Ohjelmoinnin ja laskennan perusmallit

Kevät 2005

Luennoija:
Matti Luukkainen

Tämän monisteen sisältöön vaikuttaneet henkilöt:

- Matti Luukkainen
- Matti Kääriäinen
- Patrik Floréen
- Wilhelmiina Hämäläinen
- Pekka Orponen

1. Johdanto

1.1. Kurssin esittely

- Kurssin aihepiiri liittyy kysymyksiin:
  - Mitä tietokoneella voidaan laskea?
  - Mitä “laskenta” on?
- *Laskettavuusteoria*: Mitkä onglemat voidaan ratkaista tietokoneella?
- Eri (laskennalliset) ongelmat ovat luonteeltaan eri vaikeuksisia.
  Laskennan vaativuusteoria: Mitkä ongelmat voidaan ratkaista tietokoneella *tehokkaasti*?
- Kuten luonnontieteissä yleensä, ilmiöt mallinnetaan jolloin voidaan käyttää matemaattisen eksaktia loogista päättelyä selittämään luonnon ilmiöitä
- Tällä kurssilla esitellään joitakin tärkeitä *laskennan malleja*. Nämä tarjoavat tarvittavat mallit laskettavuusteorialle ja laskennan vaativuusteorialle. Aloitamme yksinkertaisimmista malleista ja etenemme kohti monimutkaisempia.
- Tärkeimmät kurssin käsitteet luotiin 1930-50-luvuilla
- Kurssin käsitteistöllä on myös käyttöä käytännön tietojenkäsittelyssä. Tarkastellaan esimerkkinä ohjelmointikielen kääntämistä.

- Tarkastellaan käytännön esimerkkinä yksinkertaista ohjelmointikieltä, kutsuttakoon sitä nimellä **olpe03**.
- Oheinen esimerkki esittelee kielen piirteitä:

$x := \mathbf{read};$
$i := 0; y := 1;$
$\mathbf{while}\ i < x\ \mathbf{do}$
$\mathbf{begin}$
  $y := y * x;$
  $i := i + 1$
$\mathbf{end};$
$\mathbf{write}(y);$
$\mathbf{if}\ i < 10\ \mathbf{then}\ z := (y + 1) * x\ \mathbf{else}\ z := (y - 1) * x$

- **olpe03** on melko rajoittunut kieli: kaikki muuttujat ovat tyyppiä *int* eikä muita rakenteita ole kuin esimerkissä ilmenevät, siis esim. aliohjelmat puuttuvat
- Kuitenkin **olpe03** rajoitteistaan huolimatta yhtä ilmaisuvoimainen kuin mikä tahansa ohjelmointikieli ja se sopii hyvin muutamien kurssin käsitteiden hyödyllisyyden esittelyyn:
  - Miten määritellään *yksikäsitteisesti* kielen syntaksi?
  - Voisiko syntaksimäärittelyn tehdä siten että se tukisi kääntäjän tekemistä?
- Tarkastellaan kurssin aikana hieman sitä miten kääntämisen ensimmäinen vaihe eli syntaksin tarkastus mahdollista tehdä systemaattisesti.

Käytännön järjestelyistä

- 2 ov, cumun pakollinen kurssi pääaineopiskelijoille, cumun valinnainen kurssi sivuaineopiskelijoille
- Taustatiedot: riittävät matemaattiset perustiedot (lukion oppimäärä sekä mielellään appro-tason matematiikan kursseja tai Diskreetti matematiikka I). Tietorakenteet-kurssi on myös avuksi.
- Kurssin jatkokurssina on laudaturkurssi Laskennan teoria, jossa käsitellään tarkemmin laskettavuusteoriaa ja laskennan vaativuusteoriaa
- Opetus: ks opetusohjelma
- Luennoija: Matti Luukkainen
  email: Matti.Luukkainen@cs.helsinki.fi
- Kurssi suoritetaan joko
  - kurssikokeella max 54 p + laskuharjoituksilla max 6 p = yht. 60 p, tai
  - erilliskokoeella (max 60 p)
- Kurssikoe: ks. koelista laitoksen kotisivulta

Laskuharjoitukset

- Laskuharjoitukset ovat *pakolliset* siten, että harjoitustehtävistä on suoritettava vähintään kolmannes.

Kurssimateriaali

- Kurssimateriaali koostuu luentokalvoista, jotka löytyvät kurssin kotisivulta
  http://www.cs.helsinki.fi/u/mluukkai/olpek05/
- Luennot pohjautuvat pääosin Pekka Orposen luentomonisteeseen Laskennan teoria (s. 1–60), mutta lisäksi käsitellään muitakin asioita.
- Kirjallisuutta
  - Pekka Orponen: Laskennan teoria, Helsingin yliopisto, Tietojenkäsittelytieteen laitos. (luentomoniste, myytävänä monistemyynnissä)
  - John E. Hopcroft, Rajeev Motwani & Jeffrey D. Ullman: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 2001 (kurssikirjahyllyssä)
  - Harry R. Lewis & Christos H. Papadimitriou: Elements of the Theory of Computation, Second Edition, Prentice-Hall, 1998.
  - Michael Sipser: Introduction to the Theory of Computation. PWS Publishing Company, 1997.
  - Efim Kinber & Carl Smith: Theory of Computing - A Gentle Introduction. Prentice-Hall, 2001.

Kurssin sisältö

1. Johdanto
    - Kurssin esittely
    - Muutamien matemaattisten käsitteiden kertausta
    - Laskennalliset ongelmat ja ratkeavuus
2. Säännölliset kielet ja äärelliset automaatit
    - Deterministiset äärelliset automaatit
    - Äärellisen automaatin minimointi
    - Epädeterministinen äärellinen automaatti
    - Automaatin determinisointi
    - Säännölliset lausekkeet ja kielet
    - Säännöllisten kielten rajoituksista: pumppauslemma
3. Kontekstittomat kielet ja pinoautomaatit
    - Kontekstittomat kieliopit ja kielet
    - Kielten jäsennysongelma
    - Rekursiivisesti etenevä jäsentäminen
    - CYK-jäsennysalgoritmi
    - Pinoautomaatit
    - Kontekstittomien kielten rajoituksista
4. Johdanto laskettavuuden teoriaan
    - Turingin kone
    - Kieliluokkia kontekstittomien kielten tuolla puolen

1.2. Matemaattisia peruskäsitteitä

Loogisia symboleita

- Olkoon P ja Q *propositioita* eli totuusarvoisia lauseita, jotka kuvaavat jonkin asiantilan.
- Esimerkiksi $P =$ "$\sqrt{2}$ on kokonaisluku", on propositio jonka totuusarvo on epätosi ja $Q =$ "$\sqrt{4}$ on kokonaisluku" on propositio jonka totuusarvo on tosi.
- Myös $P(x) =$ "$x < 5$" ja $Q(x) =$ "x on parillinen" ovat propositioita, jos luvun x arvo on kiinnitetty. Muuten P:tä kutsutaan predikaatiksi. P:n totuusarvo riippuu nyt x:n arvosta.
- *Negaatio* $\neg P$: tosi kun P on epätosi (ohjelmointikielissä usein `not` P, `!` P)
- *Disjunktio* $P \vee Q$: tosi kun P tai Q on tosi (sisältää vaihtoehdon, että molemmat ovat tosia) (P `or` Q, P `||` Q)
- *Konjunktio* $P \wedge Q$: tosi kun sekä P että Q ovat tosia (P `and` Q, P `&&` Q)
- *Implikaatio* $P \rightarrow Q$: "jos P, niin Q", tosi kun $(\neg P) \vee Q$ on tosi. (Toisinaan käytetään merkintää $P \Rightarrow Q$.)
- *Ekvivalenssi* $P \leftrightarrow Q$: "P jos ja vain jos Q", tosi kun $(P \rightarrow Q) \wedge (Q \rightarrow P)$ on tosi. (Toisinaan käytetään merkintää $P \Leftrightarrow Q$.)

- *Universaalikvantifiointi* $\forall x : P(x)$: ”kaikilla x $P(x)$”, tosi kun x:stä riippuva preidikaatti P on tosi kaikilla mahdollisilla x:n arvoilla.
- *Eksistentiaalikvantifiointi* $\exists x : P(x)$: ”on olemassa x jolle $P(x)$”, tosi kun P on tosi jollakin x:n arvolla.
- Edellisissä perusjoukko, jossa x saa arvoja, on implisiittinen, mutta se voidaan ilmaista myös eksplisiittisesti.
- Esim. $\forall x \in \mathbb{N}$:(“x parillinen” $\vee$ “x pariton”) on tosi, mutta $\exists x \in \mathbb{N}$:(“x parillinen” $\wedge$ “x pariton”) on epätosi.

Joukot

- *Joukko* on kokoelma olioita eli joukon *alkioita*
  Esim. $A = \{a_1, a_2, ..., a_n\}$ tai $\Sigma = \{a, b, c, ..., z\}$
  Merk. $a_i \in A$ (”a_i kuuluu joukkoon A”)
- Jokainen alkio lasketaan vain kerran eikä järjestyksellä ole väliä, siis esim. $\{a, a, b\} = \{b, a\}$. Joukoille A ja B merkintä $A = B$ tarkoittaa, että $\forall x : x \in A \Leftrightarrow x \in B$.
- Erikoistapaus: *tyhjä joukko* $\emptyset$, jolle $\forall x : x \notin \emptyset$.
- $|A|$ on joukon A alkioiden lukumäärä, esim. $|\{1, 1, 2\}| = 2$.
- *Perusjoukko* on tarkastelun kohteeksi valittu kokoelma olioita, esimerkiksi luonnolliset luvut $\mathbb{N}$, reaaliluvut $\mathbb{R}$, kirjaimet { `a, b, ..., ä, ö` } (riippuu asiayhteydestä)

- Merkinnällä $\{x \in A \mid P(x)\}$ tarkoitetaan joukkoa, joka koostuu niistä $x \in A$, joille $P(x)$ on tosi.
- Esim. $\{x \in \mathbb{N} \mid x > 5 \wedge x < 10\}$ on joukko $\{6, 7, 8, 9\}$
- $A \subseteq B$: A on B:n *osajoukko*

$$A \subseteq B \Leftrightarrow (\forall x : x \in A \Rightarrow x \in B)$$

- Joukko-operaatioita:
  - $A \cup B$: A:n ja B:n *yhdiste*

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

  - $A \cap B$: A:n ja B:n *leikkaus*

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

  - $A \setminus B$: A:n ja B:n *erotus*

$$A \setminus B = \{x \mid x \in A \wedge x \notin B\}$$

  - $\overline{A} = E \setminus A$: A:n *komplementti* (perusjoukossa E)
  - $A \times B$: A:n ja B:n *karteesinen tulo*:

$$A \times B = \{(x, y) \mid x \in A \wedge y \in B\},$$

    missä kukin (x, y) on *järjestetty pari*. Järjestetyissä pareissa järjestyksellä on väliä – ts. $(a, b) = (c, d) \Leftrightarrow a = c$ ja $b = d$.
  - $\mathcal{P}(A)$: joukon A *potenssijoukko*

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

    eli joukon kaikista mahdollisista osajoukoista muodostuva joukko. Joskus potenssijoukkoa merkitään 2^A.

