T-79.1001/1002 Tietojenkäsittelyteorian perusteet

Luentomoniste

Pekka Orponen
Teknillinen korkeakoulu
Tietojenkäsittelyteorian laboratorio

Lukijalle

Tämä moniste on syntynyt Teknillisessä korkeakoulussa sekä Helsingin ja Jyväskylän yliopistoissa useina vuosina pitämieni luentojen pohjalta. Moniste antaa perustiedot siitä tietojenkäsittelyteorian alasta, jolle on englanninkielessä vakiintunut nimi “Theory of Computation”, siis automaattien, kielioppien ja muiden laskennan mallien ominaisuuksista. Aihepiiriin luonnostaan kuuluvan laskennan vaativuusteorian (ns. “NP-täydellisyysteorian”) olen kuitenkin rajannut monisteen tämän version ulkopuolelle, pääosin siitä syystä että tietojenkäsittelyteorian johdantokurssien nykyiset opetustuntimääärät eivät salli tämän laajan alueen käsittelyä, vaan siitä järjestetään itsenäisiä erikoiskursseja.

Tietojenkäsittelyteorian perusopetuksessa 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 ($*$).

Kiitän lämpimästi kaikkia monistetta vuosien mittaan omassa opetuksessaan käyttäneitä ja sen sisältöön ja esitystapaan kommenteillaan vaikuttaneita henkilöitä. Erityisen huolellista, sekä kurssin kokonaisuutta että monisteen yksityiskohtia koskevaa palautetta olen viime vuosina saanut Tapio Elomaalta, Patrik Floréenilta, Harri Haanpäältä, Wilhelmiina Hämäläiseltä, Timo Karvilta, Timo Latvalalta sekä Tommi Syrjäseltä.

Otaniemessä, 19. syyskuuta 2005

Sisältö

Luku 1

Matemaattisia peruskäsitteitä

1.1 Joukot

Joukko (engl. set) on kokoelma alkioita. Alkiot voidaan ilmoittaa joko luettelemalla, esimerkiksi

$$S = \{2, 3, 5, 7, 11, 13, 17, 19\}$$

tai jonkin säännön avulla, esimerkiksi

$$S = \{p \mid p \text{ on alkuluku}, 2 \leq p \leq 20\}.$$

Jos alkio a kuuluu joukkoon A, merkitään $a \in A$, päinvastaisessa tapauksessa $a \notin A$. Esimerkiksi edellä on $3 \in S$, $8 \notin S$.

Tärkeä erikoistapaus on *tyhjä joukko* (engl. empty set) $\emptyset$, johon ei kuulu yhtään alkiota.

Jos joukon A kaikki alkiot kuuluvat myös joukkoon B, sanotaan että A on B:n *osajoukko* (engl. subset) ja merkitään $A \subseteq B$.[1] Jos A ei ole B:n osajoukko merkitään $A \not\subseteq B$. Siis esimerkiksi edellä on $\{2, 3\} \subseteq S$, mutta $\{1, 2, 3\} \not\subseteq S$. Triviaalisti on voimassa $\emptyset \subseteq A$ kaikilla A.

Joukot A ja B ovat samat, jos niissä on samat alkiot, so. jos on $A \subseteq B$ ja $B \subseteq A$. Jos on $A \subseteq B$, mutta $A \neq B$, sanotaan että A on B:n *aito osajoukko* (engl. proper subset) ja merkitään $A \subsetneq B$. Edellä olisi siis voitu kirjoittaa myös $\{2, 3\} \subsetneq S$ ja $\emptyset \subsetneq A$ jos $A \neq \emptyset$.

Joukon alkioina voi olla myös toisia joukkoja (tällöin puhutaan usein *joukkoperheestä*), esimerkiksi

$$X = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}.$$

Jonkin perusjoukon A kaikkien osajoukkojen muodostamaa joukkoperhettä sanotaan A:n *potenssijoukoksi* (engl. powerset) ja merkitään $\mathcal{P}(A)$:lla; esimerkiksi edellä on $X = \mathcal{P}(\{1, 2\})$.[2] Selvästi on $A \subseteq B$ jos ja vain jos $A \in \mathcal{P}(B)$.

Joukkoja voidaan kombinoida *joukko-operaatioilla*, joista tärkeimmät ovat:

[1] Kirjallisuudessa esiintyy myös merkintä $A \subset B$.

[2] Koska n-alkioisen perusjoukon A potenssijoukossa on 2^n alkiota, käytetään kirjallisuudessa potenssijoukolle myös merkintää 2^A.

(i) *Yhdiste* (engl. union)

$$A \cup B = \{x \mid x \in A \text{ tai } x \in B\},$$

esimerkiksi $\{1, 2, 3\} \cup \{1, 4\} = \{1, 2, 3, 4\}$.

(ii) *Leikkaus* (engl. intersection)

$$A \cap B = \{x \mid x \in A \text{ ja } x \in B\},$$

esimerkiksi $\{1, 2, 3\} \cap \{1, 4\} = \{1\}$.

(iii) *Erotus* (engl. difference)

$$A - B = \{x \mid x \in A \text{ ja } x \notin B\},$$

esimerkiksi $\{1, 2, 3\} - \{1, 4\} = \{2, 3\}$.[3]

Joukko-operaatioita koskevat tietyt laskulait, joista tärkeimmät ovat yhdisteen ja leikkauksen *liitännäisyys*:

$$A \cup (B \cup C) = (A \cup B) \cup C, \quad A \cap (B \cap C) = (A \cap B) \cap C$$

ja *vaihdannaisuus*:

$$A \cup B = B \cup A, \quad A \cap B = B \cap A$$

sekä näiden *osittelulait*:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C),$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

Jos kaikki tarkasteltavat joukot ovat jonkin yhteisen perusjoukon U osajoukkoja, sanotaan erotusta $U - A$ joukon A *komplementiksi* (U:n suhteen) ja merkitään $\bar{A}$:lla. Yhdiste-, leikkaus- ja komplementointioperaatioita yhdistävät tärkeät *de Morganin kaavat*:

$$\overline{A \cup B} = \bar{A} \cap \bar{B}, \quad \overline{A \cap B} = \bar{A} \cup \bar{B}.$$

Lisäksi joukkojen erotus voidaan esittää leikkauksen ja komplementoinnin avulla seuraavasti:

$$A - B = A \cap \bar{B}.$$

Jos joukkoperheen $\mathcal{A}$ jäsenet on *indeksoitu*, esim. $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, niin yhdisteelle ja leikkaukselle voidaan käyttää lyhennemerkintöjä

$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n$$

[3] Joukkoerotukselle käytetään kirjallisuudessa myös merkintää $A \setminus B$.

ja

$$\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \cdots \cap A_n.$$

Samaan tapaan voidaan muodostaa myös äärettömien joukkoperheiden yhdisteitä ja leikkauksia. Jos esimerkiksi $\mathcal{A} = \{A_1, A_2, A_3, \ldots\}$, niin määritellään

$$\bigcup_{i \geq 1} A_i = \bigcup_{i=1}^{\infty} A_i = \{x \mid x \in A_i \text{ jollakin } i = 1, 2, \ldots\}$$

ja

$$\bigcap_{i \geq 1} A_i = \bigcap_{i=1}^{\infty} A_i = \{x \mid x \in A_i \text{ kaikilla } i = 1, 2, \ldots\}.$$

Indeksien ei yleisessä tapauksessa tarvitse olla edes luonnollisia lukuja, vaan *indeksijoukkona* voi olla mikä tahansa joukko I, so. $\mathcal{A} = \{A_i \mid i \in I\}$. Tällöin käytetään merkintöjä

$$\bigcup_{i \in I} A_i = \{x \mid x \in A_i \text{ jollakin } i \in I\}$$

ja

$$\bigcap_{i \in I} A_i = \{x \mid x \in A_i \text{ kaikilla } i \in I\}.$$

1.2 Relaatiot ja funktiot

Olkoot A ja B joukkoja. Alkioiden $a \in A$ ja $b \in B$ *järjestettyä paria* (engl. ordered pair) merkitään (a, b). On huomattava, että joukkoina on aina $\{a, b\} = \{b, a\}$, mutta jos $a \neq b$, niin järjestettyinä pareina on $(a, b) \neq (b, a)$.[4] Kaikkien em. tyyppisten parien kokoelmaa sanotaan joukkojen A ja B *karteesiseksi tuloksi* (engl. Cartesian product) ja merkitään:

$$A \times B = \{(a, b) \mid a \in A \text{ ja } b \in B\}.$$

Näin ollen on esimerkiksi:

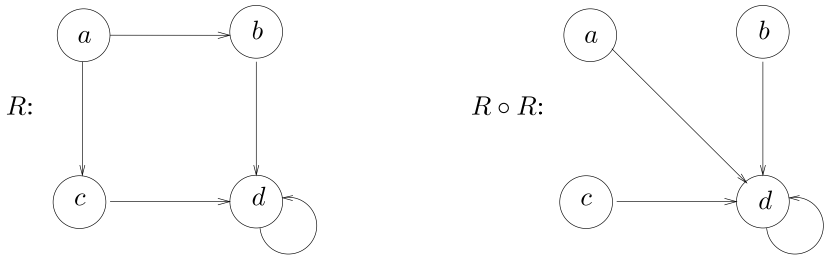
$$\{1, 2, 3\} \times \{1, 4\} = \{(1, 1), (1, 4), (2, 1), (2, 4), (3, 1), (3, 4)\}.$$

Relaatio R joukolta A joukolle B on mielivaltainen kokoelma järjestettyjä pareja (a, b), missä $a \in A$ ja $b \in B$, so. jokin karteesisen tulon $A \times B$ osajoukko:

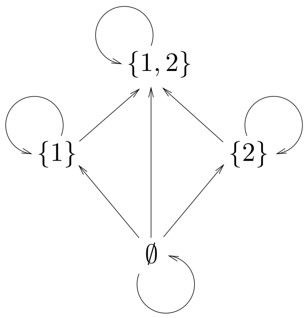
$$R \subseteq A \times B.$$

Jos relaation R *lähtöjoukko* (engl. domain) A ja *maalijoukko* (engl. codomain, range) B ovat samat, so. $R \subseteq A \times A$, sanotaan että R on relaatio *joukossa* A.

[4] Täsmällisesti, mutta epäintuitiivisesti voidaan järjestetty pari määritellä joukkona $(a, b) = \{a, \{a, b\}\}$.



Kuva 1.1: Relaatioiden R ja $R \circ R$ graafit.



Kuva 1.2: Osajoukkorelaation S graafi.

Jos $(a, b) \in R$, niin merkitään myös aRb ja sanotaan että alkio a on *relaatiossa (suhteessa)* R alkioon b. Em. *infix*-merkintää käytetään varsinkin silloin, kun relaation nimenä on jokin erikoismerkki, esimerkiksi $\leq$, $\prec$, $\equiv$ tai $\sim$. (Onhan luontevampaa kirjoittaa esimerkiksi "$a \prec b$" kuin "$(a, b) \in \prec$".)

Olkoon $R \subseteq A \times B$. Osajoukon $A' \subseteq A$ ***kuva*** (engl. image) relaatiossa R on

$$R[A'] = \{b \in B \mid \exists a' \in A' \text{ s.e. } (a', b) \in R\}$$

ja osajoukon $B' \subseteq B$ ***alkukuva*** (engl. inverse image) on

$$R^{-1}[B'] = \{a \in A \mid \exists b' \in B' \text{ s.e. } (a, b') \in R\}.$$

Relaation $R \subseteq A \times B$ ***käänteisrelaatio*** (engl. inverse relation) on relaatio $R^{-1} \subseteq B \times A$,

$$R^{-1} = \{(b, a) \mid (a, b) \in R\}.$$

Relaatioiden $R \subseteq A \times B$ ja $S \subseteq B \times C$ ***yhdistetty relaatio*** (engl. composite relation) $R \circ S \subseteq A \times C$ määritellään:

$$R \circ S = \{(a, c) \mid \exists b \in B \text{ s.e. } (a, b) \in R, (b, c) \in S\}.$$

Relaatiota $R \subseteq A \times B$ on usein — varsinkin jos joukot A ja B ovat äärellisiä — havainnollista tarkastella *suunnattuna verkkona* t. *graafina*, jonka *solmuina* ovat joukkojen A ja B alkiot ja solmusta $a \in A$ on *kaari* ("nuoli") solmuun $b \in B$, jos ja vain jos $(a, b) \in R$.

Olkoon esimerkiksi joukossa $A = \{a, b, c, d\}$ määritelty relaatio $R \subseteq A \times A$,

$$R = \{(a, b), (a, c), (b, d), (c, d), (d, d)\}.$$

Kuvassa 1.1 on esitetty relaatiota R ja yhdistettyä relaatiota $R \circ R$ vastaavat graafit. Kuvassa 1.2 puolestaan on esitetty potenssijoukossa $X = \mathcal{P}(\{1,2\})$ määriteltyä osajoukkorelaatiota

$$S = \{(A,B) \in X \times X \mid A \subseteq B\}$$

vastaava graafi.

Relaatio $f \subseteq A \times B$ on *funktio*, jos kukin $a \in A$ on relaatiossa f täsmälleen yhden $b \in B$ kanssa. Alkiota b sanotaan tällöin alkion a *kuvaksi* ja merkitään $f(a) = b$. Jos halutaan korostaa, että relaatio f on funktio, sille käytetään merkintää $f\colon A \to B$. Funktioita koskee kaikki mitä edellä yleisesti on todettu relaatioista, mutta historiallisista syistä funktioiden yhdistäminen merkitään toisin päin kuin yleisten relaatioiden: jos $f\colon A \to B$ ja $g\colon B \to C$ ovat funktioita, niin niiden yhdistetty funktio määritellään kaavalla $(g \circ f)(a) = g(f(a))$.

Funktio $f\colon A \to B$ on *surjektio* (engl. onto map), jos jokainen $b \in B$ on jonkin $a \in A$ kuva, so. jos $f[A] = B$, ja *injektio* (engl. one-to-one map), jos kaikki $a \in A$ kuvautuvat eri alkioille, so. jos $a \neq a' \Rightarrow f(a) \neq f(a')$ kaikilla $a, a' \in A$. Funktio f on *bijektio*, jos se on sekä injektio että surjektio, so. jos jokainen $b \in B$ on yhden ja vain yhden $a \in A$ kuva.

1.3 Ekvivalenssirelaatiot

Ekvivalenssirelaatiot ovat matemaattisesti täsmällinen muotoilu sille yleiselle idealle, että oliot ovat keskenään *samankaltaisia* jonkin kiinnostavan ominaisuuden $\mathcal{X}$ suhteen. Ominaisuuteen $\mathcal{X}$ perustuva ekvivalenssirelaatio osittaa tarkasteltavien olioiden joukon *ekvivalenssiluokkiin*, jotka vastaavat ominaisuuden $\mathcal{X}$ eri arvoja. (Kääntäen mielivaltainen olioiden joukon ositus Π määrää tietyn abstraktin samankaltaisuusominaisuuden, nimittäin sen että oliot ovat samankaltaisia jos ja vain jos ne sijoittuvat samaan osituksen Π luokkaan.)

Osoittautuu, että yleinen "samankaltaisuusrelaation" idea voidaan kiteyttää seuraaviin kolmeen ominaisuuteen.

Määritelmä 1.1 Relaatio $R \subseteq A \times A$ on:

(i) *refleksiivinen*, jos aRa kaikilla $a \in A$;

(ii) *symmetrinen*, jos $aRb \Rightarrow bRa$ kaikilla $a, b \in A$;

(iii) *transitiivinen*, jos $aRb, bRc \Rightarrow aRc$ kaikilla $a, b, c \in A$.

Määritelmä 1.2 Relaatio $R \subseteq A \times A$, joka toteuttaa edelliset ehdot (i)–(iii), on *ekvivalenssirelaatio*. Alkion $a \in A$ *ekvivalenssiluokka* (relaation R suhteen) on

$$R[a] = \{x \in A \mid aRx\}.$$

Ekvivalenssirelaatioita merkitään usein R:n sijaan alkioiden samankaltaisuutta korostavilla symboleilla $\sim$, $\equiv$, $\simeq$ tms.

Esimerkki. Määritellään perusjoukossa $A = \{$kaikki 1900-luvulla syntyneet ihmiset$\}$ relaatio $\sim$ asettamalla ehdoksi, että $a \sim b$ on voimassa, jos ja vain jos henkilöillä a ja b on sama syntymävuosi. Tällöin $\sim$ on selvästi ekvivalenssi (tarkasta ehdot!), jonka ekvivalenssiluokat koostuvat keskenään samana vuonna syntyneistä henkilöistä. Luokkia on 100 kappaletta — olettaen että joka vuosi on syntynyt ainakin yksi ihminen — ja abstraktisti ne vastaavat 1900-luvun vuosia 1900, ... ,1999.

Lemma 1.1 *Olkoon $R \subseteq A \times A$ ekvivalenssirelaatio. Tällöin on kaikilla $a, b \in A$ voimassa:*

$$R[a] = R[b] \quad \textit{joss} \quad aRb.$$

Todistus. Olkoot a ja b mielivaltaisia perusjoukon A alkioita. Oletetaan ensin, että $R[a] = R[b]$. Relaation R refleksiivisyyden nojalla on aina voimassa ehto bRb, joten määritelmän 1.2 mukaan on $b \in R[b] = R[a]$, ja siis myös ehto aRb on voimassa.

Väitteen toisen suunnan osoittamiseksi oletetaan, että ehto aRb on voimassa. Relaation R symmetrisyyden nojalla on tällöin voimassa myös bRa. Osoitetaan, että voimassa on sisältyvyys $R[a] \subseteq R[b]$. Vaihtamalla päättelyssä a:n ja b:n roolit nähdään, että tällöin on myös $R[b] \subseteq R[a]$, ja siten itseasiassa $R[a] = R[b]$.

Olkoon siis $x \in R[a]$ jokin ekvivalenssiluokan $R[a]$ alkio. Määritelmän 1.2 mukaan on tällöin aRx, ja koska bRa, niin R:n transitiivisuuden nojalla myös bRx, so. $x \in R[b]$. Koska em. päättely on voimassa mielivaltaisella $x \in R[a]$, on näin ollen $R[a] \subseteq R[b]$. $\square$

Lause 1.2 *Olkoon $R \subseteq A \times A$ ekvivalenssirelaatio. Tällöin R:n ekvivalenssiluokat muodostavat A:n* osituksen *erillisiin epätyhjiin osajoukkoihin, so.:*

(i) $R[a] \neq \emptyset$ *kaikilla* $a \in A$;

(ii) $A = \bigcup_{a \in A} R[a]$;

(iii) jos $R[a] \neq R[b]$, *niin* $R[a] \cap R[b] = \emptyset$, *kaikilla* $a, b \in A$.

Todistus.

(i) Selvä, koska $a \in R[a]$.

(ii) Selvä, koska

$$A = \bigcup_{a \in A} \{a\} \subseteq \bigcup_{a \in A} R[a] \subseteq A.$$

(iii) Olkoot $a, b \in A$ sellaiset, että $R[a] \cap R[b] \neq \emptyset$. Osoitetaan, että tällöin on $R[a] = R[b]$. Olkoon nimittäin c jokin leikkauksen $R[a] \cap R[b]$ alkio. Määritelmän 1.2 mukaan on tällöin voimassa aRc ja bRc. Lemman 1.1 nojalla tästä seuraa, että $R[a] = R[c] = R[b]$. $\square$

Kääntäen jokainen perusjoukon A ositus erillisiin epätyhjiin luokkiin A_i, $i \in I$, määrää vastaavan ekvivalenssirelaation:

$$a \sim b \quad \Leftrightarrow \quad a \text{ ja } b \text{ kuuluvat samaan luokkaan } A_i.$$

1.4 * Järjestysrelaatiot

Kuten edellä samankaltaisuuden idea, voidaan moninaiset matematiikassa esiintyvät olioiden "järjestykset" kiteyttää seuraavasti:

Määritelmä 1.3 Relaatio $R \subseteq A \times A$ on *antisymmetrinen*, jos kaikilla $a, b \in A$ on voimassa ehto $aRb, bRa \Rightarrow a = b$.

Määritelmä 1.4 Relaatio $R \subseteq A \times A$, joka on refleksiivinen, antisymmetrinen ja transitiivinen, on joukon A *(osittainen) järjestys* (engl. (partial) order). Järjestysrelaatioita merkitään usein R:n sijaan symboleilla $\leq$, $\preceq$ tms. Jos perusjoukossa A on määritelty järjestys $\preceq$, sanotaan että A on *järjestetty joukko*. Jos valittu järjestys ei ole yhteydestä selvä, käytetään pelkän A:n sijaan pitempää merkintää $(A, \preceq)$.

Jos järjestetyn joukon $(A, \preceq)$ alkioilla $a, b \in A$ on voimassa $a \preceq b$ tai $b \preceq a$, sanotaan että a ja b ovat *vertailtavia* (engl. comparable). Jos edelleen kaikki perusjoukon A alkiot ovat (pareittain) vertailtavia, järjestys $\preceq$ on *täydellinen* (kokonainen, lineaarinen) (engl. complete, total, linear order), tai lyhyesti *täysjärjestys* (kokonaisjärjestys, lineaarijärjestys).
Järjestetyn joukon $(A, \preceq)$ alkio $a \in A$ on:

(i) *maksimaalinen*, jos $a \preceq x \Rightarrow a = x$ kaikilla $x \in A$;

(ii) *minimaalinen*, jos $x \preceq a \Rightarrow a = x$ kaikilla $x \in A$;

(iii) *suurin alkio*, jos $x \preceq a$ kaikilla $x \in A$;

(iv) *pienin alkio*, jos $a \preceq x$ kaikilla $x \in A$.

Järjestetty joukko $(A, \preceq)$ on *hyvin järjestetty* (engl. well-ordered) t. *hyvinjärjestys*, jos jokainen epätyhjä $B \subseteq A$ sisältää järjestyksen $\preceq$ suhteen pienimmän alkion.

Lemma 1.3 *Jokainen hyvinjärjestys on täydellinen.*

Todistus. Tarkastellaan pareja $\{a, b\}$. $\square$

Esimerkkejä.

(i) Luonnollisten lukujen joukko $\mathbb{N}$ varustettuna tavanomaisella lukujen suuruusjärjestyksellä, so. struktuuri $(\mathbb{N}, \leq)$ on hyvinjärjestys.

(ii) Sen sijaan kaikkien kokonaislukujen suuruusjärjestys, so. struktuuri $(\mathbb{Z}, \leq)$ on kyllä täydellinen, mutta ei hyvinjärjestys.

(iii) Olkoon X mielivaltainen joukko. Tällöin struktuuri $(\mathcal{P}(X), \subseteq)$ on järjestys, mutta ei täydellinen jos $|X| \geq 2$. (Vrt. kuva 1.2.)

iv) Olkoot $m, n \in \mathbb{N}$. Merkitään:

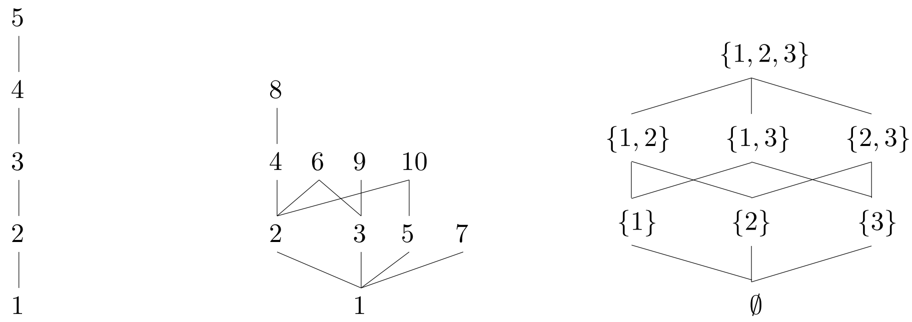
$$m \mid n \Leftrightarrow m \text{ on } n\text{:n tekijä.}$$

Struktuuri $(\mathbb{N}, |)$ on järjestys, mutta ei täydellinen.

Olkoon $(A, \preceq)$ järjestetty joukko. Määritellään lyhennemerkinnät:

$$\begin{aligned} a \prec b &\Leftrightarrow a \preceq b, a \neq b \\ a \succeq b &\Leftrightarrow b \preceq a \\ a \succ b &\Leftrightarrow a \succeq b, a \neq b. \end{aligned}$$

Alkio $a \in A$ on alkion $b \in A$ *välitön edeltäjä* ja b on a:n *välitön seuraaja*, jos:



$(\{1, 2, \ldots, 5\}, \leq)$ $\quad$ $(\{1, 2, \ldots, 10\}, |)$ $\quad$ $(\mathcal{P}(\{1, 2, 3\}), \subseteq)$

Kuva 1.3: Kolmen järjestyksen Hasse-kaaviot.

(i) $a \prec b$ ja

(ii) millään alkiolla $c \in A$ ei ole $a \prec c \prec b$.

Jokainen äärellinen järjestetty joukko $(A, \preceq)$ voidaan esittää ns. *Hasse-kaaviona*, jonka solmut vastaavat A:n alkioita, ja solmusta $a \in A$ on viivat "ylöspäin" a:n kaikkiin välittömiin seuraajiin. Kuvassa 1.3 on tästä joitakin esimerkkejä.

Järjestysrelaation Hasse-kaavio voidaan nähdä myös relaation graafiesityksen "transitiivisena reduktiona", missä graafista on selkeyden vuoksi jätetty pois ne kaaret, joiden olemassaolo voidaan päätellä graafissa esitettyjen kaarten ja tarkasteltavan relaation refleksiivisyyden ja transitiivisuuden nojalla.

1.5 Induktioperiaate

Lause 1.4 *Olkoon $(A, \preceq)$ hyvin järjestetty joukko ja $P(a)$ jokin A:n alkioita koskeva väite. Jos voidaan osoittaa kaikilla $a \in A$ induktio-ominaisuus:*

$$[P(x) \textit{ tosi kaikilla } x \prec a] \Rightarrow P(a) \textit{ tosi}, \tag{$*$}$$

niin väite $P(a)$ on tosi kaikilla $a \in A$.

