Hauskaa ja havainnollista laskennateoriaa!

Wilhelmiina Hämäläinen
Luentomateriaalia TEPE-kurssille 26.1. 2004
Tietojenkäsittelytieteen laitos
Joensuun yliopisto

Kiitokset

Kiitän professori Pekka Orposta, joka on ystävällisesti antanut käyttööni luentomonisteensa "Laskennateoria" latex-tiedostot. Hänen luentomonisteensa on toiminut monin paikoin esikuvanani ja olen pyrkinyt pysymään uskollisena hänen loogiselle ja selkeälle esitystavalleen, vaikka olenkin tietoisesti tavoitellut hauskaa ja helppolukuista esitystä.

Professori Osposen lisäksi kiitän saamastani konsultaatiosta dosentti Patrik Florenia ja FT Matti Luukkaista. Kaikkein suurimmat kiitokset kuuluvat kuitenkin assistenteilleni ja oppilailleni, joiden tarpeet, toiveet ja palaute ovat suurelta osaltaan vaikuttaneet monisteen syntyprosessiin.

Sisältö

Luku 1

Matemaattisia peruskäsitteitä

Tässä luvussa kerrataan lyhyesti joitain matemaattisista käsitteitä kuten loogiset symbolit, joukot, relaatiot ja funktiot, joukkojen numeroituvuus ja ylinumeroituvuus sekä esitellään hyödyllisiä matemaattisia todistusmenetelmiä.

1.1 Loogisia symboleita

Olkoon P ja Q *propositioita* eli totuusarvoisia lauseita, jotka kuvaavat jonkin asiantilan. Esimerkiksi P="Kuu on juustoa", Q="Napoleon asuu Kuussa". Tällöin P:stä ja Q:sta voidaan muodostaa uusia, rakenteisia propositioita seuraavilla loogisilla operaattoreilla:

- *Negaatio* $\neg P$: P on epätosi (ohjelmointikielissä not P, $!P$)
- *Disjunktio* $P \vee Q$: joko P tai Q on tosi (tai molemmat ovat tosia) (ohjelmointikielissä P or Q, $P||Q$)
- *Konjunktio* $P \wedge Q$: sekä P että Q ovat tosia (P and Q, $P\&\&Q$)
- *Implikaatio* $P \Rightarrow Q$: "jos P, niin Q" ($\equiv \neg P \vee Q$)
- *Ekvivalenssi* $P \Leftrightarrow Q$: "P jos ja vain jos Q" ($\equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$)

Lisäksi usein tarvitaan symboleita $\forall$ (*universaalikvanttori*, "kaikille", "jokaiselle") sekä $\exists$ (*eksistenssikvanttori*, "on olemassa", "jollekin"). Esimerkiksi $\forall x, x \in \mathbb{N}$ "jokaiselle luonnolliselle luvulle x pätee...", $\exists x, x \in \mathbb{N}$ "jollekin luonnolliselle luvulle x pätee..."

1.2 Joukot

Joukko on kokoelma olioita eli joukon *alkioita*, esimerkiksi $A = \{a_1, a_2, ..., a_n\}$ tai $\Sigma = \{a, b, c, ..., z\}$. Merkitään $a_i \in A$ ("a_i kuuluu joukkoon A"). Joukon erikoistapaus on *tyhjä joukko* $\emptyset$, joka ei sisällä yhtään alkiota. *Perusjoukko* on (asiayhteydestä riippuva) kokoelma oliota, joka on valittu tarkastelun kohteeksi, esimerkiksi luonnolliset luvut $\mathbb{N}$, reaaliluvut $\mathbb{R}$, tai kirjainmerkit.

Joukkojen välillä voi vallita seuraavia suhteita:

- A on B:n *osajoukko*: $A \subseteq B \Leftrightarrow \forall x(x \in A \rightarrow x \in B)$.
- A on B:n *aito osajoukko*: $A \subset B$: $A \subseteq B \wedge A \neq B$.
- A:n ja B:n *yhdiste*: $A \cup B = \{x | x \in A \vee x \in B\}$.
- A:n ja B:n *leikkaus*: $A \cap B = \{x | x \in A \wedge x \in B\}$.
- A:n ja B:n *erotus*: $A \setminus B = \{x | x \in A \wedge x \notin B\}$.
- A:n *komplementti* perusjoukossa E: $\overline{A} = E \setminus A$.
- A:n ja B:n *karteesinen tulo*: $A \times B = \{(x, y) | x \in A \wedge y \in B\}$, missä kukin (x, y) on *järjestetty pari*.
- joukon A *potenssijoukko*: $\mathcal{P}(A) = \{X | X \subseteq A\}$.
  esim. jos $A = \{a, b, c\}$, niin $\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

1.3 Relaatiot

Relaatio R kuvaa kahden (tai useamman) joukon alkioiden välillä vallitsevaa suhdetta. Esimerkiksi joukkojen A ja B välinen (binääri-)relaatio määritellään joukon $A \times B$ osajoukkona:

$$R = \{(a, b) | a \in A \wedge b \in B \wedge R(a, b)\}.$$

Esimerkiksi olkoon A ja B luonnollisia lukuja ja R on seuraajarelaatio:

$$R(a, b), \text{ jos ja vain jos } b = a + 1.$$

Tällöin relaatio koostuu pareista $\{(0, 1), (1, 2), (2, 3), ...\}$. Relaation $R \subseteq A \times B$ *käänteisrelaatio* $R^{-1} \subseteq B \times A$ on relaatio

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

1.4 Funktiot

Funktio on relaation erikoistapaus, joka liittää annetun joukon jokaiseen alkioon toisen annetun joukon määrätyn alkion. Määritellään funktio seuraavasti:

Määritelmä 1.1 *Olkoon f joukkojen A ja B välinen relaatio $f \subseteq A \times B$, joka täyttää seuraavat kaksi ehtoa:*

1. *(Määrittelyehto) Jokaista A:n alkiota x vastaa B:n alkio y eli*

$$\forall x \in A \; \exists y \in B \; (x, y) \in f.$$

2. *(Yksikäsitteisyysehto) Jokaista A:n alkiota x vastaa <u>vain yksi</u> alkio y B:ssä eli*

$$\forall x \in A, \forall y_1, y_2 \in B \; ((x, y_1) \in f \wedge (x, y_2) \in f \Rightarrow y_1 = y_2.$$

Tällöin f on funktio eli kuvaus $f : A \to B$ määrittelyjoukosta A maalijoukkoon B(kuva 1.1).

Mikäli funktio f kuvaa pisteen $x \in A$ pisteelle $y \in B$, sanotaan y:tä x:n *kuvaksi*. Merkitään $y = f(x) \Leftrightarrow (x, y) \in f$. Jos $C \subseteq A$, niin joukko $f(C)$ on C:n kuvajoukko

$$f(C) = \{f(x) | x \in C\}.$$

Funktion $f : A \to B$ *arvojoukko* on määrittelyjoukon A kuvajoukko $f(A)$.

Joukko $f^{-1}(D)$ on D:n *alkukuva*

$$f^{-1}(D) = \{x | f(x) \in D\},$$

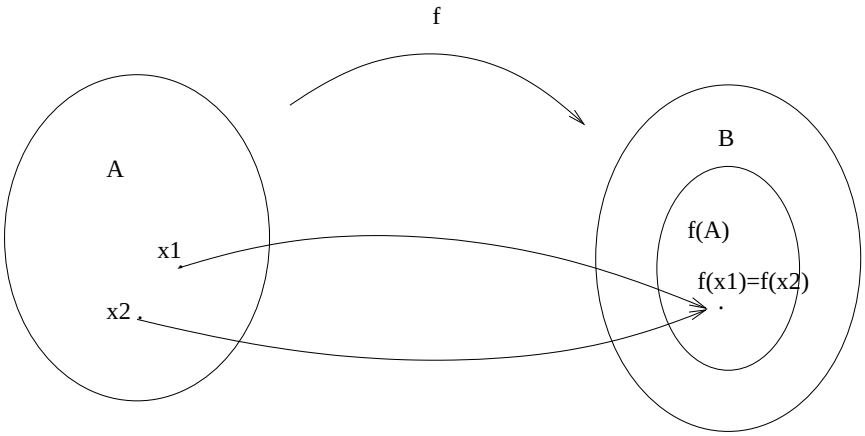
missä f^{-1} on f:n käänteisrelaatio. Relaatio f^{-1} on f:n *käänteisfunktio* eli *käänteiskuvaus*, jos f^{-1} täyttää funktion ehdot.

Funktioiden erikoistapauksia ovat *injektio*, *surjektio* ja *bijektio*, jotka määritellään seuraavasti:

Funktio $f : A \to B$ on

- **injektio**, jos jokaisella arvojoukon kuva-alkiolla on vain yksi alkukuva eli

$$\forall x_1, x_2 \in A \; (f(x_1) = f(x_2) \Rightarrow x_1 = x_2).$$



Kuva 1.1: A=funktion määrittelyjoukko, B=maalijoukko, f(A)=arvojoukko. f ei ole injektio, koska x1 ja x2 kuvautuvat samalle pisteelle, eikä surjektio, koska kaikilla B:n pisteillä ei ole alkukuvaa A:ssa.

- **surjektio**, jos jokaisella B:n alkiolla on alkukuva A:ssa eli

$$\forall y \in B \; \exists x \in A(y = f(x)).$$

- **bijektio**, jos f on sekä surjektio että injektio.

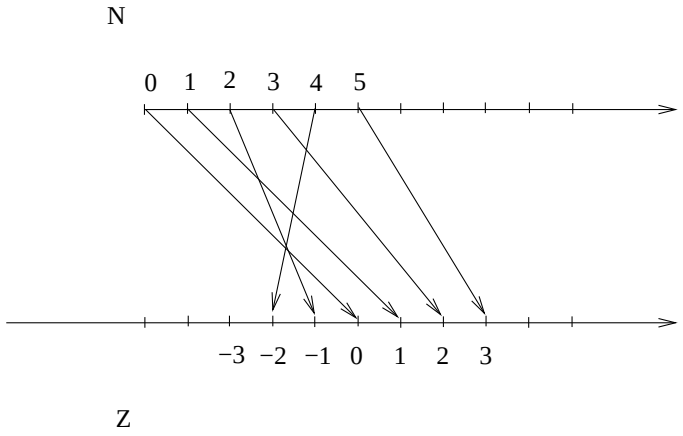
  Huom! f on bijektio $\Leftrightarrow$ f:llä on käänteisfunktio.

- esim. 1. $f : \mathbb{Z} \to \mathbb{Z}^+ \cup \{0\}$, $f(x) = |x|$ on surjektio, mutta ei injektio.
- esim. 2. $f : \mathbb{Z}^+ \to \mathbb{Q}$, $f(x) = \frac{1}{x}$ on injektio, mutta ei surjektio.
- esim. 3. $f : \mathbb{R}^+ \to \mathbb{R}^+$, $f(x) = x^2$ on bijektio, jonka käänteiskuvaus on $f(y) = \sqrt{y}$.

1.5 Numeroituvuus

Määritelmä 1.2 *Joukko X on* numeroituva*, jos*

1. *X on äärellinen tai*



Kuva 1.2: Esimerkiksi kokonaisluvut voidaan numeroida luonnollisilla luvuilla.

2. On olemassa bijektio $f : \mathbb{N} \to X$, jolle

$$X = \{f(n) | n \in \mathbb{N}\}.$$

Intuitiivisesti ajatellen X:n alkiot voidaan siis järjestää ja indeksoida luonnollisilla luvuilla: $X = \{x_0, x_1, x_2, ...\}$. Esimerkiksi parittomien luonnollisten lukujen joukko $X = \{1, 3, 5, ...\}$ on numeroituva, sillä voidaan määritellä bijektio $f : \mathbb{N} \to X$

$$f(n) = 2n + 1.$$

Jos X ei ole numeroituva, se on *ylinumeroituva*. Esimerkiksi reaalilukujen joukko $\mathbb{R}$ sekä luonnollisten lukujen joukon $\mathbb{N}$ potenssijoukko $\mathcal{P}(\mathbb{N})$ ovat ylinumeroituvia.

1.6 Todistusmenetelmiä

Seuraavassa esitellään joitain hyödyllisiä todistusmenetelmiä.

Matemaattinen induktio

Halutaan todistaa, että jokin väite $P(n)$ pätee kaikille luonnollisille luvuille. Väite todistetaan kahdessa osassa:

1. Perustapaus $n = 0$: Todistetaan, että väite pätee tapauksessa $P(0)$

2. Induktioaskel: Todistetaan, että kaikilla n $P(n) \rightarrow P(n+1)$

Kohdista 1+2 seuraa, että väite pätee kaikilla $n \in \mathbb{N}$ ($P(0)$ pätee, josta saadaan induktioaskeleella $P(1)$, tästä puolestaan $P(2)$ jne.) Induktioaskel todistetaan yleensä *induktio-oletuksen* avulla:

- Oletetaan, että $P(n)$ pätee kaikilla $n \leq k$.
- Osoitetaan väite tapauksessa $n = k + 1$.

Esimerkki 1.1 *Väite:* $\Sigma_{i=1}{}^{n} i = \frac{n^2+n}{2} \forall n \geq 0$
Todistus:

1. $n = 0$ $\Sigma_{i=1}^{n} i = 0 = \frac{0^2+0}{2}$
2. *Induktio-oletus:* $\exists k \in \mathbb{N}$ *s.e. väite pätee kaikilla* $n \leq k$
   Tapaus $n = k + 1 : \Sigma_{i=1}^{k+1} i = 1 + 2 + 3 + ... + k + (k + 1)$
   Ol. mukaan $\Sigma_{i=1}^{k} i = \frac{k^2+k}{2}$, *joten*
   $\Sigma_{i=1}^{k+1} i = \frac{k^2+k}{2} + (k + 1) = \frac{(k+1)^2+(k+1)}{2}$ $\square$

Epäsuora todistus (Todistus antiteesillä eli vastaväitteellä)

Halutaan todistaa väite ”jos P, niin Q”. Oletetaan P ja tehdään vastaväite $\neg Q$. Mikäli näistä voidaan johtaa ristiriita, on väite tosi.

- Idea: implikaatio $P \Rightarrow Q$ tosi muulloin, paitsi kun $P \wedge \neg Q$ ($P \Rightarrow Q \equiv (\neg P \vee Q) \equiv \neg(P \wedge \neg Q)$)
- Huom! Ei tiedetä, mikä ristiriita halutaan johtaa!

Esimerkki 1.2 *Olkoon U ääretön joukko ja S sen äärellinen osajoukko ja T S:n komplementti U:ssa.*

Väite: T on ääretön.
Todistus: Vastaväite: T on äärellinen. Koska sekä S ja T ovat äärellisiä, on myös U äärellinen, mikä merkitsee ristiriitaa (U oli ääretön). $\square$

Tehtävä: Kuinka todistaisit väitteen: ”Maailmassa on vähintään kaksi ihmistä, joilla on yhtä monta hiusta päässään”?

Kontrapositiolla todistaminen

Halutaan todistaa väite "jos P, niin Q". Todistetaan sen sijaan ekvivalentti lause "jos ei Q, niin ei P" ($P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$)

- Voidaan ajatella antiteesimenetelmän erikoistapauksena. Oletetaan P ja $\neg Q$ ja yritetään johtaa $\neg P$ – nyt siis tiedetään, mikä ristiriita halutaan johtaa $(P \wedge \neg P)$.

Eksistenssi- ja universaalilauseiden todistuksesta

Muotoa "On olemassa $x \in X$, jolle pätee..." ja "Kaikille $X \in X$ pätee..." olevat väitteet voidaan joskus todistaa suoraan.

- Eksistenssilause $\exists x \in X\ P(x)$: konstruoi tällainen x (arvaa, tuota, keksi generoiva algoritmi jne.) Huom! Osoitettava, että haluttu ominaisuus myös pätee!
- esim. väite "Himalajalla on vuori, joka on korkeampi kuin mikään muu vuori maailmassa."
- Universaalilause $\forall x \in X\ P(x)$: valitse mielivaltainen x X:stä ja osoita, että sille pätee haluttu ominaisuus $P(X)$.
- esim. Olk. $S = \{x \in \mathbb{R} | (x^2 - 3x + 2 \leq 0)\}$ ja $T = \{x \in \mathbb{R} | 1 \leq x \leq 2\}$.
  Väite: $S = T$.

Epätodeksi osoittaminen

Muotoa $\forall x \in X\ P(x)$ olevien väitteiden osoittaminen epätodeksi on huomattavasti helpompaa kuin oikeaksi todistaminen – riittää antaa yksikin vastaesimerkki joukosta X, jolla väite ei päde. Esimerkiksi väite "Kaikki linnut osaavat lentää" on epätosi, koska esimerkiksi pingviinit eivät osaa lentää.

1.7 *Ekskursio: Transfiniittiset ordinaaliluvut

Transfiniittiset ordinaali- eli järjestysluvut tarkoittavat äärettömiä lukuja, joille silti voidaan määritellä järjestys. Mm. David Hilbert, Georg Cantor, Kurt Gödel ja Rudy Rucker ovat tarkastelleet tällaisiin lukuihin liittyviä ongelmia.

ω Omega

ω eli $\aleph_0$ (alef-nolla) on suurin normaali järjestysluku, ts. lukujonon $0, 1, 2, 3, ...$ suurin alkio $\lim n$. ω voidaan määritellä seuraavasti: ω on ensimmäinen sellainen luku a, jolle $a + 1 = a$. Esim. $1 + \omega{=}\omega$, $2\omega = \omega$.

ω:lle voidaan kuitenkin määritellä toimivat yhteen- ja kertolasku uusien transfiniittisten lukujen generoimiseksi. Sovitaan, että $\omega{+}1{=}$ ω:n seuraaja ja $\lim_{n\to\omega}(\omega{+}n) = \omega + \omega = 2\omega$.

Nyt $\lim_{n\to\omega}(\omega \times 2 + n) = \omega \times 3$, $\lim_{n\to\omega}(\omega \times n) = \omega \times \omega = \omega^2$. Samoin saadaan $\omega^3, \omega^4, ..., \omega^\omega$.

ω^2 ja ω^ω voidaan määritellä seuraavasti: ω^2 on ensimmäinen sellainen ordinaaliluku a, jolle $\omega + a = a$ ja ω^ω on ensimmäinen ordinaaliluku a, jolle $\omega \times a = a$. (Tämä voidaan järkeillä seuraavasti: $\omega \times \omega^\omega = \omega^{\omega+1} = \omega^\omega$, koska $1 + \omega = \omega$).

Huom! Transfiniittisten lukujen yhteen- ja kertolasku eivät ole vaihdannaisia! $1{+}\omega = \omega \neq \omega + 1$

ϵ_0 Epsilon-0

Vielä suurempia lukuja saadaan konstruoimalla lukuja sisäkkäisillä potenssiinkorotuksilla ts. $\omega^{\omega^{\omega^{\omega^{...}}}}$. Määritellään ϵ_0 ensimmäiseksi ordinaaliluvuksi a, jolle $\omega^a = a$. ϵ_0 voidaan kuvailla parhaiten ottamalla käyttöön *tetraatio-operaatio* ${}^ab$ ("b teroituna a:han"), joka tarkoittaa b:n a-kertaista potenssia eli eksponetiaalipinoa $b^{b^{b^{...^b}}}$, jossa b toistuu a kertaa. Tetraatio-operaatiolla saadaan nopeasti suuria lukuja, esim. ${}^310 = 10^{10^{\text{ miljardia}}}$ (1 ja perässä 10 miljardia 0:aa), ${}^2\omega = \omega^\omega$, ${}^3\omega = \omega^{\omega^\omega}$. Nyt $\epsilon_0 = {}^\omega\omega$.

Vieläkin suurempia ordinaalilukuja saadaan sisäkkäisillä tetraatio-operaatioilla!

$\aleph_1$ Alef-1

$\aleph_1$ on ensimmäinen ei-ordinaaliluku ts. $\aleph_1$:stä alkiosta koostuvaa joukkoa ei voi mitenkään järjestää numerojärjestykseen. $\aleph_1$:n määritelmä perustuu *mahtavuuden* käsitteeseen. Sanotaan, että joukot A ja B ovat yhtä mahtavia, jos on olemassa bijektio $f : A \to B$ (ts. funktio on määritelty kaikille A:n alkioille ja jokaista B:n alkiota vastaa täsmälleen yksi A:n alkio). Määritellään $\aleph_1$ seuraavasti: $\aleph_1$ on ensimmäinen ordinaaliluku, joka on mahtavampi kuin ω. Ts. ei ole olemassa mitään bijektiota, joka kuvaisi $\aleph_1$ alkiota ω:lle alkiolle.

Hilbertin hotelli

$\aleph_1$:n suuruus voidaan ymmärtää Hilbertin hotellivertauksen avulla. Hilbertin hotellissa on ω huonetta, jotka on numeroitu luonnollisilla luvuilla $0, 1, 2, 3, \ldots$. Hilbertin hotelli on sikäli paradoksaalinen, että täytyttyäänkin siihen mahtuu vielä vieraita, eikä kukaan joudu jakamaan huonettaan! Oletetaan esimerkiksi, että hotellissa on ω vierasta, yksi kussakin huoneessa, ja hotelliin saapuu vielä yksi vieras. Kuinka hänet voidaan majoittaa? Helposti! Vieras sijoitetaan huoneeseen 0, joka saadaan tyhjäksi siirtämällä vieras 0 huoneeseen 1, joka puolestaan saadaan tyhjäksi sijoittamalla vieras 1 huoneeseen 2 jne. Entäpä jos hotelliin saapuu ω:n matkailijan seurue? Nyt voidaan sijoittaa ensimmäiset ω vierasta parillisiin huoneisiin ja uudet ω vierasta parittomiin huoneisiin. Samoin voidaan sopivilla järjestelyillä sijoittaa ω^2, ω^ω tai jopa ϵ_0 vierasta. Tämän uskomattoman hotellin majoituskapasiteetilla on kuitenkin raja – nimittäin $\aleph_1$. $\aleph_1$ on ensimmäinen sellainen luku, jonka suuruista matkustajajoukkoa hotelliin ei saada mahtumaan millään järjestelyllä.

Cantorin argumentti

Cantor osoitti v. 1873, että matemaattisessa avaruudessa on vähintään $\aleph_1$ pistettä. (Yleisemmin tämä tulos ilmaistaan sanomalla, että reaalilukujen joukko on ylinumeroituva). Sen sijaan ns. *jatkumo-ongelma* – onko matemaattisessa avaruudessa kenties enemmän kuin $\aleph_1$ pistettä – on edelleen avoin ongelma. Cantorin todistuksessaan käyttämä tekniikka (ns. *Cantorin diagonaaliargumentti*) on siksi näppärä, että esitämme idean seuraavaksi:

Väite: Reaalilujen joukko $\mathbb{R}$ on ylinumeroituva.

Todistus: Meidän riittää tutkia jotain $\mathbb{R}$:n osajoukkoa ja osoittaa, että se on ylinumeroituva. Valitaan tutkittavaksi osaväli $]0, 1[$ ja tehdään vastaväite:
Vastaväite: Väli $]0, 1[$ on numeroituva. Toisin sanoen voimme numeroida kaikki

reaaliluvut x, $0 < x < 1$ luonnollisilla luvuilla. Olkoon luvun x järjestysnumero $r(x)$ (ts. on bijektio $r : \ ]0,1[\rightarrow \mathbb{N}$). Oletetaan, että voisimme esittää välin $]0,1[$ reaaliluvut äärettömänä matriisina M, jossa kutakin luonnollista lukua vastaa välin reaaliluku esitettynä äärettömällä tarkkuudella. Matriisin alku voisi olla esim. seuraava:

$r(1)$: .**1**41592...
$r(2)$: .3**3**3333...
$r(3)$: .71**8**281...
$r(4)$: .414**2**13...
$r(5)$: .5000**0**0...

Muodostetaan nyt uusi reaaliluku seuraavasti: Luetaan matriisin diagonaalilla olevat digitit järjestyksessä ja vaihdetaan jokainen digitti joksikin toiseksi. Toisin sanoen luvun desimaalikehitelmässä $.d_1d_2d_3d_4...$ i:s desimaali $d_i \neq M[i][i]$ (i:nnen rivin i:s desimaali). Luvun alku voisi siis olla esimerkiksi .02719.... Tämä luku ei kuitenkaan voi esiintyä matriisissa, sillä se poikkeaa jokaisesta matriisin luvusta: 1. luvusta 1. digitin osalta, 2. luvusta 2. digitin osalta, 3.luvusta 3. digitin osalta jne. Kaikki välin $]0,1[$ reaaliluvut luettelevaa funktiota r ei siis voi olla olemassa!

(Huomaa, ettei auta, vaikka lisäisimmekin näin saamamme reaaliluvun matriisiin, sillä voisimme taas generoida samaan tapaan uuden reaaliluvun, joka ei esiinny listassa.)

Kirjallisuutta

Rucker, Rudy: White Light, or, What is Cantor's Continuum Problem? Ace Books, New York, 1982. (Huima scifi-romaani, jossa leikitellään äärettömyyksillä! Vieraillaan myös Hilbertin hotellissa.)

Rucker, Rudy: Mieli ja äärettömyys. Art House, 1998. (Helppolukuinen tietokirja, joka jatkaa White Light:in teemoista.)

Lem, Stanislaw: N. Ya. Vilenkin, Stories about Sets. Academic Press, New York, 1968. (Novellikokoelma, joka sisältää hauskan novellin Hilbertin hotellista.)

Hofstadter, Douglas R.: Gödel, Esser, Bach: An Eternal Golden Braid. Vintage Books, New York, 1989. (Luku XIII esittää helppotajuisesti Cantorin diagonaaliargumentin avulla, että on olemassa ohjelmallisesti ratkeamattomia ongelmia. Muutenkin sopivaa oheislukemistoa kurssille!)

1.8 Tehtäviä lukuun 1

1. Ns. Kyyhkyslakkaperiaate (Pigeonhole Principle) sanoo: Jos kyyhkysiä on enemmän kuin pesäkoloja ja jokainen kyyhkynen lentää johonkin pesäkoloon, niin ainakin yhdessä kolossa on enemmän kuin yksi kyyhkynen.

   Kuinka käy, jos pesäkoloja on yhtä monta kuin luonnollisia lukuja ja kyyhkysiä yhtä monta kuin kokonaislukuja? Entä jos kyyhkysiä on yhtä monta kuin luonnollisia lukuja, mutta jokainen kyyhkynen yrittää pesiä jokaisen toisen kyyhkysen kanssa eri pesissä? (Yhteen koloon mahtuu vain yksi pesä.)

2. Miten sijoittaisit $\omega \times \omega$ vierasta Hilbertin hotellin ω:aan huoneeseen?

3. Hilbertin hotellin vieraskirjan jokaisella sivulla on vain äärellisen monta nimeä ja uusien vieraiden on kirjoitettava nimensä aina seuraavalle tyhjälle riville. Kuinka monta sivua kirjassa on oltava, jotta uusien vieraiden nimille olisi tilaa (järjestämättä nimiä uudelleen), niin kauan kuin vieraita voitaisiin sijoittaa hotelliin (mahdollisesti järjestämällä uudelleen)?

4. Etsi virhe seuraavasta todistuksesta, jonka mukaan 2=1. Tarkastellaan yhtälöä $a = b$. Kerro molemmat puolet a:lla, jolloin saat $a^2 = ab$. Vähennä b^2 molemmilta puolilta, jolloin saat $a^2 - b^2 = ab - b^2$. Jaa kumpikin puoli tekijöihin, $(a - b)(a + b) = b(a - b)$, ja jaa $(a - b)$:llä, jolloin saat $a + b = b$. Lopuksi oletetaan, että a ja b ovat 1, jolloin pätee $2 = 1$.

5. Olkoon X joukko ja X:n koko $n = |X|$. Todista induktiolla, että X:n potenssijoukon koko on $|\mathcal{P}(X)| = 2^n$.

6. Mitä vikaa on seuraavassa induktiotodistuksessa, jonka mukaan kaikki kissat ovat samanvärisiä?

   Olkoon n kissojen lukumäärä. Jos $n = 1$, niin väite selvästi tosi (yksi kissa on aina samanvärinen, olipa väri mikä tahansa). Oletetaan nyt, että mille tahansa n:n kissan joukolle pätee, että kaikki kissat ovat samanvärisiä. Tarkastellaan sitten $n+1$:n kissan joukkoa. Valitsemalla näistä mitkä tahansa n kissaa (jotka voidaan valita $n + 1$ eri tavalla) saadaan induktio-oletuksen perusteella samanväristen kissojen joukko. Siispä kaikki $n + 1$ kissaa ovat samanvärisiä.

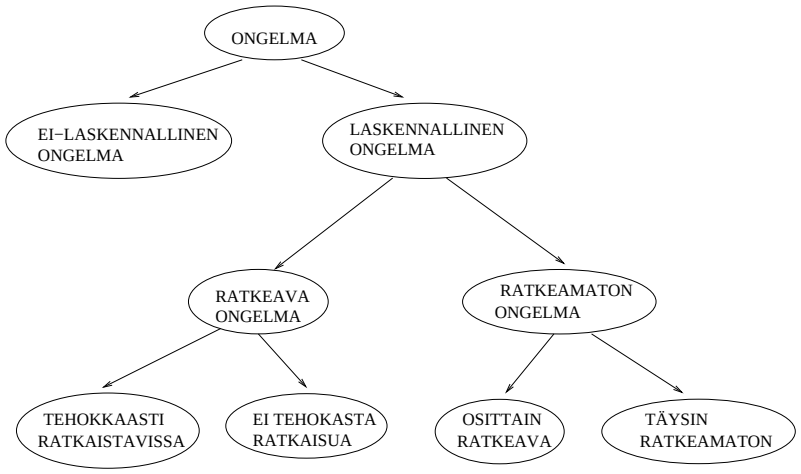
7. Todista seuraava väite: Jos juhlissa on $n (n \geq 2)$ henkilöä, niin vähintään kahdella henkilöllä on yhtä monta ystävää juhlissa.

8. Todista kontrapositiolla: Jos c on pariton kokonaisluku, niin yhtälöllä $n^2 + n - c = 0$ ei ole paritonta kokonaislukuratkaisua n:lle.

Luku 2

Laskennalliset ongelmat

Ongelmat voidaan jakaa yleisellä tasolla kahteen luokkaan: *laskennallisiin* ja *laskennallisiin ongelmiin*. Laskennallisia ongelmia ovat kaikki sellaiset ohgelmat, jotka voidaan mallintaa ratkaistaviksi digitaalisella tietokoneella. Huomaa, että laskennalisen ongelman ei silti tarvitse olla ratkeava, ts. voi olla, ettei periaatteessakaan ole mahdollista laatia tietokoneohjelmaa, joka ratkaisisi ongelman. Esimerkkejä laskennallisista ongelmista ovat vaikkapa kokonaislukujen kertolasku, kirjastokortiston aakkostaminen, yrityksen palkanlaskenta, yliopistollisen kurssin kurssitietojen ylläpito. Sen sijaan sellaiset ongelmat kuin "Kuinka opettaa sodat?", "Onko oikein huijata tentissä?" ovat luonteeltaan ei-laskennallisia. Jatkossa viittaamme sanalla "ongelma" aina laskennalliseen ongelmaan.



Kuva 2.1: Ongelmien luokittelu.

Laskennallisen ngelman ratkaiseva ohjelma on sen yksi esitystapa, mutta kaikille ongelmille emme voi tai osaa laatia ratkaisevaa ohjelmaa. Yleisempi ja abstraktimpi esitystapa helpottaa myös ongelman analysointia, ja voimme analysoida sen ratkeavuuden ja vaikeuden, ennen kuin kukaan on keksinyt ongelman ratkaisevaa ohjelmaa. Tästä syystä laskennanteoriassa ongelmia mallinnetaan formaaleilla malleilla. Formaalien mallien ei kuitenkaan tarvitse olla tylsiä, kuten seuraava esimerkki osoittaa.

Luvun loppupuolella tarkastelemme laskennallisten ongelmien formalisointia yleensä sekä erästä tärkeää ongelmien alaluokkaa, *päätösongelmia*, ja niitä vastaavia *formaaleja kieliä*. Lopuksi annamme esimerkin ratkeamattomasta ongelmasta.

2.1 Ongelma: MIU-järjestelmä

Kissastanian logiikakoulussa on tunnin aiheena MIU-järjestelmä, jonka kaikki lauseet koostuvat kolmesta kirjaimesta M, I ja U. Järjestelmässä on vain yksi aksiooma MI. Uusia lauseita (teoreemoja) saa muodostaa edellisistä lauseista (aksioomiasta tai teoreemoista) seuraavilla säännöillä:

1. Jos merkkijonon viimeinen kirjain on I, saat lisätä U:n loppuun.
2. Jos sinulla on Mx (missä x voi olla mikä tahansa merkkijono, myös tyhjä merkkijono), saat päätellä merkkijonon Mxx.
3. Jos jossain merkkijonossa esiintyy III, saat korvata sen U:lla.
4. Jos merkkijonossa esiintyy UU, sen saa pudottaa pois merkkijonosta.

Sääntöjä saa soveltaa vapaasti aina, kun ne sopivat aksioomaan tai jo johdettuihin teoreemoihin, mutta mitään muuta ei saa tehdä (siksi järjestelmää kutsutaan formaaliksi).

Kissakoululaisten tehtävänä on johtaa aksioomasta MI teoreema MUI. Osaatko suorittaa päättelyn? Entä osaatko johtaa aksioomasta MI teoreeman MU?!

MIU-järjestelmää voidaan siis ajatella eräänlaisena kissojen kielipelinä. Käytössä on *aakkosto* $\Sigma = \{M, I, U\}$ ja kielioppi, jonka mukaan voidaan muodostaa sanoja. Esitetään sananmuodostussäännöt hieman formaalimmin:

$xI \rightarrow xU$
$Mx \rightarrow Mxx$
$xIIIy \rightarrow xUy$
$xUUy \rightarrow xy$,

missä x ja y voivat olla mitätahansa merkkijonoja (myös tyhjä merkkijono).

Tarkastellaan aakkoston kaikkien mahdollisten merkkijonojen joukkoa Σ^*, joka koostuu merkkijonoista $\{\epsilon, M, I, U, MM, MI, MU, IM, II, IU, UM, UI, UU, MMM, MMI, MMU, MIM, ...\}$ (ϵ = tyhjä merkkijono). Nyt voidaan esittää erilaisia kysymyksiä aakkoston merkkijonoista. Esim. ongelma $\pi_1(x)$: "Mitä merkkijonoja voit tuottaa annetusta merkkijonosta x järjestelmän säännöillä?" tai $\pi_2(x)$: "Voitko tuottaa merkkijonon x MI:stä järjestelmän säännöillä?" Ensimmäisen kysymyksen vastaus on joukko merkkijonoja, itse asiassa Σ^*:n osajoukko (tai mahdollisesti tyhjä joukko), kun taas jälkimmäisen kysymyksen vastaus on "Kyllä" tai "Ei". Tällaisia kyllä/ei-ongelmia kutsutaan *päätösongelmiksi*. Formaalisti voidaan määritellä päätösongelma π kuvauksena $\pi : \Sigma^* \rightarrow \{0, 1\}$. Se liittää siis jokaiseen aakkoston merkkijonoon joko vastauksen 1 tai 0.

Toisaalta voimme kysyä, mitä kaikkia Σ^*:n merkkijonoja päätösongelma π_A hyväksyy? (T.s. mille x $\pi(x) = 1$ tai käänteisrelaatio $\pi^{-1}(1) = x$?) Vastausjoukkoa A kutsutaan *formaaliksi kieleksi* ja vastaavaa päätösongelmaa π_A kielen A tunnistusongelmaksi.

Tehtävä: Mistä sanoista seuraavat kielet koostuvat?

- Kaikki M:stä johdettavat merkkijonot.
- Kaikki U:sta johdettavat merkkijonot.
- Kaikki MU:sta johdettavat merkkijonot.

(MIU-järjestelmän idea on alkujaan Hofstadterin hauskasta kirjasta Gödel, Escher, Bach.)

2.2 Formalisointi

- ongelmalla potentiaalisesti ääretön joukko *tapauksia* ("syötteitä")

- ongelman ratkaisu on *algoritmi*, joka liittää kuhunkin tapaukseen sen oikean *vastauksen* ("tulosteen").
- esim. kokonaislukujen kertolaskuongelma
  - tapaukset: kaikki mahdolliset kokonaislukuparit
  - annettuun tapaukseen liittyvä vastaus: lukuparin tulo
  - ongelman ratkaisu: mikä tahansa yleinen kertolaskualgoritmi.
- kunkin yksittäisen tapauksen ja sen vastauksen oltava *äärellisesti esitettäviä*.
- Laskennallinen ongelma = *kuvaus äärellisesti esitettävien tapausten joukosta äärellisesti esitettävien vastausten joukkoon*

Äärellinen esitys

- kaikki tietokoneen käsittelemä tieto on viime kädessä voitava koodata bittijonoiksi.
- luontevaa sallia koodaukseen käytettävän myös muita merkkejä kuin 0 ja 1 (muut merkit voidaan tietenkin tarvittaessa edelleen esittää bittijonoina).
- Määr. "äärellinen esitys" = jonkin aakkoston *merkkijono*.
- esim. em. kertolaskun syöte (kaksi kokonaislukua) voitaisiin esittää merkkijonona $x\#y$ tai $mul(x, y)$, missä x ja y ovat kokonaislukujen merkkijonoesitykset

Merkkijonoihin liittyviä peruskäsitteitä ja merkintöjä

- *Aakkosto* on äärellinen, epätyhjä joukko alkeis*merkkejä* t. *symboleita*. esim. *binääriaakkosto* $\{0, 1\}$ ja *latinalainen aakkosto* {A,B,...,Z}.
- *Merkkijono* on äärellinen järjestetty jono jonkin aakkoston merkkejä. Esim. "01001" ja "000" ovat binääriaakkoston merkkijonoja, ja "LTE" ja "XYZZY" ovat latinalaisen aakkoston merkkijonoja.
- Merkkijonon x *pituus* on siihen sisältyvien merkkien määrä. Merk. $|x|$. Esim.

$$|01001| = |\text{XYZZY}| = 5, \qquad |000| = |\text{OTE}| = 3.$$

- *Tyhjä merkkijonon* ϵ pituus on $|\epsilon| = 0$.

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

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

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

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

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

- Aakkoston Σ kaikkien merkkijonojen joukko on Σ^*. Esim. $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, \ldots\}$.

Päätösongelmat ja formaalit kielet

- yleisesti laskennallinen ongelma π on kuvaus

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

  missä Σ ja Γ ovat aakkostoja

- päätösongelmat ovat tärkeä laskennallisten ongelmien aliluokka, jossa kunkin ongelman tapauksen vastaus on "kyllä" tai "ei"

- formaalisti ongelma on muotoa $\pi : \Sigma^* \to \{0, 1\}$.

- esimerkiksi päätösongelma "onko annettu luku alkuluku?" voidaan esittää aakkoston $\Sigma = \{0, 1, 2, \ldots, 9\}$ kuvauksena

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

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

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

  so. niiden ongelman tapausten joukko, joihin vastaus on "kyllä"

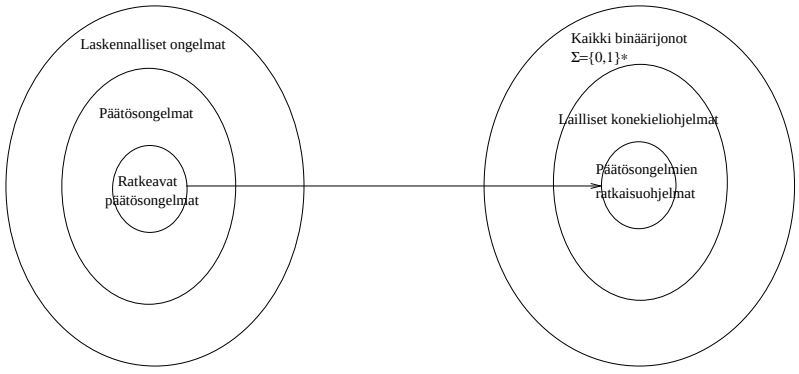
- jokaista merkkijonojoukkoa $A \subseteq \Sigma^*$ vastaa päätösongelma

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

- aakkoston Σ *(formaali) kieli* (engl. formal language) = mielivaltainen merkkijonojoukko $A \subseteq \Sigma^*$
- kielen A *tunnistusongelma* (engl. recognition problem) = merkkijonojoukkoon liittyvä päätösongelmaa π_A
- ts. formaali kieli $\sim$ päätösongelma

2.3 Laskennallisten ongelmien ratkeavuus

- sanotaan, että ohjelma P *ratkaisee* laskentaongelman π, jos kullakin syötteellä x ohjelma P laskee ja tulostaa arvon $\pi(x)$.
- Voidaanko kaikki mahdolliset laskentaongelmat ratkaista ohjelmilla?
- Osoittautuu, ettei voida, sillä kaikkien merkkijonojen (millä tahansa ohjelmointikielellä mahdollisten ohjelmien) joukko on numeroituva, mutta kaikkien päätösongelmien joukko on ylinumeroituva ($\aleph_1$ vierasta ei voi majoittaa ω:aan huoneeseen).
- Laskentaongelmia on siis tietyssä mielessä "enemmän" kuin niiden mahdollisia ratkaisuja, ja siksi *millään ohjelmointikielellä ei voida laatia ratkaisuja kaikille laskentaongelmille.*



Kuva 2.2: Vain pieni osa päätösongelmista on tietokoneella ratkeavia.

Lause 1 Minkä tahansa aakkoston Σ merkkijonojen joukko Σ^* on numeroituvasti ääretön.

Todistus: Olkoon $\Sigma = \{a_1, a_2, ..., a_n\}$. Kiinnitetään merkeille "aakkosjärjestys", esim. $a_1 < a_2 < ... < a_n$.

Joukon Σ^* merkkijonot voidaan järjestää seuraavasti (*kanoniseen järjestykseen*):

1. ensin luetellaan 0:n mittaiset merkkijonot ($= \epsilon$), sitten 1:n mittaiset ($= a_1, a_2, \ldots, a_n$), sitten 2:n mittaiset jne.;
2. kunkin pituusryhmän sisällä merkkijonot luetellaan aakkosjärjestyksessä.

- jokaiseen luonnolliseen lukuun n voidaan siis liittää Σ^*:n merkkijono ja päinvastoin $\rightarrow \Sigma^*$ on numeroituva.

Vaadittu bijektio $f : \mathbb{N} \rightarrow \Sigma^*$ on:

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

□

Lause 2 Minkä tahansa aakkoston Σ päätösongelmien joukko on ylinumeroituva.

**Todistus:* (Cantorin diagonaaliargumentti.)

Merkitään kaikkien Σ:n päätösongelmien kokoelmaa Π:llä:

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

Vastaväite: Oletetaan, että Π on numeroituva, so. on olemassa numerointi

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

Olkoot Σ^*:n merkkijonot kanonisessa järjestyksessä lueteltuina $x_0, x_1, x_2, \ldots$.

Muodostetaan uusi päätösongelma $\hat{\pi}$:

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

Koska $\hat{\pi} \in \Pi$,niin $\hat{\pi} = \pi_k$ jollakin $k \in \mathbb{N}$.
Tällöin

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

RISTIRIITA. Siis oletus, että joukko Π on numeroituva, on väärä. □

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

- Käy kuin Valehtelijan paradoksissa! (”Puhuuko x totta, jos x sanoo: ’Minä valehtelen’?”)

- kaikista laskentaongelmista voidaan esimerkiksi C-ohjelmilla ratkaista vain häviävän pieni osa: ylinumeroituvan joukon numeroituva osajoukko.
- sama pätee kaikilla ohjelmointikielillä, sillä kaikki "riittävän vahvat" ohjelmointikielet määrittävät täsmälleen saman ratkeavien ongelmien luokan (ns. Churchin–Turingin teesi).
- useimmat laskennalliset ongelmat ovat *absoluuttisesti ratkeamattomia*.
- ratkeamattomat ongelmat käsittävät myös mielenkiintoisia/käytännöllisiä ongelmia
- esim. *pysähtymisongelma*: annettu ohjelma P ja sen syöte x; pääteltävä pysähtyykö P:n laskenta syötteellä x, vai jääkö se ikuiseen silmukkaan.

2.4 Ekskursio: pysähtymisongelman ratkeamattomuus

Pysähtymisongelman C-tulkinta on: "Ei ole olemassa totaalista (aina pysähtyvää) C-ohjelmaa, joka ratkaisisi, pysähtyykö annettu C-ohjelma P annetulla syötteellä w".

Oletetaan, että voitaisiin kirjoittaa totaalinen C-funktio

int H(char *p, char *w),

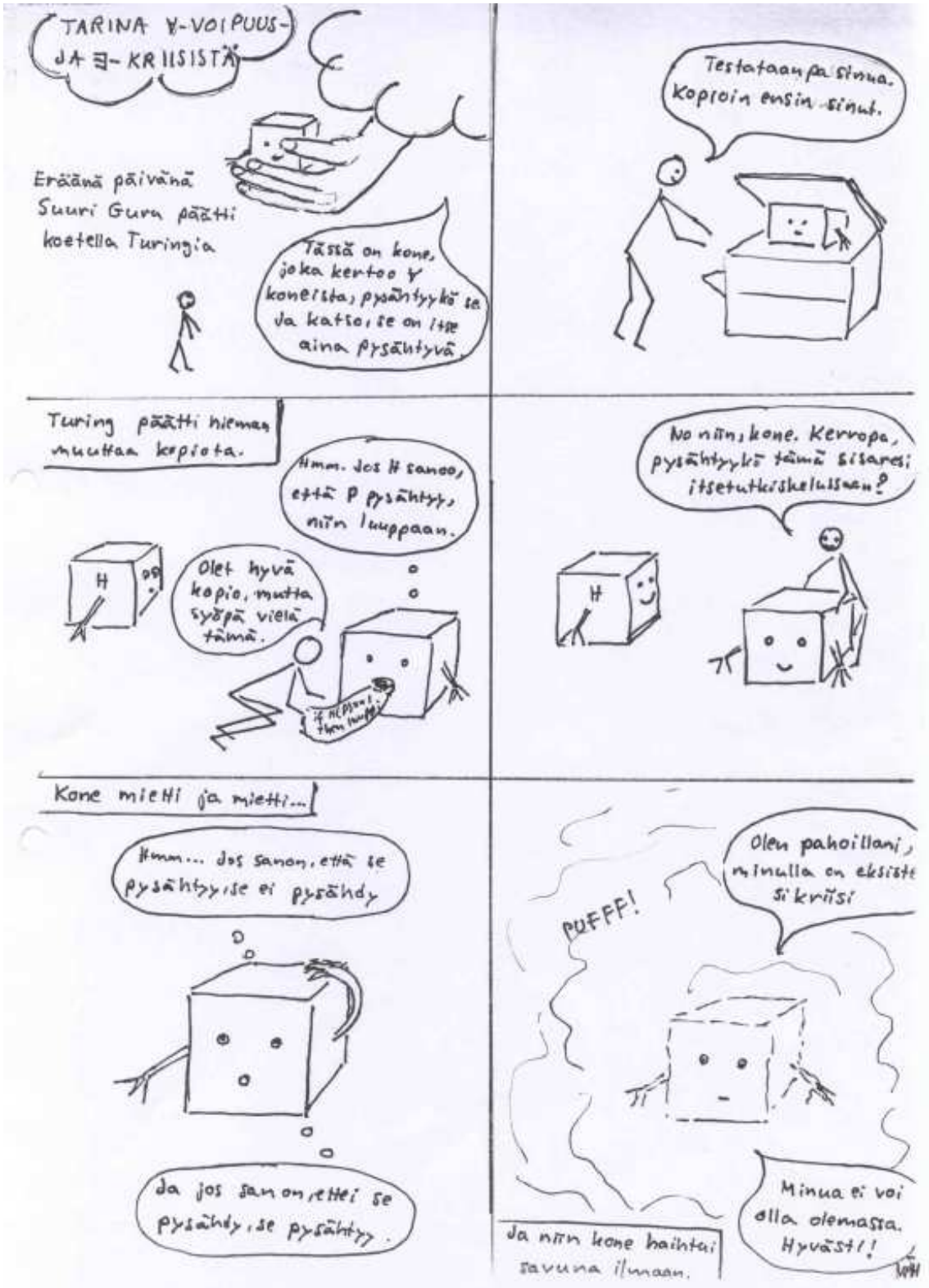
joka saa arvon **1**, jos merkkijonoparametrin p esittämä funktio pysähtyy syötteellä w, ja **0** muuten. Kirjoitetaan tämän perusteella toinen C-funktio $\hat{H}$:

void $\hat{H}$(char *p){
    if $H(p,p)$ while (1) ;
}

Merkitään edellä kuvattua funktion $\hat{H}$ ohjelmatekstiä $\hat{h}$:lla ja tarkastellaan funktion $\hat{H}$ laskentaa tällä omalla kuvauksellaaan. Saadaan ristiriita:

$$\hat{H}(\hat{h}) \text{ pysähtyy} \quad \Leftrightarrow \quad H(\hat{h},\hat{h}) = \mathbf{0} \quad \Leftrightarrow \quad \hat{H}(\hat{h}) \text{ ei pysähdy.}$$

Ristiriidasta seuraa, että oletettua totaalista pysähtymisentestausohjelmaa H ei voi olla olemassa.


