \Sigma, \delta, q_0, F)$, missä siirtymäfunktio δ on äärellinen kuvaus

$$\delta : Q \times \mathrm{RE}_\Sigma \to \mathcal{P}(Q)$$

[4]Tässä kohden voi olla syytä huomauttaa, että luvussa 2.4 esitetty minimointialgoritmi koskee vain deterministisiä äärellisiä automaatteja, ja että annetulla säännöllisellä kielellä voi olla useita erilaisia epädeterministisiä minimiautomaatteja.



Kuva 2.18: Lausekeautomaatin lopputilojen yhdistäminen.



Kuva 2.19: Tilan poistaminen lausekeautomaatista.

(so. $\delta(q,r) \neq \emptyset$ vain äärellisen monella parilla $(q,r) \in Q \times \mathrm{RE}_\Sigma$). Yhden askelen tilannejohto määritellään nyt:

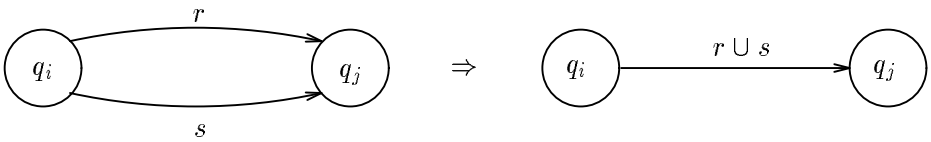
$$(q,w) \underset{M}{\vdash} (q',w')$$

jos on $q' \in \delta(q,r)$ jollakin sellaisella $r \in \mathrm{RE}_\Sigma$, että $w = zw'$, $z \in L(r)$. Muut määritelmät ovat samat kuin aiemmin.

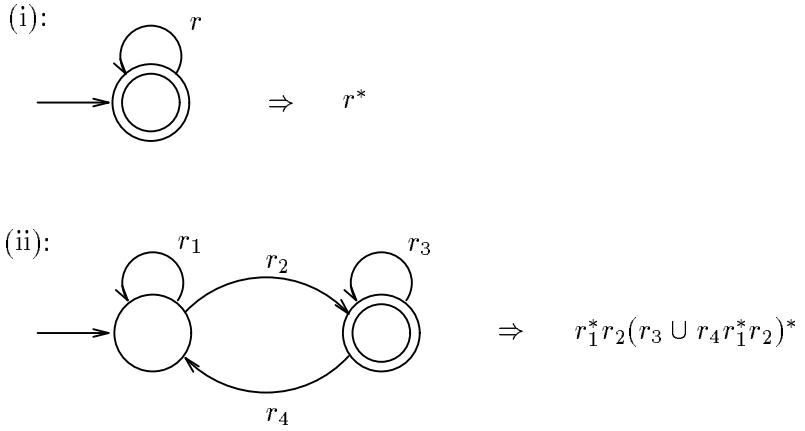
Todistetaan vaadittua tulosta näennäisesti vahvempi väite: *jokainen lausekeautomaatilla tunnistettava kieli on säännöllinen.*

Olkoon M jokin lausekeautomaatti. Säännöllinen lauseke, joka kuvaa M:n tunnistaman kielen, voidaan muodostaa seuraavasti:

1. Tiivistetään M seuraavilla, tunnistettavan kielen säilyttävillä automaattimuunnoksilla lausekeautomaatiksi, jossa on enintään kaksi tilaa:



Kuva 2.20: Rinnakkaisten siirtymien yhdistäminen lausekeautomaatissa.



Kuva 2.21: Säännöllisen lausekkeen muodostaminen redusoidusta lausekeautomaatista.

(a) Jos M:ssä on useampia kuin yksi lopputila, ne korvataan yhdellä kuvan 2.18 osoittamalla tavalla.

(b) Niin kauan kuin M:ssä on muita tiloja kuin alku- ja lopputila, ne poistetaan yksi kerrallaan seuraavasti. Olkoon q jokin M:n tila, joka ei ole alku- eikä lopputila; tarkastellaan kaikkia "reittejä", jotka M:ssä kulkevat q:n kautta. Olkoot q_i ja q_j q:n edeltäjä- ja seuraajatila jollakin tällaisella reitillä (mahdollisesti on $q_i = q_j$). Poistetaan q reitiltä $q_i \rightarrow q_j$ tekemällä kuvan 2.19 (i) esittämä automaattimuunnos, jos tilasta q ei ole siirtymää itseensä, ja kuvan 2.19 (ii) esittämä automaattimuunnos, jos tilasta q on siirtymä itseensä. Rinnakkaiset siirtymät voidaan tarvittaessa yhdistää kuvan 2.20 esittämällä tavalla.

2. Tiivistyksen päättyessä automaatissa on jäljellä vain alku- ja lopputila, jotka voivat olla sama. Automaatin tunnistaman kielen kuvaava säännöllinen lauseke saadaan kuvan 2.21 esittämällä tavalla: vaihtoehdon (i) mukaan, jos alku- ja lopputila ovat sama, ja vaihtoehdon (ii) mukaan, jos ne ovat eri tiloja. □

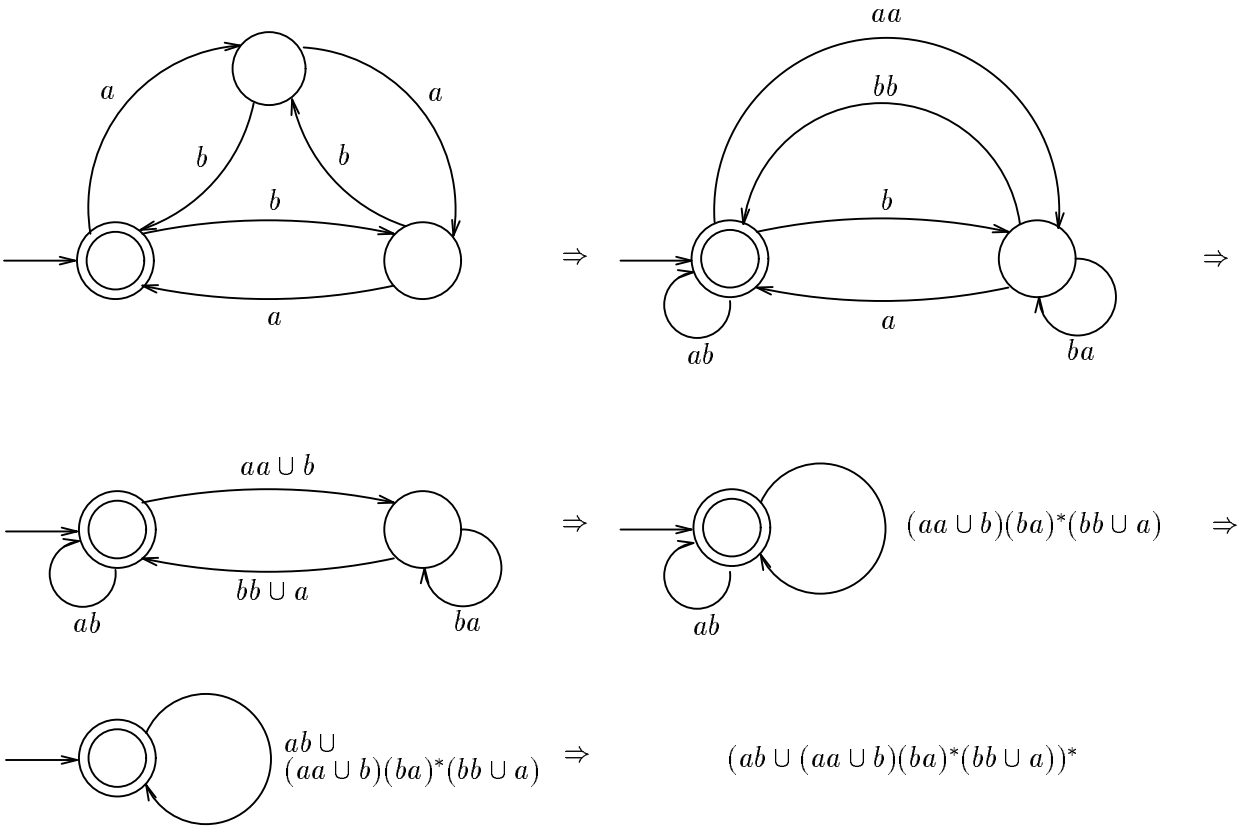
Kuvassa 2.22 on esimerkki konstruktion soveltamisesta.

2.8 Säännöllisten kielten rajoituksista

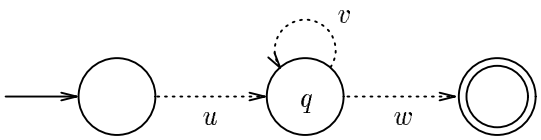
Koska minkä tahansa aakkoston formaaleja kieliä on ylinumeroituva ja säännöllisiä lausekkeita vain numeroituva määrä (lauseet 1.1 ja 1.2), eivät kaikki kielet mitenkään voi olla säännöllisiä. Mutta voidaanko löytää konkreettinen, *mielenkiintoinen* esimerkki kielestä, joka ei olisi säännöllinen?

Valitettavasti tällaisia esimerkkejä on helppo löytää: säännöllisten kielten luokka riittää käytännössä vain hyvin rajoitettuihin tarpeisiin. Esimerkiksi ohjelmointikielten perusalkiot (lukuesitykset, muuttujan- ja käskynnimet) ovat tyypillisesti rakenteeltaan säännöllisiä, mutta mutkikkaammat konstruktiot (aritmeettiset lausekkeet, kootut lauseet) eivät.

Säännöllisten kielten perusrajoitus seuraa siitä, että äärellisillä automaateilla on vain rajallinen "muisti". Siten ne eivät (likimäärin sanoen) pysty ratkaisemaan ongelmia, joissa



Kuva 2.22: Säännöllisen lausekkeen muodostaminen äärellisestä automaatista.



Kuva 2.23: Merkkijonon $x = uvw \in A$ pumppaus.

vaaditaan mielivaltaisen suurten lukujen tarkkaa muistamista. Äärellisillä automaateilla ei esimerkiksi pystytä tunnistamaan tasapainoisten sulkujonojen muodostamaa kieltä

$$L_{\text{match}} = \{(^k)^k \mid k \geq 0\},$$

eikä siten myöskään yleisemmin mielivaltaisia hyvinmuodostettuja aritmeettisia lausekkeita.

Seuraava apulause, ns. "pumppauslemma" formalisoi tämän rajallisen muistin idean käyttökelpoiseen muotoon. Lemman nimi tulee siitä, että se osoittaa mitä tahansa annetun säännöllisen kielen riittävän pitkää merkkijonoa voitavan "pumpata" keskeltä, ilman että kielen tunnistava äärellinen automaatti huomaa muutosta.

Lemma 2.6 (Pumppauslemma) *Olkoon A säännöllinen kieli. Tällöin on olemassa sellainen $n \geq 1$, että mikä tahansa $x \in A$, $|x| \geq n$, voidaan jakaa osiin $x = uvw$ siten, että $|uv| \leq n$, $|v| \geq 1$, ja $uv^iw \in A$ kaikilla $i = 0, 1, 2, \ldots$*

Todistus. Olkoon M jokin A:n tunnistava deterministinen äärellinen automaatti, ja olkoon n M:n tilojen määrä. Tarkastellaan automaatin läpikäymiä tiloja sen tunnistaessa merkkijonoa $x \in A$, $|x| \geq n$. Koska M jokaisella x:n merkillä siirtyy tilasta toiseen, sen täytyy kulkea jonkin tilan kautta (ainakin) kaksi kertaa — itse asiassa jo x:n n:ää ensimmäistä merkkiä käsitellessään. Olkoon q ensimmäinen tila, jonka automaatti toistaa x:ää käsitellessään.

Olkoon u M:n käsittelemä x:n alkuosa sen tullessa ensimmäisen kerran tilaan q, v se osa x:stä jonka M käsittelee ennen ensimmäistä paluutaan q:hun, ja w loput x:stä (kuva 2.23). Tällöin on $|uv| \leq n$, $|v| \geq 1$, ja $uv^iw \in A$ kaikilla $i = 0, 1, 2, \ldots$ □

Esimerkkinä pumppauslemman soveltamisesta tarkastellaan sulkulausekekieltä L_{match}. Selvyyden vuoksi merkitään '(' $= a$, ')' $= b$; kieli on siis

$$L = L_{\text{match}} = \{a^k b^k \mid k \geq 0\}.$$

Oletetaan, että L olisi säännöllinen. Tällöin pitäisi lemman 2.6 mukaan olla jokin $n \geq 1$, jota pitempiä L:n merkkijonoja voidaan pumpata. Valitaan $x = a^n b^n$, jolloin $|x| = 2n > n$. Lemman mukaan x voidaan jakaa pumpattavaksi osiin $x = uvw$, $|uv| \leq n$, $|v| \geq 1$; siis on oltava

$$u = a^i,\ v = a^j,\ w = a^{n-(i+j)}b^n, \quad \text{missä } i \leq n-1,\ j \geq 1.$$

Mutta esimerkiksi "0-kertaisesti" pumpattu merkkijono $uv^0w = a^i a^{n-(i+j)} b^n = a^{n-j} b^n$ ei kuulu kieleen L. Siten L ei voi olla säännöllinen.

Luku 3

Kontekstittomat kieliopit ja kielet

3.1 Kieliopit ja merkkijonojen tuottaminen

Kuten edellisen luvun lopussa todettiin, tasapainoisten sulkumerkkijonojen muodostama kieli

$$L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$$

ei ole säännöllinen, so. sitä ei voida kuvata millään säännöllisellä lausekkeella. Toisaalta kieli ei sinänsä ole kovin mutkikas: se voidaan kuvata yksinkertaisella rekursiivisella määritelmällä, jos otetaan käyttöön muuttuja S, jonka arvona on "mielivaltainen tasapainoinen sulkumerkkijono":

S on tasapainoinen sulkumerkkijono, jos

(i) $S = \lambda$ tai

(ii) S on muotoa (S'), missä S' on tasapainoinen sulkumerkkijono.

Sama asia voidaan ilmaista sanomalla, että seuraavat merkkijonojen *muunnossäännöt tuottavat* täsmälleen kielen L_{match} merkkijonot symbolista S:

(i) $S \to \lambda$,

(ii) $S \to (S)$.

Esimerkiksi merkkijono ((())) voidaan tuottaa muunnosjonolla:

$$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow (((\lambda))) = ((())).$$

Tässä on kolmessa ensimmäisessä muunnoksessa sovellettu sääntöä (ii) ja neljännessä sääntöä (i).

Tällaista muunnossysteemiä, jossa kuvattavat merkkijonot tuotetaan korvaamalla erityisiä muuttuja- t. *välikesymboleita* annettujen sääntöjen mukaan yksi kerrallaan, symbolia ympäröivän merkkijonon rakenteesta riippumatta, kutsutaan *kontekstittomaksi kieliopiksi* (engl. context-free grammar).

Tarkastellaan toisena esimerkkinä kielioppia Pascal-tyyppisen ohjelmointikielen aritmeettisten lausekkeiden rakenteen kuvailuun. Kielioppia on yksinkertaistettu ottamalla mukaan

vain yhteen- ja kertolaskuoperaatiot ja merkitsemällä mielivaltaista alkeisoperandia (lukuvakiota, muuttujaa tms.) pelkällä a:lla. Välikesymboleita on kolme: E ("expression"), T ("term") ja F ("factor"); näistä E on *lähtösymboli*, josta lausekkeen tuottaminen aloitetaan. Muunnossäännöt ovat seuraavat:

$$\begin{array}{lcl} E & \to & T \mid E+T \\ T & \to & F \mid T*F \\ F & \to & a \mid (E). \end{array}$$

(Sääntöjen esityksessä on tässä käytetty lyhennysmerkintää

$$A \to \omega_1 \mid \omega_2 \mid \ldots \mid \omega_k$$

kuvaamaan joukkoa samaan välikesymboliin A liittyviä vaihtoehtoisia sääntöjä

$$A \to \omega_1,\ A \to \omega_2,\ \ldots A \to \omega_k.)$$

Esimerkiksi lauseke $(a+a)*a$ voidaan näitä sääntöjä käyttäen tuottaa seuraavasti (kussakin muunnosaskelessa korvattava välike on alleviivattu):

$$\begin{array}{lllll} \underline{E} & \Rightarrow & \underline{T} & \Rightarrow & \underline{T}*F \\ & \Rightarrow & (\underline{E})*F & \Rightarrow & (\underline{E}+T)*F \\ & \Rightarrow & (\underline{F}+T)*F & \Rightarrow & (a+\underline{T})*F \\ & \Rightarrow & (a+a)*\underline{F} & \Rightarrow & (a+a)*a. \end{array} \quad \begin{array}{ll} \Rightarrow & \underline{F}*F \\ \Rightarrow & (\underline{T}+T)*F \\ \Rightarrow & (a+\underline{F})*F \end{array}$$

Määritelmä 3.1 *Kontekstiton kielioppi* (engl. context-free grammar) on nelikko

$$G = (V, \Sigma, P, S),$$

missä

- V on kieliopin aakkosto;
- $\Sigma \subseteq V$ on kieliopin *päätemerkkien* (engl. terminal symbols) joukko; sen komplementti $N = V - \Sigma$ on kieliopin *välikemerkkien* t. *-symbolien* (engl. nonterminal symbols) joukko;
- $P \subseteq N \times V^*$ on kieliopin *sääntöjen* t. *produktioiden* (engl. rules, productions) joukko;
- $S \in N$ on kieliopin *lähtösymboli* (engl. start symbol).

Produktiota $(A, \omega) \in P$ merkitään tavallisesti $A \to \omega$.

Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa suoraan* (engl. derives directly) merkkijonon $\gamma' \in V^*$ kieliopissa G, merkitään

$$\gamma \underset{G}{\Rightarrow} \gamma'$$

jos voidaan kirjoittaa $\gamma = \alpha A \beta$, $\gamma' = \alpha\omega\beta$ ($\alpha, \beta, \omega \in V^*$, $A \in N$), ja kieliopissa G on produktio $A \to \omega$. Jos kielioppi G on yhteydestä selvä, relaatiota voidaan merkitä yksinkertaisesti $\gamma \Rightarrow \gamma'$.

Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa* (engl. derives) merkkijonon $\gamma' \in V^*$ kieliopissa G, merkitään

$$\gamma \overset{*}{\underset{G}{\Rightarrow}} \gamma'$$

jos on olemassa jono V:n merkkijonoja $\gamma_0, \gamma_1, \ldots, \gamma_n$ $(n \geq 0)$, siten että

$$\gamma = \gamma_0 \underset{G}{\Rightarrow} \gamma_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \gamma_n = \gamma'.$$

Erikoistapauksena $n = 0$ saadaan $\gamma \underset{G}{\Rightarrow}^* \gamma$ millä tahansa $\gamma \in V^*$. Jälleen, jos kielioppi G on yhteydestä selvä, merkitään yksinkertaisesti $\gamma \Rightarrow^* \gamma'$.

Merkkijono $\gamma \in V^*$ on kieliopin G *lausejohdos* (engl. sentential form), jos on $S \underset{G}{\Rightarrow}^* \gamma$. Pelkästään päätemerkeistä koostuva G:n lausejohdos $x \in \Sigma^*$ on G:n *lause* (engl. sentence).

Kieliopin G *tuottama* t. *kuvaama kieli* (engl. language generated by G) koostuu G:n lauseista; määritellään siis:

$$L(G) = \{x \in \Sigma^* \mid S \underset{G}{\Rightarrow}^* x\}.$$

Formaali kieli $L \subseteq \Sigma^*$ on *kontekstiton* (engl. context-free), jos se voidaan tuottaa jollakin kontekstittomalla kieliopilla. Esimerkiksi tasapainoisten sulkujonojen muodostaman kielen $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ tuottaa kielioppi

$$G_{\text{match}} = (\{S, (,)\}, \{(,)\}, \{S \to \lambda, S \to (S)\}, S),$$

ja yksinkertaisten aritmeettisten lausekkeiden muodostaman kielen L_{expr} tuottaa kielioppi

$$G_{\text{expr}} = (V, \Sigma, P, E),$$

missä

$$\begin{aligned} V &= \{E, T, F, a, +, *, (,)\}, \\ \Sigma &= \{a, +, *, (,)\}, \\ P &= \{E \to T,\ E \to E + T,\ T \to F,\ T \to T * F,\ F \to a,\ F \to (E)\}. \end{aligned}$$

Toinen kielioppi kielen L_{expr} tuottamiseen on

$$G'_{\text{expr}} = (V, \Sigma, P, E),$$

missä

$$\begin{aligned} V &= \{E, a, +, *, (,)\}, \\ \Sigma &= \{a, +, *, (,)\}, \\ P &= \{E \to E + E,\ E \to E * E,\ E \to a,\ E \to (E)\}. \end{aligned}$$

Liite: Vakiintuneita merkintätapoja

Kielioppeihin liittyville käsitteille ovat seuraavat merkintäkäytännöt vakiintuneet:

- Välikesymboleita: $A, B, C, \ldots, S, T$.
- Päätemerkkejä: kirjaimet $a, b, c, \ldots, s, t$; numerot $0, 1, \ldots, 9$; erikoismerkit; lihavoidut tai alleviivatut varatut sanat (**if**, **for**, **end**, ...).
- Mielivaltaisia merkkejä (kun välikkeitä ja päätteitä ei erotella): X, Y, Z.

- Päätemerkkijonoja: u, v, w, x, y, z.
- Sekamerkkijonoja: $\alpha, \beta, \gamma, \ldots, \omega$.
- Produktiot, joilla on yhteinen vasen puoli A, voidaan kirjoittaa yhteen: joukon

$$A \to \omega_1,\ A \to \omega_2,\ \ldots A \to \omega_k$$

sijaan kirjoitetaan

$$A \to \omega_1 \mid \omega_2 \mid \ldots \mid \omega_k.$$

- Kielioppi esitetään usein pelkkänä sääntöjoukkona:

$$\begin{array}{lcl} A_1 & \to & \omega_{11} \mid \ldots \mid \omega_{1k_1} \\ A_2 & \to & \omega_{21} \mid \ldots \mid \omega_{2k_2} \\ \vdots & & \\ A_m & \to & \omega_{m1} \mid \ldots \mid \omega_{mk_m}. \end{array}$$

Tällöin päätellään välikesymbolit edellisten merkintäsopimusten mukaan tai siitä, että ne esiintyvät sääntöjen vasempina puolina; muut esiintyvät merkit ovat päätemerkkejä. *Lähtösymboli* on tällöin *ensimmäisen säännön vasempana puolena* esiintyvä välike; tässä siis A_1.

3.2 Säännölliset kielet ja kontekstittomat kieliopit

Edellä on jo todettu, että kontekstittomilla kieliopeilla voidaan kuvata joitakin ei-säännöllisiä kieliä (esimerkiksi kielet L_{match} ja L_{expr}). Seuraavassa nähdään, että myös kaikki säännölliset kielet voidaan kuvata kontekstittomilla kieliopeilla. Kontekstittomat kielet ovat siten säännollisten kielten aito yliluokka.

Kontekstiton kielioppi on *oikealle lineaarinen* (engl. right linear), jos sen kaikki produktiot ovat muotoa $A \to \lambda$ tai $A \to aB$, ja *vasemmalle lineaarinen* (engl. left linear), jos sen kaikki produktiot ovat muotoa $A \to \lambda$ tai $A \to Ba$[1]. Osoittautuu, että sekä vasemmalle että oikealle lineaarisilla kieliopeilla voidaan tuottaa täsmälleen säännölliset kielet, minkä takia näitä kielioppeja nimitetään myös yhteisesti *säännöllisiksi*. Todistetaan tässä väite vain oikealle lineaarisille kieliopeille; vasemmalle lineaarisia kielioppeja koskeva todistus sujuu hyvin samaan tapaan.

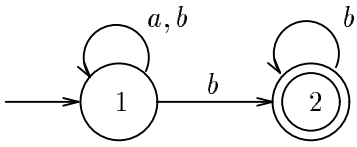
Lause 3.1 *Jokainen säännöllinen kieli voidaan tuottaa oikealle lineaarisella kieliopilla.*

Todistus. Olkoon L aakkoston Σ säännöllinen kieli, ja olkoon $M = (Q, \Sigma, \delta, q_0, F)$ sen tunnistava (deterministinen tai epädeterministinen) äärellinen automaatti. Seuraavalla konstruktiolla voidaan muodostaa kielioppi G_M, jolla on $L(G_M) = L(M) = L$.

Kieliopin G_M pääteaakkosto on sama kuin M:n syöteaakkosto Σ, ja sen välikeaakkostoon otetaan yksi välike A_q kutakin M:n tilaa q kohden. Kieliopin lähtösymboli on A_{q_0}, ja sen produktiot muodostetaan M:n siirtymiä jäljitellen seuraavaan tapaan:

(i) kutakin M:n lopputilaa $q \in F$ kohden kielioppiin otetaan produktio $A_q \to \lambda$;

[1] Usein sallitaan oikealle ja vasemmalle lineaaaristen kielioppien määritelmissä myös muotoa $A \to a$ olevat produktiot. On helppo todeta, että tämä laajennus ei muuta kielioppien kuvausvoimaa.



Kuva 3.1: Yksinkertainen äärellinen automaatti.

(ii) kutakin M:n siirtymää $q \xrightarrow{a} q'$ (so. $q' \in \delta(q,a)$) kohden kielioppiin otetaan produktio $A_q \to aA_{q'}$.

Konstruktion oikeellisuuden tarkastamiseksi merkitään välikkeestä A_q tuotettavien päätejonojen joukkoa

$$L(A_q) = \{x \in \Sigma^* \mid A_q \underset{G_M}{\Rightarrow}{}^* x\}.$$

Induktiolla merkkijonon x pituuden suhteen voidaan osoittaa, että kaikilla q on

$$x \in L(A_q) \text{ joss } (q,x) \underset{M}{\vdash}{}^* (q_f, \lambda) \text{ jollakin } q_f \in F.$$

Erityisesti on siis

$$\begin{aligned} L(G_M) = L(A_{q_0}) &= \{x \in \Sigma^* \mid (q_0,x) \underset{M}{\vdash}{}^* (q_f,\lambda) \text{ jollakin } q_f \in F\} \\ &= L(M) = L. \quad \square \end{aligned}$$

Esimerkiksi kuvan 3.1 automaattia vastaava kielioppi on:

$$\begin{aligned} A_1 &\to aA_1 \mid bA_1 \mid bA_2 \\ A_2 &\to \lambda \mid bA_2. \end{aligned}$$

Lause 3.2 *Jokainen oikealle lineaarisella kieliopilla tuotettava kieli on säännöllinen.*

Todistus. Olkoon $G = (V, \Sigma, P, S)$ oikealle lineaarinen kielioppi. Muodostetaan kielen $L(G)$ tunnistava epädeterministinen äärellinen automaatti $M_G = (Q, \Sigma, \delta, q_S, F)$ seuraavasti:

- Automaatissa M_G on yksi tila kutakin G:n välikettä kohden:

$$Q = \{q_A \mid A \in V - \Sigma\}.$$

- M_G:n alkutila on G:n lähtösymbolia S vastaava tila q_S.
- M_G:n syöteaakkosto on sama kuin G:n pääteaakkosto Σ.
- M_G:n siirtymäfunktio δ jäljittelee G:n produktioita siten, että kutakin produktiota $A \to aB$ kohden automaatissa on siirtymä $q_A \xrightarrow{a} q_B$ (so. $q_B \in \delta(q_A, a)$).
- M_G:n lopputiloja ovat ne tilat, joita vastaaviin välikkeisiin liittyy G:ssä λ-produktio:

$$F = \{q_A \in Q \mid A \to \lambda \in P\}.$$

Konstruktion oikeellisuus voidaan jälleen tarkastaa induktiolla kieliopin G tuottamien ja automaatin M_G hyväksymien merkkijonojen pituuden suhteen. $\square$

3.3 Kontekstittomien kielioppien jäsennysongelma

Keskeinen kontekstittomiin kielioppeihin liittyvä laskennallinen ongelma on niiden *jäsennys-* t. *tunnistusongelma* (engl. parsing problem, recognition problem):

> "Annettu kontekstiton kielioppi G ja merkkijono x. Onko $x \in L(G)$?"

(Esimerkiksi: "Annettu Pascal-kielen kielioppi ja merkkijono P. Onko P syntaktisesti virheetön Pascal-ohjelma?")

Tälle ongelmalle ja sen erikoistapauksille, missä G on jotakin rajoitettua muotoa, on kehitetty useita ratkaisumenetelmiä, *jäsennysalgoritmeja*. Mikään tunnetuista menetelmistä ei kuitenkaan ole yksiselitteisesti paras, vaan eri algoritmit sopivat erilaisiin tilanteisiin.

Seuraavassa esitellään kontekstittomien kielten jäsentämisen perustana olevaa käsitteistöä, tavoitteena parin suhteellisen yksinkertaisen jäsennysmenetelmän esittely.

Jäsennysten esittäminen: johdot ja jäsennyspuut

Olkoon $\gamma \in V^*$ kieliopin $G = (V, \Sigma, P, S)$ lausejohdos, so. merkkijono, jolla $S \underset{G}{\Rightarrow}^* \gamma$. Lähtösymbolista S merkkijonoon γ johtavaa suorien johtojen jonoa

$$S = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n = \gamma$$

sanotaan γ:n *johdoksi* (engl. derivation) G:ssä. Johdon *pituus* on siihen kuuluvien suorien johtojen määrä; edellä siis n.

Lausejohdoksella on tavallisesti useita johtoja; esimerkiksi lause $a + a$ voidaan johtaa kieliopissa G_{expr} (s. 35) seuraavilla kolmella tavalla, ja muillakin:

$$\begin{array}{lccccccccccc}
\text{(i)} & E & \Rightarrow & E+T & \Rightarrow & T+T & \Rightarrow & F+T & \Rightarrow & a+T & \Rightarrow & a+F & \Rightarrow & a+a \\
\text{(ii)} & E & \Rightarrow & E+T & \Rightarrow & E+F & \Rightarrow & T+F & \Rightarrow & F+F & \Rightarrow & F+a & \Rightarrow & a+a \\
\text{(iii)} & E & \Rightarrow & E+T & \Rightarrow & E+F & \Rightarrow & E+a & \Rightarrow & T+a & \Rightarrow & F+a & \Rightarrow & a+a.
\end{array}$$

Kahdenlaiset johtotavat ovat erikoisasemassa: johto $\gamma \Rightarrow^* \gamma'$ on *vasen johto* (engl. leftmost derivation), merkitään

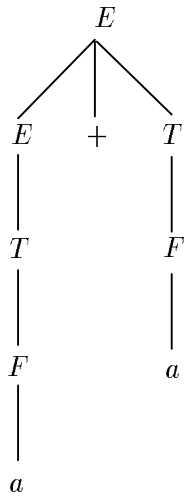
$$\gamma \underset{\text{lm}}{\Rightarrow}^* \gamma',$$

jos kussakin johtoaskelessa on produktiota sovellettu merkkijonon vasemmanpuoleisimpaan välikkeeseen: esimerkiksi edellä johto (i) on vasen johto. Vastaavasti määritellään *oikea johto* (engl. rightmost derivation). Tätä merkitään

$$\gamma \underset{\text{rm}}{\Rightarrow}^* \gamma';$$

esimerkiksi edellä (iii) on oikea johto. Suoria vasempia ja oikeita johtoaskelia merkitään $\gamma \underset{\text{lm}}{\Rightarrow} \gamma'$ ja $\gamma \underset{\text{rm}}{\Rightarrow} \gamma'$.

Suurin osa vaihtoehtoisten johtotapojen eroista muodostuu vain välikkeiden laventamisesta eri järjestyksessä: esimerkiksi edellä johdot (i) – (iii) ovat kaikki "pohjimmiltaan" samanlaisia. Esitystapa, jossa nämä epäoleelliset erot on abstrahoitu pois, on lausejohdoksen *jäsennyspuu* (syntaksipuu, johtopuu) (engl. parse tree, syntax tree, derivation tree). Jäsennyspuu kertoo ainoastaan, *miten* välikkeet on lavennettu, ei *missä järjestyksessä* lavennukset



Kuva 3.2: Lauseen $a + a$ jäsennyspuu kieliopissa G_{expr}.

on tehty. Esimerkiksi kaikkia kolmea edellä esitettyä johtoa vastaa sama, kuvassa 3.2 esitetty jäsennyspuu.

Täsmällisemmin voidaan määritellä seuraavasti: olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Kieliopin G mukainen jäsennyspuu on järjestetty puu (siis puu, jossa kunkin solmun jälkeläisten kesken on määritelty järjestys vasemmalta oikealle), jolla on seuraavat ominaisuudet:

(i) puun solmut on nimetty joukon $V \cup \{\lambda\}$ alkioilla siten, että sisäsolmujen nimet ovat välikkeitä (so. joukosta $N = V - \Sigma$) ja juurisolmun nimenä on lähtösymboli S;

(ii) jos A on puun jonkin sisäsolmun nimi, ja $X_1, \ldots, X_k$ ovat sen jälkeläisten nimet järjestyksessä, niin $A \to X_1 \ldots X_k$ on G:n produktio.

Jäsennyspuun τ *tuotos* (engl. yield) on merkkijono, joka saadaan liittämällä yhteen sen lehtisolmujen nimet esijärjestyksessä ("vasemmalta oikealle"). Esimerkiksi kuvan 3.2 puun tuotos on merkkijono "$a + a$".

Johtoa

$$S = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n = \gamma$$

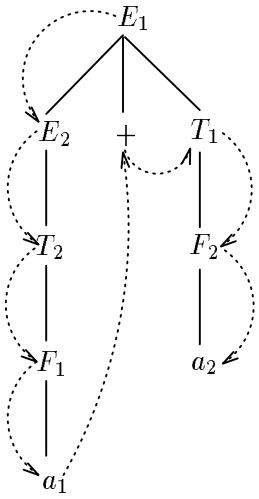
vastaava jäsennyspuu muodostetaan seuraavasti:

(i) puun juuren nimeksi tulee S; jos $n = 0$, niin puussa ei ole muita solmuja; muuten

(ii) jos ensimmäisessä johtoaskelessa on sovellettu produktiota $S \to X_1 X_2 \ldots X_k$, niin juurelle tulee k jälkeläissolmua, joiden nimet vasemmalta oikealle ovat $X_1, X_2, \ldots, X_k$;

(iii) jos seuraavassa askelessa on sovellettu produktiota $X_i \to Y_1 Y_2 \ldots Y_l$, niin juuren i:nnelle jälkeläissolmulle tulee l jälkeläistä, joiden nimet vasemmalta oikealle ovat $Y_1, Y_2, \ldots, Y_l$; ja niin edelleen.

Konstruktiosta huomataan, että jos τ on jotakin johtoa $S \Rightarrow^* \gamma$ vastaava jäsennyspuu, niin τ:n tuotos on γ.



Solmut esijärjestyksessä:

$E_1 E_2 T_2 F_1 a_1 + T_1 F_2 a_2$

Vasen johto:

$$E \underset{\text{lm}}{\Rightarrow} E + T \underset{\text{lm}}{\Rightarrow} T + T \underset{\text{lm}}{\Rightarrow} F + T \underset{\text{lm}}{\Rightarrow} a + T \underset{\text{lm}}{\Rightarrow} a + F \underset{\text{lm}}{\Rightarrow} a + a$$

Kuva 3.3: Vasemman johdon muodostaminen jäsennyspuusta.

Olkoon τ kieliopin G mukainen jäsennyspuu, jonka tuotos on päätemerkkijono x. Tällöin τ:sta saadaan vasen johto x:lle käymällä puun solmut läpi esijärjestyksessä ("ylhäältä alas, vasemmalta oikealle") ja laventamalla vastaan tulevat välikkeet järjestyksessä puun osoittamalla tavalla (ks. kuva 3.3). Oikea johto saadaan käymällä puu läpi käänteisessä esijärjestyksessä ("ylhäältä alas, oikealta vasemmalle"). Muodostamalla annetusta vasemmasta johdosta $S \underset{\text{lm}}{\Rightarrow}^* x$ ensin jäsennyspuu edellä esitetyllä tavalla, ja sitten jäsennyspuusta vasen johto, saadaan takaisin alkuperäinen johto; vastaava tulos pätee myös oikeille johdoille.

Näiden tarkastelujen perusteella voidaan esittää seuraava tärkeä lause:

Lause 3.3 *Olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Tällöin:*

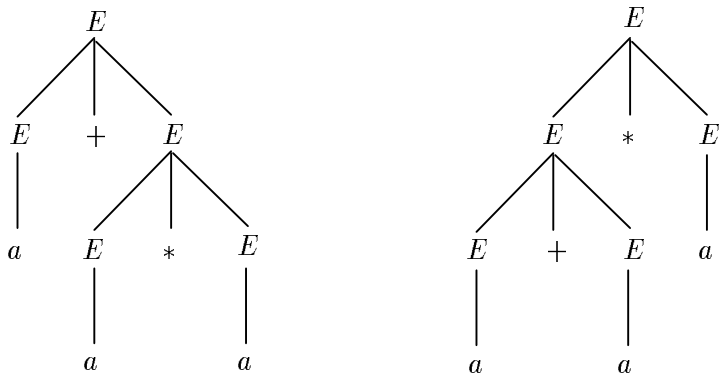
(i) jokaisella G:n lausejohdoksella γ on G:n mukainen jäsennyspuu τ, jonka tuotos on γ;

(ii) jokaista G:n mukaista jäsennyspuuta τ, jonka tuotos on päätemerkkijono x, vastaavat yksikäsitteiset vasen ja oikea johto $S \underset{\text{lm}}{\Rightarrow}^* x$ ja $S \underset{\text{rm}}{\Rightarrow}^* x$.

Seuraus 3.4 *Jokaisella G:n lauseella on vasen ja oikea johto.*

Kontekstittoman kieliopin tuottamien lauseiden jäsennyspuut, vasemmat ja oikeat johdot vastaavat siis yksikäsitteisesti toisiaan[2]. Kieliopin G jäsennysongelman ratkaisuun katsotaan usein kuuluvan pelkän päätösongelman "Onko $x \in L(G)$?" ratkaisemisen lisäksi jonkin näistä jäsennysesityksistä tuottaminen kieleen kuuluville lauseille x.

[2]Vastaavuus ei päde mielivaltaisille "keskeneräisille" lausejohdoksille: kaikilla lausejohdoksilla on jäsennyspuu, mutta ei välttämättä vasenta eikä oikeaa johtoa (esimerkiksi lausejohdokset $T + F$ ja $F + F$ sivun 38 johtoesimerkin kohdassa (ii)).



Kuva 3.4: Lauseen $a + a * a$ kaksi erilaista jäsennystä.

Kieliopin moniselitteisyys

Lauseella voi olla kieliopissa useita jäsennyksiä, joskin tämä on yleensä kieliopilta ei-toivottu ominaisuus. Esimerkiksi lauseella $a + a * a$ on kieliopissa G'_{expr} (s. 35) kuvan 3.4 esittämät kaksi jäsennystä.