Todistus. Oletetaan, että ominaisuus $(*)$ on voimassa, mutta silti joukko

$$B = \{a \in A \mid P(a) \text{ on epätosi } \}$$

on epätyhjä. Koska A on hyvin järjestetty, joukossa B on järjestyksen $\preceq$ suhteen pienin alkio $b \in B$. Mutta tällöin on voimassa:

$$[P(x) \text{ tosi kaikilla } x \prec b],$$

joten oletuksen $(*)$ mukaan pitäisi olla myös $P(b)$ tosi. Saadusta ristiriidasta seuraa, että on oltava $B = \emptyset$, ja siis $P(a)$ tosi kaikilla $a \in A$. $\square$

Seuraus 1.5 (Luonnollisten lukujen vahva induktio) *Olkoon $P(k)$ jokin luonnollisten lukujen ominaisuus. Jos on voimassa:*

(i) $P(0)$ *ja*

(ii) kaikilla $k \geq 0$*:* $[P(0) \& P(1) \& \ldots \& P(k)] \Rightarrow P(k+1)$*,*

niin $P(n)$ *on tosi kaikilla* $n \in \mathbb{N}$. $\square$

Seuraus 1.6 (Luonnollisten lukujen heikko induktio) *Olkoon* $P(k)$ *jokin luonnollisten lukujen ominaisuus. Jos on voimassa:*

(i) $P(0)$ *ja*

(ii) kaikilla $k \geq 0$*:* $P(k) \Rightarrow P(k+1)$*,*

niin $P(n)$ *on tosi kaikilla* $n \in \mathbb{N}$. $\square$

Esimerkkinä induktioperiaatteen 1.6 soveltamisesta todistetaan oikeaksi seuraava:

Väite. Kaikilla $n \in \mathbb{N}$ on voimassa kaava

$$P(n): \quad (1+2+\cdots+n)^2 = 1^3+2^3+\cdots+n^3.$$

Todistus.

(i) *Perustapaus:* $P(0): \quad 0^2 = 0.$

(ii) *Induktioaskel:* Oletetaan, että annetulla $k \geq 0$ kaava

$$P(k): \quad (1+2+\cdots+k)^2 = 1^3+2^3+\cdots+k^3$$

on voimassa. Tällöin on myös:

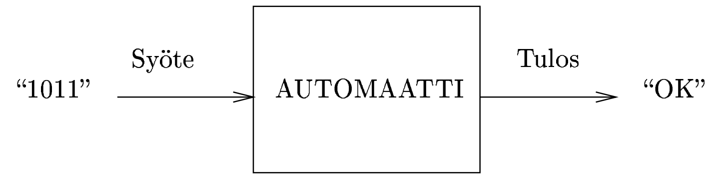
$$\begin{aligned}
(1+2+\cdots+k+(k+1))^2 &= (1+\cdots+k)^2 + 2\cdot(1+\cdots+k)\cdot(k+1)+(k+1)^2 \\
&= (1^3+\cdots+k^3) + 2\cdot\frac{1}{2}k(k+1)\cdot(k+1)+(k+1)^2 \\
&= (1^3+\cdots+k^3) + k(k+1)^2+(k+1)^2 \\
&= (1^3+\cdots+k^3) + (k+1)\cdot(k+1)^2 \\
&= 1^3+\cdots+k^3+(k+1)^3.
\end{aligned}$$

On siis todettu, että kaavan $P(k)$ totuudesta seuraa kaavan $P(k+1)$ totuus, so. että $P(k) \Rightarrow P(k+1)$, kaikilla $k \geq 0$. Luonnollisten lukujen induktioperiaatteen 1.6 nojalla voidaan nyt päätellä, että kaava $P(n)$ on voimassa kaikilla $n \in \mathbb{N}$. $\square$

1.6 Automaatit, aakkostot, merkkijonot ja kielet

Automaattiteoria tarkastelee diskreetin signaalinkäsittelyn perusmalleja ja -menetelmiä, tai tietojenkäsittelytermein ilmaistuna diskreettien I/O-kuvausten (syöte/vaste-kuvausten) yleistä teoriaa. Perustilanne on kuvassa 1.4 esitetyn kaltainen: halutaan suunnitella mekanismi tietyn I/O-kuvauksen toteuttamiseen, tai kääntäen halutaan ymmärtää millaisia kuvauksia tietynlaisilla mekanismeilla voidaan toteuttaa. Yleisesti tarkasteltuna automaatin käsite on *matemaattinen abstraktio*. Matemaattisin menetelmin suunniteltu automaatti voidaan toteuttaa eri tavoin, vaikkapa sähköpiirinä, mekaanisena laitteena tai kaikkein tyypillisimmin tietokoneohjelmana.

Tässä monisteessa keskitytään pääosin automaatteihin, joiden:



Kuva 1.4: Automaatti.

(i) syötteet ovat äärellisiä, diskreettejä merkkijonoja;

(ii) vasteet ovat yksinkertaisia binääriarvoja 1/0 ("syöte OK"/"syöte ei kelpaa").

Tällaisen binäärivasteisen automaatin voidaan myös tulkita ratkaisevan jonkin syötteisiin liittyvän ***päätösongelman***: se ***hyväksyy*** syötteet, joilla se antaa vasteen 1 ja ***hylkää*** muut.[5] Eri automaatit tietenkin ratkaisevat eri ongelmia, ja jonkin ennalta-annetun päätösongelman ratkaisuautomaatin löytäminen on yleensä epätriviaali tehtävä. Itse asiassa yksi automaattiteorian tärkeistä teemoista on määrittää, minkä tyyppisillä päätösongelmilla ylipäätään *on* ratkaisuautomaatteja tietyissä automaattiluokissa.

Yleisemmin automaattiteoriassa tarkastellaan myös automaatteja, joiden syötejonot ovat äärettömiä (tällaisia malleja tarvitaan esimerkiksi jatkuvatoimisia "reaktiivisia" systeemeitä kuten käyttöjärjestelmiä, tietoliikenneohjelmistoja tai prosessinohjausjärjestelmiä tarkasteltaessa) sekä automaatteja, jotka laskevat mutkikkaampia kuin binääriarvoisia funktioita. Tässä tarkasteltavien perusmallien tuntemus muodostaa kuitenkin pohjan myös näiden rikkaampien mallien käsittelylle.

Peruskäsitteitä ja merkintöjä

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

Merkkijono (engl. string) on äärellinen järjestetty jono jonkin aakkoston merkkejä. Esimerkiksi "01001" ja "0000" ovat binääriaakkoston merkkijonoja ja "TKTP" ja "XYZZY" ovat latinalaisen aakkoston merkkijonoja. Tärkeä erikoistapaus on *tyhjä merkkijono* (engl. empty string), jossa ei ole yhtään merkkiä. Havaittavuuden parantamiseksi tyhjän merkkijonon paikka usein osoitetaan erikoismerkillä ε.

Merkkijonon x ***pituutta***, so. siihen sisältyvien merkkien määrää, merkitään $|x|$:llä. Esimerkiksi $|01001| = |XYZZY| = 5$, $|0000| = |TKTP| = 4$ ja $|\varepsilon| = 0$.

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

(i) KALA⌢KUKKO = KALAKUKKO;

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

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

[5] Määritelmään sisältyy eräs tärkeä piilo-oletus: automaatin tulkitaan "hylkäävän" myös sellaiset syötteet, joilla se ei tuota minkäänlaista vastetta esimerkiksi laskennan päättymättömyyden takia. Tämä mahdollisuus osoittautuu luvussa 6 esitettävän yleisen laskettavuusteorian kannalta aivan keskeiseksi.

(iv) kaikilla x, y, z on $(xy)z = x(yz)$;

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

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

Mielivaltaista merkkijonojoukkoa $A \subseteq \Sigma^*$ sanotaan aakkoston Σ *(formaaliksi) kieleksi* (engl. formal language).

Automaatit ja formaalit kielet

Olkoon M edellä kuvatun kaltainen binäärivasteinen automaatti, jonka syötteet ovat jonkin aakkoston Σ merkkijonoja. Merkitään automaatin M syötteellä x antamaa vastetta $M(x)$:llä. (Oletuksen mukaan on siis $M(x) \in \{0, 1\}$ kaikilla $x \in \Sigma^*$.) Automaatin M hyväksymien syötteiden joukkoa

$$A_M = \{x \in \Sigma^* \mid M(x) = 1\} \quad \subseteq \quad \Sigma^*$$

sanotaan M:n *tunnistamaksi* kieleksi (engl. language recognised by M).

Automaattiteorian yksi perusidea on, että *automaatin M rakenne heijastuu kielen A_M ominaisuuksissa.* Tai kääntäen: oletetaan, että haluttaisiin toteuttaa jokin päätösongelmatyyppinen I/O-kuvaus $\pi\colon \Sigma^* \to \{0, 1\}$. Tarkastelemalla kieltä

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

saadaan vihjeitä siitä, millainen automaatti tarvitaan kuvauksen π toteuttamiseen.

Vakiintuneita merkintöjä

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 $\Sigma = \{a_1, \dots, a_n\}$ aakkosto; tällöin $|\Sigma| = n$.
- Merkkijonot: u, v, w, x, y, ... (pieniä loppupään latinalaisia kirjaimia).
- Merkkijonojen katenaatio: $x\frown y$ tai vain xy.
- Merkkijonon pituus: $|x|$. *Esimerkkejä:*
  - (i) $|abc| = 3$;
  - (ii) olkoon $x = a_1 \dots a_m$, $y = b_1 \dots b_n$; tällöin $|xy| = m + n$.
- Tyhjä merkkijono: ε.

- 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) $|x^k| = k|x|$;
  (iii) $x^0 = \varepsilon$.
- Aakkoston Σ kaikkien merkkijonojen joukko: Σ^*.
  Esimerkki: $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$.

Merkkijonoinduktio

Automaattiteoriassa tehdään usein konstruktioita "induktiolla merkkijonon pituuden suhteen." Tämä tarkoittaa, että määritellään ensin toiminto tyhjän merkkijonon ε (tai joskus yksittäisen aakkosmerkin) tapauksessa. Sitten oletetaan, että toiminto on määritelty kaikilla annetun pituisilla merkkijonoilla u ja esitetään, miten se tällöin määritellään yhtä merkkiä pitemmillä merkkijonoilla $w = ua$.

Esimerkki. Olkoon Σ mielivaltainen aakkosto. Merkkijonon $w \in \Sigma^*$ ***käänteisjono*** (engl. reversal) w^R määritellään induktiivisesti säännöillä:

(i) $\varepsilon^R = \varepsilon$;

(ii) jos w on muotoa $w = ua$, missä $u \in \Sigma^*$, $a \in \Sigma$, niin $w^R = a\frown u^R$.

Tällaista induktiivista, tai kuten myös sanotaan "rekursiivista" määritelmää voidaan tietenkin käyttää laskujen perustana esimerkiksi seuraavaan tapaan:

$$(011)^R = 1\frown(01)^R = 1\frown(1\frown 0^R) = 11\frown(0\frown\varepsilon^R) = 110\frown\varepsilon^R = 110\frown\varepsilon = 110.$$

Tärkeämpää on kuitenkin konstruktioiden ominaisuuksien todistaminen määritelmää noudattelevalla induktiolla. Esimerkkinä todistetaan seuraava yksinkertainen käänteisjonojen ominaisuus:

Väite. Olkoon Σ aakkosto. Kaikilla $x, y \in \Sigma^*$ on voimassa $(xy)^R = y^R x^R$.

Todistus. Induktio merkkijonon y pituuden suhteen.

(i) *Perustapaus* $y = \varepsilon$. $(x\varepsilon)^R = x^R = \varepsilon^R x^R$.

(ii) *Induktioaskel.* Olkoon y muotoa $y = ua$, missä $u \in \Sigma^*$, $a \in \Sigma$. Oletetaan, että väite on voimassa merkkijonoilla x, u. Tällöin on:

$$\begin{array}{rcll} (xy)^R & = & (xua)^R & \\ & = & a\frown(xu)^R & [R\text{:n määritelmä}] \\ & = & a\frown(u^R x^R) & [\text{induktio-oletus}] \\ & = & (a\frown u^R)x^R & [\frown\text{:n liitännäisyys}] \\ & = & (ua)^R x^R & [R\text{:n määritelmä}] \\ & = & y^R x^R. \quad \square & \end{array}$$

1.7 Numeroituvat ja ylinumeroituvat joukot

Määritelmä 1.5 Joukko X on *numeroituvasti ääretön* (engl. countably infinite), jos on olemassa bijektio $f\colon \mathbb{N} \to X$. Joukko on *numeroituva* (engl. countable), jos se on äärellinen tai numeroituvasti ääretön. Joukko, joka ei ole numeroituva on *ylinumeroituva* (engl. uncountable).

Intuitiivisesti sanoen joukko X on numeroituva, jos sen alkiot voidaan järjestää ja indeksoida luonnollisilla luvuilla: joko $X = \{x_0, x_1, \ldots, x_{n-1}\}$ (jos X on n-alkioinen äärellinen joukko) tai $X = \{x_0, x_1, \ldots\}$ (jos X on numeroituvasti ääretön joukko). On helppo osoittaa (HT), että numeroituvan joukon kaikki osajoukot ovat myös numeroituvia, mutta ylinumeroituvilla joukoilla on aina sekä ylinumeroituvia että numeroituvasti äärettömiä osajoukkoja. Siten ylinumeroituvat joukot ovat jossain mielessä "isompia" kuin numeroituvat.

Lause 1.7 *Minkä tahansa aakkoston Σ merkkijonojen joukko Σ^* on numeroituvasti ääretön.*

Todistus. Muodostetaan bijektio $f\colon \mathbb{N} \to \Sigma^*$ 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* t. *leksikografisessa järjestyksessä* (engl. canonical t. lexicographic order) seuraavasti:

(i) ensin luetellaan 0:n mittaiset merkkijonot ($= \varepsilon$), 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:

$$
\begin{array}{rcl}
0 & \mapsto & \varepsilon \\
1 & \mapsto & a_1 \\
2 & \mapsto & a_2 \\
\vdots & & \vdots \\
n & \mapsto & a_n \\
n+1 & \mapsto & a_1a_1 \\
n+2 & \mapsto & a_1a_2 \\
\vdots & & \vdots \\
2n & \mapsto & a_1a_n \\
2n+1 & \mapsto & a_2a_1 \\
\vdots & & \vdots \\
3n & \mapsto & a_2a_n \\
\vdots & & \vdots \\
n^2+n & \mapsto & a_na_n \\
n^2+n+1 & \mapsto & a_1a_1a_1 \\
n^2+n+2 & \mapsto & a_1a_1a_2 \\
\vdots & & \vdots
\end{array} \qquad \square
$$

Mielenkiintoinen huomio on, että millä tahansa ohjelmointikielellä kirjoitetut ohjelmat ovat oikeastaan vain kielen perusaakkoston (esimerkiksi C-kielessä ASCII-merkistön) merkkijonoja. Lauseen 1.7 mukaan minkä tahansa aakkoston merkkijonojen joukko on numeroituvasti ääretön, joten myös millä tahansa ohjelmointikielellä mahdollisten ohjelmien joukko on numeroituva.

Seuraavan lauseen mukaan kuitenkin kaikkien formaalien kielten joukko on ylinumeroituva. Formaaleja kieliä on siis "enemmän" kuin mahdollisia tietokoneohjelmia, ja siksi *millään ohjelmointikielellä ei voida laatia tunnistusautomaatteja kaikille formaaleille kielille.* Toisin sanoen: on olemassa "periaatteessa mahdollisia" binäärivasteisia I/O-kuvauksia, joita ei voida toteuttaa tietokoneella.

Lause 1.8 *Minkä tahansa aakkoston Σ kaikkien formaalien kielten perhe on ylinumeroituva.*

Todistus (ns. Cantorin diagonaaliargumentti). Merkitään aakkoston Σ kaikkien formaalien kielten perhettä $\mathcal{P}(\Sigma^*) = \mathcal{A}$. Oletetaan, että todistettavan väitteen vastaisesti olisi olemassa kaikki Σ:n formaalit kielet kattava numerointi:

$$\mathcal{A} = \{A_0, A_1, A_2, \dots\}.$$

Olkoot Σ^*:n merkkijonot kanonisessa järjestyksessä lueteltuina $x_0, x_1, x_2, \dots$. Määritellään em. numerointeja käyttäen formaali kieli $\tilde{A}$:

$$\tilde{A} = \{x_i \in \Sigma^* \mid x_i \notin A_i\}.$$

Koska $\tilde{A} \in \mathcal{A}$ ja $\mathcal{A}$:n numerointi oletettiin kattavaksi, pitäisi olla $\tilde{A} = A_k$ jollakin $k \in \mathbb{N}$. Mutta tällöin olisi $\tilde{A}$:n määritelmän mukaan

$$x_k \in \tilde{A} \Leftrightarrow x_k \notin A_k = \tilde{A}.$$

Saadun ristiriidan takia oletus, että perhe $\mathcal{A}$ on numeroituva, ei voi pitää paikkaansa. □

Kuvallisesti todistuksen idea voidaan esittää seuraavasti. Muodostetaan kielten $A_0, A_1, A_2, \dots$ ja merkkijonojen $x_0, x_1, x_2, \dots$ "insidenssimatriisi", jonka rivin i sarakkeessa j on arvo 1 jos $x_i \in A_j$ ja muuten 0. Tällöin kieli $\tilde{A}$ poikkeaa kustakin kielestä A_k matriisin "diagonaalilla":

$$
\begin{array}{c|ccccc}
\overset{\tilde{A}}{\searrow} & A_0 & A_1 & A_2 & A_3 & \cdots \\
\hline
x_0 & \overset{1}{\cancel{0}} & 0 & 0 & 1 & \cdots \\
x_1 & 0 & \overset{0}{\cancel{1}} & 0 & 0 & \cdots \\
x_2 & 1 & 1 & \overset{0}{\cancel{1}} & 1 & \cdots \\
x_3 & 0 & 0 & 0 & \overset{1}{\cancel{0}} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

1.8 * Ekskursio: Turingin pysähtymisongelma

Edellisen kappaleen tulosten mukaan on siis olemassa formaaleja kieliä (binäärivasteisia I/O-kuvauksia, päätösongelmia), joita ei voida tunnistaa (toteuttaa, ratkaista) esimerkiksi C-ohjelmilla. Mutta onko kyseessä vain Lauseen 1.8 eksistenssitodistuksen luoma matemaattinen illuusio ("ongelma joka todistettavasti on olemassa vaikka kukaan ei ole sitä nähnyt"), vai voidaanko tämäntyyppisestä *ratkeamattomasta* päätösongelmasta antaa jokin konkreettinen, mielenkiintoinen esimerkki?

Ensimmäisen ja tunnetuimman tällaisen esimerkin esitti brittimatemaatikko Alan Turing jo vuonna 1936. Ongelma koskee syötteenä annettujen, mielivaltaisten ohjelmatekstien pysähtymistestausta. C-kielen formalismia käyttäen tämä ns. *Turingin pysähtymisongelma* voidaan muotoilla seuraavasti.

Väite. Ei ole olemassa C-funktiota `halt(p,x)`, joka saa syötteenään mielivaltaisen C-funktion tekstin `p` ja tälle tarkoitetun syötteen `x` ja toimii seuraavasti:

$$\texttt{halt(p,x)} = \begin{cases} 1, & \text{jos funktion } \texttt{p} \text{ laskenta syötteellä } \texttt{x} \text{ pysähtyy;} \\ 0, & \text{jos } \texttt{p}\text{:n laskenta } \texttt{x}\text{:llä ei pysähdy.} \end{cases}$$

Todistus. Oletetaan väitteen vastaisesti, että tällainen funktio `halt` voitaisiin laatia. Muodostetaan tätä käyttäen toinen funktio `confuse`:[6]

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

Merkitään edellä kuvattua funktion `confuse` ohjelmatekstiä `c`:llä ja tarkastellaan funktion `confuse` laskentaa tällä omalla kuvauksellaan. Saadaan ristiriita:

$$\begin{array}{lll} \texttt{confuse(c)} \text{ pysähtyy} & \Leftrightarrow \texttt{halt(c,c)} == 1 & [\text{funktion } \texttt{halt} \text{ määritelmä}] \\ & \Leftrightarrow \texttt{confuse(c)} \text{ ei pysähdy} & [\text{funktion } \texttt{confuse} \text{ määritelmä}]. \end{array}$$

Ristiriidasta seuraa, että oletettua pysähtymistestausfunktiota `halt` ei voi olla olemassa. □

Monisteen luvussa 6 tullaan näkemään, että tämäntapaiset ratkeamattomat ongelmat ovat itse asiassa tietojenkäsittelytehtävissä hyvin tavallisia.

[6] Tässä on oletettu yksinkertaisuuden vuoksi, että C-funktioiden rungot voisivat sisältää toisten C-funktioiden määrittelyjä. Tarkkaan ottaen ANSI-standardin mukainen C-kielen syntaksi ei salli tätä, mutta todistuksen muotoilu täsmälleen ANSI-C:n vaatimusten mukaiseksi hämärtäisi sen keskeisen idean. Jotkin C-kääntäjät, esimerkiksi GNU C, sallivat tässä käytetyn standardin laajennuksen.

Luku 2

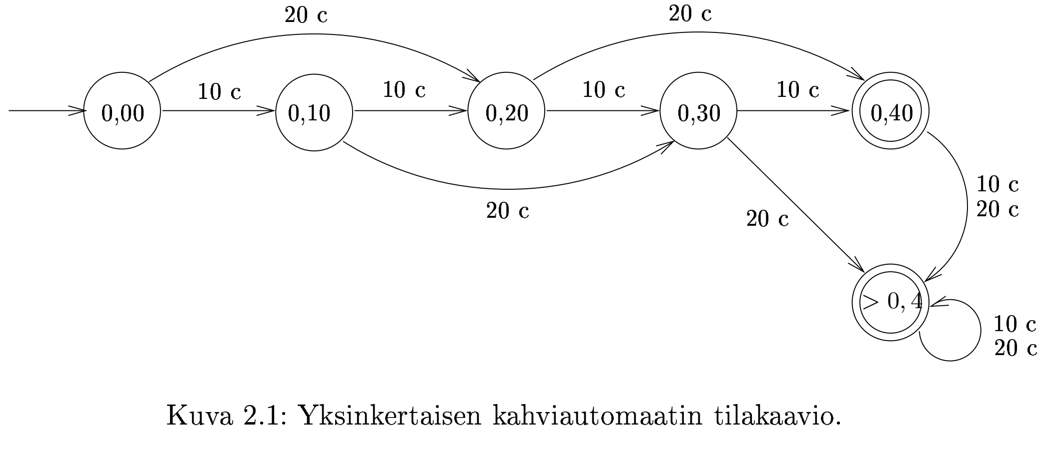
Äärelliset automaatit ja säännölliset kielet

2.1 Tilakaaviot ja tilataulut

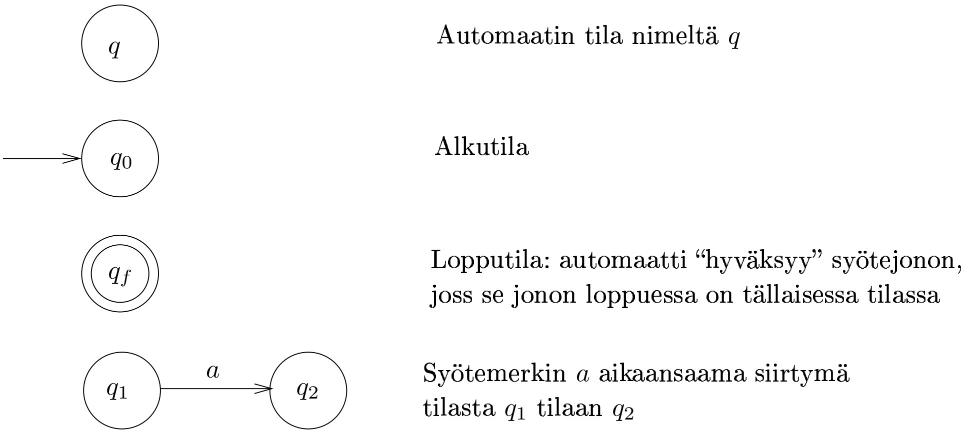
Tässä monisteen toisessa luvussa tarkastellaan sellaisten yksinkertaisten tietojenkäsittelyjärjestelmien kuvaamista ja ominaisuuksia, joilla on vain äärellisen monta mahdollista tilaa. Tällaisen järjestelmän toiminta voidaan kuvata *äärellisenä automaattina* t. *äärellisenä tilakoneena* (engl. finite automaton, finite state machine). Äärellisillä automaateilla puolestaan on useita vaihtoehtoisia esitystapoja, joista *tilakaaviot* (engl. state transition diagrams) lienee havainnollisin.

Esimerkiksi kuvassa 2.1 on esitetty yksinkertaisen, neljänkymmenen sentin hintaista kahvia tarjoavan kahviautomaatin toimintaa kuvaava tilakaavio. Kaavioesityksessä käytettyjä merkintöjä on selostettu kuvassa 2.2. Automaatti ottaa vastaan syötteenä jonon 10 ja 20 sentin rahoja, ja "hyväksyy" syötejonon, jos siihen sisältyvien rahojen summa on vähintään neljäkymmentä senttiä. Automaatti ei anna vaihtorahaa ja tarjoilee vain yhdenlaista kahvia.