TARINA ∀-VOIPUUS-
JA ∃-KRIISISTÄ
Eräänä päivänä
Suuri Guru päätti
koetella Turingia
Tässä on kone,
joka kertoo ∀
koneista, pysähtyykö se
ja katso, se on itse
aina pysähtyvä.
Testataan pa sinua.
Kopioin ensin sinut.
Turing päätti hieman
muuttaa kopiota.
Hmm. Jos H sanoo,
että P pysähtyy,
niin luuppaan.
H
Olet hyvä
kopio, mutta
syötä vielä
tämä.
No niin, kone. Kerropa,
pysähtyykö tämä sisäresi
itsetutkiskelussaan?
H
Kone miehi ja mietti...
Hmm... Jos sanon, että se
pysähtyy, se ei pysähdy
Ja jos sanon, ettei se
pysähdy, se pysähtyy.
PUFFF!
Olen pahoillani,
minulla on eksistentiaalinen
kriisi
Minua ei voi
olla olemassa.
Hyvästi!
Ja niin kone haihtui
savuna ilmaan.

2.5 Liite: Vakiintuneita merkintätapoja

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

2.6 Tehtäviä lukuun 1

1. Lue satu päätösongelmista
   `http://www.cs.joensuu.fi/pages/whamalai/tepe04/satu.htm` ja täydennä se loppuun!

2. Tarkastellaan taas Kissastanian logiikkakoulua. Tällä kertaa tunnin aiheena on hieman monimutkaisempi MIAU-järjestelmä, joka koostuu seuraavista säännöistä:

   xUAx → xAUy
   xUUx → xIUy
   x → MxM
   x → xUI
   xx → x
   xI → xUA,

   missä x ja y voivat olla mitätahansa merkkijonoja.

   Tehtävänä on osoittaa, että tyhjästäkin (merkkijonosta) voi syntyä kunnon naukaisu (MIAU) järjestelmän säännöillä!

3. Tarkastellaan aakkostoa $\Sigma = \{m, i, u\}$. Määritellään aakkoston "potenssit" seuraavasti:

   $\Sigma^0 = \{\epsilon\}$ (tyhjä merkkijono)
   $\Sigma^{k+1} = \Sigma \times \Sigma^k = \{ax | a \in \Sigma \text{ ja } x \in \Sigma^k\}$.

   Esim. $\Sigma^1 = \{m, i, u\}$, $\Sigma^2 = \{mm, mi, mu, im, ii, iu, um, ui, uu\}$. Montako alkiota ("sanaa") on Σ^n:ssä? Entä montako sanaa on koko kielessä $\Sigma^* = \bigcup_{i=0}^{\infty}$?

Luku 3

Säännölliset kielet ja äärelliset automaatit

Tässä luvussa tutustumme yksinkertaisimpaan formaalien kielten luokkaan, säännöllisiin kieliin. Säännölliset kielet voidaan kuvata säännöllisten lausekkeiden avulla ja niitä vastaavat (päätös-)ongelmat voidaan ratkaista äärellisillä automaateilla.

Ongelma: Ohjelmointikielten muuttujat, vakiot jne. ovat tietynmuotoisia, ohjelmointikielen *syntaksin* ("oikeankirjoitusopin") mukaisia. Esim. 0.123 on luku, mutta 0.1.2.3 tai $z.33$ eivät ole. Mutta miten määritellä lukujen syntaksi? Entä miten voidaan tunnistaa täsmälleen oikeanmuotoiset merkkijonot? Osoittautuu, että oikeanmuotoiset merkkijonot voidaan kuvata säännöllisillä lausekkeilla, ja niiden tunnistus voidaan suorittaa äärellisellä automaatilla.

Tunnet ehkä UNIX:in käskyn `grep`, jolla voidaan etsiä tekstistä hahmoja. Esim.

`grep [A-Z][a-z]*[0-9] teksti` etsii tiedostosta `teksti` rivit, joilla on määrittelyn muotoisia sanoja: ensin suuri kirjain, sitten mielivaltainen määrä pieniä kirjaimia ja lopuksi numero. `grep` etsii nimenomaan säännöllisiä lausekkeita, ja nimi onkin lyhenne sanoista "Global search for Regular Expression and Print"

3.1 Säännölliset lausekkeet ja kielet

- Esim. Hyväksy merkkijonot, joissa esiintyy sana "kissa". Ts. merkkijonot ovat muotoa

  [0 tai useampia kirjaimia]*kissa*[0 tai useampia kirjaimia]

- Esim. Etsi tekstitiedostosta osoitteita, jotka ovat muotoa
  [*x*katu tai *x*tie][*numero*] [mahd. rapun kirjain]
  [mahd. asunnonnumero][postinumero][kunnan nimi]

- Kuinka esittää kompaktisti sallitut merkkijonot (ts. kuvata *kieli*, jonka tunnistusohjelma hyväksyy)?

- Määritellään avuksi kolme operaatiota kielten yhdistämiseen: Olkoot A ja B aakkoston Σ kieliä. Tällöin

  - A:n ja B:n *yhdiste* on kieli

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

  - A:n ja B:n *tulo* on kieli

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

  - A:n *potenssit* A^k, $k \geq 0$, määritellään iteratiivisesti:

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

  - A:n *sulkeuma* on kieli

    $$\begin{aligned} A^* &= \bigcup_{k=0}^{\infty} A^k \\ &= \{x_1 \ldots x_k \mid k \geq 0,\ x_i \in A \quad \forall i = 1, \ldots, k\} \end{aligned}$$

- *Määritelmä:* Aakkoston Σ *säännölliset lausekkeet* (engl. regular expression) määritellään induktiivisesti säännöillä:
  (i) $\boldsymbol{\emptyset}$ ja $\boldsymbol{\epsilon}$ ovat Σ:n säännöllisiä lausekkeita;
  (ii) $\boldsymbol{a}$ on Σ:n säännöllinen lauseke kaikilla $a \in \Sigma$;
  (iii) jos r ja s ovat Σ:n säännöllisiä lausekkeita,
  niin $(r \cup s)$, (rs) ja r^* ovat Σ:n säännöllisiä lausekkeita;
  (iv) muita Σ:n säännöllisiä lausekkeita ei ole

- Kukin Σ:n säännöllinen lauseke r *kuvaa* kielen $L(r)$:
  (i) $L(\boldsymbol{\emptyset}) = \emptyset$;
  (ii) $L(\boldsymbol{\epsilon}) = \{\epsilon\}$;
  (iii) $L(\boldsymbol{a}) = \{a\}$ kaikilla $a \in \Sigma$;
  (iv) $L((r \cup s)) = L(r) \cup L(s)$;
  (v) $L((rs)) = L(r)L(s)$;
  (vi) $L(r^*) = (L(r))^*$

- Aakkoston $\{a, b\}$ säännöllisiä lausekkeita:

$$r_1 = \boldsymbol{((ab)b)}, \quad r_2 = \boldsymbol{(ab)^*},$$

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

  Lausekkeiden kuvaamat kielet:

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

- Sulkumerkkien vähentämissääntöjä:
  - Operaattoreiden prioriteetti:

$$\boldsymbol{*} \quad \succ \quad \cdot \quad \succ \quad \cup$$

– Yhdiste- ja tulo-operaatioiden assosiatiivisuus:

$$\begin{aligned} L(((r \cup s) \cup t)) &= L((r \cup (s \cup t))) \\ L(((rs)t)) &= L((r(st))) \end{aligned}$$

$\Rightarrow$ peräkkäisiä yhdisteitä ja tuloja ei tarvitse suluttaa

– Käytetään tavallisia kirjasimia, mikäli sekaannuksen vaaraa merkkijonoihin ei ole.
Yksinkertaisemmin siis:

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

- *Määritelmä:* Kieli on *säännöllinen*, jos se voidaan kuvata säännöllisellä lausekkeella

- Esim. Olkoon aakkosto $\Sigma = \{a, b, c, ...,\}$. Hyväksytään merkkijonot, jotka ovat muotoa

$$l^* \text{kissa} l^*,$$

missä l on lyhenne lausekkeelle $l = (\text{a} \cup \text{b} \cup ... \cup \text{ö})$ (ts. $l^* \in \Sigma^*$)

- Esim. 2: Olkoon $\Sigma = \{A, B, ..., , a, b, ..., , 0, 1, 2, ..., 9\}$. Osoite on muotoa

$$(Ll^*)(\text{katu } \cup \text{tie })dd^*(l \cup \epsilon)(dd^* \cup \epsilon)\ ddddd\ Ll^*,$$

missä d on lyhenne lausekkeelle

$$d = (0 \cup 1 \cup ... \cup 9)$$

ja L on lyhenne lausekkeelle

$$l = (\text{A} \cup \text{B} \cup ... \cup \text{Ö}).$$

- Esim. 3 C-kielen etumerkittömät liukuluvut (float, double, long double) määritellään seuraavasti:

– (kokonaisosa).(desimaaliosa) (e tai E) [+ tai −] (eksponetti) [suffiksi]

– kokonaisosa ja desimaaliosa koostuvat digiteistä

– joko kokonaisosa tai desimaaliosa voi puuttua (mutta eivät molemmat)

  - joko (i) desimaalipiste tai (ii) (e tai E) ja eksponentti voivat puuttua (mutta eivät molemmat)
  - suffiksi: F tai f: float, L tai l: long double, muuten double

- Etumerkittömät liukuluvut tunnistava kieli voidaan määritellä säännöllisellä lausekkeella (ilman suffikseja):
  number $= (d^+.d^* \cup .d^+)(\epsilon \cup ((e \cup E)(+ \cup - \cup \epsilon)d^+)) \cup$
  $d^+(e \cup E)(+ \cup - \cup \epsilon)d^+$

  Vai? number $= d^*(.dd^*) \cup$
  $d^*(.dd^* \cup \epsilon)(\mathrm{E} \cup \mathrm{e})(+ \cup - \cup \epsilon)dd^* \cup$
  $dd^*(\mathrm{E} \cup \mathrm{e})(+ \cup - \cup \epsilon)dd^*$

- Kieleen kuuluvat esim. seuraavat merkkijonot: 12., .12, 1.2, 1.2E3, 1.2e3, 1.2E-3,1E2, 1e23

3.1.1 Säännöllisten lausekkeiden sieventäminen

- Säännöllisillä kielillä on yleensä useita vaihtoehtoisia kuvauksia, esim.:

$$\begin{aligned} \Sigma^* &= L((a \cup b)^*) \\ &= L((a^*b^*)^*) \\ &= L(a^*b^* \cup (a \cup b)^*ba(a \cup b)^*). \end{aligned}$$

- *Määritelmä:* Säännölliset lausekkeet r ja s ovat *ekvivalentit*, merk. $r = s$, jos $L(r) = L(s)$

- Lisäksi merkitään $r \subseteq s$ tarkoittamaan $L(r) \subseteq L(s)$

- Lausekkeen sieventäminen = "yksinkertaisimman" ekvivalentin lausekkeen määrittäminen

- Huom: Usein merkitään $r^+ = rr^* = r^*r$

3.1.2 Sievennyssääntöjä

$$
\begin{aligned}
r \cup r &= r \text{ (mutta } rr \neq r, \text{ kun } r \neq \emptyset, \epsilon) \\
r \cup (s \cup t) &= (r \cup s) \cup t \\
r(st) &= (rs)t \\
r \cup s &= s \cup r \\
r(s \cup t) &= rs \cup rt \\
(r \cup s)t &= rt \cup st \\
\emptyset^* &= \epsilon \\
\emptyset r &= \emptyset \text{ (mutta } \emptyset \cup r = r) \\
\epsilon r &= r \text{ (mutta } \epsilon \cup r \neq r, \text{ kun } r \neq \epsilon) \\
r^* &= r^* r \cup \epsilon = r^+ \cup \epsilon \\
r^* &= (r \cup \epsilon)^* \\
(r^*)^* &= r^*
\end{aligned}
$$

- Lisäksi pätee päättelysääntö:

  Jos $r = rs \cup t$, niin $r = ts^*$, kun $\epsilon \notin L(s)$

- $L(r) = L(s) \Leftrightarrow L(r) \subseteq L(s) \wedge L(s) \subseteq L(r)$ eli $r = s \Leftrightarrow r \subseteq s \wedge s \subseteq r$

- Esim. yllä:

  1. $(a \cup b) \subseteq (a^*b^*) \Rightarrow (a \cup b)^* \subseteq (a^*b^*)^*$
  2. $((a^*b^*)^*) \subseteq a^*b^* \cup (a \cup b)^*ba(a \cup b)^*$:
     - jos muotoa a^*b^*, niin selvä
     - muuten sisältää osajonon ba
  3. $a^*b^* \cup (a \cup b)^*ba(a \cup b)^*) \subseteq (a \cup b)^*$, sillä $(a \cup b)^*$ kuvaa kaikki Σ:n merkkijonot

- Voidaan todistaa seuraavat sulkeumaominaisuudet: Jos L ja M ovat säännöllisiä kieliä, myös

  1. $L \cap M$ (kielten leikkaus)
  2. $\overline{L} = \Sigma^* \setminus L$ (kielen komplementti)
  3. $L^R = \{w^R | w \in L\}$ (kielen käänteiskieli, jossa sanat on kirjoitettu takaperin)

  ovat säännöllisiä

3.1.3 *Tavalliset joukko-operaatiot vs. säännöllisten kielten operaatiot

Tavallisten joukkojen joukko-operaatiot ja säännöllisten kielten operaatiot muistuttavat toisiaan. Huomaa, että kielikin on joukko – nimittäin joukko merkkijonoja eli sanoja.

Olkoon $A = \{a, b\}$ ja $B = \{c, d\}$.

Joukko-operaatio	Säännöllisen kielen operaatio
Joukkojen A ja B yhdiste $A \cup B = \{a, b, c, d\}$	Kielten A ja B yhdiste $A \cup B = \{a, b, c, d\}$
Joukkojen A ja B karteesinen tulo $A \times B = \{(a, c), (a, d), (b, c), (b, d)\}$	Kielten A ja B tulo $AB = \{ac, ad, bc, bd\}$
Joukon potenssijoukko $\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$	Kielen A sulkeuma $A^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa,$ $bab, bba, bbb, aaaa, aaab, aaba, aabb, ...\}$
Joukon X, $\|X\| = n$ potenssijoukon koko $\|\mathcal{P}(X)\| = 2^n$	Kielen X, $\|X\| = n$, sulkeuman koko, missä n on sanojen lukumäärä kielessä X $\|X^*\| = \infty$

Harjoituksia säännöllisistä lausekkeista

1. UNIX:in egrep-komennolla (extended grep) voi etsiä tekstistä hahmoja, jotka on määritelty säännöllisinä lausekkeina. egrepin perussyntaksi on seuraava: `egrep <lauseke> <tiedosto>`, missä lauseke voi olla

   - hakasuluissa lista merkkejä, esim. $[abcd]$: mikä tahansa merkeistä a, b, c, d
   - $(lauseke)(lauseke)$: kahden lausekkeen katenaatio
   - $(lauseke1)|(lauseke2)$: joko lauseke1 tai lauseke2
   - $(lauseke)*$: lauseke toistuu 0 kertaa tai useampia kertoja (sulkeuma)
   - $\backslash b$: tyhjä merkki sanan reunassa $\backslash B$: tyhjä merkki sanan keskellä

   Huom! Lauseke kannattaa laittaa hipsuihin ('lauseke'). Lisää tietoa komennolla man egrep.

   Testaa egrep-komentoa! Voit käyttää syötetiedostona mitätahansa (C-)ohjelmakoodia, esim. kotisivun tiedostoa esim.c.

   Millaisia hahmoja etsivät seuraavat komennot:
   egrep '[1]*'
   egrep '[1][0]'
   egrep '[1]|[0]'

2. Millaisella egrep-komennolla löydät seuraavat rivit:
   a) Rivit, joilla on numeroita
   b) Rivit, joilla esiintyy sana *while* tai *for*
   c) Rivit, joilla esiintyy numero 10
   d) Rivit, joilla esiintyy kokonaislukuja. (Huom! Komentosi ei siis saa hyväksyä desimaalilukuja.)

3. Tarkastellaan seuraavia aakkoston $\Sigma = \{a, b\}$ kieliä. Anna kustakin kielestä kaksi merkkijonoa, jotka kuuluvat kieleen, ja kaksi, jotka eivät kuulu kieleen!
   a) a^*b^*
   b) $a(ba)^*b$
   c) $a^* \cup b^*$
   d) $(aaa)^*$
   e) $(\epsilon \cup a)b$
   f) $\Sigma^*a\Sigma^*a\Sigma^*a\Sigma^*$

4. Mitä merkkijonoja kuuluu seuraavaan lausekkeen kuvaamaan kieleen?
   $(c \cup h \cup m \cup r)at((c \cup t)a \cup (s \cup t)o)ught(m \cup l \cup tw \cup r)ice$

5. Kuinka määrittelet säännöllisen lausekkeen, jolla löydät tietoa lumimyrskystä? Huomaa, että ilmaisu voi esiintyä myös taipuneena (sanavartalo ei onneksi taivu) tai yhdistettynä, esim. "lumi- ja ukkosmyrskyvaroitus".

6. Mitä merkkijonoja kuuluu kieleen $L(\emptyset^*)$? Entä $L(\epsilon^*)$?

7. Etsi lyhin merkkijono, joka kuuluu seuraavan lausekkeen kuvaamaan kieleen!

   a) $a^*(b \cup abb)b^*b$
   b) $a^*b^*b(a \cup (ab)^*)^*b^*$
   c) $(a \cup ab)(a^* \cup ab)^*b$

8. Muodosta seuraavia kieliä vastaavat säännölliset lausekkeet:
   a) $\{w \in \{a,b\}^* | w$:n kolmanneksi viimeinen merkki on $a\}$
   b) $\{w \in \{a,b\}^* | w$ sisältää joko merkkijonon ab tai $ba\}$
   c) $\{w \in \{a,b\}^* | w$ ei merkkijonoa $bab\}$

9. Muodosta seuraavia kieliä vastaavat säännölliset lausekkeet:
   a) $\{w \in \{a,b\}^* | w$ sisältää parillisen määrän merkkiä $a\}$
   b) $\{w \in \{a,b\}^* | w$:n pituus on pariton$\}$
   c) $\{w \in \{a,b\}^* | w$:n sisältämien b-merkkien lukumäärä on kolmella jaollinen$\}$

10. Esitä yksikertaisemmassa muodossa seuraavat lausekkeet! (s.e. ne yhä generoivat saman kielen!)
    a) $(0 \cup 1 \cup 01 \cup 11)^*$
    b) $(0^* \cup 10^*)^*$
    c) $1^*(011^*)^* \cup 1^*(011^*)^*0$

3.2 Äärelliset automaatit

- Ongelma: Kahviautomaatti, joka ei anna vaihtorahaa, hyväksyy vain 50 sentin ja yhden euron kolikoita ja minimimaksu on 2 euroa. Millaisia syötejonoja kahviautomaatti hyväksyy?

- Kelvollisia syötejonoja ovat esim. seuraavat (yksikkönä snt):
  50 + 50 + 50 + 50
  100 + 100
  50 + 100 +100
  100 + 50 + 50 + 100

- Ts. kahviautomaatti hyväksyy syötejonot, jotka ovat muotoa

  ```
  1 euro +  1 euro +
  [0 tai useampia 50 sentin tai 1 euron kolikoita]
  ```

  tai

  ```
  1 euro +  50 senttiä +
  [1 tai useampia 50 sentin tai 1 euron kolikoita]
  ```

  tai

  ```
  50 senttiä + 1 euro +
  [1 tai useampia 50 sentin tai 1 euron kolikoita]
  ```

  tai

  ```
  50 senttiä +  50 senttiä + 1 euro +
  [0 tai useampia 50 sentin tai 1 euron kolikoita]
  ```

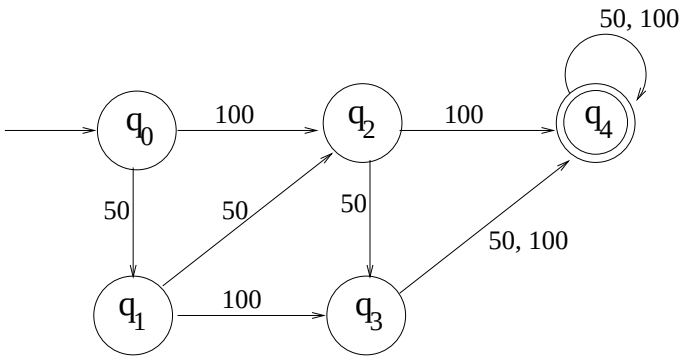
  tai

  ```
  50 senttiä + 50 senttiä + 50 senttiä +
  [1 tai useampia 50 sentin tai 1 euron kolikoita]
  ```

- Kahviautomaatin toiminta voidaan kuvata *äärellisenä automaattina*

- Automaatin syötteitä ovat 50 sentin ja 1 euron kolikot ja automaatti hyväksyy ”syötejonon”, jos siihen sisältyvien rahojen summa on vähintään 2 euroa

- Automaatti voidaan esittää *tilasiirtymäkaaviona*



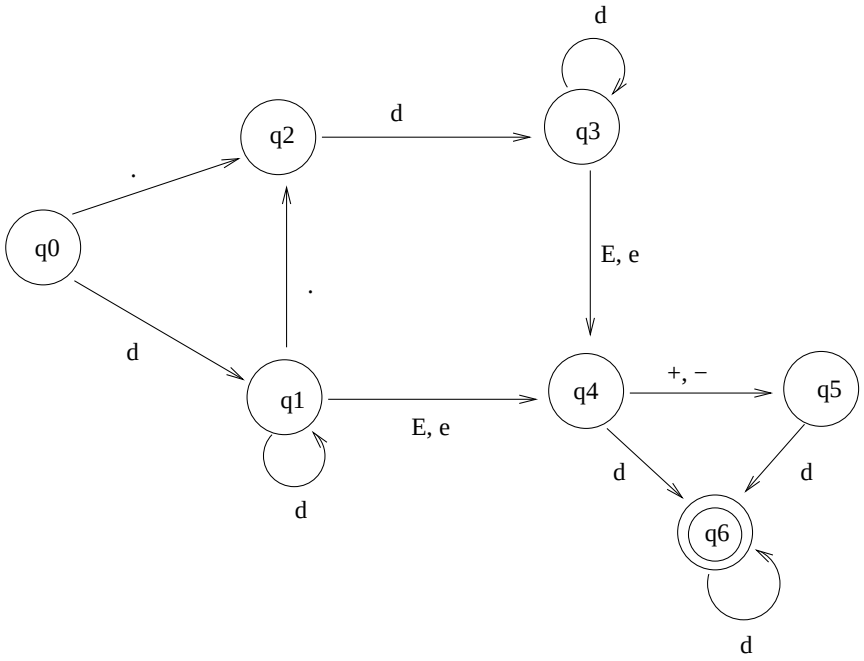
3.2.1 Äärellisen automaatin esitys

- tilasiirtymäkaavio

- tilasiirtymätaulukko

	50 snt	1 euro
$\rightarrow q_0$	q_1	q_2
q_1	q_2	q_3
q_2	q_3	q_4
q_3	q_4	q_4
$\leftarrow q_4$	q_4	q_4

- Esimerkki: C-kielen mukaisen etumerkittömän liukuluvun tunnistava automaatti:



– Tilasiirtymätaulukkona:

		d	.	E, e	+, −
→	q_0	q_1	q_2		
	q_1	q_1	q_3	q_4	
	q_2	q_3			
←	q_3	q_3		q_4	
	q_4	q_6			q_5
	q_5	q_6			
←	q_6	q_6			

Tässä d $= \{0, 1, \ldots, 9\}$. Taulukon puuttuvat kohdat vastaavat virhetilaa "Error", joka käytännössä jätetään merkitsemättä selkeyden takia

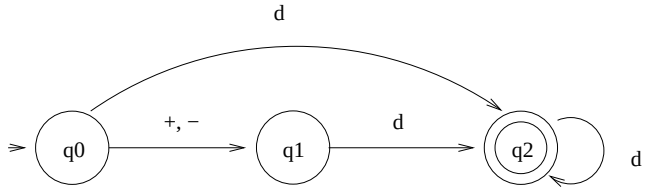
– Ohjelmana:

```
int IsDigit(char c); /* palauttaa 1, jos c on numero, 0 muuten */

int q = 0
char c while ( (c = fgetc(stdin))!=EOF )
{
   switch ( q )
   {
```

```
        case 0: if (IsDigit(c) ) q = 1;
                else if (c=='.') q = 2 else q = 99;
                break;
        case 1: if (IsDigit(c) ) q = 1;
                else if (c=='.') q=3;
                else if (c=='e' || c=='E') q=4; else q=99;
                break;
        case 2: if (IsDigit(c)) q=3; else q=99;
                break;
        case 3: if (IsDigit(c)) q=3;
                else if (c=='e' || c=='E') q = 4 else q = 99;
                break;
        case 4: if (IsDigit(c)) q=6;
                else if (c=='+' || c=='-') q = 5 else q = 99;
                break;
        case 5: if (IsDigit(c)) q=6; else q = 99;
                break;
        case 6: if (IsDigit(c)) q=6; else q = 99;
                break;
        case 99: break;
    }
}
if ( q == 3 || q == 6 ) printf("luku OK!");
else printf("Virheellinen luku");
```

- Äärellisen automaatin pohjalta laadittuun ohjelmaan voidaan liittää myös semanttisia toimintoja
- Esim. Etumerkillisen kokonaisluvun tunnistaminen



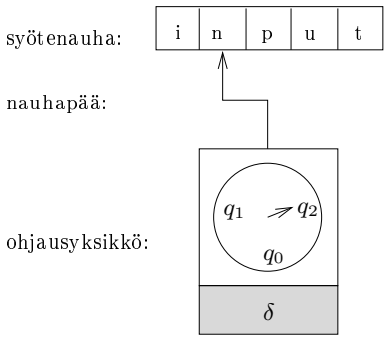
- Vastaava ohjelma, joka lisäksi evaluoi luvun arvon:

```
int IsDigit(char c); /* palauttaa 1, jos c on numero, 0 muuten */
```

```
int q = 0
char c int sign = 1; int val = 0; while ( (c = fgetc(stdin))!=EOF )
{
   switch ( q )
   {
      case 0: if (c=='+' || c=='-') {
              q=1;
              if (c=='-') sign = -1;
              }
              else if (IsDigit(c) {
              q=2;
              val = c -' 0';
              }
              break;
      case 1: if (IsDigit(c) {
              q=2;
              val = c -' 0';
              }
              else q=99;
              break;
      case 2: if (IsDigit(c)) {
              q=2;
              val = 10 * val + (c -' 0');
              }
              else q=99;
              break;
      case 99: break;
   }
}
if ( q == 2) printf("luvun arvo on %d", sgn * val);
else printf("Virheellinen luku");
```

3.2.2 Äärellisen automaatin formaali määrittely

- Äärellinen automaatti M
  - äärellistilainen *ohjausyksikkö*, jonka toimintaa säätelee automaatin *siirtymäfunktio* δ
  - merkkipaikkoihin jaettu *syötenauha*



  - *nauhapää*, joka kullakin hetkellä osoittaa yhtä syötenauhan merkkiä

- Automaatin "toiminta":
  - 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

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

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

  missä
  - Q on automaatin *tilojen* äärellinen joukko;
  - Σ on automaatin *syöteaakkosto*;
  - $\delta : Q \times \Sigma \to Q$ on automaatin *siirtymäfunktio*;
  - $q_0 \in Q$ on automaatin *alkutila*;
  - $F \subseteq Q$ on automaatin *(hyväksyvien) lopputilojen* joukko.

- Esim. reaalilukuautomaatin formaali esitys:

$$\begin{aligned} M \;=\; & (\{q_0, \ldots, q_6, error\}, \{\texttt{0,1,...,9,.,E,e,+,-}\}, \\ & \delta, q_0, \{q_3, q_6\}), \end{aligned}$$

missä δ on kuten aiemmin taulukossa; esim.

$$\delta(q_0, \mathtt{0}) = \delta(q_0, \mathtt{1}) = \cdots = \delta(q_0, \mathtt{9}) = q_1,$$
$$\delta(q_0, .) = q_2, \quad \delta(q_0, E) = error, \quad \delta(q_1, \mathtt{E}) = q_4 \quad \text{jne.}$$

- Automaatin *tilanne* on pari $(q, w) \in Q \times \Sigma^*$
  - automaatin *alkutilanne syötteellä* x on pari (q_0, x)
  - q on nykyinen tila ja w on syötemerkkijonon käsittelemätön osa
- Tilanne (q, w) *johtaa suoraan* tilanteeseen (q', w'), merk.
$$(q, w) \underset{M}{\vdash} (q', w'),$$
jos on $w = aw'$ $(a \in \Sigma)$ ja $q' = \delta(q, a)$.
Tilanne (q', w') on tilanteen (q, w) *välitön seuraaja*
- Tilanne (q, w) *johtaa tilanteeseen* (q', w') eli tilanne (q', w') on tilanteen (q, w) *seuraaja*, merk.
$$(q, w) \underset{M}{\vdash^*} (q', w'),$$
jos on olemassa välitilannejono (q_0, w_0), (q_1, w_1), $\ldots$, (q_n, w_n), $n \geq 0$, siten että
$$(q, w) = (q_0, w_0) \underset{M}{\vdash} (q_1, w_1) \underset{M}{\vdash} \cdots \underset{M}{\vdash} (q_n, w_n) = (q', w').$$
Erikoistapaus: $n = 0$, $(q, w) \underset{M}{\vdash^*} (q, w)$ millä tahansa tilanteella (q, w)
- Automaatti M *hyväksyy* merkkijonon $x \in \Sigma^*$, jos on voimassa
$$(q_0, x) \underset{M}{\vdash^*} (q_f, \epsilon) \qquad \text{jollakin } q_f \in F;$$
muuten M *hylkää* x:n. Ts. automaatti hyväksyy x:n, jos sen alkutilanne syötteellä x johtaa johonkin hyväksyvään lopputilanteeseen
- *Määritelmä:* Automaatin M *tunnistama kieli*
$$L(M) = \{x \in \Sigma^* \mid (q_0, x) \underset{M}{\vdash^*} (q_f, \epsilon) \quad \text{jollakin } q_f \in F\}$$
- Esim. merkkijonon "`0.25E2`" käsittely edellä esitetyllä liukulukuautomaatilla:
$$\begin{array}{lllll} (q_0, \mathtt{0.25E2}) & \vdash & (q_1, \mathtt{.25E2}) & \vdash & (q_3, \mathtt{25E2}) \\ & \vdash & (q_3, \mathtt{5E2}) & \vdash & (q_3, \mathtt{E2}) \\ & \vdash & (q_4, \mathtt{2}) & \vdash & (q_6, \epsilon). \end{array}$$
Koska $q_6 \in F = \{q_3, q_6\}$, on siis $\mathtt{0.25E2} \in L(M)$

3.3 Automaattien minimointi

- Kaksi automaattia, jotka tunnistavat täsmälleen saman kielen ovat keskenään *ekvivalentteja*
- Äärellinen automaatti on *minimaalinen* jos se on tilamäärältään pienin ekvivalenttien automaattien joukossa
- Automaatti, jossa on enemmän tiloja kuin ekvivalentissa minimaalisessa automaatissa on *redundantti*
- Automaatteja muodostava algoritmit eivät aina tuota minimaalista automaattia
- On helpompi nähdä mikä on minimaalisen automaatin tunnistama kieli kuin redundantin automaatin tunnistama kieli
- On turha tallettaa ylimääräisiä tiloja
- Minimaalisen automaatin käsittely on tehokkaampaa kuin redundantin automaatin

3.3.1 Apukäsitteitä

- Määritellään M:lle laajennettu siirtymäfunktio δ^*, joka voi saada parametrinaan merkkijonon:
  jos $q \in Q$, $x \in \Sigma^*$, niin

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

- Tilojen ekvivalenssi: M:n tilat q ja q' ovat *ekvivalentit*, merk.

$$q \equiv q',$$

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

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

  (ts. jos automaatti q:sta ja q':sta lähtien hyväksyy täsmälleen samat merkkijonot)

- *k-ekvivalenssi*: tilat q ja q' ovat k-ekvivalentit, merk.

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

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

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

(ts. jos mikään enintään k:n pituinen merkkijono ei pysty erottamaan tiloja toisistaan)

- Selvästi pätee:

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

- Minimoinnin idea: syötteenä annetun automaatin tilojen k-ekvivalenssiluokkia ositetaan $(k+1)$-ekvivalenssiluokiksi kunnes saavutetaan täysi ekvivalenssi

3.3.2 Äärellisen automaatin minimointi -algoritmi

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

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)

```
while not(tilasiirtymäfkt yhteensopiva luokkajaon kanssa) {
    jaa luokan sisällä eri tavalla käyttäytyvät tilat eri luokkiin;
}
return M̂ = (Q̂, Σ, δ̂, q̂0, F̂),
```

missä

- $\hat{Q}$=M:n tilaluokat
- $\hat{\delta}$=luokkien välinen siirtymäfunktio
- $\hat{q}_0$=M:n alkutilan luokka

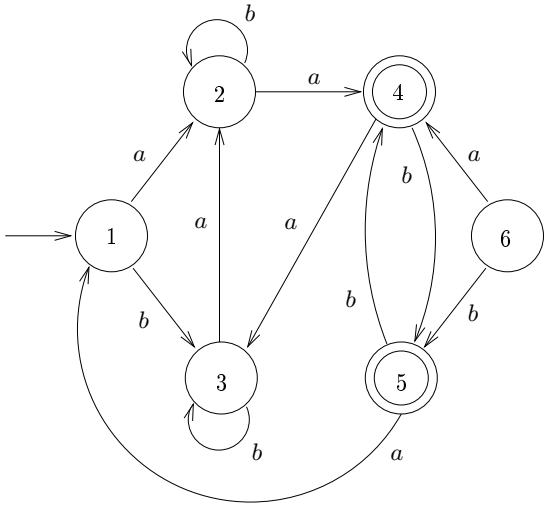
  - $\widehat{F}$=M:n lopputilojen luokat

- Lopputulos
  - M:n kanssa ekvivalentti äärellinen automaatti $\widehat{M}$, jossa on minimimäärätiloja
  - $\widehat{M}$ on tilojen nimeämistä vaille yksikäsitteinen
- Huom: Tiloja on alun perin äärellinen määrä ja joka askeleessa nro 3 (paitsi viimeisessä) ositetaan vähintään yksi tilaluokka, joten algoritmi päättyy aina
- Todistus siitä, että algoritmin tuottama automaatti $\widehat{M}$ on minimiautomaatti ja yksikäsitteinen sivuutetaan (Orposen moniste s. 17–18)

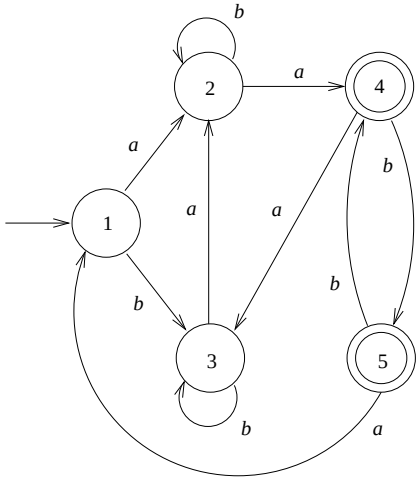
Esimerkki: Olkoon $M = (Q, \Sigma, \delta, q_0, F)$, missä

- tilojen joukko $Q = \{1, 2, 3, 4, 5, 6\}$,
- syöteaakkosto $\Sigma = \{a, b\}$,
- alkutila $q_0 = \{1\}$,
- lopputilojen joukko $F = \{4, 5\}$ ja
- siirtymäfunktio δ:

		a	b
$\rightarrow$	1	2	3
	2	4	2
	3	2	3
$\leftarrow$	4	3	5
$\leftarrow$	5	1	4
	6	4	5



Askel 1: Turhien tilojen poisto



Askel 2: 0-ekvivalenssi

- Osita M:n jäljelle jääneet tilat kahteen luokkaan: lopputiloihin ja muihin tiloihin

			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

- Nyt osituksesta voidaan muodostaa automaatti, jossa
  - kutakin luokkaa vastaa yksi tila ja
  - kustakin tilasta on kaikki erilliset siirtymät, jotka luokkaan kuuluvilla tiloilla on

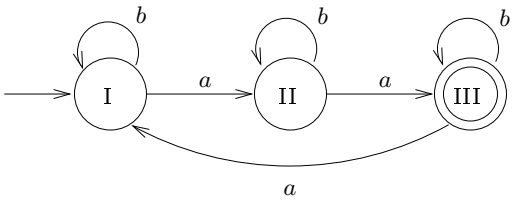
- Tila on *epädeterministinen*, jos siitä voidaan siirtyä jollain merkillä useampaan kuin yhteen tilaan
- Esimerkissä tila I on epädeterministinen, koska merkillä a voidaan siirtyä tilaan I tai tilaan II

Askel 3: k-ekvivalenssi $\Rightarrow$ $(k+1)$-ekvivalenssi

- Jos $\widehat{M}$:ssä ei ole epädeterministisiä tiloja, niin algoritmi päättyy ja palauttaa $\widehat{M}$:n
- Muutoin hienonna kutakin $\widehat{M}$:n epädeterminististä tilaa vastaavan luokan ositusta edelleen:
  - Jaa sen sisällä alkuperäiset tilat eri luokkiin s.e. kustakin luokasta on vain samanlaisia siirtymiä
  - Suorita askel 3 uudestaan
- Esimerkissämme jaetaan tila I kahtia
- Tämän jälkeen ei ole enää epädeterminisiä tiloja ja algoritmi päättyy

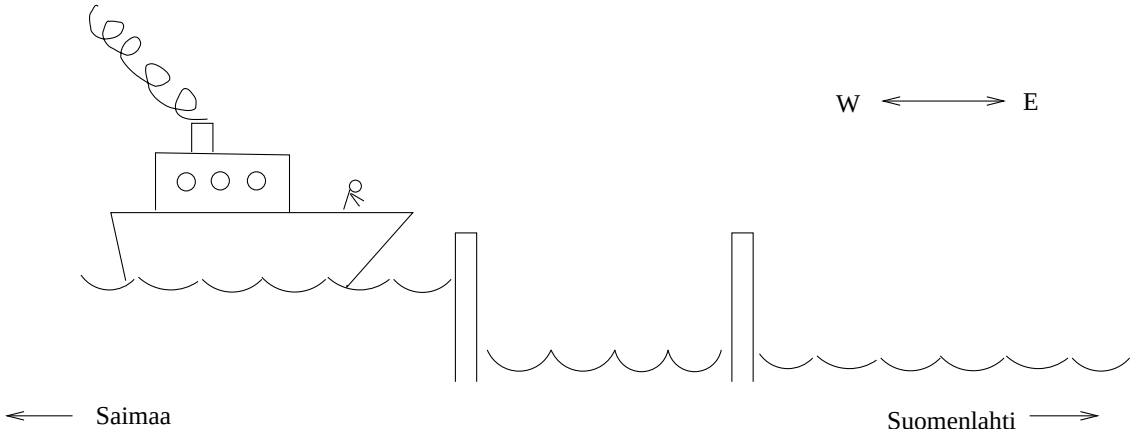
Lopputulos:

			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



Harjoituksia äärellisistä automaateista

1. Olkoon aakkosto $\Sigma = \{a, b\}$. Muodosta automaatti, joka hyväksyy seuraavan kielen:
   a) $L(M) = L(\emptyset)$
   b) $L(M) = L(\emptyset^*)$
   c) $L(M) = L(\epsilon)$
   d) $L(M) = L(\epsilon^*)$
   e) $L(M) = L(\Sigma^*)$



2. Laadi kanavan sulkuautomaatti, joka avaa ja sulkee sulut sekä huolehtii veden pinnan laskusta ja nostamisesta automaattisesti. Länsisulun saa avata vain kun vesi on ylhäällä ja itäsulun vain kun vesi on alhaalla. Vettä saa nostaa ja laskea vain, kun sulkuportit ovat kiinni. Automaatti saa valvontajärjestelmältä seuraavia syötetietoja:
   VL = vesi laskettu
   VN = vesi nostettu
   LL = laiva lännestä
   LI = laiva idästä
   SK = sulut kiinni

LS = laiva suluissa

Voit olettaa, että laivojen välillä automaatti odottaa toinen sulku auki, kunnes saa tiedon uuden laivan saapumisesta.

(Huom! Automaatilla ei siis ole erityisiä alku- ja lopputiloja.)

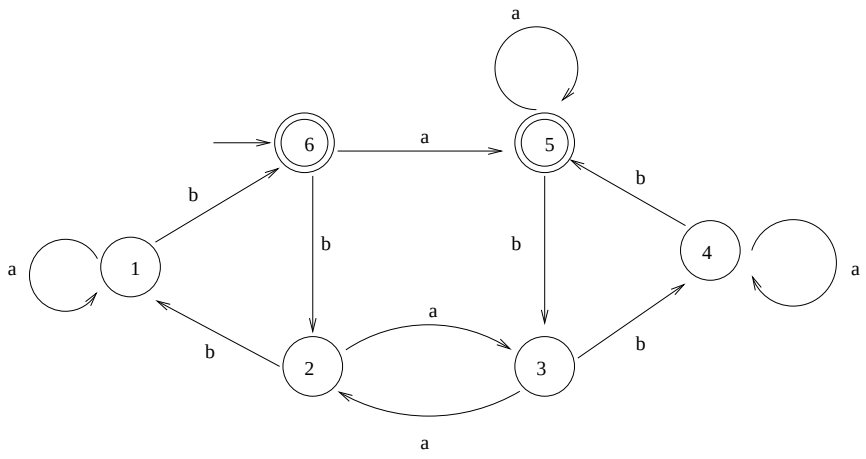
3. Laadi äärellinen automaatti, joka kuvaa kahden kerroksen väliä kulkevan hissin toimintaa. Hissi voi olla joko ylhäällä tai alhaalla. Kummastakin kerroksessa on yksinkertainen "tänne"-nappi ja hissin sisällä "ylös"- ja "alas"-nappit. Hississä on lisäksi ovi, jonka voi avata tai sulkea; hissi liikkuu vain oven ollessa kiinni. (Automaatilla ei tarvitse olla erityisiä lopputiloja.)

   Syöte: napin painallukset, oven avaus ja sulkeminen. Tilat: ylhäällä/alhaalla; ovi auki/kiinni. Merkitään Y = ylhäällä, A = alhaalla, a = ovi auki, k = ovi kiinni.

4. Olkoon aakkosto $\Sigma = \{0, 1\}$. Laadi deterministinen äärellinen automaatti M, joka tunnistaa seuraavan kielen

   a) $L(M) = \{w | w$:n pituus on pariton$\}$
   b) $L(M) = \{w | 1$:ien lukumäärä w:ssä on kolmella jaollinen$\}$
   c) $L(M) = \{w | w$:ssä on parillinen määrä 1:iä ja 0:ia$\}$

5. Muodosta seuraavaa determinististä äärellistä automaattia vastaava minimiautomaatti:



6. Säännöllisen kielen generointi: ”Runoautomaatti”

   Toteuta ohjelma, joka tuottaa säännöllistä lauseketta

   (ATTR)* SUBJ PRED (ATTRa)* OBJ(ATTRIB $\cup \epsilon$)

   vastaavia rivejä. Kielet ATTR, SUBJ, OBJ, PRED ja ATTRIB koostuvat esim. seuraavista sanoista:

   ATTR = {paksu, musta}
   SUBJ={kissa, kuu, kala}
   OBJ={kissaa, kuuta, kalaa}
   PRED={paistaa, katselee, ui}
   ATTRIB={taivaalla, järvessä}

   Ohjelmasi voi siis tuottaa esimerkiksi seuraavanlaisia ”runonsäkeitä”:
   Musta kissa syö paksua kalaa järvessä
   Kuu katselee taivaalla

   Huom! Määrittele, ettei objekti voi seurata kaikkia predikaatteja (esim. verbi ”uida”).

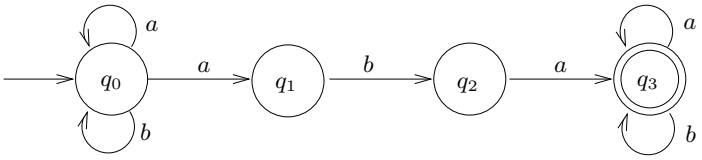
   Täydennä esimerkkiä mielestäsi (runoon) sopivilla sanoilla! Lisää huudahduslauseet

   ”Mikä (ATTR) (SUBJ)!”

   sekä kysymykset

   ”Oletko (ATTR) (SUBJ)?”