Kontekstiton kielioppi G on *moniselitteinen* (engl. ambiguous), jos jollakin G:n lauseella x on kaksi erilaista G:n mukaista jäsennyspuuta. Muuten kielioppi on *yksiselitteinen* (engl. unambiguous). Kontekstiton kieli, jonka tuottavat kieliopit ovat kaikki moniselitteisiä, on *luonnostaan moniselitteinen* (engl. inherently ambiguous).

Esimerkiksi kielioppi G'_{expr} on moniselitteinen, kieliopit G_{expr} ja G_{match} yksiselitteisiä. Kieli $L_{\text{expr}} = L(G'_{\text{expr}})$ ei ole luonnostaan moniselitteinen, koska sillä on myös yksiselitteinen kielioppi G_{expr}. Luonnostaan moniselitteinen on esimerkiksi kieli

$$\{a^i b^j c^k \mid i = j \text{ tai } j = k\},$$

mutta tämän väitteen todistus on suhteellisen mutkikas ja sivuutetaan tässä.

3.4 Rekursiivisesti etenevä jäsentäminen

LL(1)-kieliopit ja niiden jäsentäminen

Tarkastellaan seuraavaa yksinkertaista, pelkästään yhteen- ja vähennyslaskuja sisältäviä aritmeettisia lausekkeita tuottavaa kielioppia G:

$$\begin{aligned} E &\rightarrow T + E \mid T - E \mid T \\ T &\rightarrow a \mid (E). \end{aligned}$$

Muokataan G:stä välikkeen E produktiot "tekijöimällä" ekvivalentti (so. saman kielen tuottava) kielioppi G':

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +E \mid -E \mid \lambda \\ T &\rightarrow a \mid (E). \end{aligned}$$

Kieliopin G' tuottamille lauseille on hyvin helppoa muodostaa vasemmat johdot suoraan lähtösymbolista E alkaen, sillä jäsennyksen joka vaiheessa määrää tavoitteena olevan lauseen *seuraava merkki* yksikäsitteisesti sen, mikä lavennettavana olevaan välikkeeseen liittyvä produktio on valittava. Esimerkiksi lauseen $a + a$ jäsentäminen sujuu seuraavasti:

$$\begin{array}{lccccccccccccc} \text{Vuorossa oleva merkki:} & & & & a & & + & & & & a & & \langle\text{eof}\rangle & \\ \text{Vasen johto:} & E & \underset{\text{lm}}{\Rightarrow} & TE' & \underset{\text{lm}}{\Rightarrow} & aE' & \underset{\text{lm}}{\Rightarrow} & a+E & \underset{\text{lm}}{\Rightarrow} & a+TE' & \underset{\text{lm}}{\Rightarrow} & a+aE' & \underset{\text{lm}}{\Rightarrow} & a+a. \end{array}$$

Kielioppia, jolla on tämä ominaisuus, sanotaan *LL(1)-kieliopiksi*[3].

LL(1)-tyyppiselle kieliopille on helppo kirjoittaa jäsennysohjelma suoraan rekursiivisina proseduureina. Esimerkiksi kieliopin G' pohjalta voidaan muodostaa seuraava proseduurikokoelma, joka syötejonon jäsennyksen yhteydessä tulostaa sen tuottavan vasemman johdon produktiot järjestyksessä. ("Todellisessa" jäsennysohjelmassa tietenkin produktioiden tulostamisen sijaan tehtäisiin jotakin tuottavaa työtä, esimerkiksi koodingenerointia tai laskentaa.)

```
var next : char;

function getnext : char; ...          {Seuraavan merkin selaus.}

procedure ERROR(msg : text); ...      {Virheiden käsittely.}

procedure E;
begin writeln('E → TE');
   T; E'
end;

procedure E';
begin
   if next = '+' then
   begin writeln('E' → +E');
      next := getnext;
      E
   end
   else if next = '-' then
   begin writeln('E' → -E');
      next := getnext;
      E
   end
   else writeln('E' → λ')
end;

procedure T;
begin
   if next = 'a' then
```

[3] Lyhenne LL(1), tai yleisemmin LL(k), tulee jäsennysprosessin englanninkielisestä kuvauksesta: "*l*eft-to-right scan, producing a *l*eft parse, with k symbol lookahead".

```
        begin writeln('T → a');
            next := getnext
        end
        else if next = '(' then
        begin writeln('T → (E)');
            next := getnext;
            E;
            if next ≠ ')' then ERROR(') expected.');
            next := getnext
        end
        else ERROR('T cannot start with this.')
    end;

    begin {Pääohjelma}
        next := getnext;
        E
    end.
```

Esimerkiksi syötejonoa `a-(a+a)` käsitellessään ohjelma tulostaa seuraavat rivit:

```
E → TE'
T → a
E' → -E
E → TE'
T → (E)
E → TE'
T → a
E' → +E
E → TE'
T → a
E' → λ
E' → λ.
```

Tämä tulostus vastaa vasenta johtoa:

$$\begin{aligned} E &\Rightarrow TE' \Rightarrow aE' \Rightarrow a - E \Rightarrow a - TE' \Rightarrow a - (E)E' \Rightarrow a - (TE')E' \\ &\Rightarrow a - (aE')E' \Rightarrow a - (a + E)E' \Rightarrow a - (a + TE')E' \Rightarrow a - (a + aE')E' \\ &\Rightarrow a - (a + a)E' \Rightarrow a - (a + a). \end{aligned}$$

LL(1)-kielioppien yleinen muoto

Yleisessä tapauksessa LL(1)-kieliopit voivat olla hieman edellä esitettyä mutkikkaampia: lisäpiirteitä tuovat produktiot, joiden oikeat puolet alkavat välikkeellä, ja *tyhjentyvät* (engl. nullable) välikkeet: sellaiset A, joilla $A \Rightarrow^* \lambda$.

Tarkastellaan esimerkkinä seuraavaa, säännöllisen lausekkeen $a^*b \cup c^*d$ kuvaaman kielen tuottavaa kielioppia:

$$\begin{array}{lcl} S & \rightarrow & Ab \mid Cd \\ A & \rightarrow & aA \mid \lambda \\ C & \rightarrow & cC \mid \lambda. \end{array}$$

Jos tätä kielioppia käytettäessä jäsennettävä lause alkaa a:lla tai b:llä, pitää ensin soveltaa produktiota $S \to Ab$, jos taas c:llä tai d:llä, niin produktiota $S \to Cd$. Kielioppi on siten LL(1)-tyyppinen, vaikka alussa sovellettavaa produktiota ei voidakaan päätellä pelkästään S:n produktioiden perusteella.

Tämänkaltaisten tilanteiden käsittelemiseksi määritellään seuraavat annetun kieliopin $G = (V, \Sigma, P, S)$ välikkeisiin liittyvät päätemerkkijoukot[4]:

$$\begin{aligned}
\mathrm{FIRST}(A) &= \{a \in \Sigma \mid A \Rightarrow^* ax \text{ jollakin } x \in \Sigma^*\} \\
&\quad \cup \{\lambda \mid A \Rightarrow^* \lambda\} \\
&= \{A\text{:sta johdettavien päätejonojen ensimmäiset merkit}\} \\
&\quad \cup \{\lambda, \text{jos } A \text{ voi tyhjentyä}\}; \\
\mathrm{FOLLOW}(A) &= \{a \in \Sigma \mid S \Rightarrow^* \alpha A a \beta \text{ joillakin } \alpha, \beta \in V^*\} \\
&\quad \cup \{\lambda \mid S \Rightarrow^* \alpha A \text{ jollakin } \alpha \in V^*\} \\
&= \{\text{ne päätemerkit, jotka voivat seurata } A\text{:ta jossakin } G\text{:n lausejohdoksessa}\} \\
&\quad \cup \{\lambda, \text{jos } A \text{ voi sijaita lausejohdoksen lopussa}\}.
\end{aligned}$$

Esimerkiksi edellisen esimerkkikieliopin tapauksessa saadaan:

$$\begin{array}{lll}
\mathrm{FIRST}(S) = \{a, b, c, d\}, & \mathrm{FIRST}(A) = \{a, \lambda\}, & \mathrm{FIRST}(C) = \{c, \lambda\} \\
\mathrm{FOLLOW}(S) = \{\lambda\}, & \mathrm{FOLLOW}(A) = \{b\}, & \mathrm{FOLLOW}(C) = \{d\}.
\end{array}$$

Laajennetaan FIRST-joukkojen määritelmä mielivaltaisille merkkijonoille seuraavasti:

$$\begin{aligned}
\mathrm{FIRST}(\lambda) &= \{\lambda\}; \\
\mathrm{FIRST}(a) &= \{a\} \qquad \text{kaikilla } a \in \Sigma; \\
\mathrm{FIRST}(X_1 \ldots X_k) &= \begin{cases}
\mathrm{FIRST}(X_1) \cup \ldots \cup \mathrm{FIRST}(X_i) - \{\lambda\}, \\
\qquad \text{jos } \lambda \in \mathrm{FIRST}(X_1), \ldots, \mathrm{FIRST}(X_{i-1}), \lambda \notin \mathrm{FIRST}(X_i); \\
\mathrm{FIRST}(X_1) \cup \ldots \cup \mathrm{FIRST}(X_k), \\
\qquad \text{jos } \lambda \in \mathrm{FIRST}(X_i) \text{ kaikilla } i = 1, \ldots, k.
\end{cases}
\end{aligned}$$

Vielä tarvitaan määritelmän suoraviivainen laajennus yksittäisiltä merkkijonoilta merkkijonojoukkoihin:

$$\mathrm{FIRST}(L) = \bigcup_{\omega \in L} \mathrm{FIRST}(\omega).$$

Kieliopin LL(1)-ehto voidaan nyt ilmaista täsmällisesti seuraavasti: kielioppi on LL(1)-muotoinen, jos sen millä tahansa kahdella samaan välikkeeseen A liittyvällä produktiolla $A \to \omega_1$ ja $A \to \omega_2$, $\omega_1 \neq \omega_2$, on voimassa:

$$\mathrm{FIRST}(\{\omega_1\}\mathrm{FOLLOW}(A)) \cap \mathrm{FIRST}(\{\omega_2\}\mathrm{FOLLOW}(A)) = \emptyset.$$

LL(1)-kieliopin rekursiivisesti etenevä jäsentäjä muodostetaan yleisessä tapauksessa seuraavan kaavan mukaan:

[4] Tarkkaan ottaen joukot voivat sisältää yksittäisten päätemerkkien lisäksi myös tyhjän merkkijonon λ.

Apurutiinit:

```
function getnext : token;          {Seuraavan päätesymbolin selaus
...                                   — voi olla > 1 kirjainmerkkiä.}
procedure ERROR(msg: text);        {Virheiden käsittely; parametri msg
...                                   sisältää virheilmoitustekstin.}
```

Välikettä A vastaava proseduuri:

```
procedure A;                       {A:n produktiot A → ω1 | ... | ωn}
begin
  if next in [a11,...,a1m1] then   {FIRST({ω1}FOLLOW(A)) = {a11,...,a1m1}}
  begin {produktio A → ω1}
    parse(ω1)
  end else
  :
  if next in [an1,...,anmn] then   {FIRST({ωn}FOLLOW(A)) = {an1,...,anmn}}
  begin {produktio A → ωn}
    parse(ωn)
  end else
  ERROR('A cannot start with this.')
end;
```

Tässä lyhennemerkintä "$parse(\omega_i)$" tarkoittaa seuraavalla tavalla muodostettavaa käskyjonoa:

$parse(X_1 \ldots X_k)$	$\equiv$	$parse(X_1); \ldots; parse(X_k)$,	missä
$parse(a)$	$\equiv$	**if** $next \neq a$ **then** ERROR('a `expected.`');	
		$next := getnext$,	kun a on päätemerkki;
$parse(B)$	$\equiv$	B,	kun B on välike.

Kielioppien muokkaaminen LL(1)-muotoon

LL(1)-kieliopit ovat varsin suppea, joskin hyödyllinen kielioppiluokka[5]. Seuraavassa esitetään kaksi kielioppimuunnosta, joilla "melkein" LL(1)-muotoisia kielioppeja voidaan muuntaa tähän muotoon.

1. Vasen tekijöinti

Kielioppi, jossa on produktiot

$$A \to \alpha\beta_1 \mid \alpha\beta_2, \qquad \alpha \neq \lambda,\ \beta_1 \neq \beta_2$$

[5]Sallimalla yhden merkin sijaan k merkin mittaiset "kurkistusjonot" saadaan yleisemmät LL(k)-kieliopit, jotka voidaan jäsentää samaan tapaan kuin tässä. Vielä yleisempi, mutta vaikeammin jäsennettävä, on ns. LR(k)-kielioppien luokka.

ei voi täyttää LL(1)-ehtoa[6]. Tällainen ongelmapaikka voidaan kuitenkin korjata ottamalla käyttöön uusi välike A' ja korvaamalla em. produktiot produktioilla:

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta_1 \mid \beta_2. \end{aligned}$$

Tässä on oletettu, että α on $\alpha\beta_1$:n ja $\alpha\beta_2$:n pisin yhteinen alkuosa, so. että jonot β_1 ja β_2 alkavat eri tavalla.

Esimerkiksi kieliopissa G (s. 41) korvattiin produktiot

$$E \rightarrow T + E \mid T - E \mid T$$

tekijöidyllä esityksellä

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +E \mid -E \mid \lambda. \end{aligned}$$

2. Välittömän vasemman rekursion poistaminen

Kielioppi on *vasemmalle rekursiivinen*, jos jollakin välikkeellä A ja merkkijonolla γ on

$$A \Rightarrow^+ A\gamma,$$

missä merkintä $\alpha \Rightarrow^+ \beta$ tarkoittaa, että α:sta voidaan johtaa β johdolla, jonka pituus on vähintään yksi askel.

Vasemmalle rekursiivinen kielioppi ei voi täyttää LL(1)-ehtoa[7]. *Välitön* vasen rekursio, so. suorat johdot $A \Rightarrow A\gamma$, voidaan kuitenkin välttää korvaamalla produktiot

$$A \rightarrow A\beta \mid \alpha, \qquad \beta \neq \lambda,$$

produktioilla

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta A' \mid \lambda. \end{aligned}$$

Muotoa $A \rightarrow A$ olevat produktiot, jos sellaisia on, voidaan yksinkertaisesti jättää pois.

Esimerkiksi produktioista

$$E \rightarrow E + T \mid E - T \mid T$$

voidaan poistaa välitön vasen rekursio korvaamalla ne produktioilla

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid -TE' \mid \lambda. \end{aligned}$$

Vasemman rekursion poisto on periaatteessa mahdollista yleisemminkin. Jokainen kontekstiton kielioppi voidaan nimittäin muuntaa ns. *Greibachin normaalimuotoon* (engl. Greibach normal form), missä kaikki produktiot ovat muotoa

$$A \rightarrow aB_1 \ldots B_k, \qquad k \geq 0,$$

tai $S \rightarrow \lambda$, missä a on päätemerkki, $B_1, \ldots, B_k$ välikkeitä ja S lähtösymboli.

[6] Paitsi siinä poikkeuksellisessa tapauksessa, että ainoa α:n tuottama päätejono on λ.

[7] Paitsi siinä tapauksessa, että rekursiivinen johto on välikkeelle A ainoa mahdollinen – mutta silloin A:sta ei voida johtaa mitään päätejonoa, ja se voidaan poistaa kieliopista tuotettua kieltä muuttamatta.

* Liite: FIRST- ja FOLLOW-joukkojen laskeminen

Annetun kieliopin $G = (V, \Sigma, P, S)$ välikkeisiin liittyvät FIRST- ja FOLLOW-joukot voidaan muodostaa systemaattisesti seuraavalla menetelmällä.

Ensin lasketaan FIRST-joukot:

1. Asetetaan aluksi kaikille kieliopin päätteille $a \in \Sigma$:

$$\text{FIRST}(a) := \{a\},$$

ja kaikille välikkeille $A \in V - \Sigma$:

$$\begin{aligned} \text{FIRST}(A) \;:=\; & \{a \in \Sigma \mid A \to a\beta \text{ on } G\text{:n produktio}\} \\ & \cup \{\lambda \mid A \to \lambda \text{ on } G\text{:n produktio}\}. \end{aligned}$$

2. Käydään sitten kieliopin produktioita läpi jossakin järjestyksessä ja toistetaan seuraavaa, kunnes FIRST-joukot eivät enää kasva: kullekin produktiolle $A \to X_1 \ldots X_k$ asetetaan:

$$\begin{aligned} \text{FIRST}(A) \;:=\; & \text{FIRST}(A) \cup \\ & \bigcup\{\text{FIRST}(X_i) \mid 1 \le i \le k,\ \lambda \in \text{FIRST}(X_j) \text{ kaikilla } j < k\} \\ & \cup \{\lambda \mid \lambda \in \text{FIRST}(X_j) \text{ kaikilla } j = 1, \ldots, k\}. \end{aligned}$$

FOLLOW-joukot määritetään sitten FIRST-joukkojen avulla seuraavasti:

1. Asetetaan aluksi kaikille välikkeille $B \in V - \Sigma^*$:

$$\text{FOLLOW}(B) := \bigcup\{\text{FIRST}(\beta) - \{\lambda\} \mid A \to \alpha B\beta \text{ on } G\text{:n produktio}\},$$

ja lähtösymbolille S lisäksi:

$$\text{FOLLOW}(S) := \text{FOLLOW}(S) \cup \{\lambda\}.$$

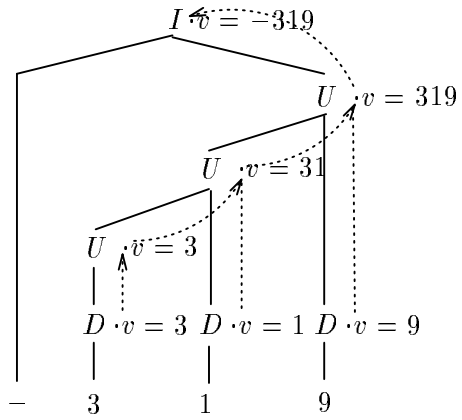
2. Sitten toistetaan, kunnes FOLLOW-joukot eivät enää kasva: kullekin produktiolle $A \to \alpha B\beta$, missä $\lambda \in \text{FIRST}(\beta)$, asetetaan:

$$\text{FOLLOW}(B) := \text{FOLLOW}(B) \cup \text{FOLLOW}(A).$$

3.5 Attribuuttikieliopit

Kätevä tapa liittää kontekstittomiin kielioppeihin yksinkertaista kielen semantiikan kuvausta ovat ns. *attribuuttikieliopit* (engl. attribute grammars).

Ideana tässä on, että kukin kieliopin mukaisen jäsennyspuun solmu, jonka nimenä on symboli X, ajatellaan "tietueeksi", joka on "tyyppiä" X. "Tietuetyyppiin" X kuuluvia "kenttiä" sanotaan X:n *attribuuteiksi* (engl. attributes) ja merkitään $X.s$, $X.t$ jne. Kussakin X-tyyppisessä jäsennyspuun solmussa ajatellaan olevan X:n attribuuteista eri *ilmentymät* (engl. instances).



Kuva 3.5: Attributoitu jäsennyspuu.

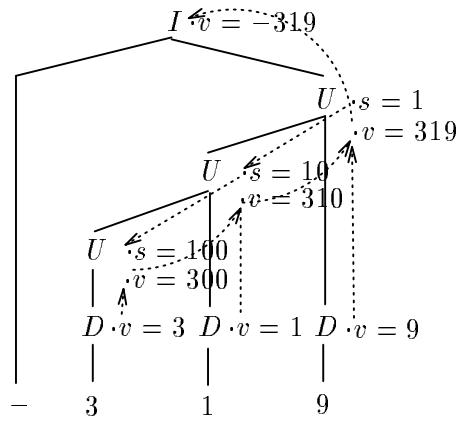
Kieliopin produktioihin $A \to X_1 \ldots X_k$ liitetään sitten attribuuttien *evaluointisääntöjä* (engl. evaluation rules), jotka ilmaisevat miten annetun jäsennyspuun solmun attribuutti-ilmentymien arvot määräytyvät sen isä- ja jälkeläissolmujen attribuutti-ilmentymien arvoista. Säännöt voivat olla periaatteessa minkälaisia funktioita tahansa, kunhan niiden argumentteina esiintyy vain paikallisesti saatavissa olevaa tietoa. Tarkemmin sanoen: produktioon $A \to X_1 \ldots X_k$ liitettävissä säännöissä saa mainita vain symbolien $A, X_1, \ldots, X_k$ attribuutteja.

Esimerkiksi seuraavassa on etumerkillisiä kokonaislukuja tuottavaan kontekstittomaan kielioppiin liitetty attribuutit ja niiden evaluointisäännöt kieliopin tuottamien lukujen arvojen määrittämiseen. Kuhunkin jäsennyspuun X-tyyppiseen välikesolmuun liitetään attribuutti-ilmentymä $X.v$, jonka arvoksi tulee X:stä tuotetun numerojonon lukuarvo; erityisesti juurisolmun v-ilmentymän arvoksi tulee koko puun tuotoksena olevan numerojonon lukuarvo.

Produktiot:			*Evaluointisäännöt:*		
I	$\to$	$+U$	$I.v$	$:=$	$U.v$
I	$\to$	$-U$	$I.v$	$:=$	$-U.v$
I	$\to$	U	$I.v$	$:=$	$U.v$
U	$\to$	D	$U.v$	$:=$	$D.v$
U	$\to$	$U\,D$	$U_1.v$	$:=$	$10 * U_2.v + D.v$
D	$\to$	0	$D.v$	$:=$	0
D	$\to$	1	$D.v$	$:=$	1
$\vdots$					
D	$\to$	9	$D.v$	$:=$	9

Produktioon $U \to U\,D$ liittyvässä evaluointisäännössä on tässä käytetty indeksimerkintää saman välikkeen eri esiintymien erottamiseen: U_1 tarkoittaa ensimmäistä produktiossa esiintyvää U:ta, U_2 toista jne. Tässä tapauksessa U_1 viittaa produktion vasemman puolen ja U_2 oikean puolen U:hun.

Kuvassa 3.5 on esitetty evaluointisääntojen mukainen *attributoitu jäsennyspuu* kieliopin tuottamalle lauseelle "-319". Kuvaan on selvyyden vuoksi merkitty katkoviivoilla näkyviin attribuutti-ilmentymien väliset evaluointiriippuvuudet.



Kuva 3.6: Positiokerrointa käyttäen attributoitu jäsennyspuu.

Attribuuttikieliopin attribuutti t on *synteettinen* (engl. synthetic), jos sen kuhunkin produktioon $A \to X_1 \ldots X_k$ liittyvä evaluointisääntö on muotoa $A.t := f(A, X_1, \ldots, X_k)$. Tämä merkitsee sitä, että jäsennyspuussa kunkin solmun mahdollisen t-ilmentymän arvo riippuu vain solmun omien ja sen jälkeläisten attribuutti-ilmentymien arvoista; esimerkiksi edellä attribuutti v on synteettinen. Muunlaiset attribuutit ovat *periytyviä* (engl. inherited).

Attribuuttisemantiikan kuvauksessa pyritään käyttämään pääasiassa synteettisiä attribuutteja, koska ne voidaan evaluoida helposti yhdellä jäsennyspuun lehdistä juureen suuntautuvalla läpikäynnillä. Mitään periaatteellista estettä myös perittyjen attribuuttien käyttöön ei kuitenkaan ole — kunhan attribuutti-ilmentymien riippuvuusverkkoihin ei tule syklejä. Esimerkiksi edellä olevaan, etumerkillisiä kokonaislukuja tuottavaan kielioppiin voitaisiin liittää lukujen arvot määrittävä semantiikka myös seuraavasti, periytyvää "positiokerroin"-attribuuttia s ja synteettistä "arvo"-attribuuttia v käyttäen:

Produktiot:		*Evaluointisäännöt:*		
$I \to +U$		$U.s := 1,$		$I.v := U.v$
$I \to -U$		$U.s := 1,$		$I.v := -U.v$
$I \to U$		$U.s := 1,$		$I.v := U.v$
$U \to D$				$U.v := (D.v) * (U.s)$
$U \to UD$		$U_2.s := 10 * (U_1.s),$		$U_1.v := U_2.v + (D.v) * (U_1.s)$
$D \to 0$				$D.v := 0$
$D \to 1$				$D.v := 1$
$\vdots$				
$D \to 9$				$D.v := 9$

Kuvassa 3.6 on esitetty tämän semantiikan mukainen attributoitu jäsennyspuu lauseelle "-319".

Attribuutti-ilmentymien arvot voidaan usein laskea suoraan jäsennysrutiineissa, tarvitsematta muodostaa jäsennyspuuta eksplisiittisesti. Esimerkkinä tästä on seuraavassa ohjelma, joka muuntaa syötteenä annettuja aritmeettisia lausekkeita ns. *postfix*-esitykseen.

Aritmeettisen lausekkeen postfix-esityksessä kirjoitetaan ensin operandit, sitten operaattori; esimerkiksi lauseke "$(a + b) * c$" kirjoitettaisiin "$ab + c*$". Tällaisen esityksen etu ta-

vanomaiseen *infix*-esitykseen verrattuna on, että postfix-esityksessä ei milloinkaan tarvita sulkuja operaatioiden suoritusjärjestyksen ilmaisemiseen. (Tästä syystä postfix-laskujärjestystä käytetään mm. eräissä taskulaskimissa.)

Seuraavassa on tavanomaiseen, yksinkertaisia aritmeettisia lausekkeita tuottavaan kielioppiin liitetty yksi synteettinen, merkkijonoarvoinen attribuutti pf; kuhunkin välikkeeseen X liittyvän attribuutti-ilmentymän $X.pf$ arvo on X:stä tuotetun alilausekkeen postfix-esitys.

Produktiot:			*Evaluointisäännöt:*		
E	$\rightarrow$	$T + E$	$E_1.pf$	:=	$(T.pf)\hat{}(E_2.pf)\hat{}('+')$
E	$\rightarrow$	T	$E.pf$	:=	$T.pf$
T	$\rightarrow$	$F * T$	$T_1.pf$	:=	$(F.pf)\hat{}(T_2.pf)\hat{}('*')$
T	$\rightarrow$	F	$T.pf$	:=	$F.pf$
F	$\rightarrow$	a	$F.pf$	:=	$'a'$
F	$\rightarrow$	(E)	$F.pf$	:=	$E.pf$

Kieliopin rekursiivisesti etenevä jäsentäjä, jossa attribuutti-ilmentymien arvot evaluoidaan suoraan jäsennyksen yhteydessä, on seuraava[8]:

```
var next : char;
    pf   : text;                    {Tuloksena saatava postfix-esitys}

function getnext: char; ...

procedure ERROR(msg : text); ...

procedure E(var pf : text);
var pf1, pf2: text;
begin {E → T + E | T}
    T(pf1);
    if next = '+' then
    begin
        next := getnext;
        E(pf2);
        pf := pf1^pf2^'+'
    end
    else pf := pf1
end;

procedure T(var pf : text);
var pf1, pf2: text;
begin {T → F * T | F}
    F(pf1);
    if next = '*' then
    begin
```

[8] Kieliopin saattaminen LL(1)-muotoon edellyttäisi tarkkaan ottaen välikkeiden E ja T produktioiden vasemman tekijöinnin. Tekijöinti voidaan kuitenkin sisällyttää implisiittisesti jäsennysrutiineihin esimerkistä ilmenevällä tavalla.

```
            next := getnext;
            T(pf2);
            pf := pf1^pf2^'*'
        end
        else pf := pf1
    end;

    procedure F(var pf : text);
    var pf1: text;
    begin {F → a | (E)}
        if next = 'a' then
        begin
            pf := 'a';
            next := getnext
        end else
        if next = '(' then
        begin
            next := getnext;
            E(pf1);
            if next ≠ ')' then ERROR(') expected.');
            next := getnext;
            pf := pf1
        end else
        ERROR('F cannot start with this.')
    end;

    begin {Pääohjelma}
        next := getnext;
        E(pf);
        writeln(pf)
    end.
```

3.6 Eräs yleinen jäsennysmenetelmä

Rekursiivisesti etenevä jäsentäminen on selkeä ja tehokas jäsennysmenetelmä LL(1)-kieliopeille: n merkin mittaisen syötemerkkijonon käsittely sujuu ajassa $O(n)$.

Mielivaltaisen kontekstittoman kieliopin tuottaminen merkkijonojen tunnistaminen ei sen sijaan ole aivan helppoa. Yksi suoraviivainen ratkaisutapa olisi muuntaa kielioppi ensin edellä mainittuun Greibachin normaalimuotoon. Tämänmuotoisella kieliopilla on se ominaisuus, että millä tahansa n merkin mittaisella lauseella on johto, jonka pituus on n askelta. Merkkijonon x, $|x| = n$, kuuluminen tuotettuun kieleen voitaisiin sitten testata käymällä läpi kaikki n askelen mittaiset johdot ja katsomalla, tuottaako jokin niistä x:n. Tämä on kuitenkin hyvin tehoton ratkaisutapa, sillä jokaisessa epätriviaalissa kieliopissa on n askelen mittaisia johtoja eksponentiaalinen määrä, so. $O(c^n)$ kappaletta jollakin $c \geq 2$.

Seuraavassa esitetään eräs yleinen menetelmä, ns. *Cocke–Younger–Kasami*-t. *CYK-algoritmi*, mielivaltaisen kontekstittoman kieliopin tuottamien merkkijonojen tunnistamiseen. Menetel-

mä toimii ajassa $O(n^3)$, missä n on tutkittavan merkkijonon pituus.

CYK-algoritmia varten käsitellään ensin joitakin kielioppimuunnoksia.

λ-produktioiden poistaminen

Olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Välike $A \in V - \Sigma$ on *tyhjentyvä* (engl. nullable), jos $A \underset{G}{\Rightarrow}^* \lambda$.

Lemma 3.5 *Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa enintään lähtösymboli on tyhjentyvä.*

Huom. Lähtösymbolin tyhjentymistä ei voida välttää, jos $\lambda \in L(G)$.

Todistus. Olkoon $G = (V, \Sigma, P, S)$. Selvitetään ensin G:n tyhjentyvät välikkeet seuraavasti:

(i) asetetaan aluksi

$$\text{NULL} := \{A \in V - \Sigma \mid A \to \lambda \text{ on } G\text{:n produktio}\};$$

(ii) toistetaan sitten seuraavaa NULL-joukon laajennusoperaatiota, kunnes joukko ei enää kasva:

$$\begin{aligned} \text{NULL} \;:=\; & \text{NULL} \cup \\ & \{A \in V - \Sigma \mid A \to B_1 \ldots B_k \text{ on } G\text{:n produktio, } B_i \in \text{NULL kaikilla } i = 1, \ldots, k\}. \end{aligned}$$

Tämän jälkeen korvataan kukin G:n produktio $A \to X_1 \ldots X_k$ kaikkien sellaisten produktioiden joukolla, jotka ovat muotoa

$$A \to \alpha_1 \ldots \alpha_k, \qquad \text{missä} \quad \alpha_i = \begin{cases} X_i, & \text{jos } X_i \notin \text{NULL}; \\ X_i \text{ tai } \lambda, & \text{jos } X_i \in \text{NULL}. \end{cases}$$

Lopuksi poistetaan kaikki muotoa $A \to \lambda$ olevat produktiot. Jos poistettavana on myös produktio $S \to \lambda$, otetaan muodostettavaan kielioppiin G' uusi lähtösymboli S' ja sille produktiot $S' \to S$ ja $S' \to \lambda$. $\square$

Esimerkki λ-produktioiden poistamisesta:

$$\begin{aligned} S &\to A \mid B \\ A &\to aBa \mid \lambda \qquad \Rightarrow \quad (\text{NULL} = \{A, B, S\}) \\ B &\to bAb \mid \lambda \end{aligned}$$

$$\begin{aligned} S &\to A \mid B \mid \lambda \\ A &\to aBa \mid aa \mid \lambda \qquad \Rightarrow \\ B &\to bAb \mid bb \mid \lambda \end{aligned}$$

$$\begin{aligned} S' &\to S \mid \lambda \\ S &\to A \mid B \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb. \end{aligned}$$

Yksikköproduktioiden poistaminen

Produktio muotoa $A \to B$, missä A ja B ovat välikkeitä, on *yksikköproduktio* (engl. unit production).

Lemma 3.6 *Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa ei ole yksikköproduktioita.*

Todistus. Olkoon $G = (V, \Sigma, P, S)$. Selvitetään ensin G:n kunkin välikkeen "yksikköseuraajat" seuraavasti:

(i) asetetaan aluksi kullekin $A \in V - \Sigma$:

$$F(A) := \{B \in V - \Sigma \mid A \to B \text{ on } G\text{:n produktio}\};$$

(ii) toistetaan sitten seuraavia F-joukkojen laajennusoperaatiota, kunnes joukot eivät enää kasva:

$$F(A) := F(A) \cup \bigcup \{F(B) \mid A \to B \text{ on } G\text{:n produktio}\}.$$

Tämän jälkeen poistetaan G:stä kaikki yksikköproduktiot ja lisätään niiden sijaan kaikki mahdolliset produktiot muotoa $A \to \omega$, missä $B \to \omega$ on G:n ei-yksikköproduktio jollakin $B \in F(A)$. □

Esimerkkinä poistetaan yksikköproduktiot edellä saadusta kieliopista

$$\begin{aligned} S' &\to S \mid \lambda \\ S &\to A \mid B \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb. \end{aligned}$$

Välikkeiden yksikköseuraajat ovat: $F(S') = \{S, A, B\}$, $F(S) = \{A, B\}$, $F(A) = F(B) = \emptyset$. Korvaamalla yksikköproduktiot edellä esitetyllä tavalla saadaan kielioppi

$$\begin{aligned} S' &\to aBa \mid aa \mid bAb \mid bb \mid \lambda \\ S &\to aBa \mid aa \mid bAb \mid bb \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb. \end{aligned}$$

Chomskyn normaalimuoto

Kontekstiton kielioppi $G = (V, \Sigma, P, S)$ on *Chomskyn normaalimuodossa* (engl. Chomsky normal form), jos sen välikkeistä enintään S on tyhjentyvä, ja mahdollista produktiota $S \to \lambda$ lukuunottamatta muut produktiot ovat muotoa $A \to BC$ tai $A \to a$, missä A, B ja C ovat välikkeitä ja a on päätemerkki. Lisäksi vaaditaan yksinkertaisuuden vuoksi, että lähtösymboli S ei esiinny minkään produktion oikealla puolella.

Lause 3.7 *Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti Chomskyn normaalimuotoinen kielioppi G'.*

Todistus. Olkoon $G = (V, \Sigma, P, S)$. Poistetaan ensin G:stä λ-produktiot ja yksikköproduktiot lemmojen 3.5 ja 3.6 konstruktioilla. Tämän jälkeen kaikki G:n produktiot ovat muotoa $A \to a$ tai $A \to X_1 \ldots X_k$, $k \geq 2$ (tai $S \to \lambda$).

Lisätään ensin kielioppiin kutakin päätemerkkiä a varten uusi välike C_a ja sille produktio $C_a \to a$. Korvataan sitten kussakin muotoa $A \to X_1 \ldots X_k$, $k \geq 2$, olevassa produktiossa ensin kaikki päätemerkit em. uusilla välikkeillä, ja sitten koko produktio produktiojoukolla

$$\begin{aligned} A &\to X_1 A_1 \\ A_1 &\to X_2 A_2 \\ &\vdots \\ A_{k-2} &\to X_{k-1} X_k, \end{aligned}$$

missä $A_1, \ldots, A_{k-2}$ ovat jälleen uusia välikkeitä.

(Tarkkaan ottaen uusi produktiojoukko on siis oikeastaan

$$\begin{aligned} A &\to X_1' A_1 \\ A_1 &\to X_2' A_2 \\ &\vdots \\ A_{k-2} &\to X_{k-1}' X_k', \end{aligned}$$

missä

$$X_i' = \begin{cases} X_i, & \text{jos } X_i \in V - \Sigma; \\ C_a, & \text{jos } X_i = a \in \Sigma.) \end{cases} \quad \square$$

Esimerkiksi sovellettaessa konstruktiota kielioppiin

$$\begin{aligned} S &\to aBCd \mid bbb \\ B &\to b \\ C &\to c \end{aligned}$$

saadaan tuloksena kielioppi

$$\begin{aligned} S &\to C_a S_1^1 \\ S_1^1 &\to B S_2^1 \\ S_2^1 &\to C C_d \\ S &\to C_b S_1^2 \\ S_1^2 &\to C_b C_b \\ B &\to b \\ C &\to c \\ C_a &\to a \\ C_b &\to b \\ C_c &\to c \\ C_d &\to d. \end{aligned}$$

Cocke–Younger–Kasami-algoritmi

Olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Lauseen 3.7 nojalla voidaan olettaa, että G on Chomskyn normaalimuodossa. Kysymys, kuuluuko annettu merkkijono x kieleen $L(G)$ voidaan tällöin ratkaista seuraavasti:

- Jos $x = \lambda$, niin $x \in L(G)$ joss $S \to \lambda$ on G:n produktio.
- Muussa tapauksessa merkitään $x = a_1 \dots a_n$ ja tarkastellaan x:n osajonot tuottavien välikkeiden joukkoja

$$N_{ij} = \{A \in V - \Sigma \mid A \underset{G}{\Rightarrow}^* a_i \dots a_j\}, \qquad 1 \le i \le j \le n.$$

  Nämä joukot voidaan laskea taulukoimalla lyhyistä osajonoista pitempiin alla esitettävällä tavalla. Selvästi on $x \in L(G)$ joss $S \in N_{1n}$.

  Joukkojen N_{ij} laskeminen:

  (i) asetetaan aluksi kaikilla $i = 1, \dots, n$:

$$N_{ii} := \{A \in V - \Sigma \mid A \to a_i \text{ on } G\text{:n produktio}\};$$

  (ii) lasketaan sitten kaikilla $k = 1, \dots, n-1$ ja kullakin k kaikilla $i = 1, \dots, n-k$:

$$N_{i,i+k} := \{A \in V - \Sigma \mid \text{jollakin } j,\ i \le j < i+k, \text{ on välikkeet } B \in N_{ij} \text{ ja } C \in N_{j+1,i+k} \text{ s.e. } A \to BC \text{ on } G\text{:n produktio}\}. \quad \square$$

Esimerkkinä CYK-algoritmin soveltamisesta tarkastellaan Chomskyn normaalimuotoista kielioppia

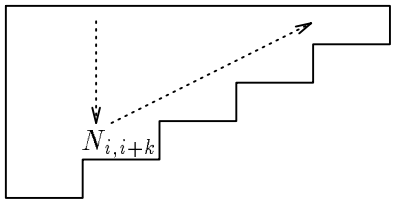
$$\begin{array}{lcl} S & \to & AB \mid BC \\ A & \to & BA \mid a \\ B & \to & CC \mid b \\ C & \to & AB \mid a \end{array}$$

ja syötemerkkijonoa $x = baaba$. Algoritmin laskenta etenee tässä tapauksessa seuraavasti:

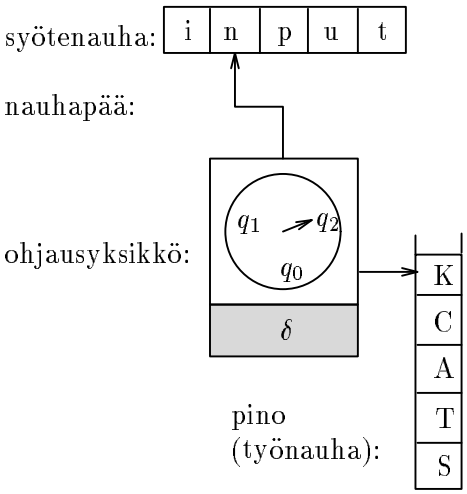
		$i \to$				
	$N_{i,i+k}$	$1:b$	$2:a$	$3:a$	$4:b$	$5:a$
	0	B	A, C	A, C	B	A, C
	1	S, A	B	S, C	S, A	–
$k \downarrow$	2	$\emptyset$	B	B	–	
	3	$\emptyset$	S, A, C	–		
	4	S, A, C	–			

Tässä tapauksessa lähtösymboli S kuuluu joukkoon N_{15}, joten päätellään, että x kuuluu kieliopin tuottamaan kieleen.