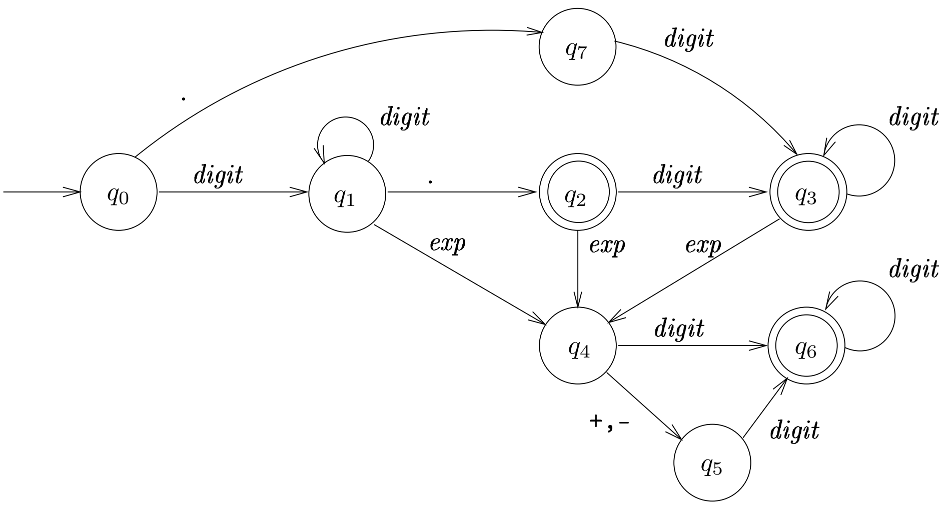
Päätösongelmanäkökulmasta voidaan ajatella, että kuvan 2.1 automaatti ratkaisee ongelman "riittävätkö annetut rahat kahvin ostamiseen?" Äärellisiä automaatteja voidaan yleensäkin käyttää yksinkertaisten päätösongelmien ratkaisujen mallintamiseen. Automaattimallista on muitakin kuin binäärivasteisten järjestelmien kuvaamiseen tarkoitettuja versioita (ns. Moore- ja Mealy-automaatit), mutta niitä ei käsitellä tässä monisteessa.



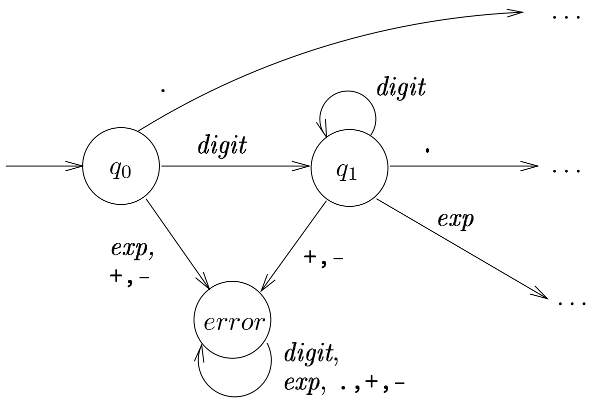
Kuva 2.1: Yksinkertaisen kahviautomaatin tilakaavio.



Kuva 2.2: Tilakaavioiden merkinnät.



Kuva 2.3: C-kielen etumerkittömät reaaliluvut tunnistava automaatti.



Kuva 2.4: Äärellisen automaatin varustaminen virhetilalla.

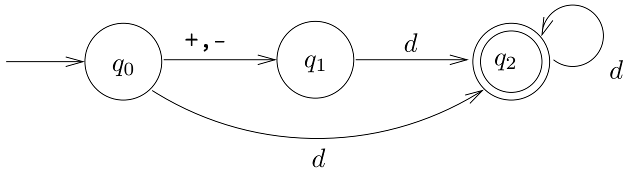
Toisena esimerkkinä tarkastellaan kuvassa 2.3 esitettyä automaattia, joka tutkii, onko syötteenä annettu merkkijono C-kielen syntaksin mukainen etumerkitön reaaliluku (C-tyyppi "unsigned double"). Automaatin esityksessä on käytetty lyhennettä *digit* merkkijoukolle $\{\texttt{0}, \texttt{1}, \ldots, \texttt{9}\}$ ja lyhennettä *exp* merkkijoukolle $\{\texttt{E}, \texttt{e}\}$.

Äärellinen automaatti voidaan esittää myös *tilatauluna* (engl. state transition table), joka kuvaa automaatin uuden tilan vanhan tilan ja syötemerkin funktiona. Esimerkiksi kuvan 2.3 automaatin tilatauluesitys olisi:

		digit	.	*exp*	+	−
$\rightarrow$	q_0	q_1	q_7			
	q_1	q_1	q_2	q_4		
$\leftarrow$	q_2	q_3		q_4		
$\leftarrow$	q_3	q_3		q_4		
	q_4	q_6			q_5	q_5
	q_5	q_6				
$\leftarrow$	q_6	q_6				
	q_7	q_3				

Tilataulun tyhjät paikat, tai vastaavasti tilakaavion "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 tauluesitys seuraavanlainen:

		digit	.	*exp*	+	−
$\rightarrow$	q_0	q_1	q_7	*error*	*error*	*error*
	q_1	q_1	q_2	q_4	*error*	*error*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\leftarrow$	q_6	q_6	*error*	*error*	*error*	*error*
	error	*error*	*error*	*error*	*error*	*error*



Kuva 2.5: Etumerkilliset kokonaisluvut tunnistava automaatti.

2.2 Äärellisiin automaatteihin perustuva ohjelmointi

Annetun äärellisen automaatin pohjalta on helppo laatia automaatin toimintaa vastaava ohjelma. Esimerkiksi kuvan 2.3 automaatin perusteella voitaisiin laatia seuraavanlainen C-ohjelma sen testaamiseksi, onko syötteenä annettu merkkijono C-kielen syntaksin mukainen etumerkittömän reaaliluvun esitys:

```
#include <stdio.h>
#include <ctype.h>

int main(void) {
  int q, c;
  q = 0;
  while ((c = getchar()) != '\n') { /* Funktio getchar() palauttaa   */
    switch (q) {                    /* syötevirran seuraavan merkin. */
    case 0:
      if (isdigit(c)) q = 1;        /* Funktio isdigit(c) testaa,    */
      else if (c == '.') q = 7;     /* onko syötemerkki numero.      */
      else q = 99;
      break;
    case 1:
      if (isdigit(c)) q = 1;
      else if (c == '.') q = 2;
      else if (c == 'E' || c == 'e') q = 4;
      else q = 99;
      break;
    ...
    case 99:
      break;
    }
    if (q == 2 || q == 3 || q == 6)
      printf("SYÖTE ON REAALILUKU.\n");
    else
      printf("SYÖTE EI OLE REAALILUKU.\n");
}
```

Äärellisten automaattien pohjalta laadittuihin ohjelmiin voidaan liittää "syntaktisen" merkkijonon oikeellisuuden testaamisen rinnalle myös "semanttisia" toimintoja. Laaditaan tämän tekniikan sovelluksena yksinkertainen ohjelma, joka muuttaa syötteenä annetun luvun kahdeksanjärjestelmästä kymmenjärjestelmään, tai tarkemmin sanoen laskee ja tulostaa syötteenä annetun kahdeksanjärjestelmän luvun lukuarvon kymmenjärjestelmässä.

Ohjelman suunnittelun lähtökohtana on kuvan 2.5 mukainen, etumerkillisiä kahdeksanjärjestelmän kokonaislukuesityksiä tunnistava äärellinen automaatti. (Kuvassa on käytetty lyhennysmerkintää d merkkijoukolle $\{0, 1, \ldots, 7\}$.) Tämä automaatti voidaan toteuttaa C-ohjelmana edellisen esimerkin tapaan:

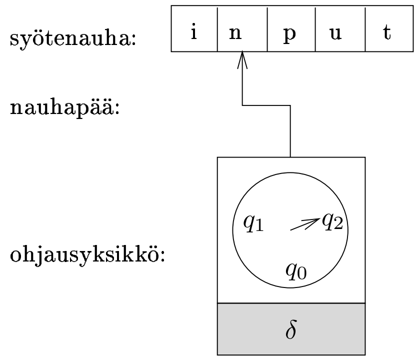
```
#include <stdio.h>

int main(void)  {
  int q, c;
  q = 0;
  while ((c = getchar()) != '\n') {
    switch (q) {
    case 0:
      if (c == '+' || c == '-') q = 1;
      else if ('0' <= c && c <= '7') q = 2;
      else q = 99;
      break;
    case 1:
      if ('0' <= c && c <= '7') q = 2;
      else q = 99;
      break;
    case 2:
      if ('0' <= c && c <= '7') q = 2;
      else q = 99;
      break;
    case 99:
      break;
    }
  }
  if (q == 2)
  {
    printf("SYÖTE OK.\n");
    exit(0);
  }
  else
  {
    printf("VIRHEELLINEN KAHDEKSANJÄRJESTELMÄN LUKU.\n");
    exit(1);
  }
}
```

Tähän toteutukseen voidaan nyt helposti liittää syötteenä saadun luvun (desimaali)arvon laskevat operaatiot (merkitty seuraavassa kommentilla `/* SEM */`):

```
#include <stdio.h>

int main(void)  {
  int q, c;
  int sgn, val;          /* SEM: sgn = etumerkki, val = luvun itseisarvo */
  sgn = 1; val = 0;                            /* SEM */
  q = 0;
  while ((c = getchar()) != '\n') {
    switch (q) {
    case 0:
      if (c == '+') q = 1;
      else if (c == '-') {
        sgn = -1;                              /* SEM */
        q = 1;
      }
      else if ('0' <= c && c <= '7') {
        val = c - '0';                         /* SEM */
        q = 2;
      }
      else q = 99;
      break;
    case 1:
      if ('0' <= c && c <= '7') {
        val = c - '0';                         /* SEM */
        q = 2;
      }
      else q = 99;
      break;
    case 2:
      if ('0' <= c && c <= '7') {
        val = 8 * val + (c - '0');             /* SEM */
        q = 2;
      }
      else q = 99;
      break;
    case 99:
      break;
    }
  }
  if (q == 2)
  { printf("LUVUN DESIMAALIARVO ON %d.\n", sgn*val); /* SEM */
    exit(0); }
  else
  { printf("VIRHEELLINEN KAHDEKSANJÄRJESTELMÄN LUKU.\n");
    exit(1); }
}
```



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 tilakaaviot ja -taulut ovat juuri tämän siirtymäfunktion vaihtoehtoisia esitystapoja.

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

$$M = (\{q_0, \ldots, q_7, \mathit{error}\}, \{\mathtt{0}, \mathtt{1}, \ \ldots \ , \mathtt{9}, \mathtt{.}, \mathtt{E}, \mathtt{e}, \mathtt{+}, \mathtt{-}\}, \delta, q_0, \{q_2, q_3, q_6\}),$$

missä δ on kuten sivun 19 tilataulussa on esitetty; esimerkiksi

$$\begin{aligned}
&\delta(q_0, \mathtt{0}) = \delta(q_0, \mathtt{1}) = \cdots = \delta(q_0, \mathtt{9}) = q_1, \quad \delta(q_0, .) = q_7, \\
&\delta(q_0, \mathtt{E}) = \delta(q_0, \mathtt{e}) = \mathit{error}, \\
&\delta(q_1, .) = q_2, \quad \delta(q_1, \mathtt{E}) = \delta(q_1, \mathtt{e}) = q_4, \quad \text{jne.}
\end{aligned}$$

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

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

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:

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

Esimerkkinä tarkastellaan merkkijonon "$\texttt{0.25E2}$" käsittelyä kuvan 2.3 mukaisella reaalilukuautomaatilla:

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

Koska $q_6 \in F = \{q_2, q_3, q_6\}$, 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 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

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

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:

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

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:

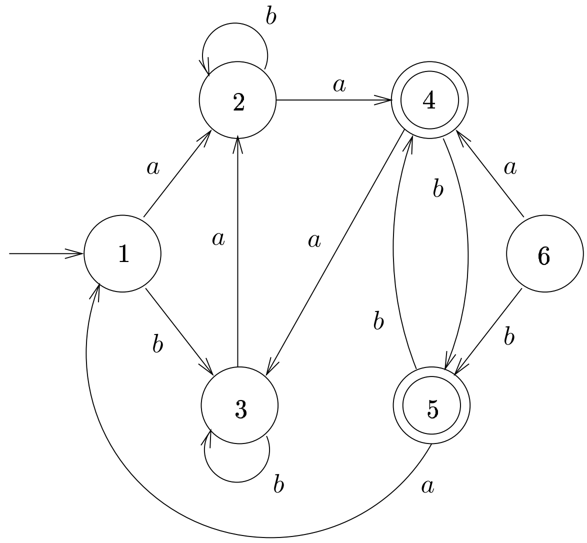
1. [Turhien tilojen poisto.] Poista M:stä kaikki tilat, joita ei voida saavuttaa tilasta q_0 millään syötemerkkijonolla.
2. [0-ekvivalenssi.] Osita M:n jäljelle jääneet tilat kahteen luokkaan: ei-lopputiloihin ja lopputiloihin.
3. [k-ekvivalenssi $\rightarrow$ $(k+1)$-ekvivalenssi.] Tarkastele M:n tilasiirtymien 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.

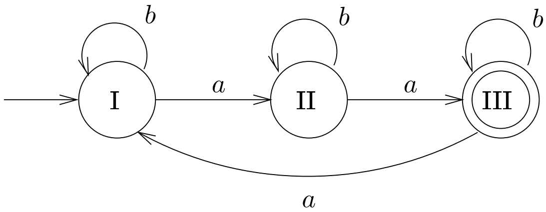
On melko helppo osoittaa, että askelen 3 $(k+1)$:nnen suorituskerran $(k = 0, 1, \dots)$ alussa kaksi tilaa kuuluu samaan muodostetun osituksen luokkaan, jos ja vain jos ne ovat k-ekvivalentteja. Tästä seuraa edelleen, että algoritmin suorituksen päättyessä, kun ositus ei enää hienone, muodostuneet tilaluokat 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 tilakaaviona 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

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

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.

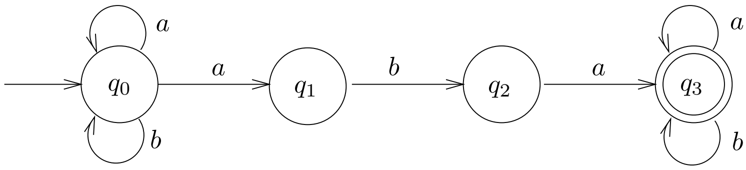
Mutta koska merkkijonot x ja y johtavat automaatin $\widetilde{M}$ samaan tilaan, se hyväksyy merkkijonon xz, jos ja vain jos se hyväksyy merkkijonon yz. Siis $\widetilde{M}$ ei olekaan ekvivalentti M:n kanssa. Saadusta ristiriidasta seuraa, että vastaoletus on väärä, ja kuvaus f on hyvin määritelty.

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] 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.

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:

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

Automaatin olisi tosin mahdollista päätyä myös hylkäävään tilaan:

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

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
$\to q_0$	$\{q_0, q_1\}$	$\{q_0\}$
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*.

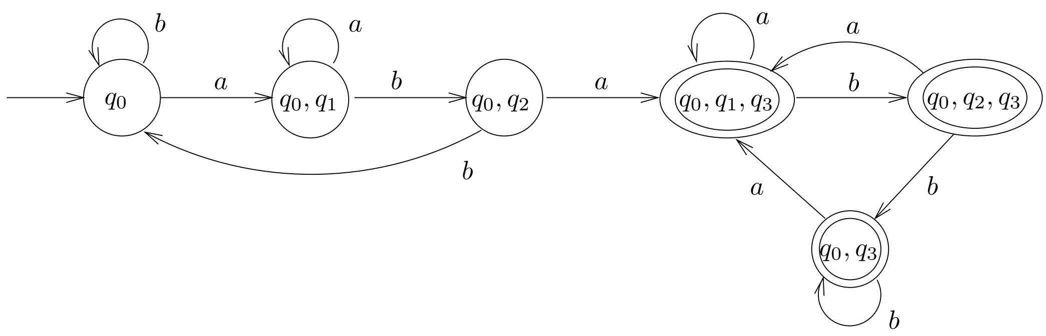
Useamman askelen mittaiset tilannejohdot, merkkijonojen hyväksyminen ja hylkääminen ym. käsitteet määritellään aivan samoin sanoin kuin aiemminkin, mutta koska perustava yhden askelen johdon määritelmä nyt on toinen, niiden sisältö muovautuu hieman erilaiseksi.

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.*

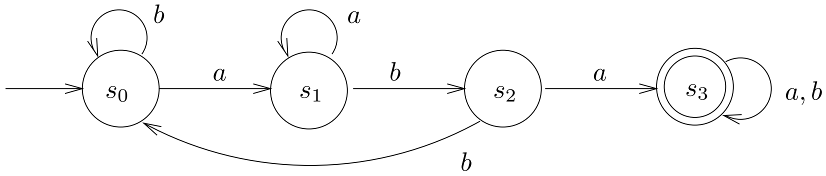
Lause 2.2 *Olkoon $A = L(M)$ jonkin epädeterministisen äärellisen automaatin M tunnistama kieli. Tällöin on olemassa myös deterministinen äärellinen automaatti $\widehat{M}$, jolla $A = L(\widehat{M})$.*

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*:



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



Kuva 2.11: Minimoitu ja uudelleennimetty deterministinen automaatti.

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

missä

$$\begin{aligned} \hat{Q} &= \mathcal{P}(Q) = \{S \mid S \subseteq Q\}, \\ \hat{q}_0 &= \{q_0\}, \\ \widehat{F} &= \{S \subseteq Q \mid S \text{ sisältää jonkin } q_f \in F\}, \\ \hat{\delta}(S, a) &= \textstyle\bigcup_{q \in S} \delta(q, a). \end{aligned} \tag{2.4}$$

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

Muodollisesti M:n determinististen 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. Käytännössä automaattia $\widehat{M}$ muodostettaessa lähdetään liikkeelle sen alkutilasta $\{q_0\}$ ja tarkastetaan ensin, mihin tiloihin tästä on siirtymiä tarkasteltavan syöteaakkoston merkeillä a ja b. Syötemerkillä a voi epädeterministinen automaatti M joko pysyä alkutilassa q_0 tai siirtyä tilaan q_1: siten automaattiin $\widehat{M}$ kirjataan saavutettava tila $\{q_0, q_1\}$ ja siihen johtava siirtymä $\hat{\delta}(\{q_0\}, a) = \{q_0, q_1\}$. Toisaalta syötemerkillä b automaatti M pysyy aina alkutilassa q_0, joten automaattiin $\widehat{M}$ kirjataan siirtymä $\hat{\delta}(\{q_0\}, b) = \{q_0\}$. (Vrt. kuva 2.10.)

Seuraavassa vaiheessa siirrytään tarkastelemaan automaatin $\widehat{M}$ saavutettavaksi todettua tilaa $\{q_0, q_1\}$ ja siihen liittyviä siirtymiä. Syötteellä a automaatti M voi siirtyä tilasta q_0 tilaan q_0 tai q_1; toisaalta tilasta q_1 automaatilla M ei ole syötteellä a mitään jatkomahdollisuuksia. Kaikkiaan saadaan siis automaattiin $\widehat{M}$ siirtymä $\hat{\delta}(\{q_0, q_1\}, a) = \{q_0, q_1\}$. Syötteellä b taas automaatti M tilasta q_0 lähtiessään pysyy siinä ja tilasta q_1 lähtiessään siirtyy aina tilaan q_2. Näitä mahdollisuuksia vastaten kirjataan automaattiin $\widehat{M}$ uusi saavutettava tila $\{q_0, q_2\}$ ja siihen johtava siirtymä $\hat{\delta}(\{q_0, q_1\}, b) = \{q_0, q_2\}$.

Tähän tapaan jatkaen, niin kauan kuin automaattista $\widehat{M}$ paljastuu uusia saavutettavia tiloja, päädytään lopulta kuvan 2.10 mukaiseen lopputulokseen. Minimoimalla automaatti luvun 2.4 menetelmällä voidaan edelleen todeta, että sen kolme lopputilaa voidaan yhdistää yhdeksi. Nimeämällä vielä minimiautomaatin tilat uudelleen saadaan kuvassa 2.11 esitetty deterministinen automaatti.

$*$ *Todistus jatkuu.* Tarkastetaan sitten yleisesti, että kaavan (2.4) kuvaamalla 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 mieleen, että määritelmien mukaan on

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

ja

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

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

$$(q_0, x) \underset{M}{\vdash}^* (q, \varepsilon) \text{ joss } (\{q_0\}, x) \underset{\widehat{M}}{\vdash}^* (S, \varepsilon) \text{ ja } q \in S. \tag{2.5}$$

Väite voidaan todistaa induktiolla merkkijonon x pituuden suhteen:

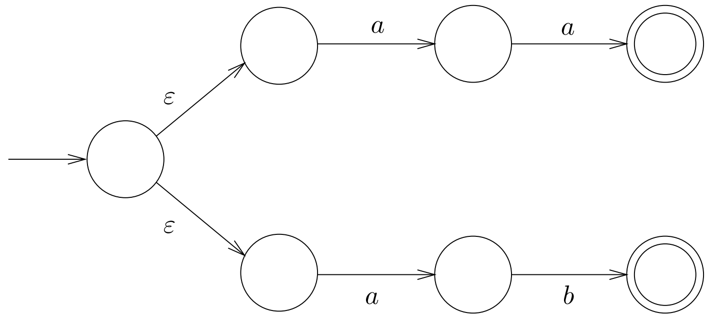
(i) Tapaus $|x| = 0$: $(q_0, \varepsilon) \underset{M}{\vdash}^* (q, \varepsilon)$ joss $q = q_0$; samoin $(\{q_0\}, \varepsilon) \underset{\widehat{M}}{\vdash}^* (S, \varepsilon)$ joss $S = \{q_0\}$.

(ii) Induktioaskel: Olkoon $x = ya$; oletetaan, että väite 2.5 pätee y:lle. Tällöin:

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

ε-automaatit

Jatkossa tarvitaan vielä sellaista epädeterminististen äärellisten automaattien mallin laajennusta, jossa automaatti voi sisältää *ε-siirtymiä*, so. siirtymiä, joissa se ei lue yhtään syötemerkkiä. Esimerkiksi kuvassa 2.12 on esitetty ε-automaatti, joka tunnistaa kielen $\{aa, ab\}$.



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

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

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

Automaattimalliin liittyvät muut määritelmät ovat samat kuin tavallisilla epädeterministisillä äärellisillä automaateilla, paitsi suoran tilannejohdon määritelmä: ε-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 $q' \in \delta(q, \varepsilon)$.

Lemma 2.3 *Olkoon $A = L(M)$ jollakin ε-automaatilla M. Tällöin on olemassa myös ε-siirtymätön epädeterministinen automaatti $\widehat{M}$, jolla $A = L(\widehat{M})$.*

Todistus. Olkoon $M = (Q, \Sigma, \delta, q_0, F)$ jokin ε-automaatti. Automaatti $\widehat{M}$ toimii muuten aivan samoin kuin M, mutta se "harppaa" ε-siirtymien yli suorittamalla kustakin tilasta lähtien vain ne "aidot" siirtymät, jotka ovat siitä käsin jotakin ε-siirtymäjonoa pitkin saavutettavissa.

Formaalisti määritellään annetun tilan $q \in Q$ ε-*sulkeuma* $\varepsilon^*(q)$ automaatissa M kaavalla

$$\varepsilon^*(q) = \{q' \in Q \mid (q, \varepsilon) \underset{M}{\vdash}^* (q', \varepsilon)\},$$

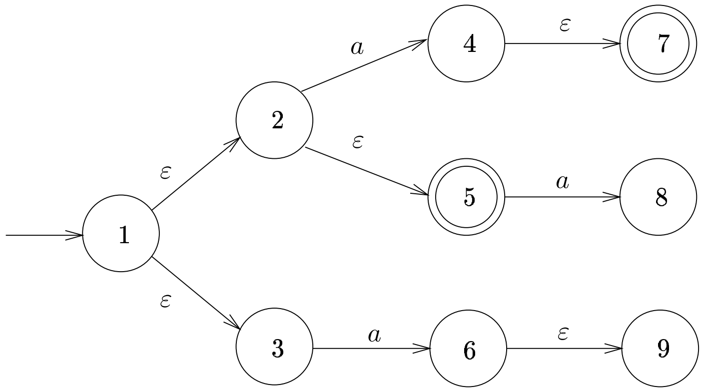
so. joukkoon $\varepsilon^*(q)$ kuuluvat kaikki ne automaatin M tilat, jotka ovat saavutettavissa tilasta q pelkillä ε-siirtymillä. Tätä merkintää käyttäen voidaan automaatin $\widehat{M}$ siirtymäsäännöt kuvata seuraavasti:

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

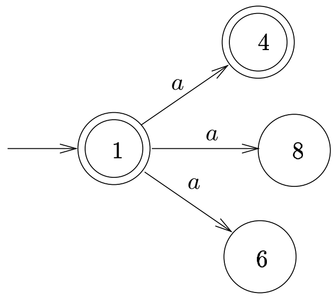
missä

$$\begin{aligned} \hat{\delta}(q, a) &= \bigcup_{q' \in \varepsilon^*(q)} \delta(q', a); \\ \widehat{F} &= \{q \in Q \mid \varepsilon^*(q) \cap F \neq \emptyset\}. \end{aligned}$$

Esimerkkinä konstruktiosta on kuvassa 2.13 esitetty eräs ε-automaatti M, ja kuvassa 2.14 vastaava epädeterministinen automaatti $\widehat{M}$, josta ε-siirtymät on poistettu em. menettelyllä. $\square$



Kuva 2.13: ε-automaatti M.



Kuva 2.14: “Aito” epädeterministinen automaatti $\widehat{M}$.

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 UN*X-käyttöjärjestelmien 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 *katenaatio* (engl. (con)catenation) 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:

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

(iv) A:n *(Kleenen) sulkeuma* t. *tähti* (engl. (Kleene) closure, star) on kieli

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

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

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

On huomattava ***tyhjän merkkijonon*** ε ja ***tyhjän kielen*** $\emptyset$ ero. Kielessä $\{\varepsilon\}$ on yksi merkkijono, nimittäin ε, se ei siis ole tyhjä: $\{\varepsilon\} \neq \emptyset$. Kleenen tähti -operaation erikoistapauksena saadaan kuitenkin:

$$\emptyset^* = \{\varepsilon\}.$$

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

(i) $\boldsymbol{\emptyset}$ ja $\boldsymbol{\varepsilon}$ 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 $(r \boldsymbol{\cup} s)$, (rs) ja $r\boldsymbol{^*}$ 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) $L(\boldsymbol{\varepsilon}) = \{\varepsilon\}$;

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

(iv) $L((r \boldsymbol{\cup} s)) = L(r) \cup L(s)$;

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

(vi) $L(r\boldsymbol{^*}) = (L(r))^*$.