3.4 Epädeterministiset äärelliset automaatit



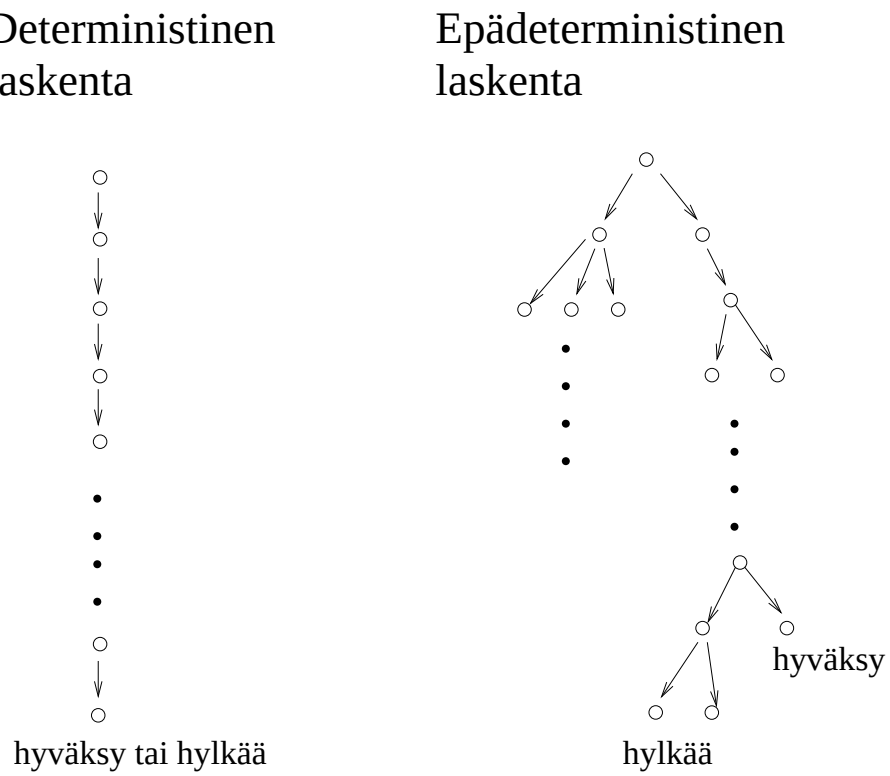
- Esimerkiksi alimerkkijonon *aba* etsiminen aakkoston $\{a, b\}$ merkkijonoista on luontevasti kuvattavissa epädeterministisellä automaatilla. Epädeterminismi siis helpottaa automaatin konstruointia
- Epädeterminististen automaattien avulla luodaan yhteys determinististen äärellisten automaattien ja *säännöllisten kielten* välille
  - deterministiset ja epädeterministiset automaatit tunnistavat täsmälleen samat kielet
  - epädeterministiset automaatit tunnistavat täsmälleen säännölliset kielet
  - $\Rightarrow$ Siis deterministiset automaatit tunnistavat täsmälleen säännölliset kielet
- Epädeterministisellä automaatilla siirtymäfunktio liittää vanhan tilan ja syötemerkin pariin (q, x) *joukon* mahdollisia seuraavia tiloja
- Epädeterministinen automaatti hyväksyy merkkijonon jos jokin mahdollisten tilojen jono johtaa lopputilaan. Jos yhtään tällaista jonoa ei ole, niin epädeterministinen automaatti hylkää syötemerkkijonon
- Esim. kuvan automaatti hyväksyy syötejonon *abbaba*, koska se voidaan käsitellä seuraavasti:

$$(q_0, abbaba) \vdash (q_0, bbaba) \vdash (q_0, baba)$$
$$\vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_2, a) \vdash (q_3, \epsilon)$$

- Toisaalta voidaan myös päätyä hylkäävään tilaan

$$(q_0, abbaba) \vdash (q_0, bbaba) \vdash (q_0, baba)$$
$$\vdash (q_0, aba) \vdash (q_0, ba) \vdash (q_0, a) \vdash (q_0, \epsilon)$$

- Epädeterministinen automaatti voidaan ajatella suorittavan kaikki johdot rinnakkain



- *Määritelmä:* 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,
  - $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ on (joukkoarvoinen) siirtymäfunktio,
  - $q_0 \in Q$ on alkutila ja
  - $F \subseteq Q$ lopputilojen joukko.

- Kuvan automaatin siirtymäfunktio

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

- Nyt virhetilanne on helposti ilmaistavissa tyhjän seuraajatilajoukon avulla (esim. $\delta(q, a) = \emptyset$ merkitsee sitä, että a aiheuttaa virheen tilassa q_1)
- (q, w) *voi* johtaa suoraan tilanteeseen (q', w'), $(q, w) \underset{M}{\vdash} (q', w')$, jos $w = aw'$ ja $q' \in \delta(q, a)$. Tilanne (q', w') on (q, w):n *mahdollinen* välitön seuraaja
- Muutoin määritelmät epädeterministisille automaateille ovat samat kuin aiemmin

- Deterministiset automaatit ovat epädeterminististen erikoistapaus $\Rightarrow$ kaikki edellisillä tunnistettavat kielet ovat tunnistettavissa myös jälkimmäisillä
- Mutta myös kääntäen: *deterministiset ja epädeterministiset äärelliset automaatit ovat yhtä vahvoja*

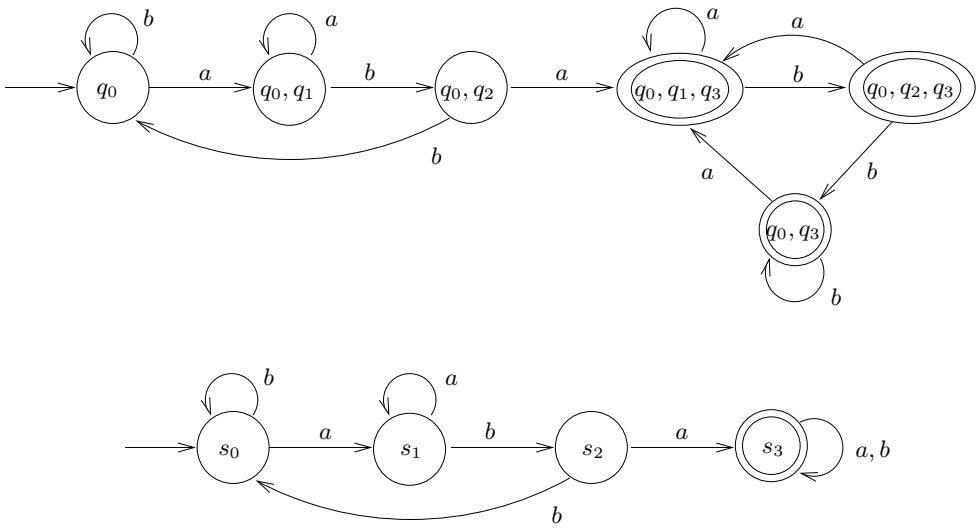
3.4.1 Automaatin determinisointi

- Muodostetaan epädeterminististä automaattia M vastaava deterministinen automaatti $\widehat{M}$:

  1. Muodosta $\widehat{M}$:n tilat $S \subseteq \mathcal{P}(Q)$
     - ts. kaikki M:n tilojen joukot $\mathcal{P}(Q)$
     - Merk. $\mathcal{P}(Q) = \{\emptyset, s_1, s_2, ..., s_m\}$, missä tyhjä joukko vastaa virhetilaa ja $m = 2^n - 1$
  2. Lisää $\widehat{M}$:n tilojen välille siirtymät:
     - $s_i \xrightarrow{a} s_j$, missä $s_j = \bigcup\{q' | \delta(q, a) = q', q \in s_i\}$
     - ts. tilajoukkojen s_i ja s_j välille siirtymä $s_i \xrightarrow{a} s_j$, jos niiden osajoukkojen välillä oli siirtymä: $\forall q' \in s_j \; \exists q \in s_i$ siten että $q \xrightarrow{a} q'$.
     - s_i:n seuraajajoukkoon merkillä a kuuluvat siis kaikki ne tilat q', jotka voidaan saavuttaa s_i:n tiloista q merkillä a
     - Huom! Voisimme valita seuraajat myös seuraavalla ehdolla: $\exists q \in s_i \; \exists q' \in s_j$ siten että $q \xrightarrow{a} q'$. Nyt tulee kuitenkin paljon turhia siirtymiä, joista on riesaa minimointivaiheessa!
  3. Alkutilaksi $\{q_0\}$ (alkuperäisestä alkutilasta muodostettu joukko)
  4. Lopputiloiksi kaikki alkuperäisen lopputilan q_f sisältävät tilajoukot s_i, $q_f \in s_i$
  5. Karsi turhat tilat, joita ei voi saavuttaa alkutilasta
  6. Minimoi automaatti

  - jaa loppu- ja muihin tiloihin
  - hienonna luokkajakoa, kunnes yhdenmukainen siirtymäfkt:n kanssa

- Esimerkki:



- *Lause:* Olk. $A = L(M)$ jonkin epädeterministisen äärellisen automaatin M tunnistama kieli. Tällöin on olemassa deterministinen automaatti $\widehat{M}$, jolla $L(\widehat{M}) = A$.
  *Todistus: Olk. $A = L(M)$, $M = (Q, \Sigma, \delta, q_0, F)$. Laaditaan determ. automaatti $\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F})$, joka simuloi M:n toimintaa kaikissa sen kullakin hetkellä mahdollisissa tiloissa rinnakkain. Automaatin $\widehat{M}$ tilat vastaavat M:n tilojen joukkoja

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

  Tarkastetaan, että $L(\widehat{M}) = L(M)$. Kielten ekvivalenssi seuraa, kun todistetaan kaikilla $x \in \Sigma^*$ ja $q \in Q$:

$$(q_0, x) \underset{M}{\vdash^*} (q, \epsilon) \Leftrightarrow (\{q_0\}, x) \underset{\widehat{M}}{\vdash^*} (S, \epsilon) \text{ ja } q \in S.$$

  Todistus induktiolla merkkijonon x pituuden suhteen:

1. $|x| = 0$: $(q_0, \epsilon) \underset{M}{\vdash}^* (q, \epsilon) \Leftrightarrow q = q_0$.
   Samoin $(\{q_0\}, \epsilon) \underset{\widehat{M}}{\vdash}^* (S, \epsilon) \Leftrightarrow S = \{q_0\}$.
2. *Induktio-oletus*: väite pätee kun $|x| \leq k$.
3. $|x| = k + 1$: tällöin $x = ya$ jollakin y, $|y| = k$, jolle väite pätee induktio-oletuksen perusteella. Nyt

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

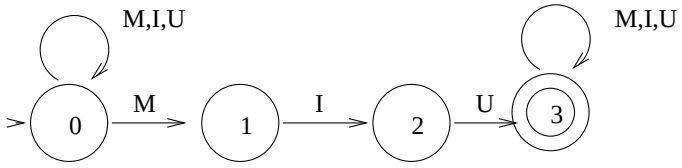
□

3.4.2 Hahmontunnistus epädeterministisellä automaatilla

Epädeterministisillä automaateilla voidaan kätevästi kuvata hahmontunnistusongelmia. Ohjelmatoteutusta verten automaatti on kuitenkin determinisoitava. Edellä huomasimme, että pahimmassa tapauksessa vastaavan deterministisen automaatin tilamäärä on eksponentiaalinen. Hahmontunnistusongelmissa determinisointialgoritmi tuottaa kuitenkin automaatin, jossa on *yhtä monta tilaa* kuin alkuperäisessä epädeterministisessä automaatissa, ja joka on samalla *minimiautomaatti!*

Esim. Olkoon aakkosto $\{M, I, U\}$. Esiintyykö hahmo MIU merkkijonossa? Vastaava epädeterministinen automaatti:

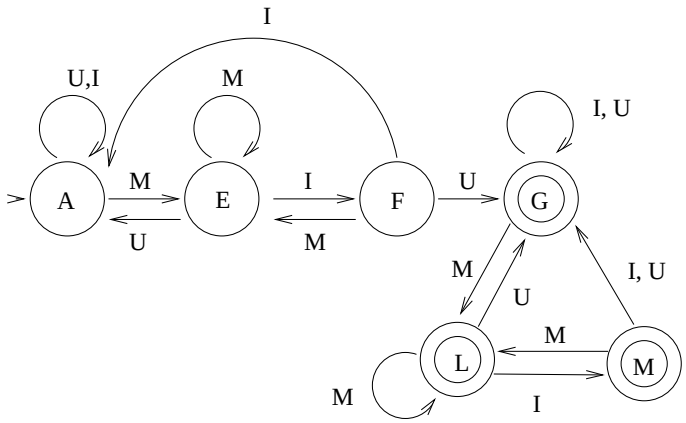
Muodostetaan vastaava deterministinen automaatti. Taulukossa on esitetty kaikki tilat, myös ne joita ei voi saavuttaa alkutilasta. Käytännössä voit oikaista, kun



Kuva 3.1: Epädeterministinen MIU-automaatti.

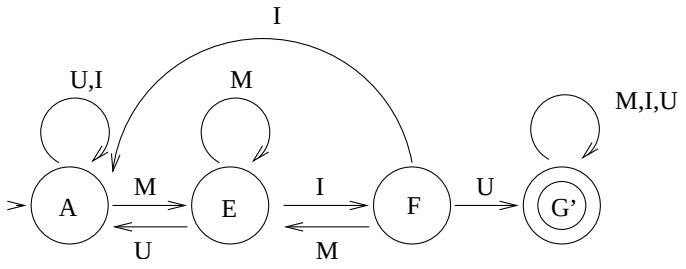
lähdet liikkeelle alkutilasta ja lasket ensin sen seuraajat, sitten näiden tilojen seuraajat, jne. (taulukossa lihavoidut kirjaimet). Nyt mukaan tulevat vain ne tilat, jotka voit oikeasti saavutata alkutilasta.

			M	I	U
		$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow$	**A**	$\{0\}$	$\{0,1\}$=E	$\{0\}$=A	$\{0\}$=A
	B	$\{1\}$	$\emptyset$	$\{2\}$=C	$\emptyset$
	C	$\{2\}$	$\emptyset$	$\emptyset$	$\{3\}$=D
$\leftarrow$	D	$\{3\}$	$\{3\}$=D	$\{3\}$=D	$\{3\}$=D
	E	$\{0,1\}$	$\{0,1\}$=E	$\{0,2\}$=F	$\{0\}$=A
	F	$\{0,2\}$	$\{0,3\}$=E	$\{0\}$=A	$\{0,3\}$=G
$\leftarrow$	**G**	$\{0,3\}$	$\{0,1,3\}$=L	$\{0,3\}$=G	$\{0,3\}$=G
	H	$\{1,2\}$	$\emptyset$	$\{2\}$=C	$\{3\} = D$
$\leftarrow$	I	$\{1,3\}$	$\{3\}$=D	$\{2,3\}$=J	$\{3\}$=D
$\leftarrow$	J	$\{2,3\}$	$\{3\}$=D	$\{3\}$=D	$\{3\}$=D
	K	$\{0,1,2\}$	$\{0,1\}$=E	$\{0,2\}$=F	$\{0,3\}$=G
$\leftarrow$	**L**	$\{0,1,3\}$	$\{0,1,3\}$=L	$\{0,2,3\}$=M	$\{0,3\}$=G
$\leftarrow$	**M**	$\{0,2,3\}$	$\{0,1,3\}$=L	$\{0,3\}$=G	$\{0,3\}$=G
$\leftarrow$	N	$\{1,2,3\}$	$\{3\}$=D	$\{2,3\}$=J	$\{3\}$=D
$\leftarrow$	O	$\{0,1,2,3\}$	$\{0,1,3\}$=L	$\{0,2,3\}$=M	$\{0,3\}$=G



Kuva 3.2: Deterministinen MIU-automaatti.

Lopuksi yhdistetään lopputilat (minimointialgoritmin mukaan).

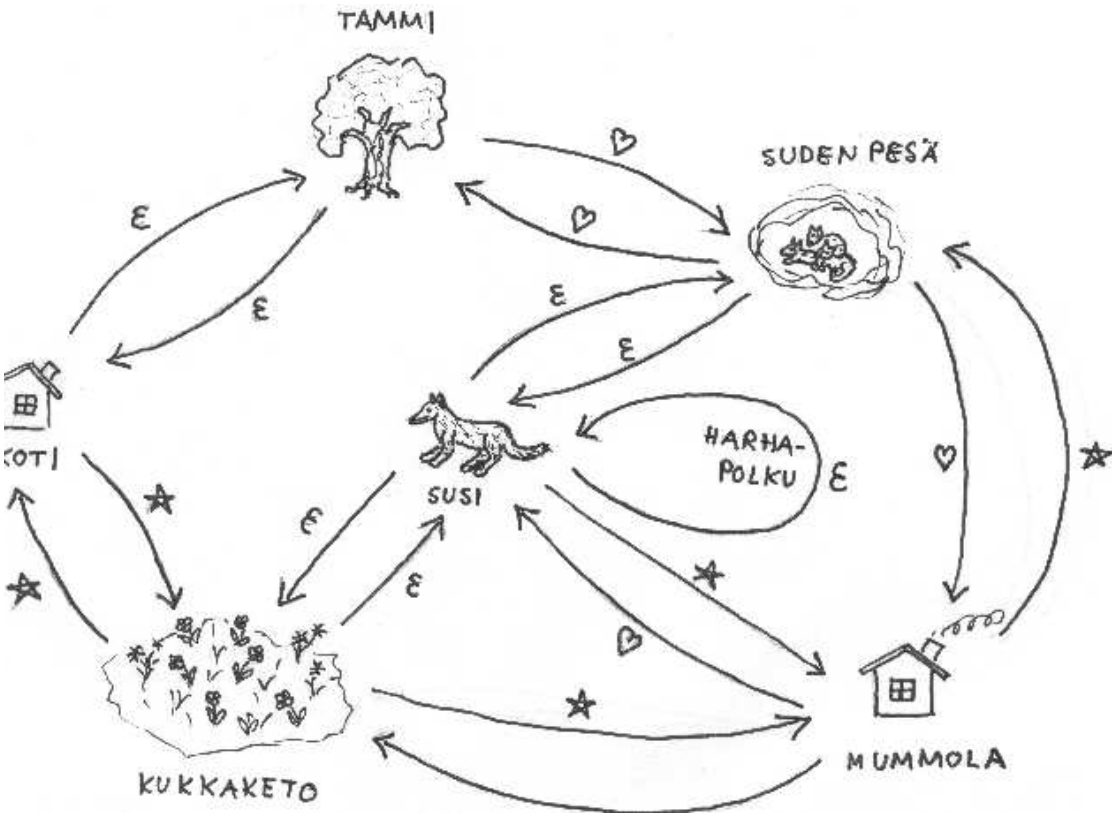


Kuva 3.3: Lopputulos: minimaalinen deterministinen MIU-automaatti.

3.4.3 ϵ-automaatti

Ongelma: Peikkojen pipariautomaatti

Punahilkka haluaa vierailla Mummonsa luona. Metsässä risteilee useita polkuja, joista osaa voi kulkea vapaasti, mutta osaa vartioi peikko. Peikot vaativat läpipääsystä tullia yhden piparkakun – joko täden- tai sydämenmuotoisen mieltymyksensä mukaan. Karttaan on merkitty peikkojen kullakin polulla vaatimat piparkakut. ϵ tarkoittaa sitä, ettei polulta peritä mitään tullia. Äiti on kuitenkin antanut Punahilkalle vain yhden tähtipiparkakun. Miten pitkälle Punahilkka pääsee? Entä yhdellä sydänpiparkakulla?



Kiltti susi kutsuu Punahilkan pesäänsä ja tarjoaa yhden sydänpiparkakun. Nyt Punahilkka pääsee Mummolaan, mutta miten hän pääsee takaisin kotiin?

ϵ-automaatti on epädeterministinen äärellinen automaatti, jossa sallitaan *ϵ-siirtymät* ts. siirtymät tyhjällä merkkijonolla. Kyseessä on siis äärellisten automaattien mallin laajennus

Määritelmä: ϵ-automaatti on viisikko $M = (Q, \Sigma, \delta, q_0, F)$, missä siirtymäfunktio δ on kuvaus

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

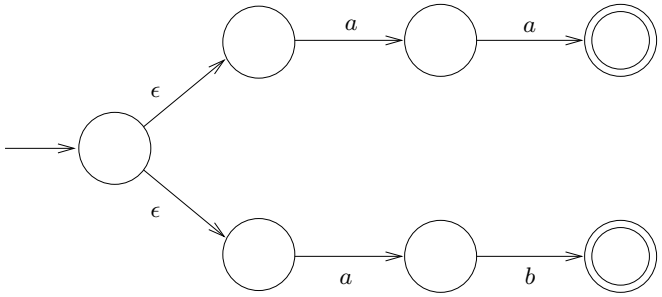
Tilanne (q, w) voi johtaa suoraan tilanteeseen (q', w')

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

jos

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

Muut määritelmät kuten tavallisilla epädeterministisillä äärellisillä automaateilla



Kuva 3.4: Esimerkki: Kielen $\{aa, ab\}$ tunnistava ϵ-automaatti.

Lemma: 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 kuten M, mutta simuloi kunkin askelensa yhteydessä myös kaikki M:n mahdolliset ϵ-siirtymät.

Formaalisti:

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

missä

$$\hat{\delta}(q, a) = \{q' \in Q \mid (q, a) \underset{M}{\vdash^*} (q', \epsilon)\};$$

$$\widehat{F} = \begin{cases} F \cup \{q_0\}, & \text{jos } (q_0, \epsilon) \underset{M}{\vdash^*} (q_f, \epsilon) \text{ jollain } q_f \in F; \\ F, & \text{muuten.} \end{cases}$$

□

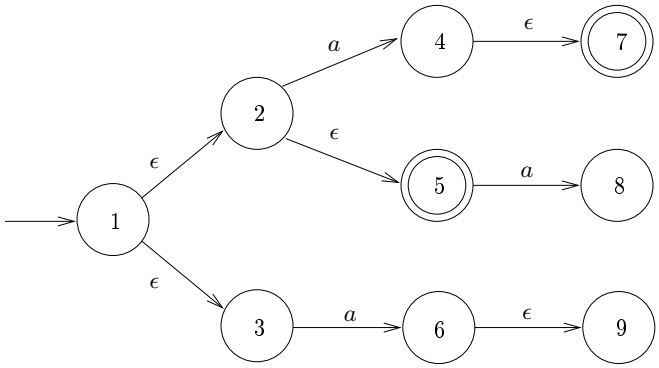
3.4.4 ϵ-siirtymien poisto

- Muodosta siirtymä $q \xrightarrow{a} q'$, jos $(q, a) \underset{M}{\vdash}^* (q', \epsilon)$

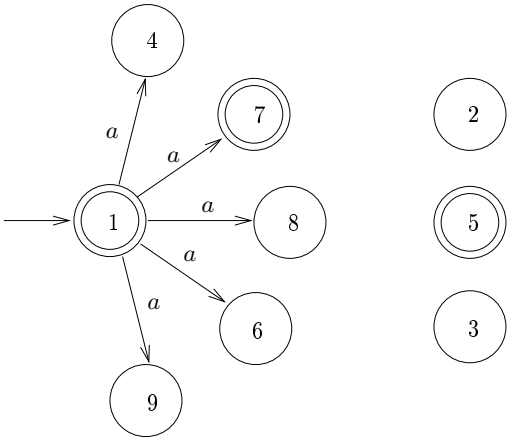
  esim. $q \overset{\epsilon}{\rightarrow} q_1 \overset{a}{\rightarrow} q_2 \overset{\epsilon}{\rightarrow} q'$

- Lisää alkutila q_0 lopputiloihin, jos $(q_0, \epsilon) \underset{M}{\vdash}^* (q_f, \epsilon)$ $(q_f \in F)$

  esim. $q_0 \overset{\epsilon}{\rightarrow} q_1 \overset{\epsilon}{\rightarrow} q_f$



Kuva 3.5: ϵ-automaatti M.



- Huom! muodostettavissa ϵ-automaateissa aina yksikäsitteiset alku- ja lopputilat.

3.4.5 Lausekeautomaatit

Määritellään vielä yksi äärellisten automaattien laajennus: lausekeautomaatti:

Määritelmä: Merk. RE_Σ = aakkoston Σ säännöllisten lausekkeiden joukko. *Lausekeautomaatti* on viisikko

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

missä siirtymäfunktio δ on äärellinen kuvaus

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

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

Yhden askelen tilannejohto määritellään:

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

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

Harjoituksia epädeterministisistä automateista

1. Tarkastele luentomateriaalista löytyvää sarjakuvaa epädeterministisestä automaatista tohtorin vastaanotolla. Mistä välivaiheista operaatio koostuu? Meneekö kaikki niin kuin pitääkin?

2. Keksi ainakin yksi arkielämän ilmiö, jota voit parhaiten kuvata epädeterministisellä automaatilla! Piirrä automaattisi siirtymäkaavio. Osaatko determinisoida sen?

3. Laadi epädeterministinen äärellinen automaatti, joka testaa, sisältääkö aakkoston $\{a, b\}$ merkkijonoa *"abaa"* ja determinisoi se.

4. Muodosta epädeterministinen äärellinen automaatti, joka hyväksyy seuraavan kielen. Yritä hyödyntää epädeterminismiä niin paljon kuin mahdollista!
   a) Aakkoston $\{0, 1, ..., 9\}$ ne merkkijonot, joiden viimeinen merkki on esiintynyt aiemmin.
   b) Aakkoston $\{0, 1, ..., 9\}$ ne merkkijonot, joiden viimeinen merkki *ei* ole esiintynyt aiemmin.
   c) Aakkoston $\{0, 1\}$ ne merkkijonot, joissa on kaksi 0:aa, joiden välissä on neljällä jaollinen määrä ykkösiä. Huom! Myös 0 on neljällä jaollinen.

5. Poista ϵ-siirtymät Peikkojen Piparkakkuautomaatista (ks. luentokalvot) ja determinisoi se!

3.5 Äärelliset automaatit ja säännölliset kielet

Osoitetaan seuraava tulos:
Kieli on säännöllinen $\Leftrightarrow$ Kieli voidaan tunnistaa äärellisellä automaatilla.

Todistuksen idea:

1. Kieli $L(r)$ on säännöllinen $\Rightarrow$ $L(r)$ voidaan tunnistaa äärellisellä automaatilla M:

   - Muodostetaan säännöllistä lauseketta r vastaava ϵ-automaatti
   - Tällöin on olemassa vastaava ϵ-siirtymätön epädeterministinen automaatti

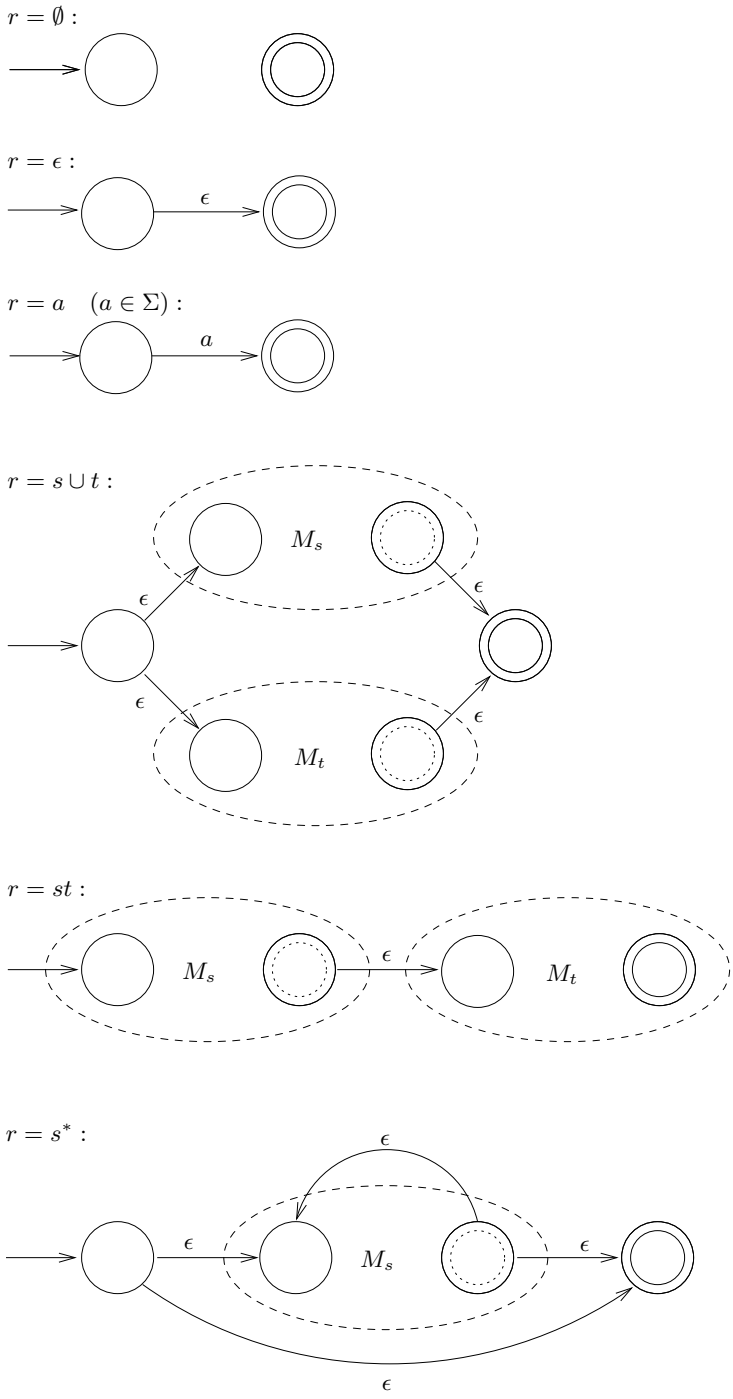
   - Haluattessa epädeterministinen automaatti voidaan vielä determinisoida (ja minimoida)

2. Kieli $L(M)$ voidaan tunnistaa äärellisellä automaatilla $M \Rightarrow L(M)$ on säännöllinen kieli:

   - Tarkastellaan äärellisten automaattien laajennosta, *lausekeautomaatteja* - jos väite pätee lausekeautomaateille, se pätee myös tavallisille äärellisille automaateille (jotka ovat niiden erikoistapaus)
   - Redusoidaan lausekeautomaatti korkeintaan 2-tilaiseksi automaatiksi, josta voidaan lukea suoraan vastaava säännöllinen lauseke

Lause: Jokainen säännöllinen kieli voidaan tunnistaa äärellisellä automaatilla.
Todistus:

- Muodostetaan mielivaltaista säännöllistä lauseketta r vastaava ϵ-automaatti M_r, jolla $L(M_r) = L(r)$ (ks. kuva 3.6)
- M_r:stä voidaan poistaa ϵ-siirtymät edellisen lemman mukaisesti, ja tarvittaessa voidaan syntyvä epädeterministinen automaatti determinisoida aiemmin esitetyn konstruktion avulla. □



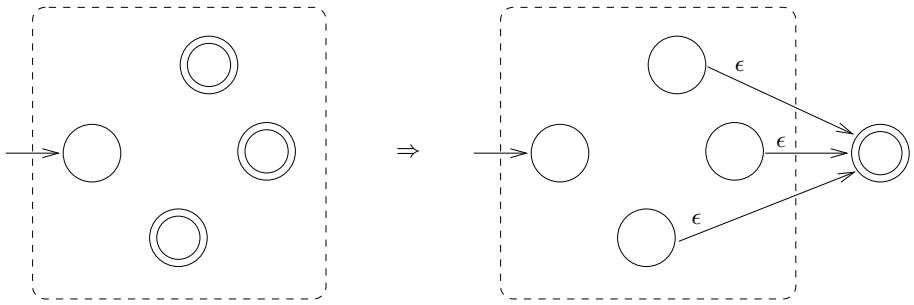
Kuva 3.6: Lauseketta r vastaavan ϵ-automaatin M_r muodostaminen.

Lause: Jokainen äärellisellä automaatilla tunnistettava kieli on säännöllinen.
Todistus: Osoitetaan, että jokainen lausekeautomaatilla tunnistettava kieli on säännöllinen.

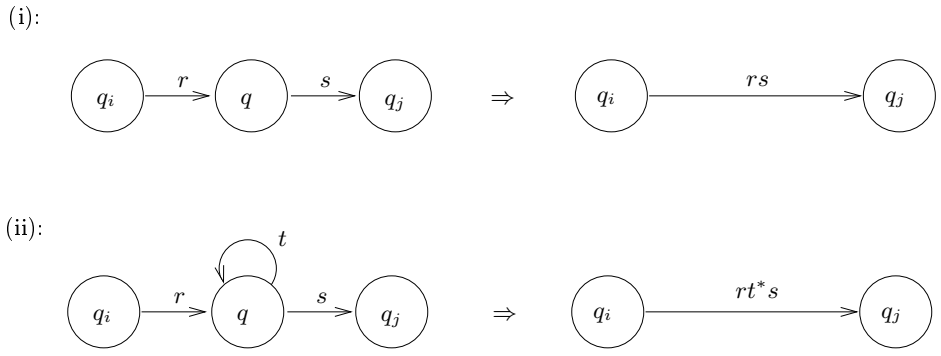
Idea: Redusoidaan lausekeautomaatti s.e. siitä voidaan lukea vastaava säännöllinen lauseke:

- Yhdistetään M:n lopputilat yhdeksi ϵ-siirtymillä (ks. kuva 3.7)
- while (muita kuin alku- ja lopputiloja) {
  - valitse q, $q \neq q_0$, $q \neq q_f$ ($q_f \in F$)
  - poista q reitiltä $q_i \rightarrow q \rightarrow q_j$ ($q_i = q$:n edeltäjätila ja $q_j = q$:n seuraajatila) reduktiosäännöillä (kuva 3.8) (Huom! Muista käydä läpi kaikki q:ta sisältävät polut!)
  - yhdistä rinnakkaiset siirtymät (kuva 3.9)

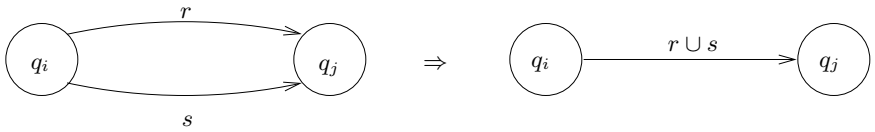
  }



Kuva 3.7: Lausekeautomaatin lopputilojen yhdistäminen.

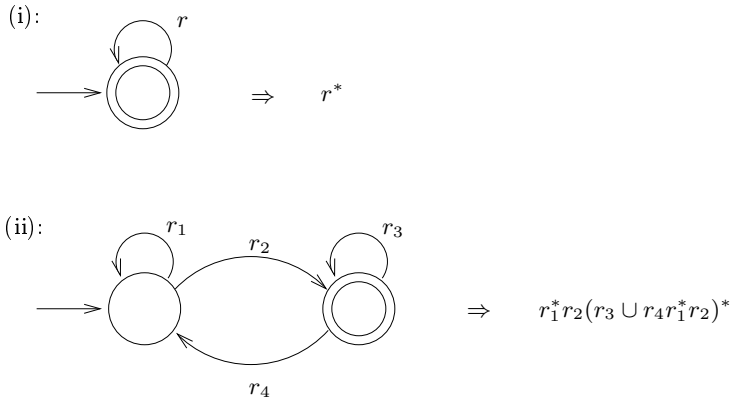


Kuva 3.8: Tilan poistaminen lausekeautomaatista.



Kuva 3.9: Rinnakkaisten siirtymien yhdistäminen lausekeautomaatissa.

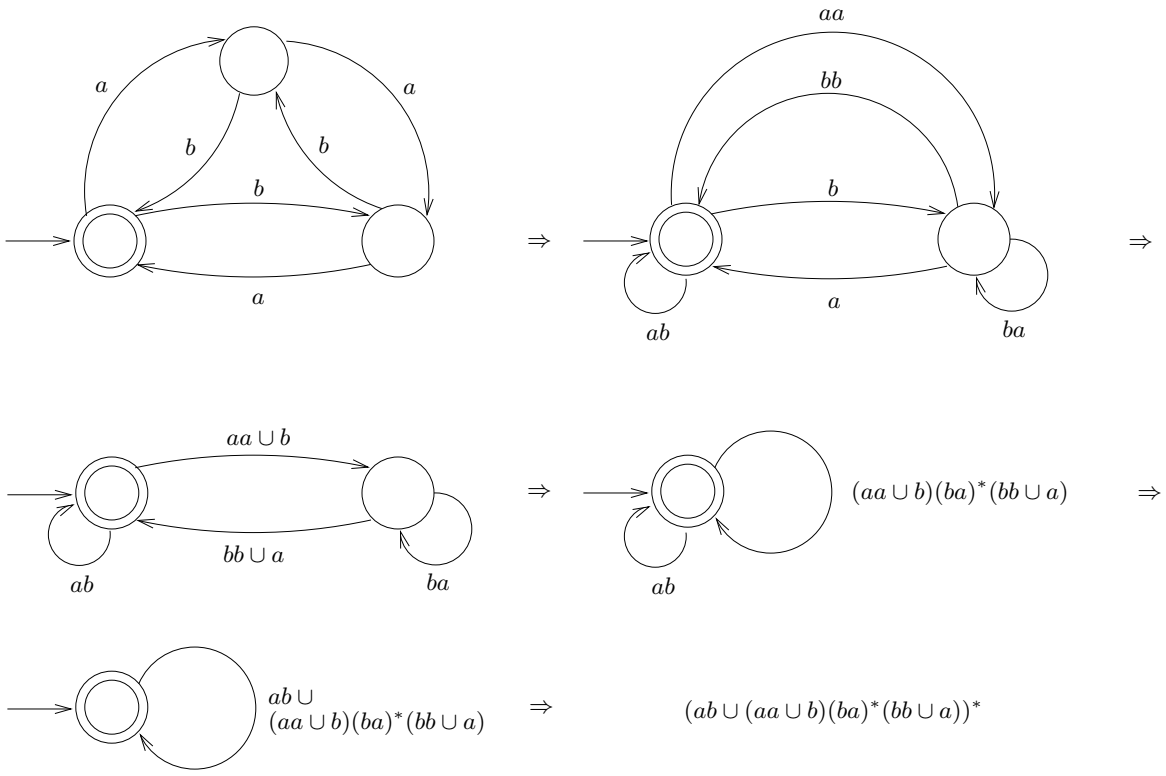
- Tiivistyksen päättyessä jäljellä olevaa enintään 2-tilaista automaattia vastaava säännöllinen lauseke muodostetaan kuten kuvassa 3.10.



Kuva 3.10: Säännöllisen lausekkeen muodostaminen redusoidusta lausekeautomaatista.

□

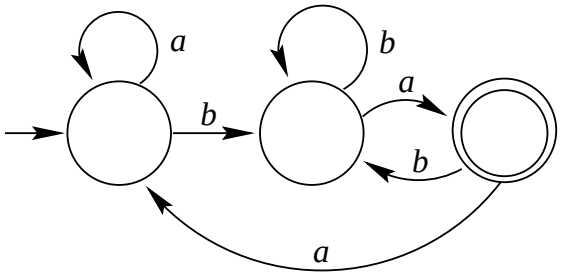
Esimerkki:


a
a
b
b
b
a
⇒
aa
bb
b
a
ab
ba
⇒
aa ∪ b
bb ∪ a
ab
ba
⇒
(aa ∪ b)(ba)*(bb ∪ a)
ab
⇒
ab ∪
(aa ∪ b)(ba)*(bb ∪ a)
⇒
(ab ∪ (aa ∪ b)(ba)*(bb ∪ a))*

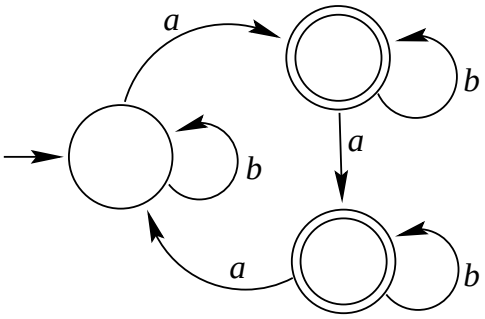
Harjoituksia säännöllisten lausekkeiden ja äärellisten automaattien välisistä muunnoksista

1. Millaisen kielen Peikkojen Piparkakkuautomaatti hyväksyy? Esitä kieli säännöllisenä lausekkeena!

2. Muodosta seuraavia äärellisiä automaatteja vastaavat säännölliset lausekkeet:



(a)



(b)

3. Muodosta seuraavia säännöllisiä lausekkeita vastaavat deterministiset äärelliset automaatit!
a)$(ab)^*(ba)^* \bigcup aa^*$
b)$((ab \bigcup abb)^*a^*)^*)$

3.6 Hauska lisätieto: Säännöllisen kielen sulkeumaominaisuudet

Automaattien avulla voidaan osoittaa seurava tulos:

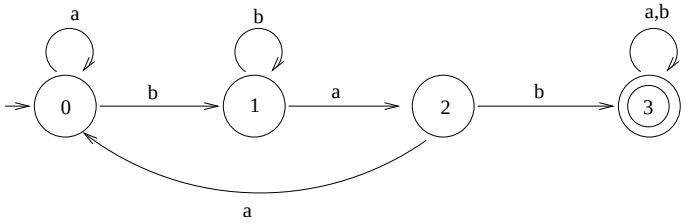
Lause: Olkoon L_1 ja L_2 säännölisiä kieliä. Tällöin myös

1. $L_1 \bigcup L_2$ (kielten yhdiste)
2. $L_1 \bigcap L_2$ (kielten leikkaus)
3. $L_1 L_2$ (kielten katenaatio)
4. $\overline{L_1} = \Sigma^* \setminus L_1$ (kielen komplementti)
5. $(L_1)^*$ (kielen sulkeuma)
6. $(L_1)^R$ (käänteiskieli, jossa kaikki L_1:n sanat on kirjoitettu takaperin)

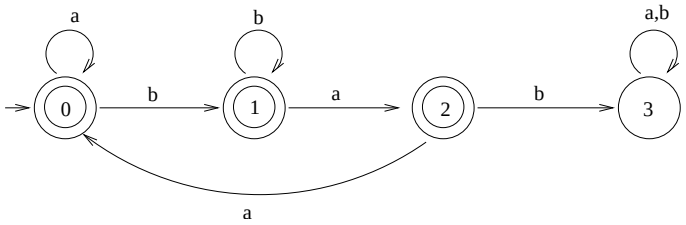
ovat säännöllisiä.

Todistus: Harjoitustehtävä!

Esim.



Kuva 3.11: Automaatti M, joka tunnistaa kielen $L(M) = \{w \in \{a,b\}^* | w$ sisältää merkkijonon $bab\}$.



Kuva 3.12: Komplementtiautomaatti $\overline{M}$, joka tunnistaa kielen $L(\overline{M}) = \{w \in \{a,b\}^* | w$ ei sisällä merkkijonoa $bab\}$.

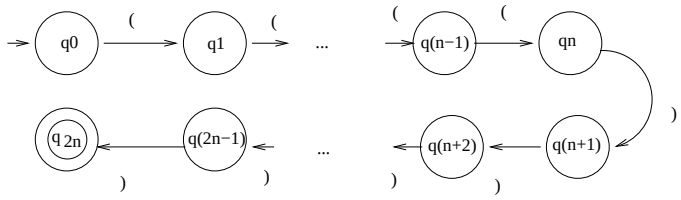
3.7 Harjoituksia

1. Osoita, että säännöllisten kielten luokka on suljettu leikkauksen ja katenaation suhteen! Ts. jos L_1 ja L_2 ovat säännöllisiä kieliä, niin myös $L_1 \bigcap L_2$ ja $L_1 L_2$ ovat säännöllisiä. (Vihje: automaatit.)

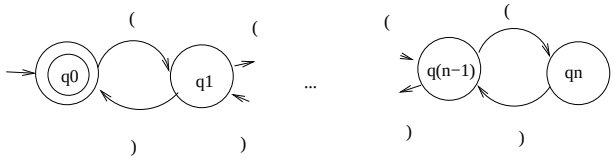
3.8 Säännöllisten kielten rajoituksista

- Minkä tahansa aakkoston formaaleja kieliä (päätösongelmia) on ylinumeroituva määrä, mutta säännöllisiä lausekkeita vain numeroituva määrä ⇒ kaikki kielet eivät voi olla säännöllisiä
- Perusrajoitus: äärellisillä automaateilla on vain rajallinen "muisti"
- Äärelliset kielet ovat aina säännöllisiä
- Milloin on ääretön kieli säännöllinen?
  - oltava jokin toistuva rakenne (sulkeuma)
  - vastaavassa automaatissa silmukka
- Esim. tasapainoisten sulkujonojen muodostama kieli $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ ei ole säännöllinen, eli sitä ei voi tunnistaa äärellisellä automaatilla.

a)



b)



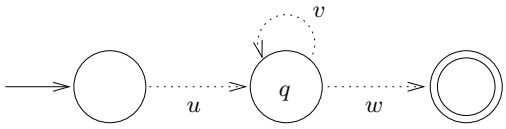
  - tunnistaa tasan n sulkuparia sisältävän merkkijonon
  - entä jos enemmän tai vähemmän sisäkkäisiä sulkuja?
  - esim. $a^{n-1} \in L_{\text{match}}$, $a^{n+1} \in L_{\text{match}}$, mutta automaatti ei tunnista niitä
- Entä kieli $\{a^k b^k \mid k \geq 0\}*$?

  ⇒ ei voida tunnistaa äärellisellä automaatilla (eikä siten myöskään mv. aritmeettisia lausekkeita)
- "Pumppauslemma" formalisoi tämän rajallisen muistin idean

- Idea: mitä tahansa annetun säännöllisen kielen riittävän pitkää merkkijonoa voidaan "pumpata" keskeltä, ilman että kielen tunnistava äärellinen automaatti huomaa muutosta

- *Lemma*[Pumppauslemma]: Olkoon A säännöllinen kieli. Tällöin on olemassa $n \geq 1$ s.e. mikä tahansa $x \in A$, $|x| \geq n$, voidaan jakaa osiin $x = uvw$, $|uv| \leq n$, $|v| \geq 1$, ja $uv^iw \in A$ kaikilla $i = 0, 1, 2, \ldots$
  Todistus:

  - Olk. M jokin A:n tunnistava deterministinen äärellinen automaatti ja n = M:n tilojen määrä.
  - Tarkastellaan automaatin läpikäymiä tiloja sen tunnistaessa merkkijonoa $x \in A$, $|x| \geq n$:
    * Koska $|x| \geq n \Rightarrow$ täytyy kulkea jonkin tilan kautta (ainakin) kaksi kertaa (itse asiassa jo x:n n:ää ensimmäistä merkkiä käsitellessään)
    * Olk. q ensimmäinen tila, jonka automaatti toistaa x:ää käsitellessään.
    * Olk. $u = M$:n käsittelemä x:n alkuosa sen tullessa ensimmäisen kerran tilaan q
    * Olk. $v =$ se osa x:stä u:n jälkeen jonka M käsittelee ennen ensimmäistä paluutaan q:hun, ja
    * $w =$ loput x:stä
    * Tällöin on $|uv| \leq n$, $|v| \geq 1$, ja $uv^iw \in A$ kaikilla $i = 0, 1, 2, \ldots$ □



Kuva 3.13: Merkkijonon $x = uvw \in A$ pumppaus.

- Pumppauslemma on kiinnostava vain äärettömille kielille. (Äärellisille kielille, jotka kaikki ovat säännöllisiä, voidaan valita $n = \max\{|x| \mid x \in L\}+1$, jolloin ei ole olemassa yhtään $x \in L$ s.e. $|x| \geq n$)

- Esim. Väite: Sulkulausekekieli

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

  on epäsäännöllinen.
  Tod.: Vastaoletus: L on säännöllinen. $\Rightarrow \exists$ jokin $n \geq 1$, jonka pituisia L:n merkkijonoja voidaan pumpata.

Valitaan $x = ({}^n)^n$, jolloin $|x| = 2n > n$. Lemman mukaan x voidaan jakaa pumpattavaksi osiin $x = uvw$, $|uv| \leq n$, $|v| \geq 1$; siis on oltava

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

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

- Huom! Pumppauslemma ei sano, että voimme valita u:n ja v:n haluamallamme tavalla, vaan todistuksessa meidän on kokeiltava kaikkia jakoja $x = uvw$ s.e. $|uv| \leq n$ ja $|v| \geq 1$

- Todistusstrategia: Valitaan tarkasteltavan jonon pituudeksi esim. $2n$, jossa ensimmäiset n merkkiä muodostavat uv:n, muttei kiinnitetä tarkkaan ottaen miten

- Esim. Väite: $L = \{a^k b^l c^{k+l} \mid k, l \geq 0\}$ ei ole säännöllinen.
  Tod.: Vastaoletus: L on säännöllinen $\Rightarrow \exists n \geq 1$, jonka pituisia merkkijonoja voidaan pumpata. Valitaan $x = a^n b^l c^{n+l}$, jollakin $l \geq 0$. Nyt $|x| = 2n+2l > n$. Koska $|uv| \leq n$, uv koostuu vain a-merkeistä ja $m = |v| \geq 1$. Tällöin $uv^0w = a^{n-m}b^l c^{n+l} \notin L$. Ristiriita. Siten L ei voi olla säännöllinen. □

  Esim. 2: Väite: $L(a^m b^n c^{m+n}$ ei ole säännöllinen.
  Tod: Tarkastellaan merkkijonoja, joiden pituus vähintään $n = k + l$. Valitaan $x = a^k b^l c^{k+l}$, $|x| = 2k + 2l > n$. Nyt osa uv koostuu x:n korkeintaan $k + l$:stä ensimmäisestä merkistä (sisältää vain a:ta ja mahd. b:tä) ja v:n olt. vähintään 1-merkkinen. Kaksi jakovaihtoehtoa:

  1. $u = a^i$, $v = a^{k-i}b^j$, $w = b^{l-j}c^{k+l}$, missä $i \neq k$ tai $j > 0$. Tällöin 0-kertaisesti pumpattu jono on $uv^0w{=}a^i b^{l-j} c^{k+l}$, joka kuuluu kieleen vain jos, $i = k$ ja $j = 0$, mikä ei mahd.
  2. $u = a^k b^i$, $v = b^j$, $w = b^{l-i-j}c^{k+l}$, missä $j \geq 1$. $uv^0w = a^k b^i b^{l-i-j} c^{k+l} = a^k b^{l-j} c^{k+l}$, joka kuuluu kieleen vain, jos $j = 0$, mikä ei mahd.