Yleisesti ottaen CYK-algoritmin laskentajärjestys on sellainen, että jotakin joukkoa $N_{i,i+k}$ määritettäessä edetään samanaikaisesti sarakkeessa N_{ij} joukkoa $N_{i,i+k}$ "kohti" ja diagonaalia $N_{j+1,i+k}$ pitkin siitä "poispäin" (kuva 3.7).



Kuva 3.7: CYK-algoritmin laskentajärjestys.



Kuva 3.8: Pinoautomaatti.

3.7 Pinoautomaatit

Samaan tapaan kuin säännölliset kielet voidaan tunnistaa äärellisillä automaateilla, saadaan kontekstittomille kielille automaattikarakterisointi ns. *pinoautomaattien* (engl. pushdown automata) avulla.

Intuitiivisesti pinoautomaatti on äärellinen automaatti, johon on lisätty yksi potentiaalisesti ääretön *työnauha* (kuva 3.8). Työnauhan käyttö ei kuitenkaan ole rajoittamatonta, vaan tieto on sillä organisoitu *pinoksi*: automaatti voi lukea ja kirjoittaa vain nauhan toiseen päähän, ja päästäkseen lukemaan aiemmin kirjoittamiaan merkkejä sen täytyy pyyhkiä viimeisin merkki pois. Formaalisti tämä idea voidaan kuvata seuraavasti:

Määritelmä 3.2 *Pinoautomaatti* (engl. pushdown automaton) on kuusikko

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

missä

- Q on *tilojen* äärellinen joukko;
- Σ on *syöteaakkosto*;
- Γ on *pinoaakkosto*;

- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\lambda\}))$ on (joukkoarvoinen) *siirtymäfunktio*;
- $q_0 \in Q$ on *alkutila*;
- $F \subseteq Q$ on *(hyväksyvien) lopputilojen* joukko.

Siirtymäfunktion arvon

$$\delta(q, \sigma, \gamma) = \{(q_1, \gamma_1), \ldots, (q_k, \gamma_k)\}$$

tulkinta on, että ollessaan tilassa q ja lukiessaan syötemerkin σ ja pinomerkin γ automaatti voi siirtyä johonkin tiloista $q_1, \ldots, q_k$ ja korvata vastaavasti pinon päällimmäisen merkin jollakin merkeistä $\gamma_1, \ldots, \gamma_k$. (Pinoautomaatit ovat siis yleisessä tapauksessa *epädeterministisiä*.) Jos $\sigma = \lambda$, automaatti tekee siirtymän syötemerkkiä lukematta; vastaavasti jos $\gamma = \lambda$, automaatti ei lue pinomerkkiä ja uusi kirjoitettu merkki tulee pinon päälle vanhaa päällimmäistä merkkiä poistamatta ("push"-operaatio). Jos pinosta luettu merkki on $\gamma \neq \lambda$ ja kirjoitettavana on $\gamma_i = \lambda$, pinosta poistetaan sen päällimmäinen merkki ("pop"-operaatio).

Automaatin *tilanne* on kolmikko $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$; erityisesti automaatin *alkutilanne syötteellä* x on kolmikko (q_0, x, λ). Tilanteen (q, w, α) intuitiivinen tulkinta on, että automaatti on tilassa q, syötemerkkijonon käsittelemätön osa on w ja pinossa on ylhäältä alas lukien merkkijono α.

Tilanne (q, w, α) *johtaa suoraan* tilanteeseen (q', w', α'), merkitään

$$(q, w, \alpha) \underset{M}{\vdash} (q', w', \alpha'),$$

jos voidaan kirjoittaa $w = \sigma w'$, $\alpha = \gamma\beta$, $\alpha' = \gamma'\beta$ ($|\sigma|, |\gamma|, |\gamma'| \leq 1$), siten että

$$(q', \gamma') \in \delta(q, \sigma, \gamma).$$

Tilanne (q, w, α) *johtaa tilanteeseen* (q', w', α'), merkitään

$$(q, w, \alpha) \underset{M}{\vdash^*} (q', w', \alpha'),$$

jos on olemassa tilannejono (q_0, w_0, α_0), (q_1, w_1, α_1), $\ldots$, (q_n, w_n, α_n), $n \geq 0$, siten että

$$(q, w, \alpha) = (q_0, w_0, \alpha_0) \underset{M}{\vdash} (q_1, w_1, \alpha_1) \underset{M}{\vdash} \cdots \underset{M}{\vdash} (q_n, w_n, \alpha_n) = (q', w', \alpha').$$

Pinoautomaatti M *hyväksyy* merkkijonon $x \in \Sigma^*$, jos

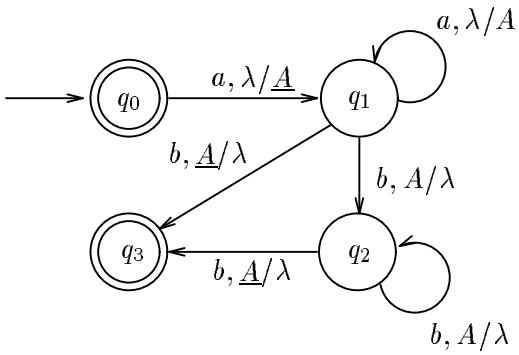
$$(q_0, x, \lambda) \underset{M}{\vdash^*} (q_f, \lambda, \alpha) \qquad \text{joillakin } q_f \in F \text{ ja } \alpha \in \Sigma^*,$$

siis jos se syötteen loppuessa on jossakin hyväksyvässä lopputilassa; muuten M *hylkää* x:n. Automaatin M *tunnistama kieli* on:

$$L(M) = \{x \in \Sigma^* \mid (q_0, x, \lambda) \underset{M}{\vdash^*} (q_f, \lambda, \alpha) \text{ joillakin } q_f \in F \text{ ja } \alpha \in \Sigma^*\}.$$

Esimerkiksi ei-säännöllinen kontekstiton kieli $\{a^k b^k \mid k \geq 0\}$ voidaan tunnistaa seuraavanlaisella pinoautomaatilla:

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, \underline{A}\}, \delta, q_0, \{q_0, q_3\}),$$



Kuva 3.9: Kielen $\{a^k b^k \mid k \geq 0\}$ tunnistava pinoautomaatti.

missä

$$\begin{aligned}
\delta(q_0, a, \lambda) &= \{(q_1, \underline{A})\}, \\
\delta(q_1, a, \lambda) &= \{(q_1, A)\}, \\
\delta(q_1, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_1, b, \underline{A}) &= \{(q_3, \lambda)\}, \\
\delta(q_2, b, A) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, \underline{A}) &= \{(q_3, \lambda)\}, \\
\delta(q, \sigma, \gamma) &= \emptyset \qquad \text{muilla } (q, \sigma, \gamma).
\end{aligned}$$

Esimerkiksi syötteellä *aabb* automaatti M toimii seuraavasti:

$$\begin{aligned}
(q_0, aabb, \lambda) &\vdash (q_1, abb, \underline{A}) \vdash (q_1, bb, A\underline{A}) \\
&\vdash (q_2, b, \underline{A}) \vdash (q_3, \lambda, \lambda).
\end{aligned}$$

Koska $q_3 \in F = \{q_0, q_3\}$, on siis $aabb \in L(M)$.

Myös pinoautomaateille voidaan kehittää kaavioesitys äärellisten automaattien tilasiirtymäkaavioita suoraviivaisesti yleistämällä. Esimerkiksi edellinen automaatti M voitaisiin esittää kuvan 3.9 mukaisena kaaviona.

Pinoautomaateilla on kontekstittomien kielten teoriassa ja uusien jäsennysmenetelmien kehittelyssä erittäin tärkeä asema, joka perustuu niiden vastaavuuteen kontekstittomien kielioppien kanssa:

Lause 3.8 *Kieli on kontekstiton, jos ja vain jos se voidaan tunnistaa pinoautomaatilla.*

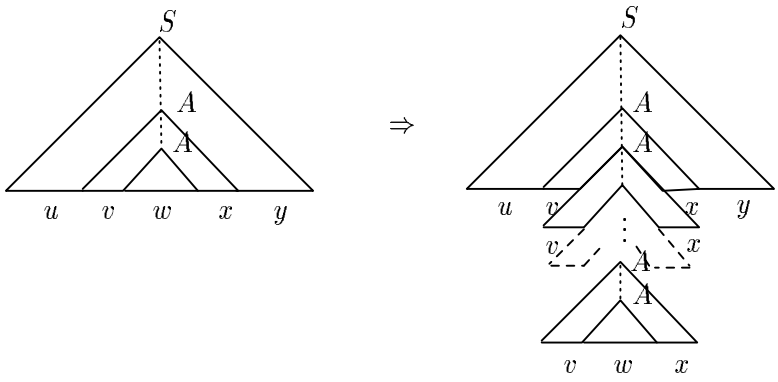
Todistus. Sivuutetaan. □

Pinoautomaatti M on *deterministinen*, jos jokaisella tilanteella (q, w, α) on enintään yksi mahdollinen seuraaja (q', w', α'), jolla

$$(q, w, \alpha) \underset{M}{\vdash} (q', w', \alpha')$$

Hieman yllättäen äärellisten automaattien peruslauseen 2.2 (s. 19) vastine ei pinoautomaattien kohdalla pidäkään paikkaansa, vaan *epädeterministiset pinoautomaatit ovat aidosti vahvempia kuin deterministiset.* Esimerkiksi kieli $\{ww^R \mid w \in \{a, b\}^*\}$[9] voidaan tunnistaa epädeterministisellä, mutta ei deterministisellä pinoautomaatilla. (Todistus sivuutetaan.)

[9] Merkintä w^R tarkoittaa merkkijonoa w takaperin kirjoitettuna: jos $w = a_1 a_2 \ldots a_k$, niin $w^R = a_k \ldots a_2 a_1$.



Kuva 3.10: Kontekstittoman kielen merkkijonon pumppaus.

Kontekstiton kieli on *deterministinen*, jos se voidaan tunnistaa jollakin deterministisellä pinoautomaatilla. Deterministisen kielten luokka on kontekstittomien kielten jäsennysteoriassa keskeinen, sillä siihen kuuluvat kielet voidaan jäsentää oleellisesti tehokkaammin kuin yleiset, mahdollisesti epädeterministisen automaatin vaativat kontekstittomat kielet.

3.8 ∗ Kontekstittomien kielten rajoituksista

Kontekstittomille kielille voidaan todistaa samantapainen pumppauslemma kuin säännöllisillekin kielille (Lemma 2.6, s. 32). Erona on vain se, että nyt merkkijonoa on pumpattava samanaikaisesti kahdesta paikasta. Muotoilunsa takia tulos tunnetaan myös "$uvwxy$-lemman" nimellä.

Lemma 3.9 (uvwxy-lemma) *Olkoon L kontekstiton kieli. Tällöin on olemassa sellainen $n \geq 1$, että mikä tahansa $z \in L$, $|z| \geq n$, voidaan jakaa osiin $z = uvwxy$ siten, että*

(i) $|vx| \geq 1$,

(ii) $|vwx| \leq n$,

(iii) $uv^iwx^iy \in L$ *kaikilla* $i = 0, 1, 2, \ldots$.

Todistus. Olkoon $G = (V, \Sigma, P, S)$ Chomskyn normaalimuotoinen kielioppi L:lle. Tällöin missä tahansa G:n jäsennyspuussa, jonka korkeus (= pisimmän juuresta lehteen kulkevan polun pituus) on h, on enintään 2^h lehteä. Toisin sanoen, minkä tahansa $z \in L$ jokaisessa jäsennyspuussa on polku, jonka pituus on vähintään $\log_2 |z|$.

Olkoon $k = |V - \Sigma|$ kieliopin G välikkeiden määrä. Asetetaan $n = 2^{k+1}$. Tarkastellaan jotakin $z \in L$, $|z| \geq n$, ja sen jotakin jäsennyspuuta.

Edellisen nojalla puussa on polku, jonka pituus on $\geq k + 1$; tällä polulla on siis jonkin välikkeen toistuttava. Olkoon A polun "alimmainen" toistuva välike (ks. kuva 3.10). Merkkijono z voidaan nyt osittaa $z = uvwxy$, missä w on A:n alimmasta ilmentymästä tuotettu osajono ja vwx seuraavaksi ylemmästä A:n ilmentymästä tuotettu osajono; osajonot saadaan johdosta

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy.$$

Koska siis $S \Rightarrow^* uAy$, $A \Rightarrow^* vAx$ ja $A \Rightarrow^* w$, osajonoja v ja x voidaan "pumpata" w:n ympärillä:

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uv^2Ax^2y \Rightarrow^* \ldots \Rightarrow^* uv^iAx^iy \Rightarrow^* uv^iwx^iy.$$

Siten $uv^iwx^iy \in L$ kaikilla $i = 0, 1, 2, \ldots$.

Koska kielioppi G on Chomskyn normaalimuodossa ja $A \Rightarrow^* vAx$, on oltava $|vx| \geq 1$. Koska edelleen tarkasteltavana on tietyn polun alimman toistuvan välikkeen kaksi alinta ilmentymää, on polun ylemmästä A:n ilmentymästä alkavan loppuosan pituus oltava enintään $k+1$. Tästä seuraa vastaavan alipuun tuotokselle pituusraja $|vwx| \leq 2^{k+1} = n$. □

Esimerkkinä lemman 3.9 soveltamisesta osoitetaan, että kieli $L = \{a^kb^kc^k \mid k \geq 0\}$ ei ole kontekstiton. Oletetaan nimittäin, että L olisi kontekstiton; valitaan parametri n lemman mukaisesti ja tarkastellaan merkkijonoa $z = a^nb^nc^n \in L$. Lemman mukaan z voidaan jakaa pumpattavaksi osiin

$$z = uvwxy, \qquad |vx| \geq 1, \qquad |vwx| \leq n.$$

Viimeisen ehdon takia merkkijono vx ei voi sisältää sekä a:ta, b:tä että c:tä. Merkkijonossa $uv^0wx^0y = uwy$ on siten ylijäämä jotakin merkkiä muihin merkkeihin nähden, eikä se voi olla kielen L määritelmässä vaadittua muotoa, vaikka lemman mukaan pitäisi olla $uwy \in L$.

Luku 4

Turingin koneet

4.1 Kielten tunnistaminen Turingin koneilla

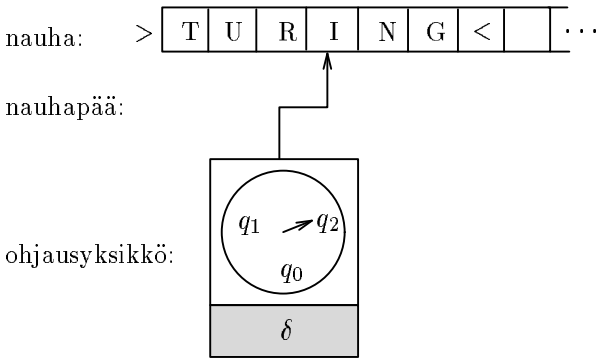
Seuraavassa esitettävän automaattimallin kehitti brittimatemaatikko Alan Turing vuosina 1935–36 — siis noin 10 vuotta ennen ensimmäisten tietokoneiden kehittämistä — pohtiessaan mekaanisen laskennan rajoja. Näennäisestä yksinkertaisuudestaan huolimatta tämä Turingin automaattimalli on huomattavan voimakas. Ns. *Churchin–Turingin teesin* mukaan peräti *mikä tahansa mekaanisesti* (lue: tietokoneella) *ratkeava ongelma voidaan ratkaista Turingin koneella.*

Churchin–Turingin teesiä ei voi muodollisesti todistaa oikeaksi, koska "mekaanisesti ratkeava ongelma" on intuitiivinen käsite, jonka kaikkia tulevia ilmentymiä on mahdoton ennustaa. Väitettä voi kuitenkin perustella sillä, että hyvin monet, riippumattomasti kehitetyt ja peruslähtökohdiltaan hyvinkin erilaiset mekaanisen laskennan formalisoinnit ovat lopulta osoittautuneet laskentavoimaltaan ekvivalenteiksi Turingin koneiden kanssa: esimerkkeinä mainittakoon Gödelin ja Kleenen rekursiivisesti määritellyt funktiot (1936), Churchin λ-kalkyyli (1936), Postin (1936) ja Markovin (1951) merkkijonomuunnossysteemit, sekä kaikki nykyiset ohjelmointikielet.

Tämän monisteen tavoitteiden kannalta Turingin koneita voidaan ajatella hyvin yksinkertaisena ohjelmointiformalismina, jolla voidaan ilmaista kaikki mitä vahvemmillakin ohjelmointikielillä — tosin kömpelösti. Turingin koneiden etu on siinä, että juuri yksinkertaisuutensa takia niitä on helppo käyttää mekaanisen laskettavuuden (so. ohjelmoitavuuden) rajoja koskevissa yleisissä tarkasteluissa.

Intuitiivisesti Turingin kone on kuin äärellinen automaatti, jolla syötenauhan sijaan on toiseen suuntaan loputtoman pitkä työnauha, jota kone pystyy nauhapään välityksellä lukemaan ja kirjoittamaan merkin kerrallaan (kuva 4.1). Nauhan alussa on erityinen alkumerkki '>', ja sen käytettyä osaa seuraa loppumerkki '<'. Kone pystyy lukiessaan havaitsemaan nämä merkit, mutta se ei pysty kirjoittamaan niitä. (Tarkemmin sanoen: alkumerkin tilalle kone ei saa kirjoittaa mitään muuta merkkiä, ja loppumerkki siirtyy automaattisesti eteenpäin sitä mukaa kuin kone kirjoittaa nauhalle lisää merkkejä.)

Tarkastellaan ensin Turingin koneiden käyttämistä formaalien kielten tunnistamiseen; myöhemmin käsitellään myös funktioiden laskemista näillä automaateilla. Kielten tunnistamista varten on kussakin Turingin koneessa kaksi lopputilaa, hyväksyvä q_{yes} ja hylkäävä q_{no}. Annetun merkkijonon tarkastamiseksi se kirjoitetaan koneen nauhalle sen vasempaan laitaan (alkumerkkiä lukuunottamatta), nauhapää sijoitetaan osoittamaan jonon ensimmäistä



Kuva 4.1: Turingin kone.

merkkiä, ja kone käynnistetään alkutilassa q_0. Tästä lähtien kone toimii askeleittain siirtymäfunktionsa ohjaamana: yhdessä siirtymässä se lukee nauhapään kohdalla olevan merkin ja päättää sitten tilansa ja luetun merkin perusteella, mikä on uusi tila, nauhapään kohdalle kirjoitettava uusi merkki, ja siirtyykö nauhapää käsitellyn merkin kohdalta yhden askelen verran vasemmalle vai oikealle[1]. Jos kone aikanaan pysähtyy hyväksyvässä lopputilassa q_{yes}, merkkijono kuuluu koneen tunnistamaan kieleen; jos se taas pysähtyy lopputilassa q_{no} *tai jää pysähtymättä*, merkkijono ei kuulu kieleen.

Täsmällisesti voidaan määritellä:

Määritelmä 4.1 *Turingin kone* (engl. Turing machine) on seitsikko

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}}),$$

missä

- Q on koneen *tilojen* äärellinen joukko;
- Σ on koneen *syöteaakkosto*;
- $\Gamma \supseteq \Sigma$ on koneen *nauha-aakkosto*; oletetaan, että merkit $>, < \notin \Gamma$;
- $\delta : (Q - \{q_{\text{yes}}, q_{\text{no}}\}) \times (\Gamma \cup \{>, <\}) \to Q \times (\Gamma \cup \{>, <\}) \times \{L, R\}$ on koneen *siirtymäfunktio*; siirtymäfunktion arvoilta
$$\delta(q, a) = (q', b, \Delta)$$
vaaditaan, että
  (i) jos $b = >$, niin $a = >$;
  (ii) jos $a = >$, niin $b = >$ ja $\Delta = R$;
  (iii) jos $b = <$, niin $a = <$ ja $\Delta = L$;
- $q_0 \in Q$ on koneen *alkutila*;
- $q_{\text{yes}} \in Q$ on koneen *hyväksyvä* ja $q_{\text{no}} \in Q$ sen *hylkäävä lopputila*.

[1] Määritelmien yksinkertaistamiseksi nauhapään ei sallita pysyä paikallaan siirtymässä. Paikallaan pysymistä voidaan tarvittaessa jäljitellä siirtämällä nauhapäätä yhden askelen verran edestakaisin.

Siirtymäfunktion arvon

$$\delta(q, a) = (q', b, \Delta)$$

tulkinta on, että ollessaan tilassa q ja lukiessaan nauhamerkin (tai alku- tai loppumerkin) a, kone siirtyy tilaan q', kirjoittaa lukemaansa paikkaan merkin b, ja siirtää nauhapäätä yhden merkkipaikan verran suuntaan Δ ($L \sim$ "left", $R \sim$ "right"). Sallittuja kirjoitettavia merkkejä ja siirtosuuntia on rajoitettu, mikäli $a =$ '$>$' tai '$<$', ja siirtymäfunktion arvo on aina määrittelemätön, kun $q = q_{\text{yes}}$ tai $q = q_{\text{no}}$. Joutuessaan jompaan kumpaan näistä tiloista kone pysähtyy heti.

Koneen *tilanne* on nelikko $(q, u, a, v) \in Q \times \Gamma^* \times (\Gamma \cup \{\lambda\}) \times \Gamma^*$, missä voi olla $a = \lambda$, mikäli myös $u = \lambda$ tai $v = \lambda$. Tilanteen (q, u, a, v) intuitiivinen tulkinta on, että kone on tilassa q, nauhan sisältö sen alusta nauhapään vasemmalle puolelle on u, nauhapään kohdalla on merkki a ja nauhan sisältö nauhapään oikealta puolelta käytetyn osan loppuun on v. Mahdollisesti on $a = \lambda$, jos nauhapää sijaitsee aivan nauhan alussa tai sen käytetyn osan lopussa. Ensimmäisessä tapauksessa ajatellaan, että kone "havaitsee" merkin '$>$' ja toisessa tapauksessa merkin '$<$'. *Alkutilanne syötteellä* $x = a_1 a_2 \ldots a_n$ on nelikko $(q_0, \lambda, a_1, a_2 \ldots a_n)$. Tilannetta (q, u, a, v) merkitään yleensä yksinkertaisemmin $(q, u\underline{a}v)$, ja alkutilannetta syötteellä x yksinkertaisesti $(q_0, \underline{x})$.

Relaatiota, jossa tilanne (q, w) *johtaa suoraan* tilanteeseen (q', w'), merkitään tavalliseen tapaan

$$(q, w) \underset{M}{\vdash} (q', w'),$$

ja se määritellään koneen M siirtymäfunktion pohjalta seuraavasti: kaikilla $q, q' \in Q$, $u, v \in \Gamma^*$, $a, b \in \Gamma$ ja $c \in \Gamma \cup \{\lambda\}$:

- jos $\delta(q, a) = (q', b, R)$, niin $(q, u\underline{a}cv) \underset{M}{\vdash} (q', ub\underline{c}v)$;
- jos $\delta(q, a) = (q', b, L)$, niin $(q, uc\underline{a}v) \underset{M}{\vdash} (q', u\underline{c}bv)$;
- jos $\delta(q, >) = (q', >, R)$, niin $(q, \underline{\lambda}cv) \underset{M}{\vdash} (q', \underline{c}v)$;
- jos $\delta(q, <) = (q', b, R)$, niin $(q, u\underline{\lambda}) \underset{M}{\vdash} (q', ub\underline{\lambda})$;
- jos $\delta(q, <) = (q', b, L)$, niin $(q, uc\underline{\lambda}) \underset{M}{\vdash} (q', u\underline{c}b)$;
- jos $\delta(q, <) = (q', <, L)$, niin $(q, uc\underline{\lambda}) \underset{M}{\vdash} (q', u\underline{c})$.

Tilanteet, jotka ovat muotoa (q_{yes}, w) tai (q_{no}, w) eivät johda mihinkään muuhun tilanteeseen. Näissä tilanteissa kone *pysähtyy*.

Tilanne (q, w) *johtaa tilanteeseen* (q', w'), merkitään

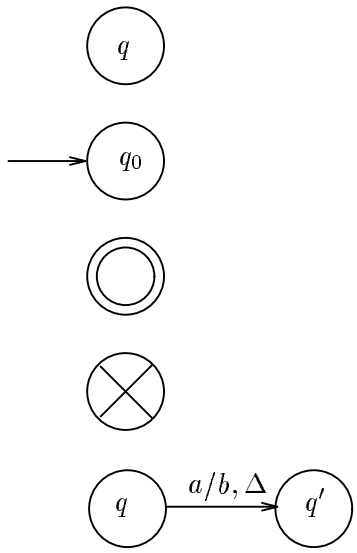
$$(q, w) \underset{M}{\vdash^*} (q', w'),$$

jos on olemassa tilannejono $(q_0, w_0), (q_1, w_1), \ldots, (q_n, w_n)$, $n \geq 0$, siten että

$$(q, w) = (q_0, w_0) \underset{M}{\vdash} (q_1, w_1) \underset{M}{\vdash} \cdots \underset{M}{\vdash} (q_n, w_n) = (q', w').$$

Turingin kone M *hyväksyy* merkkijonon $x \in \Sigma^*$, jos

$$(q_0, \underline{x}) \underset{M}{\vdash^*} (q_{\text{yes}}, w) \qquad \text{jollakin } w \in \Gamma^*;$$



Kuva 4.2: Turingin koneiden kaavioesityksen merkinnät.

muuten M *hylkää* x:n. Koneen M *tunnistama kieli* on:

$$L(M) = \{x \in \Sigma^* \mid (q_0, \underline{x}) \vdash_M^* (q_{\text{yes}}, w) \text{ jollakin } w \in \Gamma^*\}.$$

Esimerkiksi kieli $\{a^{2^k} \mid k \geq 0\}$ voidaan tunnistaa Turingin koneella

$$M = (\{q_0, q_1, q_{\text{yes}}, q_{\text{no}}\}, \{a\}, \{a\}, \delta, q_0, q_{\text{yes}}, q_{\text{no}}),$$

missä

$$\begin{aligned} \delta(q_0, a) &= (q_1, a, R), \\ \delta(q_1, a) &= (q_0, a, R), \\ \delta(q_0, <) &= (q_{\text{yes}}, <, L), \\ \delta(q_1, <) &= (q_{\text{no}}, <, L). \end{aligned}$$

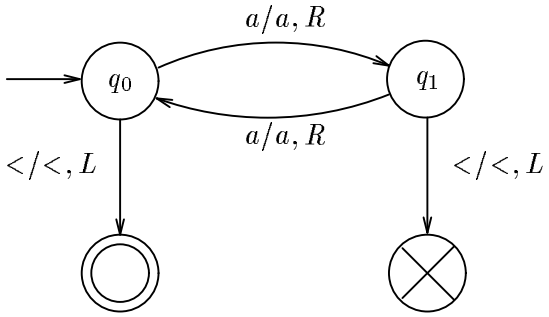
Koneen M laskenta esimerkiksi syötteellä aaa etenee seuraavasti:

$$(q_0, \underline{a}aa) \vdash_M (q_1, a\underline{a}a) \vdash_M (q_0, aa\underline{a}) \vdash_M (q_1, aaa\underline{\lambda}) \vdash_M (q_{\text{no}}, aa\underline{a}).$$

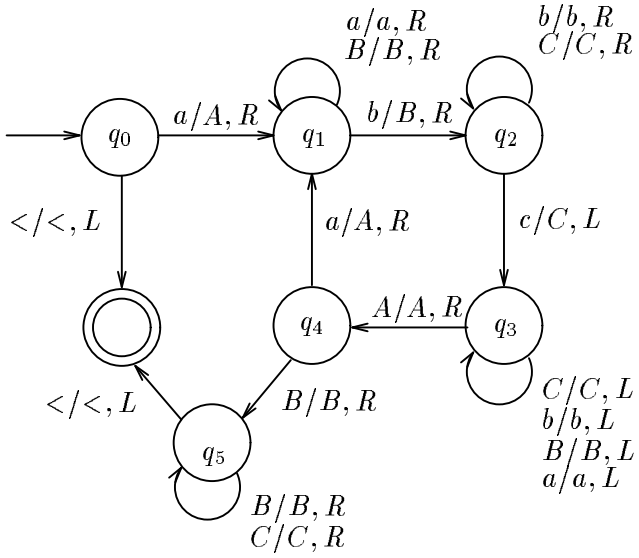
Koska kone pysähtyy tilassa q_{no}, päätellään että $aaa \notin L(M)$.

Turingin koneet voidaan kätevimmin esittää samantapaisilla kaavioilla kuin oli käytössä äärellisille automaateille ja pinoautomaateille; kaavioesityksissä käytetyt merkinnät on esitelty kuvassa 4.2. Esimerkiksi edellistä, kielen $\{a^{2^k} \mid k \geq 0\}$ tunnistavaa Turingin konetta vastaava kaavio on esitetty kuvassa 4.3.

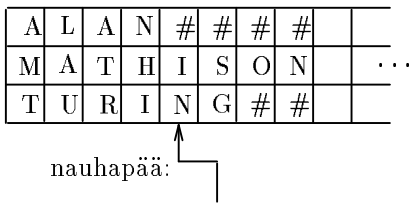
Mutkikkaampana esimerkkinä on kuvassa 4.4 esitetty ei-kontekstittoman kielen $\{a^k b^k c^k \mid k \geq 0\}$ tunnistava kone. Kaavion yksinkertaistamiseksi on tässä noudatettu käytäntöä, jonka mukaan siirtymiä hylkäävään lopputilaan ei merkitä näkyviin. Kuvatun koneen idea on, että se pitää kirjaa syötteestä tapaamistaan a-, b- ja c-merkeistä muuttamalla ne yksi kerrallaan A:ksi, B:ksi ja C:ksi. Muutettuaan viimeisen pienen a:n isoksi kone tarkastaa, että myöskään



Kuva 4.3: Kielen $\{a^{2k} \mid k \geq 0\}$ tunnistava Turingin kone.



Kuva 4.4: Kielen $\{a^k b^k c^k \mid k \geq 0\}$ tunnistava Turingin kone.



Kuva 4.5: Kolmeuraisen Turingin koneen nauha.

pieniä b- tai c-kirjaimia ei ole jäljellä. Esimerkiksi syötteeseen $aabbcc$ liittyvä laskenta etenee seuraavasti:

$$
\begin{array}{lll}
(q_0, \underline{a}abbcc) & \vdash & (q_2, AABB\underline{C}c) \quad \vdash \\
(q_1, A\underline{a}bbcc) & \vdash & (q_2, AABBC\underline{c}) \quad \vdash \\
(q_1, Aa\underline{b}bcc) & \vdash & (q_3, AABB\underline{C}C) \quad \vdash \\
(q_2, AaB\underline{b}cc) & \vdash & (q_3, AAB\underline{B}CC) \quad \vdash \\
(q_2, AaBb\underline{c}c) & \vdash & (q_3, AA\underline{B}BCC) \quad \vdash \\
(q_3, AaB\underline{b}Cc) & \vdash & (q_3, A\underline{A}BBCC) \quad \vdash \\
(q_3, Aa\underline{B}bCc) & \vdash & (q_4, AA\underline{B}BCC) \quad \vdash \\
(q_3, A\underline{a}BbCc) & \vdash & (q_5, AAB\underline{B}CC) \quad \vdash \\
(q_3, \underline{A}aBbCc) & \vdash & (q_5, AABB\underline{C}C) \quad \vdash \\
(q_4, A\underline{a}BbCc) & \vdash & (q_5, AABBC\underline{C}) \quad \vdash \\
(q_1, AA\underline{B}bCc) & \vdash & (q_5, AABBCC\underline{\lambda}) \quad \vdash \\
(q_1, AAB\underline{b}Cc) & \vdash & (q_{\mathrm{yes}}, AABBC\underline{C}).
\end{array}
$$

4.2 Turingin koneiden laajennuksia

Edellä esitettyä Turingin koneiden perusmääritelmää voidaan laajentaa monin eri tavoin koneilla tunnistettavien kielten luokan muuttumatta. Seuraavassa esitellään muutama hyödyllisin laajennus.

Moniuraiset koneet

Tässä laajennuksessa sallitaan, että Turingin koneen nauha koostuu k:sta rinnakkaisesta urasta, jotka kaikki kone lukee ja kirjoittaa yhdessä laskenta-askelessa. Koneen nauha on siis kuvassa 4.5 esitetyn tapainen (kuvassa $k = 3$). Koneen siirtymäfunktion arvot ovat vastaavasti muotoa:

$$\delta(q, (a_1, \ldots, a_k)) = (q', (b_1, \ldots, b_k), \Delta),$$

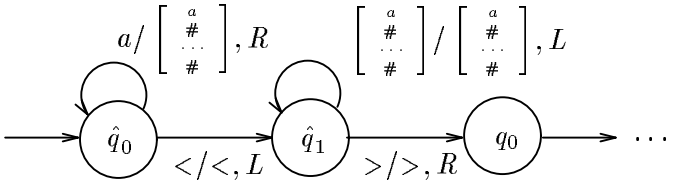
missä $a_1, \ldots, a_k$ ovat urilta $1, \ldots, k$ luetut merkit, $b_1, \ldots, b_k$ niiden tilalle kirjoitettavat merkit, ja $\Delta \in \{L, R\}$ on nauhapään siirtosuunta. Laskennan aluksi tutkittava syöte sijoitetaan ykkösuran vasempaan laitaan; muille urille tulee sen kohdalle erityisiä tyhjämerkkejä #. (Oletetaan, että tyhjämerkki kuuluu kaikkien moniuraisten koneiden nauha-aakkostoon.)

Formaalisti voidaan määritellä *k-urainen Turingin kone* (engl. k-track Turing machine) seitsikkona

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{yes}}, q_{\mathrm{no}}),$$

missä muut komponentit ovat kuten standardimallissa, paitsi siirtymäfunktio:

$$\delta : (Q - \{q_{\mathrm{yes}}, q_{\mathrm{no}}\}) \times (\Gamma^k \cup \{>, <\}) \to Q \times (\Gamma^k \cup \{>, <\}) \times \{L, R\}.$$



Kuva 4.6: Esiprosessori moniuraisen koneen syötteen nostamiseksi ykkösuralle.

Seuraajatilannerelaation $\underset{M}{\vdash}$, alkutilan jne. määritelmät ovat myös pieniä muutoksia lukuunottamatta samanlaiset kuin standardimallissa.

Moniuraisia Turingin koneita on hyvin helppo simuloida standardimallisilla, sillä kyseessä on oikeastaan vain nauha-aakkoston laajennus: k-uraisen koneen k päällekkäistä merkkiä voidaan nähdä yhtenä standardimallisen koneen "supermerkkinä".

Lause 4.1 *Jos formaali kieli L voidaan tunnistaa k-uraisella Turingin koneella, se voidaan tunnistaa myös standardimallisella Turingin koneella.*

Todistus. Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ k-urainen Turingin kone, joka tunnistaa kielen L. Vastaava standardimallinen kone $\widehat{M}$ voidaan muodostaa seuraavasti:

$$\widehat{M} = (\hat{Q}, \Sigma, \hat{\Gamma}, \hat{\delta}, \hat{q}_0, q_{\text{yes}}, q_{\text{no}}),$$

missä $\hat{Q} = Q \cup \{\hat{q}_0, \hat{q}_1, \hat{q}_2\}$, $\hat{\Gamma} = \Sigma \cup \Gamma^k$ (pääsääntöisesti yksi koneen $\widehat{M}$ merkki vastaa k:ta päällekkäistä M:n merkkiä; syötemerkit muodostavat poikkeuksen) ja kaikilla $q \in Q$ on

$$\hat{\delta}(q, \begin{bmatrix} a_1 \\ \vdots \\ a_k \end{bmatrix}) = (q', \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}, \Delta), \qquad \text{kun} \quad \delta(q, (a_1, \ldots, a_k)) = (q', (b_1, \ldots, b_k), \Delta).$$

Ainoa pieni ongelma koneen $\widehat{M}$ konstruktiossa on, että kunkin laskennan aluksi täytyy syötejono "nostaa" ykkösuralle, so. korvata nauhalla merkkijono $a_1 a_2 \ldots a_n$ merkkijonolla

$$\begin{bmatrix} a_1 \\ \# \\ \vdots \\ \# \end{bmatrix} \begin{bmatrix} a_2 \\ \# \\ \vdots \\ \# \end{bmatrix} \cdots \begin{bmatrix} a_n \\ \# \\ \vdots \\ \# \end{bmatrix}.$$

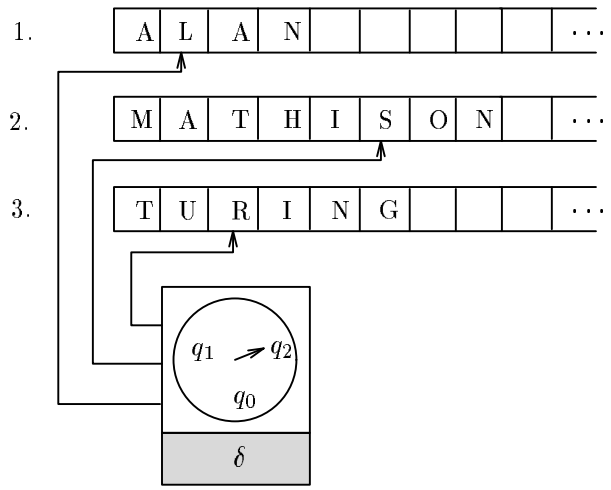
Tämä voidaan tehdä liittämällä M:stä kopioituun siirtymäfunktion osaan kuvassa 4.6 esitetty "esiprosessori" (kuvassa symboli a tarkoittaa "mitä tahansa aakkoston Σ merkkiä"). □

Moninauhaiset koneet

Tässä laajennuksessa sallitaan, että Turingin koneella on k toisistaan riippumatonta nauhaa, joilla on kullakin oma nauhapäänsä (kuva 4.7). Kone lukee ja kirjoittaa kaikki nauhat yhdessä laskenta-askelessa. Laskennan aluksi syöte sijoitetaan ykkösnauhan vasempaan laitaan ja kaikki nauhapäät nauhojensa alkuun. Tällaisen koneen siirtymäfunktion arvot ovat muotoa

$$\delta(q, a_1, \ldots, a_k) = (q', (b_1, \Delta_1), \ldots, (b_k, \Delta_k)),$$

missä $a_1, \ldots, a_k$ ovat nauhoilta $1, \ldots, k$ luetut merkit, $b_1, \ldots, b_k$ niiden tilalle kirjoitettavat merkit, ja $\Delta_1, \ldots, \Delta_k \in \{L, R\}$ nauhapäiden siirtosuunnat.