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

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

Lausekkeiden kuvaamat kielet ovat:

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

Sulkumerkkien vähentämiseksi sovitaan operaattoreiden prioriteettisäännöiksi, että sulkeumaoperaattori (*) sitoo vahvemmin kuin katenaatio, joka puolestaan sitoo vahvemmin kuin yhdiste. Lisäksi huomataan, että yhdiste- ja katenaatio-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 katenaatioita 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: C-kielen etumerkittömät reaaliluvut, joita tarkasteltiin jo luvussa 2.1, voidaan kuvata lausekkeella[3]

$$number = (dd^*.d^* \cup .dd^*)(e(+ \cup - \cup \varepsilon)dd^* \cup \varepsilon) \cup (dd^*e(+ \cup - \cup \varepsilon)dd^*),$$

missä d on lyhennemerkintä lausekkeelle

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

ja e on lyhennemerkintä lausekkeelle

$$e = (\mathtt{E} \cup \mathtt{e}).$$

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: $(d^+.d^* \cup .d^+)(e(+ \cup - \cup \varepsilon)d^+ \cup \varepsilon) \cup (d^+e(+ \cup - \cup \varepsilon)d^+)$.

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:

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

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 $\varepsilon \notin L(s)$.

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 $s \subseteq r$.

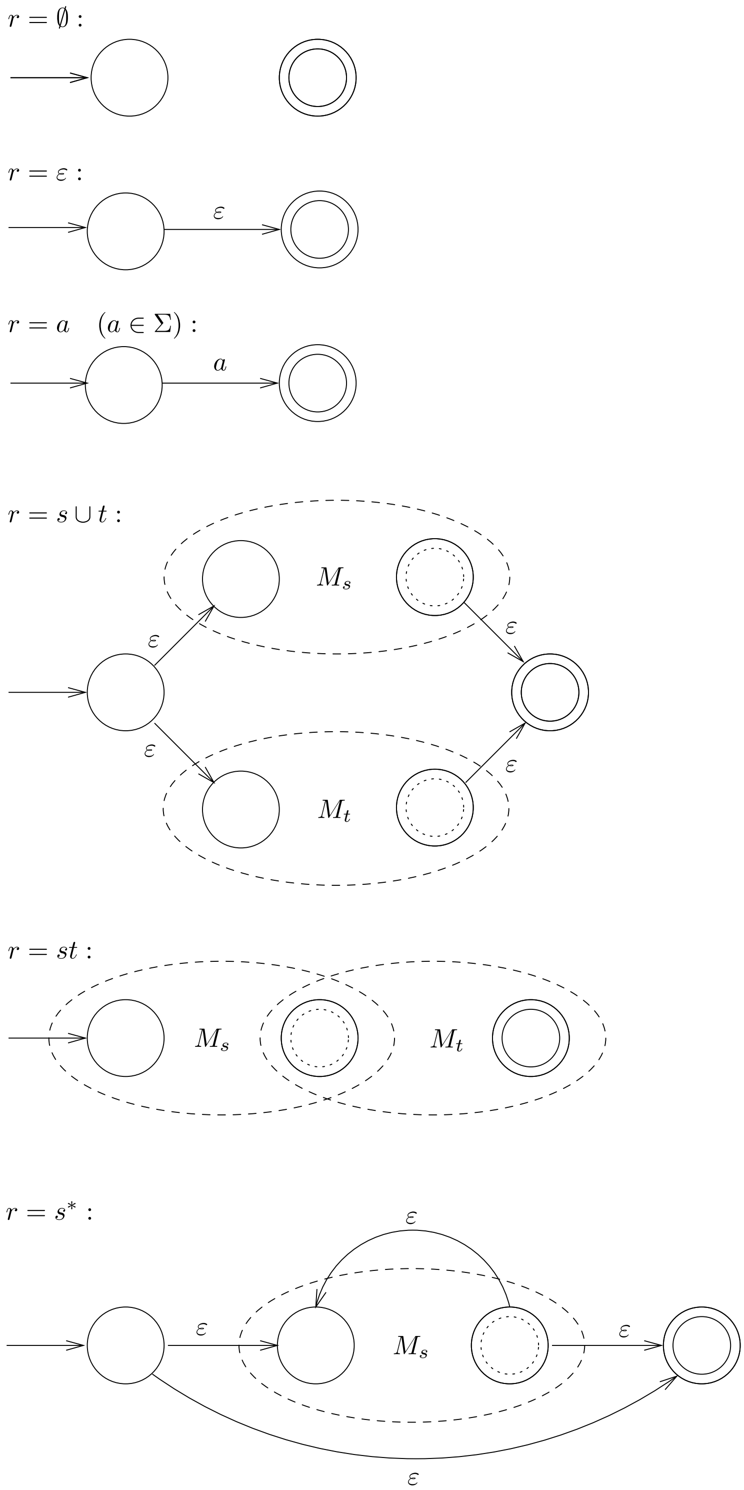
Esimerkiksi kappaleen alussa mainittujen ekvivalenssien toteamiseksi on ensinnäkin selvää, että $(a^*b^*)^* \subseteq (a \cup b)^*$ ja $a^*b^* \cup (a \cup b)^*ba(a \cup b)^* \subseteq (a \cup b)^*$, koska lauseke $(a \cup b)^*$ kuvaa *kaikkia* aakkoston $\{a, b\}$ merkkijonoja. Toisaalta, koska selvästi 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

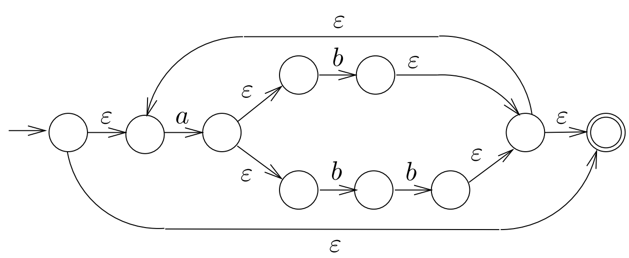
Tässä kappaleessa todistetaan säännöllisten kielten teorian perustulos: *kieli on säännöllinen, jos ja vain jos se voidaan tunnistaa äärellisellä automaatilla.* Väitteen molempien suuntien todistukset sisältävät tärkeitä konstruktioita, joten ne esitetään erikseen.

Lause 2.4 *Jokainen säännöllinen kieli voidaan tunnistaa äärellisellä automaatilla.*

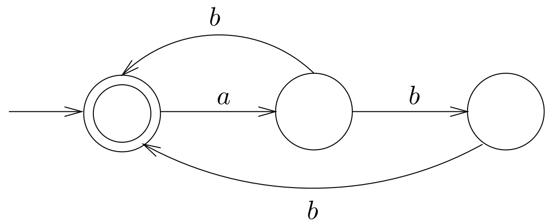
Todistus. Kuvassa 2.15 on esitetty induktiivinen konstruktio, jonka avulla voidaan mielivaltaisen säännöllisen lausekkeen r rakennetta seuraten muodostaa ε-automaatti M_r, jolla $L(M_r) = L(r)$. Tästä automaatista voidaan sitten poistaa ε-siirtymät lemmassa 2.3 esitetyllä



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



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



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

tavalla, ja tarvittaessa voidaan näin syntyvä epädeterministinen automaatti edelleen determinisoida lauseen 2.2 konstruktiolla. Kuvan 2.15 konstruktion oikeellisuudesta vakuuttautumiseksi on syytä huomata, että siinä muodostettavissa ε-automaateissa on aina yksikäsitteiset alku- ja lopputila, ja ettei lauseketta r vastaavan automaatin M_r lopputilasta lähde eikä alkutilaan tule yhtään automaatin M_r sisäistä siirtymää.

Esimerkiksi lausekkeesta $r = (a(b \cup bb))^*$ näiden sääntöjen mukaan muodostettu ε-automaatti on esitetty kuvassa 2.16. Automaatti on selvästi hyvin redundantti; ε-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 ε-siirtymätön automaatti.

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

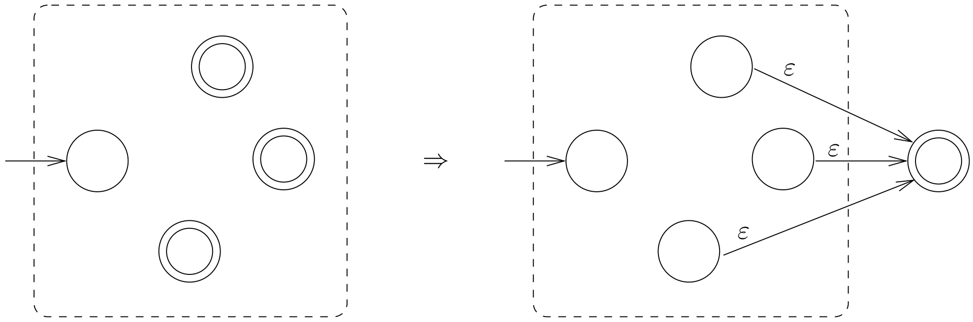
Todistus. Määritellään vielä yksi äärellisten automaattien laajennus: *lausekeautomaatissa* voidaan siirtymien ehtoina käyttää mielivaltaisia säännöllisiä lausekkeita.

Vaikka tämä yleistys ei ole käsitteellisesti juuri sen vaikeampi kuin ε-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

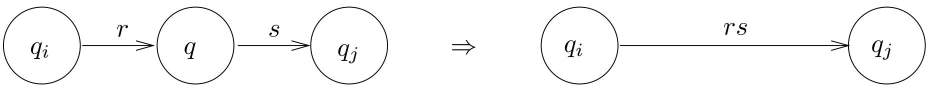
$$\delta : Q \times \mathrm{RE}_\Sigma \to \mathcal{P}(Q)$$

[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.

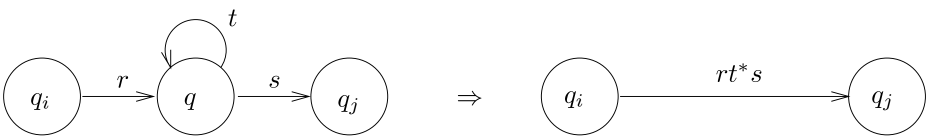


Kuva 2.18: Lausekeautomaatin lopputilojen yhdistäminen.

(i):



(ii):



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:

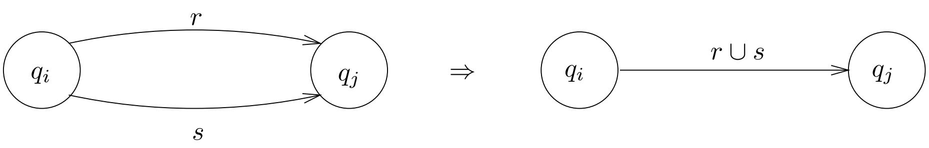
$$(q, w) \underset{M}{\vdash} (q', w')$$

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.

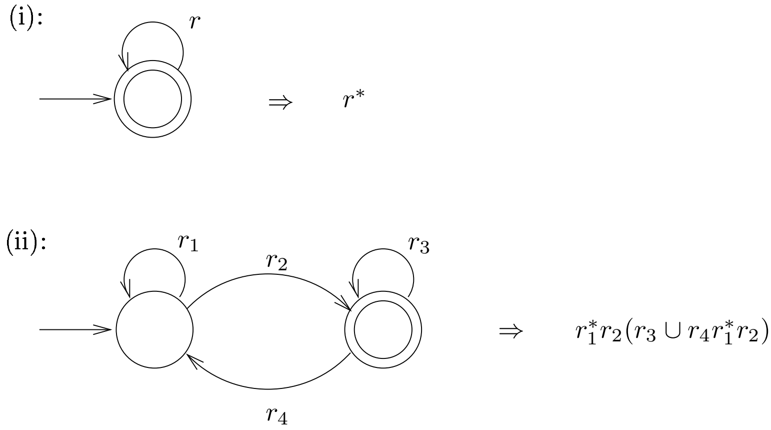
Todistetaan vaadittua tulosta näennäisesti vahvempi väite: *jokainen lausekeautomaatilla tunnistettava kieli on säännöllinen.*

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 välitön edeltäjä- ja seuraajatila jollakin tällaisella reitillä (mahdollisesti on $q_i = q_j$). Poistetaan q reitiltä $q_i \to q_j$ 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. □

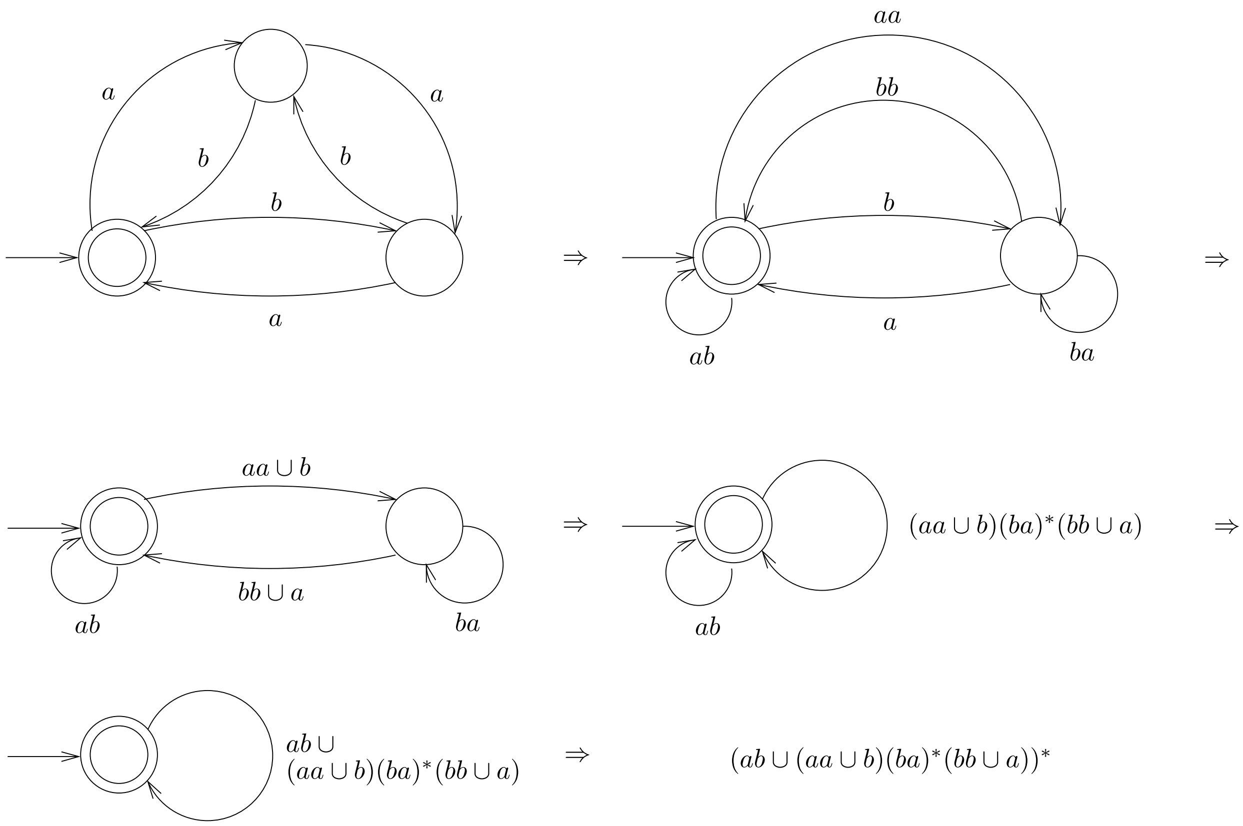
Kuvassa 2.22 on esimerkki konstruktion soveltamisesta.

2.8 Säännöllisten kielten rajoituksista

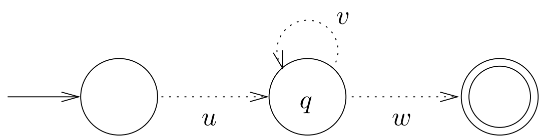
Koska minkä tahansa aakkoston formaaleja kieliä on ylinumeroituva ja säännöllisiä lausekkeita vain numeroituva määrä (lauseet 1.7 ja 1.8), eivät kaikki kielet mitenkään voi olla säännöllisiä. Miltä ei-säännölliset kielet siis “näyttävät”?

Esimerkkejä ei-säännöllisistä kielistä on itse asiassa 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, rakenteiset lauseet) eivät.

Säännöllisten kielten perusrajoitus aiheutuu siitä, että äärellisillä automaateilla on vain rajallinen “muisti”. Siten ne eivät (likimäärin sanoen) pysty ratkaisemaan ongelmia, joissa vaaditaan mielivaltaisen suurten lukujen tarkkaa muistamista. Äärellisillä automaateilla ei



Kuva 2.22: Säännöllisen lausekkeen muodostaminen äärellisestä automaatista.



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

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 $uv^\ell w \in A$ kaikilla $\ell = 0, 1, 2, \ldots$*

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 u:ta seuraava 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 $uv^\ell w \in A$ kaikilla $\ell = 0, 1, 2, \ldots$ □

Pumppauslemma on perustyökalu kielten ei-säännöllisyyden osoittamiseen. Esimerkkinä sen 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 $uv^0 w = a^i a^{n-(i+j)} b^n = a^{n-j} b^n$ ei kuulu kieleen L. Siten L ei voi olla säännöllinen.

Pumppauslemman käyttö formaalin kielen ei-säännöllisyyden todistamiseen vaatii huolellisuutta. Yksi tavallinen virhe on valita merkkijonosta pumpattavaksi jokin yksinkertainen, todistajalle "mieluinen" osajono. Tämä on kuitenkin väärin: pumppauslemman mukaan jokaista säännöllisen kielen A riittävän pitkää merkkijonoa voidaan pumpata *jostakin* kohden sen ensimmäisten n merkin alueelta. Jos siis halutaan todistaa, että A *ei* ole säännöllinen, pitää osoittaa, että jotakin vähintään n-merkkistä merkkijonoa ei voida pumpata *mistään* sen n-merkkisen alkuosan kohdasta.

Esimerkiksi edellä kielen $L = L_{\mathrm{match}}$ tapauksessa ei voitaisi päättää valita pumpattavaksi jonon $a^n b^n$ ensimmäistä a-merkkiä ja päätellä, että koska $a^n b^n \in L$ ja $a^{n+1} b^n \notin L$, niin L ei ole säännöllinen. Pätevä todistus todella vaatii yllä esitetyn yleisen tarkastelun. Em. "oikaistun" todistuksen virheellisyyden huomaa esimerkiksi tarkastelemalla kielen L sijaan säännöllisen lausekkeen $(aa)^*(bb)^*$ kuvaamaa kieltä L_1. Jos valittu n on parillinen, niin tälläkin kielellä on voimassa $a^n b^n \in L_1$ ja $a^{n+1} b^n \notin L_1$, mutta kuitenkin L_1 on määritelmänsä mukaan säännöllinen.

On syytä huomata myös, että vaikka pumpattavuus on säännöllisten kielten tyyppiominaisuus, on olemassa myös joitakin pumpattavia ei-säännöllisiä kieliä. (Ja siksi pumppauslemmaa mm. ei voida käyttää kielten *säännöllisyyden* osoittamiseen.) Esimerkiksi kieli

$$A = \{c^r a^k b^k \mid r \geq 1, k \geq 0\} \cup \{a^k b^l \mid k, l \geq 0\} \tag{2.6}$$

täyttää Lemman 2.6 ehdot — yhdisteen ensimmäiseen komponenttiin kuuluviin jonoihin voi pumpata c-merkkiä ja toiseen osaan kuuluviin a:ta tai b:tä — mutta A ei silti ole säännöllinen, kuten seuraavassa todetaan.

Tilanteet, joissa pumppauslemman suora soveltaminen kielen ei-säännöllisyyden osoittamiseen on vaikeaa tai peräti mahdotonta, voidaan usein palauttaa helpommin hallittaviksi säännöllisten kielten *sulkeumaominaisuuksien* avulla seuraavasti: oletetaan, että tarkasteltava kieli A olisi säännöllinen; päätellään tästä säännöllisten kielten tunnettujen ominaisuuksien perusteella, että jonkin A:sta johdetun kielen B tulisi tällöin myös olla säännöllinen; todetaan (esim. pumppauslemman avulla) että B ei kuitenkaan voi olla säännöllinen; saadusta ristiriidasta seuraa, että myöskään alkuperäinen kieli A ei voinut olla säännöllinen.

Esimerkiksi kaavan (2.6) määrittelemä kieli A sisältää ensimmäiseen komponenttiinsa "upotettuna" aiemmin ei-säännölliseksi todetun kielen $L = \{a^k b^k \mid k \geq 0\}$. Tätä havaintoa voidaan käyttää hyväksi, kun tiedetään että (i) kahden säännöllisen kielen leikkaus on aina säännöllinen (HT), ja että (ii) jos aakkoston $\{a, b, c\}$ kieli B on säännöllinen, niin myös kieli $B|\{a, b\}$, joka saadaan poistamalla B:n jonoista kaikki c-merkit, on säännöllinen (HT).

Jos nimittäin nyt kaavan (2.6) määrittelemä kieli A olisi säännöllinen, niin ominaisuuden (i) nojalla pitäisi myös kielen

$$B = A \cap L(cc^*a^*b^*) = \{c^r a^k b^k \mid r \geq 1, k \geq 0\}$$

olla säännöllinen, ja samoin edelleen ominaisuuden (ii) nojalla kielen

$$L = B|\{a, b\} = \{a^k b^k \mid k \geq 0\}.$$

Koska kuitenkin L tiedetään ei-säännölliseksi, seuraa saadusta ristiriidasta, ettei myöskään kieli A voi olla säännöllinen.

Luku 3

Yhteydettömät kieliopit ja kielet

3.1 Kieliopit ja merkkijonojen tuottaminen

Kieliopilla (engl. grammar) tarkoitetaan yleisesti tietynlaista muunnossysteemiä merkkijonojen (kielen "sanojen") tuottamiseen määrätystä lähtöjonosta alkaen, osajonoja toistuvasti annettujen sääntöjen mukaan uudelleenkirjoittamalla.

Kielioppi on *yhteydetön* (engl. context-free), jos kussakin uudelleenkirjoitusaskelessa korvataan yksi erityinen muuttuja- t. *välikesymboli* jollakin siihen liitetyllä korvausjonolla, ja korvaus voidaan aina tehdä symbolia ympäröivän merkkijonon rakenteesta riippumatta. (Yleisempiin kielioppeihin palataan lyhyesti luvussa 5.)

Yhteydettömillä kieliopeilla on runsaasti sovelluksia erilaisten rakenteisten tekstien (esimerkiksi ohjelmointikielten BNF-syntaksikuvaukset, XML-kuvauskielen DTD/Schema-määrittelyt) tai yleisemmin rakenteisten "olioiden" kuvaamisessa (esimerkiksi ns. syntaktisen hahmontunnistuksen menetelmissä).

Esimerkkinä kielioppikuvauksesta tarkastellaan jälleen tasapainoisten sulkujonojen muodostamaa kieltä

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

Kuten edellisen luvun lopussa todettiin, tämä kieli ei ole säännöllinen, so. sitä ei voida kuvata millään säännöllisellä lausekkeella. Kieli voidaan kuitenkin määrittää seuraavilla yksinkertaisilla muunnossäännöillä, joita toistuvasti soveltamalla voidaan mikä tahansa tasapainoinen sulkujono tuottaa lähtösymbolista S alkaen:

(i) $S \to (S)$

(ii) $S \to \varepsilon$.

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

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

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

Toisena esimerkkinä tarkastellaan kielioppia yksinkertaisen 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:[1]

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

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

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

Määritelmä 3.1 *Yhteydetön 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

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

[1]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 $\{A \to \omega_1,\ A \to \omega_2,\ \ldots A \to \omega_k\}$.

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

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

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 *yhteydetön* (engl. context-free), jos se voidaan tuottaa jollakin yhteydettömällä kieliopilla. Esimerkiksi tasapainoisten sulkujonojen muodostaman kielen $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ tuottaa kielioppi

$$G_{\text{match}} = (\{S, (,)\}, \{(,)\}, \{S \rightarrow \varepsilon, S \rightarrow (S)\}, S),$$

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

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

missä

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

Toinen kielioppi kielen L_{expr} tuottamiseen on

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

missä

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

Vaikka kielioppi G'_{expr} päällisin puolin näyttää yksinkertaisemmalta kuvaukselta kielelle L_{expr} kuin kielioppi G_{expr}, sen haittapuolena on ns. rakenteellinen moniselitteisyys joka on monissa tilanteissa ei-toivottu ominaisuus (ks. sivu 53).

Kielioppien suunnittelusta

Annetun yhteydettömän kielen kielioppikuvauksen laatimiseen ei ole mitään mekaanisesti sovellettavaa menetelmää; kielioppeja oppii kirjoittamaan tutustumalla esimerkkeihin ja harjoittelemalla. Joidenkin tyypillisten kielioppikonstruktioiden tunteminen voi kuitenkin helpottaa opiskelua.

Merkitään seuraavassa mielivaltaisesta välikkeestä T johdettavissa olevien päätejonojen joukkoa $L(T)$:llä, ja oletetaan että on jo laadittu produktiokokoelma P, joka välikkeestä B lähtien tuottaa kielen $L(B)$ ja välikkeestä C lähtien kielen $L(C)$. Oletetaan lisäksi, että P:n produktioissa ei esiinny välikettä A. Tällöin saadaan kielioppi yhdisteelle $L(B) \cup L(C)$ lisäämällä P:hen produktiot "$A \to B \mid C$", tai täsmällisemmin sanoen on täydennetyn produktiokokoelman $P' = P \cup \{A \to B \mid C\}$ suhteen voimassa $L(A) = L(B) \cup L(C)$. Vastaavasti saadaan kielioppi katenaatiolle $L(B)L(C)$ lisäämällä P:hen produktio "$A \to BC$", so. produktiokokoelmaa $P' = P \cup \{A \to BC\}$ käyttäen on voimassa $L(A) = L(B)L(C)$.