- Esimerkki:
  - Olkoon $A = \{6, 7, 8\}$ ja
    $B = \{x \in \mathbb{N} \mid x > 2 \ \wedge \ x < 10 \wedge \text{x parillinen}\} = \{4, 6, 8\}$.
  - $A \cap B = \{6, 8\}$
  - Nyt $A \cap B \subseteq A$ ja $A \cap B \subseteq B$
  - $A \cup B = \{4, 6, 7, 8\}$
  - $A \setminus B = \{7\}$
  - $\overline{B} = \{x \in \mathbb{N} \mid (x \leq 2 \vee x \geq 10) \wedge \text{x pariton}\}$
  - $A \times B = \{(6, 4), (6, 6), (6, 8), (7, 4), (7, 6), (7, 8), (8, 4), (8, 6), (8, 8)\}$
  - $\mathcal{P}(A) = \{\emptyset, \{6\}, \{7\}, \{8\}, \{6, 7\}, \{6, 8\}, \{7, 8\}, \{6, 7, 8\}\}$
  - Huom: Alkio 6 **ei** kuulu joukkoon, $\mathcal{P}(A)$, sen sijaan kyllä $\{6\} \in \mathcal{P}(A)$, eli joukko jonka ainoa alkio on 6 kuuluu kyllä A:n potenssijoukkoon.
- Esim. Todistamme väitteen $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$.
  Tod.:

$$\begin{aligned} x \in (A \cup B) \cap C &\Leftrightarrow x \in A \cup B \wedge x \in C \\ &\Leftrightarrow (x \in A \vee x \in B) \wedge x \in C \\ &\Leftrightarrow (x \in A \wedge x \in C) \vee (x \in B \wedge x \in C) \\ &\Leftrightarrow (x \in A \cap C) \vee (x \in B \cap C) \\ &\Leftrightarrow x \in (A \cap C) \cup (B \cap C). \ \square \end{aligned}$$

Funktiot

- Merkinnällä
$$f : A \to B$$
tarkoitetaan että f on funktio (lähtö)joukolta A (maali)joukolle B
- Funktio f liittää jokaiseen alkioon $a \in A$ yksikäsitteisen alkion $f(a) \in B$. Sanotaan, että f kuvaa joukon A alkion a joukon B alkioksi $f(a)$.
- Esim. Määritellään $f : \mathbb{N} \to \mathbb{N}$ kaavalla $f(x) = x + 1$. Nyt f on funktio joka kuvaa kokonaisluvun x sen seuraajalle, esim. $f(2) = 3$.
- Olkoon $Q = \{q_0, \ldots, q_n\}$ ja $\Sigma = \{a_0, \ldots a_m\}$.
  Jos $f : Q \times \Sigma \to Q$, niin f kuvaa parin (q_i, a_i) missä $q_i \in Q$ ja $a_i \in \Sigma$ alkioksi $f(q_i, a_i)$ joka kuuluu joukkoon Q.
  Esim. voisi olla $f(q_1, a_2) = q_4$.
  Jos taas $f : Q \times \Sigma \to \mathcal{P}(Q)$, niin f kuvaa parin (q_i, a_i) joukon $\mathcal{P}(Q)$ alkioksi $f(q_i, a_i)$. Siis $f(q_i, a_i)$ on Q:n osajoukko.
  Esim. voisi olla $f(q_1, a_2) = \{q_2, q_4, q_5\}$ tai $f(q_1, a_3) = \emptyset$.
- Funktio $f : A \to B$ on *bijektio*, jos
  - $\forall b \in B : \exists a \in A : f(a) = b$ (f on *surjektio*)
  - $\forall a, a' \in A : f(a) = f(a') \Rightarrow a = a'$ (f on *injektio*).

Todistusmenetelmiä

Halutaan todistaa, että oletuksesta A seuraa väite B eli $A \Rightarrow B$.

Todistamiseen on useita erilaisia tekniikoita. Se, mikä kulloinkin sopii parhaiten, riippuu tilanteesta.

- *Deduktiivinen todistus*: Oletetaan A ja näytetään, että tästä seuraa B. (Kutsutaan myös suoraksi todistukseksi.)
  - Esim. Olkoon $x \in \mathbb{N}$ pariton. Väite: x^2 on pariton.
    Tod.: Jos $x \in \mathbb{N}$ on pariton (oletus), niin $x = 2k + 1$ jollain $k \in \mathbb{N}$ (parittomuuden määritelmä). Silloin
    $$x^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1, \text{ eli } x^2 \text{ on pariton.} \square$$
- *Kontrapositiivinen todistus*: Näytetään että olettamalla $\neg B$ seuraa $\neg A$
  - Pätee, koska $(\neg B \to \neg A) \Leftrightarrow$
    $(\neg(\neg(B)) \vee (\neg A)) \Leftrightarrow (\neg A \vee B) \Leftrightarrow (A \to B)$
  - Esim. Väite: Jos $x > 0$ on irrationaaliluku, niin $\sqrt{x}$ on irrationaaliluku.
    Tod.: Osoitetaan, että jos $\sqrt{x}$ on rationaaliluku, niin x on rationaaliluku. Siis $\sqrt{x} = p/q$, joillakin $p, q \in \mathbb{Z}, q \neq 0$, joten $x = \frac{p^2}{q^2}$ ja $p^2, q^2 \in \mathbb{Z}, q^2 \neq 0$. $\square$
- *Vastaväitetodistus (Reductio ad absurdum)*: Näytetään että oletuksesta $A \wedge \neg B$ seuraa ristiriita. (Kutsutaan myös epäsuoraksi todistukseksi.)

– Pätee, koska $(A \wedge \neg B \rightarrow F) \Leftrightarrow (\neg(A \wedge \neg B)) \Leftrightarrow (\neg A \vee B) \Leftrightarrow (A \rightarrow B)$

– Esim. Väite: Jos $a + b = 2$, niin $a \geq 1$ tai $b \geq 1$.
Tod.: Oletetaan, että $a+b = 2$ ja että $a < 1$ ja $b < 1$. Tällöin $2 = a + b < 1 + 1 = 2$, ristiriita (F).

• *Induktio*

– Menetelmä on seuraava:
Halutaan todistaa, että jokin väite $P(n)$ pätee kaikille luonnollisille luvuille n. Todistus koostuu kahdesta osasta:

1. Perustapaus $n = 0$: Todistetaan, että väite $P(0)$ pätee
2. Induktioaskel: Osoitetaan, että kaikilla $n \in \mathbb{N}$: $P(n) \rightarrow P(n + 1)$.

Induktioperiaatteen nojalla kohdista 1+2 seuraa, että väite $P(n)$ pätee kaikilla $n \in \mathbb{N}$.
($P(0)$ pätee, josta saadaan induktioaskeleella $P(1)$, tästä puolestaan $P(2)$ jne.)
(Variaatio kohdasta 2: Osoitetaan, että kaikille $n \in \mathbb{N}$: $P(0) \wedge P(1) \wedge \ldots \wedge P(n) \rightarrow P(n + 1)$).

– Esimerkki induktiotodistuksesta.
Väite: $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$ kaikilla $n \geq 0$
Tod:

1. $n = 0$: $\sum_{i=0}^{0} i = 0 = \frac{0 \cdot 1}{2}$
2. Induktio-oletus: $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.
Tapaus $n+1$: $\sum_{i=0}^{n+1} i = \sum_{i=0}^{n} i + (n+1) =$ (induktio-oletuksen mukaan) $\frac{n(n+1)}{2} + \frac{2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$.□

- *Epätodeksi* voidaan osoittaa antamalla vastaesimerkki
  – Esim. Väite “Kaikki alkuluvut ovat parittomia” on epätosi.
  Tod.: 2 on alkuluku ja parillinen. □
- *Diagonalisointi* esitetään myöhemmin.

Numeroituvuus

- Joukko X on *äärellinen*, jos on olemassa bijektio

$$f : \{1, 2, \ldots, n\} \to X$$

jollakin $n \in \mathbb{N}$. Muuten joukko on *ääretön*.

- Miten mitata äärettömän monta alkiota sisältävän joukon kokoa?

- *Määritelmä:* Joukko X on *numeroituva*, jos X on äärellinen tai on olemassa bijektio $f : \mathbb{N} \to X$, eli

  1. f on funktio joka kuvaa jokaisen luonnollisen luvun erilliselle joukon X alkiolle
  2. ja kääntäen: jokaiselle X:n alkiolle kuvautuu jokin luonnollinen luku

- Toisin sanoen äärettömän numeroituvan X:n alkiot voidaan järjestää ja antaa niille yksikäsitteiset järjestysnumerot: $X = \{x_0, x_1, x_2, \ldots\}$, missä $x_0 = f(0), x_1 = f(1), \ldots$

- Jos X ei ole numeroituva, on se *ylinumeroituva*.

- Jos X ja Y ovat äärettömiä joukkoja, joista X numeroituva ja Y ylinumeroituva, niin joukossa Y on "enemmän" alkioita – joukko $\mathbb{N}$ riittää numeroimaan X:n alkiot, sen sijaan joukosta $\mathbb{N}$ ei riitä numeroa jokaiselle Y:n alkiolle.

- Parittomien luonnollisten lukujen joukko $\{1, 3, 5, \ldots\}$ on numeroituva, sillä funktio $f(n) = 2n + 1$ kuvaa luonnolliset luvut joukolle bijektiivisesti. Jokaista lukua joukossa $\mathbb{N}$ siis vastaa luku joukossa $\{1, 3, 5, \ldots\}$, joten parittomia lukuja on tässä mielessä "yhtä paljon" kuin kokonaislukuja.