Kuva 4.7: Kolmenauhainen Turingin kone.

Formaalisti voidaan määritellä *k-nauhainen Turingin kone* (engl. k-tape Turing machine) seitsikkona

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}}),$$

missä muut komponentit ovat kuten standardimallissa, paitsi siirtymäfunktio:

$$\delta : (Q - \{q_{\text{yes}}, q_{\text{no}}\}) \times (\Gamma \cup \{>, <\})^k \to Q \times ((\Gamma \cup \{>, <\}) \times \{L, R\})^k.$$

Seuraajatilannerelaatio ym. peruskäsitteet määritellään pienin muutoksin entiseen tapaan.

Lause 4.2 *Jos formaali kieli L voidaan tunnistaa k-nauhaisella Turingin koneella, se voidaan tunnistaa myös standardimallisella Turingin koneella.*

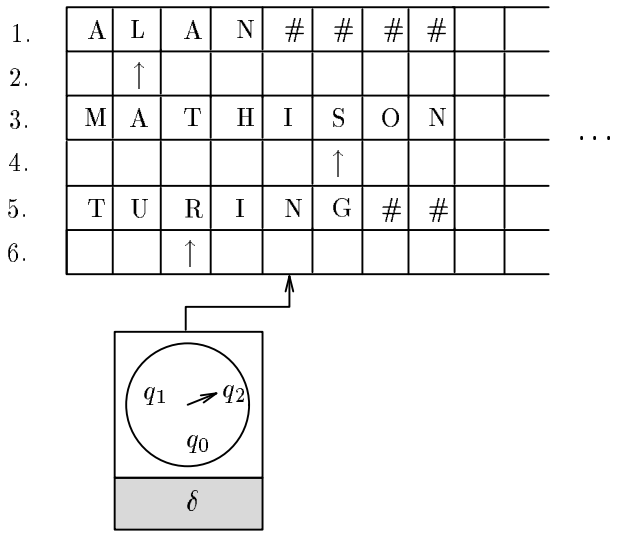
Todistus. Todistuskonstruktio on tässäkin tapauksessa käsitteellisesti suhteellisen yksinkertainen, mutta siihen kuuluu valitettavan paljon sotkuisia yksityiskohtia — niinpä seuraavasssa esitetään vain konstruktion keskeiset ideat.

Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ k-nauhainen Turingin kone, joka tunnistaa kielen L. Konetta M voidaan simuloida $2k$-uraisella koneella $\widehat{M}$ siten, että koneen $\widehat{M}$ parittomat urat $1, 3, 5, \ldots, 2k-1$ vastaavat M:n nauhoja $1, 2, \ldots, k$, ja kutakin paritonta uraa seuraavalla parillisella uralla on merkillä $\uparrow$ merkitty vastaavan nauhan nauhapään sijainti (kuva 4.8).

Simuloinnin aluksi syötemerkkijono sijoitetaan normaalisti koneen $\widehat{M}$ ykkösuralle, ja ensimmäisessä siirtymässään $\widehat{M}$ merkitsee nauhapääosoittimet $\uparrow$ parillisten urien ensimmäisiin merkkipaikkoihin.

Tämän jälkeen $\widehat{M}$ toimii "pyyhkimällä" nauhaa edestakaisin sen alku- ja loppumerkin välillä. Vasemmalta oikealle pyyhkäisyllä $\widehat{M}$ kerää tiedot kunkin osoittimen kohdalla olevasta M:n nauhamerkistä. Kun kaikki merkit ovat selvillä, $\widehat{M}$ simuloi yhden M:n siirtymän, ja takaisin oikealta vasemmalle suuntautuvalla pyyhkäisyllä kirjoittaa $\uparrow$-osoittimien kohdalle asianmukaiset uudet merkit ja siirtää osoittimia. Koneen $\widehat{M}$ siirtymäfunktion ohjelmoinnin tarkemmat yksityiskohdat sivuutetaan.

Moniurainen kone $\widehat{M}$ voidaan edelleen palauttaa standardimalliseksi edellisen lauseen konstruktiolla. □



Kuva 4.8: Kolmenauhaisen Turingin koneen simulointi kuusiuraisella.

Epädeterministiset koneet

Samaan tapaan kuin äärellisistä automaateista ja pinoautomaateista, myös Turingin koneista voidaan määritellä epädeterministinen versio[2]. "Ennustuskykynsä" vuoksi epädeterministiset Turingin koneet eivät sinänsä ole realistinen mekaanisen laskennan malli. Sen sijaan ne ovat, epädeterminististen äärellisten automaattien tapaan, tärkeä apuneuvo tietynlaisten laskennallisten ongelmien kuvaamiseen ja ongelmien osoittamiseen periaatteessa ratkeaviksi.

Formaalisti *epädeterminististen Turingin kone* (engl. nondeterministic Turing machine) voidaan määritellä seitsikkona

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}}),$$

missä muut komponentit ovat kuten deterministisessä standardimallissa, paitsi siirtymäfunktio:

$$\delta : (Q - \{q_{\text{yes}}, q_{\text{no}}\}) \times (\Gamma \cup \{>, <\}) \to \mathcal{P}(Q \times (\Gamma \cup \{>, <\}) \times \{L, R\}).$$

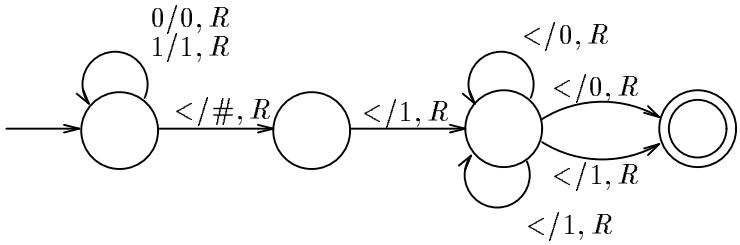
Siirtymäfunktion arvon

$$\delta(q, a) = \{(q_1, b_1, \Delta_1), \ldots, (q_k, b_k, \Delta_k)\}$$

tulkinta on, että ollessaan tilassa q ja lukiessaan merkin a kone voi toimia jonkin (intuitiivisesti "edullisimman") kolmikon (q_i, b_i, Δ_i) mukaisesti.

Epädeterministisen koneen tilanteet, tilannejohdot jne. määritellään formaalisti lähes samoin kuin deterministisenkin koneen tapauksessa: ainoa ero on, että ehdon $\delta(q, a) = (q', b, \Delta)$ sijaan kirjoitetaan $(q', b, \Delta) \in \delta(q, a)$. Tällä muutoksella on kuitenkin se tärkeä seuraus, että seuraajatilannerelaatio $\underset{M}{\vdash}$ ei ole enää yksiarvoinen: koneen tilanteella (q, w) voi nyt olla

[2]Pinoautomaattien tapauksessahan itse asiassa jo perusmääritelmä sisältää epädeterminismin, ja deterministinen versio on epädeterministisen *rajoitus*.



Kuva 4.9: Epädeterministinen Turingin kone GEN_INT.

useita vaihtoehtoisia seuraajia, so. tilanteita (q', w'), joilla $(q, w) \underset{M}{\vdash} (q', w')$[3].

Kerrataan vielä koneen M tunnistaman kielen määritelmä:

$$L(M) = \{x \in \Sigma^* \mid (q_0, \underline{x}) \underset{M}{\vdash}^* (q_{\text{yes}}, w) \text{ jollakin } w \in \Gamma^*\}.$$

Tämän määritelmän merkitys epädeterministisen koneen M tapauksessa on siis, että merkkijono x kuuluu M:n tunnistamaan kieleen, jos *jokin* M:n kelvollinen tilannejono johtaa alkutilanteesta syötteellä x hyväksyvään lopputilanteeseen.

Esimerkkinä epädeterministisen Turingin koneiden käytöstä tarkastellaan yhdistettyjen lukujen tunnistamista. Ei-negatiivinen kokonaisluku n on *yhdistetty*, jos sillä on kokonaislukutekijät $p, q \geq 2$, joilla $pq = n$. Luku, joka ei ole yhdistetty, on *alkuluku*. Kaikki tunnetut deterministiset yhdistettyjen lukujen testit joutuvat pahimmassa tapauksessa käymään läpi suuren joukon syötteen n potentiaalisia tekijöitä. Kuten seuraavassa nähdään, epädeterministisellä Turingin koneella yhdistettyjen lukujen "tunnistaminen" ei ole paljon yhtä kertolaskua työläämpää — mutta epädeterministinen kone ei oikeastaan annakaan mitään algoritmia lukujen tunnistamiseen, vaan on vain laskennallinen kuvaus sille, mitä yhdistetyt luvut *ovat*.

Oletetaan, että on jo suunniteltu deterministinen kone CHECK_MULT, joka tunnistaa kielen

$$L(\text{CHECK\_MULT}) = \{n\#p\#q \mid n, p, q \text{ binäärilukuja}, n = pq\}.$$

Tällaisen koneen suunnittelu on hieman työlästä, mutta ei periaatteessa kovin vaikeata. Olkoon lisäksi GO_START deterministinen Turingin kone, joka siirtää nauhapään osoittamaan nauhan ensimmäistä merkkiä. (Suunnittelu HT.)

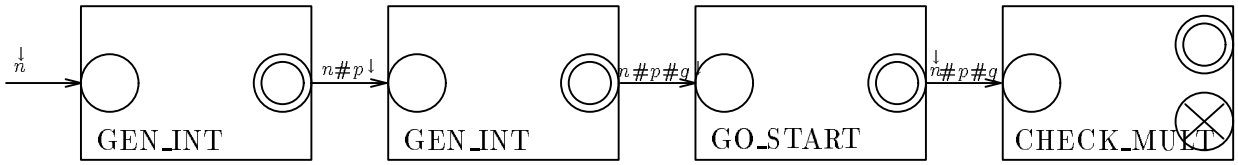
Olkoon edelleen GEN_INT kuvassa 4.9 esitetty, mielivaltaisen ykköstä suuremman binääriluvun nauhan loppuun tuottava epädeterministinen Turingin kone. Epädeterministinen Turingin kone TEST_COMPOSITE, joka tunnistaa kielen

$$L(\text{TEST\_COMPOSITE}) = \{n \mid n \text{ on binäärimuotoinen yhdistetty luku}\}$$

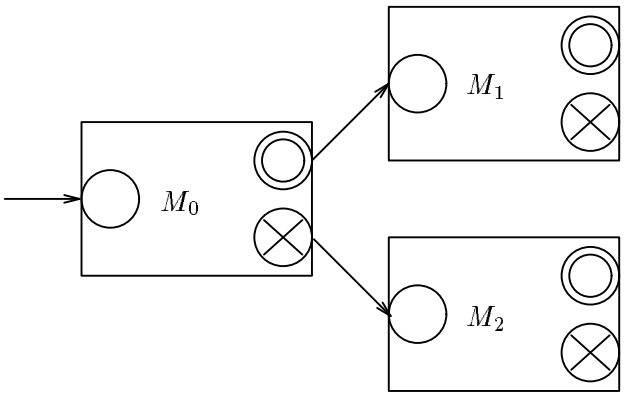
voidaan nyt muodostaa näistä komponenteista yhdistämällä kuvan 4.10 esittämällä tavalla. Nähdään, että yhdistetty kone hyväksyy syötteenä annetun binääriluvun n, jos ja vain jos on olemassa binääriluvut $p, q \geq 2$, joilla $n = pq$ — siis jos ja vain jos n on yhdistetty luku.

[3] Myöhemmin tullaan tarvitsemaan sitä tärkeää havaintoa, että näiden vaihtoehtoisten seuraajatilanteiden määrä on kuitenkin aina äärellinen, ja itse asiassa sitä rajoittaa vakio $r = r_M$, joka *riippuu vain koneen M rakenteesta*, ei tarkasteltavasta tilanteesta. Tarkemmin sanoen: vakioksi voidaan valita suurin M:n siirtymäfunktion arvojoukon koko,

$$r = \max\{|\delta(q, a)| \mid q \in Q, a \in \Gamma \cup \{>, <\}\}.$$



Kuva 4.10: Epädeterministinen Turingin kone TEST_COMPOSITE.



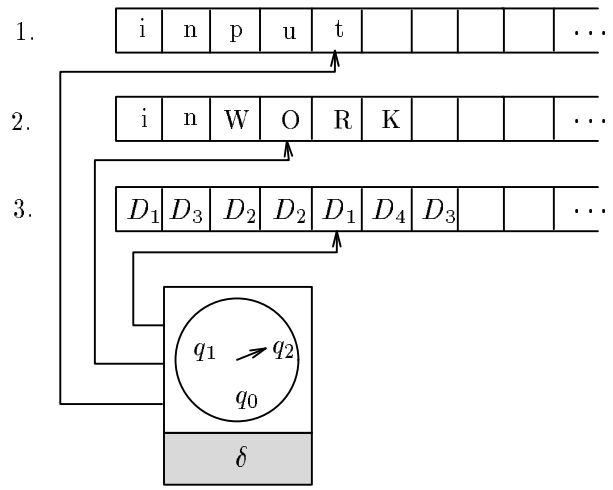
Kuva 4.11: Turingin koneiden yhdistäminen.

Kuvassa 4.10 on käytetty Turingin koneiden yhdistämiselle luonnollista kaaviomerkintää, joka yleisessä muodossaan voi olla kuvan 4.11 mukainen: jos koneet M_0, M_1 ja M_2 ovat mielivaltaisia Turingin koneita, niin kuvan esittämässä yhdistetyssä koneessa suoritetaan ensin koneen M_0 siirtymät ja sitten siirtymä M_0:n hyväksyvästä lopputilasta M_1:n alkutilaan ja vastaavasti M_0:n hylkäävästä lopputilasta M_2:n alkutilaan. — Tai tarkemmin sanoen koneen M_0 hyväksyvä (hylkäävä) lopputila samaistetaan M_1:n (M_2:n) alkutilan kanssa. Nämä tilat eivät tietenkään enää ole yhdistetyn koneen lopputiloja.

Lause 4.3 *Jos formaali kieli L voidaan tunnistaa epädeterministisellä Turingin koneella, se voidaan tunnistaa myös standardimallisella deterministisellä Turingin koneella.*

Todistus. Tyydytään jälleen esittämään vain simulointikonstruktion keskeiset ideat. Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ epädeterministinen Turingin kone, joka tunnistaa kielen L. Konetta M voidaan simuloida kuvan 4.12 mukaisella kolmenauhaisella deterministisellä koneella $\widehat{M}$, joka käy systemaattisesti läpi M:n mahdollisia laskentoja (tilannejonoja), kunnes löytää hyväksyvän — jos sellainen on olemassa. Kone $\widehat{M}$ voidaan edelleen muuntaa standardimalliseksi edellisten lauseiden konstruktioilla.

Yksityiskohtaisemmin sanoen kone $\widehat{M}$ toimii seuraavasti. Nauhalla 1 $\widehat{M}$ säilyttää kopiota syötejonosta ja nauhalla 2 se simuloi koneen M työnauhaa; kunkin simuloitavan laskennan aluksi $\widehat{M}$ kopioi syötteen nauhalta 1 nauhalle 2 ja pyyhkii pois (so. korvaa tyhjämerkeillä) nauhalle 2 edellisen laskennan jäljiltä mahdollisesti jääneet merkit. Nauhalla 3 $\widehat{M}$ pitää kirjaa vuorossa olevan laskennan "järjestysnumerosta". Tarkemmin sanoen, olkoon r suurin M:n siirtymäfunktion arvojoukon koko. Tällöin $\widehat{M}$:lla on erityiset nauhamerkit $D_1, \ldots, D_r$, joista koostuvia jonoja se generoi nauhalle 3 kanonisessa järjestyksessä (λ, D_1, D_2, …, D_r, D_1D_1,



Kuva 4.12: Epädeterministisen Turingin koneen simulointi deterministisellä.

D_1D_2, ..., D_1D_r, D_2D_1, ...). Kutakin generoitua jonoa kohden $\widehat{M}$ simuloi yhden M:n osittaisen laskennan, jossa epädeterministiset valinnat tehdään kolmosnauhan koodijonon ilmaisemalla tavalla. Esimerkiksi jos kolmosnauhalla on jono $D_1D_3D_2$, niin ensimmäisessä siirtymässä valitaan vaihtoehto 1, toisessa vaihtoehto 3, kolmannessa vaihtoehto 2; ellei tämä laskenta johtanut M:n hyväksyvään lopputilaan, generoidaan seuraava koodijono $D_1D_3D_3$ ja aloitetaan alusta. Jos koodijono on epäkelpo, so. jos siinä jossakin kohden on tilanteeseen liian suuri koodi, simuloitu laskenta keskeytetään ja generoidaan seuraava jono. On melko ilmeistä, että tämä systemaattinen koneen M laskentojen läpikäynti johtaa koneen $\widehat{M}$ hyväksymään syötejonon, jos ja vain jos koneella M on syötteen hyväksyvä laskenta. Jos hyväksyvää laskentaa ei ole, kone $\widehat{M}$ ei pysähdy. □

Luku 5

Rajoittamattomat ja kontekstiset kieliopit

Jos kontekstittomia kielioppeja yleistetään sallimalla produktioissa yhden välikkeen sijaan minkä tahansa välikkeistä ja päätteistä koostuvan epätyhjän merkkijonon korvaaminen toisella (korvaava merkkijono voi olla tyhjä), päästään *rajoittamattomien kielioppien* (engl. unrestricted grammars t. type 0 grammars) eli *yleisten muunnossysteemien* (engl. string rewriting systems) luokkaan.

Määritelmä 5.1 *Rajoittamaton kielioppi* on nelikko

$$G = (V, \Sigma, P, S),$$

missä

- V on kieliopin aakkosto;
- $\Sigma \subseteq V$ on kieliopin *päätemerkkien* joukko; $N = V - \Sigma$ on *välikemerkkien* t. *-symbolien* joukko;
- $P \subseteq V^+ \times V^*$ on kieliopin *sääntöjen* t. *produktioiden* joukko ($V^+ = V^* - \{\lambda\}$);
- $S \in N$ on kieliopin *lähtösymboli.*

Produktiota $(\omega, \omega') \in P$ merkitään tavallisesti $\omega \to \omega'$.

Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa suoraan* merkkijonon $\gamma' \in V^*$ kieliopissa G, merkitään

$$\gamma \underset{G}{\Rightarrow} \gamma'$$

jos voidaan kirjoittaa $\gamma = \alpha\omega\beta, \gamma' = \alpha\omega'\beta$ $(\alpha, \beta, \omega' \in V^*, \omega \in V^+)$, ja kieliopissa on produktio $\omega \to \omega'$. Jos kielioppi G on yhteydestä selvä, relaatiota voidaan merkitä yksinkertaisesti $\gamma \Rightarrow \gamma'$.

Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa* merkkijonon $\gamma' \in V^*$ kieliopissa G, merkitään

$$\gamma \underset{G}{\Rightarrow}^* \gamma'$$

jos on olemassa jono V:n merkkijonoja $\gamma_0, \gamma_1, \ldots, \gamma_n$ $(n \geq 0)$, siten että

$$\gamma = \gamma_0 \underset{G}{\Rightarrow} \gamma_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \gamma_n = \gamma'.$$

Jälleen, jos kielioppi G on yhteydestä selvä, merkitään yksinkertaisesti $\gamma \Rightarrow^* \gamma'$.

Merkkijono $\gamma \in V^*$ on kieliopin G *lausejohdos*, jos on $S \underset{G}{\Rightarrow}^* \gamma$. Pelkästään päätemerkeistä koostuva G:n lausejohdos $x \in \Sigma^*$ on G:n *lause*.

Kieliopin G *tuottama* t. *kuvaama kieli* $L(G)$ koostuu G:n lauseista, s.o.:

$$L(G) = \{x \in \Sigma^* \mid S \underset{G}{\Rightarrow}^* x\}.$$

Esimerkiksi seuraava rajoittamaton kielioppi tuottaa kielen $\{a^k b^k c^k \mid k \geq 0\}$, joka luvussa 3.8 todettiin ei-kontekstittomaksi:

$$\begin{aligned}
S &\rightarrow LT \mid \lambda \\
T &\rightarrow ABCT \mid ABC \\
BA &\rightarrow AB \\
CB &\rightarrow BC \\
CA &\rightarrow AC \\
LA &\rightarrow a \\
aA &\rightarrow aa \\
aB &\rightarrow ab \\
bB &\rightarrow bb \\
bC &\rightarrow bc \\
cC &\rightarrow cc.
\end{aligned}$$

Ideana tässä on, että kielioppi voi tuottaa epätyhjän lauseen, so. pelkästään päätemerkeistä $\{a, b, c\}$ koostuvan jonon ainoastaan suurinpiirtein seuraavalla tavalla[1]:

1. ensin johdetaan lähtösymbolista välikejono, joka on muotoa $L(ABC)^k$, jollakin $k \geq 1$;
2. sitten järjestetään välikkeet A, B, C aakkosjärjestykseen; tulos: $LA^k B^k C^k$;
3. lopuksi muutetaan välikkeet vastaaviksi päätteiksi vasemmalta alkaen; tulos: $a^k b^k c^k$.

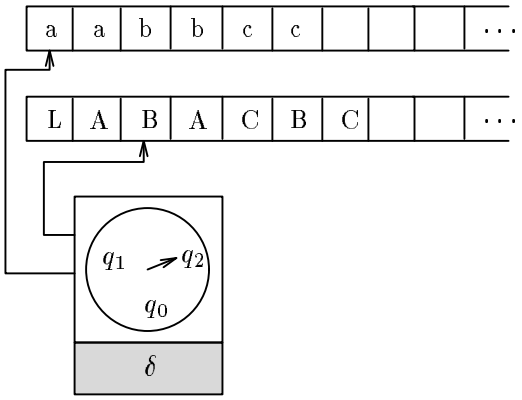
Esimerkiksi lause *aabbcc* voitaisiin johtaa:

$$\begin{aligned}
\underline{S} &\Rightarrow L\underline{T} \Rightarrow LABC\underline{T} \Rightarrow LAB\underline{CA}BC \Rightarrow LA\underline{BA}CBC \Rightarrow LAAB\underline{CB}C \\
&\Rightarrow \underline{LA}ABBCC \Rightarrow \underline{aA}BBCC \Rightarrow a\underline{aB}BCC \Rightarrow aab\underline{B}CC \\
&\Rightarrow aab\underline{bC}C \Rightarrow aabbc\underline{C} \Rightarrow aabbcc.
\end{aligned}$$

Sangen mielenkiintoinen ja ehkä yllättävä tulos on, että rajoittamattomat kieliopit ovat kuvausvoimaltaan täsmälleen ekvivalentteja Turingin koneiden kanssa. Tulos todistetaan seuraavassa kahdessa osassa.

Lause 5.1 *Jos formaali kieli L voidaan tuottaa rajoittamattomalla kieliopilla, se voidaan tunnistaa Turingin koneella.*

[1] Lause λ on tässä kieliopissa käsitelty erikoistapauksena.



Kuva 5.1: Rajoittamattoman kieliopin tuottaman kielen tunnistaminen Turingin koneella.

Todistus. Olkoon $G = (V, \Sigma, P, S)$ kielen L tuottava rajoittamaton kielioppi. Kieliopin G perusteella voidaan seuraavassa hahmoteltavalla tavalla muodostaa kielen L tunnistava kaksinauhainen epädeterministinen Turingin kone M_G. Kone M_G voidaan edelleen muuntaa yksinauhaiseksi ja determinisoida luvun 4.2 konstruktioilla.

Koneen M_G rakenne on kuvan 5.1 mukainen. Nauhalla 1 kone säilyttää kopiota syötejonosta. Nauhalla 2 on kullakin hetkellä jokin G:n lausejohdos, jota kone pyrkii muuntamaan syötejonon muotoiseksi. Toimintansa aluksi M_G kirjoittaa kakkosnauhalle yksinkertaisesti kieliopin lähtösymbolin S.

Koneen M_G laskenta koostuu vaiheista. Kussakin vaiheessa kone:

1. vie kakkosnauhan nauhapään epädeterministisesti johonkin kohtaan nauhalla;
2. valitsee epädeterministisesti jonkin G:n produktion, jota yrittää soveltaa valittuun nauhankohtaan (produktiot on koodattu koneen M_G siirtymäfunktioon);
3. jos produktion vasen puoli sopii yhteen nauhalla olevien merkkien kanssa, M_G korvaa ao. merkit produktion oikean puolen merkeillä (tämä voi edellyttää kakkosnauhan loppupään sisällön siirtämistä oikealle tai vasemmalle);
4. vaiheen lopuksi M_G vertaa ykkös- ja kakkosnauhan merkkijonoja toisiinsa: jos jonot ovat samat, kone siirtyy hyväksyvään lopputilaan ja pysähtyy, muuten aloittaa uuden vaiheen (kohta 1).

Konstruktion tarkemmat yksityiskohdat sivuutetaan. □

Lause 5.2 *Jos formaali kieli L voidaan tunnistaa Turingin koneella, se voidaan tuottaa rajoittamattomalla kieliopilla.*

Todistus. Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ kielen L tunnistava standardimallinen Turingin kone. Koneen M rakenteeseen nojautuen voidaan seuraavassa esitettävällä tavalla muodostaa kielen L tuottava rajoittamaton kielioppi G_M.

Konstruktion keskeinen idea on, että kieliopin G_M välikkeiksi otetaan (muiden muassa) kaikkia M:n tiloja $q \in Q$ edustavat symbolit. Koneen M tilanne $(q, u\underline{a}v)$ voidaan sitten

esittää merkkijonona $[uqav]$[2], ja M:n siirtymäfunktion perusteella G_M:ään muodostetaan produktiot, joiden ansiosta

$$[uqav] \underset{G_M}{\Rightarrow} [u'q'a'v'] \quad \text{jos ja vain jos} \quad (q, uav) \underset{M}{\vdash} (q', u'a'v').$$

Tämän seurauksena siis M hyväksyy syötteen x, jos ja vain jos $[q_0x] \underset{G_M}{\Rightarrow}^* [uq_{\text{yes}}v]$ joillakin $u, v \in \Sigma^*$.

Kaikkiaan kielioppiin G_M tulee kolme ryhmää produktioita:

1. Produktiot, joilla lähtösymbolista S voidaan tuottaa mikä tahansa merkkijono muotoa $x[q_0x]$, missä $x \in \Sigma^*$ ja $[$, q_0 ja $]$ ovat kieliopin G_M välikkeitä.

2. Edellä mainitut produktiot, joilla merkkijonosta $[q_0x]$ voidaan tuottaa merkkijono $[uq_{\text{yes}}v]$, jos ja vain jos M hyväksyy x:n.

3. Produktiot, joilla muotoa $[uq_{\text{yes}}v]$ oleva merkkijono muutetaan tyhjäksi merkkijonoksi.

Kieleen $L(M)$ kuuluvan merkkijonon x tuottaminen tapahtuu tällöin seuraavan kaavan mukaan:

$$S \overset{(1)}{\Rightarrow}^* x[q_0x] \overset{(2)}{\Rightarrow}^* x[uq_{\text{yes}}v] \overset{(3)}{\Rightarrow}^* x.$$

Täsmällisesti määritellään $G = (V, \Sigma, P, S)$, missä

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

ja produktiot P muodostuvat seuraavista kolmesta ryhmästä:

1. Alkutilanteen tuottaminen:

$$\begin{array}{lcll} S & \to & T[q_0] & \\ T & \to & \lambda & \\ T & \to & aTA_a & (a \in \Sigma) \\ A_a[q_0 & \to & [q_0A_a & (a \in \Sigma) \\ A_ab & \to & bA_a & (a, b \in \Sigma) \\ A_a] & \to & a] & (a \in \Sigma) \end{array}$$

2. M:n siirtymien simulointi ($a, b \in \Gamma$, $c \in \Gamma \cup \{[\}$) :

Siirtymät:			*Produktiot:*		
$\delta(q, a)$	$=$	(q', b, R)	qa	$\to$	bq'
$\delta(q, a)$	$=$	(q', b, L)	cqa	$\to$	$q'cb$
$\delta(q, >)$	$=$	$(q', >, R)$	$q[$	$\to$	$[q'$
$\delta(q, <)$	$=$	(q', b, R)	$q]$	$\to$	$bq']$
$\delta(q, <)$	$=$	(q', b, L)	$cq]$	$\to$	$q'cb]$
$\delta(q, <)$	$=$	$(q', <, L)$	$cq]$	$\to$	$q'c]$

[2] Tarkkaan ottaen vaatii lisäksi niiden tilanteiden esittäminen, joissa koneen nauhapää on alkumerkin kohdalla, myös muotoa $q[v]$ olevien merkkijonojen käyttämistä. Tämä mahdollisuus on otettu huomioon seuraavassa konstruktiossa.

3. Lopputilanteen siivous:

$$\begin{array}{lcll} q_{\text{yes}} & \rightarrow & E_L E_R & \\ q_{\text{yes}}[& \rightarrow & E_R & \\ aE_L & \rightarrow & E_L & (a \in \Gamma) \\ [E_L & \rightarrow & \lambda & \\ E_R a & \rightarrow & E_R & (a \in \Gamma) \\ E_R] & \rightarrow & \lambda & \end{array}$$

□

Tärkeä rajoittamattomien kielioppien osaluokka ovat ns. *kontekstiset kieliopit* (engl. context-sensitive grammars), joissa produktiot ovat muotoa $\omega \rightarrow \omega'$, missä $|\omega'| \geq |\omega|$, tai mahdollisesti $S \rightarrow \lambda$, missä S on lähtösymboli. Lisäksi vaaditaan, että jos kieliopissa on produktio $S \rightarrow \lambda$, niin lähtösymboli S ei esiinny minkään produktion oikealla puolella.

Nimitys "kontekstinen kielioppi" tulee tällaisten kielioppien eräästä normaalimuodosta, jossa produktiot ovat muotoa $S \rightarrow \lambda$ tai $\alpha A \beta \rightarrow \alpha \omega \beta$, missä A on kieliopin välike ja $\omega \neq \lambda$. Jälkimmäisen muotoisen produktion mukaan siis korvaussääntöä $A \rightarrow \omega$ saa soveltaa "kontekstissa" $\alpha \_ \beta$.

Formaali kieli L on *kontekstinen*, jos se voidaan tuottaa jollakin kontekstisella kieliopilla. Tälläkin kieliluokalla on automaattikarakterisointi (todistus sivuutetaan):

Lause 5.3 *Formaali kieli L on kontekstinen, jos ja vain jos se voidaan tunnistaa epädeterministisellä Turingin koneella, joka ei tarvitse enempää työtilaa kuin syötejonon pituuden verran — siis koneella, jolla ei ole muotoa $\delta(q, <) = (q', b, \Delta)$ olevia siirtymiä, missä $b \neq$ '<'.*

□

Lauseen 5.3 kone saa kirjoittaa syötejonon päälle muita merkkejä; ainoastaan lisätilan käyttöönotto on kiellettyä. Tällaista konetta sanotaan *lineaarisesti rajoitetuksi automaatiksi* (engl. linear bounded automaton). Mielenkiintoinen, mutta luultavasti hyvin vaikea avoin ongelma on, onko em. karakterisoinnissa välttämätöntä käyttää epädeterministisiä koneita, vai riittäisivätkö deterministiset. Tämä "LBA = DLBA"-ongelma on läheisessä yhteydessä luvussa 7.6 esiteltävään kuuluisaan "P = NP"-ongelmaan.

Kieliopit ja niillä tuotettavat kielet ryhmitellään usein ns. *Chomskyn luokkiin* seuraavasti:

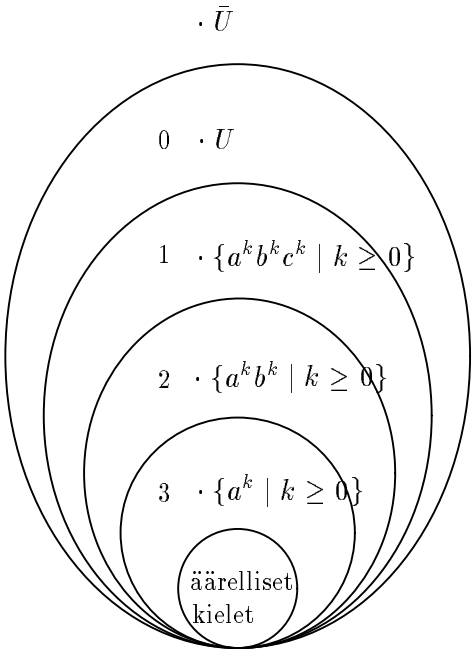
Luokka 0: rajoittamattomat kieliopit / rajoittamattomilla kieliopeilla tuotettavat kielet (ns. rekursiivisesti lueteltavat kielet, ks. luku 6.1);

luokka 1: kontekstiset kieliopit / kontekstiset kielet;

luokka 2: kontekstittomat kieliopit / kontekstittomat kielet;

luokka 3: oikealle ja vasemmalle lineaariset (säännölliset) kieliopit / säännölliset kielet.

Kuvassa 5.2 on esitetty kaavio Chomskyn kielihierarkiasta. Kaavioon on merkitty näkyviin esimerkkejä kielistä, jotka erottavat hierarkian peräkkäisiä tasoja toisistaan. Tasot 0 ja 1 erottava kieli U määritellään tuonnempana, luvussa 6.4. Samassa luvussa osoitetaan, että kielen U komplementti $\bar{U}$ sijaitsee tyystin Chomskyn hierarkian ulkopuolella.



Kuva 5.2: Chomskyn kieliluokat.

Luku 6

Laskettavuusteoriaa

6.1 Rekursiiviset ja rekursiivisesti lueteltavat kielet

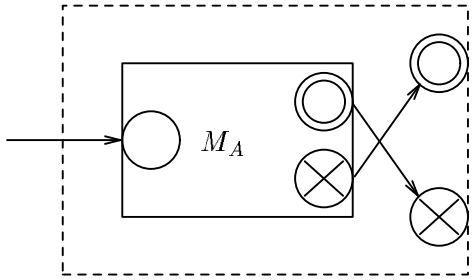
Churchin–Turingin teesin mukaan siis Turingin koneet ovat periaatteelliselta laskentakyvyltään yhtä vahvoja kuin mitkä tahansa mekaaniset laskulaitteet — erityisesti yhtä vahvoja kuin kaikki nykyiset tietokoneet. Seuraavassa tullaan kuitenkin osoittamaan, että Turingin koneiden laskentakyvyllä on vakavia rajoituksia: monet luonnolliset ja mielenkiintoiset laskennalliset ongelmat ovat *algoritmisesti ratkeamattomia*.

Erityisesti tarkastellaan, mitä rajoituksia seuraa siitä, että Turingin koneen vaaditaan pysähtyvän kaikilla syötteillä. Tämä kaikilla ohjelmointikursseilla korostettava kelvollisten algoritmien perusvaatimus ("ohjelma ei saa joutua ikuiseen silmukkaan") osoittautuu teoreettisesti yllättävän hankalaksi.

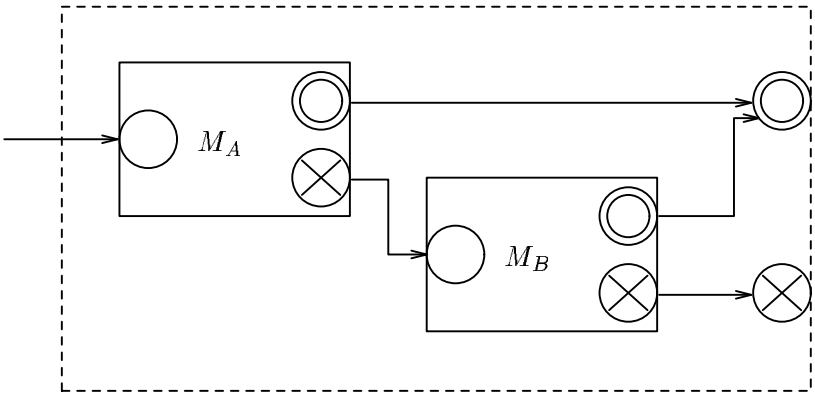
Määritelmä 6.1 Turingin kone $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ on *totaalinen*, jos se pysähtyy kaikilla syötteillä. Formaali kieli A on *rekursiivisesti lueteltava* (engl. recursively enumerable), jos se voidaan tunnistaa jollakin Turingin koneella, ja *rekursiivinen* (engl. recursive), jos se voidaan tunnistaa jollakin totaalisella Turingin koneella[1].

Palautetaan mieliin luvusta 1.1, että formaaleja kieliä voidaan tarkastella myös päätösongelmien esityksinä: tiettyä päätösongelmaa vastaavaan kieleen kuuluvat täsmälleen niiden ongelman tapausten koodit, joihin vastaus on "kyllä". Kielen tunnistava Turingin kone on tällöin samalla vastaavan päätösongelman ratkaisualgoritmi: annetulla syötteellä kone päätyy tilaan q_{yes}, jos vastaus syötteen esittämään ongelman tapaukseen on "kyllä", ja päätyy tilaan q_{no} tai jää pysähtymättä, jos vastaus on "ei". Päätösongelmaa sanotaan *ratkeavaksi* (engl. decidable, solvable), jos sitä vastaava formaali kieli on rekursiivinen, ja *osittain ratkeavaksi* (engl. semidecidable), jos sitä vastaava formaali kieli on rekursiivisesti lueteltava. Ongelma on siis ratkeava, jos sillä on kelvollinen, aina pysähtyvä ratkaisualgoritmi, ja osittain ratkeava, jos sillä on ratkaisualgoritmi, joka "kyllä"-tapauksissa vastaa aina oikein, mutta "ei"-tapauksissa

[1] Merkillisen tuntuisiin nimityksiin "rekursiivinen" ja "rekursiivisesti lueteltava" on historialliset syyt. Ensimmäinen mekaanisen laskennan formalisointi olivat nimittäin Gödelin ja Kleenen "rekursiiviset", so. tietynlaisilla rekursiokaavoilla määritellyt kokonaislukufunktiot. Tätä formalismia käyttäen voitaisiin määritellä, että kokonaislukujoukko on rekursiivinen, jos sen karakteristinen funktio on rekursiivinen funktio, ja rekursiivisesti lueteltava, jos se on tyhjä tai jonkin rekursiivisen funktion kuvajoukko. Samaistamalla formaalit kielet merkkijonojen kanonisen järjestyksen (ks. sivu 3) välityksellä kokonaislukujoukkoihin saadaan tässä määritellyt käsitteet (vrt. lause 6.15).



Kuva 6.1: Rekursiivisen kielen komplementin tunnistaminen Turingin koneella.



Kuva 6.2: Kahden rekursiivisen kielen yhdisteen tunnistaminen Turingin koneella.

voi jäädä pysähtymättä. Ongelma, joka ei ole ratkeava, on *ratkeamaton* (engl. undecidable, unsolvable). (*Huom.*: ratkeamaton ongelma voi siis olla osittain ratkeava.)