Kielen $L(B)$ Kleenen sulkeuma saadaan tuotettua lisäämällä kielioppiin joko säännöt "$A \to BA \mid \varepsilon$" (ns. ***oikea rekursio***) tai "$A \to AB \mid \varepsilon$" (***vasen rekursio***). Molemmissa tapauksissa on voimassa $L(A) = L(B)^* = \{x_1 \ldots x_k \mid k \geq 0,\ x_1, \ldots, x_k \in L(B)\}$.

Yhteydettömille kieliopeille ominainen konstruktio, joka yleensä johtaa ei-säännöllisiin kieliin, on välikkeiden ***keskeisupotus***. Esimerkiksi produktioiden "$A \to BAC \mid \varepsilon$" lisääminen produktiojoukkoon P tekee mahdolliseksi tuottaa välikkeestä A sellaiset jonot, joiden lopussa on täsmälleen yhtä monta C-muotoista osajonoa kuin alussa on B-muotoisia osajonoja:

$$L(A) = \{x_1 \ldots x_k y_k \ldots y_1 \mid k \geq 0,\ x_1, \ldots, x_k \in L(B),\ y_1, \ldots, y_k \in L(C)\}.$$

Kuten aiemmin todettiin, merkkijonojen mielivaltaisen pituisten osajonojen tarkan vastaavuuden testaaminen ei onnistu äärellisellä automaatilla, ja siten keskeisupotusta käyttäen kuvatut kielet eivät yleensä ole säännöllisiä. On kuitenkin mahdollista, että keskeisupotuksen määräämät vastaavuudet hukkuvat kieliopin tarjoamiin muihin johtomahdollisuuksiin. Näin käy esimerkiksi kieliopissa $\{S \to aSb \mid aS \mid Sb \mid \varepsilon\}$, joka keskeisupotuksestaan huolimatta tuottaa säännöllisen kielen a^*b^*. (Selkeämpi kielioppi tälle kielelle olisi $\{S \to aS \mid T,\ T \to bT \mid \varepsilon\}$.)

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:

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

  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 yhteydettömät kieliopit

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

Yhteydetön kielioppi on *oikealle lineaarinen* (engl. right linear), jos sen kaikki produktiot ovat muotoa $A \to \varepsilon$ tai $A \to aB$, ja *vasemmalle lineaarinen* (engl. left linear), jos sen kaikki produktiot ovat muotoa $A \to \varepsilon$ tai $A \to Ba$.[2] 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:

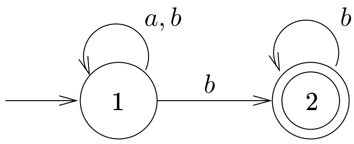
(i) kutakin M:n lopputilaa $q \in F$ kohden kielioppiin otetaan produktio $A_q \to \varepsilon$;

(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

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

[2] 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.

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

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

Erityisesti on siis

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

Esimerkiksi kuvan 3.1 automaattia vastaava kielioppi on:

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

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 ε-produktio:

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

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

3.3 Yhteydettömien kielioppien jäsennysongelma

Jotta yhteydettömistä kieliopeista kuvaustekniikkana olisi merkittävää hyötyä, olisi toivottavaa pystyä määrittämään, kuuluuko annettu merkkijono tietyn kieliopin kuvaamaan kieleen. Tarvitaan siis menetelmä seuraavan *jäsennys-* t. *tunnistusongelman* (engl. parsing problem, recognition problem) ratkaisemiseen:

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

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 yhteydettömien 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. 47) seuraavilla kolmella tavalla, ja muillakin:

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

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

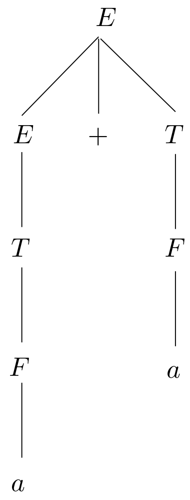
$$\gamma \underset{\text{lm}}{\Rightarrow}^* \gamma',$$

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 $G = (V, \Sigma, P, S)$ yhteydetön 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 $V \cup \{\varepsilon\}$ 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:

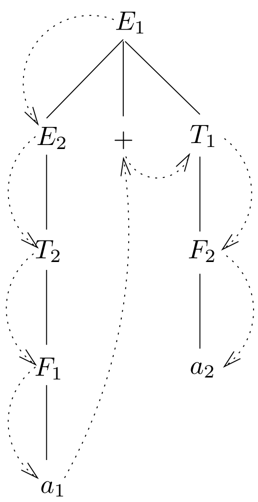
(i) puun juuren nimeksi tulee S; jos $n = 0$, niin puussa ei ole muita solmuja; muuten

(ii) jos ensimmäisessä johtoaskelessa on sovellettu produktiota $S \to X_1X_2 \ldots X_k$, niin juurelle tulee k jälkeläissolmua, joiden nimet vasemmalta oikealle ovat $X_1, X_2, \ldots, X_k$;

(iii) jos seuraavassa askelessa on sovellettu produktiota $X_i \to Y_1Y_2 \ldots Y_l$, 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.

Jäsennyspuu:



Solmut esijärjestyksessä:

$E_1 E_2 T_2 F_1 a_1 + T_1 F_2 a_2$

Vasen johto:

$$
\begin{aligned}
E &\underset{\text{lm}}{\Rightarrow} E + T \underset{\text{lm}}{\Rightarrow} T + T \underset{\text{lm}}{\Rightarrow} F + T \\
&\underset{\text{lm}}{\Rightarrow} a + T \underset{\text{lm}}{\Rightarrow} a + F \underset{\text{lm}}{\Rightarrow} a + a
\end{aligned}
$$

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 $G = (V, \Sigma, P, S)$ yhteydetön 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 $S \underset{\text{rm}}{\Rightarrow}^* x$.

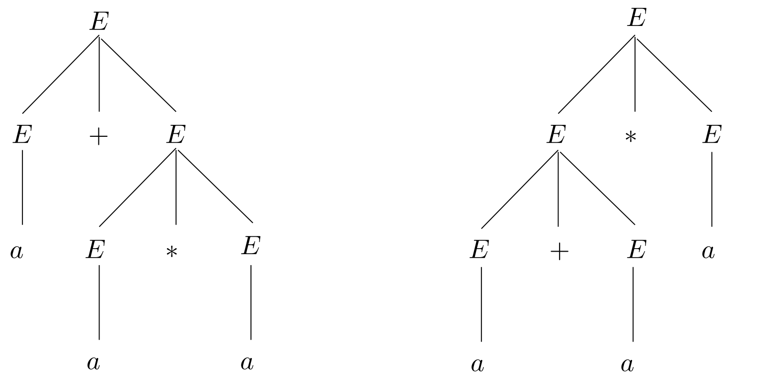
Seuraus 3.4 *Jokaisella G:n lauseella on vasen ja oikea johto.*

Yhteydettömän kieliopin tuottamien lauseiden jäsennyspuut, vasemmat ja oikeat johdot vastaavat siis yksikäsitteisesti toisiaan.[3] Kieliopin G jäsennysongelman ratkaisuun katsotaan usein kuuluvan pelkän päätösongelman "Onko $x \in L(G)$?" ratkaisemisen lisäksi jonkin näistä jäsennysesityksistä tuottaminen kieleen kuuluville lauseille x.

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. 47) kuvan 3.4 esittämät

[3] 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 51 johtoesimerkin kohdassa (ii)).



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

kaksi jäsennystä.

Yhteydetön 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). Yhteydetön 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 Osittava jäsentäminen

Yksi (yleisessä muodossa tehoton!) tapa etsiä vasenta johtoa, tai yhtäpitävästi jäsennyspuuta, annetun kieliopin G mukaiselle lauseelle x on aloittaa G:n lähtösymbolista ja generoida systemaattisesti kaikki mahdolliset vasemmat johdot (jäsennyspuut), samalla sovittaen muodostetun lausejohdoksen päätemerkkejä (puun lehtiä) x:n merkkeihin. Ei-yhteensopivuuden ilmetessä peruutetaan viimeksi tehty produktiovalinta ja kokeillaan järjestyksessä seuraavaa vaihtoehtoa.

Tällaista lauseenjäsennystapaa sanotaan *osittavaksi*, koska siinä tarkasteltu lause yritetään johtaa kieliopin lähtösymbolista osittamalla se valittujen produktioiden mukaisiin rakenneosiin ja yrittämällä näin, tarvittaessa toistuvasti edelleen osittamalla, sovittaa kieliopin tuottamaa rakennetta yhteen lauseen osajonojen kanssa.

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

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

Esimerkiksi lauseen $a - a$ osittava jäsennys kieliopin G suhteen etenee seuraavasti, kun kunkin välikkeen vaihtoehtoiset produktiot kokeillaan yllä annetussa järjestyksessä systemaattisesti vasemmalta oikealle:

$$
\begin{array}{llllllll}
E & \underset{\text{lm}}{\Rightarrow} & T+E & \underset{\text{lm}}{\Rightarrow} & a+T & & & \text{[ristiriita; peruutetaan]} \\
 & & & \underset{\text{lm}}{\Rightarrow} & (E)+T & & & \text{[ristiriita; peruutetaan]} \\
 & \underset{\text{lm}}{\Rightarrow} & T-E & \underset{\text{lm}}{\Rightarrow} & a-E & \underset{\text{lm}}{\Rightarrow} & a-T+E \underset{\text{lm}}{\Rightarrow} a-a+E & \text{[ristiriita; peruutetaan]} \\
 & & & & & & \underset{\text{lm}}{\Rightarrow} a-(E)+E & \text{[ristiriita; peruutetaan]} \\
 & & & & & \underset{\text{lm}}{\Rightarrow} & a-T-E \underset{\text{lm}}{\Rightarrow} a-a-E & \text{[ristiriita; peruutetaan]} \\
 & & & & & & \underset{\text{lm}}{\Rightarrow} a-(E)-E & \text{[ristiriita; peruutetaan]} \\
 & & & & & \underset{\text{lm}}{\Rightarrow} & a-T \underset{\text{lm}}{\Rightarrow} a-a & \text{[OK!]}
\end{array}
$$

On huomattava, että tätä menetelmää yksinkertaisimmillaan käytettäessä kielioppi ei saa olla vasemmalle rekursiivinen, so. millään välikkeellä A ei saa olla mahdollista johtaa $A \Rightarrow^{+} A\gamma$. Vasen rekursio voi johtaa osituksen ikuiseen silmukkaan, ellei siihen jollain tavoin varauduta.

LL(1)-kieliopit ja niiden jäsentäminen

Osittava jäsennystekniikka saadaan huomattavasti tehokkaammaksi, jos kieliopilla on sellainen ominaisuus, että jäsennyksen joka vaiheessa määrää tavoitteena olevan lauseen *seuraava merkki* yksikäsitteisesti sen, mikä lavennettavana olevaan välikkeeseen liittyvä produktio on valittava. Kielioppia, jolla on tämä ominaisuus, sanotaan *LL(1)-tyyppiseksi*.[4]

Esimerkiksi kielioppi G voidaan muokata LL(1)-muotoon "tekijöimällä" välikkeen E produktiot (ks. s. 59). Näin saadaan G:n kanssa ekvivalentti, so. saman kielen tuottava kielioppi G':

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

Kieliopin G' tuottamille lauseille on helppoa muodostaa vasemmat johdot suoraan ja peruuttamatta lähtösymbolista E alkaen. Esimerkiksi lauseen $a-a$ jäsentäminen sujuu seuraavasti:

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

LL(1)-tyyppiselle kieliopille voidaan kirjoittaa osittava jäsennysohjelma suoraan kieliopin rakennetta noudattelevina rekursiivisina proseduureina. Esimerkiksi kieliopin G' pohjalta voidaan muodostaa seuraava C-kielinen funktiokokoelma, 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.)

[4] 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". Jonkin verran laajempi tehokkaasti jäsennettävien kielioppien luokka ovat ns. LR(k)-kieliopit ("*l*eft-to-right scan, *r*ight parse"). Näiden jäsennysalgoritmit ovat kuitenkin sen verran mutkikkaampia että niitä ei käsitellä tässä.

```
#include <stdio.h>

int next;
void E(void); void Eprime(void); void T(void);

void ERROR(char *msg)
{ printf("%s\n",msg); exit(1); }

void E(void)
{
  printf("E  -> TE'\n");
  T(); Eprime();
}

void Eprime(void)
{
  if (next == '+') {
    printf("E' -> +E\n");
    next = getchar();
    E();
  }
  else if (next == '-') {
    printf("E' -> -E\n");
    next = getchar();
    E();
  }
  else
    printf("E' -> \n");
}

void T(void)
{
  if (next == 'a') {
    printf("T  -> a\n");
    next = getchar();
  }
  else if (next == '(') {
    printf("T  -> (E)\n");
    next = getchar();
    E();
    if (next != ')')
      ERROR(") expected.");
    next = getchar();
  }
  else ERROR("T cannot start with this.");
}
```

```
int main(void)
{
  next = getchar();
  E();
  exit(0);
}
```

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

```
T  -> TE'
T  -> a
E' -> -E
T  -> TE'
T  -> (E)
T  -> TE'
T  -> a
E' -> +E
T  -> TE'
T  -> a
E' ->
E' ->
```

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 $A \Rightarrow^* \varepsilon$.

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

$$\begin{array}{lcll} S & \to & Ab & \mid\ Cd \\ A & \to & aA & \mid\ \varepsilon \\ C & \to & cC & \mid\ \varepsilon. \end{array}$$

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, Σ, P, S) välikkeisiin liittyvät päätemerkkijoukot:[5]

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

Esimerkiksi edellisen esimerkkikieliopin tapauksessa saadaan:

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

Laajennetaan FIRST-joukkojen määritelmä mielivaltaisille merkkijonoille seuraavasti:

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

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 osittava jäsentäjä muodostetaan yleisessä tapauksessa seuraavan kaavan mukaan (esitys selkeyden vuoksi syntaksiltaan Pascal-ohjelmointikieltä muistuttavalla pseudokoodilla):

Apurutiinit:

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

[5]Tarkkaan ottaen joukot voivat sisältää yksittäisten päätemerkkien lisäksi myös tyhjän merkkijonon ε.

... sisältää virheilmoitustekstin.}

Välikettä A vastaava proseduuri:

procedure A; {A:n produktiot $A \to \omega_1 \mid \ldots \mid \omega_n$}
begin
  if *next* **in** $[a_{11}, \ldots, a_{1m_1}]$ **then** {FIRST({ω_1}FOLLOW(A)) = $\{a_{11}, \ldots, a_{1m_1}\}$}
  begin {produktio $A \to \omega_1$}
    $parse(\omega_1)$
  end else
  ⋮
  if *next* **in** $[a_{n1}, \ldots, a_{nm_n}]$ **then** {FIRST({ω_n}FOLLOW(A)) = $\{a_{n1}, \ldots, a_{nm_n}\}$}
  begin {produktio $A \to \omega_n$}
    $parse(\omega_n)$
  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 teoriassa suppea, mutta käytännössä sangen hyödyllinen kielioppiluokka. Seuraavassa esitetään kaksi kielioppimuunnosta, joilla "melkein" LL(1)-muotoisia kielioppeja voidaan muuntaa tähän muotoon.

1. Vasen tekijöinti

Kielioppi, jossa on produktiot

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

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

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

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. 55) korvattiin produktiot

$$E \to T + E \mid T - E \mid T$$

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

tekijöidyllä esityksellä

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

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

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

produktioilla

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

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

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

Vasemman rekursion poisto on periaatteessa mahdollista yleisemminkin. Jokainen yhteydetön 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 $S \rightarrow \varepsilon$, missä a on päätemerkki, $B_1, \ldots, B_k$ välikkeitä ja S lähtösymboli.

* 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:

[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.

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

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

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

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

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

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

FOLLOW-joukot määritetään sitten FIRST-joukkojen avulla seuraavasti:

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

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

   ja lähtösymbolille S lisäksi:

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

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

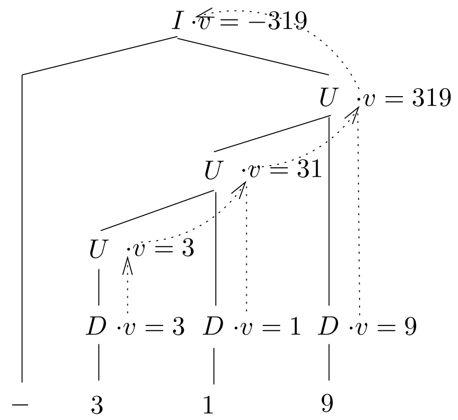
$$\text{FOLLOW}(B) := \text{FOLLOW}(B) \cup \text{FOLLOW}(A).$$

3.5 * Ekskursio: Attribuuttikieliopit

Kätevä tapa liittää yhteydettömiin 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).

Kieliopin produktioihin $A \to X_1 \dots X_k$ 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 $A \to X_1 \dots X_k$ liitettävissä säännöissä saa mainita vain symbolien $A, X_1, \dots, X_k$ attribuutteja.



Kuva 3.5: Attributoitu jäsennyspuu.

Esimerkiksi seuraavassa on etumerkillisiä kokonaislukuja tuottavaan yhteydettömään 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.

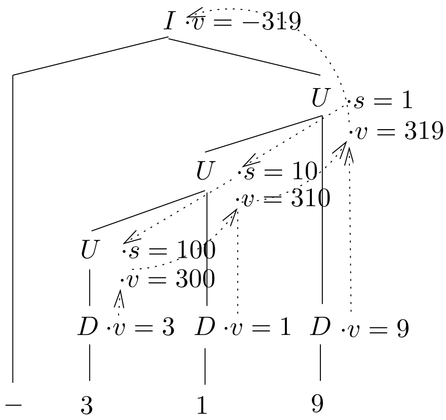
Produktiot:			*Evaluointisäännöt:*		
I	$\rightarrow$	$+U$	$I.v$	$:=$	$U.v$
I	$\rightarrow$	$-U$	$I.v$	$:=$	$-U.v$
I	$\rightarrow$	U	$I.v$	$:=$	$U.v$
U	$\rightarrow$	D	$U.v$	$:=$	$D.v$
U	$\rightarrow$	UD	$U_1.v$	$:=$	$10 * U_2.v + D.v$
D	$\rightarrow$	0	$D.v$	$:=$	0
D	$\rightarrow$	1	$D.v$	$:=$	1
$\vdots$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

Produktioon $U \rightarrow UD$ 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.

Attribuuttikieliopin attribuutti t on *synteettinen* (engl. synthetic), jos sen kuhunkin produktioon $A \rightarrow X_1 \dots X_k$ liittyvä evaluointisääntö on muotoa $A.t := f(A, X_1, \dots, X_k)$. 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 suun-



Kuva 3.6: Positiokerrointa käyttäen attributoitu jäsennyspuu.

tautuvalla 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	$\rightarrow$	$+U$	$U.s$	$:=$	$1,$	$I.v$	$:=$	$U.v$
I	$\rightarrow$	$-U$	$U.s$	$:=$	$1,$	$I.v$	$:=$	$-U.v$
I	$\rightarrow$	U	$U.s$	$:=$	$1,$	$I.v$	$:=$	$U.v$
U	$\rightarrow$	D				$U.v$	$:=$	$(D.v) * (U.s)$
U	$\rightarrow$	UD	$U_2.s$	$:=$	$10 * (U_1.s),$	$U_1.v$	$:=$	$U_2.v + (D.v) * (U_1.s)$
D	$\rightarrow$	0				$D.v$	$:=$	0
D	$\rightarrow$	1				$D.v$	$:=$	1
$\vdots$								
D	$\rightarrow$	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 tavanomaiseen *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:*
$E \to T + E$	$E_1.pf := (T.pf)\hat{}(E_2.pf)\hat{}('+')$
$E \to T$	$E.pf := T.pf$
$T \to F * T$	$T_1.pf := (F.pf)\hat{}(T_2.pf)\hat{}('*')$
$T \to F$	$T.pf := F.pf$
$F \to a$	$F.pf := 'a'$
$F \to (E)$	$F.pf := E.pf$

Kieliopille voidaan näiden evaluointisääntöjen pohjalta suhteellisen suoraviivaisesti laatia seuraava C-kielinen osittava jäsentäjä, jossa attribuutti-ilmentymien arvot evaluoidaan suoraan jäsennyksen yhteydessä:[8]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLEN 80                          /* Maks. lausekkeenpituus */

int next;
char *pf;                                  /* Tuloslauseke           */
void E(char *); void T(char *); void F(char *);

void ERROR(char *msg)
{
  printf("%s\n", msg); exit(1);
}

/* Produktiot: E -> T+E | T  */
void E(char *pf)
{
  char *pf1, *pf2;
  pf1 = malloc(MAXLEN+1);
  pf2 = malloc(MAXLEN+1);
  T(pf1);                                  /* pf1  = T.pf            */
  if (next == '+') {
    next = getchar();
    E(pf2);                                /* pf2  = E(2).pf         */
    sprintf(pf, "%s%s%s", pf1, pf2, "+");  /* E.pf = pf1^pf2^('+')   */
  }
  else sprintf(pf, "%s", pf1);             /* E.pf = T.pf            */
  free(pf1); free(pf2);
}
```

[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.

```
/* Produktiot: T -> F*T | F  */
void T(char *pf)
{
  char *pf1, *pf2;
  pf1 = malloc(MAXLEN+1);
  pf2 = malloc(MAXLEN+1);
  F(pf1);                                   /* pf1  = F.pf           */
  if (next == '*') {
    next = getchar();
    T(pf2);                                 /* pf2  = T(2).pf        */
    sprintf(pf, "%s%s%s", pf1, pf2, "*");   /* T.pf = pf1^pf2^('*')  */
  }
  else sprintf(pf, "%s", pf1);              /* T.pf = F.pf           */
  free(pf1); free(pf2);
}

/* Produktiot: F -> a | (E)  */
void F(char *pf)
{
  if (next == 'a') {
    sprintf(pf, "a");                        /* F.pf = 'a'            */
    next = getchar();
  }
  else if (next == '(') {
    next = getchar();
    E(pf);                                   /* F.pf = E.pf           */
    if (next != ')')
      ERROR(") expected.");
    next = getchar();
  }
  else ERROR("F cannot start with this.");
}