  $\Rightarrow$ Kieli ei siis säännöllinen.

- Esim. 3: Väite: Kieli $L_{pr} = \{1^p | p \text{ on alkuluku}\}$ ei ole säännöllinen.
  Tod.: Vastaoletus: L_{pr} on säännöllinen $\Rightarrow \exists$ jokin $n \geq 1$, jota pitempiä L_{pr}:n merkkijonoja voidaan pumpata.
  Valitaan $x = 1^p$, jollakin alkuluvulla $p \geq n + 2$. Pumppauslemman mukaan $x = uvw, |uv| \leq n, m = |v| \geq 1$. Nyt $|uv^{p-m}w| = |uw| + (p - m)|v| = p-m+(p-m)m = (m+1)(p-m)$. Tämä on yhdistetty luku, koska $m+1 \geq 2$ ja $p - m \geq n + 2 - m \geq 2$ (sillä $m = |v| \leq |uv| \leq n$). Siis $uv^{p-m}w \notin L_{pr}$. Ristiriita. □

Näin näytät pitkää nenää Pumppauslemmalle

(heuristisia ohjeita PL-todistuksiin)

1. Mieti annettua kieltä: mikä ehto täytyy rikkoa, jotta merkkijono ei kuulu kieleen?

   - ehto voi koskea esim. tiettyjen merkkien lukumäärien keskinäistä suhdetta (esim. $L_1 = \{a^k b^m c^m | k, m = 0, 1, 2, ...\}$, $L_2 = \{a^m b^{2m} | m = 0, 1, 2, ...\}$)
   - tai liittyä alkulukuihin, joiden jono on epäsäännöllinen $L_3 = \{a^{p_i} | p_i$ on i:s alkuluku$\}$
   - tai sanan osia: esim. sanan alku- ja loppuosa riippuvat jotenkin toisistaan $L_4 = \{ww^R | w \in \Sigma^*, w^R$ on w takaperin kirjoitettuna$\}$, $L_5 = \{ww | w \in \Sigma^*\}$

2. Mieti, mikä olisi yksinkertaisin merkkijono, jossa esiintyvät eheysehdon osapuolet. Joskus kielessä on todistuksen kannalta täysin turhia (säännöllisiä) osia, esim. L_1:ssä a:n lukumäärällä ei ole mitään väliä – voidaan valita merkkijono $b^m c^m$.

   - jos ehdon osapuolten välissä on tuollainen säännöllinen osa, se saattaa olla tarpeen osapuolien erottamiseen toisistaan, esim. $L_6 = \{a^m b^k a^m | m, k = 0, 1, 2, ...\}$:ssä tarvitaan ainakin yksi b erottamaan alku- ja loppu-a:t. Valitaan esim. $a^m b a^m$.
   - jos kielen alku- ja loppuosa riippuvat jotenkin toisistaan, mutta muuten ne saavat olla mitä tahansa, riittää erottaa alku- ja loppuosa toisistaan, esim. kielen L_5 kohdalla voidaan valita $a^m b a^m b$ tai $b a^m b a^m$.

3. Valitse nyt se mystinen n siten, että eheysehdon toinen osapuoli kuuluu ensimmäiseen n:ään merkkiin ja sitä päästään pumppaamaan. Toinen tavoite on, että merkkijonon mahdollisia jakoja osiin uvw olisi mahdollisimman vähän (tämä ei ole kuitenkaan yhtä kriittistä, säästää vain työtä).

   - esim. $x = a^m b^m$. Kannattaa valita $n = m$, jolloin osat uv kuuluvat a^m:ään. Tällöin pääsemme pumppaamaan a:ta ja eheysehto rikkoontuu. Huom! Voimme valita myös $n = 2m$, jolloin tulee lisää töitä: Pumpattava osa voi koostua vain a:sta, vain b:stä tai molemmista (muotoa $a^i b^j$). Viimeisessä tapauksessa pumppaus uv^2w sotkee asiat ($a^i b^j a^i b^i$ ei kuulu kieleen).

4. Testaa nyt **kaikki** Pumppauslemman mukaiset jaot $x = uvw, |uv| \leq n$ ja $v \neq \epsilon$. Jokaisella jaolla kokeile pumppausta i:n arvoilla 0,2,3,... kunnes löytyy sellainen i, että $uv^i w \notin A$.

- tavallisesti arvo $i = 0$ tai viimeistään $i = 2$ tuottaa halutun tuloksen.

5. Jos onnistuit rikkomaan eheysehdon kaikilla jaoilla uvw, voit huudahtaa voitonriemuisena: "Heureka!" Onneksi olkoon!

Miksi pumppauslemmaa käytetään näin?

Esitetään PL loogisena lauseena:

$R(A) \Rightarrow \exists n\ \forall x P(x)\ \exists uvw\ Q(x,u,v,w)\ \forall i\ S(u,v,w,i)$, missä

$P(x)$ on lyhenne ehdolle $x \in A \wedge |x| \geq n$
$Q(x,u,v,w)$ ehdolle $x = uvw \wedge |uv| \leq n| \wedge |v| \geq 1$, ja
$S(u,v,w,i)$ ehdolle $uv^i w \in A$.

Käytetään kontrapositiosääntöä:

$A \Rightarrow B \equiv \neg B \Rightarrow \neg A$

Toisaalta tiedämme:

$\neg\exists x P(x) \equiv \forall x \neg P(x)$
$\neg\forall x P(x) \equiv \exists x \neg P(x)$.

Käännetään Pumppauslemma näiden avulla ja se on yhä tosi:

$\neg(\exists n\ \forall x P(x)\ \exists uvw\ Q(x,u,v,w)\ \forall i\ S(u,v,w,i)) \Rightarrow \neg R(A)$
$\equiv \forall n\ \neg(\forall x P(x)\ \exists uvw\ Q(x,u,v,w)\ \forall i\ S(u,v,w,i)) \Rightarrow \neg R(A)$
$\equiv \forall n\ \exists x P(x) \neg(\exists uvw\ Q(x,u,v,w))\ \forall i\ S(u,v,w,i)) \Rightarrow \neg R(A)$
$\equiv \forall n\ \exists x P(x)\ \forall uvw\ Q(x,u,v,w)\ \neg(\forall i\ S(u,v,w,i)) \Rightarrow \neg R(A)$
$\equiv \forall n \exists x P(x)\ \forall uvw\ Q(x,u,v,w)\ \exists i\ \neg(S(u,v,w,i)) \Rightarrow \neg R(A)$.

Siis: Jos kaikilla n on joku x $|x| \geq n$, kaikilla uvw $|uv| \leq n$, $|v| \geq 1$, on olemassa i siten että $uv^i w \notin A$, niin A ei ole säännöllinen.

Punahilkkapeli

Ahneet Peikot ovat niin innostuneita piparibisneksestään, että pyörittävät pipariautomaatteja ympäri metsää. Kuitenkin myös vanha sairas Mummo kaipaisi kipeästi pipareita. Punahilkka kumppaneineen keksii seuraavan ratkaisun: jos äiti alkaa leipoa epäsäännöllisen kielen mukaisia piparijonoja, eivät Peikot pysty rakentamaan koko kieltä tunnistavaa automaattia - eivätkä siis pysty saamaan kaikkia pipareita.

Pelin idea

Punahilkkapelissä on kaksi joukkuetta, joista toiseen kuuluvat Punahilkka, Mummo ja Susi ja toiseen Peikot. Punahilkan tehtävänä on viedä Mummolle niin paljon pipareita, kuin hän pystyy, ilman että Peikot syövät niitä matkalla. Peikot näet syövät vain säännöllisiä piparijonoja (esim. (sydän $*$ tähti sydän$*$)).

Äiti leipoo jatkuvasti uusia pipareita, jotka voivat olla tähden tai sydämenmuotoisia. Punahilkan ja Mummon pitäisi keksiä sellainen piparikieli, jonka mukaiset piparijonot Punahilkka pystyy kuljettamaan Peikkojen ohitse. Kiltti Susi auttaa Punahilkkaa ja Mummoa. Esim. kielen (tähti^{n}sydän^n) $n \geq 1$ piparijonot (sanat) eivät kelpaa Peikoille.

Saatte korttipakan, joista kukin kuvaa joko säännöllisen tai epäsäännöllisen piparikielen.

Pelin kulku

Punahilkan puoli: Valitkaa satunnainen kortti pakasta.
Peikot: Yrittäkää osoittaa, että kyseinen piparikieli on säännöllinen. Jos onnistutte, niin saatte kaikki kielen kuvaamat piparit.
Punahilkan puoli: Jos Peikot eivät onnistuneet, niin voitte yrittää kuljetusta. Osoittakaa, että piparikieli ei ole säännöllinen. Jos onnistutte, saatte vietyä piparit onnellisesti Mummolle.

Peikot ja Punahilkan puoli kinastelevat piparikielestä niin kauan, kunnes joko Peikot saavat sen tai Punahilkka saa vietyä sen läpi. Tämän jälkeen Punahilkka kumppaineineen yrittää seuraavaa piparikieltä.

Se puoli, joka on lopuksi kerännyt eniten piparikieliä, voittaa pelin. Vaihtakaa välillä puolia!

3.9 Harjoituksia säännöllisyyden tutkmisesta

1. Ovatko seuraavat kielet säännöllisiä? Perustele!

   (a) $\{w|w$ on a:sta ja b:stä koostuva merkkijono, jonka pituus on $3\}$

   (b) $\{ww|w \in \{a,b\}^*\}$

   (c) $\{w^*|w$:ssä yhtä monta 1:ä ja 0:aa$\}$

   (d) $\{w|w$:ssä kaksi kertaa niin monta 0:aa kuin 1:ä$\}$

2. Mitä tapahtuu, jos yrität todistaa Pumppauslemmalla epäsäännölliseksi kielen, joka onkin oikeasti säännöllinen? Tarkastele esimerkiksi kieliä $\emptyset$, $\{aa, bb\}$, $\{ab^*a^*b\}$.

3. Mitä vikaa on seuraavissa Pumppauslemmatodistuksissa?

   a) Olkoon $L = \{(aa)^i(bb)^j|i,j \geq 0\}$.
   Väite: L on epäsäännöllinen.
   Todistus: Olkoon $x = (aa)^n(bb)^k = a^{2n}b^{2n}$, $|x| = 4n$. Nyt x voidaan jakaa osiin uvw vain yhdellä tavalla: $u = a^i$, $v = a^j$, $j \geq 1$, $w = a^{2n-i-j}b^{2n}$. Nyt $uv^0w = a^{2n-j}b^{2n} \notin L$, joten L on epäsäännöllinen.

   b) Olkoon $L = \{c^ra^kb^k|r \geq 1, k \geq 0\} \cup \{a^kb^l|k,l \geq 0\}$.
   Väite: L on säännöllinen.
   Todistus: $L = L_1 \cup L_2$, missä $L_1 = \{c^ra^kb^k|r \geq 1, k \geq 0\}$ ja $L_2 = \{a^kb^l|k,l \geq 0\}$. Jokaiselle $x \in L$ pätee $x \in L1$ tai $x \in L2$. Tutkitaan erikseen kumpaakin tapausta:
   1) Jos $x \in L_1$, valitaan $x = c^na^kb^k$. $|x| = n + 2k > n$. x voidaan jakaa osiin uvw vain yhdellä tavalla:
   $u = c^i$, $v = c^j$, $j \geq 1$, $w = c^{n-i-j}a^kb^k$. Nyt $uv^kw \in L$ kaikilla $k = 0, 1, 2, ...$, eli voimme aina pumpata x:ää.
   2) Jos $x \in L_2$, valitaan $x = a^nb^l$, $|x| = n+l$. x voidaan jakaa osiin vain yhdellä tavalla: $u = a^i$, $v = a^j$, $j \geq 1$, $w = a^{n-i-j}b^l$. $uv^kw \in L$ kaikilla $k = 0, 1, 2, ...$
   $\Rightarrow L$ on säännöllinen.

4. Tiedetään, että yleisessä tapauksessa ongelma $REG(L)$ (ts. onko annettu kieli L säännöllinen vai ei) on ratkeamaton. Mistä tämä johtuu? Voisitko silti laatia ohjelman, joka auttaisi ihmistä osoittamaan Pumppauslemmalla, että kieli on epäsäännöllinen?

5. Vahvempi Pumppauslemman variaatio antaa sekä riittävät että välttämättömät ehdot kielen säännöllisyydelle:

Pumppauslemma 2: Kieli $A \in \Sigma^*$ on säännöllinen, jos ja vain jos on olemassa vakio $n \geq 1$ s.e. kaikilla $x \in \Sigma^*$, jos $|x| \geq n$ on olemassa u, v, w s.e. $x = uvw$ ja $|v| \geq 1$ ja kaikilla $i \geq 0$ ja kaikilla $y \in \Sigma^*$ $xy \in A$ jos ja vain jos $uv^iwy \in A$.

Voiko tämän pohjalta laatia ohjelman, joka ratkaisee minkä tahansa kielen A osalta, onko A säännnöllinen vai ei?

6. Osoita pumppauslemmalla, että seuraavat kielet eivät ole säännöllisiä:

   (a) $\{a^n b^n c^k | n, k = 0, 1, ...\}$

   (b) $\{a^n b^k c^k | n, k = 0, 1, ...\}$

   (c) $\{a^n b^n a^m b^m | n, k = 0, 1, ...\}$

7. Merkitään w^R:llä merkkijonoa w takaperin kirjoitettuna (so. jos $w = a_1 a_2 \ldots a_n$, niin $w^R = a_n \ldots a_2 a_1$). Merkkijono on palindromi, jos $w = w^R$ (esimerkiksi "isorikassikakissakirosi"). Tarkastellaan aakkoston $\{a, b\}$ palindromien muodostamaa kieltä $L_{pal} = \{ww^R | w \in \{a, b\}^*\}$

   Osoita Pumppauslemmalla, että palindromikieli L_{pal} on epäsäännöllinen!

3.10 Ekskursio: Säännöllisten kielten sovelluksia

3.10.1 Hahmontunnistus

Tavoite: halutaan löytää tekstistä y hahmo x. Edellä tutustuimme jo yhteen tapaan ratkaista tämä ongelma äärellisten automaattien avulla. Laadimme äärellisen automaatin, joka tunnistaa hahmon x. Jos haluamme lisäksi sanan x sijainnin merkkijonossa, täytyy lisätä laskuri, joka pitää kirjaa luetuista merkeistä (siirtymistä), kunnes hahmo tunnistettu.

Äärellisiä automaatteja voidaan käyttää toisellakin tapaa hahmon tunnistukseen. Muodostetaan äärellinen automaatti tekstin y suffikseista. Automaatin kuhunkin kaareen (siirtymään) liittyy sekä kirjain että numero. Kunkin siirtymän yhteydessä katenoidaan kaaren kirjain luettujen merkkien jonoon. Lisäksi kasvatetaan sijaintilaskuria siirtymän ilmoittamalla askelmäärällä. Tuloksena saadaan pari (p, i), missä p on suffiksi i kertoo sen sijainnin (montako merkkiä luettava tekstin alusta, jotta kyseinen suffiksi löytyy).

Tällainen *suffiksiautomaatti* toimii itse asiassa tekstin hakemistorakenteena, jonka avulla voidaan etsiä mikä tahansa merkkijono tekstistä.

Muistathan myös, että säännöllisillä lausekkeilla voidaan helposti kuvata etsittäviä hahmoja, esim. tiedon haussa. Vastaava hakukone voidaan toteuttaa äärellisenä automaattina.

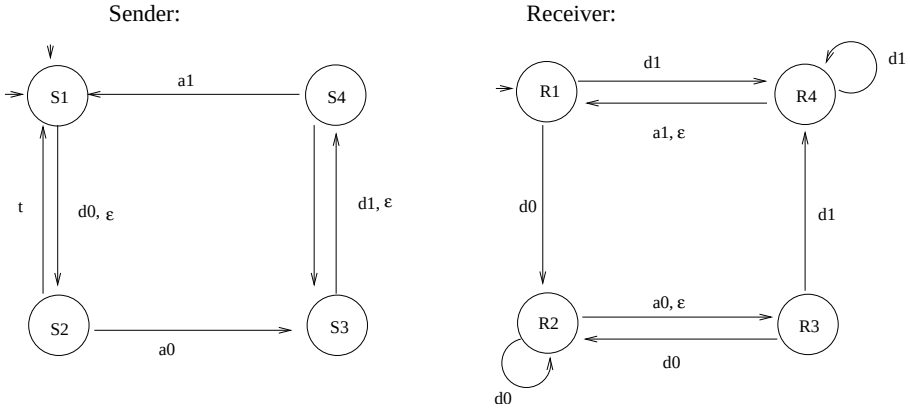
3.10.2 Viestinvälitysprotokollat äärellisinä automaatteina

Myös erilaisia protokollia voidaan kuvata äärellisinä automaatteina. Tälla tavalla voidaan esimerkiksi verifioida, että jokin viestinvälitysprotokolla toimii oikein.

Tarkastellaan yksikertaista AB (alternating bit)-protokollaa, jossa lähettäjäprosessi S ja vastaanottajaprosessi R kommunikoivat *vuorosuuntaisen kanavan* kautta. Vuorosuuntaisessa kanavassa viestejä saa lähettää vain yhteen suuntaan kerrallaan. Kanava voi hukata tai vääristää sanomia, mutta viestien järjestys säilyy eikä kanava monista sanomia.

Koska kanavassa on kerrallaan vain yksi viesti, riittää viestien esittämiseen kaksi bittiä, 1 ja 0. Lähettäjä yrittää esnsin lähettää viestin $d0$ ja jos se saa siitä kuittauksen $a0$, se siirtyy lähettämään viestiä $d1$. Jos se saa $d1$:stäkin kuittauksen $a1$, se voi lähettää taas viestin, jota merkitään $d0$:lla jne. Lähettäjä ei kuitenkaan lähetä uutta viestiä ennen kuin se on saanut edellisestä kuittauksen. Jos kuittausta ei kuulu tietyn ajan t kuluessa se lähettää viestin uudestaan. Vastaanottaja puolestaan lukee kaikki kanavasta tulevat viestit ja lähettää niistä kuittauksen, mahdollisesti useaankin kertaan (jos sama viesti tulee useita kertoja).

Lähettäjän ja vastaanottajan toiminta voidaan kuvata seuraavanlaisina epädeterministisinä (ϵ-) automaatteina.



3.10.3 Leksikaalinen analyysi

Leksikaalinen analyysi on kääntäjän osa, joka jakaa lähdekoodin loogisesti yhteenkuuluviin osiin (tokens). Tällaisia osia voivat olla esimerkiksi varatut sanat, tunnisteet, ym. Leksikaalinen analysaattori voidaan toteuttaa esim. UNIX-komentojen `lex` ja `flex` (edellisen GNU-versio) avulla. Lex- ja flex-komento saavat syötteenään listan säännöllisistä lausekkeista sekä ohjeet, mitä tehdä vastaavan osan kohdalla, ja tuottavat tämän pohjalta leksikaalisen analysaattorin kyseisille säännöille.

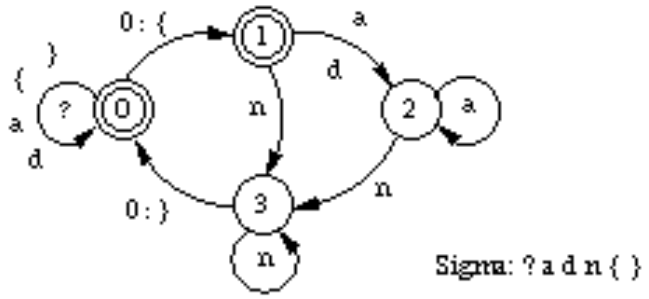
Syöte voisi olla esimerkiksi seuraavanlainen:

else	{return(ELSE); }
[A-Za-z][A-Za-z0-9]*	{code to enter the found identifier in the symbol table; return(ID); }
>=	{return(GE);}
=	{return(EQ); }
...	

Huom! Tällaisessa määrittelyssä varattu sana ”else” täsmää myös tunnisteen kuvaukseen. Ongelma voidaan ratkaista antamalla säännöille prioriteetijärjestys: valitaan ensimmäinen lauseke, johon merkkijono täsmää.

3.10.4 Luonnollisen kielen esiprosessointi

Luonnollisen kielen ymmärtäminen tai tiedon eristäminen tekstistä (joka ei pyrikään koko tekstin ymmärtämiseen, vaan etsimään siitä ainoastaan haluttua tietoa) on hyvin haastava tehtävä, johon säännölliset kielet eivät riitä. Niitä voidaan kuitenkin käyttää apuna tekstin esiprosessoinnissa samaan tapaan kuin ohjelmointikielen leksikaalisessa analyysissa.
Eräs lähestymistapa on muodostaa *transduktori* (engl. *transducer*), joka on eräänlainen uudelleenkirjoitava äärellinen automaatti. Laajennetaan esimerkiksi lausekeautomaattia siten, että jokaisen siirtymän yhteydessä automaatti sekä lukee merkkijonon että korvaa sen jollain toisella merkkijonolla (tai tyhjällä merkkijonolla). Tällaisella transduktorilla voidaan esimerkiksi lukea pois turhat täytesanat (kuten englannin artikkelit), tunnistaa prepositiot (esim. korvataan symbolilla PRE) tai keksiä heuristiikka erisnimien tunnistamiseen (korvataan sovitulla symbolilla). Alla esimerkki transduktorista, joka pyrkii erottamaan (englanninkielisestä) lauseesta substantiivilmaukset (noun phrases), jotka ovat muotoa $[(d)a * n+]$, missä d on artikkeli, a on adjektiivi ja n on substantiivi. (Huom! Sanaluokat on siis jo tunnistettu jollain tapaa.)



Kuva 3.14: Transduktori, joka kirjoittaa sulut kaikkien tunnistamiensa substantiivilausekkeiden ympärille.

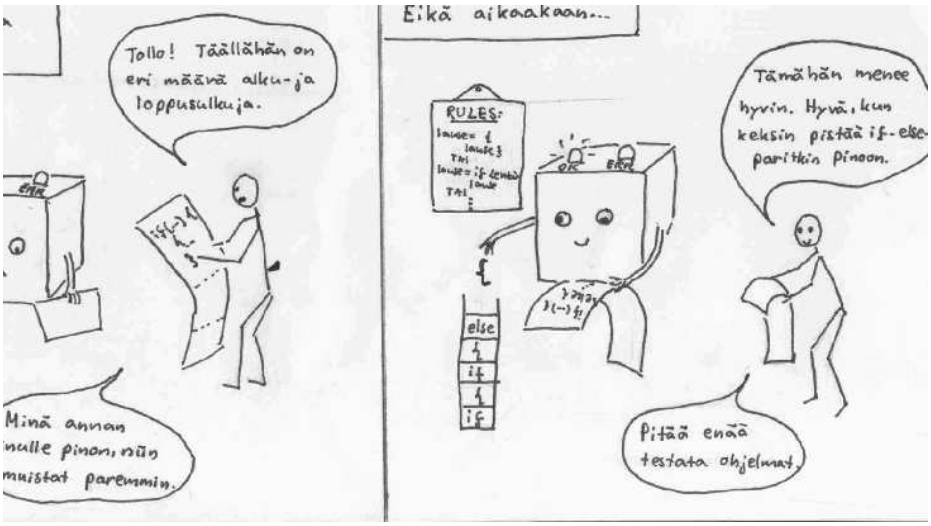
Harjoituksia

1. Mihin käytännön sovelluksiin sinä voisit käyttää äärellisiä automaatteja tai säännöllisiä lausekkeita? Mainitse ainakin kolme sovellusta!

2. Mitä osaat sanoa säännöllisten kielten luokkaan kuuluvien ongelmien aikavaativuudesta? Entä tilavaativuudesta? (Vihje: Mieti äärellisen automaatin pohjalta laadittua ohjelmaa.) Onko mitään eroa, onko vastaava automaatti deterministinen vai epädeterministinen?

Luku 4

Kontekstittomat kielet ja pinoautomaatit



- Kuten edellä nähtiin, esim. tasapainoisten sulkulausekkeiden muodostama kieli
  $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ ja
  if-else-parien muodostama kieli
  $L_{if-else} = \{\text{if}^k\text{else}^l | l \leq k\}$
  eivät ole säännöllisiä (Pumppauslemma)

- Mutta näitä kieliä voi kuvata *kontekstittomilla kieliopeilla*
- Kuten edellä säännöllistiä kieliä tunnistava kone oli äärellinen automaatti, kontekstitonta kieltä tunnistava kone on *pinoautomaatti*

4.1 Kontekstittomat kieliopit ja kielet

- Esim. sulkulausekekielellä ysinkertainen rekursiivinen kuvaus: Merk. $S =$ "mielivaltainen tasapainoinen sulkumerkkijono". Tällöin S on tasapainoinen sulkumerkkijono, jos

  (i) $S = \epsilon$ tai
  (ii) S on muotoa (S'), missä S' on tasapainoinen sulkumerkkijono

- Toinen kuvaustapa: seuraavat *muunnossäännöt tuottavat* täsmälleen kielen L_{match} merkkijonot symbolista S:

  (i) $S \to \epsilon$,
  (ii) $S \to (S)$

- Esim. merkkijonon $((()))$ tuottaminen:

$$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow (((\epsilon))) = ((()))$$

- Kontekstiton kielioppi on muunnossysteemi, jossa kuvattavat merkkijonot tuotetaan korvaamalla erityisiä muuttuja- t. *välikesymboleita* annettujen sääntöjen mukaan yksi kerrallaan, symbolia ympäröivän merkkijonon rakenteesta riippumatta

- Merkinnällä

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

  lyhennetään

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

- Esim. Yksinkertainen kielioppi tietyille aritmeettisille lausekkeille:

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

  Esim. lausekkeen $(a + a) * a$ tuottaminen:

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

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

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

missä

- V on kieliopin aakkosto;
- $\Sigma \subseteq V$ on kieliopin *päätemerkkien* joukko; sen komplementti $N = V \setminus \Sigma$ on kieliopin *välikemerkkien* t. *-symbolien* joukko;
- $P \subseteq N \times V^*$ on kieliopin *sääntöjen* t. *produktioiden* joukko;
- $S \in N$ on kieliopin *lähtösymboli*

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

- Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa suoraan* merkkijonon $\gamma' \in V^*$ kieliopissa G, merk.
$$\gamma \underset{G}{\Rightarrow} \gamma'$$
jos voidaan kirjoittaa $\gamma = \alpha A \beta$, $\gamma' = \alpha\omega\beta$ $(\alpha, \beta, \omega \in V^*, A \in N)$, ja kieliopissa G on produktio $A \to \omega$
- Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa* merkkijonon $\gamma' \in V^*$ kieliopissa G, merk.
$$\gamma \underset{G}{\Rightarrow}^* \gamma'$$
jos on olemassa jono V:n merkkijonoja $\gamma_0, \gamma_1, \ldots, \gamma_n$ $(n \geq 0)$, siten että
$$\gamma = \gamma_0 \underset{G}{\Rightarrow} \gamma_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \gamma_n = \gamma'$$
- Erikoistapaus: $n = 0$, $\gamma \underset{G}{\Rightarrow}^* \gamma$ millä tahansa $\gamma \in V^*$
- Merkkijono $\gamma \in V^*$ on kieliopin G *lausejohdos*, jos on $S \underset{G}{\Rightarrow}^* \gamma$

- G:n *lause* on pelkästään päätemerkeistä koostuva G:n lausejohdos $x \in \Sigma^*$
- Kieliopin G *tuottama* t. *kuvaama kieli* koostuu G:n lauseista:

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

Määritelmä: Formaali kieli $L \subseteq \Sigma^*$ on *kontekstiton*, jos se voidaan tuottaa jollakin kontekstittomalla kieliopilla

- Esim. 1 Tasapainoisten sulkujonojen muodostaman kielen $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ tuottaa kielioppi

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

- Esim. 2 Aiemmin tarkisteltujen yksinkertaisten aritmeettisten lausekkeiden muodostaman kielen L_{expr} tuottaa kielioppi

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

missä

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

Kieliopissa voidaan johtaa esim. seuraavat lausejohdokset:
$E \underset{G}{\Rightarrow} E+T \underset{G}{\Rightarrow} T+T \underset{G}{\Rightarrow} T*F+T \underset{G}{\Rightarrow} F*F+T \underset{G}{\Rightarrow} a*F+T \underset{G}{\Rightarrow} a*(E)+T \underset{G}{\Rightarrow} a*(T)+T \underset{G}{\Rightarrow} a*(F)+T \underset{G}{\Rightarrow} a*(a)+T \underset{G}{\Rightarrow} a*(a)+F \underset{G}{\Rightarrow} a*(a)+a$. Lopputulos $a*a+a$ on kieliopin lause. Kieliopin tuottamaan kieleen kuuluvat kaikki a:n summa- ja tulolausekkeet, joissa voi lisäksi esiintyä sulkuja.

Toinen kielioppi kielen L_{expr} tuottamiseen on

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

missä

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

- Esim. 3 ($\sim$ Orponen teht.44) Tarkastellaan suomen kielen virkettä, joka koostuu yksinkertaisesta päälauseesta + 0 tai useammasta sisäkkäisestä relatiivilauseesta:

$$L_{rel} = \{subj(\text{ joka } pred\ attr\ obj)^*\ pred\ attr\ obj\}$$

Tällaisia virkkeitä voidaan tuottaa esim. seuraavilla kontekstittoman kieliopin G_{rel} säännöillä:

(Esitetään yksinkertaisuuden vuoksi säännöt merkkijonoilla)

VIRKE $\rightarrow$ SUBJ SL PRED ATTR OBJ
SL $\rightarrow$ joka PRED ATTR OBJ SL | ϵ
SUBJ $\rightarrow$ poika | tyttö | jänis | susi | peikko
PRED $\rightarrow$ pelkäsi | metsästi
ATTR $\rightarrow$ suurta | pientä | vihaista | hirmuista | arkaa
OBJ $\rightarrow$ poikaa | tyttöä | jänistä | sutta | peikkoa

Kieleen kuuluvat mm. seuraavat virkkeet:

$VIRKE \underset{G}{\Rightarrow}^*$ poika joka metsästi sutta joka pelkäsi peikkoa joka pelkäsi suurta tyttöä pelkäsi hirmuista jänistä

$VIRKE \underset{G}{\Rightarrow}^*$ tyttö joka metsästi arkaa poikaa pelkäsi vihaista jänistä

Vakiintuneita merkintätapoja

- 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: $\alpha, \beta, \gamma, \ldots, \omega$

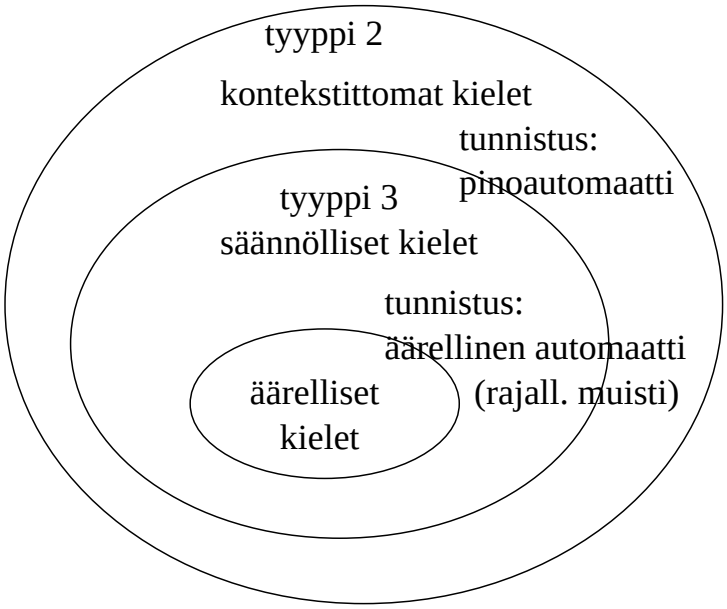
- Kielioppi esitetään usein pelkkänä sääntöjoukkona:

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

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

4.2 Säännölliset kielet ja kontekstittomat kieliopit

- Kontekstittomilla kieliopeilla voidaan kuvata joitakin ei-säännöllisiä kieliä (esimerkiksi kielet L_{match} ja L_{expr})
- Osoitetaan, että myös kaikki säännölliset kielet voidaan kuvata kontekstittomilla kieliopeilla
- Kontekstittomat kielet ovat siten säännollisten kielten aito yliluokka



4.2.1 Oikealle ja vasemmalle lineaariset kieliopit

Määritelmä: Kontekstiton kielioppi on *oikealle lineaarinen*, jos sen kaikki produktiot ovat muotoa $A \to \epsilon$ tai $A \to aB$, ja *vasemmalle lineaarinen*, jos sen kaikki produktiot ovat muotoa $A \to \epsilon$ tai $A \to Ba$

Osoittautuu, että sekä vasemmalle että oikealle lineaarisilla kieliopeilla voidaan tuottaa täsmälleen säännölliset kielet. $\Rightarrow$ lineaarisia kielioppeja nimitetään myös yhteisesti *säännöllisiksi* kieliopeiksi.

Muunnosten äärellinen automaatti $\leftrightarrow$ oikealle lineaarinen kielioppi ideat ovat seuraavat (Tarkastelemme tässä vain oikealle lineaarisia kielioppeja, joille muunnos on hieman yksinkertaisempi):

Äärellisestä automaatista kielioppi

1. Luo kielioppiin yksi välikesymboli A_q kutakin automaatin tilaa q kohden. Lähtötilaa q_0 vastaa kieliopin lähtösymboli A_{q_0}.
2. Kutakin automaatin siirtymää $q \xrightarrow{a} q'$ kohden luo kielioppiin sääntö $A_q \to A_{q'}$.
3. Kutakin automaatin lopputilaa $q \in F$ kohden luo kielioppiin sääntö $A_q \to \epsilon$

Lineaarinen kieliopista äärellinen automaatti

1. Luo automaattiin tila q_A kutakin kieliopin välikesymbolia A kohden. Lähtösymbolia S vastaa automaatin alkutila q_S.
2. Kutakin kieliopin sääntöä $A \to aB$ kohden luo automaattiin siirtymä tilasta q_A tilaan q_B merkillä a.
3. Mikäli välikkeellä A on sääntö $A \to \epsilon$, on q_A lopputila.

Nyt voimme todistaa seuraavat lauseet:

Lause: 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. Muodostetaan kielioppi G_M, jolla on $L(G_M) = L(M) = L$.

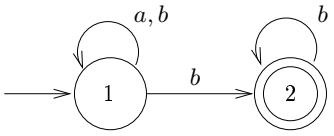
- G_M:n pääteaakkosto = M:n syöteaakkosto Σ

- G_M:n välikeaakkostoon otetaan yksi välike A_q kutakin M:n tilaa q kohden.
- G_M:n lähtösymboli on A_{q_0}
- G_M:n produktiot vastaavat M:n siirtymiä:

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

  (ii) kutakin M:n siirtymää $q \xrightarrow{a} q'$ (so. $q' \in \delta(q,a)$) kohden kielioppiin otetaan produktio $A_q \to aA_{q'}$
- Tarkastetaan konstruktion oikeellisuus:
  - Merk. A_q:sta tuotettavien päätejonojen joukkoa
    $$L(A_q) = \{x \in \Sigma^* \mid A_q \underset{G_M}{\Rightarrow}{}^* x\}$$
  - Induktiolla merkkijonon x pituuden suhteen voidaan osoittaa, että kaikilla q on
    $$x \in L(A_q) \;\Leftrightarrow\; (q,x) \underset{M}{\vdash}{}^* (q_f, \epsilon) \text{ jollakin } q_f \in F$$
  - Erityisesti on siis
    $$\begin{aligned} L(G_M) = L(A_{q_0}) &= \{x \in \Sigma^* \mid (q_0,x) \underset{M}{\vdash}{}^* (q_f,\epsilon) \\ &\qquad \text{jollakin } q_f \in F\} \\ &= L(M) = L. \quad \square \end{aligned}$$

Esim.



Kuva 4.1: Yksinkertainen äärellinen automaatti.

Automaattia vastaava kielioppi on:

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

Lause: 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:

- M_G:n tilat vastaavat G:n välikkeitä:

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

- M_G:n alkutila on lähtösymbolia S vastaava tila q_S
- M_G:n syöteaakkosto on 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:

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

- Konstruktion oikeellisuus voidaan jälleen tarkastaa induktiolla G:n tuottamien ja M_G:n hyväksymien merkkijonojen pituuden suhteen. □

4.2.2 Esimerkkejä

Ohessa esimerkkejä säännöllisistä lausekkeista ja vastaavista kontekstittomista kieliopeista. Huom! Kannattaa piirtää vastaavat äärelliset automaatit!

1. Lauseke: a^*
   Kielioppi:$S \to aS|\epsilon$
2. Lauseke: $a^+ = aa^*$
   Kielioppi: $S \to aS|a$
3. Lauseke: $(aa)^*$
   Kielioppi: $S \to aSa|\epsilon$
   (merkkijono koostuu vain parillisesta määrästä a:ta)
4. Lauseke: $(b^*ab^*ab^*)^*$
   Kielioppi:
   $S \to BaBaBS|\epsilon$
   $B \to bB|\epsilon$
   (merkkijono sisältää parillisen näärän a:ta, lisäksi saa olla b:tä missä tahansa)
5. Lauseke: $(0 \bigcup 1 \bigcup ... \bigcup 9)(0 \bigcup 1 \bigcup ... \bigcup 9)^*$
   Kielioppi:
   $S \to DN$

$N \rightarrow DN|\epsilon$
$D \rightarrow 0|1|...|9$
(vähintään yhdestä digitistä koostuva numero)

4.3 *Ekskursio: Kasvikieliopit (Lindenmayer Systems)

Esimerkki 1 Olkoon symbolit $\Sigma = \{SIEMEN, SIRKKALEHDET, LEHDET, VARSI, NUPPU, PUNKUKKA, SINKUKKA\}$. Kukkaketo voidaan nyt esittää Σ^*:n merkkijonona. Kukin kasvi kehittyy seuraavien sääntöjen mukaan:

SIEMEN → SIRKKALEHDET
SIRKKALEHDET → LEHDET | VARSI
LEHDET → VARSI
VARSI → NUPPU | LEHDET
NUPPU → PUNKUKKA | SINKUKKA
PUNKUKKA → SIEMEN | SIEMEN SIEMEN |ϵ
SINKUKKA → SIEMEN | SIEMEN SIEMEN |ϵ

missä ϵ kuvaa kasvin kuolemaa (se katoaa kylvämättä edes siemeniä).

Kyseessä on *kontekstiton kielioppi*, sillä säännön seuraus määräytyy ainoastaan edeltävän symbolin perusteella (jos useita vaihtoehoisia seurauksia, niin valitaan jokin niistä).

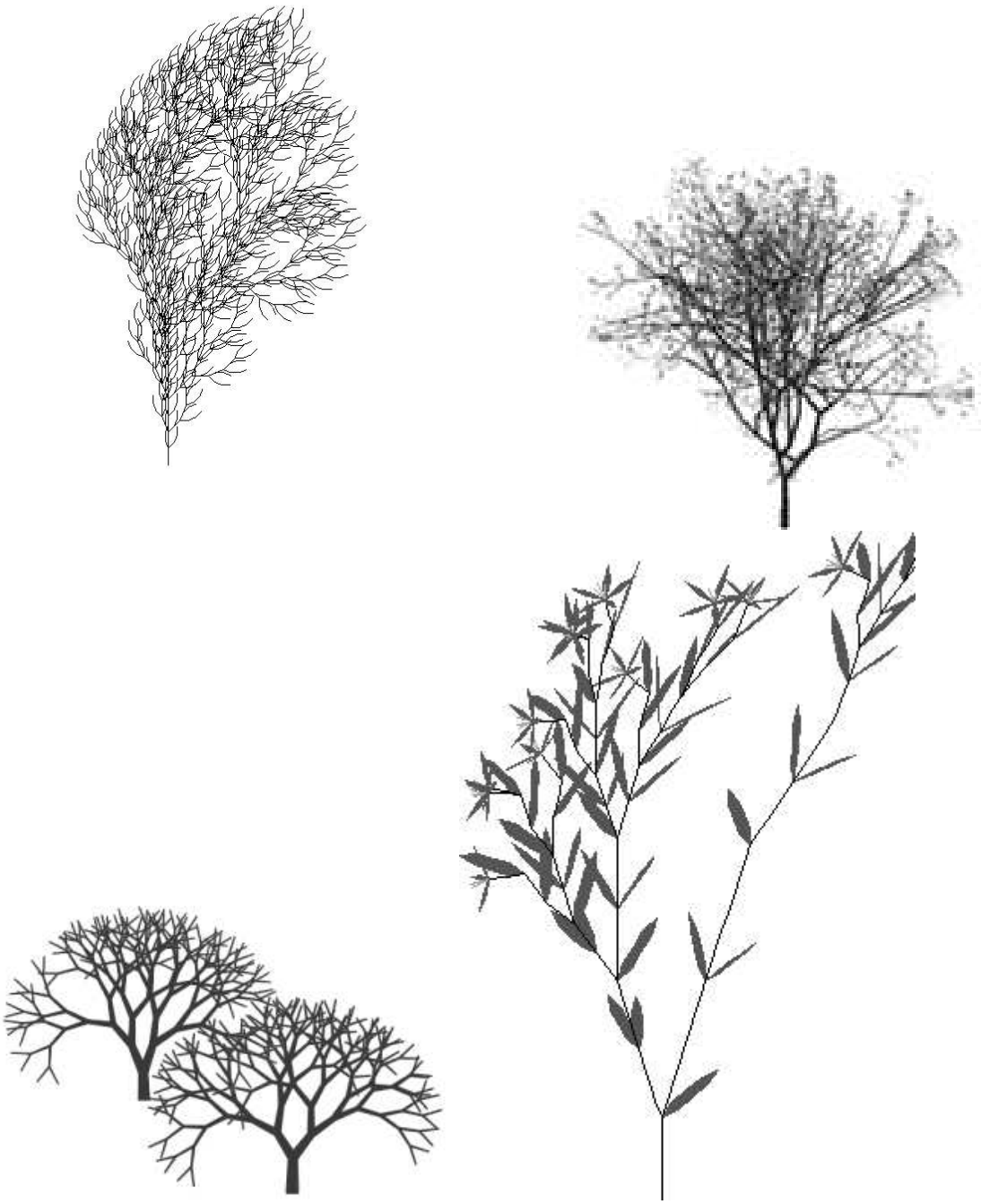
Esimerkki 2 Laajennetaan symbolijoukkoa: $\Sigma_2 = \Sigma \bigcup \{PAIVA, YO\}$ ja määritellään uudet säännöt:

SIEMEN YO→ SIRKKALEHDET YO
SIRKKALEHDET PAIVA→ LEHDET PAIVA| VARSI PAIVA
LEHDET PAIVA→ VARSI PAIVA
VARSI PAIVA→ NUPPU PAIVA| LEHDET PAIVA
NUPPU YO→ PUNKUKKA YO| SINKUKKA YO
PUNKUKKA → SIEMEN | SIEMEN SIEMEN |ϵ
SINKUKKA → SIEMEN | SIEMEN SIEMEN |ϵ
PAIVA → YO
YO → PAIVA

Nyt kukat kasvavat vain päivisin, mutta siemenet itävät ja kukat aukeavat vain öisin. Tämä kielioppi on *kontekstillinen*, sillä myös symbolin konteksti (ympäristötekijät) vaikuttavat säännön seuraukseen.

- L-järjestelmät $\sim$ yksinkertaistettu abstrakti kielioppi, jonka tuottama kieli kuvaa kasvien (tms. elollisten organismien) kasvua ja kehitystä
- alkujaan botanisti Aristid Lindenmayerin kehittämä matemaattinen malli, jolla voi ennustaa kasvien kasvua
- sittemmin käytetty keinotekoisten organismien luomiseen ja kaikkeen mahdoliseen!
- L-järjestelmä tuottaa vain kieliopin mukaisen merkkijonon $\rightarrow$ voidaan tulkita graafisena kuvana
- muistuttavat fraktaaleja, mutta eivät välttämättä fraktaaleja
- Yleinen idea: jokaisella aika-askelella merkkijonon symboli lavennetaan naapurisymbolien ja ehtoihin sopivan säännön perusteella (voi pysyä ennallaankin) $\leftrightarrow$ *s*oluautomaatti, jossa kukin hilan (vektorin, 2- tai useampiulotteisen taulukon) solu on äärellinen automaatti
- Leikkaavat Chomskyn hierarkian kaikkia luokkia:
  - voivat noudattaa säännöllisen kielen sääntöjä, ts. ovat muotoa $A \rightarrow aB$ tai $A \rightarrow Ba$ (muunnoksen tuloksena yksi päätesymboli ja yksi välikesymboli) tai $A \rightarrow \epsilon$ ("solun kuolema", symboli katoaa) tai
  - kontekstitonta kielioppia, ts. säännöt muotoa $A \rightarrow V^*$ (ts. muunnetaan vain yhtä symbolia kerrallaan, riippumatta sen naapureista, muunnoksen tuloksena voi tulla mitä tahansa välike- ja päätesymboleja, myös ϵ) tai
  - kontekstillista kielioppia, ts. säännöt muotoa $\alpha A\beta \rightarrow \alpha\omega\beta$ (ts. muunnos vain tietyssä "kontekstissa", kun naapureina α ja β)
  - rajoittamatonta kielioppia, ts. säännöt muotoa $\omega \rightarrow \omega'$ (ts. mikä tahansa merkkijono voi muuntua miksi tahansa merkkijonoksi, merkkijonot voivat sisältää niin pääte- kuin välikesymboleja)

- esim. erään levämuodon kasvun mallinnus:
  - alkuvaiheessa voi olla kahdenlaisia soluja, merk. A ja B (”aksioomat”)
  - säännöt:
    $A \to AB$
    $B \to A$
  - esim. $AB \Rightarrow ABA \Rightarrow ABAAB \Rightarrow ABAABABA$ (3 aika-askelta)
- yksinkertaisimmat L-järjestelmät ns. DOL-järjestelmiä (kontekstiton ja deterministinen L-järjestelmä)
- esim. 2: säännöt:
  $A \to CB$
  $B \to A$
  $C \to DA$
  $D \to C$
  - lähtökohta: mikä tahansa aakkoston sallittu ”siemen” (lähtösymboli)
  - esim. $A \Rightarrow CB \Rightarrow DAA \Rightarrow CCBCB \Rightarrow DADAADAA$
- esim. 3: Puurakenteen muodostaminen:
  $A \to C[B]D$
  $B \to A$
  $C \to C$
  $D \to C(E)A$
  $E \to D$
  - tulkitaan hakasulut haarana vasemmalle ja tavalliset sulut haarana oikealle
  - esim. $A \Rightarrow C[B]D \Rightarrow C[A]C(E)A$ (2 aika-askelta)
  - puuna:



Kuva 4.2: Esimerkkejä kasvikieliopeilla tuotetuista kuvista.