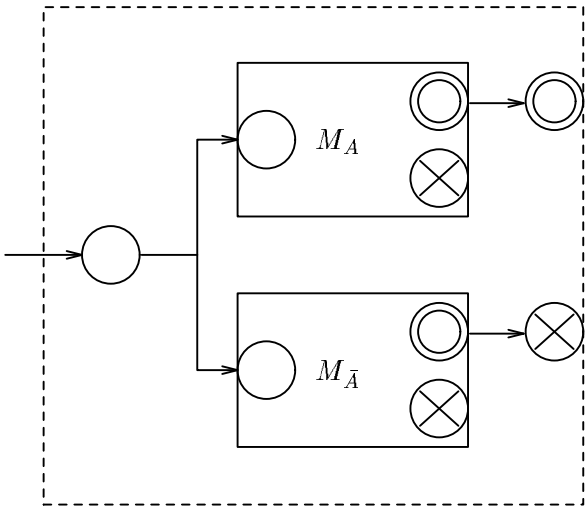
6.2 Rekursiivisten ja rekursiivisesti lueteltavien kielten perusominaisuuksia

Lause 6.1 *Olkoot $A, B \subseteq \Sigma^*$ rekursiivisia kieliä. Tällöin myös kielet $\bar{A} = \Sigma^* - A$, $A \cup B$ ja $A \cap B$ ovat rekursiivisia.*

Todistus.

(i) *$\bar{A}$ on rekursiivinen.* Olkoon M_A totaalinen Turingin kone, joka tunnistaa kielen A. Kielen $\bar{A}$ tunnistava totaalinen Turingin kone saadaan vaihtamalla M_A:n hyväksyvä ja hylkäävä lopputila keskenään kuvan 6.1 esittämällä tavalla. (*Huom.* : Samaa konstruktiota ei voida käyttää osoittamaan, että rekursiivisesti lueteltavien kielten luokka olisi suljettu komplementoinnin suhteen. HT: Miksi ei?)

(ii) *$A \cup B$ on rekursiivinen.* Olkoot M_A ja M_B totaaliset Turingin koneet kielten A ja B tunnistamiseen. Kielen $A \cup B$ tunnistava totaalinen Turingin kone saadaan yhdistämällä nämä kuvan 6.2 konstruktiolla. Toisin sanoen: jos kone M_A hyväksyy syötteen, myös yhdistetty kone hyväksyy; mutta jos M_A päätyy hylkäämiseen, kokeillaan vielä



Kuva 6.3: Totaalisen Turingin koneen muodostaminen kahdesta rinnakkain toimivasta koneesta.

konetta M_B. Jälkimmäistä mahdollisuutta varten koneen M_A tulee toimia siten, että se pysähtyessään jättää nauhalle alkuperäisen syötteen ja täyttää lopun käyttämästään nauhanosasta jollakin sopivasti valitulla tyhjämerkillä. Tämä ei merkitse oleellista rajoitusta M_A:n toimintaan.

(iii) *$A \cap B$ on rekursiivinen.* Väite seuraa edellisistä kohdista ja siitä, että $A \cap B = \overline{\bar{A} \cup \bar{B}}$. □

Lause 6.2 *Olkoot $A, B \subseteq \Sigma^*$ rekursiivisesti lueteltavia kieliä. Tällöin myös kielet $A \cup B$ ja $A \cap B$ ovat rekursiivisesti lueteltavia.*

Todistus. HT. □

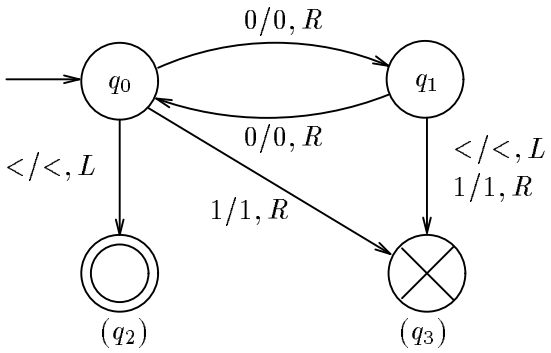
Lause 6.3 *Kieli $A \subseteq \Sigma^*$ on rekursiivinen, jos ja vain jos kielet A ja $\bar{A}$ ovat rekursiivisesti lueteltavia.*

Todistus. Väite vasemmalta oikealle seuraa lauseesta 6.1(i), joten riittää tarkastella väitettä oikealta vasemmalle.

Olkoot M_A ja $M_{\bar{A}}$ Turingin koneet kielten A ja $\bar{A}$ tunnistamiseen. Koska kaikilla $x \in \Sigma^*$ *joko M_A tai $M_{\bar{A}}$ pysähtyy* ja hyväksyy x:n, voidaan niistä yhdistämällä muodostaa kielelle A totaalinen tunnistajakone M kuvan 6.3 esittämällä tavalla.

Kone M voidaan toteuttaa helpoimmin kaksinauhaisena mallina, joka rinnakkaisesti simuloi ykkösnauhallaan konetta M_A ja kakkosnauhallaan konetta $M_{\bar{A}}$. Jos ykkössimulaatio pysähtyy hyväksyvään lopputilaan, M hyväksyy syötteen; jos taas kakkossimulaatio hyväksyy, M hylkää syötteen. □

Seuraus 6.4 *Olkoon $A \subseteq \Sigma^*$ rekursiivisesti lueteltava kieli, joka ei ole rekursiivinen. Tällöin kieli $\bar{A}$ ei ole rekursiivisesti lueteltava.* □



Kuva 6.4: Kielen $\{0^{2k} \mid k \geq 0\}$ tunnistava Turingin kone.

6.3 Turingin koneiden kooodaus ja eräs ei rekursiivisesti lueteltava kieli

Tarkastellaan standardimallisia Turingin koneita, joiden syöteaakkosto on $\Sigma = \{0, 1\}$. Jokainen tällainen kone

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$$

voidaan, mahdollisesti tiloja ja nauha-aakkoston merkkejä uudelleen nimeämällä, esittää binäärijonona seuraavasti:

- Oletetaan, että $Q = \{q_0, q_1, \ldots, q_n\}$, missä $q_{\text{yes}} = q_{n-1}$ ja $q_{\text{no}} = q_n$; ja että $\Gamma \cup \{>, <\} = \{a_0, a_1, \ldots, a_m\}$, missä $a_0 = 0$, $a_1 = 1$, $a_2 = >$ ja $a_3 = <$. Merkitään lisäksi $\Delta_0 = L$ ja $\Delta_1 = R$.

- Siirtymäfunktion δ arvot koodataan seuraavasti: säännön

  $$\delta(q_i, a_j) = (q_r, a_s, \Delta_t)$$

  koodi on

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

- Koko koneen M koodi on

  $$c_M = 111c_{00}11c_{01}11\ldots11c_{0m}11c_{10}11\ldots11c_{1m}11\ldots11c_{n-2,0}11\ldots11c_{n-2,m}111.$$

Esimerkiksi kuvan 6.4 kielen $\{0^{2k} \mid k \geq 0\}$ tunnistavan koneen koodi tätä koodausta käyttäen olisi:

$$c_M = 111\underbrace{01010010100}_{\delta(q_0,0)=(q_1,0,R)}11\underbrace{0100100001001001}_{\delta(q_0,1)=(q_3,1,R)}11\ldots$$

Jokaisella jonkin aakkoston $\{0, 1\}$ kielen tunnistavalla standardimallisella Turingin koneella M on siis binäärikoodi ("konekieliesitys") c_M. Kääntäen voidaan jokaiseen binäärijonoon c liittää jokin Turingin kone. Tosin kaikki binäärijonot eivät ole edellisen koodauksen mukaisia Turingin koneiden koodeja, mutta näihin epäkelpoihin jonoihin voidaan kaikkiin sopia liitettäväksi jokin triviaali, kaikki syötteet hylkäävä kone M_{triv}. Määritellään siis:

$$M_c = \begin{cases} \text{kone } M, \text{ jolla } c_M = c, \text{ jos } c \text{ on kelvollinen Turingin koneen koodi;} \\ \text{kone } M_{\text{triv}}, \text{ muuten.} \end{cases}$$

Näin on saatu aikaan käyttökelpoinen luettelo kaikista aakkoston $\{0,1\}$ Turingin koneista (tilojen ja nauhamerkkien uudelleennimeämistä vaille), ja epäsuorasti myös kaikista aakkoston $\{0,1\}$ rekursiivisesti lueteltavista kielistä. Koneet ovat M_λ, M_0, M_1, M_{00}, M_{01}, ..., ja kielet vastaavasti $L(M_\lambda)$, $L(M_0)$, $L(M_1)$, ... (indeksit kanonisessa järjestyksessä). Kukin kieli voi esiintyä luettelossa monta kertaa.

Cantorilaisella "diagonalisointitekniikalla" (vrt. lause 1.2) pystytään nyt todistamaan ensimmäinen konkreettisen ongelman ratkeamattomuustulos:

Lemma 6.5 *Kieli*

$$D = \{c \in \{0,1\}^* \mid c \notin L(M_c)\}$$

ei ole rekursiivisesti lueteltava.

Todistus. Oletetaan, että olisi $D = L(M)$ jollakin standardimallisella Turingin koneella M. Olkoon d koneen M binäärikoodi, so. $D = L(M_d)$. Tällöin on

$$d \in D \quad \Leftrightarrow \quad d \notin L(M_d) = D.$$

Ristiriidasta seuraa, että kieli D ei voi olla rekursiivisesti lueteltava. □

Kieltä D vastaava päätösongelma on hyvin määritelty, mutta ei kovin luonnollinen: "Hylkääkö annetun koodin c esittämä Turingin kone syötteen c?" Luontevampia esimerkkejä seuraa jatkossa.

Kuvallisesti voidaan kielen D muodostaminen esittää samaan tapaan kuin lauseen 1.2 päätösongelman $\hat{\pi}$: jos kielten $L(M_\lambda)$, $L(M_0)$, $L(M_1)$, ... karakteristiset funktiot esitetään taulukkona, niin kieli D poikkeaa kustakin kielestä taulukon "diagonaalilla"[2]:

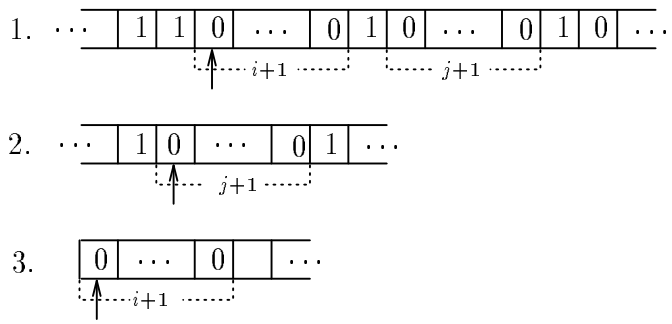
D $\searrow$	$L(M_\lambda)$	$L(M_0)$	$L(M_1)$	$L(M_{00})$	$\cdots$
λ	$\not{0}^{1}$	0	0	0	$\cdots$
0	0	$\not{1}^{0}$	1	0	$\cdots$
1	0	0	$\not{1}^{0}$	1	$\cdots$
00	0	0	0	$\not{0}^{1}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

6.4 Universaalikieli U ja universaalit Turingin koneet

Tarkastellaan seuraavaa aakkoston $\{0,1\}$ *universaalikieltä* U:

$$U = \{c_M w \mid w \in L(M)\}.$$

[2] Kuva on tosin tässä hieman harhaanjohtava siinä suhteessa, että koska mikään koodeista λ, 0, 1, 00 ei ole valitun koodauksen mukainen kelvollinen Turingin koneen koodi, pitäisi oikeastaan kaikkien näkyvissä olevien taulukon alkioiden olla nollia.



Kuva 6.5: Universaalikoneen M_U nauhat.

Kieli U sisältää yksinkertaisella tavalla koodattuina tiedot kaikista aakkoston $\{0,1\}$ rekursiivisesti lueteltavista kielistä. Olkoon nimittäin $A \subseteq \{0,1\}^*$ jokin rekursiivisesti lueteltava kieli, ja olkoon M kielen A tunnistava standardimallinen Turingin kone. Tällöin on

$$A = \{w \in \{0,1\}^* \mid c_M w \in U\}.$$

Ehkä hieman yllättäen kieli U on itsekin rekursiivisesti lueteltava. Kielen U tunnistavia Turingin koneita sanotaan *universaaleiksi Turingin koneiksi* (engl. universal Turing machines); seuraavassa todistuksessa hahmotellaan yhden tällaisen koneen konstruktio.

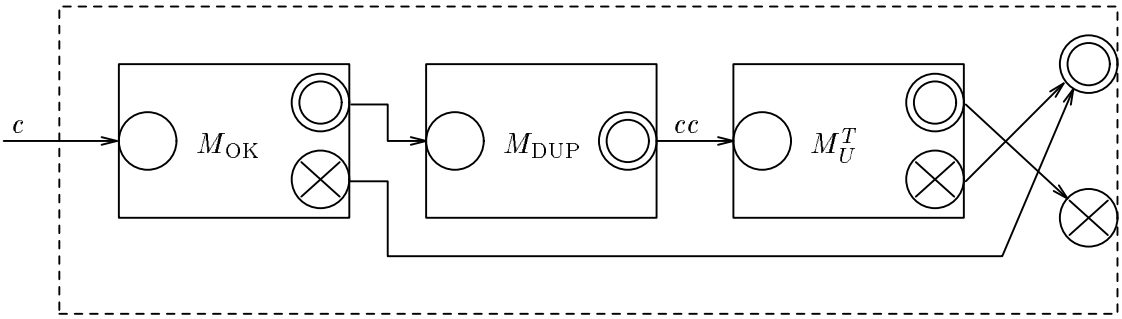
Lause 6.6 *Kieli U on rekursiivisesti lueteltava.*

Todistus. Kielen U tunnistava kone M_U on helpointa kuvata kolmenauhaisena mallina; standardimallinen kone voidaan muodostaa tästä luvun 4.2 tekniikoilla. Laskennan aluksi tarkastettava syöte sijoitetaan koneen M_U ykkösnauhan alkuun. Tämän jälkeen kone toimii seuraavasti:

1. Aluksi M_U tarkastaa, että syöte on muotoa cw, missä c on kelvollinen Turingin koneen koodi. Jos syöte ei ole kelvollista muotoa, M_U hylkää sen; muuten se kopioi merkkijonon $w = a_1 a_2 \ldots a_k \in \{0,1\}^*$ kakkosnauhalle muodossa $00010^{a_1+1}10^{a_2+1}1 \ldots 10^{a_k+1}10000$.

2. Jos syöte on muotoa cw, missä $c = c_M$ jollakin koneella M, M_U:n on selvitettävä, hyväksyisikö kone M syötteen w. Tässä tarkoituksessa M_U säilyttää ykkösnauhalla M:n kuvausta c, kakkosnauhalla simuloi M:n nauhaa, ja kolmosnauhalla säilyttää tietoa M:n simuloidusta tilasta muodossa $q_i \sim 0^{i+1}$ (aluksi siis M_U kirjoittaa kolmosnauhalle tilan q_0 koodin 0).

3. Alkutoimien jälkeen M_U toimii vaiheittain, simuloiden kussakin vaiheessa yhden koneen M siirtymän. Vaiheen aluksi M_U etsii ykkösnauhalta M:n kuvauksesta kohdan, joka vastaa M:n simuloitua tilaa q_i ja merkkiä a_j (kuva 6.5). Olkoon ykkösnauhalla oleva koodinkohta $0^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1}$. Tällöin M_U korvaa kolmosnauhalla merkkijonon 0^{i+1} merkkijonolla 0^{r+1}, kakkosnauhalla merkkijonon 0^{j+1} merkkijonolla 0^{s+1} (mahdollisesti siirtäen nauhojen loppupäiden sisältöjä vasemmalle tai oikealle), ja siirtää kakkosnauhan nauhapäätä yhden simuloidun merkin vasemmalle, jos $t = 0$, ja oikealle, jos $t = 1$.



Kuva 6.6: Diagonaalikielen D tunnistava kone M_D.

Jos ykkösnauhalla ei ole yhtään simuloituun tilaan q_i liittyvää koodia, simuloitu kone M on tullut hyväksyvään tai hylkäävään lopputilaan; tällöin $i = k+1$ tai $i = k+2$, missä q_k on viimeinen ykkösnauhalla kuvattu tila. Kone M_U siirtyy vastaavasti lopputilaan q_{yes} tai q_{no}. □

Lause 6.7 *Kieli U ei ole rekursiivinen.*

Todistus. Oletetaan, että kielellä U olisi totaalinen tunnistajakone M_U^T. Tällöin voitaisiin lemman 6.5 kielelle D muodostaa totaalinen tunnistajakone M_D seuraavasti. Olkoon M_{OK} totaalinen Turingin kone, joka testaa, onko syötteenä annettu merkkijono kelvollinen Turingin koneen koodi, ja olkoon M_{DUP} totaalinen Turingin kone, joka muuntaa syötejonon c muotoon cc. Kone M_D muodostetaan koneista M_U^T, M_{OK} ja M_{DUP} yhdistämällä kuvan 6.6 esittämällä tavalla.

Selvästi kuvan 6.6 esittämä kone M_D on totaalinen, jos kone M_U^T on, ja

$$\begin{aligned} c \in L(M_D) &\Leftrightarrow c \notin L(M_{\mathrm{OK}}) \text{ tai } cc \notin L(M_U^T) \\ &\Leftrightarrow c \notin L(M_c) \\ &\Leftrightarrow c \in D. \end{aligned}$$

Mutta lemman 6.5 mukaan kieli D ei ole rekursiivinen (ei edes rekursiivisesti lueteltava). Saadusta ristiriidasta päätellään, että kielen M_U tunnistavaa totaalista konetta M_U^T ei voi olla olemassa. □

Seuraus 6.8 *Kieli*

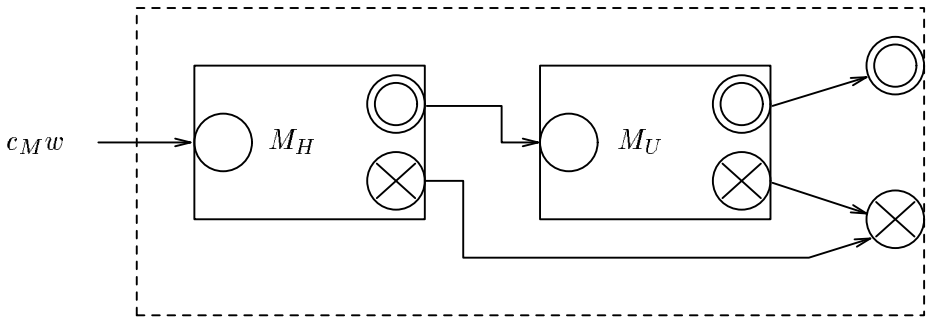
$$\tilde{U} = \{c_M w \mid w \notin L(M)\}$$

ei ole rekursiivisesti lueteltava.

Todistus. Kieli $\tilde{U}$ on oleellisesti sama kuin universaalikielen U komplementti $\bar{U}$; tarkasti ottaen on $\bar{U} = \tilde{U} \cup \mathrm{ERR}$, missä ERR on helposti tunnistettava rekursiivinen kieli

$$\mathrm{ERR} = \{x \in \{0,1\}^* \mid x \text{ ei sisällä alkuosanaan kelvollista Turingin koneen koodia}\}.$$

Jos siis kieli $\tilde{U}$ olisi rekursiivisesti lueteltava, olisi samoin lauseen 6.2 nojalla myös kieli $\bar{U}$. Koska kieli U tiedetään rekursiivisesti lueteltavaksi (lause 6.6), seuraisi tästä lauseen 6.3 nojalla, että U on peräti rekursiivinen. Mutta tämä on vastoin lauseen 6.7 tulosta, mistä päätellään, että kieli $\tilde{U}$ ei voi olla rekursiivisesti lueteltava. □



Kuva 6.7: Universaalikielen U tunnistaminen koneen M_H avulla.

6.5 Turingin koneiden pysähtymisongelma

Oleellisin vaikeus universaalikielen U tunnistamisessa piilee kysymyksessä, *pysähtyykö* annettu Turingin kone M syötteellä w. Seuraava todistus tämän tärkeän *Turingin koneiden pysähtymisongelman* (engl. Turing machine halting problem) ratkeamattomuudelle osoittaa, että jos tämä ongelma voitaisiin ratkaista, myös universaalikieli voitaisiin helposti tunnistaa totaalisesti.

Lause 6.9 *Kieli*

$$H = \{c_M w \mid M \text{ pysähtyy syötteellä } w\}$$

on rekursiivisesti lueteltava, mutta ei rekursiivinen.

Todistus. Todetaan ensin, että kieli H on rekursiivisesti lueteltava. Lauseen 6.6 todistuksessa esitetystä universaalikoneesta M_U on helppo muokata kone, joka syötteellä $c_M w$ simuloi koneen M laskentaa syötteellä w ja pysähtyy hyväksyvään lopputilaan, jos ja vain jos simuloitu laskenta ylipäätään pysähtyy.

Osoitetaan sitten, että kieli H ei ole rekursiivinen. Oletetaan nimittäin, että olisi $H = L(M_H)$ jollakin totaalisella Turingin koneella M_H. Oletetaan lisäksi (tämä ei merkitse lisärajoitusta), että kone M_H pysähtyessään jättää nauhalle alkuperäisen syötteensä, mahdollisesti tyhjämerkeillä # jatkettuna. Olkoon M_U lauseen 6.6 todistuksessa konstruoitu universaalikone. Kielelle U voitaisiin nyt muodostaa totaalinen tunnistaja yhdistämällä koneet M_H ja M_U kuvan 6.7 esittämällä tavalla. Lauseen 6.7 mukaan tällaista kielen U tunnistajakonetta ei kuitenkaan voi olla olemassa. Saatu ristiriita osoittaa, että H ei voi olla rekursiivinen. □

Seuraus 6.10 *Kieli*

$$\tilde{H} = \{c_M w \mid M \text{ ei pysähdy syötteellä } x\}$$

ei ole rekursiivisesti lueteltava.

Todistus. Päätellään kuten seurauslauseessa 6.8. □

6.6 Ratkeamattomuustuloksia Pascalilla

Turingin koneisiin liittyvät käsitteet vastaavat tavanomaisia ohjelmointikäsitteitä seuraavasti:

Turingin kone -formalismi	$\sim$	ohjelmointikieli
Turingin kone	$\sim$	ohjelma
Turingin koneen koodi	$\sim$	ohjelman konekieliesitys
Universaalikone	$\sim$	konekielen tulkki

Koska mikä tahansa Turingin kone voidaan toteuttaa Pascal-ohjelmana ja kääntäen[3], siirtyvät Turingin koneita koskevat ratkeavuus- ja ratkeamattomuustulokset välittömästi koskemaan myös Pascal-ohjelmia. Esimerkiksi pysähtymisongelman Pascal-tulkinta on: "Ei ole olemassa totaalista Pascal-ohjelmaa, joka ratkaisisi, pysähtyykö annettu Pascal-ohjelma P annetulla syötteellä w".

Jotkin ratkeamattomuustulokset voidaan todistaa kätevästi myös suoraan Pascal-formalismissa. Tarkastellaan esimerkkinä pysähtymisongelmaa: oletetaan, että voitaisiin kirjoittaa totaalinen Pascal-funktio

function $H(p, w : \mathit{text}) : \mathit{Boolean}$,

joka saa arvon **true**, jos merkkijonoparametrin p esittämä proseduuri pysähtyy syötteellä w, ja **false** muuten. Kirjoitetaan tämän perusteella toinen Pascal-proseduuri $\hat{H}$:

```
procedure Ĥ(p : text);
    function H(p, w : text) : Boolean;
    begin {Funktion H runko}
        ...
    end;
    begin {Pääohjelma}
        if H(p, p) then while true do;
    end.
```

Merkitään edellä kuvattua proseduurin $\hat{H}$ ohjelmatekstiä $\hat{h}$:lla ja tarkastellaan proseduurin $\hat{H}$ laskentaa tällä omalla kuvauksellaaan. Saadaan ristiriita:

$$\hat{H}(\hat{h}) \text{ pysähtyy} \quad \Leftrightarrow \quad H(\hat{h}, \hat{h}) = \textbf{false} \quad \Leftrightarrow \quad \hat{H}(\hat{h}) \text{ ei pysähdy.}$$

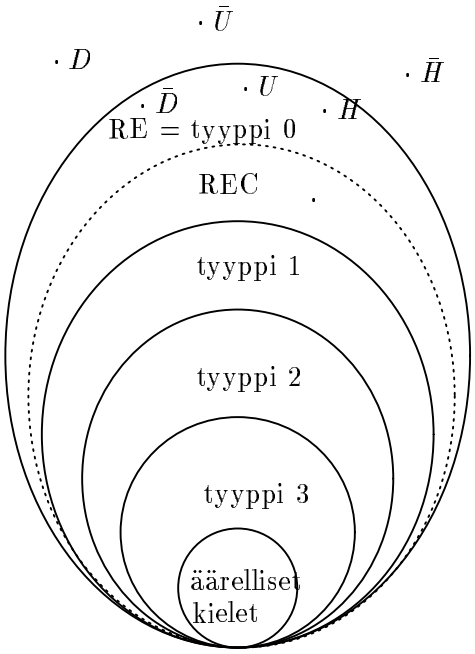
Ristiriidasta seuraa, että oletettua totaalista pysähtymisentestausohjelmaa H ei voi olla olemassa.

6.7 Rekursiiviset kielet ja Chomskyn kieliluokat

Merkitään rekursiivisesti lueteltavien kielten luokkaa RE:llä ja rekursiivisten kielten luokkaa REC:llä. Lauseiden 5.1 ja 5.2 mukaan on siis

$$\text{RE} = \text{rajoittamattomilla kieliopeilla tuotettavat kielet} = \text{Chomskyn luokka 0}.$$

Toisaalta voidaan osoittaa, että kaikki kontekstiset kielet (Chomskyn luokka 1) ovat rekursiivisia, ja että on olemassa rekursiivisia kieliä, jotka eivät ole kontekstisia — tosin on sangen vaikea löytää *luonnollista* esimerkkiä kielestä, joka kuuluisi näiden luokkien erotukseen.



Kuva 6.8: Chomskyn kielihierarkia ja rekursiiviset kielet.

Kuvassa 6.8 on esitetty kaavio Chomskyn kielihierarkiasta täydennettynä rekursiivisten kielten luokalla.

6.8 Lisää ratkeamattomia ongelmia

Valitettavan monet tietojenkäsittelyongelmat ovat ratkeamattomia. Seuraavassa osoitetaan, että esimerkiksi jokseenkin kaikki ohjelmien toimintaa, tai tarkemmin sanoen niiden laskemia syöte/tulos-kuvauksia koskevat kysymykset ovat ratkeamattomia. Johdantona tähän yleiseen tulokseen, ns. Ricen lauseeseen, tarkastellaan ensin yhtä sen erikoistapausta.

Epätyhjyysongelma

Tarkastellaan päätösongelmaa "Hyväksyykö annettu Turingin kone yhtään syötemerkkijonoa?" Ongelman esitys formaalina kielenä on

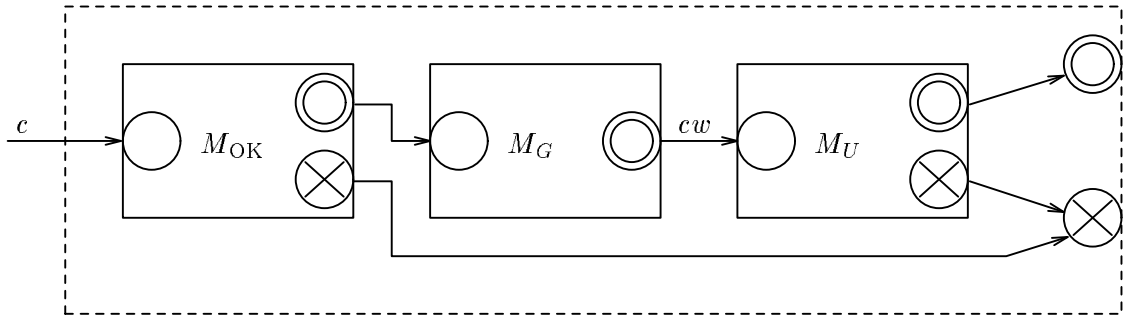
$$\mathrm{NE} = \{c \in \{0,1\}^* \mid L(M_c) \neq \emptyset\}$$

(NE $\sim$ engl. nonempty). Osoittautuu, että tämäkin ongelma on ratkeamaton.

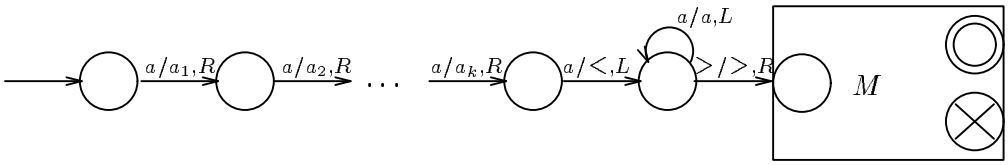
Lause 6.11 *Kieli NE on rekursiivisesti lueteltava, mutta ei rekursiivinen.*

Todistus. Todetaan ensin, että kieli NE on rekursiivisesti lueteltava muodostamalla sille tunnistajakone M_{NE}. Kone M_{NE} on helpointa suunnitella epädeterministisenä; se voidaan tarvittaessa determinisoida lauseen 4.3 mukaisesti.

[3] Tarkemmin sanoen: voidaan laatia sekä Pascal-kielinen Turingin kone -tulkki että Turingin kone, joka kykenee simuloimaan mielivaltaisia Pascal-ohjelmia.



Kuva 6.9: Turingin koneen M_{NE} rakenne.



Kuva 6.10: Turingin koneen M^w rakenne.

Olkoon M_{OK} jo lauseen 6.7 todistuksessa esiintynyt Turingin kone, joka testaa onko annettu syöte kelvollinen Turingin koneen koodi, ja olkoon M_G epädeterminististinen Turingin kone, joka kirjoittaa nauhalla jo olevan merkkijonon perään mielivaltaisen binäärijonon w. Kone M_{NE} voidaan muodostaa yhdistämällä koneet M_{OK}, M_G ja universaalikone M_U (lauseesta 6.6) kuvan 6.9 esittämällä tavalla.

Selvästi on:

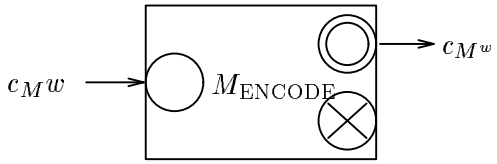
$$\begin{aligned} c \in L(M_{\mathrm{NE}}) &\Leftrightarrow c \text{ on kelvollinen Turingin koneen koodi ja on olemassa } w \text{ s.e. } cw \in U \\ &\Leftrightarrow c \text{ on kelvollinen Turingin koneen koodi ja on olemassa } w \text{ s.e. } w \in L(M_c) \\ &\Leftrightarrow L(M_c) \neq \emptyset. \end{aligned}$$

Osoitetaan sitten, että kieli NE ei ole rekursiivinen. Tämä nähdään olettamalla, että kielellä NE olisi totaalinen tunnistajakone M^T_{NE}, ja muodostamalla tämän perusteella totaalinen tunnistajakone M^T_U kielelle U. Koska U ei ole rekursiivinen (lause 6.7), tämä on mahdottomuus ja osoittaa, että konetta M^T_{NE} ei voi olla olemassa.

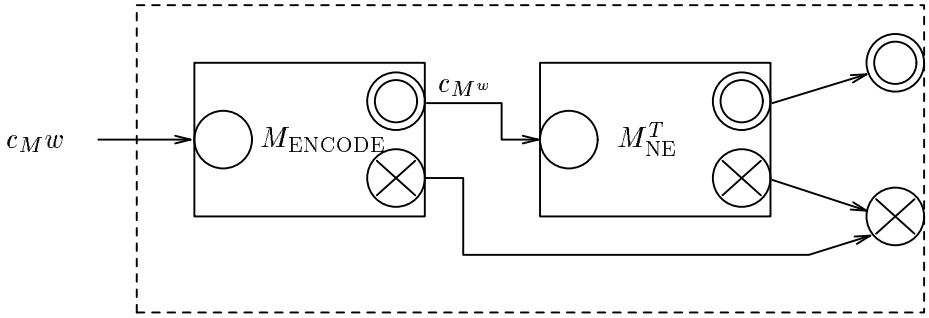
Koneen M^T_U konstruointi koneesta M^T_{NE} perustuu eräänlaiseen syötteiden koodaamiseen Turingin koneiden "ohjelmavakioiksi". Olkoon M mielivaltainen Turingin kone, jonka toimintaa syötteellä w halutaan tutkia. Merkitään M^w:llä konetta, joka kulloisestakin todellisesta syötteestään riippumatta korvaa sen merkkijonolla w ja toimii sitten kuten M. Kuvassa 6.10 on esitetty koneen M^w rakennekaavio, kun $w = a_1 a_2 \ldots a_k$; kuvassa on lyhyyden vuoksi merkitty a:lla "mitä tahansa merkkiä" joukosta $\{0, 1, <\}$.

Koska koneen M^w toiminta ei riipu lainkaan sen todellisesta syötteestä, se joko hyväksyy tai hylkää kaikki merkkijonot, sen mukaan miten M suhtautuu w:hen:

$$L(M^w) = \begin{cases} \{0,1\}^*, & \text{jos } w \in L(M); \\ \emptyset, & \text{jos } w \notin L(M). \end{cases}$$



Kuva 6.11: Turingin koneen M_{ENCODE} toiminta.



Kuva 6.12: Turingin koneen M_U^T rakenne.

Olkoon sitten M_{ENCODE} Turingin kone, joka saa syötteenään mielivaltaisen Turingin koneen M koodista c_M ja binäärijonosta w muodostuvan jonon $c_M w$ ja jättää tuloksenaan nauhalle edellä kuvatun koneen M^w koodin c_{M^w}. (Jos syöte ei ole muotoa cw, missä c on kelvollinen Turingin koneen koodi, kone M_{ENCODE} päätyy hylkäävään lopputilaan.) Kone M_{ENCODE} operoi siis Turingin koneiden *koodeilla* kuvan 6.11 esittämällä tavalla. Annetun koneen M koodiin se lisää siirtymäviisikoita ("konekäskyjä") ja muuttaa tilojen numerointia siten, että koodi tulee koneen M sijaan esittämään konetta M^w.

Universaalikielelle U voitaisiin nyt koneet M_{ENCODE} ja hypoteettinen M^T_{NE} kuvan 6.12 esittämällä tavalla yhdistämällä muodostaa totaalinen tunnistajakone M_U^T. Selvästi kone M_U^T on totaalinen, jos M^T_{NE} on, ja $L(M_U^T) = U$, koska:

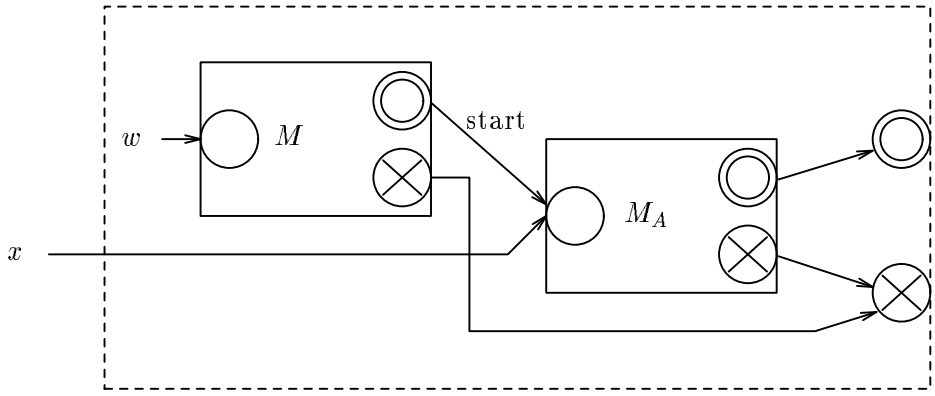
$$\begin{aligned} & c_M w \in L(M_U^T) \\ \Leftrightarrow\ & c_{M^w} \in L(M^T_{\text{NE}}) = \text{NE} \\ \Leftrightarrow\ & L(M^w) \neq \emptyset \\ \Leftrightarrow\ & w \in L(M). \end{aligned}$$

Mutta kieli U ei ole rekursiivinen, joten tällainen totaalinen tunnistajakone M_U^T ei ole mahdollinen. Saadusta ristiriidasta päätellään, että myöskään kielellä NE ei voi olla totaalista tunnistajaa M^T_{NE}. □

Ricen lause

Edellisen lauseen todistustekniikkaa hieman yleistämällä voidaan todistaa seuraavassa esitettävä erittäin vahva ratkeamattomuustulos, ns. *Ricen lause*.

Sanotaan Turingin koneen M *semanttiseksi ominaisuudeksi* mitä tahansa sellaista ominaisuutta, joka riippuu vain koneen M tunnistamasta kielestä, ei sen syntaktisesta rakenteesta.



Kuva 6.13: Turingin koneen M^w rakenne (Ricen lause).

Esimerkkejä semanttisista ominaisuuksista ovat siten "M hyväksyy tyhjän syötejonon", "M hyväksyy jonkin syötejonon" (kielen NE kuvaama ominaisuus), "M hyväksyy äärettömän monta merkkijonoa", "M:n tunnistama kieli on säännöllinen" jne. Jos kahdella Turingin koneella M_1 ja M_2 on $L(M_1) = L(M_2)$, niin koneilla M_1 ja M_2 on tämän määritelmän mukaan täsmälleen samat semanttiset ominaisuudet.

Määritellään hieman abstraktimmin, että *semanttinen ominaisuus* $\mathcal{S}$ on mikä tahansa kokoelma rekursiivisesti lueteltavia aakkoston $\{0,1\}$ kieliä; koneella M *on ominaisuus* $\mathcal{S}$, jos $L(M) \in \mathcal{S}$. *Triviaalit ominaisuudet* ovat $\mathcal{S} = \emptyset$ (ominaisuus, jota ei ole millään koneella) ja $\mathcal{S} = RE$ (ominaisuus, joka on kaikilla koneilla).

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

$$\text{codes}(\mathcal{S}) = \{c \mid L(M_c) \in \mathcal{S}\}$$

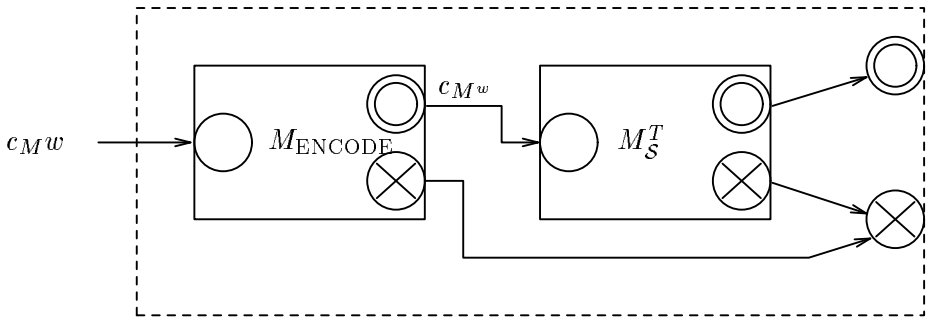
on rekursiivinen. Toisin sanoen: ominaisuus on ratkeava, jos annetusta Turingin koneen koodista voidaan algoritmisesti päätellä, onko koneella kysytty semanttinen ominaisuus.