- Esim. reaalilukujen joukko $\mathbb{R}$ ja $\mathbb{N}$:n potenssijoukko $\mathcal{P}(\mathbb{N})$ ovat esimerkkejä ylinumeroituvista joukoista. Ne ovat siis "suurempia" kuin ääretön $\mathbb{N}$.

1.3. Laskennalliset ongelmat

- Laskennallinen ongelma $\sim$ mikä tahansa syöte-tuloste -muotoinen tehtävä, joka voidaan mallintaa ratkaistavaksi digitaalisella tietokoneella
- Esimerkkejä: kokonaislukujen kertolasku, kirjastokortiston aakkostaminen, ...

Formalisointi

- Ongelmalla on 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 (äärellisiksi) 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*

Merkkijonoihin liittyviä peruskäsitteitä ja merkintöjä

- *Aakkosto* on äärellinen epätyhjä joukko alkeis*merkkejä* t. *symboleita*. Esim. *binääriaakkosto* $\{0, 1\}$ ja *latinalainen aakkosto* $\{\text{A, B, } \ldots, \text{ Z}\}$. Aakkostoja merkitään yleensä isoilla kreikkalaisilla kirjaimilla ja merkkejä pienillä latinalaisilla kirjaimilla. Aakkoston Σ koko on $|\Sigma|$.
- *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* $|x|$ on siihen sisältyvien merkkien määrä. Esim.

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

- *Tyhjän merkkijonon* ϵ pituus $|\epsilon|$ on 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 $|xy| = |x| + |y|$

- a^k tarkoittaa merkkijonoa $\underbrace{a \ldots a}_{k \text{ kertaa}}$.

- Merkkijonon $x = x_1 x_2 \ldots x_{k-1} x_k$ käänteinen merkkijono on $x^R = x_k x_{k-1} \ldots x_2 x_1$

- Merkitään k:n pituisia aakkoston Σ merkkijonoja $\Sigma^k = \{x_1 x_2 \ldots x_k \mid \forall i = 1, \ldots, k : x_i \in \Sigma\}$ ja korkeintaan k:n pituisia aakkoston Σ merkkijonoja $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$.

- Aakkoston Σ kaikkien (äärellisten) merkkijonojen joukko on $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$. Merkitään myös $\Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k$. Esim. Jos $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, \ldots\}$, $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$

Formaali kieli

- Aakkoston Σ *(formaali) kieli* (engl. formal language) L on merkkijonojoukko $L \subseteq \Sigma^*$.
- Aakkoston Σ formaalin kielen L alkiot ovat siis Σ:n merkkijonoja ja L itse on joukko merkkijonoja.
- Esimerkkejä:
  - $\Sigma = \{a, b\}$, kieliä ovat esim. seuraavat:
    $L_1 = \{\epsilon, aa, aba, abba, abbba\}$ ja $L_2 = \{ab^i \mid i \in \mathbb{N}\} = \{a, ab, abb, abbb, \ldots\}$.
  - $\Sigma = \{0, 1, \ldots, 9\}$, seuraavakin on Σ:n kieli:
    $L_3 = \{w \in \Sigma^* \mid w$ on alkuluvun desimaaliesitys$\} = \{2, 3, 5, 7, 11, 13 \ldots\}$.
  - $\Sigma = \Sigma_{\text{ascii}}$ eli kaikkien ASCII-merkkien joukko [1]. Σ_{ascii}:n kieliä ovat esim:
    $L_4 = \{w \in \Sigma^* \mid$ merkkijono w esittää Javan syntaksin mukaisen liukuluvun$\}$,
    $L_5 = \{w \in \Sigma^* \mid$ merkkijono w esittää syntaktisesti oikean **olpe03**-kielisen ohjelman$\}$, ja
    $L_6 = \{w \in \Sigma^* \mid$ merkkijono w esittää suomen kielen verbin perusmuodossa$\}$.
  - Kielen L_6 määrittely on ongelmallinen, sillä aina ei ole helppo sanoa esittääkö merkkijono perusmuotoista verbiä (esim. onko kömmäytellä verbi?).

[1] Javan tapauksessa pitäisi oikeastaan tarkastella Unicode-merkistöä.

- Edellä määritelty kieli L_5 siis sisältää kaikki syntaktisesti oikeat **olpe03**-kieliset ohjelmat.
- Nyt **olpe03**-kielen syntaksitarkastus voidaan palauttaa kysymykseen, kuuluuko syötteenä saatu ASCII-merkkijono ($\sim$ lähdekooditiedosto) w kieleen L_5. Jos w on sallittua muotoa (katso sivun 3 esimerkki), niin $w \in L_5$, muuten $w \notin L_5$.
- Hyviä kysymyksiä:
  - Miten formaaleja kieliä kannattaa määritellä käytännössä (sovelluksena esim. ohjelmointikielten syntaksin määrittely)
  - Miten testata (tehokkaasti) kuuluuko annettu merkkijono formaaliin kieleen (sovelluksena esim. ohjelmalistauksen syntaksitarkastus)
- Kurssin edetessä opimme muutamia keinoja molempiin edellisistä.

Päätösongelmien ja formaalien kielten vastaavuus

- Yleisesti laskennallinen ongelma f on kuvaus

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

  missä Σ ja Γ ovat aakkostoja.

- Päätösongelmat ovat tärkeä laskennallisten ongelmien aliluokka, jossa kunkin ongelman tapauksen vastaus on “kyllä” tai “ei”, siis $f : \Sigma^* \to \{0, 1\}$.

- Esimerkiksi päätösongelma “onko annettu luonnollinen luku alkuluku?” voidaan esittää kuvauksena aakkoston $\Sigma = \{0, 1, 2, \ldots, 9\}$ merkkijonojen joukolta Σ^* joukolle $\{0, 1\}$ seuraavasti:

$$f(x) = \begin{cases} 1, & \text{jos } x \text{ on alkuluvun desimaaliesitys;} \\ 0, & \text{jos } x \text{ ei ole alkuluvun desimaaliesitys.} \end{cases}$$

- Päätösongelma “onko annettu merkkijono **olpe03**:n syntaksin mukainen ohjelma” voidaan esittää kuvauksena $f : \Sigma^*_{\text{ascii}} \to \{0, 1\}$, missä Σ_{ascii} on sivulla 20 määritelty ASCII-aakkosto:

$$f(w) = \begin{cases} 1, & \text{jos } w \text{ on } \textbf{olpe03}\text{:n syntaksin mukainen;} \\ 0, & \text{muuten} \end{cases}$$

- Koska määrittelimme että kieli L_5 sisältää täsmälleen kaikki **olpe03**:n syntaksin mukaiset ohjelmat, voidaan f määritellä myös seuraavasti:

$$f(w) = \begin{cases} 1, & w \in L_5 \\ 0, & w \notin L_5 \end{cases}$$

- Yleisemmin, jokaista merkkijonojoukkoa eli kieltä $A \subseteq \Sigma^*$ vastaa päätösongelma

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

- Kääntäen, jokaista päätösongelmaa $f : \Sigma^* \to \{0,1\}$ vastaa merkkijonojoukko, eli kieli

$$A_f = \{x \in \Sigma^* \mid f(x) = 1\},$$

  joka siis on niiden ongelman tapausten joukko, joihin vastaus on “kyllä”.

- Kielen A *tunnistusongelma* (engl. recognition problem) = merkkijonojoukkoon liittyvä päätösongelma f_A

- Ts. formaali kieli ja päätösongelma ovat eri formalismeja samalle asialle.

Laskennallisten ongelmien ratkeavuus

- Sanotaan, että ohjelma P *ratkaisee* laskentaongelman f, jos kullakin syötteellä x ohjelma P laskee ja tulostaa arvon $f(x)$ (x ja $f(x)$ siis joidenkin aakkostojen merkkijonoja).
- 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.
- Laskentaongelmia on siis tietyssä mielessä “enemmän” kuin niiden mahdollisia ratkaisuja, ja siksi *millään ohjelmointikielellä ei voida laatia ratkaisuja kaikille laskentaongelmille*.

- *Lause:* 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 (*leksikografiseen eli kanoniseen järjestykseen*):

  1. ensin luetellaan 0:n mittaiset merkkijonot (= ϵ), 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 $f : \mathbb{N} \to \Sigma^*$ on:

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

Siis k:n pituiset merkkijonot saavat listassa numerot

$$\sum_{i=0}^{k-1} n^i, \ldots, \left(\sum_{i=0}^{k} n^i \right) - 1.$$

□

Lause: Minkä tahansa aakkoston Σ päätösongelmien joukko on ylinumeroituva.
Todistus: (diagonalisointi)

Merkitään kaikkien Σ:n päätösongelmien kokoelmaa F:llä:

$$\mathcal{F} = \{f \mid f \text{ on kuvaus } \Sigma^* \to \{0,1\}\}.$$

Vastaväite: Oletetaan, että $\mathcal{F}$ on numeroituva, eli on olemassa numerointi

$$\mathcal{F} = \{f_0, f_1, f_2, \ldots\}.$$

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

Muodostetaan uusi päätösongelma f':

$$f' : \Sigma^* \to \{0,1\}, \qquad f'(x_i) = \begin{cases} 1, & \text{jos } f_i(x_i) = 0; \\ 0, & \text{jos } f_i(x_i) = 1. \end{cases}$$

f' $\searrow$	f_0	f_1	f_2	f_3	$\cdots$
x_0	$\not{\emptyset}^{1}$	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	$\not{\emptyset}^{1}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

- Koska $\mathcal{F}$ sisältää kaikki päätösongelmat ja se on oletettu numeroituvaksi, täytyisi muodostetun "diagonaalifunktion" f' olla sama kuin joku funktioista $f_0, f_1, \ldots$
- Voiko f' olla f_0? Ei sillä määrittelimme $f'(0) = 0$ joss $f_0(0) = 1$, eli ainakin argumentilla 0 funktiot f' ja f_0 saavat eri arvon, eivätkä ne siis voi olla sama funktio.
- Entä voiko f' olla f_1? Ei sillä $f'(1) = 0$ joss $f_1(1) = 1$, eli ainakin argumentilla 1 funktiot f' ja f_1 saavat eri arvon.
- Vastaavalla päättelyllä huomaamme että f' ei voi olla mikään funktioista $f_2, f_3, \ldots$. Siis oletus siitä että $\mathcal{F}$ olisi numeroituva on hylättävä:

  $\mathcal{F}$ on siis ylinumeroituva.