int main(void)
{
  next = getchar();
  pf = malloc(MAXLEN+1);
  E(pf);
  printf("%s\n", pf);
  free(pf);
  exit(0);
}
```

3.6 Eräs yleinen jäsennysmenetelmä

Osittava jäsentäminen on selkeä ja tehokas jäsennysmenetelmä LL(1)-kieliopeille: n merkin mittaisen syötemerkkijonon käsittely sujuu ajassa $O(n)$. LL(1)-kieliopit ovat kuitenkin periaatteessa melko suppea kielioppiluokka (joskin käytännössä moniin tarkoituksiin riittävä), eikä mielivaltaisen yhteydettömän kieliopin jäsennysongelman tehokas ratkaiseminen ole aivan yhtä helppoa.

Periaatteessa, ja tietyin kieliopin muotoa koskevin varauksin, ongelma voitaisiin ratkaista esimerkiksi soveltamalla yleistä (peruuttavaa) osittavaa jäsennystä, mutta käytännössä vaikeudeksi muodostuu erilaisten kokeiltavien johtovaihtoehtojen suuri määrä. Jokaisessa epätriviaalissa yhteydettömässä kieliopissa on nimittäin n askelen mittaisia johtoja eksponentiaalinen määrä, so. $O(c^n)$ kappaletta jollakin $c \geq 2$, ja pahimmassa tapauksessa yleinen osittava jäsennys joutuu kokeilemaan ne kaikki selvittääkseen jonkin n merkin mittaisen syötteen kieliopinmukaisuuden.

Seuraavassa esitetään tämän lähestymistavan tehokkaaksi vaihtoehdoksi eräs yleinen menetelmä, ns. *Cocke–Younger–Kasami-* t. *CYK-algoritmi*, mielivaltaisen yhteydettömän kieliopin tuottamien merkkijonojen tunnistamiseen. Menetelmä toimii ajassa $O(n^3)$, missä n on tutkittavan merkkijonon pituus.

CYK-algoritmia varten käsitellään ensin joitakin kielioppimuunnoksia.

ε-produktioiden poistaminen

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

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

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

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

(i) asetetaan aluksi

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

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

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

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

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

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

Esimerkki ε-produktioiden poistamisesta:

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

$$\begin{array}{lcl} S & \to & A \mid B \mid \varepsilon \\ A & \to & aBa \mid aa \mid \varepsilon \qquad\quad \Rightarrow \\ B & \to & bAb \mid bb \mid \varepsilon \end{array}$$

$$\begin{array}{lcl} S' & \to & S \mid \varepsilon \\ S & \to & A \mid B \\ A & \to & aBa \mid aa \\ B & \to & bAb \mid bb. \end{array}$$

Yksikköproduktioiden poistaminen

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

Lemma 3.6 *Mistä tahansa yhteydettömästä 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 laajennusoperaatioita, 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 $B \in F(A)$. $\square$

Esimerkkinä poistetaan yksikköproduktiot edellä saadusta kieliopista

$$\begin{array}{lcl} S' & \to & S \mid \varepsilon \\ S & \to & A \mid B \\ A & \to & aBa \mid aa \\ B & \to & bAb \mid bb. \end{array}$$

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

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

(Huomataan, että välike S on nyt itse asiassa "turha", so. se ei voi esiintyä minkään kieliopin mukaisen lauseen johdossa. Myös turhat välikkeet voidaan haluttaessa poistaa kieliopista samantapaisella algoritmilla (HT).)

Chomskyn normaalimuoto

Yhteydetön kielioppi $G = (V, \Sigma, P, S)$ on *Chomskyn normaalimuodossa* (engl. Chomsky normal form), jos sen välikkeistä enintään S on tyhjentyvä, ja mahdollista produktiota $S \rightarrow \varepsilon$ lukuunottamatta muut produktiot ovat muotoa $A \rightarrow BC$ tai $A \rightarrow a$, 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 yhteydettömästä kieliopista G voidaan muodostaa ekvivalentti Chomskyn normaalimuotoinen kielioppi G'.*

Todistus. Olkoon $G = (V, \Sigma, P, S)$. Mikäli lähtösymboli S esiintyy G:ssä jonkin produktion oikealla puolella, otetaan käyttöön uusi lähtösymboli S' ja lisätään G:hen produktio $S' \rightarrow S$. Poistetaan sitten G:stä ε-produktiot ja yksikköproduktiot lemmojen 3.5 ja 3.6 konstruktioilla. Tämän jälkeen kaikki G:n produktiot ovat muotoa $A \rightarrow a$ tai $A \rightarrow X_1 \dots X_k$, $k \geq 2$ (tai $S \rightarrow \varepsilon / S' \rightarrow \varepsilon$).

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

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

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

(Tarkkaan ottaen uusi produktiojoukko on siis oikeastaan

$$\begin{aligned} A &\rightarrow X'_1 A_1 \\ A_1 &\rightarrow X'_2 A_2 \\ &\vdots \\ A_{k-2} &\rightarrow X'_{k-1} X'_k, \end{aligned}$$

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 $G = (V, \Sigma, P, S)$ yhteydetön 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 $x = \varepsilon$, niin $x \in L(G)$ joss $S \to \varepsilon$ on G:n produktio.
- Muussa tapauksessa merkitään $x = a_1 \dots a_n$ ja tarkastellaan x:n eri osajonojen tuottamista.

  Merkitään N_{ik}:lla niiden välikkeiden A joukkoa, joista voidaan tuottaa x:n positiosta i alkava, k merkin mittainen osajono $x_{ik} = a_i \dots a_{i+k-1}$:

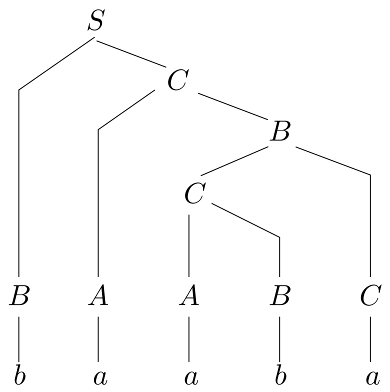
  $$N_{ik} = \{A \in V - \Sigma \mid A \underset{G}{\Rightarrow}^* a_i \dots a_{i+k-1}\}, \qquad 1 \leq i \leq i + k - 1 \leq n.$$

  Selvästi on $x \in L(G)$ joss $S \in N_{1n}$.

  Joukot N_{ik} voidaan laskea taulukoimalla lyhyistä osajonoista pitempiin seuraaviin havaintoihin nojautuen. Ensinnäkin on kaikilla välikkeillä A ja kullakin $i = 1, \dots, n$ voimassa $A \in N_{i1}$, jos ja vain jos G:ssä on produktio $A \to a_i$. Osajonopituuksilla $k \geq 2$ huomataan, että koska G on Chomskyn normaalimuodossa, niin välikkeestä A voidaan johtaa jono $x_{ik} = a_i \dots a_{i+k-1}$, jos ja vain jos jono x_{ik} voidaan jakaa kahteen lyhyempään osaan $x_{ij} = a_i \dots a_{i+j-1}$ ja $x_{(i+j)(k-j)} = a_{i+j} \dots a_{i+k-1}$, $j = 1, \dots, k-1$, niin että jostakin välikkeestä B voidaan johtaa alkuosa x_{ij}, jostakin välikkeestä C voidaan johtaa loppuosa $x_{(i+j)(k-j)}$, ja G:ssä on nämä välikkeet yhdistävä produktio $A \to BC$. Tämä päättely johtaa seuraavaan taulukointimenettelyyn joukkojen N_{ik} muodostamiseksi:

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

Kuva 3.7: CYK-algoritmin laskentataulukko.



Kuva 3.8: Eräs CYK-taulukon mukainen jäsennyspuu.

Joukkojen N_{ik} laskeminen:

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

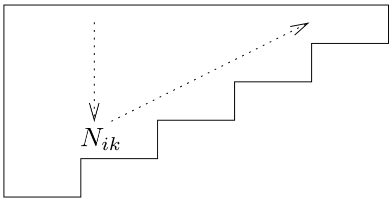
$$N_{i1} \quad := \quad \{A \in V - \Sigma \mid A \rightarrow a_i \text{ on } G\text{:n produktio}\}.$$

(ii) Määritetään sitten kaikilla osajonopituuksilla $k = 2, \dots, n$ ja kullakin k kaikilla mahdollisilla alkukohdilla $i = 1, \dots, n - k + 1$ joukon N_{ik} välikkeet seuraavan kaavan mukaan:

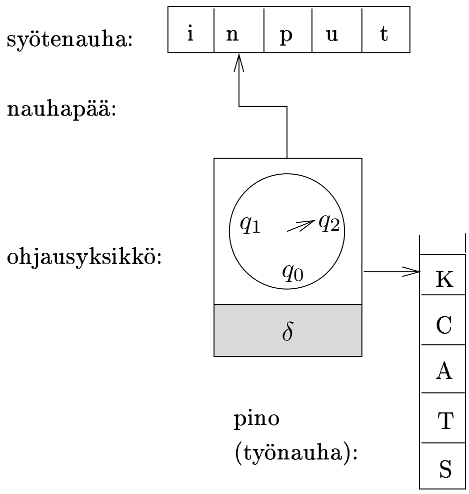
$$N_{ik} \quad := \quad \bigcup_{j=1}^{k-1} \{A \in V - \Sigma \quad | \quad G\text{:ssä on produktio } A \rightarrow BC, \text{ missä } B \in N_{ij} \text{ ja } C \in N_{i+j,k-j}\}. \quad \square$$

Esimerkkinä CYK-algoritmin soveltamisesta tarkastellaan Chomskyn normaalimuotoista kielioppia

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



Kuva 3.9: CYK-algoritmin laskentajärjestys.



Kuva 3.10: Pinoautomaatti.

ja syötemerkkijonoa $x = baaba$. Algoritmin laskenta etenee tässä tapauksessa kuvan 3.7 taulukon esittämällä tavalla. Tässä tapauksessa lähtösymboli S kuuluu joukkoon N_{15}, joten päätellään, että x kuuluu kieliopin tuottamaan kieleen. Taulukkoon 3.7 on merkitty näkyviin myös yksi lähtösymbolin $S \in N_{15}$ "perusteluketju", joka tässä tapauksessa vastaa syötejonon *baaba* kuvan 3.8 mukaista jäsennyspuuta.

Yleisesti ottaen CYK-algoritmin laskentajärjestys on sellainen, että jotakin joukkoa N_{ik} määritettäessä edetään indeksin j kasvaessa arvosta 1 arvoon $k-1$ samanaikaisesti sarakkeessa N_{ij} joukkoa N_{ik} "kohti" ja diagonaalia $N_{i+j,k-j}$ pitkin siitä "poispäin" (kuva 3.9).

3.7 Pinoautomaatit

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

Intuitiivisesti pinoautomaatti on äärellinen automaatti, johon on lisätty yksi potentiaalisesti ääretön *työnauha* (kuva 3.10). 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*;
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ 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 perusmääritelmänsä mukaan *epädeterministisiä*.) Jos $\sigma = \varepsilon$, automaatti tekee siirtymän syötemerkkiä lukematta; vastaavasti jos $\gamma = \varepsilon$, automaatti ei lue pinomerkkiä ja uusi kirjoitettu merkki tulee pinon päälle vanhaa päällimmäistä merkkiä poistamatta ("push"-operaatio). Jos pinosta luettu merkki on $\gamma \neq \varepsilon$ ja kirjoitettavana on $\gamma_i = \varepsilon$, 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, ε). 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

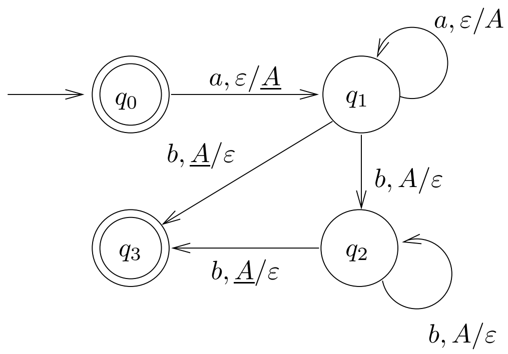
$$(q, w, \alpha) \underset{M}{\vdash}^* (q', w', \alpha'),$$

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

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



Kuva 3.11: Kielen $\{a^k b^k \mid k \geq 0\}$ tunnistava pinoautomaatti.

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

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

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

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

missä

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

Esimerkiksi syötteellä *aabb* automaatti M toimii seuraavasti:

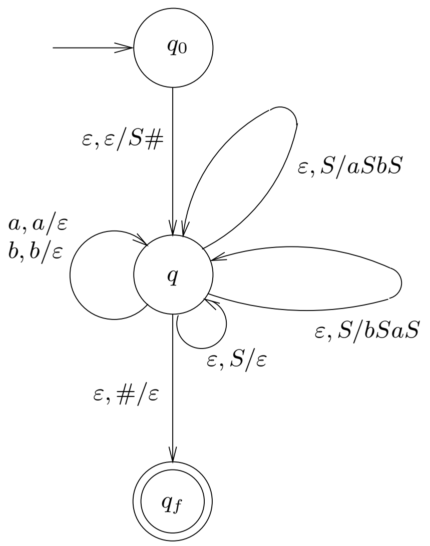
$$\begin{array}{rcccl}
(q_0, aabb, \varepsilon) & \vdash & (q_1, abb, \underline{A}) & \vdash & (q_1, bb, A\underline{A}) \\
& \vdash & (q_2, b, \underline{A}) & \vdash & (q_3, \varepsilon, \varepsilon).
\end{array}$$

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

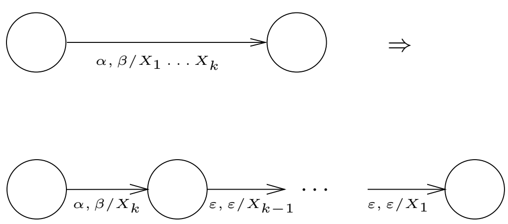
Myös pinoautomaateille voidaan kehittää kaavioesitys äärellisten automaattien tilasiirtymäkaavioita suoraviivaisesti yleistämällä. Esimerkiksi edellinen automaatti M voitaisiin esittää kuvan 3.11 mukaisena kaaviona.

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

Lause 3.8 *Kieli on yhteydetön, jos ja vain jos se voidaan tunnistaa (epädeterministisellä) pinoautomaatilla.* □



Kuva 3.12: Kielioppia $\{S \to aSbS \mid bSaS \mid \varepsilon\}$ vastaava pinoautomaatti.



Kuva 3.13: Merkkijonon painaminen pinoon.

Lauseen 3.8 todistus sivuutetaan tässä, mutta periaatteena annettua kielioppia G vastaavan pinoautomaatin M_G toiminnassa on, että M_G:n pinon käyttäytyminen syötteellä x noudattelee G:n mukaisen vasemman lausejohdon $S \underset{\text{lm}}{\Rightarrow}^* x$ etenemistä: jos pinon päällimmäisenä on välikemerkki, sovelletaan jotain G:n produktiota ja lisätään pinon pinnalle vastaavat merkit; jos pinon päällimmäisenä on päätemerkki, se sovitetaan yhteen seuraavan syötemerkin kanssa.

Esimerkiksi kielioppia $\{S \to aSbS \mid bSaS \mid \varepsilon\}$ vastaa kuvan 3.12 mukainen pinoautomaatti. (Piirroksen selventämiseksi on kuvassa 3.12 käytetty kuvan 3.13 mukaista luonnollista lyhennemerkintää.)

Esimerkiksi syötteellä $abab$ on kuvan 3.12 automaatilla seuraava hyväksyvä laskenta:

$$\begin{array}{lllll} (q_0, abab, \varepsilon) & \vdash & (q, abab, S\#) & \vdash^* & (q, abab, aSbS\#) \\ & \vdash & (q, bab, SbS\#) & \vdash^* & (q, bab, bSaSbS\#) \\ & \vdash & (q, ab, SaSbS\#) & \vdash & (q, ab, aSbS\#) \\ & \vdash & (q, b, SbS\#) & \vdash & (q, b, bS\#) \\ & \vdash & (q, \varepsilon, S\#) & \vdash & (q, \varepsilon, \#) \\ & \vdash & (q_f, \varepsilon, \varepsilon). & & \end{array}$$

Tämä vastaa annetun kieliopin mukaista lauseen $abab$ vasenta johtoa:

$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow ab\underline{S}aSbS \Rightarrow aba\underline{S}bS \Rightarrow abab\underline{S} \Rightarrow abab.$$

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

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

Ehkä hieman yllättäen äärellisten automaattien peruslauseen 2.2 (s. 30) vastine ei pinoautomaattien kohdalla ole voimassa, vaan *epädeterministiset pinoautomaatit ovat aidosti vahvempia kuin deterministiset.* Esimerkiksi kieli $\{ww^R \mid w \in \{a, b\}^*\}$ voidaan tunnistaa epädeterministisellä, mutta ei deterministisellä pinoautomaatilla. (Todistus sivuutetaan.)

Yhteydetön kieli on *deterministinen*, jos se voidaan tunnistaa jollakin deterministisellä pinoautomaatilla. Determinististen kielten luokka on yhteydettömien kielten jäsennysteoriassa keskeinen, sillä siihen kuuluvat kielet voidaan jäsentää oleellisesti tehokkaammin kuin yleiset, mahdollisesti epädeterministisen automaatin vaativat yhteydettömät kielet.

3.8 * Yhteydettömien kielten rajoituksista

Yhteydettömillä kieliopeilla voidaan kuvata suurin osa esimerkiksi ohjelmointikielten syntaksin piirteistä. Joitakin ei-yhteydettömiä syntaksin piirteitä sisältyy ohjelmointikielissä tyypillisesti ohjelman eri osien yhteensopivuusvaatimuksiin: jos esimerkiksi vaaditaan että kaikki muuttujat on esiteltävä ennen käyttöä, tai että ohjelman tulee sisältää kaikkien siinä kutsuttujen funktioiden määrittelyt.

Keskeinen työkalu annetun kielen osoittamiseen ei-yhteydettömäksi on säännöllisten kielten pumppauslemman (Lemma 2.6, s. 43) vastine, *yhteydettömien kielten pumppauslemma.* Erona säännöllisten kielten tapaukseen on se, että nyt merkkijonoa on pumpattava samanaikaisesti kahdesta paikasta. Muotoilunsa takia tulos tunnetaan myös "$uvwxy$-lemman" nimellä.

Lemma 3.9 (uvwxy-lemma) *Olkoon L yhteydetön 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) $|vwx| \leq n$,

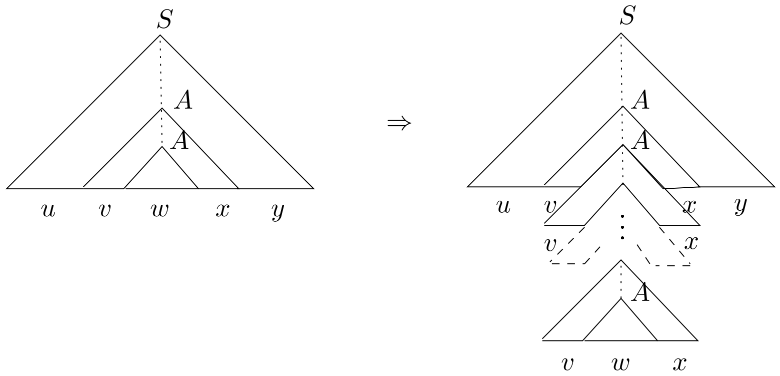
(iii) $uv^iwx^iy \in L$ *kaikilla* $i = 0, 1, 2, \ldots$.

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. Itse asiassa on jonkin välikkeen A toistuttava jo tämän polun $k + 2$ alimman solmun joukossa. Merkkijono z voidaan nyt osittaa kuvan 3.14 esittämällä tavalla $z = uvwxy$, missä w on A:n alimmasta ilmentymästä tuotettu osajono ja vwx seuraavaksi ylemmästä A:n ilmentymästä tuotettu osajono; osajonot saadaan johdosta

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



Kuva 3.14: Yhteydettömän kielen merkkijonon pumppaus.

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 välikkeen A valinnan perusteella sen toiseksi ylin ilmentymä on enintään korkeudella $k+1$ jäsennyspuun lehdistä, on tähän ilmentymään juurtuvan alipuun tuotokselle voimassa pituusraja $|vwx| \leq 2^{k+1} = n$. $\square$

Esimerkkinä lemman 3.9 soveltamisesta osoitetaan, että kieli $L = \{a^kb^kc^k \mid k \geq 0\}$ ei ole yhteydetön. Oletetaan nimittäin, että L olisi yhteydetön; 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

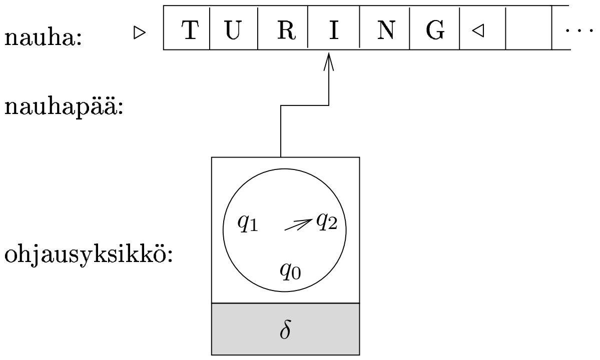
Seuraavassa esitettävän automaattimallin kehitti brittimatemaatikko Alan Turing vuosina 1935–36 — siis noin 10 vuotta ennen ensimmäisten tietokoneiden kehittämistä — pohtiessaan mekaanisen laskennan rajoja. Näennäisestä yksinkertaisuudestaan huolimatta tämä Turingin automaattimalli on huomattavan voimakas. 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 '$\triangleright$', ja sen käytettyä osaa seuraa loppumerkki '$\triangleleft$'. 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_{acc} ja hylkäävä q_{rej}. 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_{acc}, merkkijono kuuluu koneen tunnistamaan kieleen; jos se taas pysähtyy lopputilassa q_{rej} *tai jää pysähtymättä*, merkkijono ei kuulu kieleen.

Täsmällisesti voidaan määritellä:

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

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}),$$

missä

- Q on koneen *tilojen* äärellinen joukko;
- Σ on koneen *syöteaakkosto*;
- $\Gamma \supseteq \Sigma$ on koneen *nauha-aakkosto*; oletetaan, että merkit $\triangleright, \triangleleft \notin \Gamma$;
- $\delta : (Q - \{q_{\text{acc}}, q_{\text{rej}}\}) \times (\Gamma \cup \{\triangleright, \triangleleft\}) \to Q \times (\Gamma \cup \{\triangleright, \triangleleft\}) \times \{L, R\}$ on koneen *siirtymäfunktio*; siirtymäfunktion arvoilta

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

  vaaditaan, että

  (i) jos $b = \triangleright$, niin $a = \triangleright$;
  (ii) jos $a = \triangleright$, niin $b = \triangleright$ ja $\Delta = R$;
  (iii) jos $b = \triangleleft$, niin $a = \triangleleft$ ja $\Delta = L$;

- $q_0 \in Q - \{q_{\text{acc}}, q_{\text{rej}}\}$ on koneen *alkutila*;
- $q_{\text{acc}} \in Q$ on koneen *hyväksyvä* ja $q_{\text{rej}} \in Q$, $q_{\text{rej}} \neq q_{\text{acc}}$, 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 $a =$ '$\triangleright$' tai '$\triangleleft$', ja siirtymäfunktion arvo on aina määrittelemätön, kun $q = q_{\mathrm{acc}}$ tai $q = q_{\mathrm{rej}}$. Joutuessaan jompaan kumpaan näistä tiloista kone pysähtyy heti.