Lause 6.12 (Rice) *Kaikki Turingin koneiden epätriviaalit semanttiset ominaisuudet ovat ratkeamattomia.*

∗ *Todistus.* Olkoon $\mathcal{S}$ mielivaltainen epätriviaali semanttinen ominaisuus. Voidaan olettaa, että $\emptyset \notin \mathcal{S}$: toisin sanoen, että tyhjän joukon tunnistavilla Turingin koneilla ei ole tarkasteltavaa ominaisuutta. Tämä ei merkitse oleellista rajoitusta, sillä jos $\emptyset \in \mathcal{S}$, voidaan seuraavaa todistusta käyttäen osoittaa, että ominaisuus $\bar{\mathcal{S}} = \text{RE} - \mathcal{S}$ on ratkeamaton, ja päätellä edelleen tästä lauseen 6.1(i) perusteella, että myös ominaisuus $\mathcal{S}$ on ratkeamaton. Koska $\mathcal{S}$ on epätriviaali, on olemassa jokin Turingin kone M_A, jolla on ominaisuus $\mathcal{S}$ — jolla siis $L(M_A) \neq \emptyset \in \mathcal{S}$.

Olkoon tällä kertaa M_{ENCODE} Turingin kone, joka muodostaa syötteenä annetusta, muotoa $c_M w$ olevasta merkkijonosta, seuraavanlaisen Turingin koneen M^w koodin (jos syöte ei ole vaadittua muotoa, M_{ENCODE} päätyy hylkäävään lopputilaan):

> Syötteellä x kone M^w toimii ensin kuten M syötteellä w. Jos M hyväksyy w:n, M^w toimii kuten kone M_A syötteellä x. Jos M hylkää w:n, myös M^w hylkää



Kuva 6.14: Turingin koneen M_U^T rakenne (Ricen lause).

x:n (kuva 6.13). Koneen M^w tunnistama kieli on siten:

$$L(M^w) = \begin{cases} L(M_A), & \text{jos } w \in L(M); \\ \emptyset, & \text{jos } w \notin L(M). \end{cases}$$

Koska oletuksen mukaan $L(M_A) \in \mathcal{S}$ ja $\emptyset \notin \mathcal{S}$, on koneella M^w ominaisuus $\mathcal{S}$, jos ja vain jos $w \in L(M)$.

Oletetaan sitten, että ominaisuus $\mathcal{S}$ olisi ratkeava, so. että kielellä codes($\mathcal{S}$) olisi totaalinen tunnistajakone $M_{\mathcal{S}}^T$. Tällöin saataisiin edellisen todistuksen tapaan totaalinen tunnistajakone kielelle U yhdistämällä koneet M_{ENCODE} ja $M_{\mathcal{S}}^T$ kuvan 6.14 mukaisesti. Selvästi kone M_U^T on totaalinen, jos $M_{\mathcal{S}}^T$ on, ja

$$\begin{aligned} & c_M w \in L(M_U^T) \\ \Leftrightarrow \quad & c_{M^w} \in L(M_{\mathcal{S}}^T) = \text{codes}(\mathcal{S}) \\ \Leftrightarrow \quad & L(M^w) \in \mathcal{S} \\ \Leftrightarrow \quad & w \in L(M). \end{aligned}$$

Koska kieli U ei ole rekursiivinen, tämä on mahdotonta, mistä päätellään että ominaisuus $\mathcal{S}$ ei voi olla ratkeava. □

6.9 Ratkeamattomuustuloksia muilla aloilla

Ricen lauseen mukaan siis jokseenkin kaikki Turingin koneiden — tai ekvivalentisti Pascal-ohjelmien — semanttiset ominaisuudet ovat ratkeamattomia. Ratkeamattomia ongelmia esiintyy runsaasti muuallakin kuin ohjelmien toiminnan analyysin yhteydessä. Seuraavassa joitakin esimerkkejä, ilman todistuksia.

Lause 6.13 (Predikaattikalkyylin ratkeamattomuus; Church/Turing 1936) *Ei ole olemassa algoritmia, joka ratkaisisi, onko annettu ensimmäisen kertaluvun predikaattikalkyylin kaava ϕ validi ("loogisesti tosi", todistuva predikaattikalkyylin aksioomista).*

Lause 6.14 ("Hilbertin 10. ongelma"; Matijasevitsh/Davis/Robinson/Putnam 1953–70) *Ei ole olemassa algoritmia, joka ratkaisisi, onko annetulla kokonaislukukertoimisella polynomilla $P(x_1, \ldots, x_n)$ kokonaislukunollakohtia (so. sellaisia jonoja $(m_1, \ldots, m_n) \in \mathbf{Z}^n$, joilla $P(m_1, \ldots, m_n) = 0$). Ongelma on ratkematon jo, kun $n = 15$ tai deg$(P) = 4$.*

Lauseen 6.14 seurauksena ei yleisemminkään voi olla olemassa algoritmia, joka annetusta kokonaislukuaritmetiikan väitteestä ratkaisisi, onko se totta vai ei. Tässä yleisessä muodossa aritmetiikan ongelmien ratkemattomuus on ollut tunnettua jo Gödelin, Churchin ja Turingin 1930-luvun töistä lähtien. (Mainittakoon kuitenkin, että sellaisten aritmetiikan kaavojen, joissa esiintyy vain yhteenlaskuja ja suuruusvertailuja, ei kertolaskuja — ns. Presburger-aritmetiikan — totuusongelma on ratkeava.)

Formaalien kielten teoriassa esiintyy runsaasti ratkeamattomia ongelmia. Seuraavassa on pieni taulukko tyypillisten kielioppiongelmien ratkeavuudesta, kun annettuna on kieliopit G ja G' Chomskyn hierarkian tietyltä tasolta i ja merkkijono w. Taulukossa $R \sim$ "ratkeava", $E \sim$ "ei ratkeava", $T \sim$ "aina totta".

	Taso i:			
Ongelma: onko	3	2	1	0
$w \in L(G)$?	R	R	R	E
$L(G) = \emptyset$?	R	R	E	E
$L(G) = \Sigma^*$?	R	E	E	E
$L(G) = L(G')$?	R	E	E	E
$L(G) \subseteq L(G')$?	R	E	E	E
$L(G) \cap L(G') = \emptyset$?	R	E	E	E
$L(G)$ säännollinen?	T	E	E	E
$L(G) \cap L(G')$ tyyppiä i?	T	E	T	T
$\overline{L(G)}$ tyyppiä i?	T	E	T	E

6.10 Rekursiiviset funktiot

Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ mielivaltainen standardimallinen Turingin kone. Määritellään koneen M *laskema osittaiskuvaus* (t. *-funktio*)

$$f_M : \Sigma^* \to \Gamma^*$$

seuraavasti:

$$f_M(x) = \begin{cases} u, \text{ jos } (q_0, \underline{x}) \underset{M}{\vdash^*} (q, u\underline{a}v) \text{ jollakin } q \in \{q_{\text{yes}}, q_{\text{no}}\}, av \in \Gamma^*; \\ \text{määrittelemätön, muuten.} \end{cases}$$

Toisin sanoen: kone M kuvaa merkkijonon $x \in \Sigma^*$ sille merkkijonolle $u \in \Gamma^*$, joka sijaitsee koneen nauhapään vasemmalla puolen M:n laskennan syötteellä x pysähtyessä. Jos laskenta syötteellä x ei pysähdy, kuvauksen arvoa pisteessä x ei ole määritelty.

Osittaisfunktio $f : \Sigma^* \to A$ on *osittaisrekursiivinen* (engl. partial recursive), jos se voidaan laskea jollakin Turingin koneella, so. jos on $f = f_M$ jollakin $M = (Q, \Sigma, \Gamma, \ldots)$, missä $A \subseteq \Gamma^*$. Osittaisrekursiivifunktio f on *(kokonais-)rekursiivinen*, jos se voidaan laskea jollakin totaalisella Turingin koneella. Ekvivalentisti voitaisiin määritellä, että osittaisrekursiivifunktio f on rekursiivinen, jos sen arvo $f(x)$ on määritelty kaikilla x.

Lause 6.15

(i) Kieli $A \subseteq \Sigma^*$ on rekursiivinen, jos ja vain jos sen karakteristinen funktio

$$\chi_A : \Sigma^* \to \{0,1\}, \qquad \chi_A(x) = \begin{cases} 1, & \text{jos } x \in A; \\ 0, & \text{jos } x \notin A \end{cases}$$

on rekursiivinen funktio.

(ii) Kieli $A \subseteq \Sigma^*$ on rekursiivisesti lueteltava, jos ja vain jos on $A = \emptyset$ tai on olemassa rekursiivinen funktio $g : \{0,1\}^* \to \Sigma^*$, jolla

$$A = \{g(x) \mid x \in \{0,1\}^*\}.$$

Todistus. Sivuutetaan (mutta ei vaikea). □

6.11 ∗ Rekursiiviset palautukset ja RE-täydelliset kielet

Yksi laskettavuusteorian, ja seuraavassa luvussa esiteltävän laskennan vaativuusteorian, keskeisiä tehtäviä on ongelmien vaikeusvertailu. Eräs tapa formalisoida idea, että ongelma A on "helpompi" tai "enintään yhtä vaikea" kuin ongelma B (vastaavasti B "vähintään niin vaikea" kuin A) on seuraava.

Olkoot $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$ formaaleja kieliä. Kieli A voidaan *palauttaa rekursiivisesti* (engl. recursively many-one reduces to) kieleen B, merkitään

$$A \leq_m B,$$

jos on olemassa rekursiivinen funktio $f : \Sigma^* \to \Gamma^*$, jolla on ominaisuus:

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

Toisin sanoen: mikä tahansa A:n tapaus x voidaan rekursiivisesti muuntaa B:n tapaukseksi $f(x)$, siten että vastaus kysymykseen "onko x:llä ominaisuus A?" on sama kuin vastaus kysymykseen "onko $f(x)$:llä ominaisuus B?"

Palautusrelaatiolla $\leq_m$ on seuraavat perusominaisuudet:

Lemma 6.16 *Kaikilla kielillä A, B, C on voimassa:*

(i) $A \leq_m A$;

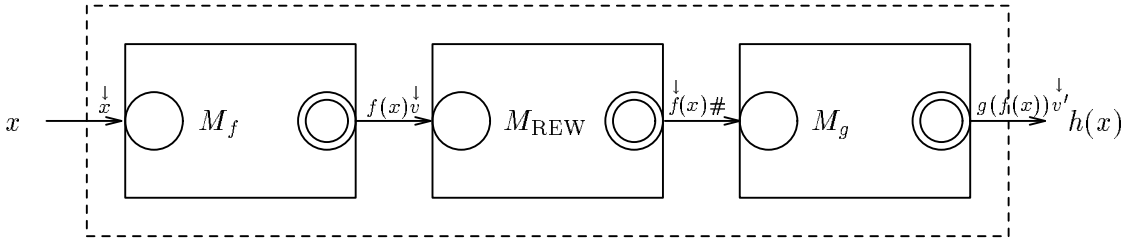
(ii) jos $A \leq_m B$ ja $B \leq_m C$, niin $A \leq_m C$;

(iii) jos $A \leq_m B$ ja B on rekursiivisesti lueteltava, niin A on rekursiivisesti lueteltava;

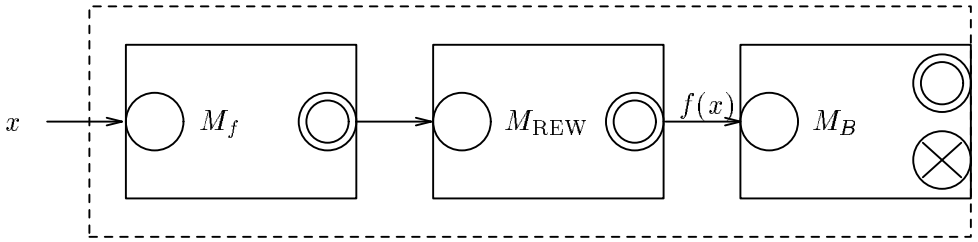
(iv) jos $A \leq_m B$ ja B on rekursiivinen, niin A on rekursiivinen.

Todistus.

(i) Selvä. (Valitaan palautusfunktioksi $f(x) = x$.)



Kuva 6.15: Yhdistetyn kuvauksen laskeva Turingin kone.



Kuva 6.16: Kielen A palautusfunktion f ja kielen B avulla tunnistava Turingin kone.

(ii) Olkoon f palautusfunktio A:sta B:hen ja g palautusfunktio B:stä C:hen. (Lyhyesti voidaan merkitä $f : A \leq_m B$, $g : B \leq_m C$.) Osoitetaan, että yhdistetty funktio h, $h(x) = g(f(x))$, on palautus A:sta C:hen.

Tarkastetaan ensin, että h on rekursiivinen. Olkoon M_f totaalinen Turingin kone, joka laskee f:n ja M_g totaalinen Turingin kone, joka laskee g:n. Oletetaan kone M_g sellaiseksi, että se tyhjämerkin # havaitessaan toimii samoin kuin loppumerkin $<$ tapauksessa. Olkoon lisäksi M_{REW} Turingin kone, joka korvaa kaikki nauhapäästä oikealle sijaitsevat merkit tyhjämerkeillä ja vie nauhapään nauhan alkuun. Funktio h voidaan tällöin laskea kuvassa 6.15 esitetyn kaltaisella totaalisella Turingin koneella.

Tarkastetaan vielä, että funktio h täyttää palautusehdon:

$$x \in A \quad \Leftrightarrow \quad f(x) \in B \quad \Leftrightarrow \quad g(f(x)) = h(x) \in C.$$

Siten $h : A \leq_m C$.

(iii),(iv) Olkoon $f : A \leq_m B$, M_B kone, joka tunnistaa kielen B, ja M_f kone, joka laskee funktion f. Tällöin kuvassa 6.16 esitetty kone tunnistaa kielen A, ja on lisäksi totaalinen jos M_B on. □

Merkitään:

$$\begin{aligned} \mathrm{RE} &= \{\text{aakkoston } \{0,1\} \text{ rekursiivisesti lueteltavat kielet}\}; \\ \mathrm{REC} &= \{\text{aakkoston } \{0,1\} \text{ rekursiiviset kielet}\}. \end{aligned}$$

Kieli $A \subseteq \{0,1\}^*$ on *RE-täydellinen* (engl. RE-complete), jos

(i) $A \in$ RE ja

(ii) $B \leq_m A$ kaikilla $B \in$ RE.

RE-täydelliset kielet ovat siis "maksimaalisen vaikeita" luokassa RE, tai täsmällisemmin sanoen ne sisältävät "rekursiivisesti koodattuina" tiedot kaikista luokan RE kielistä.

Lause 6.17 *Kieli U on RE-täydellinen.*

Todistus. Lauseen 6.6 perusteella tiedetään, että $U \in$ RE. Tarkastellaan mielivaltaista $B \in$ RE; olkoon $B = L(M_B)$. Tällöin B voidaan palauttaa U:hun funktiolla $f(x) = c_{M_B}x$. Tämä funktio on selvästi rekursiivinen, ja sillä on ominaisuus

$$x \in B = L(M_B) \quad \Leftrightarrow \quad f(x) = c_{M_B}x \in U. \quad \square$$

Lemma 6.18 *Olkoon A RE-täydellinen kieli, $B \in RE$ ja $A \leq_m B$. Tällöin myös kieli B on RE-täydellinen.*

Todistus. HT. $\square$

Lauseen 6.17 ja lemman 6.18 perusteella voidaan todeta, että luvussa 6.8 tarkasteltu kieli NE on myös RE-täydellinen. Se nimittäin kuuluu luokkaan RE, ja seuraava funktio f on palautus $U \leq_m NE$ (vrt. lauseen 6.11 todistus):

$$f(x) = \begin{cases} c_{M^w}, & \text{jos } x = c_M w; \\ \lambda, & \text{jos } x \text{ ei ole vaadittua muotoa.} \end{cases}$$

Itse asiassa Ricen lauseen (lause 6.12) todistus osoittaa, että *kaikki* Turingin koneiden semanttisiin ominaisuuksiin $\mathcal{S}$ liittyvät kielet codes($\mathcal{S}$) ovat RE-täydellisiä. Yleensäkin näyttää jostakin syystä olevan niin, että kaikki "luonnolliset" rekursiivisesti lueteltavat, ei-rekursiiviset kielet ovat RE-täydellisiä. Teoreettisesti voidaan kuitenkin osoittaa seuraava tulos (todistus sivuutetaan):

Lause 6.19 (Post) *Luokassa RE − REC on kieliä, jotka eivät ole RE-täydellisiä.* $\square$

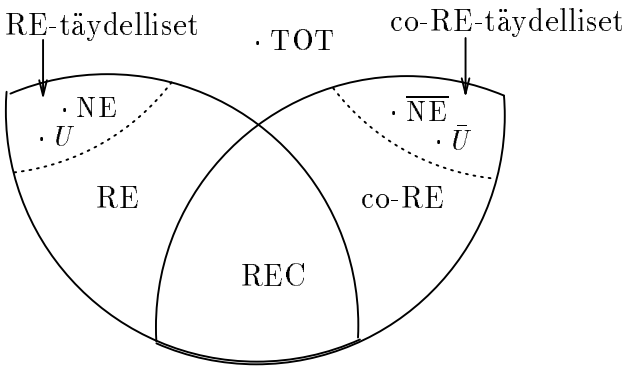
Laskettavuusteorian osuuden päätteeksi esitellään vielä yksi käsite: koska luokka RE ei ole suljettu komplementoinnin suhteen, sillä on luonnollinen duaaliluokka:

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

Lauseen 6.3 perusteella on RE $\cap$ co-RE = REC.

Luokassa co-RE voidaan määritellä täydellisen kielen käsite samoin kuin luokassa RE: kieli $A \subseteq \{0,1\}^*$ on co-RE-täydellinen, jos $A \in$ co-RE ja $B \leq_m A$ kaikilla $B \in$ co-RE. Itse asiassa on helppo todeta, että kieli A on co-RE-täydellinen, jos ja vain jos kieli $\bar{A}$ on RE-täydellinen. Kuvassa 6.17 on esitetty kaavio luokkien RE, co-RE ja REC suhteista.

Mainitaan täydellisyyden vuoksi vielä pari keskeistä laskettavuusteorian tulosta ilman todistuksia.



Kuva 6.17: Kieliluokkien RE, co-RE ja REC suhteet.

Lause 6.20 *Kieli*

$$TOT = \{c \mid \textit{Turingin kone } M_c \textit{ pysähtyy kaikilla syötteillä}\}$$

ei kuulu luokkaan RE eikä luokkaan co-RE. □

Sanotaan, että kielet $A, B \subseteq \{0,1\}^*$ ovat *rekursiivisesti isomorfisia* (engl. recursively isomorphic), jos on olemassa rekursiivinen bijektio $f : \{0,1\}^* \to \{0,1\}^*$ (tällöin myös käänteisfunktio f^{-1} on välttämättä rekursiivinen), jolla

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

Rekursiivisesti isomorfiset kielet ovat siis merkkijonojen rekursiivista uudelleenjärjestämistä lukuunottamatta "samat".

Lause 6.21 (Myhill) *Kaikki RE-täydelliset kielet ovat rekursiivisesti isomorfisia.* □

Luku 7

Laskennan vaativuusteoriaa

7.1 Työläät ongelmat

Laskettavuusteorian tulokset määrittävät absoluuttisia rajoja sille, mitä ongelmia voidaan periaatteessa ratkaista mekaanisesti — ja kuten edellä on havaittu, esimerkiksi monet tärkeät ohjelmien oikeellisuuteen liittyvät kysymykset jäävät näiden rajojen ulkopuolelle. Käytännön tietojenkäsittelyssä tilanne on vieläkin huonompi: laskettavuusteoreettinen "ratkeavuus" nimittäin ei kiinnitä mitään huomiota siihen, kuinka *kauan* ongelman ratkaiseminen kestää.

Tarkastellaan esimerkkinä tunnettua ns. *kauppamatkustajan ongelmaa* (engl. traveling salesman problem, lyh. TSP). Kuvaannollisesti sanoen tässä ongelmassa on tehtävänä löytää kauppamatkustajan avuksi annetun tiekartan perusteella lyhin kaikkien kartan kaupunkien kautta kulkeva reitti. (Yleisemmin sanoen tehtävänä on löytää lyhin ns. Hamiltonin kehä annetusta painotetusta verkosta.) Kauppamatkustajan ongelma esiintyy eri muodoissa lukuisissa käytännön sovelluksissa alkaen jakeluautojen reittisuunnittelusta ja piirilevyjen porauksen optimoinnista moniprosessorijärjestelmien skedulointiin ja geenijonojen automaattiseen sekvenointiin saakka.

Suoraviivainen algoritmi TSP-ongelman ratkaisemiseksi olisi n kaupungin kartalla kokeilla kaikki $n!$ mahdollista "reittiä" ja valita niistä lyhin. Jos kuitenkin tämä algoritmi toteutettaisiin esimerkiksi tietokoneella, jolla yhden reitin tutkiminen kestää millisekunnin, ei maailmankaikkeuden tähänastinen ikä (10–20 miljardia vuotta) aivan riittäisi 22-kaupunkisten karttojen käsittelyyn. Edes koneen vaihto nopeampaan ei tässä tapauksessa juuri auttaisi: vaikka laskentateho biljoonakertaistettaisiin suorittamalla algoritmia rinnakkain miljoonalla supertietokoneella, jotka pystyvät kukin tutkimaan yhden reitin nanosekunnissa, ei maailmankaikkeuden elinaika vielä riittäisi edes 31-kaupunkisiin karttoihin.

Tärkeä kysymys onkin, voidaanko TSP-ongelmalle löytää triviaalia kaikkien mahdollisten ratkaisujen läpikäyntiä tehokkaampaa ratkaisumenetelmää, erityisesti sellaista, jonka ajantarvetta rajoittaisi jokin kaupunkien määrän n polynomi. (Funktio $n!$ ei ole polynomisesti rajoitettu, vaan sen kasvunopeus on suurin piirtein luokkaa n^n.) TSP-ongelmaa on sen tärkeyden takia tutkittu hyvin paljon, ja on kehitetty heuristiikkoja, jotka tekevät mahdolliseksi rutiininomaisesti ratkaista muutaman kymmenen kaupungin suuruisia ongelman tapauksia. Joitakin useiden satojenkin kaupunkien tapauksia on ratkaistu näytösluontoisesti, mutta yleistä menetelmää eksponentiaalisen kaikkien vaihtoehtojen läpikäynnin välttämiseksi ei ole keksitty — toisaalta ei ole myöskään onnistuttu osoittamaan, että tällainen menetelmä olisi mahdoton. *Laskennan vaativuusteoria* (engl. computational complexity theory) tutkii TSP-

ongelman tapaisten laskennallisesti työläiden ongelmien rakennetta, tehokkaiden täsmällisten tai likimääräisten ratkaisualgoritmien olemassaoloa, ja työläiden ongelmien hyväksikäyttöä esimerkiksi todistettavasti luotettavien salakirjoitusjärjestelmien tai satunnaislukugeneraattoreiden suunnittelussa.

7.2 Turingin koneiden aika- ja tilavaativuus

Laskennan vaativuusteorian peruskäsitteiden — laskennan ajan- ja tilantarpeen jne. — määrittelemiseksi täsmällisesti on kiinnitettävä jokin eksakti laskennan malli. Seuraavassa tarkastellaan laskentaa Turingin koneilla, mutta samaan tapaan kuin laskettavuusteoriassa, myös vaativuusteorian peruskäsitteet ovat itse asiassa sangen malliriippumattomia: Turingin koneet voitaisiin korvata millä tahansa "realistisella" laskennan mallilla teorian perustulosten muuttumatta.

Määritelmä 7.1 Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ standardimallinen Turingin kone. Koneen M laskennan

$$(q_0, \underline{x}) \underset{M}{\vdash}^* (q, w), \quad q \in \{q_{\text{yes}}, q_{\text{no}}\},$$

pituus on siihen sisältyvien siirtymäaskelten ($\underset{M}{\vdash}$) määrä.

Koneen M *aika-* ja *tilavaativuus* (engl. time and space complexity) *syötteellä* x määritellään:

$$\text{time}_M(x) = \begin{cases} \text{laskennan } (q_0, \underline{x}) \underset{M}{\vdash}^* \cdots \text{ pituus, jos laskenta pysähtyy;} \\ \infty, \text{ jos laskenta syötteellä } x \text{ ei pysähdy;} \end{cases}$$

$$\begin{aligned} \text{space}_M(x) &= \max\{|w| \mid (q_0, \underline{x}) \underset{M}{\vdash}^* (q, w) \text{ joillakin } q \in Q,\ w \in \Gamma^*\} \\ &(= |w|, \text{ jos laskenta pysähtyy tilanteeseen } (q, w)). \end{aligned}$$

Vastaavat käsitteet voidaan määritellä myös moninauhaisille Turingin koneille; tilantarpeeksi otetaan tällöin maksimi eri nauhojen tilankäytöstä[1].

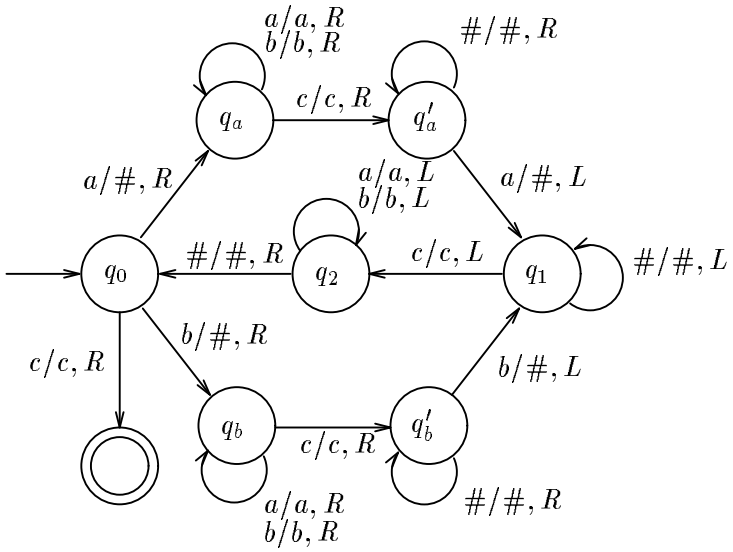
Muodollisesti Turingin koneen M aikavaativuus on siis funktio:

$$\text{time}_M : \Sigma^* \to \mathbb{N} \cup \{\infty\}.$$

Koneiden aikavaativuusfunktiot ovat tyypillisesti hyvin sotkuisia, minkä vuoksi on tapana tiivistää tieto aikavaativuudesta *kunkin pituisilla* syötteillä yhdeksi luvuksi. Tämä voidaan tehdä joko olettamalla n:n merkin mittaisille syötteille jokin todennäköisyysjakauma $P_n(x)$ ja tarkastelemalla M:n *keskimääräistä* aikavaativuutta tämän jakauman suhteen:

$$\text{time}_M^{\text{avg}}(n) = \sum_{|x|=n} P_n(x) \cdot \text{time}_M(x),$$

[1] Tilavaativuudesta puhuttaessa oletetaan usein lisäksi, että koneella on erityinen, vain luettavissa oleva syötenauha, jolla käytettyä tilaa ei lasketa mukaan tilavaativuuteen. Näin päästään käsittelemään myös syötteen pituutta vähäisemmässä työtilassa tapahtuvia laskentoja. Tällainen hienosäätö ei kuitenkaan ole tämän monisteen tavoitteiden kannalta tarpeen.



Kuva 7.1: Kielen $\{wcwz \mid w, z \in \{a, b\}^*\}$ tunnistava Turingin kone.

tai tarkastelemalla yksinkertaisesti M:n aikavaativuutta n merkin mittaisilla syötteillä *pahimmassa tapauksessa*:

$$\mathrm{time}_M^{\mathbf{max}}(n) = \max_{|x|=n} \mathrm{time}_M(x).$$

Keskimääräisen aikavaativuuden analysointi on mielenkiintoista, mutta yleensä hankalaa, joten seuraavassa käsitellään yksinkertaisesti aikavaativuutta pahimmassa tapauksessa. Merkitään (hieman harhaanjohtavasti) myös

$$\mathrm{time}_M^{\mathbf{max}}(n) = \mathrm{time}_M(n).$$

Yhteydestä on aina selvää, tarkoitetaanko merkinnällä time_M tarkkaa aikavaativuusfunktiota

$$\mathrm{time}_M : \Sigma^* \to \mathbb{N} \cup \{\infty\}$$

vai yhteenvetofunktiota

$$\mathrm{time}_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}, \quad \mathrm{time}_M(n) = \max_{|x|=n} \mathrm{time}_M(x).$$

Koneen M tilavaativuus keskimääräisessä ja pahimmassa tapauksessa määritellään vastaavasti funktion space_M pohjalta.

Esimerkkinä Turingin koneiden aikavaativuuden määrittämisestä tarkastellaan kuvan 7.1 esittämää, kielen

$$L = \{wcwz \mid w, z \in \{a, b\}^*\}$$

tunnistavaa Turingin konetta. Esimerkiksi syötteellä $abcab$ koneen laskenta etenee seuraavasti:

$$
\begin{array}{lll}
(q_0, \underline{a}bcab) & \vdash & (q_0, \#\underline{b}c\#b) \quad \vdash \\
(q_a, \#\underline{b}cab) & \vdash & (q_b, \#\#\underline{c}\#b) \quad \vdash \\
(q_a, \#b\underline{c}ab) & \vdash & (q'_b, \#\#c\underline{\#}b) \quad \vdash \\
(q'_a, \#bc\underline{a}b) & \vdash & (q'_b, \#\#c\#\underline{b}) \quad \vdash \\
(q_1, \#b\underline{c}\#b) & \vdash & (q_1, \#\#c\underline{\#}\#) \quad \vdash \\
(q_2, \#\underline{b}c\#b) & \vdash & (q_1, \#\#\underline{c}\#\#) \quad \vdash \\
(q_2, \underline{\#}bc\#b) & \vdash & (q_2, \#\underline{\#}c\#\#) \quad \vdash \\
 & & (q_0, \#\#\underline{c}\#\#) \quad \vdash \\
 & & (q_{\mathrm{yes}}, \#\#c\underline{\#}\#).
\end{array}
$$

Kone tekee tällä syötteellä yhteensä 15 siirtymää, joten $\mathrm{time}_M(abcabc) = 15$.

Yleisemmin voidaan todeta, että parittomalla syötteen pituudella n ovat koneen M kannalta pahimpia tapauksia merkkijonot, jotka ovat muotoa

$$wcw, \quad \text{missä } |w| = m = \frac{n-1}{2}.$$

Tämän muotoisella syötteellä kone tekee m edestakaista "pyyhkäisyä" nauhallaan — yhden kutakin w:n merkkiä kohti — missä kunkin pyyhkäisyn pituus on $2(m+1)+1 = 2m+3$ askelta. Lopuksi M tekee vielä yhden siirtymän tilasta q_0 tilaan q_{yes}. Siten on parittomilla n:

$$
\begin{aligned}
\mathrm{time}_M(n) &= m \cdot (2m+3) + 1 \\
&= 2m^2 + 3m + 1 \\
&= 2\left(\frac{n-1}{2}\right)^2 + 3\left(\frac{n-1}{2}\right) + 1 \\
&= \frac{1}{2}(n^2 - 2n + 1 + 3n - 3) + 1 \\
&= \frac{1}{2}(n^2 + n).
\end{aligned}
$$

Tarkistuksena voidaan todeta, että $\mathrm{time}_M(5) = \frac{1}{2}(25+5) = 15$.

Parillisilla n koneen käyttäytyminen on hieman hankalampi analysoida: tällöin ovat pahimpia tapauksia syötejonot, jotka ovat muotoa

$$wacw \text{ tai } wbcw, \quad \text{missä } |w| = m = \frac{n-2}{2}.$$

Tämän muotoisella syötteellä koneen aikavaativuus on $m(2(m+1)+3)+(m+2) = 2m^2+6m+2$ askelta, mistä saadaan

$$
\begin{aligned}
\mathrm{time}_M(n) &= 2\left(\frac{n-2}{2}\right)^2 + 6\left(\frac{n-2}{2}\right) + 2 \\
&= \frac{1}{2}n^2 + n - 2.
\end{aligned}
$$

Kaikkiaan on siis

$$\mathrm{time}_M(n) = \begin{cases} \frac{1}{2}n^2 + \frac{1}{2}n, & \text{jos } n \text{ on pariton;} \\ \frac{1}{2}n^2 + n - 2, & \text{jos } n \text{ on parillinen.} \end{cases}$$

7.3 Vaativuusfunktioiden kertaluokat

Myös pahimman tapauksen mukaan lasketut aika- ja tilavaativuusfunktiot ovat edelleen liian sotkuisia käsiteltäviksi yksityiskohtaisesti, joten tavallisesti kiinnitetään huomiota vain funktion *kertaluokkaan* (engl. order).

Olkoot $f, g : \mathbb{N} \to \mathbb{R}^+$ mielivaltaisia funktioita. Sanotaan, että f on *kertaluokkaa* $O(g)$, tai vain $f = O(g)$, jos on olemassa vakiot $c, n_0 \in \mathbb{N}$, joilla

$$f(n) \leq c \cdot g(n) \quad \text{kaikilla } n \geq n_0.$$

Toisin sanoen, $f = O(g)$ jos on $f(n) \leq cg(n)$ jollakin c ja melkein kaikilla n.

Jos $f = O(g)$ ja $g = O(f)$, sanotaan että f ja g ovat *samaa kertaluokkaa* ja merkitään $f = \Theta(g)$.

Funktioiden kertaluokkavertailussa käytetään O-merkinnän rinnalla joskus myös vahvempaa o-merkintää. Sanotaan, että f on *alempaa kertaluokkaa* kuin g ja merkitään $f = o(g)$, jos kaikilla vakioilla $c > 0$ on jokin sellainen $n_c \in \mathbb{N}$, että

$$f(n) < c \cdot g(n) \quad \text{kaikilla } n \geq n_c.$$

Selvästi jos $f = o(g)$, niin myös $f = O(g)$, mutta $g \neq O(f)$.

Jos on olemassa sellainen vakio $c > 0$, että äärettömän monella $n \in \mathbb{N}$ on

$$f(n) \geq c \cdot g(n),$$

sanotaan että g on (äärettömän usein) *alaraja* f:lle ja merkitään $f = \Omega(g)$[2].

Funktioille käytetään usein hieman epätäsmällistä merkintää, jossa esiintyy "geneerinen argumentti" n. Sanotaan esimerkiksi, että "funktio $2n^2$ on kertaluokkaa $O(n^2)$", eikä täsmällisemmin "funktio f, jolla $f(n) = 2n^2$, on kertaluokkaa $O(g)$, missä $g(n) = n^2$".

Lemma 7.1

(i) $\log_a n = \Theta(\log_b n)$ *kaikilla* $a, b > 0$;

(ii) $n^a = o(n^b)$, *jos* $a < b$;

(iii) $2^{an} = o(2^{bn})$, *jos* $a < b$;

(iv) $\log_a n = o(n^b)$ *kaikilla* $a, b > 0$;

(v) $n^a = o(2^{bn})$ *kaikilla* $a, b > 0$;

vi) $cf(n) = \Theta(f(n))$ *kaikilla* $c > 0$ *ja funktioilla* f;

(vii) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$ *kaikilla funktioilla* f, g;

(viii) jos $p(n)$ *on aidosti astetta* r *oleva polynomi, niin* $p(n) = \Theta(n^r)$.

Todistus. HT. □

Seuraava yksinkertainen lemma auttaa usein näkemään funktion kertaluokan:

[2]Kirjallisuudessa samaa merkintää käytetään joskus myös vahvemmassa "melkein kaikkialla" alarajan merkityksessä: $f = \Omega(g)$, jos $g = O(f)$

Lemma 7.2 *Olkoot $f, g : \mathbb{N} \to \mathbb{R}^+$ mielivaltaisia funktioita. Jos raja-arvo*

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

on olemassa, äärellinen ja nollasta poikkeava, niin $f = \Theta(g)$. Jos raja-arvo on 0, niin $f = o(g)$, ja jos raja-arvo on ääretön, niin $g = o(f)$.

Todistus. HT. □

7.4 Vaativuusluokat

Olkoot $t, s : \mathbb{N} \to \mathbb{R}^+$ mielivaltaisia funktioita. Sanotaan, että kieli A voidaan *tunnistaa ajassa t (tilassa s)*, jos on olemassa moninauhainen[3] deterministinen Turingin kone M, jolla $L(M) = A$ ja

$$\text{time}_M(n) \leq t(n) \quad \text{kaikilla } n$$

(vastaavasti

$$\text{space}_M(n) \leq s(n) \quad \text{kaikilla } n.)$$

Määritellään seuraavat formaalien kielten (so. päätösongelmien) *vaativuusluokat* (engl. complexity classes):

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

sekä näistä muodostettavat tärkeät yhdisteluokat:

$$\begin{aligned} \text{P} &= \bigcup\{\text{DTIME}(t) \mid t \text{ polynomi }\} = \bigcup_{k\geq 0} \text{DTIME}(n^k + k); \\ \text{PSPACE} &= \bigcup\{\text{DSPACE}(s) \mid s \text{ polynomi }\} = \bigcup_{k\geq 0} \text{DSPACE}(n^k + k); \\ \text{E} &= \bigcup_{k\geq 0} \text{DTIME}(2^{n^k}); \\ \text{ESPACE} &= \bigcup_{k\geq 0} \text{DSPACE}(2^{n^k}). \end{aligned}$$

Esimerkiksi luokkaan P kuuluvat siis ne formaalit kielet (päätösongelmat) A, jotka voidaan tunnistaa (vast. ratkaista) jollakin syötteen pituuden suhteen polynomisessa ajassa toimivalla Turingin koneella. Voidaan osoittaa (tod. siv.; ei vaikea), että jos kieli A voidaan tunnistaa jollakin nykyaikaisella tietokoneella ajassa t ja tilassa s, niin se voidaan tunnistaa jollakin moninauhaisella Turingin koneella ajassa $O(t^2)$ ja tilassa $O(s)$. Siten yhdisteluokkien P, PSPACE, E ja ESPACE määritelmät ovat kohtuullisen riippumattomia siitä, mikä laskennan malli valitaan määrittelyn perustaksi. Erityisesti luokka P sisältää täsmälleen ne ongelmat, jotka voidaan ratkaista nykyaikaisilla tietokoneilla polynomisessa ajassa.

Luokalle P on laskennan vaativuusteoriassa vakiintunut samantapainen asema kuin luokalla REC on laskettavuusteoriassa: ajatellaan, että ongelma on *käytännössä ratkeava*, jos ja vain jos se kuuluu luokkaan P. Tätä käytännössä ratkeavan ongelman käsitteen täsmennystä voidaan tosin kritisoidakin. Seuraavassa tarkastellaan kolmea tärkeintä vastaväitettä:

[3] Vakiintuneen käytännön mukaan näissä määritelmissä käytetään moninauhaisia koneita, koska ne ovat jonkin verran "tehokkaampia" kuin yksinauhaiset. Ero ei kuitenkaan ole oleellinen, kuten tuonnempana todetaan (Lemma 7.4).