- 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 siis *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

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 kuvauksellaan. Saadaan ristiriita:

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

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

2. Säännölliset kielet ja äärelliset automaatit

- Edellisessä luvussa määrittelimme aakkoston Σ formaalin kielen käsitteen. Esille nousseita tärkeitä kysymyksiä olivat mm.
  - Miten formaaleja kieliä voi määritellä käytännöllisellä tavalla?
  - Miten voi testata (tietokoneella) kuuluuko annettu merkkijono määriteltyyn formaaliin kieleen?
- Tarkastellaan Σ_{ascii} aakkoston kieltä

$$\text{HALT} = \{P, x \mid P \text{ on ohjelmalistaus ja } x \text{ syöte, jolla ohjelman } P \text{ suoritus pysähtyy.}\}.$$

- Edellisessä luvussa totesimme että tämä ns. pysähtymisongelma ei ole ratkeava — ei ole olemassa tietokoneohjelmaa, joka annetusta syötteestä pystyy päättelemään kuuluuko se kieleen HALT.
- Tarkastellaankin seuraavassa rajoitetumpia muotoja formaalien kielten määrittelyyn, joiden avulla määritellyt kielet ovat *ratkeavia*. Tällöin annetun merkkijonon kuuluvuus kieleen on ratkaistavissa algoritmisesti.
- Tarkastelemme kahta rinnakkaista ja toisiaan täydentävää menetelmää, *äärellisiä automaatteja* sekä *säännöllisiä lausekkeita*.

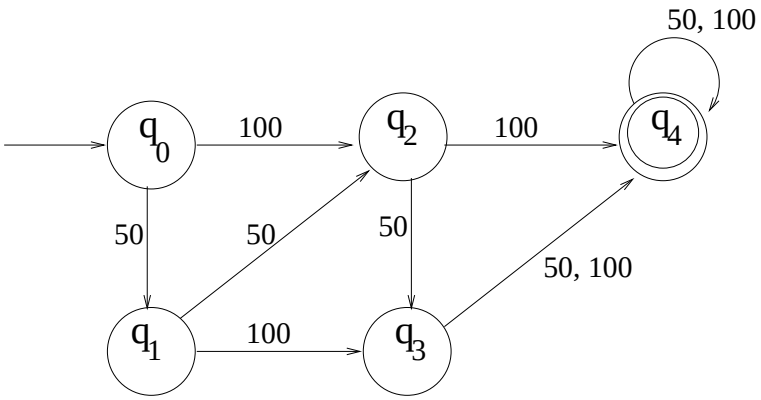
2.1. Äärelliset automaatit

- 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 kolikkojonot (yksikkönä snt):
  50 50 50 50
  100 100
  50 100 100
  100 50 50 100
- Kelvolliset syötejonot voidaan kuvata formaalina kielenä. Aakkostona on $\Sigma = \{50, 100\}$, ja automaatin hyväksymät kolikkojonot vastaavat kieltä

$$L_{\text{kahvi}} = \{a_1 \dots a_n \in \Sigma^* \mid a_1 + \dots + a_n \geq 200\}.$$

- Kahviautomaatin toiminta voidaan kuvata *äärellisenä automaattina*.
- Kahviautomaattia vastaavan äärellisen automaatin syötteitä ovat aakkoston $\{50, 100\}$ merkkijonot, ja automaatti hyväksyy syötteen jos ja vain jos se kuuluu kieleen L_{kahvi}.

- Tilasiirtymäkaavio

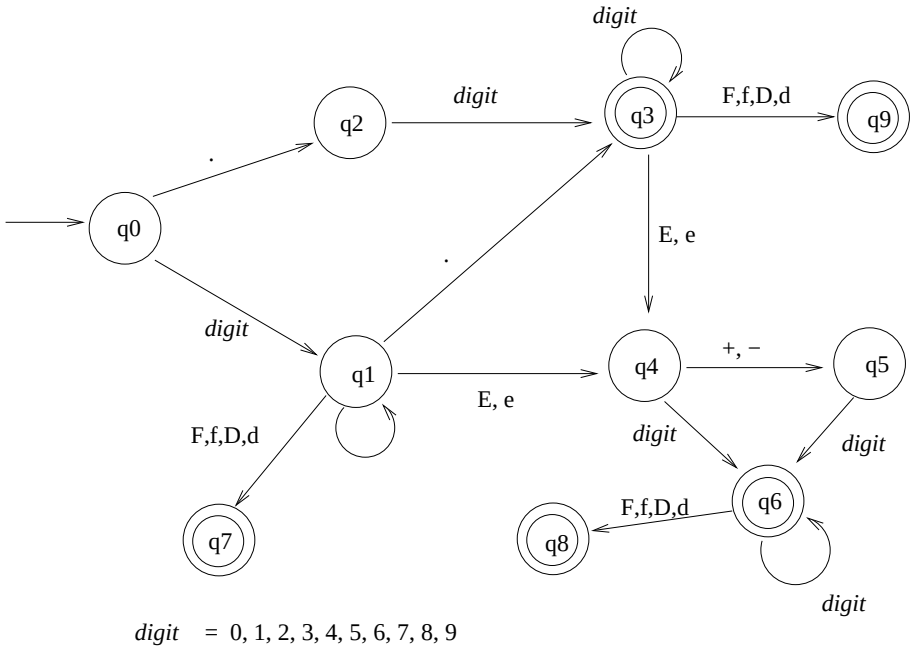


- Tilasiirtymätaulukko

	50	100
$\rightarrow q_0$	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

- Automaatin toimintaidea on seuraava:
  - Automaatti saa syötteekseen merkkijonon, jota se käsittelee merkki kerrallaan, aloittaen toimintansa alkutilasta q_0.
  - Jokaisella laskenta-askeleella automaatti
    - lukee seuraavan syötemerkkijonon merkin ja
    - siirtyy tilasiirtymäkaaviotaan seuraten luetusta merkistä riippuvaan tilaan.
  - Jos automaatti on koko syötteen luettuaan hyväksyvässä tilassa q_4, syöte hyväksytään, muuten syöte hylätään.

- Esim. Javan etumerkittömät liukulukuliteraalit (float, double) määritellään seuraavasti:
  - (kokonaisosa).(desimaaliosa) (e tai E) $[+$ tai $-]$ (eksponetti) $[$suffiksi$]$
  - kokonaisosa, desimaaliosa ja eksponentti koostuvat merkeistä $0, \ldots, 9$
  - suffiksi: F tai f: float, D tai d: double. Jos suffiksia ei ole, oletustyyppi on double.
  - luvussa täytyy esiintyä ainakin yksi numero (kokonais- tai desimaaliosassa) ja joko desimaalipiste, eksponenttiosa, tai suffiksi, mutta muuten em. osat ovat valinnaisia.
  - eksponentin etumerkki saa puuttua.
- Liukulukuliteraalin tunnistava automaatti:



- Tilasiirtymätaulukkona:

		0,1,. . . ,9	.	E, e	+, -	f, F, d, D
$\rightarrow$	q_0	q_1	q_2			
	q_1	q_1	q_3	q_4		q_7
	q_2	q_3				
$\leftarrow$	q_3	q_3		q_4		q_9
	q_4	q_6			q_5	
	q_5	q_6				q_8
$\leftarrow$	q_6	q_6				
$\leftarrow$	q_7					
$\leftarrow$	q_8					
$\leftarrow$	q_9					
	q_{-1}					

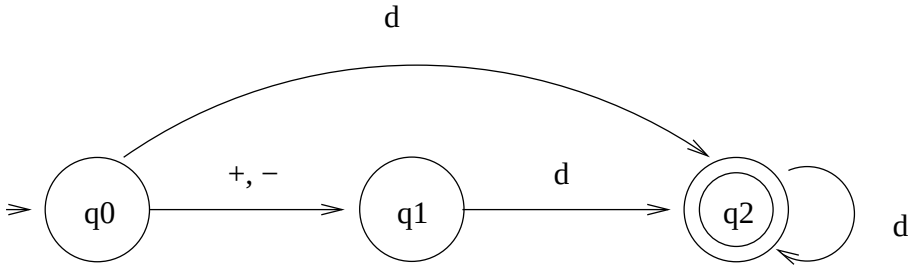
Taulukossa on yhdistelty identtisiä sarakkeita, oikeasti esim. jokaiselle numerolle $0, \ldots, 9$ pitäisi olla oma sarake. Taulukon tyhjissä kohdissa on virhetila q_{-1}, joka on jätetty merkitsemättä selkeyden takia. Lisäksi taulukkoon ei ole merkitty sarakkeita niille merkeille, jotka eivät koskaan voi esiintyä liukulukuliteraaleissa — näiden sarakkeiden jokainen alkio olisi virhetila q_{-1}.