- Sovellus (WH): muodostetaan kukkaketoa grafiikkaohjelmalla, joka noudattaa seuraavia sääntöjä:
  $SIEMEN \rightarrow SIRKKALEHDET|VVARSI|OVARSI|KVARSI$
  $VVARSI \rightarrow VLEHTI|VNUPPU$ (satunnainen väri)
  $OVARSI \rightarrow OLEHTI|ONUPPU$
  $VNUPPU \rightarrow KUKKA$
  $ONUPPU \rightarrow KUKKA$
  $KNUPPU \rightarrow KUKKA$
  $KUKKA \rightarrow SIRKKALEHDET$
  $KVARSI \rightarrow VVARSI|OVARSI|KNUPPU|SIRKKALEHDET$
  $|VKIELONVARSI|OKIELONVARSI|HAVU$
  $SIRKKALEHTI \rightarrow VVARSI|OVARSI|KVARSI|LEHDET$
  $VKIELOVARSI \rightarrow VKIELO$
  $OKIELOVARSI \rightarrow OKIELO$
  $VKIELO \rightarrow VMARJAT$
  $OKIELO \rightarrow OMARJAT$

- Kuvaruutu on jaettu soluihin, joita käydään läpi jossain järjestyksessä ja piirretään ko. paikkaan (tai naapurisoluihin) symbolin mahdollinen seuraaja

Kirjallisuutta

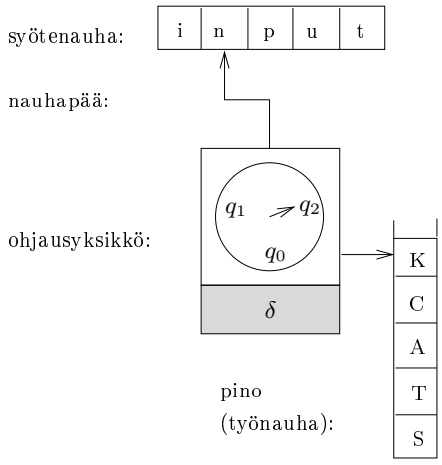
- Prusinkiewicz & Lindenmayer: The Algorithmic Beaty of Plants
- Prusinkiewicz & Hanan: Lindenmayer Systems, Fractals, and Plants
- Rozenberg & Salomaa (toim.): Lindenmayer Systems. Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology

4.4 Pinoautomaatit

Kuten jo edellä huomasimme, eivät äärelliset automaatit kykene pitämään kirjaa lukemistaan syötemerkeistä. Esimerkiksi sulkulausekkeen tasapainoisuuden tutkimiseksi meidän täytyisi käyttää jonkinlaista laskuria, jota kasvatetaan aina kun vastaan tulee alkusulku '(' ja vähennetään, kun kohdataan loppusulku ')'. Pinoautomaatit ratkaisevat tämän ongelman tarjoamalla käyttöömme pinon, johon voi tallettaa symboleja myöhempää tutkimusta varten.

Pinoautomaatit eivät ole ohjelmoijalle yhtä kullanarvoisia apulaisia kuin äärelliset automaatit. Kontekstittonmien kielioppien pohjalta voi näet suoraan laatia tehokkaita ohjelmia. Tästä syystä käsittelemme pinoautomaatteja vain lyhyesti, ikään kuin johdatuksena Turingin koneisiin. Pinoautomaatit ovat kuitenkin hyödyllinen abstrakti laskennan malli kontekstittomien kielioppien tutkimuksessa ja toisinaan voi olla helpompi laatia ongelman kuvaava (epädeterminististen) pinoautomaatti, kuin laatia vastaava kielioppi.

4.4.1 Määritelmä



Kuva 4.3: Pinoautomaatti.

- Pinoautomaatti on äärellinen automaatti, johon on lisätty yksi mielivaltaisen kokoinen pino työnauhaksi
- Ts. automaatti voi lukea ja kirjoittaa vain työnauhan toista päätä (pinon huippua), ja päästäkseen lukemaan aiemmin kirjoittamiaan merkkejä sen täytyy pyyhkiä viimeisin merkki pois

- Työnauha antaa automaatille "muistin", jonka avulla voidaan välttää äärellisen automaatin (joitakin) rajoituksia. (Huom! Tällainen pinomuisti ei riitä esimerkiksi sen tarkistamiseen, onko merkkijonossa yhtä monta a:ta, b:tä ja c:tä.)

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

- Siirtymäfunktion
$$\delta(q, \sigma, \gamma) = \{(q_1, \gamma_1), \ldots, (q_k, \gamma_k)\}$$
tulkinta: Ollessaan tilassa q ja lukiessaan syötemerkin σ ja pinomerkin γ automaatti voi siirtyä johonkin tiloista $q_1, \ldots, q_k$ ja korvata pinon päällimmäisen merkin jollakin merkeistä $\gamma_1, \ldots, \gamma_k$.
  1. Jos $\sigma = \epsilon$, niin automaatti tekee siirtymän syötemerkkiä lukematta;
  2. Jos $\gamma = \epsilon$, niin automaatti ei lue pinomerkkiä, mutta pistää uuden merkin pinon päälle ($push(\gamma_i)$);
  3. Jos $\gamma \neq \epsilon$ ja $\gamma_i = \epsilon$, niin luetaan pinosta γ, mutta ei panna uutta merkkiä pinoon ($pop(\gamma)$)
  4. Jos sekä $\gamma \neq \epsilon$ että $\gamma_i \neq \epsilon$, niin sekä luetaan pinon päällimäinen merkki γ että kirjoitetaan pinoon γ_i (ts. $pop(\gamma), push(\gamma_i)$)
- Pinoautomaatit ovat siis yleisessä tapauksessa *epädeterministisiä*
- Automaatin tilanne on kolmikko $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$

- *Alkutilanne syötteellä* x on kolmikko (q_0, x, ϵ)
- Tilanne $(q, w, \alpha) \sim$ 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'), merk.

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

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

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

- Tilanne (q, w, α) *johtaa tilanteeseen* (q', w', α'), merk.

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

  jos on olemassa tilannejono $(q_0, w_0, \alpha_0), \ldots, (q_n, w_n, \alpha_n)$, $n \geq 0$, siten että
  $(q, w, \alpha) = (q_0, w_0, \alpha_0) \underset{M}{\vdash} \cdots \underset{M}{\vdash} (q_n, w_n, \alpha_n) = (q', w', \alpha')$

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

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

  (jos se syötteen loppuessa on jossakin hyväksyvässä lopputilassa); muuten M *hylkää* x:n

- Automaatin M *tunnistama kieli* on

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

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

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

missä

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

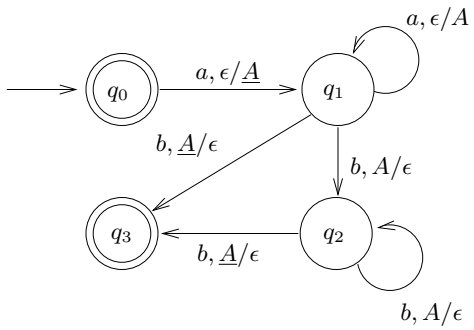
Automaatin ideana on pitää kirjaa vastaantulleista a-kirjaimista, pistämällä pinoon $\underline{A}$ tai A, ja vastaavasti lukea niitä pinosta jokaisella vastaantulevalla b-kirjaimella. Merkkiä $\underline{A}$ käytetään A:n sijasta ensimmäisen a:n kohdalla merkitsemään pinon pohjaa, jotta tiuedetään, milloin pino on tyhjä.

Esimerkiksi syötteellä $aabb$ automaatti M toimii seuraavasti:

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

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

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



Kuva 4.4: Kielen $\{a^k b^k \mid k \geq 0\}$ tunnistava pinoautomaatti, jossa pinoaakkosto on $\{A,\underline{A}\}$.

Tässä on käytetty tilasiirtymäkaaviotyyppinen esitys:

- Siirtymäehto muotoa $a_i, \gamma/\gamma'$, missä
  - automaatti lukee merkkijonosta merkin a_i ja
  - pinosta merkin γ sekä
  - kirjoittaa pinoon merkin γ'
- ”Push”-operaatio: ei lue pinosta merkkiä, mutta kirjoittaa uuden merkin γ' pinon päälle: $a_i, \epsilon/\gamma'$
- ”Pop”-operaatio: lukee pinon päällimmäisen merkin γ, mutta ei kirjoita mitään pinoon: $a_i, \gamma/\epsilon$
- jos syötemerkki $a_i = \epsilon$, niin siirtymä syötemerkkiä lukematta – voi silti lukea tai kirjoittaa pinomerkkejä!

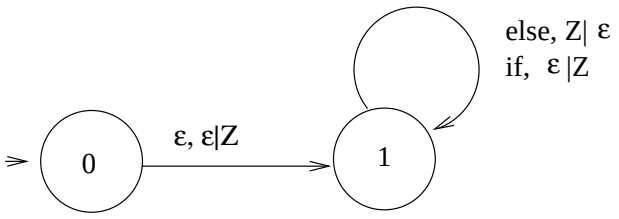
4.4.2 *Pinoautomaatin muunnos: ”tyhjän pinon automaatti”

Määritellään seuraavan todistuksen avuksi pinoautomaatin muunnos, jolla ei ole erillistä hyväksyvää lopputilaa, vaan merkkijono hyväksytään missä tahansa tilassa, kunhan vain **pino on tyhjä syötteen loppuessa**. (Tavallinen pinoautomaattihan hyväksyy syötteen, jos se on syötteen loppuessa hyväksyvässä lopputilassa, olipa pinossa mitä tahansa.) Tällaista pinoautomaattia kutsutaan usein ”tyhjän pinon automaatiksi” (*empty stach automaton*, *null stack automaton*).

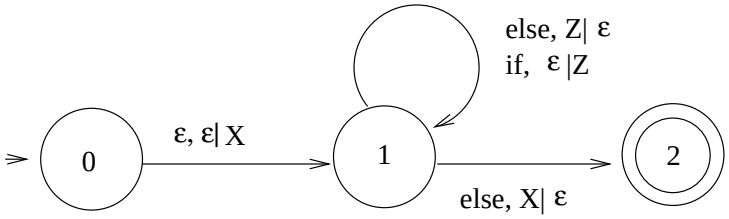
Tyhjän pinon automaatit ovat hyödyllinen apulainen, sillä ne tunnistavat *täsmälleen* samat kielet kuin tavalliset pinoautomaatit. (Ks. todistus esim. Hopcroft, Motwani, Ullman: luku 6.2)

Tavallinen pinoautomaatti hyväksyy x:n, jos	Tyhjän pinon automaatti hyväksyy x:n jos
$(q_0, x, \epsilon) \vdash_M^* (q_f, \epsilon, \alpha)$ $q_f \in F$ $\alpha \in \Gamma^*$	$(q_0, x, \epsilon) \vdash_M^* (q, \epsilon, \epsilon)$ $q \in Q$

Esim. Muodostetaan pinoautomaatti, joka tunnistaa virheellisen ohjelmakoodin, jossa *if-else*-parien lukumäärät eivät täsmää. Automaatti hyväksyy syötteen, jos siinä on enemmän *else*:jä kuin *if*:ejä.



Kuva 4.5: Tyhjän pinon automaatti $if - else$ -virheen haivaitsemiseksi.



Kuva 4.6: Saman kielen tunnistava lopputilallinen automaatti.

Lause: Kieli on kontekstiton, jos ja vain jos se voidaan tunnistaa pinoautomaatilla.
**Todistuksen idea:*
Käytetään hyväksi tyhjän pinon automaatteja. Jos väite pätee tyhjän pinon automaatille M, se pätee myös ekvivalentille lopputilalliselle pinoautomaatille M' ($L(M) = L(M')$).

1)"$\Rightarrow$" Jos kieliopin G kuvaama kieli $L(G)$ on kontekstiton, niin on olemassa tyhjän pinon pinoautomaatti M, joka hyväksyy kielen $L(M) = L(G)$.

- Muodostetaan pinoautomaatti, jossa on kaksi tilaa: alkutila S ja hyväksyvä lopputila F.
- Aluksi pistetään pinoon erikoissymboli X (pinon pohja) ja siirrytään tilaan F.
- Tämän jälkeen on kaksi vaihtoehtoista askelta:
  1. Jos pinon päällä on välikesymboli C ja kieliopissa sääntö $C \rightarrow w$, niin korvaa C w:llä pinossa (ts. $pop(C), push(w)$).
  2. Jos pinon päällä on päätesymboli (merk. a), niin lue seuraava syötemerkki (merk. a'). Jos $a = a'$, niin lue pinon huippu pois ($pop(a)$).
- Automaatin siirtymät ovat siis:
  1. $\delta(S, \epsilon, \epsilon) = \{(F, X)\}$
  2. $\delta(F, \epsilon, C) = \{(F, w)\}$
  3. $\delta(F, a, a) = \{(F, \epsilon)\}$
- Intuitiivisesti: Automaatin laskenta simuloi syötejonon *vasenta johtoa*. Jokainen johtoaskel toteutetaan pistämällä pinoon säännön $S \rightarrow w$ oikea puoli w. Johtoaskelten välissä automaatti lukee päätesymbolit pinosta ja vertaa niitä syötteeseen. Kun säännön vasemman puoleinen välike C havaitaan pinossa, automaatti suorittaa seuraavan askeleen.
- Formaalissa todistuksessa pitäisi vielä osoittaa (induktiolla), että $w \in L(G) \Leftrightarrow w \in L(M)$.

2)"$\Leftarrow$" Jos olemassa tyhjän pinon pinoautomaatti M, joka hyväksyy kielen $L(M)$, niin on olemassa kontekstiton kielioppi G, joka kuvaa kielen $L(G) = L(M)$.

- Muodostetaan kielioppi, jonka jokainen välikesymboli vastaa "tapahtumaa", jossa

1. luetaan jokin symboli X pinosta (voi koostua useasta siirtymästä)
2. muutetaan tapahtumaa edeltävä tila p tapahtuman lopussa vallitsevaksi tilaksi q (symbolin X lukeminen pinosta tapahtuu siis siirtymäpolun $p - -- > q$ aikana)

- Merkitään tällaista tapahtumaa $[pXq]$:lla.
- Kieliopin muuttujat: alkusymboli S ja kaikki tapahtumasymbolit $[pXq]$, missä $p, q \in Q$ ja $X \in \Gamma$ (pinosymboli).
- Kieliopin säännöt:
  1. Jokaista tilaa $p \in Q$ kohden siirtymä $S \rightarrow [q_0 Z_0 p]$. (ts. symboli $[q_0 Z_0 p]$ tuottaa kaikki ne merkkijonot w, jotka aiheuttavat symbolin Z_0 nostamisen pinosta kuljettaessa tilasta q_0 tilaan p eli $(q_0, w, Z_0) \underset{M}{\vdash^*} (p, \epsilon, \epsilon)$.)
  2. Olkoon siirtymäfunktion $\delta(q, a, X)$ arvojoukossa pari $(r, Y_1 Y_2 ... Y_k)$, missä a on päätesymboli tai ϵ, ja $k \geq 0$. Silloin jokaista tilajoukkoa $r_1, r_2, ..., r_k$ kohti kieliopissa on sääntö

     $$[qXr_k] \rightarrow a[rY_1r_1][r_1Y_2r_2]...[r_{k-1}Y_kr_k].$$

     (t.s. yksi tapa lukea pinosta symboli X ja siirtyä tilasta q tilaan r_k on lukea ensin a, sitten käyttää syötettä Y_1:n poppaamiseksi siirtyen samalla tilaan r_1, käyttää syötettä Y_2:n poppaamiseksi siirtyen samalla tilaan r_2, jne.)
- Formaalissa todistuksessa pitäisi vielä osoittaa (induktiolla), että $[qXp] \underset{G}{\Rightarrow^*} w$ jos ja vain jos $(q, w, X) \underset{M}{\vdash^*} (p, \epsilon, \epsilon)$. Tällöin $S \underset{G}{\Rightarrow^*} w$ joss $[q_0 Z_0 p] \underset{G}{\Rightarrow^*} w$ jollain p joss $(q, w, Z_0) \underset{M}{\vdash^*} (p, \epsilon, \epsilon)$ eli $L(G) = L(M)$.

□

4.4.3 Deterministiset ja epädeterministiset pinoautomaatit

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

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

- Huom: *Epädeterministiset pinoautomaatit ovat aidosti vahvempia kuin deterministiset*! (vrt. äärelliset automaatit)

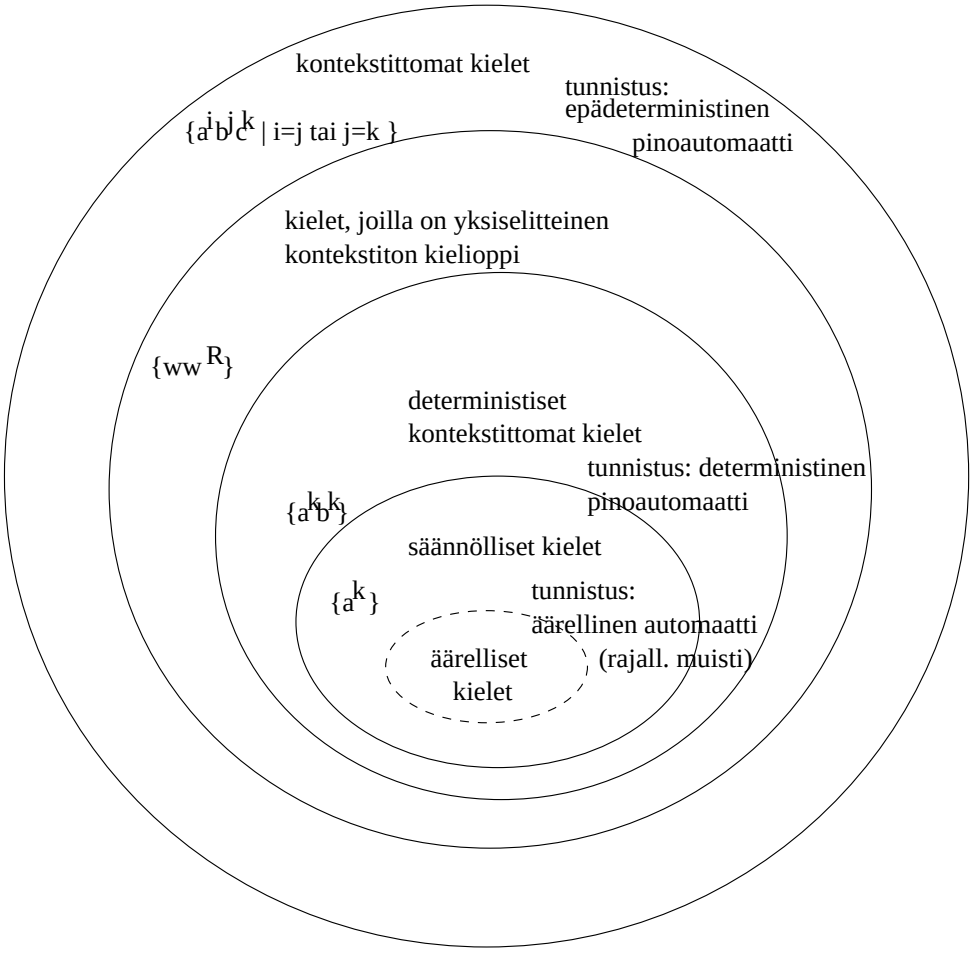
- Esim. kieli $L_1 = \{ww^R \mid w \in \{a, b\}^*\}$ voidaan tunnistaa epädeterministisellä, mutta ei deterministisellä pinoautomaatilla (todistus sivuutetaan).

- Kontekstiton kieli on *deterministinen*, jos se voidaan tunnistaa jollakin deterministisellä pinoautomaatilla, muuten se on *epädeterministinen*.

- Esim. em. kieli L_1 ja kieli $L_2 = \{a^n b^m c^k | n \neq m \text{ tai } m \neq k\}$ ovat epädeterministisiä.

- Deterministiset kielet ovat tärkeä kieliluokka, sillä siihen kuuluvat kielet voidaan jäsentää oleellisesti tehokkaammin kuin yleiset, mahdollisesti epädeterministisen automaatin vaativat kontekstittomat kielet

- Kielten keksinäisiä suhteita kuvaa seuraava kuva:



4.5 Kielioppien jäsennysongelma

Ratkaistava tehtävä:

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

- Esim. Kuuluuko virke "jänis joka pelkäsi arkaa peikkoa metsästi suurta sutta" kieleen L_{rel}?
- Ratkaisumenetelmä: *jäsennysalgoritmi*
- Useita vaihtoehtoisia menetelmiä, erityisesti kun G on jotain rajoitettua (käytännössä esiintyvää) muotoa.

- Tarkastellaan ensin jäsentämiseen liittyviä peruskäsitteitä

4.5.1 Johdot

- Olk. merkkijono $\gamma \in V^*$ kieliopin $G = (V, \Sigma, P, S)$ lausejohdos
- γ:n *johdoksi* G:ssä kutsutaan lähtösymbolista S merkkijonoon γ johtavaa suorien johtojen jonoa

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

- Johdon *pituus* on siihen kuuluvien suorien johtojen määrä (edellä n)
- Esim. lauseen $a + a$ johtoja kieliopissa G_{expr}:

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

- Johto $\gamma \Rightarrow^* \gamma'$ on
  1. *vasen johto*, merk.
  $$\gamma \underset{\text{lm}}{\Rightarrow}{}^* \gamma',$$
  jos kussakin johtoaskelessa on produktiota sovellettu merkkijonon vasemmanpuoleisimpaan välikkeeseen (edellä johto (i))
  2. *oikea johto*, merk.
  $$\gamma \underset{\text{rm}}{\Rightarrow}{}^* \gamma',$$
  jos — ” — oikeanpuoleiseen välikkeeseen (edellä (iii))
- Suoria vasempia ja oikeita johtoaskelia merkitään $\gamma \underset{\text{lm}}{\Rightarrow} \gamma'$ ja $\gamma \underset{\text{rm}}{\Rightarrow} \gamma'$

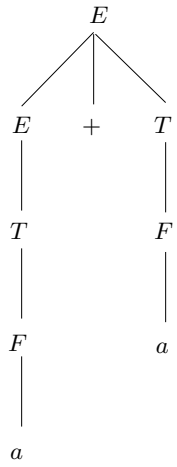
4.5.2 Jäsennyspuut

- Suurin osa vaihtoehtoisten johtotapojen eroista muodostuu vain välikkeiden laventamisesta eri järjestyksessä (esim. em. johdot (i) – (iii) ovat kaikki “pohjimmiltaan” samanlaisia)

- Lausejohdoksen *jäsennyspuu* (syntaksipuu, johtopuu) (engl. parse tree, syntax tree, derivation tree) = esitystapa, jossa nämä epäoleelliset erot on abstrahoitu pois

- Jäsennyspuu kertoo ainoastaan, *miten* välikkeet on lavennettu, ei *missä järjestyksessä* lavennukset on tehty

- Esim. kaikkia kolmea em. johtoa vastaa sama jäsennyspuu:



Kuva 4.7: Lauseen $a + a$ jäsennyspuu kieliopissa G_{expr}.

Määritelmä: Olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Kieliopin G mukainen *jäsennyspuu* on järjestetty puu, jolla on seuraavat ominaisuudet:

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

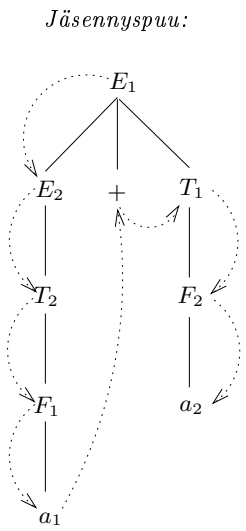
(ii) jos A on puun jonkin sisäsolmun nimi, ja $X_1, \ldots, X_k$ ovat sen jälkeläisten nimet järjestyksessä, niin
$A \to X_1 \ldots X_k$ on G:n produktio

- Jäsennyspuun τ *tuotos* = merkkijono, joka saadaan liittämällä yhteen sen lehtisolmujen nimet esijärjestyksessä ("vasemmalta oikealle")

4.5.3 Jäsennyspuun muodostaminen

- Johtoa

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



Solmut esijärjestyksessä:

$E_1E_2T_2F_1a_1 + T_1F_2a_2$

Vasen johto:

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

Kuva 4.8: Vasemman johdon muodostaminen jäsennyspuusta.

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

- Ts. jos τ on jotakin johtoa $S \Rightarrow^* \gamma$ vastaava jäsennyspuu, niin τ:n tuotos on γ
- Johdon muodostaminen: Olk. τ kieliopin G mukainen jäsennyspuu, jonka tuotos on päätemerkkijono x.
  - 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
  - Oikea johto saadaan käymällä puu läpi käänteisessä esijärjestyksessä ("ylhäältä alas, oikealta vasemmalle")
- Jos muodostetaan annetusta vasemmasta (oikeasta) johdosta $S \underset{\text{lm}}{\Rightarrow}^* x$ ($S \underset{\text{rm}}{\Rightarrow}^* x$) ensin jäsennyspuu em. tavalla, ja sitten jäsennyspuusta vasen (oikea) johto, saadaan takaisin alkuperäinen johto

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

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

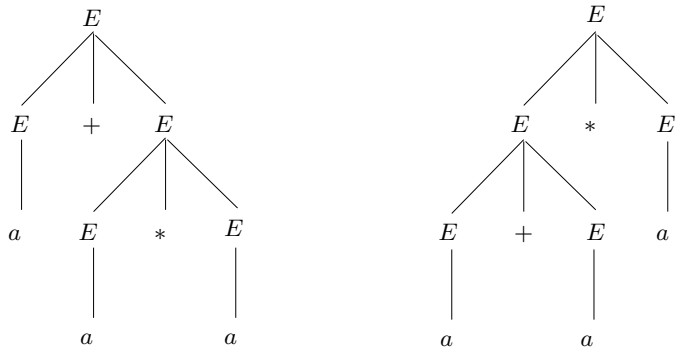
(ii) jokaista G:n mukaista jäsennyspuuta τ, jonka tuotos on päätemerkkijono x, vastaavat yksikäsitteiset vasen ja oikea johto $S \underset{\text{lm}}{\Rightarrow}^* x$ ja $S \underset{\text{rm}}{\Rightarrow}^* x$

Seuraus: Jokaisella G:n lauseella on vasen ja oikea johto

- Ts. kontekstittoman kieliopin tuottamien lauseiden jäsennyspuut, vasemmat ja oikeat johdot vastaavat yksikäsitteisesti toisiaan
- Jäsennysongelman ratkaisuksi katsotaan usein päätösongelman "Onko $x \in L(G)$?" ratkaisemisen lisäksi jonkin näistä jäsennysesityksistä tuottaminen x:lle

4.5.4 Kieliopin moniselitteisyys

- Lauseella voi olla kieliopissa useita jäsennyksiä (esim. lause $a+a*a$ kieliopissa G'_{expr})



Kuva 4.9: Lauseen $a + a * a$ kaksi erilaista jäsennystä.

- Kontekstiton kielioppi G on *moniselitteinen*, jos jollakin G:n lauseella x on kaksi erilaista G:n mukaista jäsennyspuuta
- Muuten kielioppi on *yksiselitteinen*
- Kontekstiton kieli, jonka tuottavat kieliopit ovat kaikki moniselitteisiä, on *luonnostaan moniselitteinen*

- Esimerkkejä:
  - 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}
  - Kieli $\{a^i b^j c^k \mid i = j \text{ tai } j = k\}$ on luonnostaan moniselitteinen (todistus sivuutetaan)
- Huom! Luonnostaan moniselitteiset kielet voidaan tunnistaa vain epädeterministisillä pinoautomaateilla. Itse asiassa luonnostaan moniselitteiset kielet ovat epädeterminististen kielten aito osaluokka – os siis olemassa myös yksiselitteisiä kieliä, jotka vaativat epädeterministisen pinoautomaatin.
- Huom. 2: Ohjelmointikielten kielioppien (syntaksin) tulee olla yksiselitteisiä, jotta ohjelma voitaisiin kääntää yksikselitteisesti toimivaksi ohjelmaksi.

4.6 Rekursiivisesti etenevä jäsentäminen

4.6.1 LL(1)-kielioppi

LL(1)-kielet ovat suppea mutta hyödyllinen kieliluokka. Niille voidaan helposti laatia rekursiivisesti etenevä jäsentäjä, joka lukee merkkijonoa vasemmalta oikealle ja joka asekeleella simuloi annetun kieliopin sääntöjä. Koska *LL(1)-kieliopissa* jokainen sääntö on yksikäsitteisesti määritelty, kun seuraava syötemerkki tunnetaan, voidaan merkkijonon vasemman johdon tuottava jäsennys suorittaa tietokoneohjelmana. (Nimitys LL(1)-kielioppi tuleekin sanoista "left to right scan, producing left parse with 1 symbol lookahead".)

- Tarkastellaan seuraavaa kielioppia G:

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

- Muokataan G:stä välikkeen E produktiot "tekijöimällä" ekvivalentti kielioppi G':

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

- nyt G':n lauseille voidaan helposti muodostaa vasemmat johdot suoraan lähtösymbolista E alkaen: Jäsennyksen joka vaiheessa tavoitteena olevan lauseen *seuraava merkki* määrää yksikäsitteisesti sen, mikä produktio valitaan lavennettavana olevaan välikkeeseen

4.6.2 LL(1)-kieliopin jäsennysohjelma

- Jäsennysohjelma muodostuu välikkeitä vastaavista rekursiivisista funktioista
- Esim. G':n mukainen jäsentäjä pseudokoodina:

```
function E
begin
T;
E';
end

function E'
begin
if (next == '+') then
begin
    tee jotakin;
    next = getnext;
    E;
end
else
    begin
        if (next == '-') then
            begin
                tee jotakin;
                next = getnext;
                E;
            end
        else tee jotakin;
    end
end

function T
begin
if (next == 'a') then
```

```
    begin
        tee jotakin;
        next = getnext;
        return;
    end
else
    begin
        if (next == '(') then
            begin
                tee jotakin;
                next = getnext;
                E;
                if (next ≠ ')')
                        ERROR;
                next = getnext;
            end
        else ERROR;
    end
end
```

PÄÄOHJELMASSA:
next = *getnext*;
E;

Esim. syötejonon `a-(a+a)` jäsennys, kun käsky ”tee jotakin’ tulostaa produktiot:

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

Tulostus vastaa vasenta johtoa:

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

4.6.3 LL(1)-kielioppien yleinen muoto

LL(1)-kieliopeissa sallitaan rajoitetussa määrin myös

- Produktioita, joiden oikeat puolet alkavat välikkeellä, sekä
- *Tyhjentyviä* välikkeitä A, joilla $A \Rightarrow^* \epsilon$

Esim. kielen $a^*b \cup c^*d$ tuottava kielioppi:

$$\begin{array}{lcll} S & \rightarrow & Ab & | \quad Cd \\ A & \rightarrow & aA & | \quad \epsilon \\ C & \rightarrow & cC & | \quad \epsilon. \end{array}$$

Kielioppi on LL(1), vaikka ensimmäiseksi sovellettavaa produktiota ei voikaan päätellä pelkästään S:n produktioiden perusteella

4.6.4 Apukäsite: päätemerkkijoukot FIRST ja FOLLOW

Määritellään annetun kieliopin $G = (V, \Sigma, P, S)$ välikkeisiin liittyvät päätemerkkijoukot:

- FIRST(A) = A:sta johdettavien päätejonojen 1. merkit + ϵ, jos A tyjentyvä
- FOLLOW(A) = ne päätemerkit, jotka voivat seurata A:ta jossain G:n lausejohdoksessa + ϵ, jos A voi sijaita lausejohdoksen lopussa
- Formaalisti:

$$\begin{aligned}
\text{FIRST}(A) &= \{a \in \Sigma \mid A \Rightarrow^* ax \text{ jollakin } x \in \Sigma^*\} \\
&\quad \cup \{\epsilon \mid A \Rightarrow^* \epsilon\}; \\
\text{FOLLOW}(A) &= \{a \in \Sigma \mid S \Rightarrow^* \alpha A a \beta \text{ joillakin } \alpha, \beta \in V^*\} \\
&\quad \cup \{\epsilon \mid S \Rightarrow^* \alpha A \text{ jollakin } \alpha \in V^*\}.
\end{aligned}$$

FIRST-joukkojen määritelmä laajennetaan mielivaltaisille merkkijonoille, ja edelleen merkkijonojoukkoihin:

$$\text{FIRST}(\epsilon) = \{\epsilon\};$$

$$\text{FIRST}(a) = \{a\} \qquad \text{kaikilla } a \in \Sigma;$$

$$\text{FIRST}(X_1 \ldots X_k) =$$

$$\begin{cases}
\text{FIRST}(X_1) \cup \ldots \cup \text{FIRST}(X_i) - \{\epsilon\}, \\
\qquad \text{jos } \epsilon \in \text{FIRST}(X_1), \ldots, \text{FIRST}(X_{i-1}), \\
\qquad \epsilon \notin \text{FIRST}(X_i); \\
\text{FIRST}(X_1) \cup \ldots \cup \text{FIRST}(X_k), \\
\qquad \text{jos } \epsilon \in \text{FIRST}(X_i) \text{ kaikilla } i = 1, \ldots, k;
\end{cases}$$

Määritelmä: 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:

$$\begin{aligned}
&\text{FIRST}(\{\omega_1\}\text{FOLLOW}(A)) \\
&\quad \cap \quad \text{FIRST}(\{\omega_2\}\text{FOLLOW}(A)) \quad = \quad \emptyset.
\end{aligned}$$

LL(1)-testin idea: Mitkä ovat 1. päätemerkit, jos valitaan sääntö $A \to \omega_1$? Jos sääntö $A \to \omega_2$ tuottaa samoja 1. päätemerkkejä, ei valinta ole yksinkäsitteinen!

$FIRST(\{\omega_1\}FOLLOW(A))$**:n laskeminen**

- Tutki ensin ω_1:n 1. päätemerkit
- jos voi olla $\omega_1 = \epsilon$, niin tutki myös $FOLLOW(A)$:n 1. päätemerkit
- ts. jos johdos on $\alpha A \beta$ ja $A = \epsilon$, niin seuraava päätemerkki on β:n 1. päätemerkki

Esim.1 Tarkastellaan seuraava kissakielen tuottavaa kielioppia:

$S \rightarrow Amiu|mauSmau|\epsilon$
$A \rightarrow purr|\epsilon$

$FIRST(\{Amiu\}FOLLOW(S)) = FIRST(\{purrmiu, \epsilon miu\}FOLLOW(S)) = \{p, m\}$

$FIRST(\{mauSmau\}FOLLOW(S)) = FIRST(\{mau\}FOLLOW(S)) = \{m\}$

$FIRST(\{\epsilon\}FOLLOW(S)) = FIRST(\{\epsilon\}\{\epsilon, mau\}) = \{\epsilon, m\}$

Nyt kaikki säännöt voivat tuottaa seuraavana merkkinä m:n, joten ei ole yksikäsitteistä, mikä niistä valitaan. Kielioppi ei siis ole LL(1)-muodossa.

Esim.2 Onko seuraava kielioppi LL(1)-muodossa?

$S \rightarrow Ab|Cd$
$A \rightarrow aA|\epsilon$
$C \rightarrow cC|\epsilon$

Sääntöpari $S \rightarrow Ab$ ja $S \rightarrow Cd$:

$FIRST(\{Ab\}FOLLOW(S)) = FIRST(\{Ab\}\{\epsilon\})$, koska S voi esiintyä vain johdon ensimmäisenä lausejohdoksena (pelkkä aloitussymboli S, jota siis seuraa ϵ).

$FIRST(Ab\epsilon) = a, b$, koska A:ta lavennettaessa 1. merkki voi olla a tai ϵ, jolloin seuraava merkki (b) onkin merkkijonon Ab 1. merkki. Huom! ϵ ei voi olla merkkijonon $\{Ab\}\{\epsilon\}$ 1. merkki, sillä välissä on aina b, joka ei tyhjene mihinkään.

Samaan tapaan:
$FIRST(\{Cd\}FOLLOW(S)) = FIRST(\{Cd\}\{\epsilon\}) = \{c, d\}$

Sääntöparin $A \rightarrow aA$ ja $A \rightarrow \epsilon$ käsittely:

$FIRST(\{aA\}FOLLOW(A)) = FIRST(\{aA\}\{b\})$, koska A:ta voi seurata vain b jossain lausejohdoksessa - se ei voi olla merkkijonon viimeinen, koska S:stä lähtiessä sitä seuraa b!

$FIRST(\{aA\}\{b\}) = FIRST(\{aAb\}) = \{a\}$
$FIRST(\{\epsilon\}FOLLOW(A)) = FIRST(\{\epsilon b\}) = \{b\}$

Kolmas sääntöpari:

$FIRST(\{cC\}FOLLOW(C)) = FIRST(\{cC\}\{d\}) = \{c\}$
$FIRST(\{\epsilon\}FOLLOW(C)) = FIRST(\{\epsilon\}\{d\}) = \{d\}$

Eli mikään sääntöpari ei riko LL(1)-ehtoa (leikkaus tyhjä).

Esim. 3

$S \rightarrow (L)|a$
$L \rightarrow L, S|S$

Ensimmäinen sääntöpari:

$FIRST(\{(L)\}FOLLOW(S)) = FIRST(\{(L)\}\{')','\,,\epsilon\}) = \{'('\}$
$FIRST(\{a\}FOLLOW(S)) = \{a\}$
eli leikkaus tyhjä - ok.

Toinen sääntöpari:

$FIRST(\{L,S\}FOLLOW(L)) = FIRST(\{L,S\}\{')','\,'\}) = FIRST(L) = \{'('\,,'\,a'\}$
$FIRST(\{S\}FOLLOW(L)) = FIRST(\{S\}\{')','\,'\}) = FIRST(S) = \{'('\,,'\,a'\}$
eli leikkaus epätyhjä! Ei siis LL(1).

Huom! $FIRST(\{L,S\}\{')','\,'\}) = FIRST(L)$, koska L ei voi tyhjentyä (sitä ei voi korvata ϵ:illa missään johdoksessa - siispä merkkijonon 1. merkki on L:n 1. merkki. Samoin $FIRST(\{S\}\{')','\,'\}) = FIRST(S)$.

4.6.5 *LL(1)-kieliopin rekursiivisesti etenevän jäsentäjän muodostaminen yleisesti

Perusidea: Jos seuraava syötemerkki löytyy joukosta $FIRST(\{\omega_1\}FOLLOW(A))$, niin valitse sääntö $A \rightarrow \omega_1$.

Apurutiinit:

```
token getnext;
    Seuraavan päätesymbolin selaus
...         — voi olla > 1 kirjainmerkkiä.*
void ERROR(char* msg);
        Virheiden käsittely; parametri msg
...     sisältää virheilmoitustekstin.*
```

Välikettä A vastaava funktio:

function A /*A:n produktiot $A \to \omega_1 \mid \ldots \mid \omega_n$*/
begin
  if (*next* **in** $[a_{11}, \ldots, a_{1m_1}]$) /*FIRST($\{\omega_1\}$FOLLOW($A$)) = $\{a_{11}, \ldots, a_{1m_1}\}$*/
  begin /*produktio $A \to \omega_1$*/
    $parse(\omega_1)$;
  end
  else
  ⋮
  if (*next* **in** $[a_{n1}, \ldots, a_{nm_n}]$) /*FIRST($\{\omega_n\}$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*; (a on päätemerkki)

$parse(B) \equiv B$; (B on välike)

4.6.6 Kielioppien muokkaaminen LL(1)-muotoon

LL(1)-kielet kattavat vain pienen joukon kontekstittomia kieliä – kaikkia kielioppeja ei siis voi muuntaa LL(1)-muotoon. Joskus kieli on LL(1)-luokassa, mutta sen

kuvaava kielioppi ei ole oikeassa muodossa. Tällaiset "melkein" LL(1)-kieliopit voi muokata oikeaan muotoon seuraavilla operaatioilla:

1. VASEN TEKIJÖINTI

- Kielioppi, jossa on produktiot

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

  ei voi olla LL(1)-muotoinen

- Korjaus: otetaan käyttöön uusi välike A' ja korvataan em. produktiot produktioilla:

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

  missä α on $\alpha\beta_1$:n ja $\alpha\beta_2$:n pisin yhteinen alkuosa

- Otetaan siis yhteinen tekijä eteen (vrt. matematiikassa $ax + ay = a(x + y)$).

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

- *Välitön* vasen rekursio, so. suorat johdot $A \Rightarrow A\gamma$, voidaan välttää korvaamalla produktiot

$$A \to A\beta \mid \alpha, \qquad \beta \neq \epsilon,$$

  produktioilla

$$\begin{aligned} A &\to \alpha A' \\ A' &\to \beta A' \mid \epsilon \end{aligned}$$

- Muotoa $A \to A$ olevat produktiot voidaan yksinkertaisesti jättää pois

- Perustelu: A:n johdot ovat muotoa $A \Rightarrow A\beta \Rightarrow A\beta\beta \Rightarrow A\beta\beta\beta \Rightarrow ... \Rightarrow \alpha\beta\beta...\beta$ eli $\alpha\beta^*$. Lopulta on siis valittava sääntö $A \rightarrow \alpha$ tai rekursio ei pääty ikinä.

Jokainen kontekstiton kielioppi voidaan teoriassa muuntaa vasemman rekursion välttävään *Greibachin normaalimuotoon*, missä kaikki produktiot ovat muotoa

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

tai $S \rightarrow \epsilon$, missä a on päätemerkki, $B_1, \dots, B_k$ välikkeitä ja S lähtösymboli

4.6.7 *LIITE: FIRST- ja FOLLOW-joukkojen laskeminen

Annettu kielioppi $G = (V, \Sigma, P, S)$.

Ensin lasketaan FIRST-joukot:

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

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

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

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

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

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

FOLLOW-joukot määritetään FIRST-joukkojen avulla seuraavasti:

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

$$\begin{aligned} &\text{FOLLOW}(B) := \\ &\quad \bigcup \{\text{FIRST}(\beta) - \{\epsilon\} \mid A \to \alpha B \beta \text{ on } G\text{:n produktio}\}, \end{aligned}$$

ja lähtösymbolille S lisäksi:

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

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

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

4.6.8 LL(1)-kielien vahvemmat sisarukset

Edellä esitetty rekursiivisen jäsentäjän idea voidaan yleistää tapaukseen, jossa k seuraavaa merkkiä määräävät yksikäsitteisesti, mikä sääntö valitaan kullakin vasemman johdon jäsennysaskeleella. Tällaisia kielioppeja kutsutaan *LL(k)-kieliopeiksi* ("left to right scan, producing left parse with k symbol lookahead").

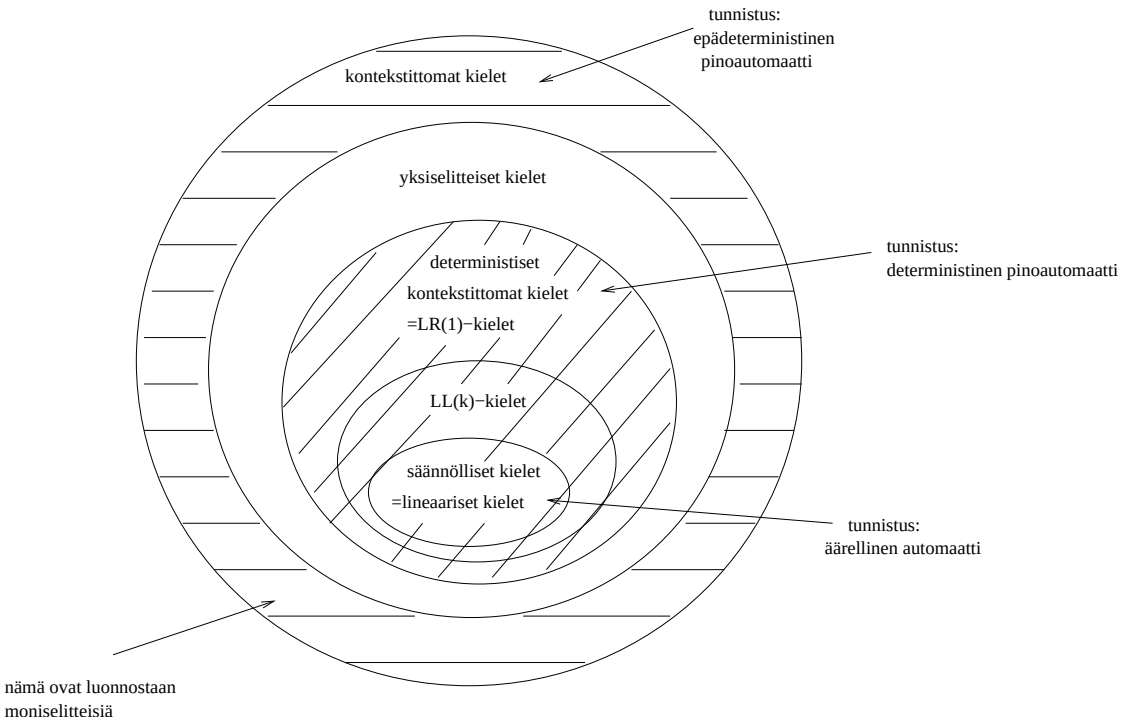
Vielä laajempi kieliluokka saadaan, kun simuloidaan merkkijon oikeaa johtoa rekursiivisesti. *LR(1)-kieliopissa* ("left to right scan, producing right parse with 1 symbol lookahead") jäsennys etenee oikealta vasemmalle (ts. lavennetaan aina oikeanpuolimmaisin välikesymboli) ja seuraava merkki määrittelee yksikäsitteisesti käytettävän säännön. Vastaavasti *LR(k)-kielissä* seuraavat k merkkiä määrittävät seuraavan johtoaskeleen. Tällaisen alhaalta-ylöspäin (*bottom-up*) [1] etenevän jäsentäjän laatiminen on kuitenkin vaikeampaa kuin LL-kielien edellyttämän ylhäältä alas (*top-down*) [2] etenevän jäsentäjän laatiminen, eikä siihen perehdytä tällä kurssilla. Lisää tietoa löytyy esim. Sudkampin kirjasta.

LR(1)-kielet ovat kuitenkin hyvin tärkeä kieliluokka, sillä ne määrittelevät täsmälleen kaikki *deterministiset* kielet ts. kielet, jotka voidaan tunnistaa deterministisellä pinoautomaatilla!

Huom! On olemassa kieliä, jotka eivät ole deterministisiä, mutta joilla silti on olemassa yksiselitteinen kielioppi. Esimerkiksi pladindromikielelle $L = \{ww^R | w \in \{a, b\}^*\}$ voidaan antaa yksiselitteinen kielioppi, mutta silti sen sanoja ei voida tunnistaa deterministisellä automaatilla. Ongelmana on se, että automaatin täytyy "arvata", milloin on tultu merkkijonon keskikohtaan.

[1] bottom-up-jäsennyksessä jäsennyspuuta muodostetaan alhaalta ylöspäin, lähtösymbolista S kohti merkkijonoa x

[2] top-down-jäsennyksessä jäsennyspuuta muodostetaan ylhäältä alaspäin, merkkijonosta x kohti lähtösymbolia S



Kuva 4.10: Kuva kontekstittomien kielten alaluokkien välisistä suhteista.

4.7 CYK-jäsennysalgoritmi

- Rekursiivisesti etenevä jäsentäminen on selkeä ja tehokas jäsennysmenetelmä LL(1)-kieliopeille: n merkin mittaisen syötemerkkijonon käsittely sujuu ajassa $O(n)$
- Entäpä mielivaltaisen kontekstittoman kieliopin tuottamien merkkijonojen tunnistaminen?
- Ratkaisu 1: muunnetaan kielioppi ensin em. Greibachin normaalimuotoon
  - Tämänmuotoisella kieliopilla millä tahansa n merkin mittaisella lauseella on n:n askeleen johto
  - Merkkijonon x, $|x| = n$, kuuluminen tuotettuun kieleen voidaan testata käymällä läpi kaikki n askelen mittaiset johdot ja katsomalla, tuottaako jokin niistä x:n
  - Tehotonta! (jokaisessa epätriviaalissa kieliopissa on n askelen mittaisia johtoja eksponentiaalinen määrä)

- Ratkaisu 2: *Cocke–Younger–Kasami-* eli *CYK-algoritmi*
  - Toimii ajassa $O(n^3)$, missä n on tutkittavan merkkijonon pituus
  - Kielopin on oltava *Chomskyn normaalimuodossa*
  - Mikä tahansa kontekstiton kielioppi voidaan kuitenkin muuntaa Chomskyn normaalimuotoon, joten sopii kaikille kontekstittomille kielille!