1. *Luokka P on liian laaja.* Ongelmaa, jonka ratkaiseminen n:n merkin syötteellä vaatii n^{100} laskenta-askelta, tuskin voidaan pitää käytännössä ratkeavana.

   Kommentti: Vaikka teoriassa voidaankin osoittaa, että tällaisia ongelmia on olemassa (so. että esimerkiksi erotus DTIME(n^{100}) – DTIME(n^{99}) on epätyhjä), niin niitä ei näytä esiintyvän käytännössä. Kaikilla tunnetuilla "luonnollisilla" luokan P ongelmilla on melko matala-asteista polynomista aikavaativuutta olevat ratkaisualgoritmit — algoritmien vaativuus on tyypillisesti enintään luokkaa $O(n^3)$, pahimmillaankin ehkä $O(n^9)$ (mikä tosin on jo aika paljon).

2. *Luokka P on liian suppea.* Käytännöllisinä täytyisi pitää myös sellaisia algoritmeja, joiden ajantarve tosin asymptoottisesti on ylipolynominen, mutta jotka kaikilla realistisen kokoisilla syötteillä toimivat kohtuullisen nopeasti.

   Kommentti: Sellaisia ongelmia tosiaan esiintyy, joissa realistisilla syötteillä on jokin luonnollinen, pieni kokoraja, ja tällöin huomautus on aivan pätevä. Esimerkiksi jos tehtävänä on valita viikon päivistä osajoukko joidenkin kriteerien mukaan, niin hyvin voidaan kokeilla kaikki mahdollisuudet: mahdollisia osajoukkoja tosin on 2^n kappaletta, mutta n on vain 7.

   Jos sen sijaan "realistisen kokoisella syötteellä" tarkoitetaan vain, että syöte mahtuu esimerkiksi nykyaikaisen tietokoneen keskusmuistiin, niin voidaan todeta, että käytännössä ei näytä esiintyvän ylipolynomisia algoritmeja, joiden suoritusaika olisi siedettävä vielä näin suurilla syötteillä.

3. *Pahimman tapauksen analysointi on harhaanjohtavaa.* Ylipolynomistakin algoritmia täytyisi pitää käytännöllisenä, jos sen "pahanlaatuiset" syötteet ovat käytännössä hyvin harvinaisia.

   Kommentti: Tämä on oikeastaan aivan pätevä vastaväite: tunnettu esimerkki teoriassa eksponentiaalisesta algoritmista, joka käytännössä on erittäin tehokas, on ns. lineaarisen ohjelmoinnin simplex-menetelmä. Voidaan kuitenkin todeta, että:

   (a) Tavallisesti kuitenkin teoriassa eksponentiaaliset algoritmit toimivat myös käytännössä huonosti.

   (b) Algoritmin keskimääräisen käyttäytymisen analysointi edellyttää aina jonkin oletuksen käytännössä esiintyvien syötteiden jakaumasta. Tällaiset oletukset ovat usein ongelmallisia: syötetyyppi, joka jossakin sovelluksessa on harvinainen, voi toisessa sovelluksessa olla tavallinen.

   (c) Keskimääräisen käyttäytymisen analysointi on kylläkin mielenkiintoista, mutta yleensä kovin vaikeaa.

7.5 Vaativuusluokkien ominaisuuksia

Seuraavassa käydään läpi edellä määriteltyjen vaativuusluokkien tärkeimmät tunnetut ominaisuudet. Todistukset pääosin sivuutetaan. Ensimmäinen tulos osoittaa, että ainoastaan rajafunktion kertaluokalla on merkitystä vaativuusluokkien määrittelyssä:

Lemma 7.3 *Olkoot $t(n), s(n) \geq n$ mielivaltaisia funktioita, ja lisäksi $t(n)$ ylilineaarinen (so. $n = o(t(n))$). Tällöin on millä tahansa vakiolla $r > 0$:*

(i) $DTIME(rt(n)) \subseteq DTIME(t(n))$;

(ii) $DSPACE(rs(n)) \subseteq DSPACE(s(n))$.

∗ *Todistus.* Ideana tässä on, että mikä tahansa tietyn kielen tunnistava Turingin kone M voidaan korvata Turingin koneella M', jonka nauhamerkit vastaavat noin r:n M:n nauhamerkin "lohkoja". Kone M' pystyy yhdessä siirtymässä käsittelemään kokonaisen tällaisen lohkon, ja toimii siten sekä vähintään r kertaa nopeammin että r kertaa pienemmässä tilassa kuin M. Tarkemmat yksityiskohdat sivuutetaan. □

Seuraava tulos osoittaa, että edellä vaativuusmääritelmien perustaksi valitut moninauhaiset Turingin koneet eivät ole kohtuuttoman paljon tehokkaampia kuin standardimalliset yksinauhaiset koneet. Yksinauhaisia koneita käyttäen voidaan määritellä vaativuusluokat:

$$\begin{aligned} \mathrm{DTIME}_1(t) &= \{A \mid A \text{ voidaan tunnistaa ajassa } t \text{ yksinauhaisella Turingin koneella}\}; \\ \mathrm{DSPACE}_1(s) &= \{A \mid A \text{ voidaan tunnistaa tilassa } s \text{ yksinauhaisella Turingin koneella}\}. \end{aligned}$$

Näiden luokkien suhteesta edellä määriteltyihin on voimassa seuraava tulos:

Lemma 7.4 *Kaikilla $t(n), s(n) \geq n$ on:*

(i) $DTIME(t(n)) \subseteq DTIME_1(t^2(n))$;

(ii) $DSPACE(s(n)) \subseteq DSPACE_1(s(n))$.

∗ *Todistus.* Simuloitaessa moninauhaista konetta yksinauhaisella (lauseiden 4.1 ja 4.2 konstruktiot) koneen aikavaativuus kasvaa enintään neliöllisesti ja tilavaativuus pysyy ennallaan. Yksityiskohdat sivuutetaan. □

Lisätietona mainittakoon, että kaksinauhaisilla koneilla on mahdollista toteuttaa jo paljon tehokkaampi simulointi. Jos merkitään

$$\mathrm{DTIME}_2(t) = \{A \mid A \text{ voidaan tunnistaa ajassa } t \text{ kaksinauhaisella Turingin koneella}\},$$

niin voidaan osoittaa, että kaikilla $t(n) \geq n$ on

$$\mathrm{DTIME}(t(n)) \subseteq \mathrm{DTIME}_2(t(n) \log t(n)).$$

(Todistus sivuutetaan.)

Tarkastellaan sitten aika- ja tilavaativuusluokkien suhdetta:

Lause 7.5 *Kaikilla $t(n), s(n) \geq n$ on:*

(i) $DTIME(t(n)) \subseteq DSPACE(t(n))$;

(ii) $DSPACE(s(n)) \subseteq \bigcup_{k \geq 0} DTIME(k^{s(n)})$.

∗ *Todistus.*

(i) Tulos perustuu siihen yksinkertaiseen havaintoon, että $t(n)$ askelessa Turingin kone ei voi kirjoittaa enempää kuin $t(n)$ merkkiä millekään nauhalleen.

(ii) Idea: Turingin koneella, joka toimii tilassa $s(n)$, on n merkin mittaisella syötteellä enintään $k^{s(n)}$ mahdollista tilannetta, jollakin vakiolla k. Lisäämällä koneeseen "kello", joka varsinaisen laskennan rinnalla laskee yhdestä $k^{s(n)}$:ään ja luvun täytyttyä katkaisee laskennan hylkäävänä, kone saadaan toimimaan aina ajassa $k^{s(n)}$. □

Seuraus 7.6 *$P \subseteq PSPACE \subseteq E \subseteq ESPACE$.* □

Seurauslauseen 7.6 sisältyvyyksistä tiedetään aidoiksi ainoastaan seuraavat:

Lause 7.7

(i) $P \neq E$;

(ii) $PSPACE \neq ESPACE$.

∗ *Todistus.* Tarkastellaan väitettä (i); väitteen (ii) todistus sujuu samaan tapaan.

Olkoon $M_\lambda, M_0, M_1, M_{00}, M_{01}, \ldots$ jokin kaikkien moninauhaisten Turingin koneiden luettelointi, luvussa 6.3 esitetyn standardimallisten koneiden koodauksen tapaan. Määritellään seuraava luokan P "universaalikieli":

$$U^P = \{0^m cx \mid M_c \text{ hyväksyy syötteen } x \text{ ajassa } |x|^{m/\log|x|}\}.$$

Voidaan todeta, että kieli U^P kuuluu luokkaan E, koska sen ratkaisemiseksi, kuuluuko syöte $0^m cx$ joukkoon U^P, riittää simuloida koneen M_c toimintaa syötteellä x $|x|^{m/\log|x|} = 2^m$ askelen verran[4].

Toisaalta ei voi olla $U^P \in$ P. Oletetaan nimittäin, että olisi $U^P \in$ DTIME(n^k) jollakin kiinteällä $k > 1$. Tällöin olisi myös kieli

$$U^k = \{cx \mid M_c \text{ hyväksyy syötteen } x \text{ ajassa } |x|^k\}$$

luokassa DTIME(n^k), samoin kieli

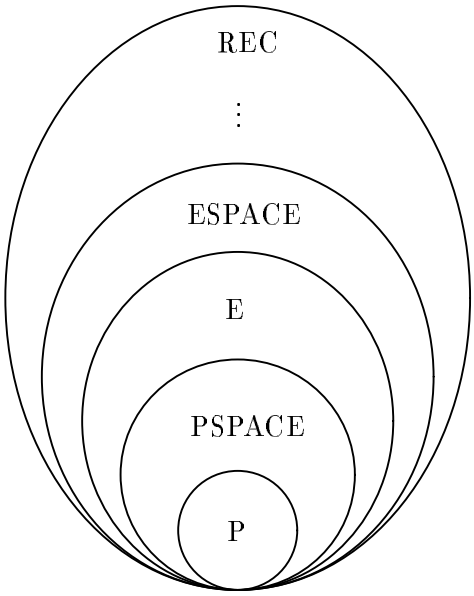
$$D^k = \{c \mid M_c \text{ ei hyväksy syötettä } c \text{ ajassa } |c|^k\}.$$

Mutta tästä seuraa ristiriita: olkoon nimittäin $D^k = L(M_d)$ jollakin sellaisella M_d, jolla $\text{time}_{M_d}(n) \leq n^k$ kaikilla n. Tällöin on:

$$\begin{aligned} & M_d \text{ hyväksyy syötteen } d \text{ ajassa } |d|^k \\ \Leftrightarrow\ & d \in D^k \\ \Leftrightarrow\ & M_d \text{ ei hyväksy syötettä } d \text{ ajassa } |d|^k. \quad \square \end{aligned}$$

Koska P $\subseteq$ PSPACE $\subseteq$ E ja P $\neq$ E, on välttämättä myös joko P $\neq$ PSPACE tai PSPACE $\neq$ E, tai mahdollisesti molemmat. Vastaavasti voidaan päätellä, että joko PSPACE $\neq$ E, E $\neq$ ESPACE, tai mahdollisesti molemmat. Yleisesti arvellaan, että kaikki nämä epäyhtälöt ovat tosia, mutta juuri minkäänlaista käsitystä ei ole siitä, miten tämänkaltaisia väitteitä voitaisiin todistaa. Vaativuusluokkien suhteita koskevat kysymykset kuuluvat yleensäkin tietojenkäsittelyteorian vaikeimpiin ja syvällisimpiin. Tässä puheena olevien kysymysten suhteesta toisiinsa tiedetään, että jos P = PSPACE, niin E = ESPACE; käänteisen implikaation todistaminen olisi huomattava tulos.

[4] Tietojenkäsittelyteoriassa vakiintuneen käytännön mukaan tarkoittaa merkintä $\log n$ tässä kaksikantaista logaritmia. Huomaa, että tällöin on $n^{1/\log n} = 2$.



Kuva 7.2: Deterministisen vaativuusluokkien suhteet.

Samaa universaalikielitekniikkaa käyttäen kuin edellisessäkin lauseessa voidaan todistaa myös, ettei ole olemassa "vaikeinta" rekursiivista kieltä. Sanotaan, että kokonaislukufunktio $t : \mathbb{N} \to \mathbb{N}$ on *rekursiivinen*, jos funktio $t^{\text{bin}} : \{0,1\}^* \to \{0,1\}^*$, joka vastaa t:tä lukujen binääriesityksillä, voidaan laskea totaalisella Turingin koneella.

Lause 7.8

(i) Jos $A \in REC$, niin on olemassa rekursiivinen funktio $t(n)$, jolla $A \in DTIME(t(n))$.

(ii) Kaikilla rekursiivisilla funktioilla $t(n)$ on olemassa kieli $A \in REC - DTIME(t(n))$.

∗ *Todistus.*

(i) Olkoon $A = L(M)$ jollakin totaalisella Turingin koneella M. Funktioksi $t(n)$ voidaan tällöin valita koneen M aikavaativuusfunktio $t(n) = \text{time}_M(n)$. Aikavaativuusfunktio on nimittäin tässä tapauksessa rekursiivinen: se voidaan laskea koneella, joka syötteellä n (oik. $\text{bin}(n) = n$:n binääriesitys) simuloi konetta M kaikilla n:n mittaisilla syötteillä ja pitää kirjaa pisimmän havaitun laskennan pituudesta. Koska M pysähtyy aina, myös aikavaativuusfunktion laskenta pysähtyy.

(ii) Olkoon $t(n)$ rekursiivinen, ylilineaarinen funktio. Määritellään

$$U^t = \{cx \mid M_c \text{ hyväksyy syötteen } x \text{ ajassa } t(|cx|)\}.$$

Samaan tapaan kuin edellä voidaan osoittaa, että $U^t \in \text{REC}$, mutta $U^t \notin \text{DTIME}(t(n))$. □

Tähän asti esitettyjen vaativuusluokkien suhteet ovat siis kuvan 7.2 mukaiset. Hyvän käsityksen luokan REC laajuudesta saa tarkastelemalla esimerkiksi seuraavaa, erittäin nopeasti kasvavaa, mutta kuitenkin rekursiivista funktiota $t(n)$: määritellään $t(n) = t'(n,n)$, missä

$$\begin{aligned} t'(0,n) &= \left.2^{2^{\cdots^2}}\right\} n \text{ kpl} \\ t'(m+1,n) &= \left.2^{2^{\cdots^2}}\right\} t'(m,n) \text{ kpl.} \end{aligned}$$

Funktion kolme ensimmäistä arvoa ovat:

$$t(0) = 1, \quad t(1) = 4, \quad t(2) = \left.2^{2^{\cdots^2}}\right\} 65536 \text{ kpl.}$$

Lauseen 7.8 mukaan luokka REC$-$DTIME$(t(n))$ on epätyhjä: on siis olemassa "periaatteessa" ratkeavia ongelmia, joiden ratkaiseminen n merkin mittaisilla syötteillä vaatii enemmän aikaa kuin funktion $t(n)$ verran.

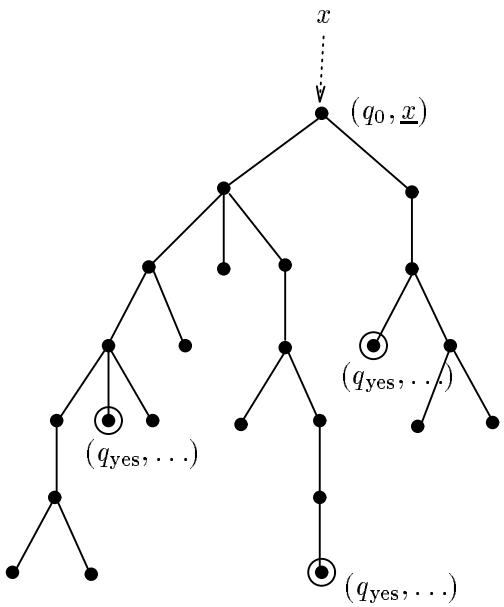
7.6 Epädeterministiset vaativuusluokat

Erittäin tärkeä, monissa sovelluksissa vastaan tuleva tyyppi päätösongelmia, joille ei toistaiseksi tunneta yleisiä polynomisia ratkaisualgoritmeja, on seuraava: halutaan tietää, onko annetulla syötteellä x tietty ominaisuus A. Ominaisuuden A voimassaolo syötteellä x riippuu siitä, onko olemassa toista, suunnilleen x:n pituista merkkijonoa w, ns. *todistetta* (engl. witness), siten että parilla (x,w) on tietty helposti testattava ominaisuus B. Täsmällisemmin sanoen: mahdolliset todistejonot ovat pituudeltaan enintään polynomisia x:n pituuden suhteen, ja "oikeellisuustarkastus" $B(x,w)$ voidaan suorittaa x:n ja w:n pituuden suhteen polynomisessa ajassa. Tällä tavoin määritellyn ominaisuuden A tunnistamisen tekee vaikeaksi se, että vaikka kukin *yksittäinen* todistejono voidaan tarkastaa nopeasti, mahdollisia todistejonoja on *kaikkiaan* eksponentiaalinen määrä, siis liian paljon käytäväksi läpi yksi kerrallaan.

Esimerkiksi luvussa 7.1 tarkasteltu kauppamatkustajan ongelma on, päätösongelmaksi muokattuna, tätä tyyppiä: syötteenä annetaan täydellinen painotettu verkko ("kartta") G ja kokonaisluku k; halutaan tietää, onko verkossa G sen kaikkien solmujen kautta kulkevaa reittiä, jonka pituus on enintään k. Mahdollisia todisteita ovat tässä tapauksessa kaikki verkon kiertävät reitit ja testattava ehto on, onko annetun reitin pituus enintään k. Jos verkossa G on n solmua, niin kukin reitti voidaan esittää yksinkertaisesti lukujen $1,\ldots,n$ permutaationa, ja pituusehdon testaaminen on kunkin yksittäisen reitin kohdalla helppoa. Vaikeaksi ongelman tekee se, että mahdollisia reittejä on $n!$ kappaletta.

Toinen esimerkki on luvussa 4.2 käsitelty yhdistettyjen lukujen tunnistaminen. Luku n on helppo todeta yhdistetyksi yhdellä kertolaskulla, jos sen tekijät p ja q on annettu; ongelmana on se, että mahdollisia tekijöitä on n:n binääriesityksen pituuden suhteen eksponentiaalinen määrä. (Lisää esimerkkejä esitetään tuonnempana, luvussa 7.7.)

Tärkeä havainto on, että kaikki tämäntyyppiset ongelmat voitaisiin ratkaista polynomisessa ajassa *epädeterministisillä Turingin koneilla*: ongelman A ratkaisemiseksi syötteellä x koneen tarvitsee vain ensin epädeterministisesti "arvata" mielivaltainen polynomisen mittainen todistejono w, ja sitten deterministisesti polynomisessa ajassa "tarkistaa", että jonot x ja w yhdessä toteuttavat ehdon B. Tätä ratkaisumenetelmää sovellettiin jo luvussa 4.2



Kuva 7.3: Epädeterministisen Turingin koneen laskentapuu.

yhdistettyjen lukujen tunnistamiseen. (Kyseessä ei tietenkään varsinaisesti ole mikään ratkaisumenetelmä, vaan tietty tapa kuvata tarkasteltavana oleva ongelma. Käytännössä sovelletut yhdistettyjen lukujen tunnistamisalgoritmit eivät vähimmässäkään määrin muistuta luvun 4.2 triviaalia epädeterminististä Turingin konetta.)

Tällaisella epädeterministisellä arvaus/tarkistus-menetelmällä kuvattavissa olevia luonnollisia ongelmia, joita ei osata ratkaista deterministisesti polynomisessa ajassa, tunnetaan nykyisin jo toista tuhatta, ja määrä kasvaa koko ajan. Näiden ongelmien suuren käytännön merkityksen takia on aikarajoitettujen epädeterminististen Turingin koneiden laskentavoiman selvittäminen yksi tietojenkäsittelyteorian keskeisimmistä tehtävistä.

Kysymysten lähempää tarkastelua varten täytyy epädeterministiseen laskentaa liittyvät vaativuuskäsitteet määritellä täsmällisesti. Olkoon siis $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ epädeterministinen yksinauhainen Turingin kone. Määritellään:

$$\begin{aligned} \text{time}_N(x) &= \textit{pisimmän } N\text{:n laskennan } (q_0, \underline{x}) \vdash_M^* \cdots \text{ pituus (voi olla } \infty); \\ \text{space}_N(x) &= \textit{eniten tilaa vievän } N\text{:n laskennan } (q_0, \underline{x}) \vdash_M^* \cdots \text{ käyttämien} \\ &\quad \text{nauhapaikkojen määrä (voi olla } \infty). \end{aligned}$$

Vastaavat käsitteet voidaan määritellä myös moninauhaisille koneille, samoin kuin deterministisessäkin tapauksessa.

Epädeterministisen Turingin koneen mahdollisia laskentoja annetulla syötteellä x on usein hyödyllistä ajatella kuvassa 7.3 esitetyn tapaisena *laskentapuuna*. Laskentapuun solmut vastaavat koneen tilanteita: juurisolmu vastaa koneen alkutilannetta syötteellä x, ja kunkin solmun jälkeläiset esittävät solmun vastintilanteen mahdollisia seuraajia. Puun lehdet vastaavat tilanteita, joissa laskenta pysähtyy. Kone hyväksyy syötteen, jos jonkin lehden vastintilanteessa koneen tila on q_{yes}.

Tämän mallin mukaan ajatellen on siis epädeterministisellä koneella N:

$$\text{time}_N(x) = \text{koneen } N \text{ syötteellä } x \text{ tuottaman laskentapuun korkeus.}$$

(HT: Minkälaisia ovat deterministisen koneen M tuottamat laskentapuut?)

Määritellään edelleen aiempaan tapaan pahimman tapauksen mukaiset aika- ja tilavaativuusfunktiot annetun pituisille syötteille:

$$\begin{aligned} \text{time}_N(n) &= \max_{|x|=n} \text{time}_N(x); \\ \text{space}_N(n) &= \max_{|x|=n} \text{space}_N(x). \end{aligned}$$

Olkoot $t, s : \mathbb{N} \to \mathbb{R}^+$. Sanotaan, että kieli A voidaan *tunnistaa epädeterministisesti ajassa t (tilassa s)*, jos on olemassa moninauhainen epädeterministinen Turingin kone N, jolla $L(N) = A$ ja

$$\text{time}_N(n) \leq t(n) \quad \text{kaikilla } n$$

(vastaavasti

$$\text{space}_N(n) \leq s(n) \quad \text{kaikilla } n).$$

Keskeiset epädeterministiset vaativuusluokat määritellään:

$$\begin{aligned} \text{NTIME}(t) &= \{A \mid A \text{ voidaan tunnistaa epädeterministisesti ajassa } t\}; \\ \text{NSPACE}(s) &= \{A \mid A \text{ voidaan tunnistaa epädeterministisesti tilassa } t\}; \end{aligned}$$

ja yhdisteluokat:

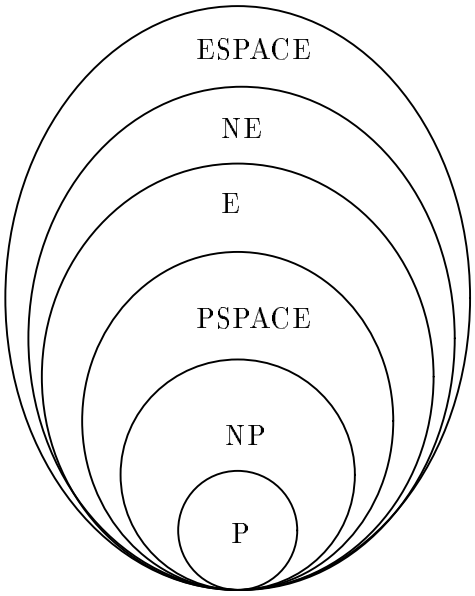
$$\begin{aligned} \text{NP} &= \bigcup\{\text{NTIME}(t) \mid t \text{ polynomi}\} = \bigcup_{k\geq 0} \text{NTIME}(n^k + k); \\ \text{NPSPACE} &= \bigcup\{\text{NSPACE}(s) \mid s \text{ polynomi}\} = \bigcup_{k\geq 0} \text{NSPACE}(n^k + k); \\ \text{NE} &= \bigcup_{k\geq 0} \text{NTIME}(2^{n^k}); \\ \text{NESPACE} &= \bigcup_{k\geq 0} \text{NSPACE}(2^{n^k}). \end{aligned}$$

Luokka NP sisältää siis kaikki luvun alussa esitellyn tyyppiset arvaus/tarkistus-ongelmat[5]. Kysymys siitä, voidaanko kaikki nämä ongelmat ratkaista polynomisessa ajassa toimivilla algoritmeilla, voidaan muotoilla lyhyesti vaativuusluokkien suhteita koskevana kysymyksenä: onko P = NP? Selvästi on P $\subseteq$ NP, koska deterministiset koneet ovat epädeterminististen erikoistapaus, mutta onko tämä sisältyvyys aito?

Päällisin puolin näyttäisi selvältä, että luokassa NP pitäisi olla kieliä, jotka eivät kuulu luokkaan P, koska epädeterministiset koneet ovat paljon "tehokkaampia" kuin deterministiset: ne pystyvät polynomisessa ajassa tutkimaan eksponentiaalisen määrän vaihtoehtoisia laskentoja. Kysymys luokkien P ja NP suhteesta on kuitenkin osoittautunut hämmästyttävän vaikeaksi: intensiivisestä tutkimuksesta huolimatta se on pysynyt avoimena jo yli 20 vuotta, Stephen Cookin vuonna 1971 ilmestyneestä, kysymyksen ensimmäisen kerran täsmällisesti muotoilleesta artikkelista lähtien.

Osoituksena siitä, että P = NP-ongelman ratkaisu ei ole aivan itsestään selvä, mainitaan seuraava hämmästyttävä tulos, ns. *Savitchin lause*:

[5]Itse asiassa voidaan myös kääntäen kaikki polynomisessa ajassa toimivat epädeterministiset Turingin koneet saattaa arvaus/tarkistus-algoritmeja vastaavaan normaalimuotoon. Tätä tietoa ei kuitenkaan tarvita jatkossa.



Kuva 7.4: Vaativuusluokkien sisältyvyyssuhteet.

Lause 7.9 (Savitch) *Olkoon $s(n) \geq n$ tilakonstruoituva funktio*[6]*. Tällöin on*

$$NSPACE(s(n)) \subseteq DSPACE(s^2(n)).$$

Todistus. Sivuutetaan. □

Seuraus 7.10

(i) $NPSPACE = PSPACE$*;*

(ii) $NESPACE = ESPACE$*.* □

Paitsi että Savitchin lause osoittaa epädeterminismin heikkouden tilarajoitetuissa laskennoissa, se auttaa myös järjestämään käsitellyt yhdistevaativuusluokat lineaariseen järjestykseen:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{NPSPACE} = \mathrm{PSPACE} \subseteq \mathrm{E} \subseteq \mathrm{NE} \subseteq \mathrm{NESPACE} = \mathrm{ESPACE}.$$

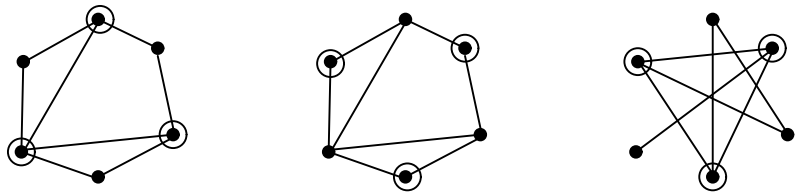
Täydennetty kaavio vaativuusluokkien suhteista on esitetty kuvassa 7.4.

Lauseen 7.7 tekniikalla voidaan osoittaa, että mitkä tahansa toisistaan "eksponentiaalisen etäällä" olevat vaativuusluokat eroavat ($\mathrm{P} \neq \mathrm{E}$, $\mathrm{NP} \neq \mathrm{NE}$, $\mathrm{PSPACE} \neq \mathrm{ESPACE}$). Sen sijaan ei tiedetä, onko $\mathrm{P} \neq \mathrm{NP}$, $\mathrm{NP} \neq \mathrm{PSPACE}$, $\mathrm{PSPACE} \neq \mathrm{E}$ jne. Kaikkien sisältyvyyksien arvellaan vahvasti olevan aitoja, mutta mitään ei ole pystytty todistamaan.

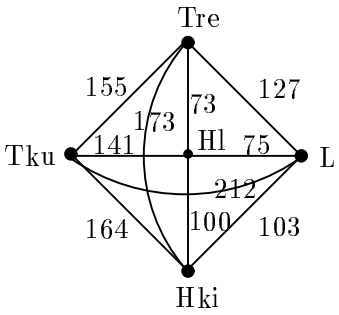
Myöhempää tarvetta varten todetaan vielä lemman 7.4 vastine epädeterministisille koneille. Määritellään yksinauhaisten epädeterminististen Turingin koneiden vaativuusluokat:

$$\begin{aligned} \mathrm{NTIME}_1(t) &= \{A \mid A \text{ voidaan tunnistaa ajassa } t \text{ yksinauhaisella epädet. koneella}\}; \\ \mathrm{NSPACE}_1(s) &= \{A \mid A \text{ voidaan tunnistaa tilassa } s \text{ yksinauhaisella epädet. koneella}\}. \end{aligned}$$

[6] Funktion "tilakonstruoituvuus" on tietty vaativuusteoriassa tavallinen säännöllisyysoletus; käsitteen tarkalla määritelmällä ei ole tässä merkitystä. Kaikilla "yksinkertaisilla" funktioilla, erityisesti kaikilla polynomeilla, on tämä ominaisuus.



Kuva 7.5: Verkko-ongelmien tapauksia: (a) solmupeite, (b) riippumaton joukko, (c) klikki.



Kuva 7.6: Kauppamatkustajan ongelman tapaus.

Lemma 7.11 *Kaikilla $t(n), s(n) \geq n$ on:*

(i) $NTIME(t(n)) \subseteq NTIME_1(t^2(n))$;

(ii) $NSPACE(s(n)) \subseteq NSPACE_1(s(n))$.

Todistus. Samoin kuin lemmassa 7.4. □

7.7 Esimerkkejä luokan NP ongelmista

Edellä on jo mainittu kaksi esimerkkiä luokan NP ongelmista, joille ei tunneta polynomisia ratkaisualgoritmeja: kauppamatkustajan ongelma ja yhdistettyjen lukujen testaus[7]. Luokan NP laajuuden ja vaihtelevuuden osoittamiseksi on seuraavaan koottu joukko käytännössä melko tärkeitä samantyyppisiä ongelmia. Kaikki listan ongelmat on helppo ratkaista polynomisella arvaus/tarkistus-menettelyllä, joten niiden kuuluminen luokkaan NP on selvää. Myöhempää viittaustarvetta varten on myös kauppamatkustajan ongelma otettu uudelleen mukaan listaan.

1. Lausekalkyylin toteutuvuusongelma (lyh. SAT, engl. Satisfiability).

Annettu muuttujista $x_1, \ldots, x_n$, vakioista 0 ("epätosi") ja 1 ("tosi"), sekä konnektiiveista $\wedge$ ("ja"), $\vee$ ("tai") ja $\neg$ ("ei") koostuva lausekalkyylin kaava ("Boolen lausekse")

[7] Jo tässä vaiheessa voi olla hyvä huomauttaa siitä, että yhdistettyjen lukujen testausongelma on tietyssä mielessä epätyypillinen esimerkki luokan NP ongelmasta; ks. tarkemmin lausetta 7.18 seuraava huomautus, sivulla 124.

F. On ratkaistava, onko F *toteutuva*, so. onko sellaista muuttujien totuusarvoasetusta

$$t : \{x_1, \ldots, x_n\} \to \{0, 1\},$$

joka tekee F:stä toden (so. $F(t(x_1), \ldots, t(x_n)) = 1$).

Esimerkiksi kaava $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ on toteutuva, asetuksella $t(x_1) = 1$, $t(x_2) = 0$. Sen sijaan kaavaa $x_1 \wedge \neg x_1$ ei mikään totuusarvoasetus tee todeksi.

Toteutuvuusongelman esitys formaalina kielenä on

$$\text{SAT} = \{F \mid F \text{ on toteutuva lausekalkyylin kaava}\}.$$

Tarvittaessa voidaan ajatella, että kaavat F on koodattu jotakin sopivaa koodausta käyttäen binäärijonoiksi.

Toteutuvuusongelma voidaan ratkaista helposti arvaus/tarkistus-menettelyllä: annetun syötekaavan F toteutuvuuden testaamiseksi tarvitsee vain arvata sopiva totuusarvoasetus t ja tarkistaa, että $F(t) = 1$. Jos kaavassa F esiintyy n muuttujaa, totuusarvoasetus t voidaan esittää n-bittisenä binäärijonona, ja sen toimivuuden tarkastaminen sujuu helposti polynomisessa ajassa.

2. Solmupeiteongelma (lyh. VC, engl. Vertex Cover).

Annettu suuntaamaton verkko G ja luonnollinen luku k. Onko verkossa G enintään k:n solmun muodostamaa *solmupeitettä*, so. solmujoukkoa, joka peittää kustakin verkon kaaresta vähintään toisen pään?

Esimerkiksi kuvan 7.5(a) verkossa muodostavat ympyröidyt kolme solmua solmupeitteen.

Solmupeiteongelman esittäminen formaalina kielenä edellyttää verkkojen koodaamista jollakin tavalla merkkijonoiksi, esimerkiksi luettelemalla verkon yhteysmatriisin alkiot järjestyksessä. Lisäksi tarvitaan jokin tapa esittää merkkijonopareja merkkijonoina: yksinkertaisinta lienee ottaa käyttöön uudet merkit alkusulku ($\langle$), pilkku ja loppusulku ($\rangle$), ja esittää merkkijonojen x ja y muodostama pari merkkijonona $\langle x, y\rangle$. Olettaen näiden koodauskysymysten yksityiskohdat ratkaistuiksi saadaan solmupeiteongelmalle esitys

$$VC = \{\langle G, k\rangle \mid \text{verkossa } G \text{ on enintään } k \text{ solmun solmupeite}\},$$

tai tarkemmin sanoen:

$$VC = \{\langle G, k\rangle \mid \begin{array}{l} G \text{ on jonkin suuntaamattoman verkon esitys, } k \text{ on binääriluku ja} \\ G\text{:n esittämästä verkosta löytyy solmupeite, jossa on } \leq k \text{ solmua}\}. \end{array}$$

Solmupeiteongelman arvaus/tarkistus-ratkaisu on jälleen helppo: syötteellä $\langle G, k\rangle$ arvataan verkosta G k solmua ja tarkistetaan, että ne peittävät kaikki verkon kaaret.

3. Riippumaton joukko -ongelma (lyh. IS, engl. Independent Set)

Annettu suuntaamaton verkko G ja luonnollinen luku k. Onko verkossa G vähintään k *riippumatonta solmua*, so. solmua, joiden välillä ei kulje yhtään kaarta?

Esimerkiksi kuvan 7.5(b) verkossa ovat ympyröidyt kolme solmua riippumattomia.

Riippumaton joukko -ongelman esitys formaalina kielenä on

$$IS = \{\langle G, k\rangle \mid \text{verkossa } G \text{ on vähintään } k \text{ riippumatonta solmua}\}.$$

4. **Klikkiongelma** (lyh. CLIQUE, engl. Clique)

   Annettu suuntaamaton verkko G ja luonnollinen luku k. Onko verkossa G vähintään k:n solmun muodostamaa *klikkiä*, so. solmujoukkoa, jonka kaikkien solmuparien välillä on kaari?

   Esimerkiksi kuvan 7.5(c) verkossa muodostavat ympyröidyt kolme solmua klikin.

   Klikkiongelman esitys formaalina kielenä on

   $$CLIQUE = \{\langle G, k\rangle \mid \text{verkossa } G \text{ on vähintään } k \text{ solmun klikki}\}.$$

5. **Hamiltonin kehä -ongelma** (lyh. HC, engl. Hamiltonian Cycle)

   Annettu suuntaamaton verkko G. Sisältääkö G *Hamiltonin kehän*, so. syklin (suljetun polun), joka kulkee kaikkien verkon solmujen kautta täsmälleen kerran?

6. **Kauppamatkustajan ongelma** (lyh. TSP, engl. Traveling Salesman Problem)

   Annettu täydellinen suuntaamaton painotettu verkko G ("kartta" tai oikeastaan "etäisyystaulukko") ja luonnollinen luku k. Onko verkossa G Hamiltonin kehää, jonka kokonaispaino (so. polulla olevien kaarten painojen summa) on enintään k?

   Esimerkiksi kuvan 7.6 esittämässä verkossa on kaikki viisi solmua kiertävä reitti, jonka pituus on 570. (Mikä?)

7. **Verkonväritysongelma** (lyh. GC, engl. Graph Coloring)

   Annettu suuntaamaton verkko G ja luonnollinen luku k. Voidaanko verkon G solmut "värittää" k:lla "värillä" niin, ettei millekään kahdelle naapurisolmulle (so. kaaren yhdistämälle solmulle) tule samaa väriä?

8. **Ositusongelma** (lyh. PARTITION, engl. Partition)

   Annettu joukko luonnollisia lukuja $\{n_1, \ldots, n_k\}$. Voidaanko näistä valita osajoukko $\{m_1, \ldots, m_r\} \subseteq \{n_1, \ldots, n_k\}$ niin, että $(m_1 + \ldots + m_r) = \frac{1}{2}(n_1 + \ldots + n_k)$?

7.8 Polynomiset palautukset ja NP-täydelliset kielet

Laskennan vaativuusteoriassa on ratkaisevan tärkeä sija laskettavuusteoreettisten palautus- ja täydellisyyskäsitteiden (luku 6.11) polynomisilla vastineilla.

Sanotaan, että funktio $f : \Sigma^* \to \Gamma^*$ voidaan *laskea polynomisessa ajassa*, jos on olemassa deterministinen Turingin kone M ja polynomi p, joilla $f = f_M$ ja $\text{time}_M(n) \leq p(n)$ kaikilla n. Olkoot $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$ formaaleja kieliä. Kieli A voidaan *palauttaa polynomisesti* (engl. polynomial time many-one reduces to) kieleen B, merkitään

$$A \leq_m^p B,$$

jos on olemassa polynomisessa ajassa laskettava funktio $f : \Sigma^* \to \Gamma^*$, jolla on ominaisuus:

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

Palautusrelaatiolla $\leq_m^p$ on seuraavat perusominaisuudet (vrt. lemma 6.16):

Lemma 7.12 *Kaikilla kielillä A, B, C on voimassa:*

(i) $A \leq_m^p A$;

(ii) Jos $A \leq_m^p B$ *ja* $B \leq_m^p C$, *niin* $A \leq_m^p C$;

(iii) Jos $A \leq_m^p B$ *ja* $B \in$ *NP (PSPACE, E, NE, ESPACE), niin* $A \in$ *NP (PSPACE, E, NE, ESPACE);*

(iv) Jos $A \leq_m^p B$ *ja* $B \in P$, *niin* $A \in P$.