- Ohjelmana (melkein Javaa):

```
int IsDigit(char c);  /* palauttaa 1, jos c on numero,
                         0 muuten */
char NextChar(); /* palauttaa seuraavan syötemerkin
                    tai symbolin EOF */

int q=0;
char c;
while((c = NextChar()) != EOF) {
  switch(q) {
  case 0:
    if(IsDigit(c)) q=1;
    else if (c == '.') q=2; else q=-1;
    break;
  case 1:
    if(IsDigit(c)) q=1;
    else if(c == '.') q=3;
    else if(c == 'e' || c=='E') q=4;
    else if(c == 'f' || c == 'F' || c == 'd' || c == 'D')
      q=7;
    else q=-1;
    break;
  case 2:
    if (IsDigit(c)) q=3; else q=-1;
    break;
  case 3:
    if (IsDigit(c)) q=3;
    else if(c == 'e' || c == 'E') q=4;
    else if(c == 'f' || c == 'F' || c == 'd' || c == 'D')
      q=9;
    else q=-1;
    break;
  case 4:
    if (IsDigit(c)) q=6;
    else if (c == '+' || c == '-') q=5; else q=-1;
    break;
  case 5: if (IsDigit(c)) q=6;
    else if(c == 'f' || c == 'F' || c == 'd' || c == 'D')
      q=8;
    else q=-1;
    break;
  case 6: if (IsDigit(c)) q=6; else q=-1;
    break;
  case 7:
  case 8:
  case 9:
    q=-1;
    break;
  case -1: /* virhetila */
    break;
  }
}
if ( q==3 || q>=6 ) System.out.println("luku OK");
else System.out.println("Virheellinen luku");
```

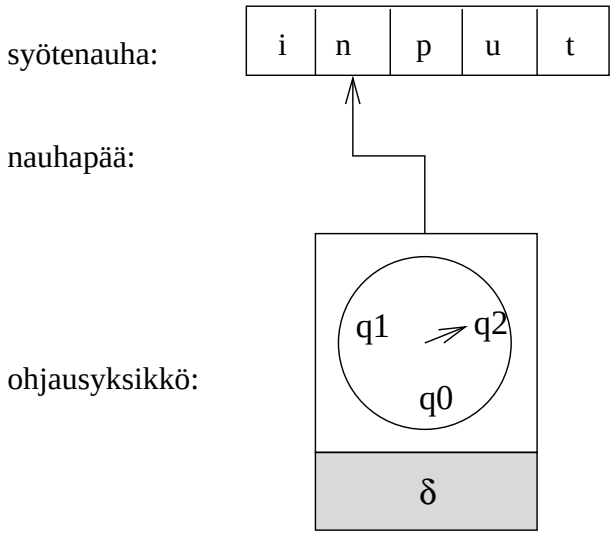
- Äärellisen automaatin pohjalta laadittuun ohjelmaan voidaan liittää myös semanttisia toimintoja. Sen lisäksi että tarkistetaan kuuluuko jokin merkkijono kieleen voidaan esim.
  - laskea merkkijonon arvo (esim. kahviautomaatin tapauksessa syötyjen kolikoiden arvojen summa)
  - tallettaa merkkijono tai sen osia johonkin tietorakenteeseen
  - tulostaa jotain nähtyjen merkkien perusteella, ...
- Esim. etumerkillisen kokonaisluvun desimaaliesityksen tunnistaminen:



- Automaattia vastaava ohjelma, joka lisäksi laskee merkkijonon esittämän luvun arvon:

```
int IsDigit(char c);  /* palauttaa 1, jos c on numero,
                         0 muuten */
char NextChar(); /* palauttaa seuraavan syötemerkin
                    tai symbolin EOF */
int q=0; char c; int sign=1; int val=0;
while((c = NextChar()) != EOF) {
  switch(q) {
  case 0:
    if (c == '+' || c == '-') {
      q=1;
      if (c=='-') sign=-1;
    }
    else if (IsDigit(c) {
      q=2;
      val=c-'0';
    }
    else q=-1;
    break;
  case 1:
    if (IsDigit(c) {
      q=2;
      val=c-'0';
    }
    else q=-1;
    break;\\
  case 2:
    if(IsDigit(c)) {
      q=2;
      val=10*val+(c-'0');
    }
    else q=-1;
    break;
  case -1:
    break;
  }
}
if(q==2) System.out.println("luvun arvo on " + sign*val);
else System.out.println("Virheellinen luku");
```

Äärellisen automaatin formaali määrittely



- Äärellinen automaatti M koostuu seuraavista osista:
  - 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, siirtyy ohjausyksikön tilan ja luetun merkin perusteella siirtymäfunktion δ mukaisesti uuteen tilaan, 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 automaatti *hylkää* syötteen.
  - 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. liukulukuliteraalit tunnistavan automaatin formaali esitys:
$$\begin{aligned} M \;=\; & (\{q_0, \ldots, q_9, q_{-1}\}, \Sigma_{\text{ascii}}, \\ & \delta, q_0, \{q_3, q_6, q_7, q_8, q_9\}), \end{aligned}$$
missä δ on kuten aiemmin taulukossa; esim.
$$\delta(q_0, \texttt{0}) = \delta(q_0, \texttt{1}) = \cdots = \delta(q_0, \texttt{9}) = q_1,$$
$$\delta(q_0, .) = q_2, \quad \delta(q_0, \texttt{E}) = q_{-1}, \quad \delta(q_1, \texttt{E}) = q_4 \quad \text{jne.}$$

- Automaatin *tilanne* on pari $(q, w) \in Q \times \Sigma^*$
  - q on nykyinen tila ja w on syötemerkkijonon käsittelemätön osa
  - automaatin *alkutilanne syötteellä* x on pari (q_0, x)
  - (q, ϵ) on lopputilanne ja jos $q \in F$ niin (q, ϵ) on hyväksyvä lopputilanne.
- Tilanne (q, w) *johtaa suoraan* tilanteeseen (q', w'), merk.

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

  jos $w = aw'$ ($a \in \Sigma$) ja $q' = \delta(q, a)$.
  Tilanne (q', w') on tilanteen (q, w) *välitön seuraaja*

  Toisin sanoen: tilanne (q, w) johtaa suoraan tilanteeseen (q', w'), jos
  - $w = a_1 a_2 \ldots a_n$
  - $\delta(q, a_1) = q'$
  - $w' = a_2 \ldots a_n$.
- 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 tilannejono $(q_0, w_0), (q_1, w_1), \ldots, (q_n, w_n)$, $n \geq 0$, siten että

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

Erikoistapaus: $n = 0$, $(q, w) \underset{M}{\vdash}^* (q, w)$ millä tahansa tilanteella (q, w)

Toisin sanoen: tilanne (q, w) johtaa tilanteeseen (q', w') jos tilanteesta (q, w) pääsee tilanteeseen (q', w') nollalla, yhdellä, tai useammalla edellisen kohdan suoralla laskenta-askeleella.

- 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*

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

Eli automaatin tunnistama kieli sisältää *kaikki* merkkijonot jotka automaatti hyväksyy.

- Esim. merkkijonon "`0.25E2`" käsittely edellä esitetyllä liukulukuautomaatilla:

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

Koska $q_6 \in F = \{q_3, q_6, q_7, q_8, q_9\}$, pätee $\texttt{0.25E2} \in L(M)$ eli automaatti hyväksyy merkkijonon.

2.2. Automaattien minimointi

- Kaksi automaattia, jotka tunnistavat täsmälleen saman kielen ovat keskenään *ekvivalentteja*.
- Äärellinen automaatti on *minimaalinen* jos se on tilamäärältään pienin ekvivalenttien automaattien joukossa. Muuten sanotaan, että automaatissa on redundantteja tiloja.
- Miksi minimaaliset automaatit ovat kiinnostavia?
  - Minimaalisesta automaatista on usein helpompi nähdä mikä sen tunnistama kieli on.
  - On turha talletta ylimääräisiä tiloja.
  - Minimaalisen automaatin käsittely on tehokkaampaa.
  - Minimaalinen automaatti on luonnollinen (tilojen nimeämistä vaille) yksikäsitteinen edustaja toistensa kanssa ekvivalenttien automaattien joukolle.
- Automaatteja muodostavat algoritmit (ja ihmiset) eivät aina tuota minimaalisia automaatteja, mistä syystä minimointialgoritmit ovat hyödyllisiä.

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. Osita M:n jäljelle jääneet tilat kahteen luokkaan: ei-lopputiloihin ja lopputiloihin. Olkoon syntynyt ositus $\hat{Q}$.
  3. Lopeta, jos

     $$\forall \hat{q} \in \hat{Q} : \forall q, q' \in \hat{q} : \forall a \in \Sigma :$$
     $\delta(q, a)$ ja $\delta(q', a)$ kuuluvat samaan $\hat{Q}$:n luokkaan.

     Muuten olkoot $\hat{q} \in \hat{Q}$ jokin tilojen luokka, josta pääsee eri tilojen luokkiin symbolilla a.

     – Jaa $\hat{q}$ osajoukkoihin s.e. kustakin osajoukosta pääsee symbolilla a tasan yhteen joukon $\hat{Q}$ tilaluokkaan ja että eri osajoukoista pääsee symbolilla a eri tilaluokkiin.
     – Poista tilaluokka $\hat{q}$ joukosta $\hat{Q}$ ja korvaa se edellä muodostetuilla osillaan.

     Palaa kohdan 3 alkuun.

- Algoritmi palauttaa automaatin

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

  missä

  - $\hat{Q}$ = Yllä muodostunut tilajoukon Q ositus.
  - $\hat{\delta} \colon \hat{Q} \times \Sigma \to \hat{Q}$ = luokkien välinen siirtymäfunktio
    $\hat{\delta}(\hat{q}, a) =$ tilan $\delta(q, a)$ luokka osituksessa $\hat{Q}$, missä $q \in \hat{q}$ on mikä tahansa tilaluokan $\hat{q}$ tila.
  - $\hat{q}_0$ = tilaluokka, johon M:n alkutila q_0 kuuluu.
  - $\widehat{F}$ = M:n lopputilojen luokat

- Lopputulos:

  - M:n kanssa ekvivalentti äärellinen automaatti $\widehat{M}$, jossa on minimimäärä tiloja
  - $\widehat{M}$ on tilojen nimeämistä vaille yksikäsitteinen

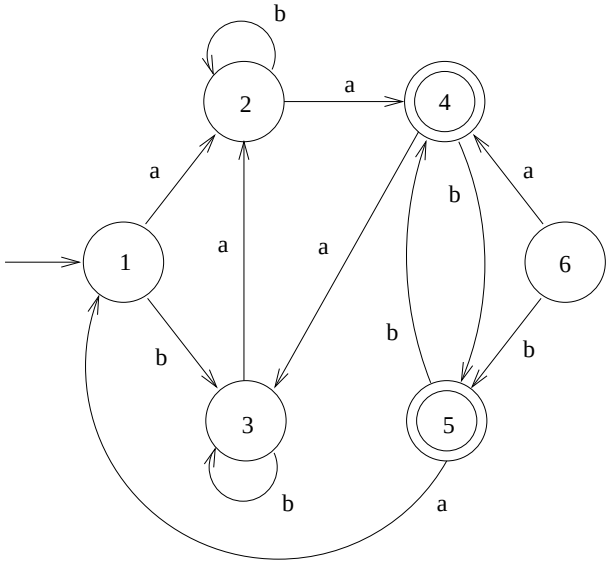
  Tämän väitteen todistus jätetään laskuharjoituksiin.