Chomskyn normaalimuoto

Määritelmä: Kontekstiton kielioppi $G = (V, \Sigma, P, S)$ on *Chomskyn normaalimuodossa*, jos

- Välikkeistä enintään S on tyhjentyvä (engl. nullable), eli $S \underset{G}{\Rightarrow}^* \epsilon$
- Muut produktiot ovat muotoa $A \to BC$ tai $A \to a$, missä A, B ja C ovat välikkeitä ja a on päätemerkki
- Lisäksi vaaditaan yksinkertaisuuden vuoksi, että lähtösymboli S ei esiinny minkään produktion oikealla puolella

Esim. seuraava kielioppi on Chomskyn normaalimuodossa:
$S \to AB|\epsilon$
$A \to BA|a$
$B \to b$

Cocke–Younger–Kasami-algoritmi

- Perustuu dynaamiseen ohjelmointiin (välitulosten taulukointiin)
- Ongelma: Olk. $G = (V, \Sigma, P, S)$ Chomskyn normaalimuodossa (ellei ole, muutetaan ensin Chomskyn normaalimuotoon). Onko $x \in L(G)$ eli $S \underset{G}{\Rightarrow}^* x$?

Algoritmi:

1. Jos $x = \epsilon$, niin $x \in L(G) \Leftrightarrow S \to \epsilon$ on G:n produktio

2. Muuten merk. $x = a_1 \dots a_n$ ja tarkastellaan x:n osajonot tuottavien välikkeiden joukkoja

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

3. $x \in L(G) \Leftrightarrow S \in N_{1,n}$

Joukkojen $N_{i,j}$ laskeminen

- muodostetaan kolmiomainen taulukko:

a_1	a_2	a_3	a_4	a_5
$N_{1,1}$				
$N_{1,2}$	$N_{2,2}$			
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$

- x-akseli vastaa merkkijonon positiota
- taulukon paikka $N_{i,j} \sim$ joukko välikesymboleita A, joille $A \underset{G}{\Rightarrow}^* a_i a_{i+1} \dots a_j$
- jokainen diagonaali vastaa tietyn pituisia x:n osajonoja
  - 1. diagonaali $\sim$ välikkeet, jotka tuottavat yhden merkin osajonot
  - 2. diagonaali $\sim$ välikkeet, jotka tuottavat kahden merkin osajonot jne.

(i) Lasketaan ensin for $i = 1$ to n:

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

(ii) Lasketaan sitten induktiivisesti

for $k = 1$ to $n - 1$
  for $i = 1$ to $n - k$:

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

- Huom! Tunnetaan jo kaikki lyhyemmät osajonot ja kaikki x:n kelvolliset prefiksit ja suffiksit
- Johdon $A \underset{G}{\Rightarrow}^* a_i a_{i+1} ... a_j$ täytyy alkaa askeleella $A \Rightarrow BC$, missä B:stä voidaan johtaa $a_i ... a_j$:n prefiksi, olk. $B \underset{G}{\Rightarrow}^* a_i a_{i+1} ... a_l$, ja C:stä voidaan johtaa loput:
  $C \underset{G}{\Rightarrow}^* a_{l+1} a_{l+2} ... a_j$
- Esim. $N_{3,7}$:ta varten tarkistellaan kaikkia pareja
  $(N_{3,3}, N_{4,7}), (N_{3,4}, N_{5,7}), (N_{3,5}, N_{6,7}), (N_{3,6}, N_{7,7})$

Esim. Sovelletaan CYK:iä Chomskyn normaalimuotoiseen kielioppiin G:

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

syötemerkkijonolla $x = baaba$:

$i \rightarrow$

$1:b$	$2:a$	$3:a$	$4:b$	$5:a$
B				
S,A	A,C			
$\emptyset$	B	A,C		
$\emptyset$	B	S,C	B	
S,A,C	S,A,C	B	S,A	A,C

Lähtösymboli S kuuluu joukkoon $N_{1,5}$, joten x kuuluu kieliopin tuottamaan kieleen $L(G)$.

Tutkitaan sitten merkkijonoa $ababab$:

$i \rightarrow$

$1:a$	$2:b$	$3:a$	$4:b$	$5:a$	$6:b$
A,C					
S,C	B				
$\emptyset$	A,S	A,C			
B	S	S,C	B		
A,S	$\emptyset$	$\emptyset$	A,S	A,C	
S	$\emptyset$	B	S	S,C	B

Siis $ababab \in L(G)$.

Jäsennystä voi joskus helpottaa valitsemalla pääteaakkoston sopivasti. Pääteaakkosten ei tarvitse olla yksittäisiä kirjaimia, vaan ne voivat olla myös kielen "atomisia" sanoja (osasia, joista pidemmät sanat koostuvat). Jos kieliopissa on sääntö $A \rightarrow w$, missä w koostuu vain päätemerkeistä, voidaan w valita uuden aakkoston päätesymboliksi.

Esim. 2. Tarkastellaan seuraavaa Kissastanian kielen mukaista kielioppia G_{miu}:

$S \rightarrow miuSmau|miu|mau|U$
$U \rightarrow Uu|u$

Valitaan pääteaakkostoksi $\Sigma = \{miu, mau, u\}$. Chomskyn normaalimuotoon muunnettuna saadaan kielioppi

$S \to IW|UU|miu|mau$
$W \to SA$
$U \to Uu|u$
$I \to miu$
$A \to mau$

Tutkitaan CYK-algoritmilla, kuuluuko merkkijono $miumiuuumaumau$ kieleen:

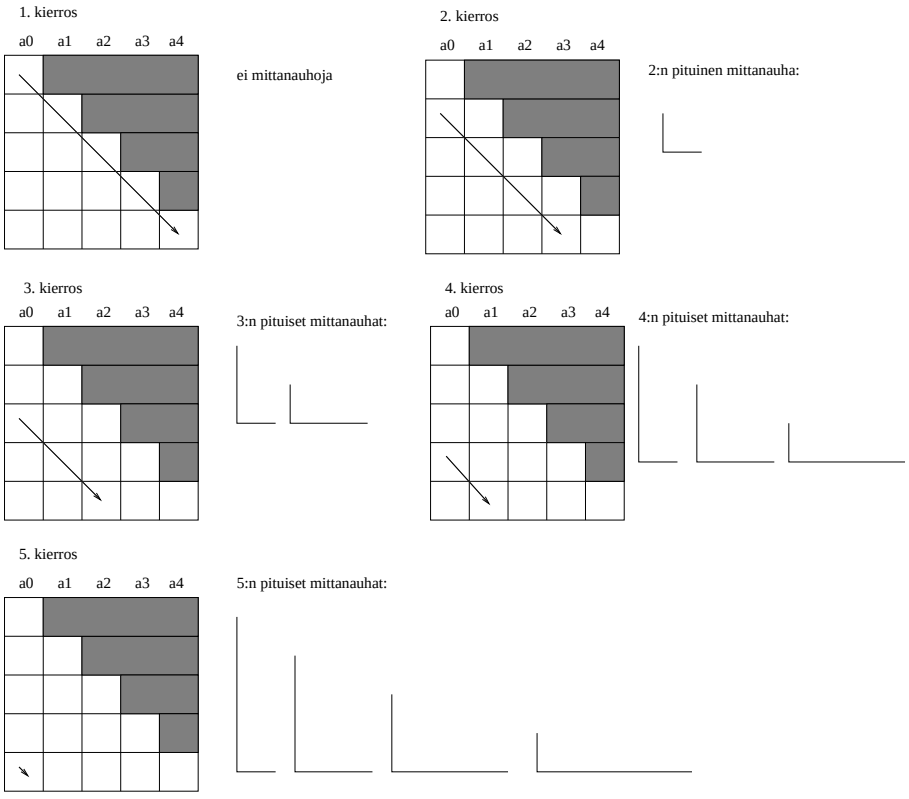
$i \to$

$1 : miu$	$2 : miu$	$3 : u$	$4 : u$	$5 : mau$	$6 : mau$
S, I					
$\emptyset$	S, I				
$\emptyset$	$\emptyset$	U			
$\emptyset$	$\emptyset$	S	U		
$\emptyset$	S	W	$\emptyset$	S, A	
S	W	$\emptyset$	$\emptyset$	W	S, A

Siis $miumiuuumaumau \in L(G_{miu})$.

Apukeino CYK:in simuloimiseen

CYK-algoritmissa edetään siis diagonaaleittain, kunnes tullaan vasempaan alakulmaan. Mikäli merkkijonon pituus on n, täytyy käydä läpi n diagonaalia. 1. diagonaalin i:teen sarakkeeseen tulevat vain ne välikkeet A, joilla on sääntö $A \to a_i$. Seuraavilla diagonaaleilla ruutu täytetään edellisten diagonaalien sisällön perusteella. Voit kuvitella mielessäsi (tai askarrella piirtoheitinkalvosta!) "mittanauhat", joiden avulla muistat, mitä edeltävistä ruuduista täytyy tutkia ruudun täyttämiseksi. 2. diagonaalilla mitan pituus on 2 ja k:nnellä diagonaalilla mitan pituus on k. Huomaa, että kahden pituisen mitan voi asettaa vain yhdellä tapaa taulukkoon, mutta kolmenpituisella mitalla on jo kaksi vaihtoehtoista tapaa. k:nnella diagonaalilla tarvittavia k:n pitusia "mittanauhoja" on siis jo $k-1$ kpl.



Kuva 4.11: CYK:in simulointi. Taulukkoa käydään läpi diagonaaleittain, ja diagonaalin joka ruudussa sovitetaan kaikkia annetuista "mittanauhoista". Mittanauhan päiden kohdalla olevat ruudut tutkitaan, ja mikäli niissä oleville välikkeille B ja C löytyy sääntö $A \to BC$, sijoitetaan A senhetkiseen ruutuun.

4.8 Muunnos Chomskyn normaalimuotoon

- Idea:
  1. Poistetaan lähtösymboli produktioiden oikealta puolelta
  2. Poistetaan ϵ-produktiot
  3. Poistetaan yksikköproduktiot $A \to B$
  4. Poistetaan produktiot $A \to X_1X_2...X_k$, $k > 2$
- Produktioiden oikealla puolella olevien lähtösymbolien poistaminen
  - Jos lähtösymboli S on jonkin produktion oikealla puolella lisätään uusi lähtösymboli S' ja sille produktio $S' \to S$

ϵ-produktioiden poistaminen

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

Lemma:
Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa enintään lähtösymboli on tyhjentyvä.
(Huom! Lähtösymbolin tyhjentymistä ei voida välttää, jos $\epsilon \in L(G)$.)
Todistus: Olkoon $G = (V, \Sigma, P, S)$.

1. Selvitetään ensin G:n tyhjentyvät välikkeet (NULL-joukko) seuraavasti:
   (i) asetetaan aluksi
   $$\text{NULL} := \{A \in V - \Sigma \mid A \to \epsilon \text{ on } G\text{:n produktio}\};$$
   (ii) toistetaan sitten seuraavaa NULL-joukon laajennusoperaatiota, kunnes joukko ei enää kasva:
   $$\begin{aligned} \text{NULL} \;:=\; & \text{NULL} \cup \\ & \{A \in V - \Sigma \mid A \to B_1 \dots B_k \text{ on } G\text{:n produktio}, \\ & B_i \in \text{NULL kaikilla } i = 1, \dots, k\}. \end{aligned}$$
2. Tämän jälkeen korvataan kukin G:n produktio $A \to X_1 \dots X_k$ kaikkien sellaisten produktioiden joukolla, jotka ovat muotoa
   $$A \to \alpha_1 \dots \alpha_k, \qquad \text{missä} \quad \alpha_i = \begin{cases} X_i, & \text{jos } X_i \notin \text{NULL}; \\ X_i \text{ tai } \epsilon, & \text{jos } X_i \in \text{NULL}. \end{cases}$$

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

Esim. Poistetaan ϵ-produktiot seuraavasta kieliopista:

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

$$\begin{array}{lcll} S & \to & A \mid B \mid \epsilon & \\ A & \to & aBa \mid aa \mid \epsilon & \Rightarrow \\ B & \to & bAb \mid bb \mid \epsilon & \end{array}$$

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

Yksikköproduktioiden poistaminen

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

  Lemma:
  Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa ei ole yksikköproduktioita.
  Todistus: Olkoon $G = (V, \Sigma, P, S)$.

  1. Selvitetään ensin G:n kunkin välikkeen "yksikköseuraajat" seuraavasti:

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

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

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

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

2. 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 $B \in F(A)$. $\square$

Esim. Poistetaan yksikköproduktiot edellä saadusta kieliopista

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

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

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

Produktioiden $A \to X_1 \dots X_k$, $k > 2$ poisto

- halutaan poistaa oikealta puolelta merkkijonot, joissa on enemmän kuin kaksi välikesymbolia, ja samalla muutetaan joukossa olevat päätesymbolit sopiviksi välikkeiksi (siis esim. $A \to BbC$, $A \to abcd$ ovat tällaisia "laittomia" sääntöjä)

- Lisätään kielioppiin kutakin päätemerkkiä a varten uusi välike C_a ja sille produktio $C_a \to a$

- Korvataan kussakin muotoa $A \to X_1 \dots X_k$, $k > 2$, olevassa produktiossa ensin kaikki päätemerkit em. uusilla välikkeillä, ja sitten koko produktio produktiojoukolla

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

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

(Tarkkaan ottaen uusi produktiojoukko on siis oikeastaan

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

missä

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

Lause: Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti Chomskyn normaalimuotoinen kielioppi G'.
Todistus: Olkoon $G = (V, \Sigma, P, S)$. Poistetaan ensin G:stä lähtösymboli produktioiden oikealta puolelta, ϵ-produktiot ja yksikköproduktiot em. lemmojen konstruktioilla. Tämän jälkeen kaikki G:n produktiot ovat muotoa $A \to a$ tai $A \to X_1 \dots X_k$, $k \geq 2$ (tai $S \to \epsilon$). Viimeksi mainitut poistetaan ed. konstruktion mukaisesti. □

Huom! Turhat välikkeet ja produktiot voidaan aina poistaa.

Esim. Muunnetaan Chomskyn normaalimuotoon kielioppi

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

Tuloksena saadaan kielioppi

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

4.9 *Attribuuttikieliopit

4.9.1 Perusidea

Attribuuttikieliopit ovat kätevä tapa ilmaista, mitä toimintoja jäsennyksen yhteydessä suoritetaan. Ne siis lisäävät puhtaasti syntaktiseen kielioppiin semantiikan (merkityksen). Attribuuttikieliopit ovat tarpeellisia esimerkiksi kääntäjissä, joissa lähdekoodin jäsennys (syntaktisen oikeellisuuden tarkistus) ei riitä, vaan koodi pitäisi myös kääntää suoritettavaksi ohjelmaksi. Ajan puutteen takia emme kuitenkaan käsittele attribuuttikielioppeja kurssillamme, mutta seuraavssa on esitlty niiden idea lyhyesti.

- liitetään kontekstittomiin kielioppeihin kielen semantiikan kuvausta
- esim. lauseke $3 + 2 \times 1$ noudattaa kieliopin G_{expr} syntaksia (ja kuuluu siis G_{expr}:in tuottamaan kieleen) – mutta mitä lauseke merkitsee?
- lausekkeen arvon evaluoiminen edellyttää sopivien attribuuttien ja niiden evaluointisääntöjen liittämistä kielioppiin
- Idea:
  - kieliopin mukaisen jäsennyspuun solmu, jonka nimi on symboli $X \sim$ tietue tyyppiä X.
  - tietuetyyppiin X kuuluvat kentät $\sim X$:n *attribuutit*, merk. $X.s$, $X.t$ jne.
  - kussakin X-tyyppisessä jäsennyspuun solmussa X:n attribuuteista eri *ilmentymät*
  - produktioihin $A \to X_1 \dots X_k$ liitetään attribuuttien *evaluointisääntöjä*, jotka ilmaisevat miten annetun jäsennyspuun solmun attribuutti-ilmentymien arvot määräytyvät sen vanhempi- ja lapsisolmujen attribuutti-ilmentymien arvoista.
- Huom! Säännöt voivat olla periaatteessa minkälaisia funktioita tahansa, kunhan niiden argumentteina esiintyy vain paikallisesti saatavissa olevaa tietoa.
- ts. produktioon $A \to X_1 \dots X_k$ liitettävissä säännöissä saa mainita vain symbolien $A, X_1, \dots, X_k$ attribuutteja

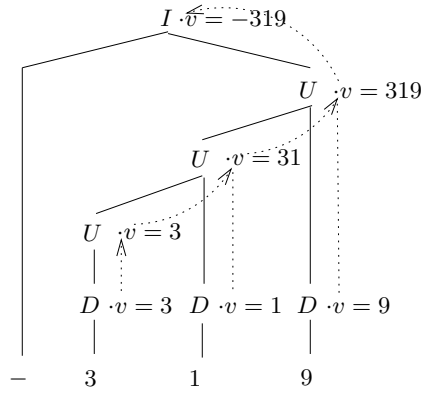
Esim. 1 Liitetään etumerkillisiä kokonaislukuja tuottavaan kontekstittomaan kielioppiin attribuutit ja niiden evaluointisäännöt kieliopin tuottamien lukujen arvojen määrittämiseen.

- kuhunkin jäsennyspuun X-tyyppiseen välikesolmuun liitetään attribuutti-ilmentymä $X.v$, jonka arvoksi tulee X:stä tuotetun numerojonon lukuarvo
- 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

- produktion evaluointisäännössä saman välikkeen eri esiintymät voidaan erottaa alaindekseillä
- yllä U_1 vastaa ensimmäistä produktiossa $U \rightarrow UD$ esiintyvää U:ta ja U_2 toista

Evaluointisääntojen mukainen *attributoitu jäsennyspuu* kieliopin tuottamalle lauseelle "-319" (katkoviivoilla on esitetty attribuutti-ilmentymien väliset evaluointiriippuvuudet):



Kuva 4.12: Attributoitu jäsennyspuu.

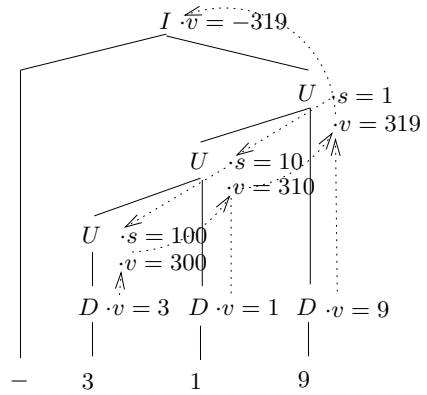
- Attribuuttikieliopin attribuutti t on *synteettinen*, jos sen kuhunkin produktioon $A \to X_1 \ldots X_k$ liittyvä evaluointisääntö on muotoa $A.t := f(A, X_1, \ldots, X_k)$.
- Tällöin jäsennyspuussa kunkin solmun mahdollisen t-ilmentymän arvo riippuu vain solmun omien ja sen jälkeläisten attribuutti-ilmentymien arvoista
- Muunlaiset attribuutit ovat *periytyviä*
- yllä esim. attribuutti v on synteettinen
- pyritään käyttämään pääasiassa synteettisiä attribuutteja, koska ne voidaan evaluoida helposti yhdellä jäsennyspuun lehdistä juureen suuntautuvalla läpikäynnillä.
- perittyjä attribuutteja voidaan käyttää, kunhan attribuutti-ilmentymien riippuvuusverkkoihin ei tule syklejä

Esimerkiksi edellä olevaan, etumerkillisiä kokonaislukuja tuottavaan kielioppiin voitaisiin liittää lukujen arvot määrittävä semantiikka myös seuraavasti, periytyvää "positiokerroin"-attribuuttia s ja synteettistä "arvo"-attribuuttia v käyttäen:

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

Kuvassa 4.13 on esitetty tämän semantiikan mukainen attributoitu jäsennyspuu lauseelle "-319".

Attribuutti-ilmentymien arvot voidaan usein laskea suoraan jäsennysrutiineissa, tarvitsematta muodostaa jäsennyspuuta eksplisiittisesti.



Kuva 4.13: Positiokerrointa käyttäen attributoitu jäsennyspuu.

4.9.2 Aritmeettinen laskin

Esim. 2 Laaditaan aritmeettinen laskin, jolla voi suorittaa etumerkillisten kokonaislukujen yhteen-, kerto- ja vähennyslaskuoperaatioita. Selvyyden vuoksi vaaditaan, että etumerkillisten lukujen ympärillä on aina sulut. Kielioppi on siis seuraava:

$E \rightarrow T + E | T - E | T$
$T \rightarrow F | F * T$
$F \rightarrow DN | D | (+DN) | (-DN) | (E)$
$N \rightarrow DN | D$
$D \rightarrow 0|1|...|9$

Esitetään lausekkeen arvon evaluointi attribuuttikielioppina:

Produktiot:			*Evaluointisäännöt:*		
E	$\rightarrow$	$T + E$	$E_1.v$	$:=$	$T.v + E_2.v$
E	$\rightarrow$	$T - E$	$E_1.v$	$:=$	$T.v - E_2.v$
E	$\rightarrow$	T	$E.v$	$:=$	$T.v$
T	$\rightarrow$	$F * T$	$T_1.v$	$:=$	$F.v \times T_2.v$
T	$\rightarrow$	F	$T.v$	$:=$	$F.v$
F	$\rightarrow$	DN	$F.v$	$:=$	$10 \times D.v + N.v$
F	$\rightarrow$	D	$F.v$	$:=$	$D.v$
F	$\rightarrow$	$(+DN)$	$F.v$	$:=$	$10 \times D.v + N.v$
F	$\rightarrow$	$(-DN)$	$F.v$	$:=$	$-1 \times (10 \times D.v + N.v)$
F	$\rightarrow$	(E)	$F.v$	$:=$	$E.v$
N	$\rightarrow$	DN	$N_1.v$	$:=$	$10 \times D.v + N_2.v$
N	$\rightarrow$	D	$N.v$	$:=$	$D.v$
D	$\rightarrow$	0	$D.v$	$:=$	0
	...			...	
D	$\rightarrow$	0	$D.v$	$:=$	9

Ohjelmassa ei lukuja tarvitse lukea merkki kerrallaan, vaan voimme käyttää jotain sopivaa apurutiinia. Välikkeitä N ja D sisältävissä säännöissä voidaan siis oikaista.

Aritmeettinen kokonaislukulaskin pseudokoodina:

```
/*Apurutiinit: */
char getnext; /* lue seuraava merkki */
int isdigit(char c); /* onko c numeromerkki? */
int readnumber; /* lue koko luku ja palauta sen arvo */
void ERROR; /* virheenkäsittely */
/* Seuraava merkki: */
char next;


int function E {
int op1, op2;
op1 = T;
if(next ==' +') { /* rule E → T + E */
    next = getnext;
    op2 = E;
    return op1 + op2;
}
```

```
else
    if (next ==' -') { /* rule E → T - E */
        next = getnext;
        op2 = E;
        return op1 - op2;
    }
    else return op1; /* rule E → T */
}


int function T {
int op1, op2;
op1 = F;
if (next ==' *') { /* rule T → F * T */
    next = getnext;
    op2 = T;
    return op1 * op2;
}
else return op1; /* rule T → F */
}


int function F {
int op
if (isdigit(next)) { /* rule F → DN */
    op = readnumber;
    next = getnext;
    return op;
}
else
    if (next ==' (') {
        next = getnext;
        if (next ==' +') { /* rule F → (+DN) */
            op = readnumber;
            if (next! =')') ERROR;
            next = getnext;
            return op;
        }
        else{ /* rule F → (-DN) */
            if (next ==' -') {
```

```
            next = getnext;
            op = readnumber;
            if (next! =')') ERROR;
            next = getnext;
            return −1 ∗ op;
        }
        else{ /* rule F → (E) */
            next = getnext;
            op = E;
            if (next! =')') ERROR;
            next = getnext;
            return op;
        }
    }
}
else ERROR;
}
```

4.10 Kontekstittomien kielten ominaisuuksia

4.10.1 Kontekstittomien kielten sulkeumaominaisuudet

Kontekstittomille kielille pätee joitain samantapaisia sulkeumaominaisuuksia kuin säännöllisille kielille.

Lause Olkoon L_1 ja L_2 kontekstittomia kieliä. Tällöin myös

1. $L_1 \bigcup L_2$ (kielten yhdiste)
2. $L_1 L_2$ (kielten katenaatio)
3. $(L_1)^*$ (kielen sulkeuma)
4. $(L_1)^R$ (kielen käänteiskieli)

ovat kontekstittomia.

Huom! Kontekstittomat kielet eivät kuitenkaan ole suljettuja leikkauksen ja komplementin suhteen! Jos L_1 ja L_2 ovat kontekstittomia, ei $L_1 \cap L_2$ silti ole välttämättä kontekstiton! Samoin $\overline{L_1} = \Sigma^* \setminus L_1$ (kielen komplementti) ei välttämättä ole kontekstiton.

Esim. kielet $L_1 = \{a^n b^n c^k | n, k = 0, 1, ...\}$ ja $L_2 = \{a^k b^n c^n | k, n = 0, 1, ...\}$ ovat kontekstittomia. Kuitenkaan kieli $L_1 \cap L_2 = \{a^n b^n c^n | n = 0, 1, ...\}$ ei ole kontekstiton!

Toisaalta, jos L on kontekstiton kieli ja R on säännöllinen kieli, niin $L \cap R$ on kontekstiton. (Ks. esim. Hopcroft, Motwani, Ullman, teoreema 7.27.)

4.10.2 Ratkeavat ja ratkeamattomat ominaisuudet

Edellä olemme jo oppineet joitain kontekstittomien kielten ominaisuuksia, joita voidaan ratkaista tietokoneella. Tärkein tällainen ominaisuus koskee jäsentämistä: mille tahansa merkkijonolle x ja annetulle kontekstittomalle kieliopille G voimme ratkaista, kuuluuko x kieleen $L(G)$. Ts. ongelma $x \in L(G)$ on algoritmisesti ratkeava. Mikä jäsennysmenetelmä on kulloinkin paras, riippuu annetusta kielestä. Jos kielioppi voidaan esittää LL(1)-muodossa, tarjoaa tämä tehokkaan (ajassa $O(n)$ toimivan) ja helpon jäsennysmenetelmän. Ohjelmointikielten kääntäjät puolestaan käyttävät LR(k)-jäsentäjiä, kuten Unixin yacc- ja bison-ohjelmien tuottamia jäsentäjiä. Mielivaltaisen kieliopin taas voimme aina muuntaa Chomskyn normaalimuotoon ja jäsentää CYK-algoritmilla ajassa $O(n^3)$.

Eräät tärkeistä kontekstittomien kielten ominaisuuksista ovat kuitenkin ratkeamattomia ongelmia. Tällaisia ongelmia ovat:

- Kahden kieliopin ekvivalenssi ts. $L(G_1) = L(G_2)$?
- Annetun kieliopin moniselitteisyys $Ambiguous(G)$?
- Onko annetun mv. kieliopin kuvaama kieli kontekstiton $Context{-}free(L(G))$?

Viimeiseen mäistä ongelmista liittyy kontekstittomien kielten pumppauslemma, jolla ihminen voi osoittaa joitain kieliä ei-kontekstittomiksi samaan tapaan kuin säännöllisten kielten pumppauslemmalla. Tämäkään pumppauslemma ei kuitenkaan anna kontekstittomuuden välttämättömiä vaan ainoastaan riittävät ehdot, eikä pumppauslemmaa voida soveltaa algoritmisesti. Ongelma pysyy siis yleisessä tapauksessa ratkeamattomana.

4.11 *Kontekstittomien kielten rajoituksista

- Kontekstittomille kielille voidaan todistaa samantapainen pumppauslemma kuin säännöllisillekin kielille
- Erona on vain se, että nyt merkkijonoa (osat uvwxy) on pumpattava samanaikaisesti kahdesta paikasta (osista v ja x)
- *Lemma* [Kontekstittomien kielten pumppauslemma eli uvwxy-lemma]: Olk. L kontekstiton kieli. Tällöin on olemassa sellainen $n \geq 1$, että mikä tahansa $z \in L$, $|z| \geq n$, voidaan jakaa osiin $z = uvwxy$ siten, että

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

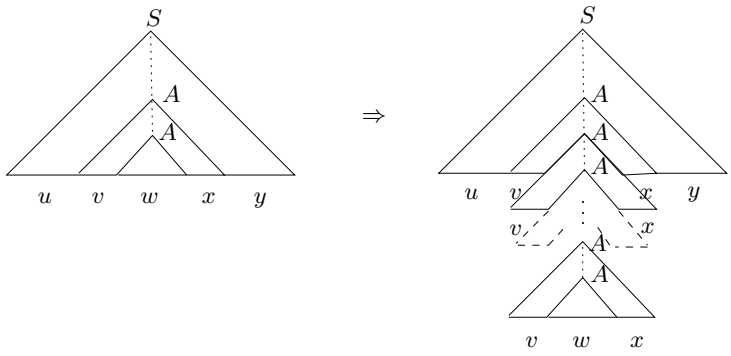
  (ii) $|vwx| \leq n$,

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

  Huom: Taas vain äärettömät kielet ovat kiinnostavia

  Todistus:

  – Olk. $G = (V, \Sigma, P, S)$ 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. Ts. minkä tahansa merkkijonon $z \in L$ jokaisessa jäsennyspuussa on polku, jonka pituus on vähintään $\log_2 |z|$



Kuva 4.14: Kontekstittoman kielen merkkijonon pumppaus.

- Olk. $k = |V - \Sigma|$ kieliopin G välikkeiden määrä. Asetetaan $n = 2^{k+1}$. Tarkastellaan jotakin $z \in L$, $|z| \geq n$, ja sen jotakin jäsennyspuuta
- $\Rightarrow$ puussa on polku, jonka pituus on $\geq k+1$. Tarkastellaan tämän polun "alinta" $k+1$:n pituista osaa. Tällä polulla, jossa on $k+1$ välikettä vastaavaa solmua ja välikkeitä on k kappaletta, on siis jonkin välikkeen toistuttava. Olkoon A joku polun toistuva välike (ks. kuva 4.14).
- Merkkijono z voidaan nyt osittaa $z = uvwxy$, missä w on A:n alimmasta ilmentymästä tuotettu osajono ja vwx seuraavaksi ylemmästä A:n ilmentymästä tuotettu osajono; osajonot saadaan johdosta

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

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

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uv^2Ax^2y \Rightarrow^* \ldots \Rightarrow^*$$

$$uv^iAx^iy \Rightarrow^* uv^iwx^iy$$

- $\Rightarrow uv^iwx^iy \in L$ kaikilla $i = 0, 1, 2, \ldots$
- Koska kielioppi G on Chomskyn normaalimuodossa ja $A \Rightarrow^* vAx$, on oltava $|vx| \geq 1$
- Ylemmästä A:n ilmentymästä alkavan jäsennyspuun polun pituus on enintään $k+1 \Rightarrow$ vastaavan alipuun tuotokselle $|vwx| \leq 2^{k+1} = n$. □

- Esim. Väite: kieli $L = \{a^kb^kc^k \mid k \geq 0\}$ ei ole kontekstiton. Tod. Vastaväite: L on kontekstiton. Valitaan parametri n lemman mukaisesti ja tarkastellaan merkkijonoa $z = a^nb^nc^n \in L$. Tällöin z voidaan jakaa pumpattavaksi osiin

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

Tällöin 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 eli $uwy \notin L$. Ristiriita. $\square$

4.12 Kontekstittomien kielten sovelluksia

Jäsentäjät

Ohjelmointikieli voidaan kuvata kontekstittomana kielioppina, jota voidaan käsitellä jäsentäjällä (*parser*). Jäsentäjä on kääntäjän osa, joka tutkii ohjelman rakenteen ja esittää sen jäsennyspuuna Esim. UNIX tarjoaa komennon `yacc`, joka generoi jäsentäjän annetusta kontekstittomasta kieliopista `yacc`:ille annetaan syötteenä kontekstiton kielioppi (`yacc`-notaatiossa säännön nuoli on korvattu kaksoispisteellä), jonka jokaiseen sääntöön voidaan liittää C-koodina määritelty toiminto. Toiminto suoritetaan aina, kun luodaan vastaava jäsennyspuun solmu. Tyypillisesti toiminto vain luo jäsennyspuun solmun, mutta joissain sovelluksissa jäsennyspuuta ei eksplisiittisesti luoda, vaan toiminto tekee jotain muuta, esim. sisällyttää palan objektikoodia kyseiseen paikkaan.

Esimerkkiksi seuraava kielioppi:

$E \rightarrow I|E+E|E*E|(E)$
$I \rightarrow a|b|Ia|Ib|I0|I1$

annettaisiin `yacc`:ille muodossa:

Exp	: Id	{...}
	| $Exp' +' Exp$	{...}
	| $Exp' *' Exp$	{...}
	| $'('Exp')'$	{...}
	;	
Id	: 'a'	{...}
	| 'b'	{...}
	| Id 'a'	{...}
	| Id 'b'	{...}
	| Id '0'	{...}
	| Id '1'	{...}
	;	

missä {...} sisältää jonkin toiminnon C-koodina.

Dokumentin rakenteen kuvaus

- ns. ”mark-up”-kielet kuten HTML ja XML

- kielen merkkijonot dokumentteja, joissa tiettyjä kielen semantiikkaa kuvaavia merkkejä (tags) — esim. <OL> ja </OL> $\sim$ järjestetty lista
- HTML: laillisen HTML-dokumentin kuvaus ja dokumentin prosessoinnin ohjaus
- XML: kuvaa tekstin semantiikan — esim.
  <ADDR>Perhospolu 17 C 3, 33000 Kukkamäki<ADDR> kertoo, että osajono on osoite.

Luku 5

Turingin koneet ja rajoittamattomat kielet

Tässä luvussa tutustumme kaikkein tärkeimpään konetyyppiin, Turingin koneisiin, joilla voidaan *Churchin-Turingin teesin* mukaan ratkaista mikä tahansa mekaanisesti ratkeava ongelma. Toisin sanoen Turingin koneet tunnistavat Chomskyn kielihierarkian vahvimman kielityypin, *rajoittamattomat kielet* (tyyppi 0). Turingin koneiden avulla voidaan siis tutkia, onko ongelma ylipäänsä ratkeava (keksitäänkö sen ratkaiseva Turingin kone) sekä analysoida ongelman vaativuutta (kuinka monta aika-askelta tai työnauhan solua kone tarvitsee työssään).

Rajoittamattomien kielten luokka voidaan jakaa kahteen luokkaan sen perusteella, pysähtyykö ongelman ratkaiseva Turingin kone kaikilla syötteillä (siis sekä hyväksyessään

että hylätessään merkkijonon se lopulta pysähtyy ja palauttaa vastauksen ”kyllä” tai ”ei”) vai pysähtyykö se vain hyväksyvässä tapauksessa (vastaus ”kyllä”). Ensimmäistä, aina pysähtyvien eli *totaalisten Turningin koneiden* tunnistamaa kieliluokkaa kustutaan *Rekursiivisiksi kieliksi* ja sitä vastaavat päätösongelmat ovat *ratkeavia* (*solvable, decidable*). Toista, vain myönteisissä tapauksissa pysähtyviä koneita vastaavaa kieliluokkaa kutsutaan *Rekursiivisesti numeroituviksi* kieliksi ja vastaavat päätösongelmat ovat *osittain ratkeavia* (*semidecidable*). Kieliluokkien ulkopuolelle jäävät täysin ratkeamattomat ongelmat. Huom! Myös osoittain ratkeavat ongelmat luetaan ratkeamattomiin ongelmiin (*unsolvable, undecidable*). [3]

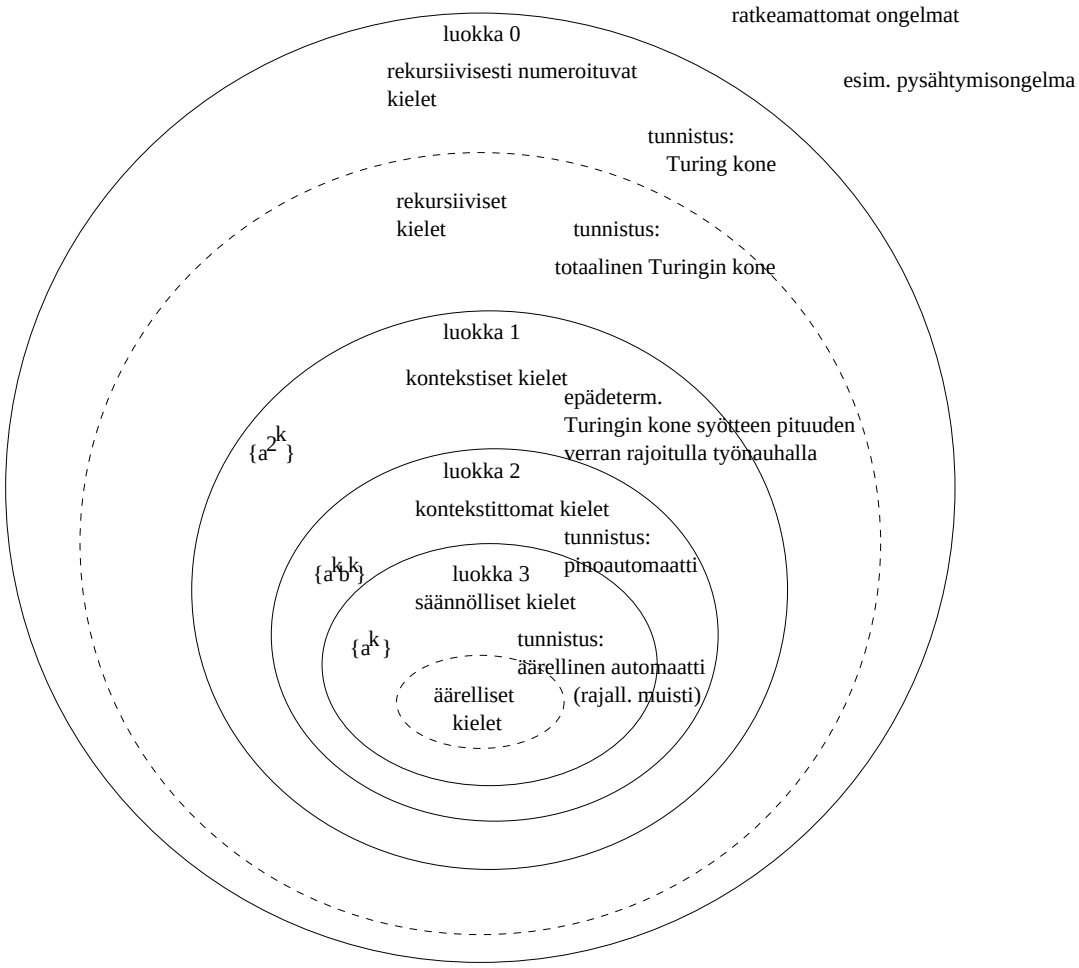
On syytä huomata, että olemme hypänneet kokonaan yhden Chomskyn luokan, nimittäin *kontekstillisten kielten* (luokka 1) yli. Tietojenkäsittelytieteen kannalta kontekstillisten kielten luokkaa ei ole pidetty merkittävänä, sillä kaikki kontekstilliset kielet voidaan tunnistaa Turingin koneiden avulla ja toisaalta törmäämme käytännössä harvoin ongelmiin, jotka olisivat (ts. vastaavat formaalit kielet olisivat) kontekstillisia, mutta eivät rajoittamattomia. Chomsky kuitenkin uskoi luonnollisten kielten kuuluvan juuri tähän luokkaan.

5.1 Churchin-Turingin teesi

Churchin-Turingin teesi: Mikä tahansa mekaanisesti ratkeava ongelma voidaan ratkaista Turingin koneella.

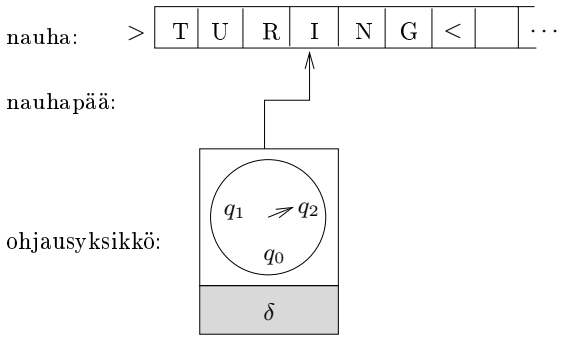
- kaikki algoritmisesti ratkeavat ongelmat
- Esimerkiksi ohjelmointikielet ja RAM-koneet laskentavoimaltaan ekvivalentteja Turingin koneiden kanssa
  - Kaikki ”riittävän vahvat” ohjelmointikielet määrittävät täsmälleen saman ratkeavien ongelmien luokan
  - Perustelu: millä tahansa riittävän vahvalla ohjelmointikielellä voidaan kirjoittaa kääntäjä (oik. tulkkiohjelma) mille tahansa toiselle ohjelmointikielelle
  - Mikä tahansa Turingin kone voidaan toteuttaa tällaisen ohjelmointikielen ohjelmana ja kääntäen (sama pätee RAM-koneelle) (ks. Hopcroft-Motwani-Ullman, luvut 8.6.1 ja 8.6.2)

[3] Huom! Nimityksillä *rekursiivinen* ja *rekursiivisesti numeroituva* ei ole mitään tekemistä tietojenkäsittelytieteilijän ymmärtämän rekursion kanssa. Termien hämäävä alkuperä johtuu matemaatikkojen (Gödel, Kleene) määrittelemistä rekursiivisista ja rekursiivisesti numeroituvista funktioista, joista lisää ekskursiossa.



Kuva 5.1: Chomskyn kielihierarkia täydennettynä rekursiivisten kielten luokalla.

- Huom! Myös kaksipinoiset pinoautomaatit yhtä vahvoja kuin Turingin koneet! (ks. Hopcroft-Motwani-Ullman: teoreema 8.13)
- Chomskyn kielihierarkiassa rekursiivisesti lueteltavat kielet: jos Turingin kone ratkaisee osittain, niin mikään muukaan malli ei kykene ratkaisemaan kuin osittain (ts. joillain syötteillä laskenta jatkuu ikuisesti...).
- Yleensä algoritmiksi kutsutaan vain sellaista algoritmia, jonka laskenta päättyy aina, kaikilla syötteillä – ts. rekursiiviset eli ratkeavat kielet. Ei ole järkeä tehdä ohjelmaa, joka ei koskaan pysähdy!
- Huom! Vain teesi, ei teoreema (lause): kukaan ei ole voinut todistaa oikeaksi,



Kuva 5.2: Turingin kone.

mutta kaikki tunnetut laskentamallit ovat osoittautuneet ekvivalenteiksi ja yleisesti uskotaan pätevän kaikille laskennan malleille.

- Jotkut ovat spekuloineet, olisivatko ns. kvanttitietokoneet vahvempia...

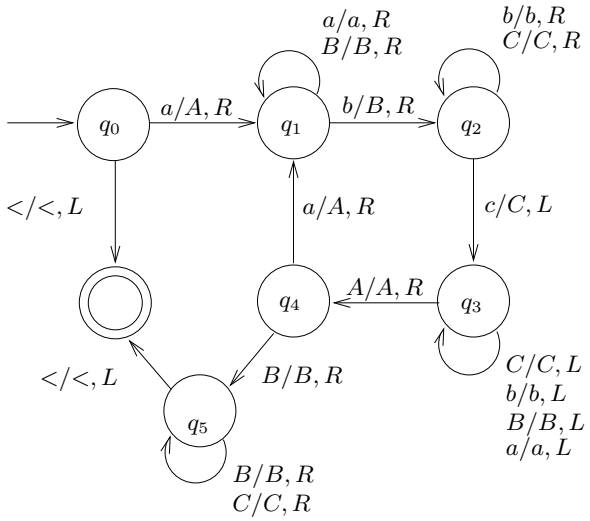
5.2 Turingin koneet

- kuten äärellinen automaatti, jolla toiseen suuntaan ääretön työnauha, jota voi lukea ja kirjoittaa merkki kerrallaan
- Aluksi nauhalla on syötemerkkijono (ja loppu nauhasta tyhjä), nauhapää osoittaa ensimmäistä nauhapään paikkaa ja kone käynnistetään alkutilassa q_0
- Kullakin laskenta-askeleella se lukee nauhapään kohdalla olevan merkin, päättää siirtymäfunktion mukaisesti uuden tilansa, kirjoittaa nauhapään kohdalla uuden merkin ja siirtää nauhapäätä yhden askeleen vasemmalle tai oikealle (ensimmäisen paikan vasemmalle puolelle ei voi kuitenkaan mennä)
- Koneella on hyväksyvä lopputila q_{yes} ja hylkäävä lopputila q_{no} (kun kyseessä on kielen tunnistaminen, jota käsittelemme). Kone pysähtyy, kun se saavuttaa lopputilan.
- Turingin kone eroaa äärellisestä automaatista siten että
  - työnauhalle voidaan kirjoittaa
  - työnauhalla voi liikkua sekä vasemmalle että oikealle
  - työnauha on rajoittamattoman pitkä

– kun lopputila on saavutettu, kone pysähtyy

Esim. kielen $\{a^k b^k c^k | k \geq 0\}$ tunnistava Turingin kone

- Idea: kone pitää kirjaa tapaamistaan a:sta, b:stä ja c:stä muuttamalla ne yksi kerrallaan A:ksi, B:ksi ja C:ksi.
- muutettuaan viimeisen a:n A:ksi tarkistaa, ettei enää jäljellä b:tä tai c:tä.



Kuva 5.3: Kielen $\{a^k b^k c^k \mid k \geq 0\}$ tunnistava Turingin kone.

Huom! Sama voidaan ratkaista kaksipinoisella pinoautomaatilla (harjoitustehtävä) tai tietokoneohjelmana:

```
while ((c=getchar())!=EOF) {
    switch(c) {
        case 'a': A++;
        case 'b': B++;
        case 'c': C++;
        else: ERROR;
    }
}
if ((A==B) && (B==C))
    printf("Ok");
```

5.2.1 Formaali määrittely

Määritelmä: *Turingin kone* on seitsikko

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

missä

- Q on koneen *tilojen* äärellinen joukko;
- Σ on koneen *syöteaakkosto*;
- $\Gamma \supseteq \Sigma$ on koneen *nauha-aakkosto*; ol. että $>, < \notin \Gamma$;
- $\delta : (Q - \{q_{\text{yes}}, q_{\text{no}}\}) \times (\Gamma \cup \{>, <\}) \to Q \times (\Gamma \cup \{>, <\}) \times \{L, R\}$ on koneen *siirtymäfunktio*;
- $q_0 \in Q$ on koneen *alkutila*;
- $q_{\text{yes}} \in Q$ on koneen *hyväksyvä* ja $q_{\text{no}} \in Q$ sen *hylkäävä lopputila*.

Siirtymäfunktion arvoilta

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

vaaditaan:

(i) jos $b = >$, niin $a = >$;

(ii) jos $a = >$, niin $b = >$ ja $\Delta = R$;

(iii) jos $b = <$, niin $a = <$ ja $\Delta = L$.

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

- Koneen *tilanne* on nelikko

$$(q, u, a, v) \in Q \times \Gamma^* \times (\Gamma \cup \{\epsilon\}) \times \Gamma^*,$$

missä voi olla $a = \epsilon$, mikäli myös $u = \epsilon$ tai $v = \epsilon$.

- Tulkinta: 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 $a = \epsilon$, jos nauhapää sijaitsee aivan nauhan alussa tai sen käytetyn osan lopussa. Ensimmäisessä tapauksessa ajatellaan, että kone "havaitsee" merkin '$>$' ja toisessa tapauksessa merkin '$<$'.

- *Alkutilanne syötteellä* $x = a_1 a_2 \ldots a_n$ on nelikko

$$(q_0, \epsilon, a_1, a_2 \ldots a_n).$$

- Tilannetta (q, u, a, v) merkitään yleensä yksinkertaisemmin $(q, u\underline{a}v)$, ja alkutilannetta syötteellä x yksinkertaisesti $(q_0, \underline{x})$.