Todistus.

(i) Selvä. (Valitaan palautusfunktioksi $f(x) = x$.)

(ii) Olkoon f palautusfunktio A:sta B:hen ja g palautusfunktio B:stä C:hen. (Lyhyesti voidaan merkitä $f : A \leq_m^p B$, $g : B \leq_m^p C$.) Osoitetaan, että yhdistetty funktio h, $h(x) = g(f(x))$, on palautus A:sta C:hen.

Todetaan ensin, että h täyttää palautusehdon:

$$x \in A \quad \Leftrightarrow \quad f(x) \in B \quad \Leftrightarrow \quad g(f(x)) = h(x) \in C.$$

Tarkastetaan sitten, että h voidaan laskea polynomisessa ajassa. Olkoon M_f Turingin kone, joka laskee funktion f polynomin p rajoittamassa ajassa, ja M_g Turingin kone, joka laskee funktion g polynomin q rajoittamassa ajassa. Voidaan olettaa, että polynomit p ja q ovat kaikkialla ei-vähéneviä. Oletetaan kone M_g sellaiseksi, että se tyhjämerkin # havaitessaan toimii samoin kuin loppumerkin $<$ tapauksessa. Olkoon lisäksi M_{REW} Turingin kone, joka korvaa kaikki nauhapäästä oikealle sijaitsevat merkit tyhjämerkeillä ja vie nauhapään nauhan alkuun. Funktio h voidaan tällöin laskea, aivan samoin kuin lemman 6.16(ii) todistuksessa, koneella M_h, joka suorittaa peräkkäin koneiden M_f, M_{REW} ja M_g laskennat. Tällaisen koneen rakenne on esitetty kuvassa 6.15 (s. 95).

Koneen M_h aikavaativuus syötteellä x on:

$$\begin{aligned} \mathrm{time}_{M_h}(x) &= \mathrm{time}_{M_f}(x) + \mathrm{time}_{M_{\mathrm{REW}}}(f(x)v) + \mathrm{time}_{M_g}(f(x)) \\ &\leq p(|x|) + 2p(|x|) + q(|f(x)|) \\ &\leq 3p(|x|) + q(p(|x|)) \\ &= O(q(p(|x|))), \end{aligned}$$

siis polynominen x:n pituuden suhteen. (Laskelman kolmannella rivillä on käytetty hyväksi sitä tietoa, että $|f(x)| \leq \mathrm{time}_{M_f}(x) \leq p(|x|)$.)

(iii) Samaan tapaan kuin seuraava kohta.

(iv) Olkoon M_f Turingin kone, joka laskee palautusfunktion $f : A \leq_m^p B$ polynomin p rajoittamassa ajassa, ja M_B Turingin kone, joka tunnistaa kielen B polynomin q rajoittamassa ajassa. Tällöin kieli A voidaan tunnistaa, lemman 6.16(iv) tapaan, koneella M_A, joka suorittaa peräkkäin koneiden M_f ja M_B laskennat (ks. kuva 6.15, s. 95). Samaan tapaan kuin kohdassa (ii) voidaan todeta, että koneen M_A aikavaativuus syötteellä x on $O(q(p(|x|)))$, siis polynominen x:n pituuden suhteen. □

Olkoon sitten $\mathcal{C}$ mikä tahansa kieliluokka (erityisen kiinnostavia ovat luokat $\mathcal{C} = \mathrm{NP}$, PSPACE jne.). Kieli A on *$\mathcal{C}$-täydellinen (polynomisten palautusten suhteen)* (engl. $\mathcal{C}$-complete w.r.t. polynomial time reductions), jos

(i) $A \in \mathcal{C}$ ja

(ii) $B \leq_m^p A$ kaikilla $B \in \mathcal{C}$.

$\mathcal{C}$-täydelliset kielet ovat siis jossakin mielessä "maksimaalisen vaikeita" luokassa $\mathcal{C}$; erityisesti $\mathcal{C}$-täydellisyys takaa, että kieli voidaan tunnistaa polynomisessa ajassa jos vain jos *kaikki muutkin* luokan $\mathcal{C}$ kielet voidaan tunnistaa polynomisessa ajassa:

Lemma 7.13 *Olkoon $\mathcal{C}$ jokin kieliluokka ja A jokin $\mathcal{C}$-täydellinen kieli. Tällöin $A \in P$ jos ja vain jos $\mathcal{C} \subseteq P$.*

Todistus. Väite seuraa helposti lemman 7.12 kohdasta (iv). □

Luokkaan NP sovellettuna tämä tarkoittaa siis, että P = NP, jos ja vain jos mikä tahansa NP-täydellinen kieli kuuluu luokkaan P: koko kysymys luokkien vertailusta voidaan keskittää yhteen ainoaan kieleen — kunhan vain jokin kieli pystytään ensin osoittamaan NP-täydelliseksi. Tämän takia on seuraava, luvussa 7.9 todistettava ns. *Cookin lause* ratkaisevan tärkeä:

Lause (Cook). *Kieli*

$$SAT = \{F \mid F \text{ on toteutuva lausekalkyylin kaava}\}$$

on NP-täydellinen.

Cookin lauseen todistus, joka nojaa suoraan käsitteiden määritelmiin, ei ole aivan helppo. Mutta kun yksi kieli on näin todistettu NP-täydelliseksi, on muut NP-täydellisyystodistukset mahdollista tehdä huomattavasti yksinkertaisemmin, seuraavan lemman perusteella:

Lemma 7.14 *Olkoon $\mathcal{C}$ jokin kieliluokka ja A jokin $\mathcal{C}$-täydellinen kieli. Jos tällöin $B \in \mathcal{C}$ ja $A \leq_m^p B$, niin myös B on $\mathcal{C}$-täydellinen.*

Todistus. HT. □

Luokkaan NP sovellettuna: jos halutaan osoittaa jokin uusi luokan NP kieli B NP-täydelliseksi, riittää muodostaa polynominen palautus jostakin jo tunnetusta NP-täydellisestä kielestä A kieleen B.

7.9 Toteutuvuusongelman NP-täydellisyys

Aluksi lyhyt kertaus edellä esitetystä:

1. Luokka P sisältää — tietyin varauksin — kaikki ongelmat, jotka tietokoneilla voidaan käytännössä ratkaista.

2. Luokka NP sisältää luokan P ongelmien lisäksi lukuisia käytännössä merkittäviä ongelmia, joille ei ole intensiivisestä tutkimuksesta huolimatta onnistuttu löytämään polynomisia ratkaisualgoritmeja.

3. Luokkaan NP kuuluva ongelma on *NP-täydellinen*, jos kaikki luokan NP ongelmat voidaan palauttaa siihen polynomisilla muunnoksilla. Tällainen ongelma voi kuulua luokkaan P vain jos *kaikki* luokan NP ongelmat kuuluvat luokkaan P, so. jos P = NP (lemma 7.13). Tämä on, edellisen kohdan nojalla, erittäin epätodennäköistä, ja siten vahva todiste sen puolesta, että tarkasteltava ongelma on laskennallisesti työläs. (Täyttä varmuutta NP-täydellisten ongelmien työläydestä ei tietenkään saada, ennen kuin väite P $\neq$ NP on osoitettu todeksi.)

4. Kun jo tunnetaan joukko NP-täydellisiä ongelmia, sujuu uuden NP-ongelman täydellisyystodistus kohtuullisen yksinkertaisesti seuraavaan tapaan:

   (a) valitaan jokin tunnettu NP-täydellinen ongelma, joka muistuttaa tarkasteltavana olevaa uutta ongelmaa;

   (b) muodostetaan polynominen palautus vanhasta ongelmasta uuteen.

   Uuden ongelman NP-täydellisyys seuraa tästä lemman 7.14 nojalla. Esimerkkejä tästä menettelystä esitetään tuonnempana, luvussa 7.10.

Ennen kuin NP-täydellisten ongelmien "kirjastoa" voidaan ryhtyä keräämään lemman 7.14 avulla, täytyy jokin ongelma osoittaa NP-täydelliseksi muulla tavoin. Tämä on Stephen Cookin kuuluisan tuloksen sisältö. Ennen lauseen todistusta kerrataan vielä lausekalkyylin toteutuvuusongelman määritelmä (s. 113):

> Annettu muuttujista $b_1, \ldots, b_n$, vakioista 0 ja 1, sekä konnektiiveista $\wedge$, $\vee$ ja $\neg$ koostuva lausekalkyylin kaava F. On ratkaistava, onko F *toteutuva*, so. voidaanko muuttujille $b_1, \ldots, b_n$ asettaa arvot 0 ja 1 siten, että koko lausekkeen arvoksi tulee 1.

Lause 7.15 (Cook) *Kieli*

$$SAT = \{F \mid F \textit{ on toteutuva lausekalkyylin kaava}\}$$

on NP-täydellinen.

$*$ *Todistus.* On helppo todeta, että SAT $\in$ NP: tunnistavan epädeterministisen koneen tarvitsee syötteellä F vain arvata F:n muuttujien totuusarvoasetus — jos sellainen on — ja tarkistaa arvauksen oikeellisuus[8].

Kielen NP-täydellisyyden todistamiseksi täytyy sitten osoittaa, että mille tahansa kielelle $A \in$ NP voidaan muodostaa polynominen palautus $f : A \leq_m^p$ SAT. Koska ainoa tapa päästä käsiksi mielivaltaiseen luokan NP kieleen on *tarkastella sen tunnistavaa epädeterminististä Turingin konetta*, menetellään seuraavasti:

Olkoon M jokin polynomisessa ajassa toimiva epädeterministinen Turingin kone, jolla $L(M) = A$. Seuraavassa esitetään, miten M:n rakennetta seuraten voidaan muodostaa eräs palautusfunktio f. Funktio f muuntaa kunkin merkkijonon x (so. koneen M mahdollisen syötteen) vastaavaksi lausekalkyylin kaavaksi F_x, joka tietyllä tapaa kuvaa kaikkia M:n mahdollisia laskentoja syötteellä x. Kaava F_x on toteutuva, jos ja vain jos M hyväksyy x:n.

[8] Tarkkaan ottaen koneen pitää ensin tarkastaa, että syötteenä annettu merkkijono F todella on syntaktisesti kelvollinen lausekalkyylin kaava, mutta tämä on helppoa.

Tarkemmin sanoen, kutakin M:n mahdollista laskentaa vastaa yksi F_x:n muuttujien totuusarvoasetus; lisäksi muuttujilla on "epäkelpoja" totuusarvoasetuksia, jotka eivät vastaa mitään laskentaa. Kaava F_x ilmaisee ne ehdot, joiden vallitessa annettu totuusarvoasetus esittää M:n hyväksyvää laskentaa syötteellä x. Siten:

$$\begin{aligned} x \in A = L(M) \\ &\Leftrightarrow \quad M \text{ hyväksyy } x\text{:n} \\ &\Leftrightarrow \quad \text{on olemassa } M\text{:n hyväksyvä laskenta syötteellä } x \\ &\Leftrightarrow \quad \text{on olemassa kaavan } F_x \text{ toteuttava totuusarvoasetus} \\ &\Leftrightarrow \quad F_x = f(x) \in \text{SAT}. \end{aligned}$$

Olkoon siis $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ epädeterministinen Turingin kone, ja $p(n) \geq n$ polynomi, jolla $\text{time}_M(n) \leq p(n)$ kaikilla n. Lemman 7.11 nojalla voidaan olettaa, että kone M on yksinauhainen. Tiloja ja nauha-aakkoston merkkejä tarvittaessa uudelleen nimeämällä voidaan olettaa, että

$$Q = \{q_0, q_1, \ldots, q_r\},$$

missä $q_{r-1} = q_{\text{yes}}$ ja $q_r = q_{\text{no}}$, ja että

$$\Gamma \cup \{>, <\} = \{s_0, s_1, \ldots, s_{m+1}\},$$

missä $s_0 = >$ ja $s_{m+1} = <$. Koska M toimii ajassa $p(n)$, mikään syötteellä x mahdollinen laskenta ei voi edetä nauhalla pidemmälle kuin merkkipaikkaan $p(|x|) + 1$.

Syötettä x, $|x| = n$, vastaavassa kaavassa F_x käytetään seuraavia Boolen muuttujia:

Tyyppi	*Indeksit*	*Merkitys*
$q[t,k]$	$0 \leq t \leq p(n)$ $0 \leq k \leq r$	Hetkellä t (so. t:n laskenta-askelen jälkeen) M on tilassa q_k.
$h[t,i]$	$0 \leq t \leq p(n)$ $1 \leq i \leq p(n)+1$	Hetkellä t M:n nauhapää on merkkipaikan i kohdalla.
$s[t,i,j]$	$0 \leq t \leq p(n)$ $1 \leq i \leq p(n)+1$ $0 \leq j \leq m+1$	Hetkellä t M:n nauhan merkkipaikassa i on merkki s_j.

Muuttujat on tässä selvyyden vuoksi ryhmitelty ikään kuin taulukoiksi, mutta ne voitaisiin tietenkin haluttaessa uudelleennimetä miten tahansa, esimerkiksi $b_1, b_2, \ldots, b_N$, missä $N = O(p(n)^3)$.

Kukin koneen M mahdollinen laskenta syötteellä x määrää näille muuttujille totuusarvot ilmeisellä tavalla. (Jos laskenta päättyy ennen hetkeä $t = p(n)$, ajatellaan koneen tilanteen pysyvän samana hetkeen $p(n)$ saakka.) Toisaalta läheskään kaikki muuttujien totuusarvot eivät vastaa mahdollisia laskentoja.

Tarkoituksena on muodostaa näistä muuttujista kaava F_x niin, että annettu totuusarvoasetus toteuttaa F_x:n, jos ja vain jos se *vastaa jotakin M:n hyväksyvää laskentaa syötteellä x*. Kun vielä todetaan, että kaava F_x voidaan muodostaa merkkijonosta x deterministisesti polynomisessa ajassa, nähdään että kuvaus $f : x \mapsto F_x$ on haluttu palautus $f : L(M) \leq_m^p \text{SAT}$.

Kaava F_x on muodoltaan kuuden alikaavan tai "osaehdon" konjunktio:

$$F_x = G_1 \wedge G_2 \wedge G_3 \wedge G_4 \wedge G_5 \wedge G_6,$$

tai indeksoitua lyhennysmerkintää käyttäen:

$$F_x = \bigwedge_{i=1}^{6} G_i.$$

Alikaavojen kuvaamat ehdot ovat:

Kaava	*Merkitys*
G_1	Kullakin hetkellä t koneen tila on yksikäsitteisesti määrätty.
G_2	Kullakin hetkellä t koneen nauhapään osoittama paikka on yksikäsitteisesti määrätty.
G_3	Kullakin hetkellä t kussakin nauhan merkkipaikassa on yksikäsitteisesti määrätty merkki.
G_4	Hetkellä 0 koneen tilanne on alkutilanne syötteellä x.
G_5	Hetkellä $p(n)$ kone on hyväksyvässä lopputilanteessa.
G_6	Kullakin hetkellä t, $0 \leq t \leq p(n)-1$, koneen tilanteen muutos hetkestä t hetkeen $t+1$ on siirtymäfunktion δ mukainen.

Viisi ensimmäistä ehtoa voidaan toteuttaa seuraavilla kaavoilla:

$$G_1 = \bigwedge_{0\leq t\leq p(n)} \bigvee_{0\leq k\leq r} q[t,k] \quad \wedge \quad \bigwedge_{\substack{0\leq t\leq p(n)\\ 0\leq k<k'\leq r}} \neg(q[t,k] \wedge q[t,k']);$$

$$G_2 = \bigwedge_{0\leq t\leq p(n)} \bigvee_{1\leq i\leq p(n)+1} h[t,i] \quad \wedge \quad \bigwedge_{\substack{0\leq t\leq p(n)\\ 0\leq i<i'\leq p(n)+1}} \neg(h[t,i] \wedge h[t,i']);$$

$$G_3 = \bigwedge_{\substack{0\leq t\leq p(n)\\ 0\leq i\leq p(n)+1}} \bigvee_{0\leq j\leq m+1} s[t,i,j] \quad \wedge \quad \bigwedge_{\substack{0\leq t\leq p(n)\\ 0\leq i\leq p(n)+1\\ 0\leq j<j'\leq m+1}} \neg(s[t,i,j] \wedge s[t,i,j']);$$

$$G_4 = q[0,0] \wedge h[0,1] \wedge s[0,0,0] \wedge \bigwedge_{1\leq i\leq n} s[0,i,j_i] \wedge \bigwedge_{n+1\leq i\leq p(n)+1} s[0,i,m+1],$$

kun $x = s_{j_1} s_{j_2} \ldots s_{j_n}$;

$$G_5 = q[p(n), r-1].$$

Alikaava G_6 kuvaa koneen tilan, nauhapään sijainnin ja kunkin nauhamerkin muuttumista yhdessä laskenta-askelessa. Se koostuu kolmesta osasta:

$$G_6 = G_6' \wedge G_6'' \wedge G_6'''.$$

Kaava G_6' toteaa vain sen, että jos hetkellä t koneen nauhapää ei ole merkkipaikan i kohdalla, niin paikassa i oleva merkki pysyy samana hetkellä $t+1$. Tämä voidaan formalisoida seuraavasti[9]:

$$G_6' = \bigwedge_{\substack{0\le t\le p(n)-1\\ 0\le i\le p(n)+1\\ 0\le j\le m+1}} ((s[t,i,j] \wedge \neg h[t,i]) \rightarrow s[t+1,i,j]).$$

Kaava G_6'' toteaa sen, että jos kone hetkellä t on lopputilassa q_{yes} tai q_{no}, niin sen tilanne hetkellä $t+1$ on sama kuin hetkellä t:

$$G_6'' = \bigwedge_{\substack{0\le t\le p(n)-1\\ k=r-1,\, r\\ 0\le i\le p(n)+1\\ 0\le j\le m+1}} [(q[t,k] \wedge h[t,i] \wedge s[t,i,j]) \rightarrow (q[t+1,k] \wedge h[t+1,i] \wedge s[t+1,i,j])].$$

Kaava G_6''' lopulta formalisoi sen keskeisen vaatimuksen, että ellei kone ole ajanhetkeen t mennessä pysähtynyt, niin hetkestä t hetkeen $t+1$ sen tila, nauhapään sijainti ja nauhapään kohdalla oleva merkki muuttuvat siirtymäfunktion δ mukaisella tavalla:

$$G_6''' = \bigwedge_{\substack{0\le t\le p(n)-1\\ 0\le k\le r-2\\ 0\le i\le p(n)+1\\ 0\le j\le m+1}} \Big[(q[t,k] \wedge h[t,i] \wedge s[t,i,j]) \rightarrow \bigvee_{\substack{(q_{k'},s_{j'},\Delta)\\ \in\delta(q_k,s_j)}} (q[t+1,k'] \wedge h[t+1,i+\Delta] \wedge s[t+1,i,j'])\Big].$$

Nauhapään siirtosuunta on tässä ajateltu koodatuksi siten, että suuntaa L vastaa arvo $\Delta = -1$ ja suuntaa R arvo $\Delta = 1$.

Suoraviivainen tarkastus osoittaa, että merkkijonosta x, $|x| = n$, näin muodostettu kaava F_x on polynomista kokoa n:n suhteen (itse asiassa kokoa $O(p(n)^3)$), se voidaan muodostaa deterministisesti polynomisessa ajassa, ja on toteutuva jos ja vain jos $x \in L(M)$. Siten kuvaus $f : x \mapsto F_x$ on palautus kielestä $A = L(M)$ kieleen SAT. □

[9] Tässä ja seuraavassa on käytetty lyhennysmerkintää "$\phi \rightarrow \psi$" kaavalle "$\neg\phi \vee \psi$".

7.10 Muita NP-täydellisiä ongelmia

1. Rajoitetut toteutuvuusongelmat

Lausekalkyylin kaava F on *konjunktiivisessa normaalimuodossa* (engl. conjunctive normal form, lyh. cnf), jos se on muotoa

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

missä kukin *tekijä* (engl. clause) C_i on disjunktio

$$C_i = \alpha_{i1} \vee \alpha_{i2} \vee \ldots \vee \alpha_{ir_i}.$$

Tässä termit α_{ij} ovat *literaaleja* (engl. literals), so. muuttujia tai niiden negaatioita.

Ongelma CSAT koskee tällaista muotoa olevien kaavojen toteutuvuutta:

$$\text{CSAT} = \{F \mid F \text{ on cnf-muotoinen toteutuva lausekalkyylin kaava}\}.$$

Lause 7.16 *Kieli CSAT on NP-täydellinen.*

∗ *Todistus.*

(i) *CSAT* $\in$ *NP.* Tämä on selvää, koska CSAT on SAT:in erikoistapaus, ja SAT $\in$ NP.

(ii) $A \in NP \Rightarrow A \leq_m^p CSAT$. Tutkimalla Cookin lauseen (lause 7.15) todistusta nähdään, että siinä muodostettavat kaavat

$$F_x = G_1 \wedge G_2 \wedge G_3 \wedge G_4 \wedge G_5 \wedge (G_6' \wedge G_6'' \wedge G_6''')$$

ovat melkein cnf-muotoisia, kun lyhennysmerkinnät "$\phi \to \psi$" kirjoitetaan auki muotoon "$\neg\phi \vee \psi$", ja tarvittaessa sovelletaan De Morganin sääntöä $\neg(\phi \wedge \psi) \equiv (\neg\phi \vee \neg\psi)$. Ainoan poikkeuksen muodostavat alikaavat G_6'' ja G_6'''. Jos keskitytään tarkastelemaan mutkikkaampaa kaavaa G_6''', niin tämä on muotoa

$$G_6''' = \bigwedge_t \Big[(q_{t0} \wedge h_{t0} \wedge s_{t0}) \to \bigvee_{1 \leq i \leq k} (q_{ti} \wedge h_{ti} \wedge s_{ti})\Big],$$

eli

$$G_6''' = \bigwedge_t \Big[\neg q_{t0} \vee \neg h_{t0} \vee \neg s_{t0} \vee \bigvee_{1 \leq i \leq k} (q_{ti} \wedge h_{ti} \wedge s_{ti})\Big],$$

missä k on n:stä riippumaton vakio[10].

Lausekalkyylin osittelulakien nojalla saadaan tästä myös kaavalle G_6''' konjunktiivinen normaalimuoto:

$$G_6''' = \bigwedge_t \bigwedge_{\alpha \in \{q,h,s\}^k} \Big[\neg q_{t0} \vee \neg h_{t0} \vee \neg s_{t0} \vee \bigvee_{1 \leq i \leq k} \alpha(i)_{ti}\Big].$$

Tämä normaalimuoto on tosin noin 3^k kertaa alkuperäisen kaavan kokoinen, mutta koska k on vakio, kasvu ei haittaa. □

[10] Tarkasti ottaen voidaan valita

$$k = \max\{|\delta(q,s)| \mid q \in Q,\ s \in \Gamma \cup \{>,<\}\}.$$

Kaava G_6'' on samaa muotoa, kun valitaan $k = 1$.

Toteutuvuusongelma pysyy NP-täydellisenä ankarampienkin rajoitusten vallitessa. Sanotaan, että lausekalkyylin kaava F on *k-konjunktiivisessa normaalimuodossa* (lyh. k-cnf), jos sen kukin tekijä sisältää täsmälleen k literaalia, ja määritellään:

$$\text{kSAT} = \{F \mid F \text{ on } k\text{-cnf-muotoinen toteutuva lausekalkyylin kaava}\}.$$

Lause 7.17 *Kieli 3SAT on NP-täydellinen.*

Huomautus. Voidaan osoittaa, että kieli 2SAT kuuluu luokkaan P.

$*$ *Todistus.*

(i) *$3SAT \in NP$.* Selvä.

(ii) *$NP \leq_m^p 3SAT$.* Lemman 7.14 nojalla riittää osoittaa, että CSAT $\leq_m^p$ 3SAT. Toisin sanoen: on osoitettava, että annetusta cnf-muotoisesta kaavasta F voidaan polynomisessa ajassa muodostaa 3-cnf-muotoinen kaava F', joka on toteutuva jos ja vain jos F on toteutuva.

Vaadittu muunnos voidaan tehdä seuraavasti: olkoon

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m.$$

Kukin F:n tekijä

$$C_k = \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_r, \quad r \geq 3,$$

korvataan 3-cnf-muotoisella kaavalla

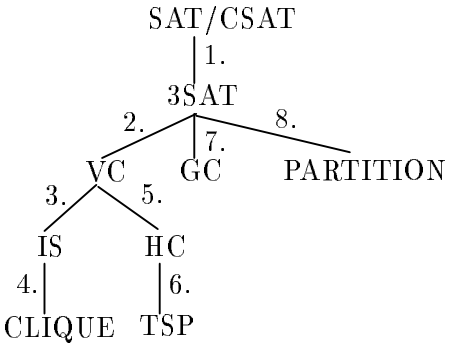
$$C_k' = (\alpha_1 \vee \alpha_2 \vee t_1) \wedge (\neg t_1 \vee \alpha_3 \vee t_2) \wedge (\neg t_2 \vee \alpha_4 \vee t_3) \wedge \ldots \wedge (\neg t_{r-3} \vee \alpha_{r-1} \vee \alpha_r),$$

missä $t_1, \ldots, t_{r-3}$ ovat uusia muuttujia. Kukin kaava C_k' voidaan selvästi muodostaa vastaavasta kaavasta C_k polynomisessa ajassa. Tarkastetaan vielä, että muunnos $F \mapsto F'$ säilyttää kaavojen toteutuvuuden ja toteutumattomuuden, so. että muunnos täyttää palautusehdon $F \in \text{CSAT} \Leftrightarrow F' \in \text{3SAT}$:

(a) *F toteutuva $\Rightarrow$ F' toteutuva:* Tarkastellaan jotakin kaavan F tekijää C_k ja vastaavaa F':n tekijää C_k'. Minkä tahansa F:n toteuttavan totuusarvoasetuksen täytyy asettaa jonkin C_k:ssa esiintyvän literaalin α_i arvoksi 1. Tästä saadaa C_k':n toteuttava totuusarvoasetus asettamalla literaalien α_j arvot samoin, ja uusien muuttujien arvot seuraavasti:

$$t_j = \begin{cases} 1, & \text{jos } j \leq i-2; \\ 0, & \text{jos } j > i-2. \end{cases}$$

(b) *F' toteutuva $\Rightarrow$ F toteutuva:* Mikä tahansa kaavan F' toteuttava totuusarvoasetus toteuttaa erityisesti kutakin F:n tekijää C_k vastaavan alikaavan C_k'. Tällöin joko jokin alikaavassa esiintyvistä literaaleista $\alpha_1, \ldots, \alpha_r$ saa arvon 1 ja tekijä C_k toteutuu, tai jollakin $i < r-3$ on $t_i = 1$, $t_{i+1} = 0$; mutta myös jälkimmäisessä tapauksessa on oltava $\alpha_{i+2} = 1$, ja tekijä C_k toteutuu.



Kuva 7.7: NP-täydellisten ongelmien välisiä palautuksia.

Edellä on tarkasteltu vain vähintään neljä literaalia sisältävien tekijöiden muuntamista 3-cnf-muotoon. Yksinkertaisemmat tekijät käsitellään seuraavasti:

$$\begin{aligned} C_k = \alpha_1 \vee \alpha_2 \vee \alpha_3 \quad &\Rightarrow \quad C'_k = C_k; \\ C_k = \alpha_1 \vee \alpha_2 \quad &\Rightarrow \quad C'_k = (\alpha_1 \vee \alpha_2 \vee t) \wedge (\alpha_1 \vee \alpha_2 \vee \neg t); \\ C_k = \alpha \quad &\Rightarrow \quad C'_k = \; (\alpha \vee t_1 \vee t_2) \wedge (\alpha \vee t_1 \vee \neg t_2) \wedge \\ & \qquad\qquad (\alpha \vee \neg t_1 \vee t_2) \wedge (\alpha \vee \neg t_1 \vee \neg t_2). \end{aligned}$$

Selvästi myös nämä muunnokset säilyttävät kaavojen toteutuvuusominaisuudet. □

2. Sekalaisia ongelmia

Luonnollisia NP-täydellisiä ongelmia tunnetaan nykyisin jo yli 1000, ja määrä kasvaa jatkuvasti. Osoittautuu, että esimerkiksi kaikki luvussa 7.7 mainitut NP-ongelmat ovat itse asiassa täydellisiä:

Lause 7.18 *Seuraavat ongelmat (oik. vastaavat formaalit kielet) ovat NP-täydellisiä:*

(i) solmupeiteongelma (VC);

(ii) riippumaton joukko -ongelma (IS);

(iii) klikkiongelma (CLIQUE);

(iv) Hamiltonin kehä -ongelma (HC);

(v) kauppamatkustajan ongelma (TSP);

(vi) verkonväritysongelma (GC);

(vii) ositusongelma (PARTITION).

Huomautus: Luvuissa 4.2 ja 7.6 tarkasteltua yhdistettyjen lukujen tunnistamisongelmaa ei tiedetä NP-täydelliseksi. Itse asiassa pidetään hyvin mahdollisena, että ongelmalle lopulta löytyy polynomisessa ajassa toimiva ratkaisualgoritmi.

Todistus. Kaikkien mainittujen ongelmien todettiin kuuluvan luokkaan NP jo luvussa 7.7. Lemman 7.14 mukaiset, ongelmien täydellisyyden osoittavat palautukset voidaan tehdä esimerkiksi kuvassa 7.7 esitetyn kaavion mukaisesti. Seuraavassa käsitellään vain kaavion palautukset 2–4. Palautus 1 esitettiin jo lauseessa 7.17, ja palautukset 5–8 sivuutetaan.

Tarkastellaan yksinkertaisuuden vuoksi ensin palautuksia 3 ja 4. Nämä ovat erityisen helppoja, koska ongelmat VC, IS ja CLIQUE ovat oikeastaan sama ongelma eri tarkastelukulmista katsottuna:

Lemma 7.19 *Olkoon $G = (V, E)$[11] suuntaamaton verkko ja $V' \subseteq V$. Tällöin ovat seuraavat kolme ehtoa ekvivalentit:*

(i) V' on G:n solmupeite;

(ii) $V - V'$ on riippumaton solmujoukko G:ssä;

(iii) $V - V'$ on klikki G:n komplementtiverkossa $\bar{G} = (V, (V \times V) - E)$.

Todistus. HT. □

Vaaditut palautukset voidaan muodostaa seuraavasti.

3. $VC \leq_m^p IS$. Merkitään verkon G solmujen määrää $|G|$:llä ja valitaan palautusfunktioksi kuvaus f,

$$f(\langle G, k\rangle) = \langle G, |G| - k\rangle.$$

Muunnos f voidaan selvästi laskea polynomisessa ajassa. Lemman 7.19 mukaan verkossa G on enintään k solmun solmupeite, jos ja vain jos siinä on vähintään $|G| - k$ solmun riippumaton joukko; toisin sanoen:

$$\langle G, k\rangle \in \text{VC} \quad \Leftrightarrow \quad f(\langle G, k\rangle) = \langle G, |G| - k\rangle \in \text{IS}.$$

4. $IS \leq_m^p CLIQUE$. Lemman 7.19 mukaan verkko G sisältää vähintään k solmun riippumattoman joukon, jos ja vain jos sen komplementtiverkko $\bar{G}$ sisältää vähintään k solmun klikin. Siten palautusfunktioksi voidaan valita kuvaus f,

$$f(\langle G, k\rangle) = \langle \bar{G}, k\rangle.$$

Kuvaus voidaan selvästi laskea polynomisessa ajassa, ja

$$\langle G, k\rangle \in \text{IS} \quad \Leftrightarrow \quad f(\langle G, k\rangle) = \langle \bar{G}, k\rangle \in \text{CLIQUE}.$$

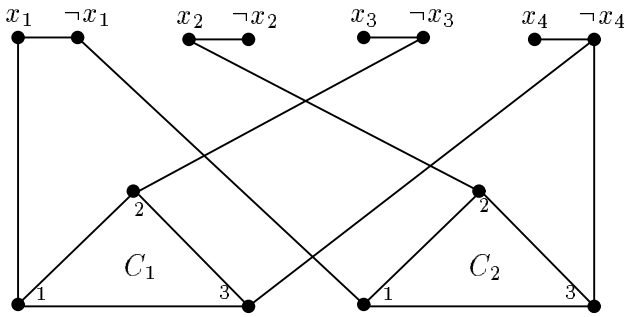
Tarkastellaan lopuksi vaikeampaa palautusta 2.

* *2. $3SAT \leq_m^p VC$.* Olkoon

$$F = C_1 \wedge \ldots \wedge C_m$$

3-cnf-muotoinen lausekalkyylin kaava, jossa esiintyvät muuttujat $x_1, \ldots, x_n$. Vastaava solmupeiteongelman tapaus $\langle G, k\rangle$ muodostetaan seuraavasti (ks. kuva 7.8). Verkossa G on solmu kullekin literaalille x_i ja $\neg x_i$, missä $i = 1, \ldots, n$, sekä kutakin F:n tekijää C_j kohden kolme solmua C_j^1, C_j^2 ja C_j^3, missä $j = 1, \ldots, m$. Verkossa on seuraavat kaaret:

[11] Tässä on käytetty tavanomaista verkkomerkintää: V on verkon G solmujoukko ja $E \subseteq V \times V$ sen kaarijoukko.



Kuva 7.8: Kaavaa $F = (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$ vastaava verkko.

- $(x_i, \neg x_i)$, $i = 1, \ldots, n$;
- (C_j^1, C_j^2), (C_j^2, C_j^3), (C_j^3, C_j^1), $j = 1, \ldots, m$;
- jos $C_j = (\alpha_1 \vee \alpha_2 \vee \alpha_3)$, niin (C_j^1, α_1), (C_j^2, α_2), (C_j^3, α_3), $j = 1, \ldots, m$.

Arvoksi k valitaan luku $n + 2m$.

Esimerkkinä konstruktion soveltamisesta on kuvassa 7.8 esitetty kaavasta

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

muodostettu verkko. Tässä tapauksessa valitaan $k = 4 + 2 \cdot 2 = 8$.

Verkko G voidaan selvästi muodostaa kaavasta F polynomisessa ajassa. Osoitetaan, että G:llä on enintään k solmun solmupeite, jos ja vain jos F on toteutuva:

(i) Olkoon $t : \{x_1, \ldots, x_n\} \to \{0, 1\}$ jokin kaavan F toteuttava totuusarvoasetus. Vastaavaan verkon G solmupeitteeseen V' otetaan ensin kutakin literaaliparia x_i, $\neg x_i$ kohden se solmu, jota vastaava literaali saa arvon 1. Tämän jälkeen on kustakin C_j-kolmiosta ainakin yhdestä kulmasta alkava kaari (C_j^r, α_r) jo peitetty, ja peitteeseen lisätään kolmion kaksi muuta kulmasolmua. Näin saatu solmujoukko V' selvästi peittää kaikki G:n kaaret ja $|V'| = n + 2m = k$.

(ii) Olkoon toisaalta V' jokin G:n solmupeite, jolla $|V'| \leq k$. Koska V':n on sisällettävä vähintään yksi solmu kustakin literaaliparista x_i, $\neg x_i$, ja vähintään kaksi solmua kustakin C_j-kolmiosta, on oltava myös $|V'| \geq n + 2m = k$. Siten on $|V'| = k$, ja V' sisältää *täsmälleen* yhden solmun kustakin literaaliparista ja *täsmälleen kaksi solmua* kustakin C_j-kolmiosta. Tästä saadaan muuttujille x_i totuusarvoasetus asettamalla

$$t(x_i) = \begin{cases} 1, & \text{jos } x_i \in V'; \\ 0, & \text{jos } \neg x_i \in V'. \end{cases}$$

Nyt kunkin C_j-kolmion kärjistä alkavista kaarista vain kaksi voi olla kolmiosta peitteeseen valittujen solmujen peittämiä; kolmannen peittää jokin literaalisolmu $\alpha \in V'$. Mutta tällöin $t(\alpha) = 1$, ja totuusarvoasetus t toteuttaa tekijän C_j. □

Luku 8

Kirjallisuutta

Laskennan teoriasta on ilmestynyt lukuisia englanninkielisiä yleisesityksiä; seuraavassa joitakin esimerkkejä:

J. G. Brookshear: *Theory of Computation: Formal Languages, Automata, and Complexity.* Benjamin/Cummings, Menlo Park, CA 1989.

D. I. A. Cohen: *Introduction to Computer Theory, 2nd Ed.* John Wiley & Sons, New York, NY, 1991.

E. Gurari: *An Introduction to the Theory of Computation.* Computer Science Press, Rockville, MD, 1989.

J. E. Hopcroft, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

H. R. Lewis, C. H. Papadimitriou: *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

J. C. Martin: *Introduction to Languages and the Theory of Computation.* McGraw-Hill, New York, NY, 1991.

T. A. Sudkamp: *Languages and Machines: An Introduction to the Theory of Computer Science.* Addison-Wesley, Reading, MA, 1988.

D. Wood: *Theory of Computation.* Harper & Row, New York, NY, 1987.

Näistä teoksista Hopcroftin/Ullmanin ja Martinin kirjat ovat lähimpänä tämän monisteen esitystä, Hopcroft/Ullman tosin tiiviimpi ja laajempi. Myös Lewisin/Papadimitrioun, Sudkampin ja Gurarin kirjat ovat erittäin suositeltavia. Gurarin teos lähestyy aihepiiriä mielenkiintoisella tavalla ohjelmoinnin näkökulmasta. Brookshearin teos on helppolukuinen, mutta ei täsmällisyydeltään aivan moitteeton. Cohenin kirja on samoin hyvin laveasanainen, mutta ei valitettavasti käsittele lainkaan laskennan vaativuusteoriaa. Woodin kirjassa on runsaasti materiaalia, jopa niin että esityksen päälinjat tahtovat hukkua yksityiskohtien runsauteen.

Erikoisalojen kirjallisuudesta mainittakoon esimerkkeinä seuraavat:

Formaalit kielet, kieliopit ja automaatit

M. A. Harrison: *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.

A. Salomaa: *Formal Languages.* Academic Press, New York, NY, 1973.

Jäsennysteoria ja kääntäjätekniikka

A. V. Aho, R. Sethi, J. D. Ullman: *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

S. Sippu, E. Soisalon-Soininen: *Parsing Theory I–II.* Springer-Verlag, Berlin, 1988/1990.

Laskettavuusteoria

G. S. Boolos, R. C. Jeffrey: *Computability and Logic, 3rd Ed.* Cambridge University Press, Cambridge, 1989.

H. Rogers, Jr.: *Theory of Recursive Functions and Effective Computability.* McGraw–Hill, New York, NY, 1967. (Nidottu uusintapainos MIT Press, Cambridge, MA, 1987.)

P. Odifreddi: *Classical Recursion Theory.* Elsevier–North-Holland, Amsterdam, 1989.

Laskennan vaativuusteoria

J. L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity I–II.* Springer-Verlag, Berlin, 1988/1990.

D. P. Bovet, P. Crescenzi: *Introduction to the Theory of Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1994.

M. R. Garey, D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, 1979.

C. H. Papadimitriou: *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

Harjoitustehtäviä