- Huom: Tiloja on alun perin äärellinen määrä ja joka askeleessa nro 3 (paitsi viimeisessä) ositetaan vähintään yksi tilaluokka. Koska osituksia voi tapahtua yhteensä korkeintaan $|Q|$ kappaletta, algoritmin suoritus päättyy aina.

Esimerkki

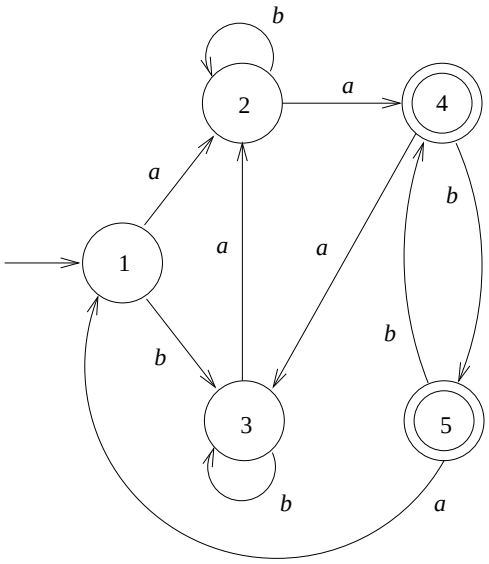
- Olkoon $M = (Q, \Sigma, \delta, q_0, F)$,
  - tilojen joukko $Q = \{1, 2, 3, 4, 5, 6\}$,
  - syöteaakkosto $\Sigma = \{a, b\}$,
  - alkutila $q_0 = \{1\}$,
  - lopputilojen joukko $F = \{4, 5\}$ ja
  - siirtymäfunktio δ ja M:n tilasiirtymäkaavio:

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



- Minimoidaan M äärellisen automaatin minimointi -algoritmilla.

- Askel 1: Turhien tilojen poisto:



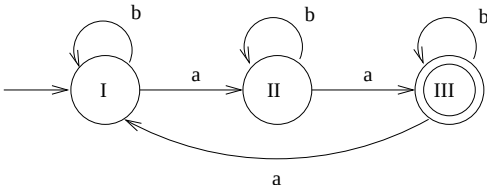
- Askel 2:
  - Osita M:n jäljelle jääneet tilat kahteen luokkaan: lopputiloihin ja muihin tiloihin

			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 ositusta voidaan ajatella automaattina, jossa
    - kutakin tilaluokkaa vastaa yksi tila
    - kustakin tilasta (M:n tilaluokasta) $\hat{q}$ on kullakin symbolilla $a \in \Sigma$ siirtymät kaikkiin tilaluokkiin, joihin $\hat{q}$:n tiloista pääsee a:lla M:n siirtymäfunktiolla.
  - Sanotaan, että tila on *epädeterministinen*, jos siitä voidaan siirtyä jollakin merkillä useampaan kuin yhteen tilaan.
  - Esimerkissä tila I on epädeterministinen, koska merkillä a voidaan siirtyä tilaan I tai tilaan II.

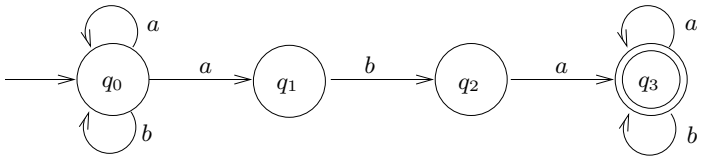
- Askel 3:
  - Jos $\widehat{M}$:ssä ei ole epädeterministisiä tiloja, niin algoritmin suoritus päättyy ja algoritmi palauttaa automaatin $\widehat{M}$.
  - Muutoin ositetaan jokin epädeterministinen tila $\hat{q} \in \hat{Q}$:
    - Valitaan symboli $a \in \Sigma$, jolla $\hat{q}$:n sisältämistä tiloista pääsee M:n siirtymäfunktiota käyttämällä eri luokissa oleviin tiloihin.
    - Jaetaan $\hat{q}$ osiin. Jokaista tilaluokkaa kohden, johon $\hat{q}$:sta pääsee, tulee yksi osa — niiden $\hat{q}$:n tilojen joukko, josta pääsee a:lla ko. tilaluokkaan.
  - Palaa askelen 3 alkuun.
  - Esimerkissämme jaetaan tila I kahtia sen mukaan, mihin sen sisältämistä tiloista pääsee merkillä a.
  - Tämän jälkeen ei ole enää epädeterminisiä tiloja ja algoritmin suoritus 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



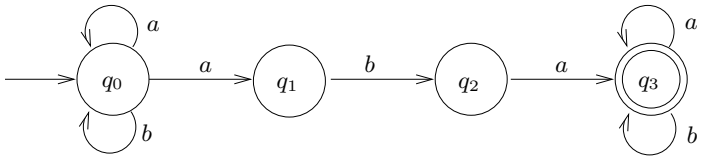
2.3. Epädeterministiset äärelliset automaatit

- Esimerkki epädeterministisestä automaatista:



- Epädeterminismi helpottaa toisinaan automaatin konstruointia. Esimerkiksi alimerkkijonon *aba* etsiminen aakkoston $\{a, b\}$ merkkijonoista on luontevasti kuvattavissa epädeterministisellä automaatilla.
- Todistamme tässä luvussa, että deterministisillä ja epädeterministisillä automaateilla voidaan tunnistaa täsmälleen samat kielet — epädeterminismi ei siis lisää automaattien ilmaisuvoimaa, mutta toisinaan tekee asioiden ilmaisemisen helpommaksi.
- Epädeterministisessä automaatissa siirtymäfunktio liittää automaatin tilan ja syötemerkin pariin (q, x) *joukon* mahdollisia seuraavia tiloja. Se, mihin näistä tiloista siirrytään, valitaan epädeterministisesti.
- Epädeterministinen automaatti hyväksyy merkkijonon jos *jokin* mahdollisten tilanteiden jono johtaa hyväksyvään lopputilaan. Jos yhtään tällaista jonoa ei ole, epädeterministinen automaatti hylkää syötemerkkijonon.

- Sama esimerkkiautomaatti uudestaan:



- Kuvan automaatti hyväksyy syötejonon *abbaba*, koska se voidaan käsitellä seuraavasti:

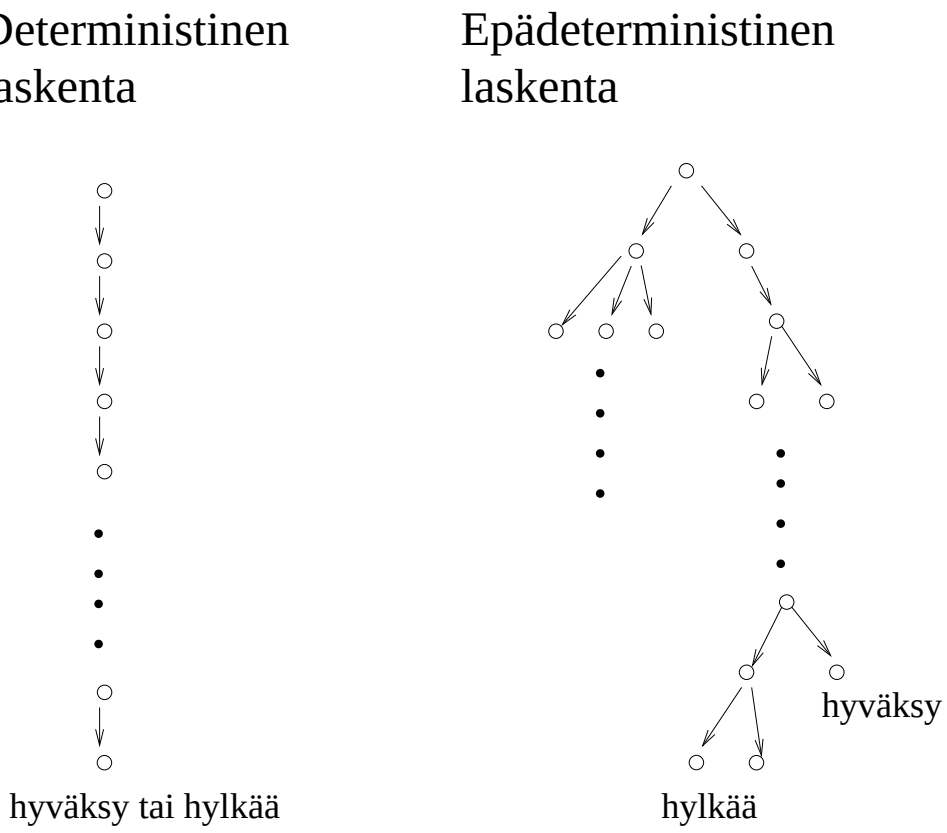
$$(q_0, abbaba) \vdash (q_0, bbaba) \vdash (q_0, baba)$$
$$\vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_2, a) \vdash (q_3, \epsilon)$$

- Kuitenkin voidaan päätyä myös hylkäävään tilaan:

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

  Niin kauan kuin hyväksyvään lopputilaankin johtavia tilannejonoja on, ei hylkäävään lopputilanteeseen päätyvillä jonoilla ole väliä.

- Epädeterministisen automaatin voidaan ajatella suorittavan kaikki mahdolliset laskennat rinnakkain:



- Toinen vaihtoehto on ajatella, että epädeterministinen automaatti löytää automaattisesti hyväksyvään lopputilanteeseen päättyvän laskentapolun jos sellainen vain on olemassa.

- *Määritelmä:* Epädeterministinen äärellinen automaatti (engl. nondeterministic finite automaton, NFA) 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

- (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 deterministisille automaateille.

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

Automaatin determinisointi

- Epädeterminististä automaattia $M = (Q, \Sigma, \delta, q_0, F)$ vastaa deterministinen automaatti $\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F})$, missä

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

- *Lause:* Olkoon $A = L(M)$ epädeterministisen äärellisen automaatin M tunnistama kieli. Tällöin on olemassa deterministinen automaatti $\widehat{M}$, jolle $L(\widehat{M}) = A$.

Todistus: Olkoon $A = L(M)$, missä $M = (Q, \Sigma, \delta, q_0, F)$. Olkoon $\widehat{M}$ kuten yllä. Osoitetaan, että $L(\widehat{M}) = L(M)$, mistä väite seuraa.