Koneen ***tilanne*** on nelikko $(q, u, a, v) \in Q \times \Gamma^* \times (\Gamma \cup \{\varepsilon\}) \times \Gamma^*$, missä voi olla $a = \varepsilon$, mikäli myös $u = \varepsilon$ tai $v = \varepsilon$. 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 $a = \varepsilon$, jos nauhapää sijaitsee aivan nauhan alussa tai sen käytetyn osan lopussa. Ensimmäisessä tapauksessa ajatellaan, että kone "havaitsee" merkin '$\triangleright$' ja toisessa tapauksessa merkin '$\triangleleft$'. ***Alkutilanne syötteellä*** $x = a_1a_2 \ldots a_n$ on nelikko $(q_0, \varepsilon, a_1, a_2 \ldots a_n)$. 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 $c \in \Gamma \cup \{\varepsilon\}$:

- 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 $\delta(q, \triangleright) = (q', \triangleright, R)$, niin $(q, \underline{\varepsilon}cv) \underset{M}{\vdash} (q', \underline{c}v)$;
- jos $\delta(q, \triangleleft) = (q', b, R)$, niin $(q, u\underline{\varepsilon}) \underset{M}{\vdash} (q', ub\underline{\varepsilon})$;
- jos $\delta(q, \triangleleft) = (q', b, L)$, niin $(q, uc\underline{\varepsilon}) \underset{M}{\vdash} (q', u\underline{c}b)$;
- jos $\delta(q, \triangleleft) = (q', \triangleleft, L)$, niin $(q, uc\underline{\varepsilon}) \underset{M}{\vdash} (q', u\underline{c})$.

Tilanteet, jotka ovat muotoa (q_{acc}, w) tai (q_{rej}, 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

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

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

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

On tärkeää huomata, että koneen M ei tarvitse pysähtyä syötteillä, jotka eivät kuulu kieleen $L(M)$. Jos M kuitenkin pysähtyy myös kaikilla hylättävillä syötteillä, sanotaan että M on ***totaalinen*** ja että se ***ratkaisee*** kielen $L(M)$. Luvussa 6 tullaan näkemään, että kysymys siitä voidaanko annettu kieli tunnistaa totaalisella Turingin koneella on erittäin epätriviaali. Siten kielen "tunnistamisen" ja "ratkaisemisen" välinen ero on Turingin koneiden tapauksessa aivan keskeinen.

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

$$M = (\{q_0, q_1, q_{\text{acc}}, q_{\text{rej}}\}, \{a\}, \{a\}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}),$$

missä

$$\begin{aligned}
\delta(q_0, a) &= (q_1, a, R), \\
\delta(q_1, a) &= (q_0, a, R), \\
\delta(q_0, \lhd) &= (q_{\text{acc}}, \lhd, L), \\
\delta(q_1, \lhd) &= (q_{\text{rej}}, \lhd, L).
\end{aligned}$$

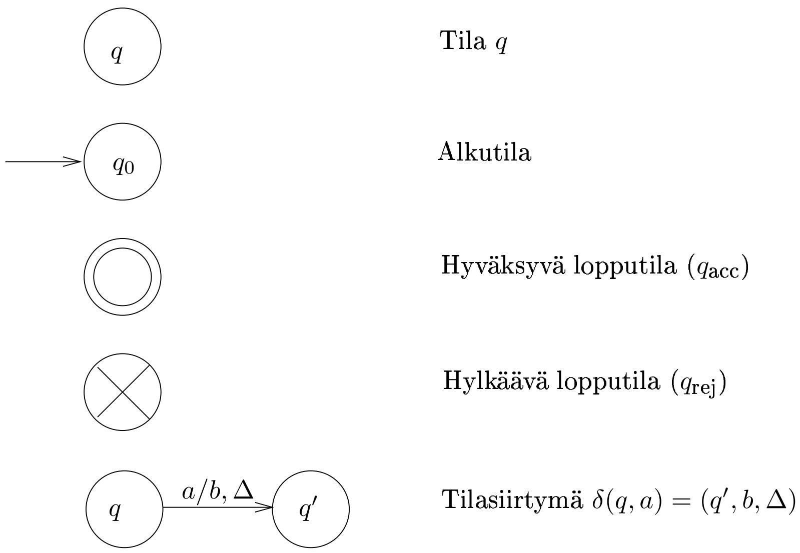
Koneen M laskenta esimerkiksi syötteellä aaa etenee seuraavasti:

$$(q_0, \underline{a}aa) \underset{M}{\vdash} (q_1, a\underline{a}a) \underset{M}{\vdash} (q_0, aa\underline{a}) \underset{M}{\vdash} (q_1, aaa\underline{\varepsilon}) \underset{M}{\vdash} (q_{\text{rej}}, aa\underline{a}).$$

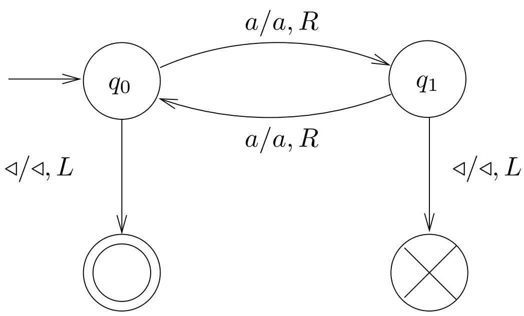
Koska kone pysähtyy tilassa q_{rej}, päätellään että $aaa \notin L(M)$.

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 $\{a^{2k} \mid k \geq 0\}$ tunnistavaa Turingin konetta vastaava kaavio on esitetty kuvassa 4.3.

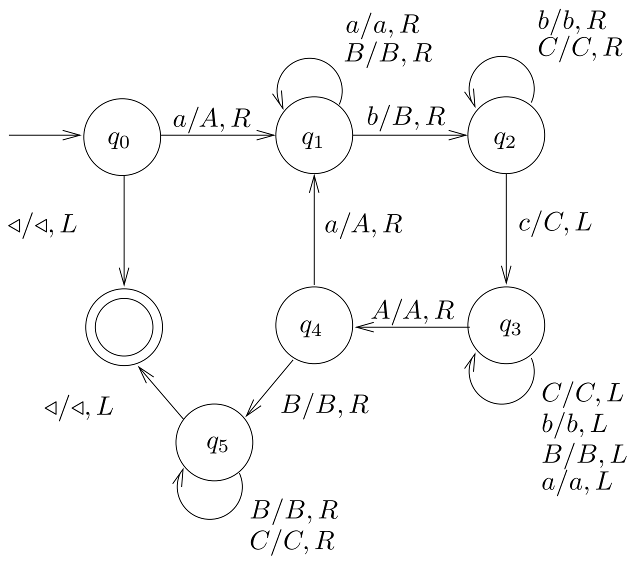
Mutkikkaampana esimerkkinä on kuvassa 4.4 esitetty ei-yhteydettömän kielen $\{a^k b^k c^k \mid k \geq 0\}$ tunnistava (ja ratkaiseva) kone. Kaavion yksinkertaistamiseksi on tässä noudatettu käytäntöä, jonka mukaan siirtymiä hylkäävään lopputilaan ei merkitä näkyviin. Kuvatun koneen idea on, että se pitää 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 pieniä b- tai c-kirjaimia ei ole jäljellä. Esimerkiksi syötteeseen $aabbcc$ liittyvä



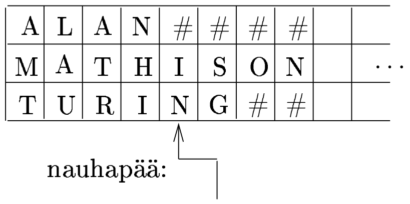
Kuva 4.2: Turingin koneiden kaavioesityksen merkinnät.



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.

laskenta etenee seuraavasti:

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

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

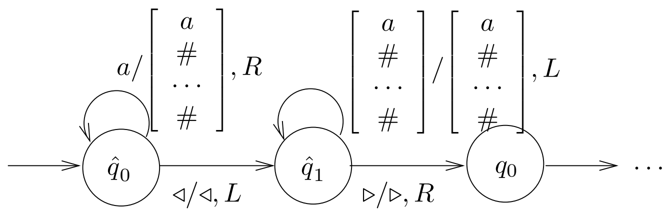
Tässä laajennuksessa sallitaan, että Turingin koneen nauha koostuu k:sta rinnakkaisesta urasta, jotka kaikki kone lukee ja kirjoittaa yhdessä laskenta-askelessa. Koneen nauha on siis kuvassa 4.5 esitetyn tapainen (kuvassa $k = 3$). Koneen siirtymäfunktion arvot ovat vastaavasti muotoa:

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

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

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}),$$



Kuva 4.6: Esiprosessori moniuraisen koneen syötteen nostamiseksi ykkösuralle.

missä muut komponentit ovat kuten standardimallissa, paitsi siirtymäfunktio:

$$\delta : (Q - \{q_{\text{acc}}, q_{\text{rej}}\}) \times (\Gamma^k \cup \{\rhd, \lhd\}) \to Q \times (\Gamma^k \cup \{\rhd, \lhd\}) \times \{L, R\}.$$

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 $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ k-urainen Turingin kone, joka tunnistaa kielen L. Vastaava standardimallinen kone $\widehat{M}$ voidaan muodostaa seuraavasti:

$$\widehat{M} = (\hat{Q}, \Sigma, \hat{\Gamma}, \hat{\delta}, \hat{q}_0, q_{\text{acc}}, q_{\text{rej}}),$$

missä $\hat{Q} = Q \cup \{\hat{q}_0, \hat{q}_1\}$, $\hat{\Gamma} = \Sigma \cup \Gamma^k$ (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

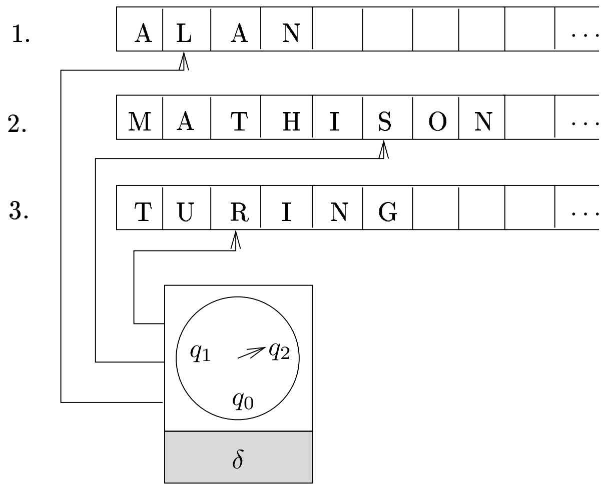
$$\begin{bmatrix} a_1 \\ \# \\ \vdots \\ \# \end{bmatrix} \begin{bmatrix} a_2 \\ \# \\ \vdots \\ \# \end{bmatrix} \cdots \begin{bmatrix} a_n \\ \# \\ \vdots \\ \# \end{bmatrix}.$$

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

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

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



Kuva 4.7: Kolmenauhainen Turingin kone.

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.

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

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}),$$

missä muut komponentit ovat kuten standardimallissa, paitsi siirtymäfunktio:

$$\delta : (Q - \{q_{\text{acc}}, q_{\text{rej}}\}) \times (\Gamma \cup \{\triangleright, \triangleleft\})^k \to Q \times ((\Gamma \cup \{\triangleright, \triangleleft\}) \times \{L, R\})^k.$$

Seuraajatilannerelaatio ym. peruskäsitteet määritellään pienin muutoksin entiseen tapaan.

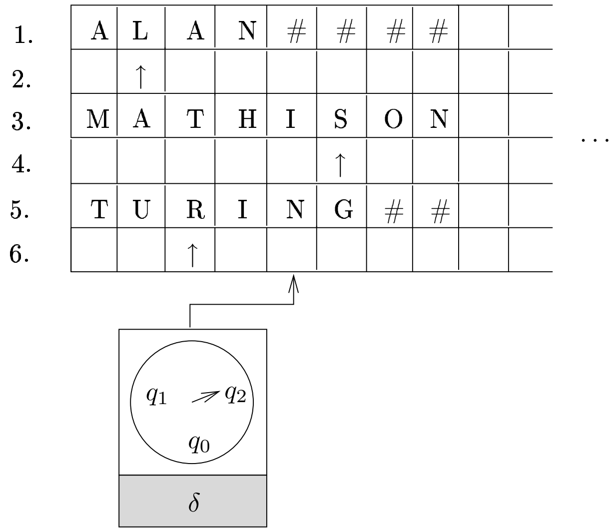
Lause 4.2 *Jos formaali kieli L voidaan tunnistaa k-nauhaisella Turingin koneella, se voidaan tunnistaa myös standardimallisella Turingin koneella.*

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

Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ 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.



Kuva 4.8: Kolmenauhaisen Turingin koneen simulointi kuusiuraisella.

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

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

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}),$$

missä muut komponentit ovat kuten deterministisessä standardimallissa, paitsi siirtymäfunktio:

$$\delta : (Q - \{q_{\mathrm{acc}}, q_{\mathrm{rej}}\}) \times (\Gamma \cup \{\triangleright, \triangleleft\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\triangleright, \triangleleft\}) \times \{L, R\}).$$

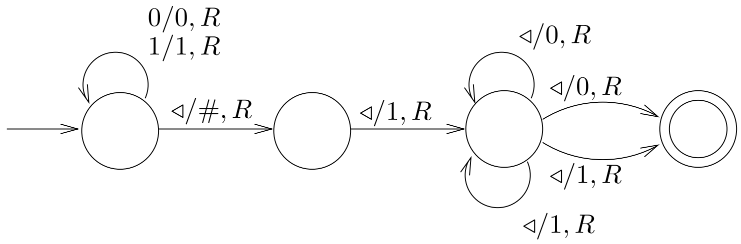
Siirtymäfunktion arvon

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

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

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



Kuva 4.9: Epädeterministinen Turingin kone GEN_INT.

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

Kerrataan vielä koneen M tunnistaman kielen määritelmä:

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

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 epädeterminististen 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

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

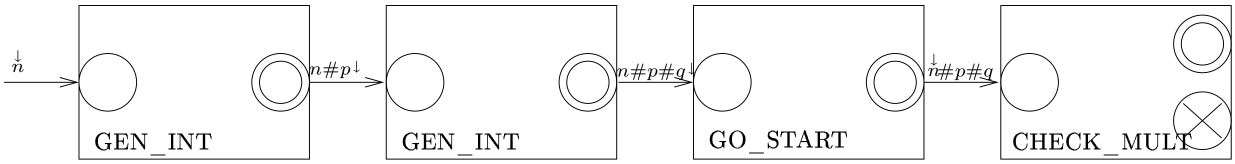
Tällaisen koneen suunnittelu on hieman työlästä, mutta ei periaatteessa kovin vaikeata. Olkoon lisäksi GO_START deterministinen Turingin kone, joka siirtää nauhapään osoittamaan nauhan ensimmäistä merkkiä. (Suunnittelu HT.)

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

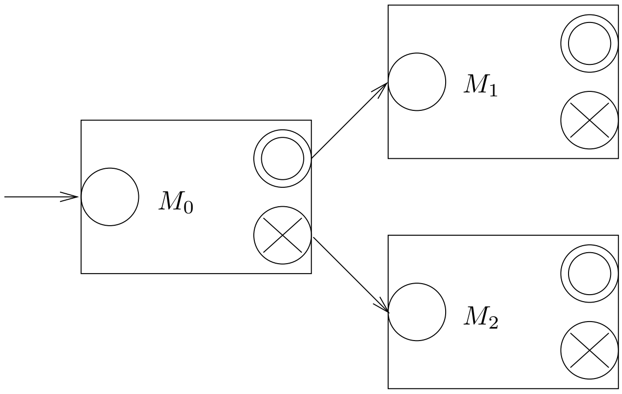
$$L(\text{TEST\_COMPOSITE}) = \{n \mid n \text{ on binäärimuotoinen yhdistetty luku}\}$$

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.

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



Kuva 4.10: Epädeterministinen Turingin kone TEST_COMPOSITE.



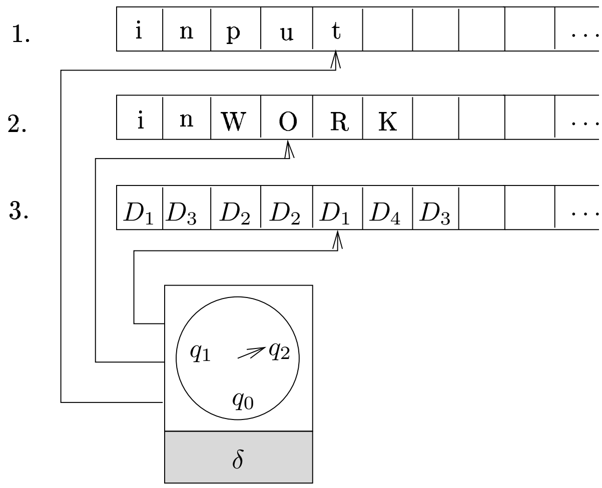
Kuva 4.11: Turingin koneiden yhdistäminen.

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 $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ 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 (ε, D_1, D_2, ..., D_r, D_1D_1, 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



Kuva 4.12: Epädeterministisen Turingin koneen simulointi deterministisellä.

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. $\square$

Luku 5

Rajoittamattomat ja yhteysherkät kieliopit

Jos yhteydettömiä 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 ($V^+ = V^* - \{\varepsilon\}$);
- $S \in N$ 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-yhteydettömäksi:

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

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: $LA^k B^k C^k$;

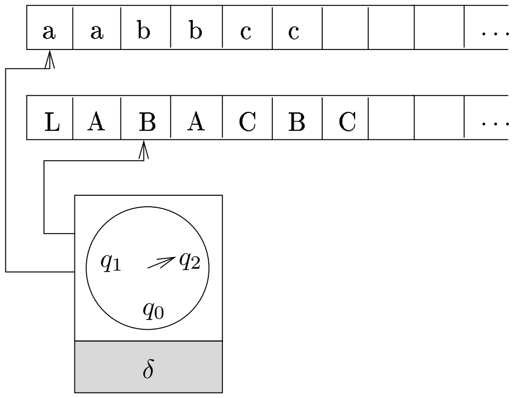
3. lopuksi muutetaan välikkeet vastaaviksi päätteiksi vasemmalta alkaen; tulos: $a^k b^k c^k$.

Esimerkiksi lause $aabbcc$ voitaisiin johtaa:

$$\begin{array}{lllllllll}
\underline{S} & \Rightarrow & L\underline{T} \Rightarrow LABC\underline{T} & \Rightarrow & LAB\underline{CA}BC & \Rightarrow & LA\underline{BA}CBC & \Rightarrow & LAAB\underline{CB}C \\
& \Rightarrow & \underline{LA}ABBCC & \Rightarrow & \underline{aA}BBCC & \Rightarrow & a\underline{aB}BCC & \Rightarrow & aa\underline{bB}CC \\
& \Rightarrow & aab\underline{bC}C & \Rightarrow & aabb\underline{cC} & \Rightarrow & aabbcc. & &
\end{array}$$

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

[1] Lause ε on tässä kieliopissa käsitelty erikoistapauksena.



Kuva 5.1: Rajoittamattoman kieliopin tuottaman kielen tunnistaminen Turingin koneella.

Lause 5.1 *Jos formaali kieli L voidaan tuottaa rajoittamattomalla kieliopilla, se voidaan tunnistaa 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 $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ 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 $[q_0x] \underset{G_M}{\Rightarrow}^* [uq_{\mathrm{acc}}v]$ joillakin $u, v \in \Gamma^*$.

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 $[uq_{\mathrm{acc}}v]$, jos ja vain jos M hyväksyy x:n.
3. Produktiot, joilla muotoa $[uq_{\mathrm{acc}}v]$ oleva merkkijono muutetaan tyhjäksi merkkijonoksi.

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

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

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:

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

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

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

[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:

$$\begin{array}{lcll} q_{\mathrm{acc}} & \to & E_L E_R & \\ q_{\mathrm{acc}}[& \to & E_R & \\ aE_L & \to & E_L & (a \in \Gamma) \\ [E_L & \to & \varepsilon & \\ E_R a & \to & E_R & (a \in \Gamma) \\ E_R] & \to & \varepsilon & \end{array}$$

□

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

Esimerkiksi sivun 90 rajoittamaton kielioppi on “melkein” yhteysherkkä: sen ainoa johdoksia lyhentävä produktio on $LA \to a$. Kielioppi voidaan muokata täsmälleen ehdot täyttävään muotoon korvaamalla välikepari LA yhdellä välikkeellä A_L, muuttamalla lähtösymboliin liittyvät produktiot muotoon $S \to A_L BCT \mid \varepsilon$ ja korvaamalla produktio $LA \to a$ produktiolla $A_L \to a$.

Nimitys “yhteysherkkä kielioppi” tulee tällaisten kielioppien eräästä normaalimuodosta, jossa produktiot ovat muotoa $S \to \varepsilon$ tai $\alpha A \beta \to \alpha \omega \beta$, missä A on kieliopin välike ja $\omega \neq \varepsilon$. Jälkimmäisen muotoisen produktion mukaan siis korvaussääntöä $A \to \omega$ saa soveltaa vain “kontekstissa” $\alpha \_ \beta$.

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

Lause 5.3 *Formaali kieli L on yhteysherkkä, 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 $\delta(q, \lhd) = (q', b, \Delta)$ olevia siirtymiä, missä $b \neq$ ‘$\lhd$’.* □

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 kuuluisaan “P = NP”-ongelmaan.

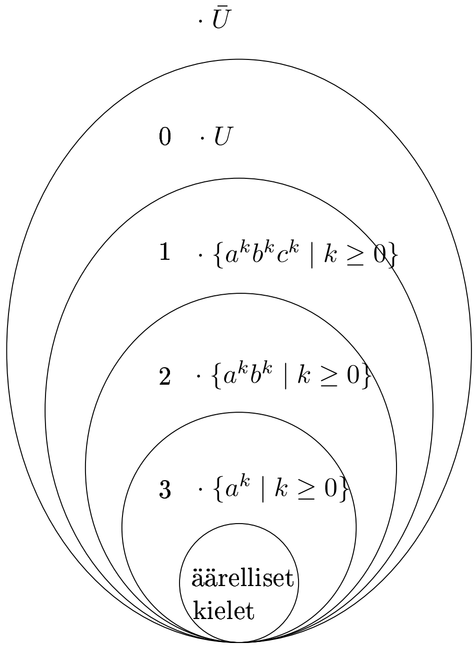
Kieliopit, niillä tuotettavat kielet ja vastaavat tunnistusautomaatit ryhmitellään usein ns. *Chomskyn luokkiin* seuraavasti:

luokka 3: oikealle ja vasemmalle lineaariset (säännölliset) kieliopit / säännölliset kielet / äärelliset automaatit;

luokka 2: yhteydettömät kieliopit / yhteydettömät kielet / pinoautomaatit;

luokka 1: yhteysherkät kieliopit / yhteysherkät kielet / lineaarisesti rajoitetut automaatit;

luokka 0: rajoittamattomat kieliopit / rajoittamattomilla kieliopeilla tuotettavat kielet (ns. rekursiivisesti numeroituvat kielet, ks. luku 6.1) / Turingin koneet.



Kuva 5.2: Chomskyn kieliluokat.

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.

Luku 6

Laskettavuusteoriaa

6.1 Rekursiiviset ja rekursiivisesti numeroituvat kielet

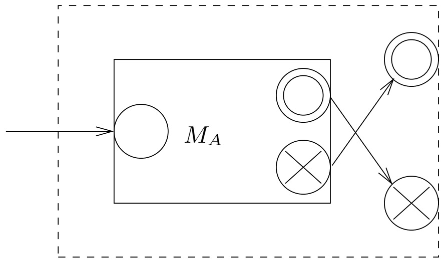
Churchin–Turingin teesin mukaan siis Turingin koneet ovat periaatteelliselta laskentakyvyltään yhtä vahvoja kuin mitkä tahansa mekaaniset laskulaitteet — erityisesti yhtä vahvoja kuin kaikki nykyiset tietokoneet. Seuraavassa tullaan kuitenkin osoittamaan, että Turingin koneiden laskentakyvyllä on vakavia 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.

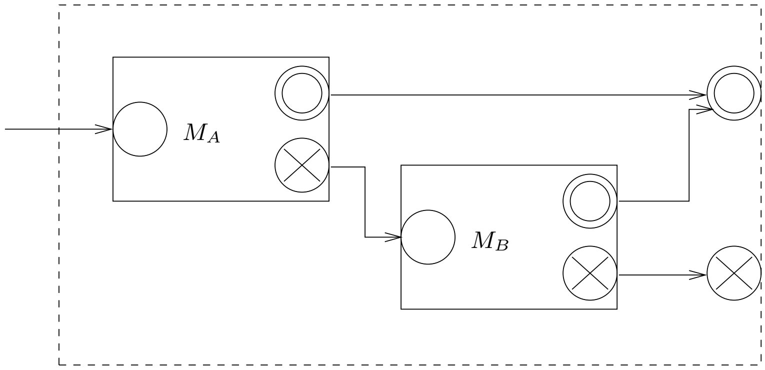
Määritelmä 6.1 Turingin kone $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ on *totaalinen*, jos se pysähtyy kaikilla syötteillä. Formaali kieli A on *rekursiivisesti numeroituva* (engl. recursively enumerable), jos se voidaan tunnistaa jollakin Turingin koneella, ja *rekursiivinen* (engl. recursive), jos se voidaan tunnistaa (so. ratkaista) jollakin totaalisella Turingin koneella.[1]

Palautetaan mieliin luvusta 1.6 (s. 11), että formaaleja kieliä voidaan tarkastella myös päätösongelmatyyppisten (binäärivasteisten) I/O-kuvausten esityksinä: tiettyä päätösongelmaa π esittävään kieleen A_π kuuluvat täsmälleen ne syötemerkkijonot x, joihin liittyy arvo $\pi(x) = 1$ ("kyllä"/"syöte OK"). Kielen A_π tunnistava Turingin kone on tällöin samalla ongelman π ratkaisualgoritmi: annetulla syötteellä x kone päätyy tilaan q_{acc}, jos $\pi(x) = 1$ ("syöte OK"), ja päätyy tilaan q_{rej} tai jää pysähtymättä, jos $\pi(x) = 0$ ("syöte ei kelpaa"). 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 numeroituva. Ongelma on siis ratkeava, jos sillä on kelvollinen, aina pysähtyvä ratkaisualgoritmi, ja osittain ratkeava, jos sillä on ratkaisualgoritmi, joka "kyllä"-tapauksissa

[1] Merkillisen tuntuisiin nimityksiin "rekursiivinen" ja "rekursiivisesti numeroituva" 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 numeroituva, jos se on tyhjä tai jonkin rekursiivisen funktion kuvajoukko. Samaistamalla formaalit kielet merkkijonojen kanonisen järjestyksen (ks. sivu 13) 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.

vastaa aina oikein, mutta "ei"-tapauksissa voi jäädä pysähtymättä. Ongelma, joka ei ole ratkeava, on *ratkeamaton* (engl. undecidable, unsolvable). (*Huom.*: ratkeamaton ongelma voi siis olla osittain ratkeava.)

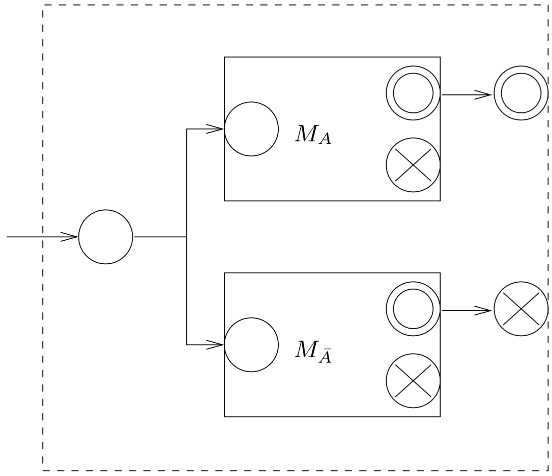
6.2 Rekursiivisten ja rekursiivisesti numeroituvien kielten perusominaisuuksia

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 numeroituvien 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 konetta M_B. Jälkimmäistä mahdollisuutta varten koneen M_A tulee toimia siten, että se pysähtyessään jättää nauhalle alkuperäisen syötteen ja täyttää lopun käyttämästään nauhan-



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

osasta 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 numeroituvia kieliä. Tällöin myös kielet $A \cup B$ ja $A \cap B$ ovat rekursiivisesti numeroituvia.*

Todistus. HT. □

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

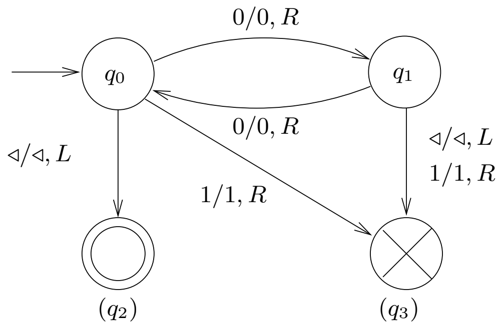
Huomautus. Luvussa 6.4 osoitetaan, että on olemassa rekursiivisesti numeroituvia, ei-rekursiivisia kieliä, so. osittain mutta ei totaalisesti ratkeavia päätösongelmia.

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 numeroituva kieli, joka ei ole rekursiivinen. Tällöin kieli $\bar{A}$ ei ole rekursiivisesti numeroituva.* □



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

6.3 Turingin koneiden kooodaus ja eräs ei rekursiivisesti numeroituva kieli

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

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$$