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

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

jos jokin seuraavista ehdoista täyttyy: kaikilla $q, q' \in Q$, $u, v \in \Gamma^*$, $a, b \in \Gamma$ ja $c \in \Gamma \cup \{\epsilon\}$:

jos $\delta(q, a) = (q', b, R)$, niin $(q, u\underline{a}cv) \underset{M}{\vdash} (q', ub\underline{c}v)$;

jos $\delta(q, a) = (q', b, L)$, niin $(q, uc\underline{a}v) \underset{M}{\vdash} (q', u\underline{c}bv)$;

jos $\delta(q, >) = (q', >, R)$, niin $(q, \underline{\epsilon}cv) \underset{M}{\vdash} (q', \underline{c}v)$;

jos $\delta(q, <) = (q', b, R)$, niin $(q, u\underline{\epsilon}) \underset{M}{\vdash} (q', ub\underline{\epsilon})$;

jos $\delta(q, <) = (q', b, L)$, niin $(q, uc\underline{\epsilon}) \underset{M}{\vdash} (q', u\underline{c}b)$;

jos $\delta(q, <) = (q', <, L)$, niin $(q, uc\underline{\epsilon}) \underset{M}{\vdash} (q', u\underline{c})$.

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

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

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

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

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

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

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

muuten M *hylkää* x:n.

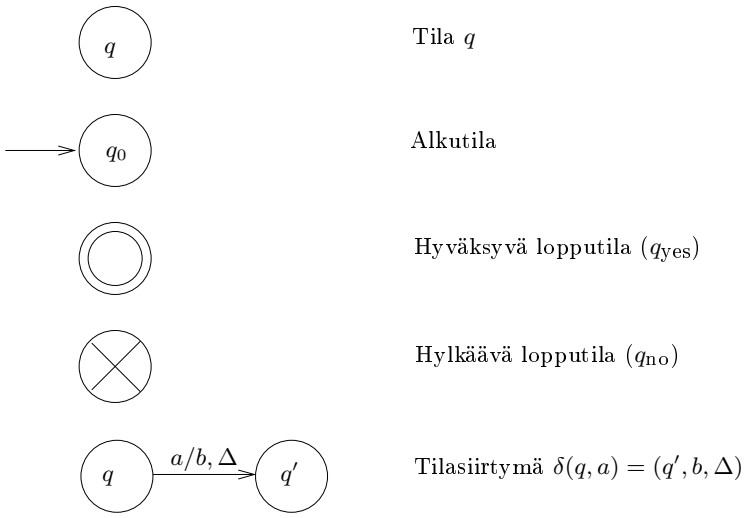
(ts. hyväksymiseen riittää, että pääsemme hyväksyvään lopputilaan, koko merkkijonoa ei välttämättä tarvitse edes lukea! Nauhalle saa luonnollisestikin jäädä merkkejä $\leftrightarrow$ automaatit)

- Koneen M *tunnistama kieli* on:

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

5.2.2 Turingin koneen esitys siirtymäkaaviona

Turingin kone voidaa esittää antamalla sen siirtymäfunktio tai siirtymäkaaviona samaan tapaan kuin äärelliset automaatit.



Kuva 5.4: Turingin koneiden kaavioesityksen merkinnät.

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

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

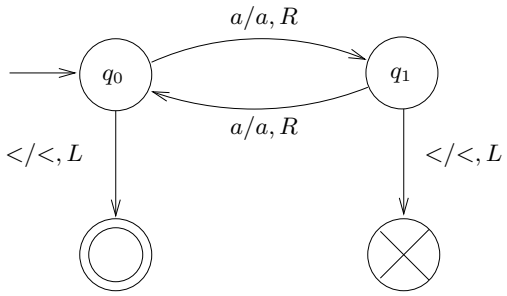
missä

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

Koneen M laskenta esimerkiksi syötteellä aaa etenee seuraavasti:

$$\begin{aligned} (q_0, \underline{a}aa) &\underset{M}{\vdash} (q_1, a\underline{a}a) \underset{M}{\vdash} (q_0, aa\underline{a}) \\ &\underset{M}{\vdash} (q_1, aaa\underline{\epsilon}) \underset{M}{\vdash} (q_{\text{no}}, aa\underline{a}). \end{aligned}$$

Kone pysähtyy tilassa q_{no}, joten $aaa \notin L(M)$.



Kuva 5.5: Kielen $\{a^{2k} \mid k \geq 0\}$ tunnistava Turingin kone.

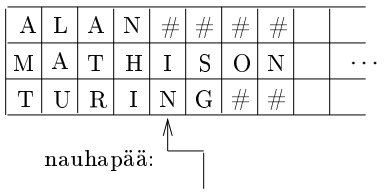
Esimerkki 2: Kielen $\{a^k b^k c^k \mid k \geq 0\}$ tunnistavan koneen laskenta syötteellä *aabbcc*:

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

5.3 Turingin koneiden laajennuksia

Turingin koneista on olemassa monia erilaisia variaatioita. Tässä esitellään kuitenkin vain kolme variaatiota: moniuraiset koneet, joiden työnauha koostuu useasta rinnakkaisesta urasta, moninauhaiset koneet, joilla on useita toisistaan riippumattomia työnauhoja, sekä tärkeimpänä kaikista, epädeterministiset Turingin koneet. Huom! Mikä tahansa näistä variaatioista voidaan esittää standardimallisena Turingin koneena.

5.3.1 *Moniuraiset koneet



Kuva 5.6: Kolmeuraisen Turingin koneen nauha.

- Sallitaan, että Turingin koneen nauha koostuu k:sta rinnakkaisesta urasta, jotka kaikki kone lukee ja kirjoittaa yhdessä laskenta-askelessa.
- Koneen siirtymäfunktion arvot ovat tällöin 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 $\#$.
- Formaalisti *k-urainen Turingin kone* on seitsikko

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

  missä muut komponentit ovat kuten standardimallissa, paitsi siirtymäfunktio:

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

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

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

Todistus.

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

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

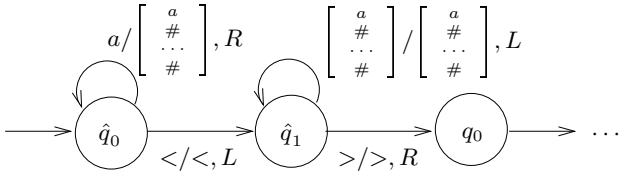
missä $\hat{Q} = Q \cup \{\hat{q}_0, \hat{q}_1, \hat{q}_2\}$, $\hat{\Gamma} = \Sigma \cup \Gamma^k$ ja kaikilla $q \in Q$ on

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

- Koneen $\widehat{M}$ laskennan aluksi täytyy syötejono "nostaa" ykkösuralle, so. korvata nauhalla merkkijono $a_1 a_2 \ldots a_n$ merkkijonolla

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

- Tätä operaatiota varten liitetään M:stä kopioidun siirtymäfunktion osan alkuun vielä pieni "esiprosessori".

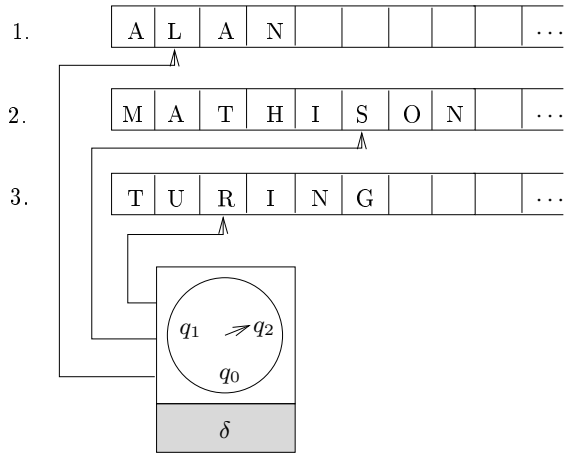


Kuva 5.7: Esiprosessori moniuraisen koneen sytteen nostamiseksi ykksuralle.

□

5.3.2 Moninauhaiset koneet

- Sallitaan, että Turingin koneella on k toisistaan riippumatonta nauhaa, joilla on kullakin oma nauhapäänsä.



Kuva 5.8: Kolmenauhainen Turingin kone.

- Kone lukee ja kirjoittaa kaikki nauhat yhdessä laskenta-askelessa.
- Laskennan aluksi syöte sijoitetaan ykkösnauhan vasempaan laitaan ja kaikki nauhapäät nauhojensa alkuun.
- Tällaisen koneen siirtymäfunktion arvot ovat muotoa

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

  missä $a_1, \ldots, a_k$ ovat nauhoilta $1, \ldots, k$ luetut merkit, $b_1, \ldots, b_k$ niiden tilalle kirjoitettavat merkit, ja $\Delta_1, \ldots, \Delta_k \in \{L, R\}$ nauhapäiden siirtosuunnat.
- Formaalisti *k-nauhainen Turingin kone* on seitsikko

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

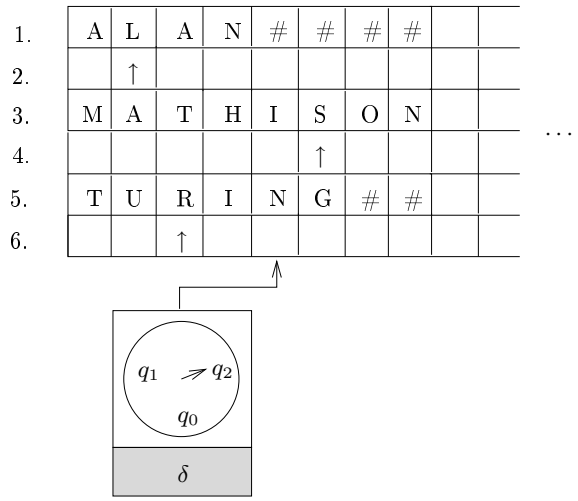
  missä muut komponentit ovat kuten standardimallissa, paitsi siirtymäfunktio:

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

  (Seuraajatilannerelaatio ym. peruskäsitteet määritellään pienin muutoksin entiseen tapaan.)

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

Todistus (idea).



Kuva 5.9: Kolmenauhaisen Turingin koneen simulointi kuusiuraisella.

- Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ k-nauhainen Turingin kone, joka tunnistaa kielen L.

- Konetta M voidaan simuloida $2k$-uraisella koneella $\widehat{M}$ siten, että koneen $\widehat{M}$ parittomat urat $1, 3, 5, \ldots, 2k-1$ vastaavat M:n nauhoja $1, 2, \ldots, k$, ja kutakin paritonta uraa seuraavalla parillisella uralla on merkillä $\uparrow$ merkitty vastaavan nauhan nauhapään sijainti.

- Simuloinnin aluksi syötemerkkijono sijoitetaan normaalisti koneen $\widehat{M}$ ykkösuralle. 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.
  - Takaisin oikealta vasemmalle suuntautuvalla pyyhkäisyllä kone kirjoittaa $\uparrow$-osoittimien kohdalle asianmukaiset uudet merkit ja siirtää osoittimia. □

5.3.3 Epädeterministiset koneet

Määritelmä: Epädeterminististinen Turingin kone on seitsikko

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

missä muut komponentit ovat kuten deterministisessä standardimallissa, paitsi siirtymäfunktio:

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

- Siirtymäfunktion arvon

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

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

- Epädeterministisen koneen tilanteet, tilannejohdot jne. määritellään formaalisti samoin kuin deterministisenkin koneen tapauksessa, paitsi että ehdon $\delta(q, a) = (q', b, \Delta)$ sijaan kirjoitetaan $(q', b, \Delta) \in \delta(q, a)$.

- Tämän muutoksen takia seuraajatilannerelaatio $\underset{M}{\vdash}$ ei ole enää yksiarvoinen: koneen tilanteella (q, w) voi nyt olla useita vaihtoehtoisia seuraajia, so. tilanteita (q', w'), joilla $(q, w) \underset{M}{\vdash} (q', w')$.

- Koneen M tunnistama kieli määritellään:

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

- Epädeterministisen koneen M tapauksessa siis merkkijono x kuuluu M:n tunnistamaan kieleen, jos *jokin* M:n kelvollinen tilannejono johtaa alkutilanteesta syötteellä x hyväksyvään lopputilanteeseen.

Esimerkki: yhdistettyjen lukujen "tunnistaminen" epädeterministisillä Turingin koneilla.

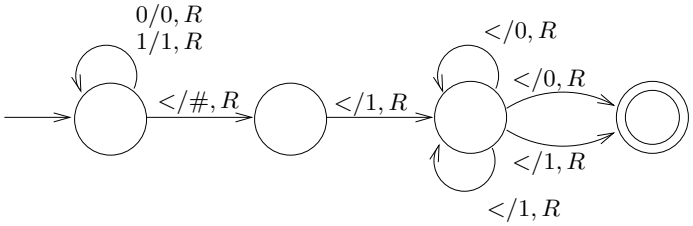
Ei-negatiivinen kokonaisluku n on *yhdistetty*, jos sillä on kokonaislukutekijät $p, q \geq 2$, joilla $pq = n$. Luku, joka ei ole yhdistetty, on *alkuluku*.

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

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

Olkoon lisäksi GO_START deterministinen Turingin kone, joka siirtää nauhapään osoittamaan nauhan ensimmäistä merkkiä.

Olkoon edelleen GEN_INT mielivaltaisen ykköstä suuremman binääriluvun nauhan loppuun tuottava epädeterminstinen Turingin kone.

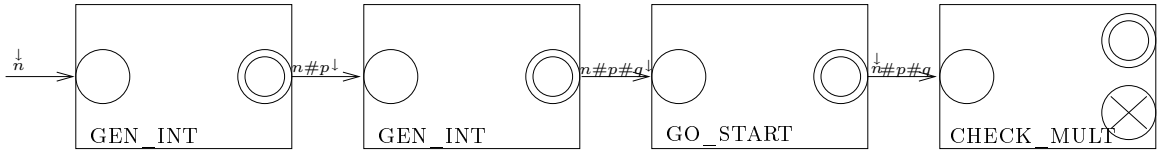


Kuva 5.10: Epädeterministinen Turingin kone GEN_INT.

Epädeterministinen Turingin kone TEST_COMPOSITE, joka tunnistaa kielen

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

voidaan muodostaa näistä komponenteista yhdistämällä.



Kuva 5.11: Epädeterministinen Turingin kone TEST_COMPOSITE.

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.

Deterministisen ja epädeterministisen Turingin koneen ekvivalenssi

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

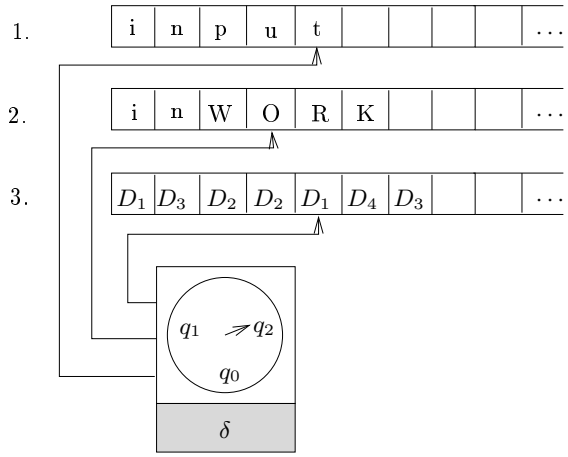
Todistus (idea).

- Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ epädeterministinen Turingin kone, joka tunnistaa kielen L.
- Konetta M voidaan simuloida 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 nauhalle 2 edellisen laskennan jäljiltä mahdollisesti jääneet merkit.
- Nauhalla 3 $\widehat{M}$ pitää kirjaa vuorossa olevan laskennan "järjestysnumerosta". Tarkemmin sanoen, olkoon r suurin M:n siirtymäfunktion arvojoukon koko. Tällöin $\widehat{M}$:lla on erityiset nauhamerkit $D_1, \ldots, D_r$, joista koostuvia jonoja se generoi nauhalle 3 kanonisessa järjestyksessä (λ, D_1, D_2, $\ldots$, D_r, D_1D_1, D_1D_2, $\ldots$, D_1D_r, D_2D_1, $\ldots$).
- Kutakin generoitua jonoa kohden $\widehat{M}$ simuloi yhden M:n osittaisen laskennan, jossa epädeterministiset valinnat tehdään kolmosnauhan koodijonon ilmaisemalla tavalla.
- Esimerkiksi jos kolmosnauhalla on jono $D_1D_3D_2$, niin ensimmäisessä siirtymässä valitaan vaihtoehto 1, toisessa vaihtoehto 3, kolmannessa vaihtoehto 2. Ellei tämä laskenta johtanut M:n hyväksyvään lopputilaan, generoidaan seuraava koodijono $D_1D_3D_3$ ja aloitetaan alusta.
- Jos koodijono on epäkelpo, so. jos siinä jossakin kohden on tilanteeseen liian suuri koodi, simuloitu laskenta keskeytetään ja generoidaan seuraava jono.

- Selvästi 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. □



Kuva 5.12: Epädeterministisen Turingin koneen simulointi deterministisellä

5.4 Rajoittamattomat ja kontekstiset kieliopit

Rajoittamattomat kieliopit (*unrestricted grammars*) eli yleiset muunnossysteemit (*string rewriting systems*)

- yleistetään kontekstittomia kielioppeja sallimalla produktiossa yhden välikkeen sijaan minkä tahansa välikkeistä ja päätteistä koostuvan merkkijonon korvaaminen toisella (mahd. tyhjällä merkkijonolla)
- esim. $aB \to BC|d$ ja $a \to B|\epsilon$ ovat nyt sallittuja sääntöjä, mutta $\epsilon \to A$ ei ole.
- säännöt siis muotoa $\omega \to \omega'$, missä $\omega, \omega' \in V^*$ (V koko aakkosto) ja $\omega \neq \epsilon$
- Tärkeä tulos: **Kieli voidaan tuottaa rajoittamattomalla kieliopilla jos ja vain jos se voidaan tunnistaa Turingin koneella.**

Määritelmä: 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 ($V^+ = V^* - \{\epsilon\}$);
- $S \in N$ 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, merkitään 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 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.:
$$L(G) = \{x \in \Sigma^* \mid S \underset{G}{\Rightarrow}^* x\}.$$

Esimerkki: rajoittamaton kielioppi ei-kontekstittomalle kielelle $\{a^k b^k c^k \mid k \geq 0\}$.

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

Esimerkiksi lauseen $aabbcc$ johto:

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

Lause: 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. Muodostetaan kielen L tunnistava kaksinauhainen epädeterministinen Turingin kone M_G seuraavasti:

- 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 kieliopin lähtösymbolin S.
- Koneen M_G laskenta koostuu vaiheista. Kussakin vaiheessa kone:

  (i) vie kakkosnauhan nauhapään epädeterministisesti johonkin kohtaan nauhalla;

(ii) valitsee epädeterministisesti jonkin G:n produktion, jota yrittää soveltaa valittuun nauhankohtaan (produktiot on koodattu M_G:n siirtymäfunktioon);

(iii) jos produktion vasen puoli sopii yhteen nauhalla olevien merkkien kanssa, M_G korvaa ao. merkit produktion oikean puolen merkeillä;

(iv) 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 (i)). □

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

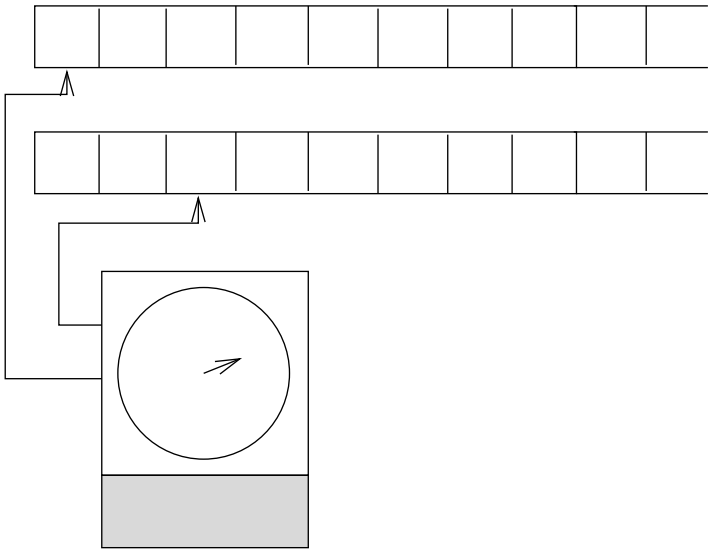
**Todistus.* Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ kielen L tunnistava standardimallinen Turingin kone. Muodostetaan kielen L tuottava rajoittamaton kielioppi G_M seuraavasti.

Idea:

- Kieliopin G_M välikkeiksi otetaan (muiden muassa) kaikkia M:n tiloja $q \in Q$ edustavat symbolit.
- Koneen M tilanne $(q, u\underline{a}v)$ esitetään merkkijonona $[uqav]$.
- M:n siirtymäfunktion perusteella G_M:ään muodostetaan produktiot, joiden ansiosta
$$[uqav] \underset{G_M}{\Rightarrow} [u'q'a'v'] \quad \text{joss} \quad (q, uav) \underset{M}{\vdash} (q', u'a'v').$$
- Siten M hyväksyy syötteen x, jos ja vain jos
$$[q_0 x] \underset{G_M}{\Rightarrow}{}^* [u q_{\text{yes}} v]$$
joillakin $u, v \in \Sigma^*$.

Kaikkiaan kielioppiin G_M tulee kolme ryhmää produktioita:

1. Produktiot, joilla lähtösymbolista S voidaan tuottaa mikä tahansa merkkijono muotoa $x[q_0x]$, missä $x \in \Sigma^*$ ja $[$, q_0, $]$ ja ovat G_M:n välikkeitä.
2. Produktiot, joilla merkkijonosta $[q_0x]$ voidaan tuottaa merkkijono $[uq_{\text{yes}}v]$, jos ja vain jos M hyväksyy x:n.
3. Produktiot, joilla muotoa $[uq_{\text{yes}}v]$ oleva merkkijono muutetaan tyhjäksi merkkijonoksi.



Kuva 5.13: Rajoittamattoman kieliopin tuottaman kielen tunnistaminen Turingin koneella.

Kieleen $L(M)$ kuuluvan merkkijonon x tuottaminen tapahtuu tällöin seuraavasti:

$$S \overset{(1)}{\Rightarrow}^* x[q_0 x] \overset{(2)}{\Rightarrow}^* x[u q_{\text{yes}} v] \overset{(3)}{\Rightarrow}^* x.$$

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

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

ja produktiot P muodostuvat seuraavista kolmesta ryhmästä:

1. Alkutilanteen tuottaminen:

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

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

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

3. Lopputilanteen siivous:

q_{yes}	$\rightarrow$	$E_L E_R$	
$q_{\text{yes}}[$	$\rightarrow$	E_R	
aE_L	$\rightarrow$	E_L	$(a \in \Gamma)$
$[E_L$	$\rightarrow$	ϵ	
$E_R a$	$\rightarrow$	E_R	$(a \in \Gamma)$
$E_R]$	$\rightarrow$	ϵ	

□

Esim. Tarkastellaan seuraavaa kasvikielioppia, johon on lisätty lähtösymboli S ja sille säännöt, jotta on saatu tavallinen rajoittamaton kielioppi. Kieliopin aakkosto on $V = \{S, SIEMEN, SIRKKALEHDET, LEHDET, VARSI, NUPPU, PUNKUKKA, SINKUKKA, PAIVA, YO\}$. (Huom! Koska tämä on kasvikielioppi, emme erottele pääte- ja välikesymboleja – ”jäsennys” jatkuu ikuisesti, ellei koko populaatio satu kuolemaan.)

$S \rightarrow$ SIEMEN PAIVA | SIEMEN YO
SIEMEN YO$\rightarrow$ SIRKKALEHDET YO
SRKKALEHDET PAIVA$\rightarrow$ LEHDET PAIVA| VARSI PAIVA
LEHDET PAIVA$\rightarrow$ VARSI PAIVA
VARSI PAIVA$\rightarrow$ NUPPU PAIVA| LEHDET PAIVA
NUPPU YO$\rightarrow$ PUNKUKKA YO| SINKUKKA YO
PUNKUKKA $\rightarrow$ SIEMEN | SIEMEN SIEMEN |ϵ
SINKUKKA $\rightarrow$ SIEMEN | SIEMEN SIEMEN |ϵ
PAIVA $\rightarrow$ YO
YO $\rightarrow$ PAIVA

Nyt lähtösymbolista S voidaan johtaa esim. seuraavat lausejohdokset:

$S \Rightarrow$ SIEMEN PAIVA$\Rightarrow$ SIEMEN YO $\Rightarrow$ SIRKKALEHDET YO $\Rightarrow$ SIRKKALEHDET PAIVA $\Rightarrow$ LEHDET PAIVA $\Rightarrow$ VARSI PAIVA $\Rightarrow$ NUPPU PAIVA $\Rightarrow$ NUPPU YO $\Rightarrow$ PUNKUKKA YO $\Rightarrow$ PUNKUKKA PAIVA $\Rightarrow$ PUNKUKKA YO $\Rightarrow$ SIEMEN SIEMEN YO $\Rightarrow$ ...

Kontekstiset kieliopit (*context-sensitive grammars*)

- Rajoittamaton kielioppi on *kontekstinen*, jos sen kaikki produktiot ovat muotoa $\omega \rightarrow \omega'$, missä $|\omega'| \geq |\omega|$, tai mahdollisesti $S \rightarrow \epsilon$, missä S on lähtösymboli.
- ts.lavennettaessa merkkijono voi vain kasvaa, ei koskaan pienentyä, paitsi lähtösymboli, joka saa tuottaa ϵ:in, jos ϵ kuuluu kieleen
- Lisäksi vaaditaan, että jos kieliopissa on produktio $S \rightarrow \epsilon$, niin lähtösymboli S ei esiinny minkään produktion oikealla puolella.
- Kontekstilliset kielet ovat siis rajoittamattomien kielten osaluokka!
- esim. säännöt
  tietojenkäsittelijä $\rightarrow$ *tietojenkäpistelijä*, *tieto* $\rightarrow$ *taito*
  $SUBJ$ *on* $PREDIKAT \rightarrow SUBJ$ *oli* $PREDIKAT | SUBJ$ *on ollut* $PREDIKAT |$
  $SUBJ$ *oli ollut* $PREDIKAT$

  ovat kontekstillisia
- Formaali kieli L on *kontekstinen*, jos se voidaan tuottaa jollakin kontekstisella kieliopilla.
- Normaalimuotolause: produktiot saadaan muotoon $S \rightarrow \epsilon$ ja $\alpha A \beta \rightarrow \alpha \omega \beta$, missä A on välike ja $\omega \neq \epsilon$. (Säännön $A \rightarrow \omega$ sovellus "kontekstissa" $\alpha \_ \beta$.)
- esim. $PAIVA\ SIEMEN \rightarrow PAIVA\ SIRKKALEHTI$, $PAIVA \rightarrow YO$: nyt sääntöä $SIEMEN \rightarrow SIRKKALEHTI$ saa soveltaa vain kontekstissa $PAIVA\_\epsilon$ (vain α muodostaa siis kontekstin ja β puuttuu)

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

Todistusidea: kontekstillisen kieliopin mukaisessa jäsennyksessä lähtösymboli voi vain kasvaa joka jäsennysaskeleella, joten jos lähdetään merkkijonosta kohti lähtösymbolia, voi symbolijono vain pienentyä joka jäsennysaskeleella. □

- Lauseen koneita sanotaan *lineaarisesti rajoitetuiksi automaateiksi.*
- Lineaarisesti rajoitetut automaatit tunnistavat siis täsmälleen kontekstilliset kielet.
- Avoin ongelma ("LBA = DLBA"): onko lauseessa vaadittu epädeterminismi välttämätöntä vai riittäisikö deterministinen kone?

Esimerkki Tiedämme, että kielen $L = \{a^k b^k c^k \mid k \geq 0\}$ voi tunnistaa syötteen vaatimassa tilassa (muutetaan merkkejä A:ksi, B:ksi ja C:ksi yksi kerrallaan), joten se on kontekstillinen. Voidaan antaa seuraava kielen kuvaava kontekstillinen kielioppi:

$S' \to S|\epsilon$
$S \to aSBc|abc$
$cB \to Bc$
$bB \to bb$

Esim. merkkijonon *aaabbbccc* johto:
$\underline{S} \Rightarrow a\underline{S}Bc \Rightarrow aa\underline{S}BcBc \Rightarrow aaab\underline{cB}cBc \Rightarrow aaa\underline{bB}ccBc \Rightarrow aaabbc\underline{cB}c \Rightarrow aaabbc\underline{cB}cc \Rightarrow aaab\underline{bB}ccc \Rightarrow aaabbbccc$

Luku 6

Ratkeavuus ja ratkeamattomuus

eli Rekursiiviset, Rekursiivisesti lueteltavat ja kieliluokkien ulkopuolelle jäävät kielet

Palautetaan vielä mieliin seuraavat käsitteet:

- Turingin kone $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ on *totaalinen*, jos se pysähtyy kaikilla syötteillä
- Formaali kieli A on *rekursiivisesti lueteltava*, jos se voidaan tunnistaa jollakin Turingin koneella, ja
- *rekursiivinen*, jos se voidaan tunnistaa jollakin totaalisella Turingin koneella.
- Päätösongelma π on *ratkeava*, jos sitä vastaava formaali kieli A_π on rekursiivinen, ja
- päätösongelma on *osittain ratkeava*, jos A_π on rekursiivisesti lueteltava.
- Ongelma, joka ei ole ratkeava, on *ratkeamaton*. (*Huom.*: ratkeamaton ongelma voi siis olla osittain ratkeava.)

Analogia:

Oletetaan että universumi on ääretön ja siinä on äärettömän monta tähteä. Kuvitellaan, että sinulla olisi ääretön tietokanta, joka sisältää tiedot kaikkien tähtien koordinaateista. Olet kiinnostunut tähtien naapureista, jotka olet määritellyt seuraavasti:

Tähdet A ja B ovat naapureita, jos A:n ja B:n etäisyys on alle R valovuotta eli $\Delta(A, B) \leq R$.

Tarkastellaan seuraavia ongelmia:

i) Onko Koirantähti Kissantähden naapuri? Kissantähden kordinaatit ovat (x, y, z) ja Koirantähden koordinaatit (r, s, t). Nyt lasketaan vain etäisyys $\sqrt{(x-r)^2 + (y-s)^2 + (z-t)^2}$ eli ongelma on selvästi ratkeava. (Vastaava "kieli" olisi $L_{Kissa} = \{x | x$ on tähden koodi ja x on Kissantähden naapuri$\}$. Ongelma palautuu siis "kielentunnistusongelmaksi" Koirantähti $\in L_{Kissa}$?)

ii) Onko Kanintähdellä yhtään naapuria? Olk. Kanintähden koordinaatit (x, y, z). Nyt käytävä pahimmassa tapauksessa koko tietokanta läpi. Kuitenkin jos Kanintähdellä on naapuri, esim. tietokannan i:s tähti, löytyy se siis äärellisessä ajassa (i:nnellä aikaskeleella). Sen sijaan emme saa koskaan varmuutta, mikäli Kanintähdellä ei ole naapuria. Ongelma ratkeaa siis vain "Kyllä"-tapauksessa, mutta ei "Ei"-tapauksessa ja on siis osittain ratkeava. (Haluamme siis ratkaista "kieltä" $L_{Kani} = \{x | x$ on tähden koodi ja x on Kanintähden naapuri$\}$ koskevan ongelman $L_{Kani} \neq \emptyset$?)

iii) Onko Kanintähti yksinäinen tähti? Tämän ongelman "Kyllä"-vastaus on sama kuin edellisen ongelman "Ei"-vastaus eli ratkeamaton. Meidän täytyisi tarkistaa kaikki tietokannan tähdet, joita on äärettömän monta, eli laskenta ei pääty koskaan. (Haluamme siis tietää, onko $L_{Kani} = \emptyset$?)

Täysin ratkeamattomia ongelmia ei ole enää luokiteltu laskennallisen vaikeutensa perusteella, vaikka voisi ajatella, että seuraava ongelma olisi vieläkin vaikeampi: Luettele kaikki universumin yksinäiset tähdet. ("Kielenä" $L_{yksin} = \{x | x$ on tähden koodi ja $L_x = \emptyset\}$, missä L_x sisältää x:n kaikki naapurit.)

6.1 Ratkeavat ongelmat eli rekursiivinen kieli

Jos ratkeavasta ongelmasta muodostetaan komplementtiongelma tai kahdesta ratkeavasta ongelmasta muodostetaan yhdiste tai leikkaus, on tulosongelma edelleen ratkeava. Ts. rekursiivisille kielille pätevät seuraavat sulkeumaominaisuudet:

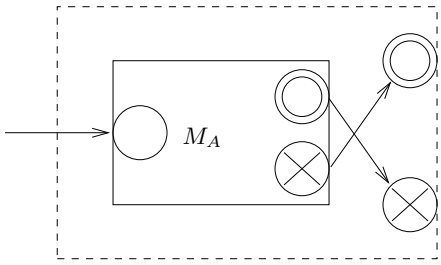
Lause: Olkoot $A, B \subseteq \Sigma^*$ rekursiivisia. Tällöin myös

i) $\bar{A} = \Sigma^* - A$,
ii) $A \cup B$ ja
iii) $A \cap B$

ovat rekursiivisia.

Todistus.

(i)Olkoon M_A totaalinen Turingin kone, joka tunnistaa kielen A (ts. $L(M_A) = A$). Kielen $\bar{A}$ tunnistava totaalinen Turingin kone saadaan vaihtamalla M_A:n hyväksyvä ja hylkäävä lopputila keskenään.



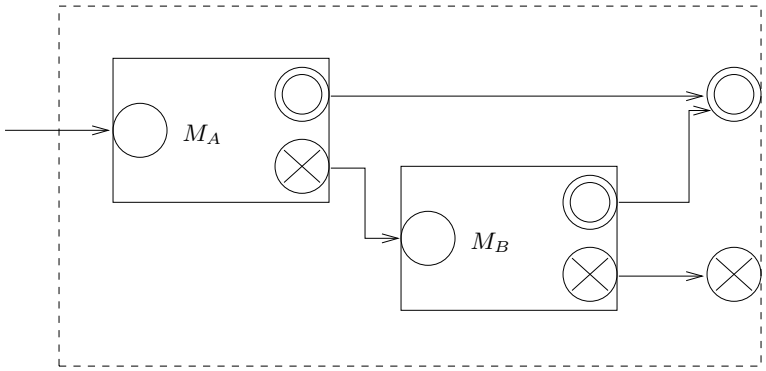
Kuva 6.1: Rekursiivisen kielen komplementin tunnistaminen Turingin koneella.

(ii)Olkoot M_A ja M_B totaaliset Turingin koneet, jotka tunnistavat kielet A ja B (ts. $L(M_A) = A$, $L(M_B) = B$). Kielen $A \cup B$ tunnistava totaalinen Turingin kone M saadaan yhdistämällä M_A ja M_B toimimaan peräkkäin: jos M_A hyväksyy syötteen, myös M hyväksyy; jos M_A päätyy hylkäämiseen, M kokeillaan vielä M_B:tä.

Huom! Tätä varten M_A:n jätettävä pysähtyessää nauhalle alkuperäinen syötejono koskemattomana (ja siivottava omat jälkensä).

(iii) $A \cap B$:lle saadaan totaalinen tunnistajakone de Morganin avulla: $A \cap B = \overline{\bar{A} \cup \bar{B}}$. Ts. muodostetaan ensin komplementtikoneet $M_{\bar{A}}$ ja $M_{\bar{B}}$, yhdistetään ne koneeksi $M_{\bar{A} \cup \bar{B}}$ ja muodostetaan vielä tämän komplementtikone $M_{\overline{\bar{A} \cup \bar{B}}}$. □

(Tehtävä: Piirrä kone $M_{\overline{\bar{A} \cup \bar{B}}}$!)



Kuva 6.2: Kahden rekursiivisen kielen yhdisteen tunnistaminen Turingin koneella.

6.2 Osittain ratkeavat ongelmat eli rekursiivisesti numeroituva kieli

Jos meillä on Turingin koneet (jotka pysähtyvät "Kyllä"-tapauksessa, mutta eivät välttämättä "Ei"-tapauksessa) M_A ja M_B, niin voimme muodostaa myös niiden yhdiste- ja leikkauskoneet $M_{A\cup B}$ ja $M_{A\cap B}$. Ts. rekursiivisesti numeroituville kielille pätevät seuraavat sulkeumaominaisuudet:

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

Todistus: H.T. $\square$

Huom! Rekursiivisesti numeroituville kielille ei päde, että komplementtikieli olisi (välttämättä) rekursiivisesti lueteltava! Ts. jos meillä on Turingin kone M_A, joka tunnistaa kielen A ja pysähtyy "Kyllä"-tapauksessa, mutta ei välttämättä "Ei"-tapauksessa, niin emme voi muodostaa konetta $M_{\overline{A}}$, joka pysähtyisi "Kyllä"-tapauksessa kaikilla syötteillä. Miksi?

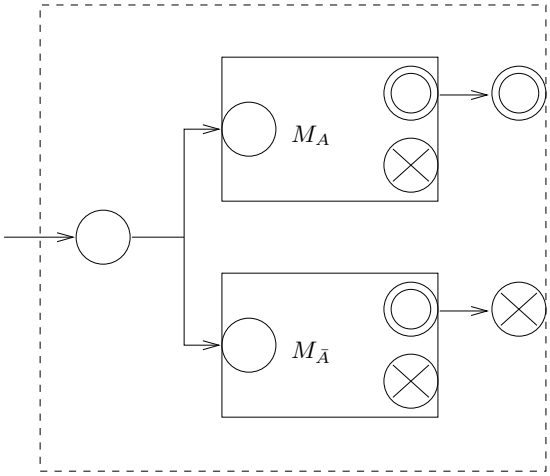
Seuraava tärkeä tulos antaa meille keinon tunnistaa, milloin ongelma on ratkeava:

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

Todistus. "$\Rightarrow$": seuraa lauseesta 6.1(i).

"$\Leftarrow$": Olkoot M_A ja $M_{\bar{A}}$ Turingin koneet kielten A ja $\bar{A}$ tunnistamiseen. Kaikilla $x \in \Sigma^*$ *joko* M_A *tai* $M_{\bar{A}}$ *pysähtyy* ja hyväksyy x:n. Muodostetaan M_A ja $M_{\bar{A}}$ "rinnakkain" yhdistämällä totaalinen kaksinauhainen tunnistajakone M: M 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. □



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

Huom! Voidaan toteuttaa 2-nauhaisena koneena, joka rinnakkaisesti simuloi toisella nauhallaan M_A:n ja toisella M_B:n laskentaa. Jos M_A pysähtyy hyväksyvään tilaan, niin M hyväksyy syötteen, jos taas M_B hyväksyy syötteen, niin M hylkää sen.

⇒ *Seuraus:* Jos $A \subseteq \Sigma^*$ on rekursiivisesti lueteltava kieli, joka ei ole rekursiivinen, niin kieli $\bar{A}$ ei ole rekursiivisesti lueteltava (ts. ongelma on täysin ratkeamaton)!.

6.3 Ratkeamattomuus

Cantorin diagonaaliargumentin perusteella tiedämme, että on olemassa ratkeamattomia ongelmia. Turingin koneiden avulla päättely olisi seuraava:

1) Turingin koneet ovat äärellisiä
2) Siispä voimme luetella kaikki Turingin koneet (ts. Turingin koneiden joukko on numeroituvasti ääretön)
3) Kaikkien ongelmian joukko ei voi olla numeroituva (Diag. arg.: kaikkien päätösongelmien ts. kielentunnistusongelmien joukko on ylinumeroituva.)

⇒ On olemassa onmgelmia, joille ei ole olemassa ratkaisevaa Turingin konetta (eikä siis mitään muutakaan laskennallista ratkaisua Churchin-Turingin teesin mukaan).

- Haluamme konkreettisen esimerkin täysin ratkeamattomasta ongelmasta
- Rajoittamattomille kielille ei ole olemassa omaa Pumppauslemmaa, mutta voimme silti käyttää diagonalisaatioargumenttia (Valehtelijan paradoksia) sellaisen konstruoimiseksi.
- Idea:
  1. Tehdään vastaväite, että Turingin koneet kykenevät ratkaisemaan mitä tahansa – siis myös itseään koskevia kysymyksiä.
  2. Muodostetaan Turingin kone, joka pysähtyy, jos ei pysähdy, ja ei pysähdy, jos pysähtyy.
  3. Saadaan RR. eli oletus ei päde.
- Ensin täytyy määritellä Turingin kone, joka kykenee tutkimaan Turingin koneiden ominaisuuksia (ts. tulkitsee TM:ien "koodeja" ja "ajaa" niitä). Tällaista Turingin konetta kutsutaan *universaalikoneeksi*.

6.4 Universaalikone ja universaalikieli

- Universaalikone
  - saa syötteenään koneen M koodin ja tämän syötteen w
  - pysähtyy vain jos M pysähtyy syötteellä w
  - tulostaa saman, mitä M tulostaa syötteellä w
- Keksittävä tapa koodata Turingin koneita siten, että
  - voimme esittää minkä tahansa TM:n äärellisen aakkoston koodina
  - universaalikone kykenee purkamaan koodin
- riittää koodata TM:n siirtymäfunktio – se kuvaa koneen toiminnan (ei edes haittaa, vaikka tilat nimettäisiin uudelleen)!

6.4.1 Turingin koneen koodaus kokonaislukuna (bittijonona)

Tarkastellaan standardimallista Turingin konetta $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$, jonka syöteaakkosto on $\Sigma = \{0, 1\}$.

Olk. $Q = \{q_0, q_1, \ldots, q_n\}$, missä $q_{\text{yes}} = q_{n-1}$ ja $q_{\text{no}} = q_n$.

$\Gamma \cup \{>, <\} = \{a_0, a_1, \ldots, a_m\}$, missä $a_0 = 0$, $a_1 = 1$, $a_2 = >$ ja $a_3 = <$.

Merk. $\Delta_0 = L$ ja $\Delta_1 = R$.

Määritellään kaikille näille omat koodit, jotta päästään koodaamaan siirtymäfunktiota.

Kooditaulu:

Tilat	q_0	0
	q_1	00
	q_2	000
	.	.
	.	.
	.	.
	$q_{n-1} = q_{yes}$	0^n
	$q_n = q_{no}$	0^{n+1}
Merkit $\Gamma \cup \{>, <\}$	0	0
	1	00
	$>$	000
	$<$	0000
	a_2	$00000 = 0^5$
	a_3	0^6
	.	.
	.	.
	.	.
	a_m	0^{m+1}
Suunnat	L	0
	R	00

Jokainen siirtymäfunktion δ arvoista voidaan nyt koodata seuraavasti:

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

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

ts. $c_{ij} = 1(q_i$:n koodi$)1(a_j$:n koodi$)1(q_r$:n koodi$)1(a_s$:n koodi$)1(\Delta_t$:n koodi$)$

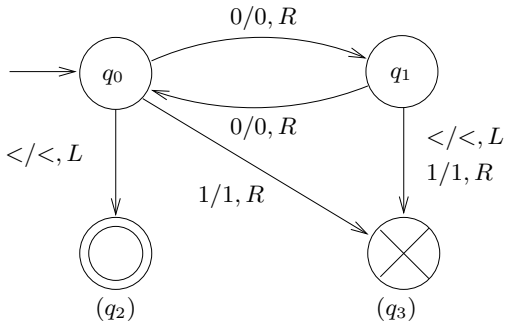
Koko koneen M koodi saadaan yhdistämällä kaikkien siirtymäfunktion arvojen koodit yhdeksi binäärijonoksi, käyttäen erotinmerkkeinä 11:tä ja lisäksi kodin alkuun ja lopuun jonot 111.

Koneen M koodi on siis

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

Esim kielen $\{0^{2k} \mid k \geq 0\}$ tunistava kone: tunnistavan koneen koodi olisi:

$$c_M = 111\underbrace{01010010100}_{\delta(q_0,0)=(q_1,0,R)}11\underbrace{0100100001001 00}_{\delta(q_0,1)=(q_3,1,R)}11\ldots$$



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

- Jokaista Turingin konetta M vastaa koodi c_M
- Jokaiseen binäärijonoon c voidaan liittää jokin Turingin kone M_c.
- Huom! Binäärijonoihin, jotka eivät ole edellisen koodauksen mukaisia Turingin koneiden koodeja, liitetään jokin triviaali, kaikki syötteet hylkäävä (eli $\emptyset$ - kielen tunnistava) kone M_{triv}.

  Määritellään siis:

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

- $\Rightarrow$ Saadaan luettelo kaikista aakkoston $\{0,1\}$ Turingin koneista, ja epäsuorasti myös kaikista aakkoston $\{0,1\}$ rekursiivisesti lueteltavista kielistä.
- Koneet ovat

$$M_\epsilon, M_0, M_1, M_{00}, M_{01}, \ldots,$$

- vastaavat kielet ovat

$$L(M_\epsilon), L(M_0), L(M_1), L(M_{00}), L(M_{01}), \ldots$$

(indeksit kanonisessa järjestyksessä).

- Huom! Kukin kieli voi esiintyä luettelossa monta kertaa (saman kielen voi tunnistaa useilla koneilla).

6.4.2 Esimerkki ei-rekursiivisesti numeroutuvasta kielestä

Nyt voidaan muodostaa e-rekursiivisesti numeroituva kieli (ts. ratkeamaton ongelma) diagonalisaatiotekniikalla:

Lemma: Kieli

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

ei ole rekursiivisesti lueteltava.

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

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

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

- Kieltä D vastaava päätösongelma: "Hylkääkö annetun koodin c esittämä Turingin kone syötteen c?"

- Kielen D muodostaminen kuvallisesti: jos kielten $L(M_\epsilon)$, $L(M_0)$, $L(M_1)$, ... karakteristiset funktiot esitetään taulukkona, niin kieli D poikkeaa kustakin kielestä taulukon diagonaalilla:

D ↘	$L(M_\epsilon)$	$L(M_0)$	$L(M_1)$	$L(M_{00})$	$\cdots$
ϵ	$\not{\overset{1}{0}}$	0	0	0	$\cdots$
0	0	$\not{\overset{0}{1}}$	1	0	$\cdots$
1	0	0	$\not{\overset{0}{1}}$	1	$\cdots$
00	0	0	0	$\not{\overset{1}{0}}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

6.4.3 Universaalikielen ratkeamattomuus

- Aakkoston $\{0,1\}$ *universaalikieli* U määritellään:

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

- Olkoon A jokin aakkoston $\{0,1\}$ rekursiivisesti lueteltava kieli, ja olkoon M kielen A tunnistava standardimallinen Turingin kone. Tällöin on

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

- Ts. universaalikieli sisältää kaikki sellaiset konekoodi-syöte-parien muodostamat merkkijonot, missä kyseinen kone hyväksyy kyseisen syötteen.
- Siihen on siis tallennettu tieto kaikesta, mitä Turingin koneet voivat ratkaista!
- Myös kieli U on rekursiivisesti lueteltava. Kielen U tunnistavia Turingin koneita sanotaan *universaaleiksi Turingin koneiksi.*
- Sen sijaan U ei ole rekursiivinen (eikä sen komplementtikieli $\overline{U}$ siis rekursiivisesti lueteltava)
  $\Rightarrow$ emme voi tunnistaa täysin ratkeamattomia ongelmia laskennallisesti!

Lause: Kieli U on rekursiivisesti lueteltava.