Olkoon $S_x \subseteq Q$ tilojen joukko, johon M voi päätyä luettuaan merkkijonon x. Siis

$$S_x = \{q \in Q \mid (q_0, x) \underset{M}{\vdash}^* (q, \epsilon)\}.$$

Ideana todistuksessa on osoittaa, että kaikilla $x \in \Sigma^*$ pätee:

$$(\hat{q}_0, x) \underset{\widehat{M}}{\vdash}^* (S_x, \epsilon). \tag{1}$$

Toisin sanoen: *Tila* S_x, johon deterministinen automaatti $\widehat{M}$ päätyy luettuaan merkkijonon x on *tilojen joukko* S_x, johon epädeterministinen M voi päätyä luettuaan merkkijonon x.

Väitteen (1) voi todistaa induktiolla x:n pituuden k suhteen:

– Tapaus $k = 0$: Välttämättä $x = \epsilon$, joten määritelmien mukaan

$$(\hat{q}_0, x) = (\{q_0\}, \epsilon) \underset{\widehat{M}}{\vdash}^* (\{q_0\}, \epsilon) = (S_\epsilon, \epsilon) = (S_x, \epsilon)$$

eli väite pätee tapauksessa $k = 0$.

– Induktioaskel: Oletetaan, että (1) pätee kaikilla $x \in \Sigma^k$ ja osoitetaan, että (1) pätee kaikilla $x \in \Sigma^{k+1}$. Olkoon siis $x \in \Sigma^{k+1}$. Tällöin $x = ya$ jollekin $y \in \Sigma^k$ ja $a \in \Sigma$. Induktio-oletuksesta seuraa, että $(\hat{q}_0, y) \underset{\widehat{M}}{\vdash}^* (S_y, \epsilon)$. Siis

$$(\hat{q}_0, x) = (\hat{q}_0, ya) \underset{\widehat{M}}{\vdash}^* (S_y, a) \underset{\widehat{M}}{\vdash} (\hat{\delta}(S_y, a), \epsilon).$$

Pitää enää osoittaa, että $\hat{\delta}(S_y, a) = S_x$. Siirtymäfunktion $\hat{\delta}$ määritelmän mukaan

$$\hat{\delta}(S_y, a) = \bigcup_{q' \in S_y} \delta(q', a);$$

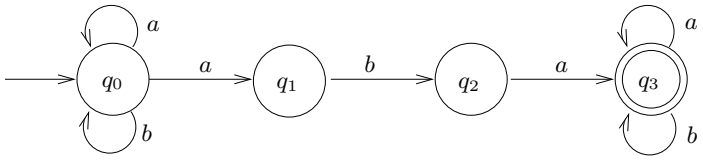
$\hat{\delta}(S_y, a)$ on siis niiden $q \in Q$ joukko, joihin pääsee siirtymäfunktiolla δ merkillä a jostakin S_y:n tilasta. Toisaalta S_y on määritelmänsä mukaan niiden tilojen joukko, joihin tilasta q_0 pääsee merkkijonolla y. Siispä $\hat{\delta}(S_y, a)$ on niiden tilojen joukko, joihin q_0:sta pääsee syötteellä $ya = x$; symbolein

$$\hat{\delta}(S_y, a) = \{q \in Q \mid (q_0, x) \underset{M}{\vdash}^* (q, \epsilon)\} = S_x.$$

Tämä todistaa induktioaskeleen. $\square_{\text{induktio}}$

Määritelmien nojalla $x \in L(M) \Leftrightarrow M$ voi päätyä hyväksyvään tilaan $q_f \in F$ syötteellä $x \Leftrightarrow S_x$ sisältää jonkin $q_f \in F \Leftrightarrow S_x \in \hat{F}$. Kaavan (1) nojalla S_x on automaatin $\widehat{M}$ lopputila syötteellä x, joten $x \in L(M) \Leftrightarrow$ automaatin $\widehat{M}$ lopputila S_x syötteellä x on hyväksyvä $\Leftrightarrow x \in L(\widehat{M})$. $\square$

- Esimerkki determinisoinnista:



- Konstruktio kannattaa aloittaa alkutilasta $\{q_0\}$ ja sen siirtymistä:
  $\hat{\delta}(\{q_0\}, a) = \delta(q_0, a) = \{q_0, q_1\}$ ja
  $\hat{\delta}(\{q_0\}, b) = \delta(q_0, b) = \{q_0\}$.
- Jatketaan tilasta $\{q_0, q_1\}$:
  $\hat{\delta}(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a) = \{q_0, q_1\}$ ja
  $\hat{\delta}(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b) = \{q_0, q_2\}$.
- Jatketaan näin kunnes uusia tiloja ei enää synny. Käytännössä siis niitä $\hat{Q}$:n tiloista jotka eivät ole saavutettavissa alkutilasta $\{q_0\}$ ei tarvitse muodostaa.
- Determinisoinnissa syntyvistä tiloista kannattaa pitää kirjaa taulukossa:

	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$

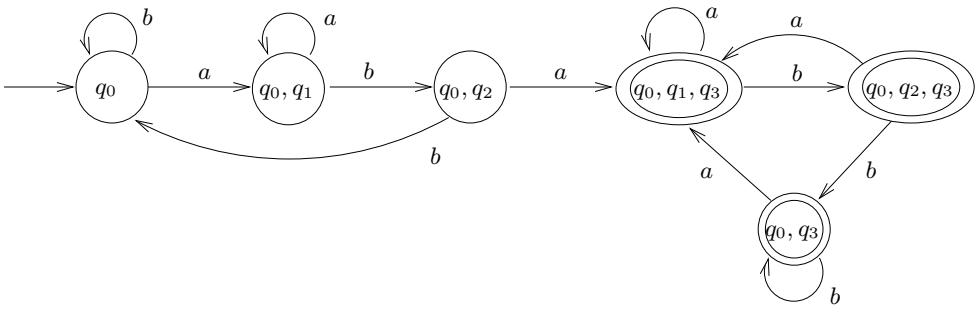
- Aina kun uusi tila löytyy, lisätään sitä varten rivi taulukkoon.

	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

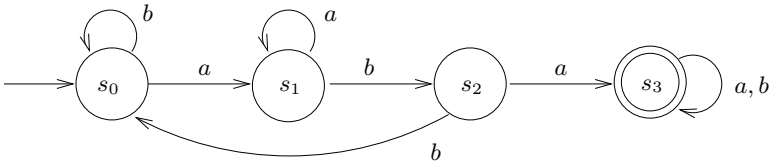
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$

...

- Jatketaan kunnes uusia tiloja ei enää synny.
- Esimerkki loppuun asti determinisoituna:



- Determinisoitu automaatti minimoituna:



2.4. Säännölliset lausekkeet ja kielet

- Ohjelmointikielten muuttujien nimet, vakiot jne. ovat ohjelmointikielen *syntaksin* (“oikeankirjoitusopin”) mukaisia merkkijonoja. Esim. 0.123 on liukulukuvakio, mutta 0.1.2.3 ei ole. Miten voidaan määritellä täsmälleen oikeanmuotoiset merkkijonot?
- Määrittelemiseen voidaan käyttää äärellisten automaattien sijasta myös *säännöllisiä lausekkeita* (engl. regular expression).
- `Unix`issa on käsky `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 säännöllisten lausekkeiden kuvaamia merkkijonoja, joihin tutustumme tässä luvussa. `grep` onkin lyhenne sanoista “Global search for Regular Expression and Print”.
- Säännöllisten lausekkeiden avulla on helppo kuvata yksinkertaisia syntaktisia rakenteita. Siksi niillä on paljon sovelluksia, mm. ohjelmointikielten kääntäjissä, tekstieditoreissa, tekstimuotoista tietoa käsittelevissä sovelluksissa, . . .

- Tarkastellaan ensin muutamia kielten välisiä operaatioita.
- Kertaus: Aakkoston Σ formaali kieli on joukko Σ:n merkkijonoja.
- Määritellään avuksi kolme operaatiota kielten yhdistämiseen: Olkoot A ja B aakkoston Σ kieliä. Tällöin
  - A:n ja B:n *yhdiste* on kieli
    $$A \cup B = \{x \in \Sigma^* \mid x \in A \text{ tai } x \in B\}.$$
    Siis: merkkijono kuuluu yhdistekieleen jos se kuuluu vähintään toiseen kielistä A tai B — kyseessä siis tavallinen joukkojen A ja B yhdiste.
  - A:n ja B:n *tulo* on kieli
    $$AB = \{xy \in \Sigma^* \mid x \in A,\ y \in B\}$$
    Siis: merkkijono kuuluu kielten A ja B tuloon jos se on saatu katenoimalla jokin kielen A jonkin kielen B merkkijonon kanssa.
  - Kielen A *potenssit* A^k, $k \geq 0$, määritellään seuraavasti:
    $$\begin{cases} A^0 &= \{\epsilon\}, \\ A^k &= AA^{k-1} \\ &= \{x_1 \ldots x_k \mid x_i \in A \quad \forall i = 1, \ldots, k\} \qquad (k \geq 1) \end{cases}$$
    Näin ollen $A^0 = \{\epsilon\}$, $A^1 = A$, $A^2 = AA^1 = AA$, $A^3 = AA^2 = AAA$, $A^4 = AA^3 = AAAA$, ...
    Siis: merkkijono kuuluu kieleen A^k jos se on saatu katenoimalla k kielen A merkkijonoa.

– A:n *sulkeuma* on kieli

$$\begin{aligned} A^* &= A^0 \cup A^1 \cup A^2 \cup \ldots \\ &= \bigcup_{k=0}^{\infty} A^k \\ &= \{x_1 \ldots x_k \mid k \geq 0,\ x_i \in A \quad \forall i = 1, \ldots, k\} \end{aligned}$$

Siis: merkkijono kuuluu kieleen A^* jos se on saatu katenoimalla nolla, yksi, tai useampia kielen A merkkijonoja.

Esim. Olkoon $A = \{00, 01\}$ ja $B = \{\epsilon, 1, 010\}$.

– $A \cup B = B \cup A = \{00, 01, \epsilon, 1, 010\}$.

– $AB = \{00, 001, 00010, 01, 011, 01010\}$,
$BA = \{00, 01, 100, 101, 01000, 01001\}$, siis $AB \neq BA$!