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

- Oletetaan, että $Q = \{q_0, q_1, \ldots, q_n\}$, missä $q_{\mathrm{acc}} = q_{n-1}$ ja $q_{\mathrm{rej}} = q_n$; ja että $\Gamma \cup \{\triangleright, \triangleleft\} = \{a_0, a_1, \ldots, a_m\}$, missä $a_0 = 0$, $a_1 = 1$, $a_2 = \triangleright$ ja $a_3 = \triangleleft$. 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:

$$c_M = 111\,\underbrace{010100010100}_{\delta(q_0,0)=(q_1,0,R)}\,11\,\underbrace{010010000100100}_{\delta(q_0,1)=(q_3,1,R)}\,11\ldots$$

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 numeroituvista kielistä. Koneet ovat M_ε, M_0, M_1, M_{00}, $M_{01}, \ldots$, ja kielet vastaavasti $L(M_\varepsilon)$, $L(M_0)$, $L(M_1)$, $\ldots$ (indeksit kanonisessa järjestyksessä). Kukin kieli voi esiintyä luettelossa monta kertaa.

Cantorilaisella "diagonalisointitekniikalla" (vrt. lause 1.8) 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 numeroituva.

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

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

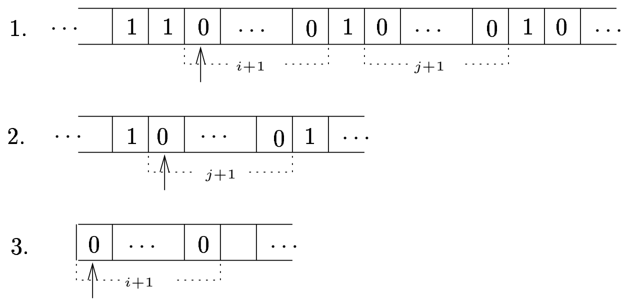
Ristiriidasta seuraa, että kieli D ei voi olla rekursiivisesti numeroituva. □

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 1.8 formaalin kielen $\tilde{A}$: jos kielten $L(M_\varepsilon)$, $L(M_0)$, $L(M_1)$, $\ldots$ karakteristiset funktiot esitetään taulukkona, niin kieli D poikkeaa kustakin kielestä taulukon "diagonaalilla":[2]

$$\begin{array}{c|ccccc}
\overset{D}{\searrow} & L(M_\varepsilon) & L(M_0) & L(M_1) & L(M_{00}) & \cdots \\
\hline
\varepsilon & \overset{1}{\cancel{0}} & 0 & 0 & 0 & \cdots \\
0 & 0 & \overset{0}{\cancel{1}} & 1 & 0 & \cdots \\
1 & 0 & 0 & \overset{0}{\cancel{1}} & 1 & \cdots \\
00 & 0 & 0 & 0 & \overset{1}{\cancel{0}} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

[2] Kuva on tosin tässä hieman harhaanjohtava siinä suhteessa, että koska mikään koodeista ε, 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.

6.4 Universaalikieli U ja universaalit Turingin koneet

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

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

Kieli U sisältää yksinkertaisella tavalla koodattuina tiedot kaikista aakkoston $\{0, 1\}$ rekursiivisesti numeroituvista kielistä. Olkoon nimittäin $A \subseteq \{0, 1\}^*$ jokin rekursiivisesti numeroituva 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 numeroituva. Kielen U tunnistavia Turingin koneita sanotaan *universaaleiksi Turingin koneiksi* (engl. universal Turing machines); seuraavassa todistuksessa hahmotellaan yhden tällaisen koneen konstruktio.

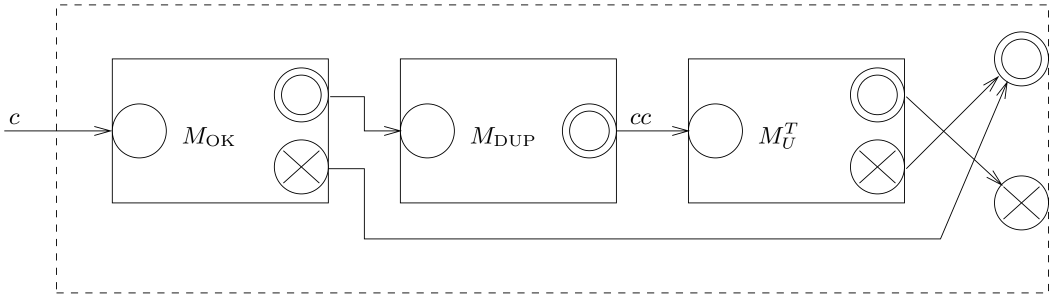
Lause 6.6 *Kieli U on rekursiivisesti numeroituva.*

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 $w = a_1 a_2 \dots a_k \in \{0, 1\}^*$ kakkosnauhalle muodossa $00010^{a_1+1}10^{a_2+1}1 \dots 10^{a_k+1}10000$.

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}



Kuva 6.6: Diagonaalikielen D tunnistava kone M_D.

(mahdollisesti siirtäen nauhojen loppupäiden sisältöjä vasemmalle tai oikealle), ja siirtää kakkosnauhan nauhapäätä yhden simuloidun merkin vasemmalle, jos $t = 0$, ja oikealle, jos $t = 1$.

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_{acc} tai q_{rej}. $\square$

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

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

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

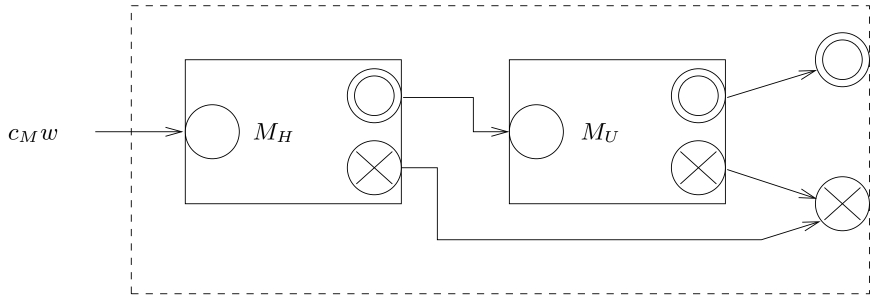
Seuraus 6.8 *Kieli*

$$\tilde{U} = \{c_M w \mid w \notin L(M)\}$$

ei ole rekursiivisesti numeroituva.

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

$$\mathrm{ERR} = \{x \in \{0,1\}^* \mid x \text{ ei sisällä alkuosanaan kelvollista Turingin koneen koodia}\}.$$



Kuva 6.7: Universaalikielen U tunnistaminen koneen M_H avulla.

Jos siis kieli $\tilde{U}$ olisi rekursiivisesti numeroituva, olisi samoin lauseen 6.2 nojalla myös kieli $\bar{U}$. Koska kieli U tiedetään rekursiivisesti numeroituvaksi (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 numeroituva. □

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*

$$H = \{c_M w \mid M \textit{ pysähtyy syötteellä } w\}$$

on rekursiivisesti numeroituva, mutta ei rekursiivinen.

Todistus. Todetaan ensin, että kieli H on rekursiivisesti numeroituva. 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.

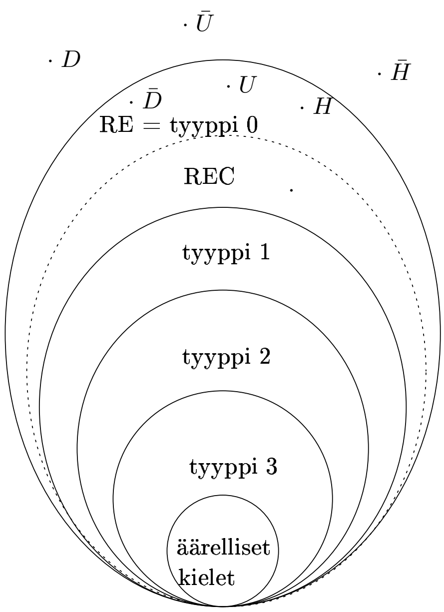
Osoitetaan sitten, että kieli H ei ole rekursiivinen. Oletetaan nimittäin, että olisi $H = L(M_H)$ jollakin totaalisella Turingin koneella M_H. Oletetaan lisäksi (tämä ei merkitse lisärajoitusta), että kone M_H pysähtyessään jättää nauhalle alkuperäisen syötteensä, mahdollisesti tyhjämerkeillä # jatkettuna. 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*

$$\tilde{H} = \{c_M w \mid M \textit{ ei pysähdy syötteellä } x\}$$

ei ole rekursiivisesti numeroituva.

Todistus. Päätellään kuten seurauslauseessa 6.8. □



Kuva 6.8: Chomskyn kielihierarkia ja rekursiiviset kielet.

6.6 Rekursiiviset kielet ja Chomskyn kieliluokat

Merkitään rekursiivisesti numeroituvien 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 yhteydettömät kielet (Chomskyn luokka 1) ovat rekursiivisia, ja että on olemassa rekursiivisia kieliä, jotka eivät ole yhteysherkkiä — tosin on sangen vaikea löytää *luonnollista* esimerkkiä kielestä, joka kuuluisi näiden luokkien erotukseen. Kuvassa 6.8 on esitetty kaavio Chomskyn kielihierarkiasta täydennettynä rekursiivisten kielten luokalla.

6.7 * Lisää ratkeamattomia ongelmia

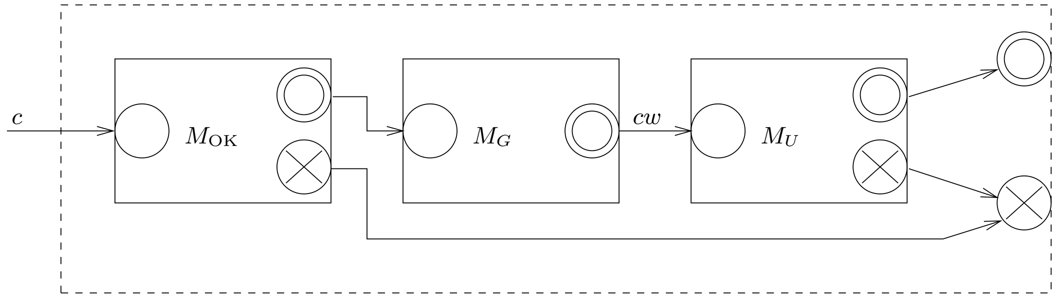
Yllättävän monet tietojenkäsittelyongelmat ovat ratkeamattomia. Seuraavassa osoitetaan, että esimerkiksi jokseenkin kaikki ohjelmien toimintaa, tai tarkemmin sanoen niiden laskemia syöte/tulos-kuvauksia koskevat kysymykset ovat ratkeamattomia. Johdantona tähän yleiseen tulokseen, ns. Ricen lauseeseen, tarkastellaan ensin yhtä sen erikoistapausta.

Epätyhjyysongelma

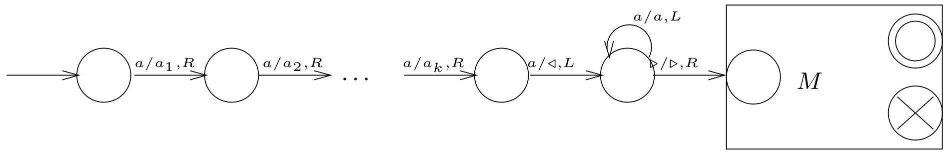
Tarkastellaan päätösongelmaa “Hyväksyykö annettu Turingin kone yhtään syötemerkkijonoa?” Ongelman esitys formaalina kielenä on

$$\text{NE} = \{c \in \{0,1\}^* \mid L(M_c) \neq \emptyset\}$$

(NE $\sim$ engl. nonempty). Osoittautuu, että tämäkin ongelma on ratkeamaton.



Kuva 6.9: Turingin koneen M_{NE} rakenne.



Kuva 6.10: Turingin koneen M^w rakenne.

Lause 6.11 *Kieli NE on rekursiivisesti numeroituva, mutta ei rekursiivinen.*

Todistus. Todetaan ensin, että kieli NE on rekursiivisesti numeroituva muodostamalla sille tunnistajakone M_{NE}. Kone M_{NE} on helpointa suunnitella epädeterministisenä; se voidaan tarvittaessa determinisoida lauseen 4.3 mukaisesti.

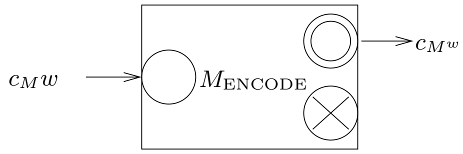
Olkoon M_{OK} jo lauseen 6.7 todistuksessa esiintynyt Turingin kone, joka testaa onko annettu syöte kelvollinen Turingin koneen koodi, ja olkoon M_G epädeterministinen Turingin kone, joka kirjoittaa nauhalla jo olevan merkkijonon perään mielivaltaisen binäärijonon w. Kone M_{NE} voidaan muodostaa yhdistämällä koneet M_{OK}, M_G ja universaalikone M_U (lauseesta 6.6) kuvan 6.9 esittämällä tavalla.

Selvästi on:

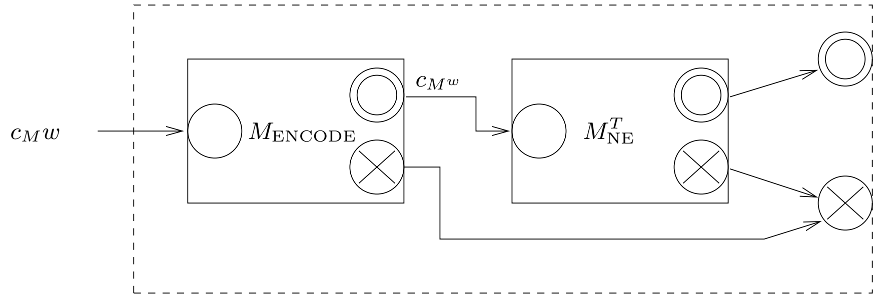
$$\begin{aligned} c \in L(M_{\mathrm{NE}}) \quad &\Leftrightarrow \quad c \text{ on kelvollinen Turingin koneen koodi ja on olemassa } w \text{ s.e. } cw \in U \\ &\Leftrightarrow \quad c \text{ on kelvollinen Turingin koneen koodi ja on olemassa } w \text{ s.e. } w \in L(M_c) \\ &\Leftrightarrow \quad L(M_c) \neq \emptyset. \end{aligned}$$

Osoitetaan sitten, että kieli NE ei ole rekursiivinen. Tämä nähdään olettamalla, että kielellä NE olisi totaalinen tunnistajakone M_{NE}^T, ja muodostamalla tämän perusteella totaalinen tunnistajakone M_U^T kielelle U. Koska U ei ole rekursiivinen (lause 6.7), tämä on mahdottomuus ja osoittaa, että konetta M_{NE}^T ei voi olla olemassa.

Koneen M_U^T konstruointi koneesta M_{NE}^T 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 $\{0, 1, \triangleleft\}$.



Kuva 6.11: Turingin koneen M_{ENCODE} toiminta.



Kuva 6.12: Turingin koneen M^T_U rakenne.

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:

$$L(M^w) = \begin{cases} \{0,1\}^*, & \text{jos } w \in L(M); \\ \emptyset, & \text{jos } w \notin L(M). \end{cases}$$

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.

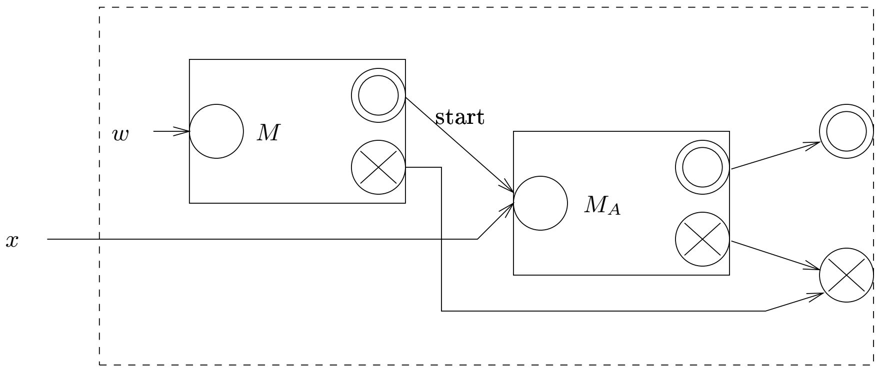
Universaalikielelle U voitaisiin nyt koneet M_{ENCODE} ja hypoteettinen M^T_{NE} kuvan 6.12 esittämällä tavalla yhdistämällä muodostaa totaalinen tunnistajakone M^T_U. Selvästi kone M^T_U on totaalinen, jos M^T_{NE} on, ja $L(M^T_U) = U$, koska:

$$\begin{aligned} & c_M w \in L(M^T_U) \\ \Leftrightarrow\ & c_{M^w} \in L(M^T_{\text{NE}}) = \text{NE} \\ \Leftrightarrow\ & L(M^w) \neq \emptyset \\ \Leftrightarrow\ & w \in L(M). \end{aligned}$$

Mutta kieli U ei ole rekursiivinen, joten tällainen totaalinen tunnistajakone M^T_U ei ole mahdollinen. Saadusta ristiriidasta päätellään, että myöskään kielellä NE ei voi olla totaalista tunnistajaa M^T_{NE}. $\Box$

Ricen lause

Edellisen lauseen todistustekniikkaa hieman yleistämällä voidaan todistaa seuraavassa esitettävä erittäin vahva ratkeamattomuustulos, ns. *Ricen lause*.



Kuva 6.13: Turingin koneen M^w rakenne (Ricen lause).

Sanotaan Turingin koneen M *semanttiseksi ominaisuudeksi* mitä tahansa sellaista ominaisuutta, joka riippuu vain koneen M tunnistamasta kielestä, ei sen syntaktisesta rakenteesta. 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 numeroituvia 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

$$\text{codes}(\mathcal{S}) = \{c \mid L(M_c) \in \mathcal{S}\}$$

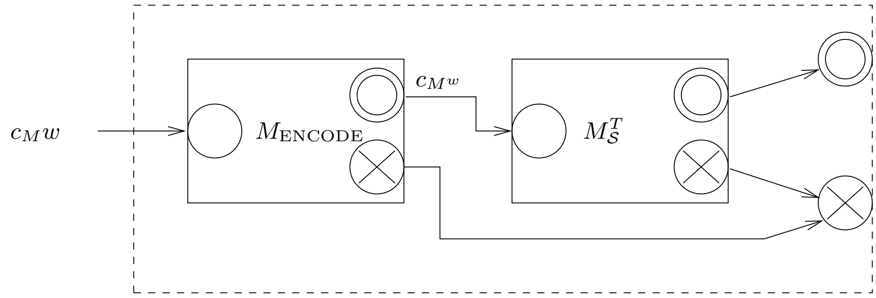
on rekursiivinen. Toisin sanoen: ominaisuus on ratkeava, jos annetusta Turingin koneen koodista voidaan algoritmisesti päätellä, onko koneella kysytty semanttinen ominaisuus.

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 x:n



Kuva 6.14: Turingin koneen M_U^T rakenne (Ricen lause).

(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

$$\begin{aligned} & c_M w \in L(M_U^T) \\ \Leftrightarrow\ & c_{M^w} \in L(M_{\mathcal{S}}^T) = \text{codes}(\mathcal{S}) \\ \Leftrightarrow\ & L(M^w) \in \mathcal{S} \\ \Leftrightarrow\ & w \in L(M). \end{aligned}$$

Koska kieli U ei ole rekursiivinen, tämä on mahdotonta, mistä päätellään että ominaisuus $\mathcal{S}$ ei voi olla ratkeava. $\square$

6.8 * Ratkeamattomuustuloksia muilla aloilla

Ricen lauseen mukaan siis jokseenkin kaikki Turingin koneiden — tai ekvivalentisti C-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

6.9 * Rekursiiviset funktiot

Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ mielivaltainen standardimallinen Turingin kone. Määritellään koneen M ***laskema osittaiskuvaus*** (t. ***-funktio***)

$$f_M : \Sigma^* \to \Gamma^*$$

seuraavasti:

$$f_M(x) = \begin{cases} u, \text{ jos } (q_0, \underline{x}) \underset{M}{\vdash}^* (q, u\underline{a}v) \text{ jollakin } q \in \{q_{\text{acc}}, q_{\text{rej}}\}, av \in \Gamma^*; \\ \text{määrittelemätön, muuten.} \end{cases}$$

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 $M = (Q, \Sigma, \Gamma, \dots)$, 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*

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

on rekursiivinen funktio.

(ii) *Kieli $A \subseteq \Sigma^*$ on rekursiivisesti numeroituva, 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). □

6.10 * Rekursiiviset palautukset ja RE-täydelliset kielet

Yksi laskettavuusteorian keskeisiä teemoja 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 ongelman A syöte x voidaan rekursiivisesti muuntaa ongelman B syötteeksi $f(x)$, 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) *$A \leq_m A$;*

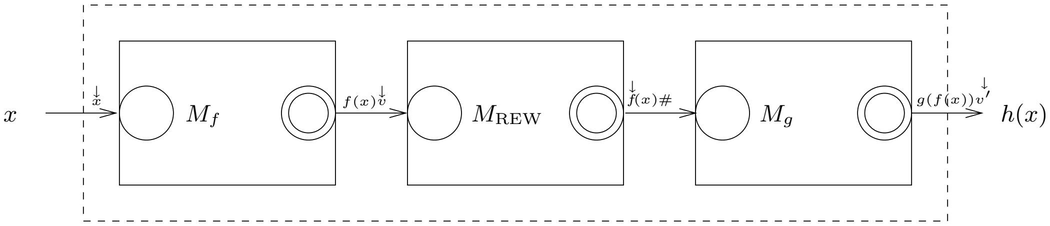
(ii) *jos $A \leq_m B$ ja $B \leq_m C$, niin $A \leq_m C$;*

(iii) *jos $A \leq_m B$ ja B on rekursiivisesti numeroituva, niin A on rekursiivisesti numeroituva;*

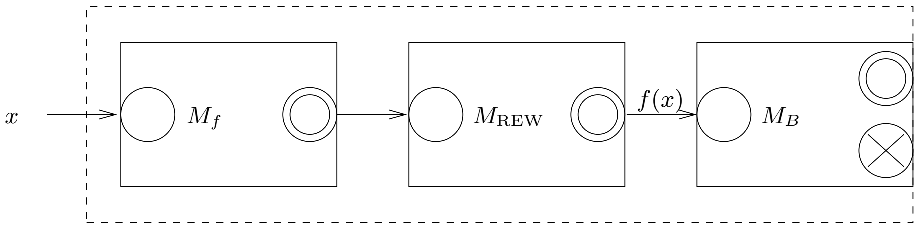
(iv) *jos $A \leq_m B$ ja B on rekursiivinen, niin A on rekursiivinen.*

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 $\lhd$ 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:

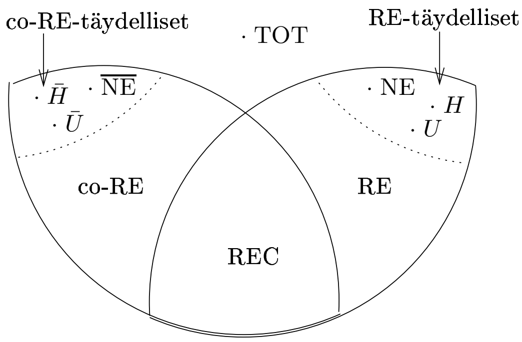
$$\begin{aligned} \mathrm{RE} &= \{\text{aakkoston } \{0,1\} \text{ rekursiivisesti numeroituvat kielet}\}; \\ \mathrm{REC} &= \{\text{aakkoston } \{0,1\} \text{ rekursiiviset kielet}\}. \end{aligned}$$

Kieli $A \subseteq \{0,1\}^*$ on *RE-täydellinen* (engl. RE-complete), jos

(i) $A \in \mathrm{RE}$ ja

(ii) $B \leq_m A$ kaikilla $B \in \mathrm{RE}$.

RE-täydelliset kielet ovat siis “maksimaalisen vaikeita” luokassa RE, tai täsmällisemmin sanoen ne sisältävät “rekursiivisesti koodattuina” tiedot kaikista luokan RE kielistä.



Kuva 6.17: Kieliluokkien RE, co-RE ja REC suhteet.

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. □

Lauseen 6.17 ja lemman 6.18 perusteella voidaan todeta, että luvussa 6.7 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):

$$f(x) = \begin{cases} c_{M^w}, & \text{jos } x = c_M w; \\ \varepsilon, & \text{jos } x \text{ ei ole vaadittua muotoa.} \end{cases}$$

Itse asiassa Ricen lauseen (lause 6.12) todistus osoittaa, että ***kaikki*** Turingin koneiden semanttisiin ominaisuuksiin $\mathcal{S}$ liittyvät kielet codes($\mathcal{S}$), joilla $\emptyset \notin \mathcal{S}$ ja codes($\mathcal{S}$) $\in$ RE ovat RE-täydellisiä. Yleensäkin näyttää jostakin syystä olevan niin, että kaikki "luonnolliset" rekursiivisesti numeroituvat, 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.* □

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.

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

Kirjallisuutta

Tässä monisteessa käsitellystä automaattien, kielioppien ja laskettavuuden teoriasta on tarjolla myös lukuisia englanninkielisiä yleisesityksiä, esimerkiksi seuraavat:

D. I. A. Cohen: *Introduction to Computer Theory, 2nd Ed.* John Wiley & Sons, New York, NY, 1996.

E. Gurari: *An Introduction to the Theory of Computation.* Computer Science Press, Rockville, MD, 1989.
`http://www.cis.ohio-state.edu/~gurari/theory-bk/theory-bk.html`

J. E. Hopcroft, R. Motwani, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation, 2nd Ed.* Addison-Wesley, Reading, MA, 2001.

E. Kinber, C. Smith: *Theory of Computing: A Gentle Introduction.* Prentice-Hall, Upper Saddle River, NJ, 2001.

D. C. Kozen: *Automata and Computability.* Springer-Verlag, New York, NY, 1997.

H. R. Lewis, C. H. Papadimitriou: *Elements of the Theory of Computation, 2nd Ed.* Prentice-Hall, Upper Saddle River, NJ, 1998.

J. C. Martin: *Introduction to Languages and the Theory of Computation, 3rd Ed.* McGraw-Hill, New York, NY, 2002.

M. Sipser: *Introduction to the Theory of Computation, 2nd Ed.* Thomson Course Technology, 2005.

T. A. Sudkamp: *Languages and Machines: An Introduction to the Theory of Computer Science, 3rd Ed.* Addison-Wesley, Reading, MA, 2005.

Näistä vaihtoehdoista on ehkä Sipserin kirja kokonaisuutena ottaen suositeltavin näkemyksellisen, mutta kuitenkin lukijaystävällisen esitystapansa ansiosta. Teokseen sisältyy myös laajahko johdatus tämän monisteen ulkopuolelle jätettyyn laskennan vaativuusteorian alaan. Teoksen virtaviivaisen esityksen kääntöpuoli toisaalta on, että se sivuuttaa monet todistusten tekniset yksityiskohdat aihepiiriä vakavasti opiskelevan kannalta ehkä liian kevyesti, eikä myöskään juuri käsittele teoksen päälinjasta syrjässä olevia tärkeitäkään aiheita. (Esimerkiksi äärellisten automaattien minimointi sivuutetaan yhdellä kirjan loppuosaan piilotetulla harjoitustehtävällä.)

Esitykseltään huolellisempia, mutta vastaavasti raskaslukuisempia ovat esimerkiksi Hopcroftin/Motwanin/Ullmanin, Lewisin/Papadimitrioun ja Martinin kirjat. Myös Kozenin, Sudkampin ja Gurarin kirjat ovat erittäin suositeltavia. Gurarin teos lähestyy aihepiiriä mielenkiintoisella tavalla ohjelmoinnin näkökulmasta ja Kozenin kirjan esitystapa on muita hieman algebrallisempi. Listan helppolukuisimmat johdatusteokset ovat Cohenin ja Kinberin/Smithin kirjat.

Näitä perusoppikirjoja pidemmälle menevästä eri tutkimusalojen erikoiskirjallisuudesta mainittakoon esimerkkeinä seuraavat:

Formaalit kielet, kieliopit ja automaatit

M. A. Harrison: *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.

J. van Leeuwen (Ed.): *Handbook of Theoretical Computer Science. Vol. B: Formal Models and Semantics.* Elsevier, Amsterdam, 1990.

G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages I–III.* Springer-Verlag, Berlin Heidelberg 1997.

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, J. Burgess: *Computability and Logic, 4th Ed.* Cambridge University Press, 2002.

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 I–II.* Elsevier–North-Holland, Amsterdam, 1989/1999.