**Todistus.*

- Muodostetaan 3-nauhainen kone universaalikone M_U, joka tunnistaa kielen U
- Laskennan aluksi tarkastettava syöte sijoitetaan koneen M_U ykkösnauhan alkuun.
- Tämän jälkeen kone toimii seuraavasti:

  1. Aluksi M_U tarkastaa, että syöte on muotoa cw, missä c on kelvollinen Turingin koneen koodi. Jos syöte ei ole kelvollista muotoa, M_U hylkää sen; muuten se kopioi merkkijonon $w = a_1 a_2 \ldots a_k \in \{0,1\}^*$ kakkosnauhalle muodossa
  $$00010^{a_1+1}10^{a_2+1}1 \ldots 10^{a_k+1}10000.$$
  2. Jos syöte on muotoa cw, missä $c = c_M$ jollakin koneella M, M_U:n on selvitettävä, hyväksyisikö kone M syötteen w.
     - tätä varten 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.
   - aluksi M_U etsii ykkösnauhalta M:n kuvauksesta kohdan, joka vastaa M:n simuloitua tilaa q_i ja merkkiä a_j.
   - 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}, 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_{yes} tai q_{no}.

□

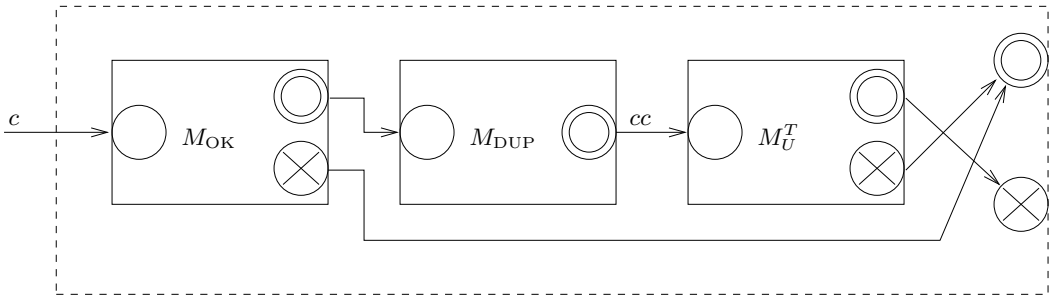
Lause: Kieli U ei ole rekursiivinen.

**Todistus.* Oletetaan, että kielellä U olisi totaalinen tunnistajakone M_U^T. Tällöin voitaisiin Lemman kielelle D muodostaa totaalinen tunnistajakone M_D seuraavasti:

- Olkoon M_{OK} totaalinen Turingin kone, joka testaa, onko syötteenä annettu merkkijono kelvollinen Turingin koneen koodi.
- 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 kaavion esittämällä tavalla.

Selvästi kone M_D on totaalinen, jos kone M_U^T on, ja

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



Kuva 6.5: Diagonaalikielen D tunnistava kone M_D.

Mutta lemman mukaan kieli D ei ole rekursiivinen; ristiriita. $\square$.

Seuraus: Kieli

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

ei ole rekursiivisesti lueteltava.

Todistus. Kieli $\tilde{U}$ on oleellisesti sama kuin universaalikielen U komplementti $\bar{U}$; tarkasti ottaen on $\bar{U} = \tilde{U} \cup \mathrm{ERR}$, missä ERR on helposti tunnistettava rekursiivinen kieli

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

Jos siis kieli $\tilde{U}$ olisi rekursiivisesti lueteltava, olisi samoin myös kieli $\bar{U}$. Koska kieli U on rekursiivisesti lueteltava, seuraisi tästä, että U on peräti rekursiivinen. Mutta tämä on vastoin edellisen lauseen tulosta, mistä päätellään, että kieli $\tilde{U}$ ei voi olla rekursiivisesti lueteltava. $\square$

6.5 *Ekskursio: Ratkeavuus ja ratkeamattomuus matematiikassa

Monessa laskettavuutta ja laskennan vaativuutta koskevassa tärkeässä tuloksessa meidän on kiittäminen matemaattikoja (mm. Gödeliä ja Churchia), joten pieni ekskursio matematiikkaan lienee paikallaan. Erityisesti Gödelin epätäydellisyyslause on suoraan sovellettavissa (universaali) Turingin koneita (mitä tahansa laskennan mallia) koskevaksi. Valotetaan ensin kuitenkin hämärien termien *rekursiivinen* ja *rekursiivisesti numeroituva kieli* alkuperää.

6.5.1 Rekursiiviset ja rekursiivisesti numeroituvat joukot matematiikassa

Jos ongelman voi ratkaista primitiivirekursiivisella funktiolla, se on laskennallisesti ratkeava ja voimme jopa antaa laskennan vaativuudelle ylärajan. Jos sen voi ratkaista vain rekursiivisella funktiolla, se ratkeaa laskennallisesti eli äärellisessä ajassa, mutta emme voi antaa mitään ylärajaa sen vaativuudelle. Jos ongelma kuuluu rekursiivisesti numeroituviin kieliin, emme voi edes antaa ratkaisevaa laskennallista funktiota, mutta kylläkin deklaratiivisen määritelmän, mitä vastauksen tulisi täyttää. Tämän pohjalta voimme muodostaa ns. *osittaisrekursiivisen funktion*. Määriteltyä vastausta ei kuitenkaan voida laskea äärellisessä ajassa (kaikilla syötteillä). Jos ongelma ei ole edes rekursiivisesti numeroituva, emme pysty antamaan edes osittaisrekursiivista funktiota, joka laskisi vastauksen.

Rekursiiviset funktiot

Sanotaan, että joukko $A \subseteq \mathbb{N}$ on rekursiivinen, jos sen tunnistusongelma π_A (oikeammin joukon A *karakteristinen funktio*) on rekursiivinen funktio. Huom! Mikä tahansa merkkijono voidaan aina koodata luonnollisena lukuna eli kieli A on itse asiassa $\mathbb{N}$:n osajoukko.

Määritellään ensin *primitiivirekursiiviset funktiot*

Määritelmä: Primitiivirekursiivisten funktioden perhe on pienin funktioperhe luonnollisten lukujen joukossa ($f : \mathbb{N} \to \mathbb{N}$), joka sisältää funktiot

$Z(n) = 0$ (nollafunktio)
$S(n) = n + 1$ (seuraajafunktio)

$Pr_i{}^n(x_1, ..., x_n) = x_i$ (projektiofunktio)

sekä näistä yhdistämällä ja seuraavalla ns. rekursiosäännöllä saatavat funktiot.

Rekursiosääntö: Olkoon f n-paikkainen primitiivirekursiivinen funktio ja g $n+2$-paikkainen primitiivirekursiivinen funktio. Tällöin $n+1$-paikkainen funktio h,

$h(0, x_1, ..., x_n) = f(x_1, ..., x_n)$
$h(y+1, x_1, ..., x_n) = g(y, h(x_1, ..., x_n), x_1, ..., x_n)$

on primitiivirekursiivinen.

Kun $n = 0$, niin $h(0) = a$ (vakio) ja $h(y+1) = g(y, h(y))$.

Esimerkiksi yhteenlasku, kertolasku, eksponenttifunktio, vakiofunktio ja luonnollisiin lukuihin rajoitettu vähennyslasku ovat primitiivirekursiivisia. Uusia primitiivirekursiivisia funktioita saadaan myös *rajoitetulla minimalisaatiolla*, merk. $f(y, x_1, ..., x_n) = (\mu z \leq y)R(z, x_1, ..., x_n)$, jos

$$f(y, x_1, .., x_n) = \begin{cases} \text{pienin } z \leq y, \text{ jolle } R(z, x_1, ..., x_n), & \text{jos tällainen } z \text{ on olemassa, ja} \\ 0, & \text{muuten} \end{cases}$$

missä R on prim.rek. relaatio.

Rekursiivisten funktioiden perhe määritellään aivan samaan tapaan kuin primitiivirekursiivisten funktioiden perhe, mutta sallimme rajoittamattoman minimalisaation. Merk. $f(x_1, ..., x_n) = \mu y R(y, x_1, ..., x_n)$, jos

$$f(x_1, ..., x_n) = \text{ pienin } y, \text{ jolle } R(y, x_1, ..., x_n), \text{ jos tällainen } y \text{ on olemassa.}$$

Primitiivirekursiivisten ja rekursiivisten funktioiden ero on seuraava: jos $f(x)$ on primitiivirekursiivinen, on $f(n)$:n laskemiseen kuluva aika ennalta arvioitavissa. (Ts. voimme antaa ratkaisevan Turingin koneen aikavaativuudelle ylärajan.) Jos funktio on vain rekursiivinen, tiedämme vain, että $f(n)$ on ratkeava, mutta emme tiedä, montako laskenta-askelta ratkaisu vaatii (ts. vastaavan TM:n laskenta päättyy äärellisessä ajassa, mutta emme voi antaa ennalta mitään arviota, kauanko laskenta tulee kestämään.)

Esimerkiksi Ackermannin funktio A on rekursiivinen, mutta ei primitiivirekursiivinen.

$A(0, x) = x + 1$
$A(n+1, 0) = A(n, 1)$
$A(n+1, x+1) = A(n, A(n+1, x))$

Ackermannin funktio on sikäli kiintoisa, että kaikille primitiivirekursiivisille funktioille f löytyy n, jolle $f(x) < A(n,x)$ kaikilla x. Ja kun n lähenee ääretöntä, antaa Ackermannin funktio ylärajan (majorantin) myös kaikille rekursiivisille funktioille! Toisin sanoen se antaa ylärajan minkä tahansa ratkeavan ongelman aikavaativuudelle.

Rekursiivisesti numeroituvat joukot

Rekursiivisen kielen tunnistusongelma (kuuluuko sana x kieleen A) on rekursiivinen eli algoritmisesti ratkeava, joskaan emme voi antaa laskennan aikavaativuudelle mitään ylärajaa.

Toisaalta jokainen epätyhjä rekursiivinen kieli A (sen sisältämät sanat) voidaan aina numeroida rekursiivisella funktiolla. Määritellään A:n sanoille numerointi: $A = \{f(n)|n \in \mathbb{N}\}$, missä $f(n) = n$, jos $\pi_A(n) = 1$ ja a_0, jos $\pi_A(n) = 0$ ($a_0 \in A$)

$$f(n) = \begin{cases} n, & \text{jos } \pi_A(n) = 1 \text{ ja} \\ a_0, & \text{jos } \pi_A(n) = 0 \ (a_0 \in A) \end{cases}$$

Määritelmä: Joukko $A \subseteq \mathbb{N}$ on rekursiivisesti numeroituva, jos $A = \emptyset$ tai on olemassa rekursiivinen funktio $f : \mathbb{N} \to \mathbb{N}$ s.e. $A = \{f(n)|n \in \mathbb{N}\}$.

Kaikki rekursiiviset joukot ovat siis myös rekursiivisesti numeroituvia. Rekursiivisesti numeroituvien joukkojen perhe on kuitenkin aidosti suurempi kuin rekursiivisten eli on olemassa joukkoja (kieliä), jotka ovat rekursiivisesti numeroituvia mutta eivät rekursiivisia (Churchin lause).

Rekursiivisuuden ja rekursiivisen numeroituvuuden välinen ero on se, että rekursiivisen funktion arvon voi aina laskea äärellisen monella laskenta-askeleella ja rekursiivisen joukon tunnistusongelma on siis ratkeava (ts. meillä on algoritmi, joka ratkaisee sen). Sen sijaan, jos joukko on vain rekursiivisesti numeroituva, emme voi määritellä mitään funktiota, joka tunnistaisi sen koska funktion määritelmä itsessään edellyttää määriteltävyyttä (voimme laskea funktion arvon kaikilla määrittelyjoukon alkioilla). Voimme kyllä antaa deklaratiivisen määritelmän, mitä jokin rekursiivisesti numeroituva joukko pitää sisällään ja kuvata tällaisia kieliä lukuteorian kaavoilla. Meillä on esim. käsitys miten kaavan $\Phi(x_1, ..., x_n)$ totuus selvitetään, vaikkei universaaliväitteen $\forall x \Phi(x)$ totuus ole laskennallisesti ratkeava (pitäisi tutkia $\Phi(n)$:n totuus äärettömän monella $n \in N$). Tällaista osittain määriteltyä ”funktiota”, joka on laskettavissa vain joillain määrittelyjoukon arvoilla, kutsutaan osittaisrekursiiviseksi funktioksi.

Mitä sitten ovat täysin ratkeamattomat eli ei-rekursiivisesti numeroituvat joukot

matematiikassa? Niille emme voi antaa edes osittaisrekursiivista funktiota (deklaratiivista määritelmää), joka jotenkin kuvaisi joukon sisällön.

6.5.2 Gödelin epätäydellisyyslause

Gödelin epätäydellisyyslause on matemaattinen vastine Turingin koneiden itsetutkiskelusta seuraavalle epätäydellisyydelle.

Gödel keksi menetelmän, jolla mikä tahansa lukuteorian kaava (tai predikaattilogiikan lause) voidaan koodata yksikäsitteisenä kokonaislukuna, kaavan Gödel-lukuna. Toisin sanoen jokaiseen kaavaan voidaan liittää tasan yksi luku ja jokaiseen lukuun liittyy korkeintaan yksi kaava (osa kokonaisluvuista ei vastaa mitään kaavaa). Tällä tavalla kaavassa päästään viittaamaan toisiin kaavoihin, mikä mahdollistaa kaavojen paradoksaalisen itsetutkiskelun. ($\leftrightarrow$ Turingin koneiden koodaus lukuina)

Gödel-numerointi tapahtuu seuraavasti. Ensin liitetään jokaiseen lukuteorian merkkiin luonnollinen luku:

$\#(0) = 1$
$\#(1) = 2$
$\#(+) = 3$
$\#(\times) = 4$
$\#(=) = 5$
$\#(() = 6$
$\#()) = 7$
$\#(\wedge) = 8$
$\#(\vee) = 9$
$\#(\neg) = 10$
$\#(\exists) = 11$
$\#(\forall) = 12$
$\#(v_0) = 13$
$\#(v_1) = 14$
$\#(v_2) = 15$
....

missä muuttujan v_n järjestysnumero on $\#(v_n) = 13 + n$.

Olkoon w nyt edellisistä merkeistä muodostuva sana $w = w_0 w_1 ... w_n$. Tällöin w:n Gödel-luku on

$$\lceil w \rceil = \Sigma_{i=0}{}^{n} p_i{}^{(\#w_i)} = p_0{}^{(\#(w_0)} p_1{}^{(\#w_1)} ... p_n{}^{(\#w_n)},$$

missä $p_0, p_1, ...$ ovat peräkkäiset alkuluvut 2,3,5,7,...

Esimerkiksi $\lceil(1 = 1)\rceil = 2^6 \times 3^2 \times 5^5 \times 7^2 \times 11^7$.

Tämä itsessään hyvin yksinkertainen työkalu mahdollisti mahtavan tuloksen: sen avulla Gödel todisti kuuluisan Epätäydellisyyslauseensa, jonka mukaan mikään konsistentti aksioomajärjestelmä (esim. aritmetiikka) ei voi olla täydellinen (t.s. se ei voi sisältää kaikki tosia lauseita teoreemoinaan). Gödelin todistus oli myös yksinkertaisuudessaan suunnattoman nerokas ja perustui Valehtelijan paradoksiin (Cantorin diagonalisointiargumenttiin): jos kaava ϕ sanoo "Kaava, jonka Gödel-luku on x ei ole todistuva" ja ϕ:n oma Gödel-luku on x, seuraa ristiriita: toisin sanoen emme voi ratkaista kaikista mahdollisista tosista lauseista, ovatko ne todistuvia vai eivät. (Lause ϕ sanoo siis "En ole todistuva".)

Huom! Voimme kyllä päätellä, että lause ϕ on tosi – sehän ei ollut todistuva kyseisessä järjestelmässä – mutta järjestelmä itse ei kykene ratkaisemaan sitä, mikäli pidämme kiinni järjestelmän konsistenssista. Jos taas sallimme, että järjestelmä on epäkonsistentti (ts. siinä pätee sekä jokin lause että sen negaatio), niin mikä tahansa lause voidaan todistaa oikeaksi. Järjestelmä voi siis täydellinen, vain jos se on epäkonsistentti, mutta jos se on konsistentti, se on epätäydellinen. [4]

Tällä viattoman oloisella vain lausekalkyylia koskevalla lauseella on kuitenkin todella laajat seuraukset. Siitä seuraa, ettei mikään aksiomatisointijärjestelmä, esim. algebra, tai mikään laskennanmalli, kuten Turingin koneet, ole täydellinen!

6.5.3 Kirjallisuutta

Hofstadter, Douglas R.: Gödel, Escher, Bach: An Eternal Golden Braid. Vintage Books, 1979. (chapters XIV-XV)
Nagel, E. & Newman, J.K.: Gödel's Proof. New York University Press, 1986.
Rucker, Rudy: Mieli ja äärettömyys.
Väänänen, Jouko: Matemaattinen logiikka. Gaudeamus, 1987.

[4] Valehtelijan paradoksin matemaattinen versio, Tarskin lause, sanoo, että joukko $T = \{x|x$ on toden lauseen ϕ Gödel-numero$\}$ ei ole rekursiivinen, eikä edes rekursiivisesti numeroituva! Sen sijaan jos kiinnitämme jonkin "efektiivisen" ($\sim$ algoritmisesti ratkeavan) aksioomajärjestelmän $\mathcal{A}$, niin joukko $T' = \{x|x$ on $\mathcal{A}$:ssa todistuvan lauseen ϕ Gödel-numero$\}$ on rekursiivisesti numeroituva, mutta ei rekursiivinen.

6.6 Konkreettisia ratkeamattomuustuloksia

Valitettavan monet tärkeät tietojenkäsittelyongelmat ovat ratkeamattomia. Ns. Ricen lauseen mukaan itse asiassa jokseenkin kaikki ohjelmien toimintaa, tai tarkemmin sanoen niiden laskemia syöte/tulos-kuvauksia koskevat kysymykset ovat ratkeamattomia.

6.6.1 Turingin koneiden pysähtymisongelma

Lause: Kieli

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

on rekursiivisesti lueteltava, mutta ei rekursiivinen.

Todistus. Todetaan ensin, että kieli H on rekursiivisesti lueteltava:

- 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, että olisi $H = L(M_H)$ jollakin totaalisella Turingin koneella M_H.
- Oletetaan lisäksi, että kone M_H pysähtyessään jättää nauhalle alkuperäisen syötteensä, mahdollisesti tyhjämerkeillä jatkettuna.
- Olkoon M_U edellä konstruoitu universaalikone.
- Kielelle U voitaisiin nyt muodostaa totaalinen tunnistaja yhdistämällä koneet M_H ja M_U kaavion esittämällä tavalla.
- Aiemman tuloksen mukaan tällaista totaalia kielen U tunnistajakonetta ei kuitenkaan voi olla olemassa! Saatu ristiriita osoittaa, että H ei voi olla rekursiivinen. □

⇒ Seuraus: Kieli

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

ei ole rekursiivisesti lueteltava. □

6.6.2 Sama tulos Pascalilla

Turingin koneiden ja "tavallisten" ohjelmien vastaavuus:

Turingin kone -formalismi	$\sim$	ohjelmointikieli
Turingin kone	$\sim$	ohjelma
Turingin koneen koodi	$\sim$	konekieliesitys
Universaalikone	$\sim$	konekielen tulkki

- Koska mikä tahansa Turingin kone voidaan toteuttaa Pascal-ohjelmana ja kääntäen, siirtyvät Turingin koneita koskevat ratkeavuus- ja ratkeamattomuustulokset välittömästi koskemaan myös Pascal-ohjelmia.
- Esimerkiksi pysähtymisongelman Pascal-tulkinta on: "Ei ole olemassa totaalista Pascal-ohjelmaa, joka ratkaisisi, pysähtyykö annettu Pascal-ohjelma P annetulla syötteellä w".

Pysähtymisongelman ratkeamattomuustodistus suoraan Pascalilla:

Oletetaan, että voitaisiin kirjoittaa totaalinen Pascal-funktio

function $H(p, w : \mathit{text}) : \mathit{Boolean}$,

joka saa arvon **true**, jos merkkijonoparametrin p esittämä proseduuri pysähtyy syötteellä w, ja **false** muuten.

Kirjoitetaan tämän perusteella Pascal-proseduuri $\hat{H}$:

```
procedure Ĥ(p : text);
    function H(p, w : text) : Boolean;
    begin {Funktion H runko}
        ...
    end;
    begin {Pääohjelma}
        if H(p, p) then while true do;
    end.
```

Merkitään proseduurin $\hat{H}$ ohjelmatekstiä $\hat{h}$:lla ja tarkastellaan proseduurin $\hat{H}$ laskentaa omalla kuvauksellaan.

Saadaan ristiriita:

$$\begin{aligned} \hat{H}(\hat{h}) \text{ pysähtyy} &\Leftrightarrow H(\hat{h}, \hat{h}) = \textbf{false} \\ &\Leftrightarrow \hat{H}(\hat{h}) \text{ ei pysähdy.} \end{aligned}$$

Ristiriidasta seuraa, että oletettua totaalista pysähtymisentestausohjelmaa H ei voi olla olemassa.

6.6.3 Semanttisten ominaisuuksien ratkeamattomuus

- Turingin koneen M *semanttinen ominaisuus* $\sim$ mikä tahansa sellainen ominaisuus $\mathcal{S}$, joka riippuu vain koneen M tunnistamasta kielestä, ei sen syntaktisesta rakenteesta. (joskus puhutaan myös *funktionaalisista ominaisuuksista*, mikä viittaa koneen toimintaan)

- tarkkaan ottaen semanttinen ominaisuus $\mathcal{S}$ on mikä tahansa kokoelma rekursiivisesti lueteltavia aakkoston $\{0,1\}$ kieliä ja koneella M on ominaisuus $\mathcal{S}$, jos $L(M) \in \mathcal{S}$.

- Esimerkkejä semanttisista ominaisuuksista:
  - "M hyväksyy tyhjän syötejonon"
  - "M hyväksyy jonkin syötejonon"
  - "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 täsmälleen samat semanttiset ominaisuudet.

- Ominaisuus $\mathcal{S}$ on *ratkeava*, jos annetusta Turingin koneen koodista voidaan algoritmisesti päätellä, onko koneella kysytty semanttinen ominaisuus.

- ts. jos joukko
$$\text{codes}(\mathcal{S}) = \{c \mid L(M_c) \in \mathcal{S}\}$$
on rekursiivinen.

- *Triviaalit ominaisuudet* ovat ominaisuus $\mathcal{S}_\emptyset$, jota ei ole millään koneella ja ominaisuus $\mathcal{S}_{RE}$, joka on kaikilla koneilla.

Ricen lause: Kaikki Turingin koneiden epätriviaalit semanttiset ominaisuudet ovat ratkeamattomia.

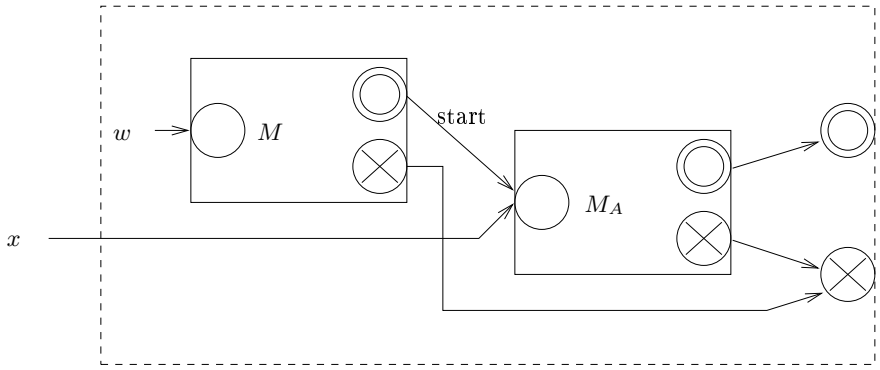
**Todistus.*

- 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 $\bar{\mathcal{S}} = \mathrm{RE} - \mathcal{S}$ on ratkeamaton, ja päätellä edelleen tästä, 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 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):



Kuva 6.6: Turingin koneen M^w rakenne (Ricen lause).

- 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 6.6).

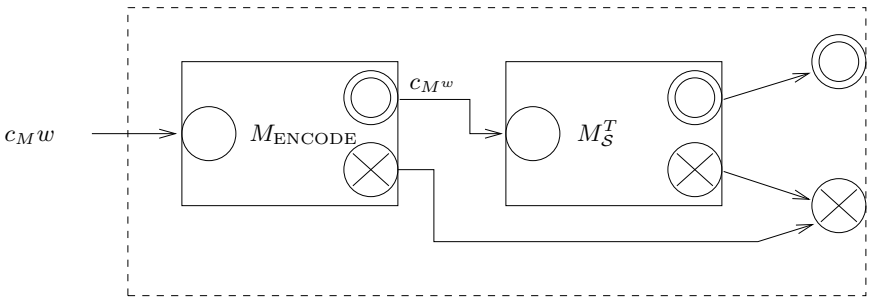
- Koneen M^w tunnistama kieli on siten:

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

- 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 $w \in L(M)$.

- Vastaväite: ominaisuus $\mathcal{S}$ on ratkeava, ts. kielellä $\mathrm{codes}(\mathcal{S})$ on totaalinen tunnistajakone $M_{\mathcal{S}}^T$.

- Tällöin saataisiin totaalinen tunnistajakone kielelle U yhdistämällä koneet M_{ENCODE} ja $M_{\mathcal{S}}^T$ kuvan mukaisesti. Selvästi kone M_U^T on totaalinen, jos $M_{\mathcal{S}}^T$



Kuva 6.7: Turingin koneen M_U^T rakenne (Ricen lause).

on, ja

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

- Koska kieli U ei ole rekursiivinen, seuraa ristiriita, eli ominaisuus $\mathcal{S}$ ei voi olla ratkeava. □

6.6.4 Muita ratkeamattomia ongelmia

- **Predikaattikalkyylin ratkeamattomuus; Church/Turing 1936**

  Ei ole olemassa algoritmia, joka ratkaisisi, onko annettu ensimmäisen kertaluvun predikaattikalkyylin kaava ϕ validi ("loogisesti tosi", todistuva predikaattikalkyylin aksioomista). □

- **"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. jonoja $(m_1, \ldots, m_n) \in \mathbf{Z}^n$, joilla $P(m_1, \ldots, m_n) = 0$). Ongelma on ratkematon jo, kun muuttujien lkm $n = 15$ tai polynomin aste $\deg(P) = 4$. □

- Eräiden kielioppiongelmien ratkeavuus, 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
$\overline{L(G)} \cap L(G')$ tyyppiä i?	T	E	T	T
$\overline{L(G)}$ tyyppiä i?	T	E	T	E

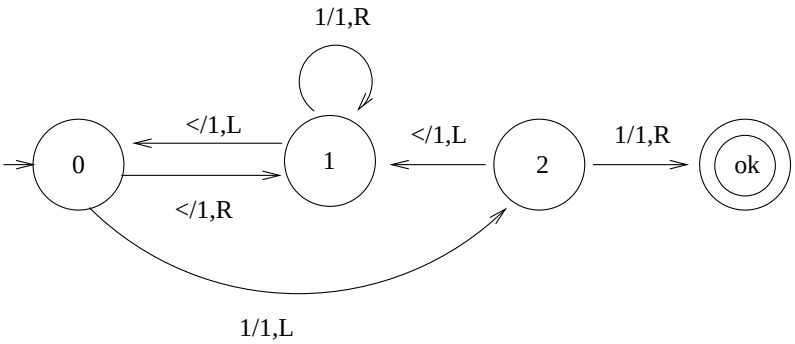
- Huom! Esim. jäsennysongelma $w \in L(G)$, kun kielioppi G on rajoittamaton, on siis ratkeamaton ongelma. Samoin mielivaltaisen kieliopin kuvaaman kielen säännöllisyys ja eräät sulkeumaominaisuudet.

6.6.5 *Joitain hauskoja ratkeamattomia ongelmia

Busy Beaver

Busy Beaver-ongelmassa tehtävänä on keksiä n-tilainen standardimallinen Turingin kone (jonka on siis joka askeleella siirryttävä joko oikealle tai vasemmalle – paikallaan ei saa pysyä), joka aloittaa toimintansa tyhjällä nauhalla ja on pysähtyessään kirjoittanut nauhalle mahdollisimman monta merkkiä. Aakkostona voidaan käyttää mitä tahansa unaariaakkostoa, esim. $\Sigma = \{a\}$, jolloin tehtävänä on kirjoittaa nauhalle mahdollisimman monta a-kirjainta. Vaihtoehtoisesti tehtävänä voi olla laatia mahdollisimman hidas Turingin kone, joka siis suorittaa mahdollisimman monta askelta ennen pysähtymistään. Kummassakin ongelmassa koneen täytyy kuitenkin pysähtyä lopulta. Tilojen lukumäärään n ei lasketa mukaan lopputiloja.

Esimerkiksi 3-tilainen Busy Beaver -kone voi kirjoittaa 6 merkkiä ja toimia 21 siirtymän ajan. Kuvassa kone, joka pysähtyy 13 siirtymän jälkeen ja kirjoittaa maksimaalisen 6 merkkiä:



Busy Beaver -funktio $f(n) = \{x|n$-tilainen kone kirjoittaa maksimissaan x merkkiä aloittaessaan tyhjältä nauhalta$\}$ kasvaa tavattoman nopeasti. Esim. 6-tilaisen koneen ennätys on 95,524,079 merkkiä ja kone pyörii parhaimmillaan 8,690,333,381,690,952 askeleen ajan. Funktiota ei kuitenkaan pysty laskemaan, joten tulokset on saatu kokeilemalla ja uusia ennätyksiä on odotettavissa.

Post correspondence problem (PCP)

PCP-ongelman voi mallintaa dominopelinä, jossa on käytettävissä äärellinen kokoelma erilaisia dominonappuloita. Kunkin nappulan ylä- ja alareunassa on merkkijono, esim. $[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]$. Kustakin nappulasta voi "taikoa" niin monta kopiota kuin ikinä haluaa. Tehtävänä on asettaa nappulat sillä tavalla peräkkäin, että ala- ja yläreunaan muodostuu sama merkkijono. Esim. edellisen kokoelman nappuloilla eräs ratkaisu olisi $[\frac{a}{ab}][\frac{b}{ca}][\frac{ca}{a}][\frac{a}{ab}][\frac{abc}{c}]$.

Yleisesti ongelma on siis seuraava: Annettuna kokoelma $P = [\frac{t_1}{s_1}], [\frac{t_2}{s_2}], ..., [\frac{t_k}{s_k}]$, onko olemassa sellaista indeksijonoa $i_1, i_2, ..., i_n$, että $s_{i_1}s_{i_2}...s_{i_n} = t_{i_1}t_{i_2}...t_{i_n}$?

6.7 *Ongelmien palautukset

Joskus ongelma voidaan palauttaa (redusoida) joksikin toiseksi tunnetuksi ongelmaksi, jolloin voimme päätellä myös alkuperäisen ongelman ratkeavuuden/ratkeamattomuuden. Ehtona kuitenkin on, että ongelman palautus on itsessään ratkeava ongelma kaikilla syötteillä (ts. se voidaan suorittaa totaalisella Turingin konella).

Idea:

1. Meillä on tuntematon kieli A ja tunnettu kieli B
2. Palautetaan A B:ksi palautusfunktiolla f siten, että $x \in A \Leftrightarrow f(x) \in B$ kaikilla $x \in \Sigma^*$.

3. Nyt A on ratkeava, jos B on, A on osittain ratkeava, jos B on, ja A on ratkeamaton, jos B on, *kunhan* palautusfunktio f on itsessään ratkeava.

Määritelmä: Turingin koneen $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ *laskema osittaiskuvaus* (t. *-funktio*)

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

määritellään:

$$f_M(x) = \begin{cases} u, & \text{jos } (q_0, \underline{x}) \underset{M}{\vdash}^* (q, u\underline{a}v) \text{ jollakin } q \in \{q_{\text{yes}}, q_{\text{no}}\}, av \in \Gamma^*; \\ \text{määrittelemätön, muuten.} \end{cases}$$

- Osittaisfunktio $f : \Sigma^* \to A$ on *osittaisrekursiivinen* jos se voidaan laskea jollakin Turingin koneella ja *(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.
- Tarvitsemme siis *rekursiivisen palautus funktion*

Määritelmä: Formaali kieli $A \subseteq \Sigma^*$ voidaan *palauttaa rekursiivisesti* kieleen $B \subseteq \Gamma^*$, merkitään

$$A \leq_m B,$$

jos on olemassa rekursiivinen funktio $f : \Sigma^* \to \Gamma^*$, jolla on ominaisuus:

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

- Huom! Palautusketju voi olla pitkäkin, sillä rekursiivinen palautusrelaatio on transitiivinen, ts. jos $A \leq_m B$ ja $B \leq_m C$, niin $A \leq_m C$.
- Voisimme siis palauttaa A:n B:ksi, B:n C:ksi, C:n D:ksi, jne. kunnes vastaan tulee tuttu kieli.
- Jos $A \leq_m B$ ja B on rekursiivisesti numeroituva, on A:kin rekursiivisesti numeroituva.
- Jos $A \leq_m B$ ja B on rekursiivinen, on A:kin rekursiivinen.
- Kaksi kieltä A ja B ovat keskenään isomorfiset, jos ne on määritelty samassa aakkostossa Σ ja rekursiivinen palautusfunktio on bijektio ts. voimme suorittaa palautuksen kumpaankin suuntaan.

- Formaalisti:

  Määritelmä: Kielet $A, B \subseteq \{0,1\}^*$ ovat *rekursiivisesti isomorfisia*, 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 numeroituvaa kieltä A sanotaan RE-täydelliseksi, jos mikä tahansa luokan RE kieli voidaan rekursiivisesti palauttaa A:han.

- Formaalisti:

  Kieli $A \subseteq \{0,1\}^*$ on *RE-täydellinen*, jos

  (i) $A \in$ RE ja
  (ii) $B \leq_m A$ kaikilla $B \in$ RE.

- esimerkiksi universaalikieli U on RE-täydellinen.

  Todistus. Tiedetään, että $U \in$ RE. Olkoon $B = L(M_B)$ mielivaltainen luokan RE kieli. Tällöin B voidaan palauttaa U:hun funktiolla $f(x) = c_{M_B}x$. Tämä funktio on selvästi rekursiivinen, ja sillä on ominaisuus

  $$x \in B = L(M_B) \quad \Leftrightarrow \quad f(x) = c_{M_B}x \in U. \quad \square$$

- Ricen lauseesta seuraa, että mm. kaikki ongelmat, joissa yritetään tehdä jotain päätelmiä Turingin koneiden tunnistamista kielistä niiden koodien perusteella ovat RE-täydellisiä.

- Yleensäkin näyttää olevan niin, että kaikki "luonnolliset" rekursiivisesti luetelta-vat, ei-rekursiiviset kielet ovat RE-täydellisiä.

- Teoreettisesti voidaan kuitenkin osoittaa, että [Post] Luokassa RE $\setminus$ REC on kieliä, jotka eivät ole RE-täydellisiä. $\square$

- Huom! Kaikki RE-täydelliset kielet ovat rekursiivisesti isomorfisia. (ts. ne voidaan palauttaa toisikseen)

Luku 7

Loppusanat

7.1 Yhteenveto

Laskennallisen ongelman vaikeutta voidaan arvioida tutkimalla siihen liittyvää päätösongelmaa sekä vastaavaa formaalia kieliluokkaa.

Säännöllisen kielen tunnistusongelma $x \in L(r)$, missä säännöllinen lauseke r kuvaa kielen, voidaan ratkaista äärellisellä automaatilla. Mikäli olemme kiinnostuneita vain siitä, onko kyseessä säännöllinen kieli, voimme 1) antaa kielen kuvaavan säännöllisen lausekkeen 2) muodostaa tunnistusongelman ratkaisevan äärellisen automaatin (automaatti voi olla epädeterministinen tai ϵ-automaatti) 3) osoittaa, että kieli saadaan sulkeumaoperaatioilla tunnetuista säännöllisistä kielistä 4) antaa kielen kuvaavan lineaarisen kieliopin. Tarvittaessa voimme aina muodostaa deterministisen äärellisen automaatin, joka on helppo ohjelmoida, ja joka ratkaisee tunnistusongelman lineaarisessa ajassa.

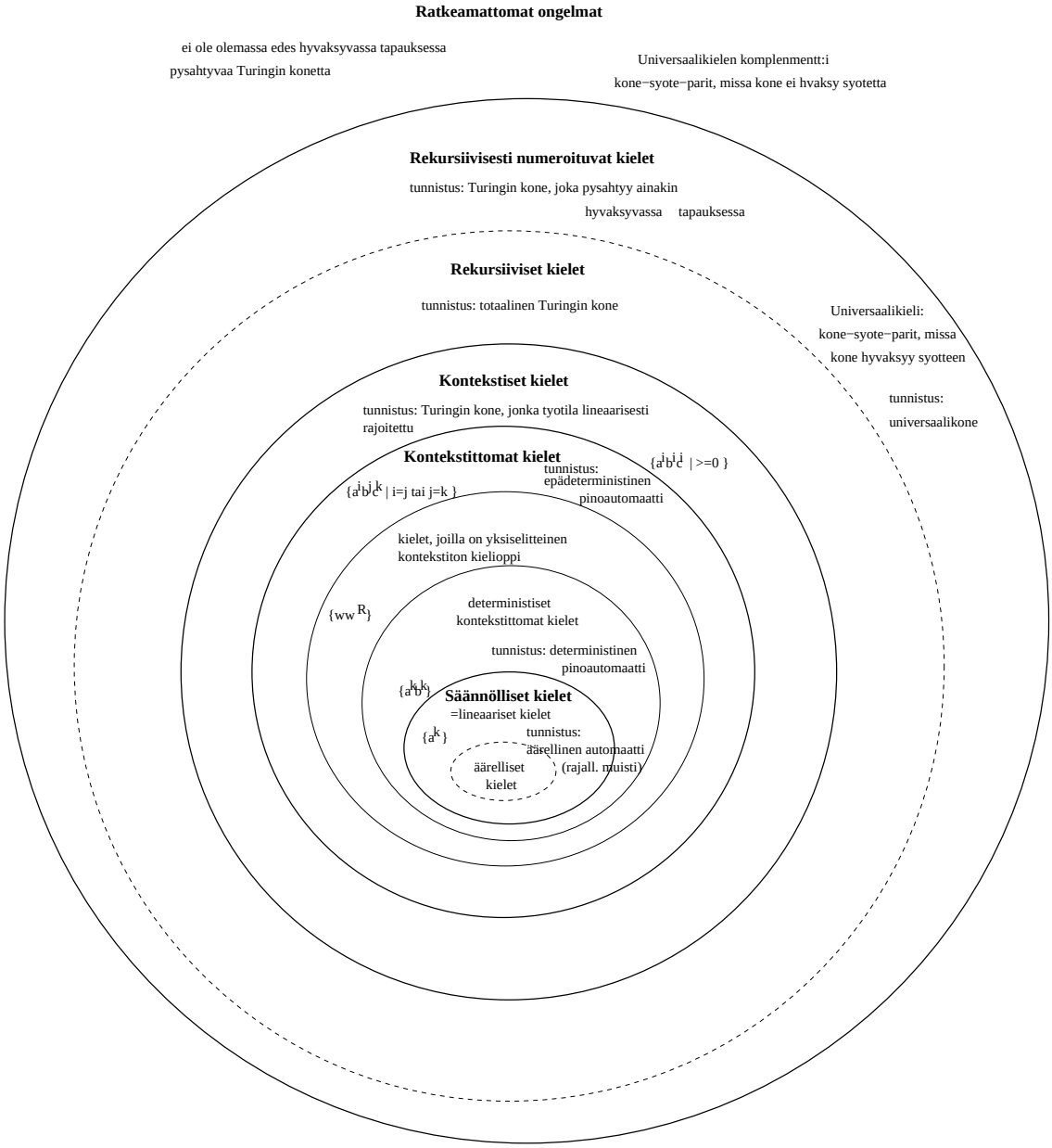
Kontekstittomien kielten tunnistusongelmaa kutsutaan yleensä jäsentämiseksi. Tehtävänä on tutkia, kuuluuko annettu merkkijono x annetun kieliopin G kuvaamaan kieleen ($x \in L(G)$?) sekä antaa x:n johto tai jäsennyspuu kieliopissa. Kontekstittoman kielen tunnistusongelma voidaan aina ratkaista epädeterministisellä pinoautomaatilla, mutta sitä on hankala ohjelmoida eikä se kerro, miten merkkijono johdetaan. Sen sijaan merkkijonon jäsennys voidaan aina suorittaa CYK-algoritmilla ajassa $O(n^3)$, kun kielioppi on ensin muunnettu Chomskyn normaalimuotoon. Joillekin kontekstittomien kieleten osaluokille on olemassa tehokkaampia jäsennysmenetelmiä. Esimerkiksi LL(1)-kielet voidaan jäsentää rekursiivisella jäsentimellä lineaarisessa ajassa ja LR(1)-jäsentimillä jäsentää kaikki deterministiset kielet.

Turingin koneilla voidaan tunnistaa rajoittamattomat kielet. Rajoittamattomat kielet

jaetaan kahteen luokkaa: rekursiivisiin ja rekursiivisesti numeroituviin kieliin, joita vastaavat päätösongelmat ovat ratkeavia tai osittain ratkeavia. Totaalisilla eli aina pysähtyvillä Turingin koneilla voidaan ratkaista kaikki ongelmat, jotka ovat laskennallisesti ratkeavia (ts. tunnistaa rekursiiviset kielet). Mikäli ongelmalle voidaan laatia ratkaiseva Turingin kone, mutta se ei pysähdy kaikilla syötteillä, on ongelma osittain ratkeava. Rekursiivisesti numeroituvan kielen tunnistusongelman ratkaiseva Turingin kone ei siis välttämättä pysähdy ”Ei”-tapauksessa.

Rajoittamattomien kielten erikoistapaus ovat kontekstilliset kielet, jotka voidaan tunnistaa epädeterministisellä Turingin koneella syötteen vaatimassa työtilassa. Tämä merkitsee sitä, että tunnistus vaatii korkeintaan eksponentiaalisen ajan. Yleisessä tapauksessa rekursiivisten kielten tunnistus voi vaatia vielä paljon suuremman ajan. Ackermannin funktio antaa kuitenkin ylärajan tunnettujen rekursiivisten funktioiden vaativuudelle. Myös Busy Beaver -funktio antaa ylärajan kaikkien rekursiivisten ongelmien vaativuudelle, mutta Busy Beaver ei ole itse ratkeava ongelma.

Hierarkian ulkopuolelle jäävät täysin ratkeamattomat ongelmat, joille ei voida laatia lainkaan pysähtyvää Turingin konetta. Vastaavan päätösongelman ”ratkaiseva” Turingin kone ei siis pysähdy edes ”Kyllä”-tapauksessa.



7.2 Mitä tämän jälkeen? Laskennan vaativuudesta

Tietojenkäsittelijälle ei riitä tieto, onko ongelma ratkeava, vaan hän haluaa myös tietää, kuinka vaativa se on. Chomskyn kielihierarkian luokat antavat yleiskuvan siitä, kuinka vaikea ongelma on. Jos joku ongelma ratkeaa äärellisellä automaatilla ja toinen vaatii Turingin koneen, on ensimmäinen jossain mielessä helpompi ongelma. Mutta entä jos ensimmäinen ongelma vaatii äärellisen automaatin, jonka

tilojen lukumäärä on eksponentiaalisesti suurempi kuin toisen ongelman ratkaisevan Turingin koneen? Enää ensimmäisen ongelman yksinkertaisuus ei olekaan niin itsestäänselvää.

Miten sitten voisimme verrata ongelmien vaativuutta objektiivisesti? Entä jos vertaisimme ratkaisevien ohjelmien koodeja tai pseudokoodialgoritmeja? Kumpikaan näistä ei kuitenkaan täysin paljasta ongelman todellista vaativuutta, vaikka käytännön ohjelmointitilanteissa niiden arviointi riittää hyvin. Ohjelma voi "piilottaa" yhteen askeleeseen hyvinkin vaihtelevan määrän operaatioita, eikä esim. assembler- ja Pascal-koodin askelien vertailu anna järkevää tulosta.

Onneksi kaikki ratkeavat ongelmat voidaan kuitenkin esittää Turingin koneina! Olipa kyseessä äärellisellä tai pinoautomaatilla ratkeava ongelma, voimme aina konstruoida myös Turingin koneen, joka ratkaisee ongelman, ja vertailla Turingin koneiden aikavaativuuksia. Tästä syystä Turingin koneet ovat hyvin tärkeitä, kun haluamme tietää ongelman todellisen aikavaativuuden, riipumatta siitä, millä kielellä tai laitteistolla se tullaan todellisuudessa toteuttamaan (tai jos haluamme tietää, kannattaako sitä ryhtyä lainkaan toteuttamaan).

Turingin koneen aika- ja tilavaativuus saadaan sen käyttämien askelten ja työnauhan solujen lukumääristä. Sanotaan, että kone M hyväksyy kielen L ajassa $T(n)$, jos kaikilla $x \in L$, missä $|x| = n$, M hyväksyy x:n $O(T(n))$:llä askeleella. Samoin kone M hyväksyy kielen L tilassa $S(n)$, jos kaikilla $x \in L$, $|x| = n$, M hyväksyy x:n käyttäen $O(S(n))$:ää työnauhan solua.

Huom! Epädeterministinen Turingin kone voi ratkaista ongelman pienemmin aika- ja tilavaatimuksin kuin deterministinen Turingin kone. Käytännössä epädeterministiset koneet täytyy ensin determinisoida, mutta teoreettisessa pohdiskelussa epädeterministisen Turingin koneiden vaativuusanalyysi voi olla hyvinkin valaisevaa. Esimerkiksi vaativuusluokan NP määritelmä perustuu tähän: luokan NP ongelmat kyetään ratkaisemaan polynomisessa ajassa epädeterministisellä Turingin koneella. Selvästi luokka P (ongelmat, jotka voidaan ratkaista polynomisessa ajassa deterministisellä Turingin koneella) sisältyy luokkaan NP ($P \subseteq NP$), mutta emme tiedä, päteekö sama toisinpäin, ts. onko $NP \subseteq NP$? Tämä $P = NP$-ongelma on ehkä tärkein tietojenkäsittelijöitä kiusaavista avoimista ongelmista.

Käytännössä merkittävimpiä luokan NP ongelmista ovat ns. *NP-täydelliset* ongelmat (*NP-complete problems*), jotka ovat eräänlaisia arkkityyppiongelmia luokassa NP. Mikä tahansa luokan NP ongelmista voidaan näet palauttaa polynomisessa ajassa NP-täydelliseksi ongelmaksi. Toisin sanoen, jos yksikin NP-täydellinen ongelma pystyttäisiin ratkaisemaan polynomisessa ajassa, niin kaikki muutkin luokan NP ongelmat ratkeaisivat polynomisessa ajassa. Tästä seuraa, että uusi ongelma voidaan osoittaa NP-täydelliseksi varsin helposti: riittää osoittaa, että se kuuluu

luokkaan NP ja että jokin NP-täydelliseksi tunnettu ongelma voidaan palauttaa siihen. Tällöin myös kaikki muut luokan NP ongelmat seuraavat perässä. [5]

Nimitystä *NP-kova* ongelma (*NP-hard problem*) käytetään sellaisesta ongelmasta, joka myös "kattaa" kaikki luokan NP ongelmat, ts. mikä tahansa luokan NP ongelma voidaan muuntaa polynomisessa ajassa kyseiseksi ongelmaksi. Emme kuitenkaan tiedä, kuuluuko NP-kova itse luokkaan NP (se on siis niin "kova" ongelma, että kukaan ei ole keksinyt sille epädeterminististä, polynomisessa ajassa toimivaa Turingin konetta). Mikäli joku ongelma havaitaan NP-kovaksi, voi hyvin suurella todennäköisyydellä arvata sen olevan eksponentiaalista tai vieläkin pahempaa aikavaativuusluokkaa.

[5] Ainut ongelma on todistaa muulla tapaa, että se ensimmäinen ongelma on NP-täydellinen. Onneksi meillä on Cookin teoreema, joka todistaa ns. SAT-ongelman

$$SAT = \{F | F \text{ on toteutuva lausekalkyylin kaava}\}$$

NP-täydelliseksi, ja jatkossa voimme nojata tähän tulokseen.