– $A^0 = \{\epsilon\}$, $A^1 = A = \{00, 01\}$,
$A^2 = AA = \{0000, 0001, 0100, 0101\}$,
$A^3 = AA^2 = \{000000, 000001, 000100, 000101, 010000, 010001,$
$010100, 010101\}$,
$A^* = \{\epsilon, 00, 01, 0000, 0001, 0100, 0101, 000000, 000001, 000100,$
$000101, 010000, 010001, 010100, 010101, \ldots\}$

– $B^0 = \{\epsilon\}$, $B^1 = B = \{\epsilon, 1, 010\}$,
$B^2 = BB = \{\epsilon, 1, 010, 11, 1010, 0101, 010010\}$,
$B^3 = BBB \ldots$

- Nyt olemme valmiit määrittelemään säännölliset lausekkeet. Tehdään määritelmä kahdessa vaiheessa.
  - Ensin annetaan ns. *induktiivinen* määritelmä säännöllisten lausekkeiden syntaksille.
  - Tämän jälkeen määritellään säännöllisten lausekkeiden semantiikka, ts. minkä kielen kukin säännöllinen lauseke kuvaa.
- *Määritelmä:* Aakkoston Σ *säännölliset lausekkeet* (engl. regular expression) määritellään induktiivisesti säännöillä:

  (i) $\emptyset$ ja ϵ ovat Σ:n säännöllisiä lausekkeita;

  (ii) 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;

  Muita Σ:n säännöllisiä lausekkeita ei ole.
- Esimerkiksi seuraavat ovat aakkoston $\{a, b\}$ säännöllisiä lausekkeita:

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

- Esim. r_1 on syntaktisesti korrekti säännöllinen lauseke, sillä:
  - $a, b \in \Sigma$, joten ne ovat säännöllisiä lausekkeita (sääntö (ii)).
  - Siispä (ab) on säännöllinen lauseke (sääntö (iii)), joten myös $((ab)b)$ on säännöllinen lauseke (sääntö (iii)).

- Seuraavana induktiivinen määritelmä säännöllisen lausekkeen r kuvaamalle kielelle $L(r)$. Idea:
  - Määritellään ensin yksinkertaisten, edellisen sivun kohtien (i) ja (ii) mukaisten säännöllisten lausekkeiden kuvaamat kielet.
  - Tämän jälkeen määritellään monimutkaisempien kohdan (iii) mukaisten säännöllisten lausekkeiden kuvaamat kielet lausekkeiden osasten kuvaamien kielten avulla.

  Täsmällisemmin:

- *Määritelmä:* Aakkoston Σ säännöllinen lauseke *kuvaa* kielen, joka määräytyy seuraavista säännöistä:

  (i) $L(\emptyset) = \emptyset$ ja $L(\epsilon) = \{\epsilon\}$.

  (ii) $L(a) = \{a\}$ kaikilla $a \in \Sigma$.

  (iii) $L((r \cup s)) = L(r) \cup L(s)$, $L((rs)) = L(r)L(s)$, ja $L(r^*) = (L(r))^*$.

- Tarkastellaan esimerkkinä lausekkeiden r_1, r_2, r_3 ja r_4 kuvaamien kielten määrittelemistä.
  - Kohdan (ii) perusteella $L(a) = \{a\}$ ja $L(b) = \{b\}$.
  - Siispä kohdan (iii) toisen säännön perusteella $L(ab) = L(a)L(b) = \{a\}\{b\} = \{ab\}$, ja edelleen $L(r_1) = L((ab)b) = L(ab)L(b) = \{abb\}$.
  - Kohdan (iii) kolmannen säännön perusteella $L(r_2) = L((ab)^*) = \{ab\}^* = \{\epsilon, ab, abab, ababab, \ldots\}$.
  - Kohdan (ii) ja kohdan (iii) kolmannen säännön perusteella $L(b^*) = L(b)^* = \{b\}^* = \{\epsilon, b, bb, bbb, \ldots\} = \{b^i \mid i \geq 0\}$, joten kohdan (iii) toisen säännön perusteella $L(r_3) = L(ab^*) = L(a)L(b*) = \{a\}\{b\}^* = \{ab^i \mid i \geq 0\}$.
  - Koska $L(b) = \{b\}$ ja $L(bb) = \{bb\}$, kohdan (iii) ensimmäisen säännön perusteella $L(b \cup bb) = L(b) \cup L(bb) = \{b\} \cup \{bb\} = \{b, bb\}$.
  - Kohdan (iii) toisen säännön perusteella siis $L(a(b \cup bb)) = \{ab, abb\}$, joten lopulta kohdan (iii) kolmannen säännön perusteella saamme $L(r_4) = L((a(b \cup (bb)))^*) = L((a(b \cup (bb))))^* = \{ab, abb\}^* = \{\epsilon, ab, abb, abab, ababb, abbab, abbabb, \ldots\}$.
- Lyhennysmerkintöjä:
  - Annetaan operaattoreille seuraavat prioriteetit: ensin sulkeuma, sitten katenaatio, lopuksi yhdiste. (Vrt. aritmeettisten operaatioiden prioriteetit: ensin potenssi, sitten kertolasku, lopuksi summa.)

– Havaitaan, että yhdiste- ja tulo-operaatiot ovat assosiatiivisia:

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

– Sovitaan, että säännöllisen lausekkeen sulkuja voi jättää merkitsemättä, jos lausekkeen merkitys on prioriteettisääntöjen ja yhdisteen ja tulon assosiatiivisuuden nojalla ilmankin selvä.

- Edelliset esimerkit voidaan siis kirjoittaa yksinkertaisemmin:

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

- Huom: Usein käytetään lyhennysmerkintää $r^+ = rr^* = r^*r$.

Säännölliset kielet

- Säännöllinen lauseke on merkkijono joka kuvaa jonkin formaalin kielen. Säännöllinen lauseke on merkkijonona aina äärellinen, mutta sen kuvaama kieli saattaa olla ääretön.
- Säännöllinen lauseke on siis äärellinen merkkijonoesitys potentiaalisesti äärettömälle formaalille kielelle.
- ***Määritelmä:*** **Kieli on *säännöllinen*, jos se voidaan kuvata säännöllisellä lausekkeella.**
- Kuten tulemme huomaamaan (ja oikeastaan tiedämmekin jo), ei kaikkia formaaleja kieliä pysty esittämään säännöllisten lausekkeiden avulla.

Esimerkkejä

- Olkoon aakkosto $\Sigma = \{\text{a}, \text{b}, \text{c}, \ldots, \text{ö}\}$. Tarkastellaan säännöllistä lauseketta

  $$r = l^* \text{teoria} l^*,$$

  missä l on lyhennysmerkintä lausekkeelle $(\text{a} \cup \text{b} \cup \ldots \cup \text{ö})$; siis $L(l^*) = \Sigma^*$. Lauseke kuvaa kielen, joka koostuu kaikista merkkijonon teoria osajononaan sisältävistä merkkijonoista.

- Javan etumerkittömistä liukulukuliteraaleista koostuva aakkoston Σ_{ascii} kieli voidaan kuvata säännöllisellä lausekkeella

  $$\begin{aligned}&(d^+.d^* \cup .d^+)(\epsilon \cup ((\text{e} \cup \text{E})(+ \cup - \cup \epsilon)d^+))(\epsilon \cup s)\\ &\cup d^+(\text{e} \cup \text{E})(+ \cup - \cup \epsilon)d^+(\epsilon \cup s)\\ &\cup d^+ s,\end{aligned}$$

  missä $d = (0 \cup \cdots \cup 9)$ ja $s = (\text{d} \cup \text{D} \cup \text{f} \cup \text{F})$.

  Kieleen kuuluvat esim. merkkijonot 12., .12f, 1.2, 7D, 1.2E3, 1.2e3, 1.2E-3f, 1E2, ja 1e23.

- Edellisiin esimerkkeihin säännölliset lausekkeet sopivat melko hyvin. Aina näin ei välttämättä ole.

- Kieli, johon kuuluvat kaikki ne merkkijonot, joissa ei esiinny sanaa “käytäntö”, on kyllä säännöllinen, mutta kielen määrittelevästä säännöllisestä lausekkeesta tulisi hiukan pitkä ja epähavainnollinen.

Säännöllisten lausekkeiden sieventäminen

- Säännöllinen kieli voidaan yleensä kuvata usealla vaihtoehtoisella säännöllisellä lausekkeella. Esimerkki:

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

- *Määritelmä:* Säännölliset lausekkeet r ja s ovat *ekvivalentit*, merkitään $r = s$, jos $L(r) = L(s)$.
- Lisäksi merkitään $r \subseteq s$ tarkoittamaan $L(r) \subseteq L(s)$.
- Lausekkeen sieventäminen tarkoittaa "yksinkertaisimman" ekvivalentin lausekkeen määrittämistä.

Sievennyssääntöjä

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

- Näitä sääntöjä voi soveltuvissa määrin käyttää, mutta usein kuitenkin tärkeintä on “järjen” käyttö ja kokeilu. Myös myöhemmin esitettävä säännöllisten lausekkeiden ja automaattien vastaavuus saattaa joissain tapauksissa helpottaa lausekkeiden sieventämistä.

- Esimerkki: Todistetaan, että pari sivua aiemmin esitetyt säännölliset lausekkeet $(a \cup b)^*$, $(a^*b^*)^*$ ja $a^*b^* \cup (a \cup b)^*ba(a \cup b)^*$ kuvaavat kaikki saman kielen.
- Määritelmistä seuraa, että $L(r) = L(s) \Leftrightarrow L(r) \subseteq L(s) \wedge L(s) \subseteq L(r)$ eli $r = s \Leftrightarrow r \subseteq s \wedge s \subseteq r$
- Kolmen lausekkeen r, s ja t ekvivalenssi voidaan osoittaa näyttämällä erikseen $r \subseteq s$, $s \subseteq t$ ja $t \subseteq r$, sillä näistä seuraa joukkoinkluusion transitiivisuuden nojalla että myös $s \subseteq r$, $t \subseteq s$ ja $r \subseteq t$.
- Esimerkin tapauksessa:
  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

  Siispä $(a \cup b)^* = (a^*b^*)^* = a^*b^* \cup (a \cup b)^*ba(a \cup b)^*)$.
- Voidaan todistaa (todistus sivuutetaan): Jos L ja M ovat säännöllisiä kieliä, myös
  1. $L \cap M$
  2. $\overline{L} = \Sigma^* \setminus L$
  3. $L^R = \{w^R | w \in L\}$

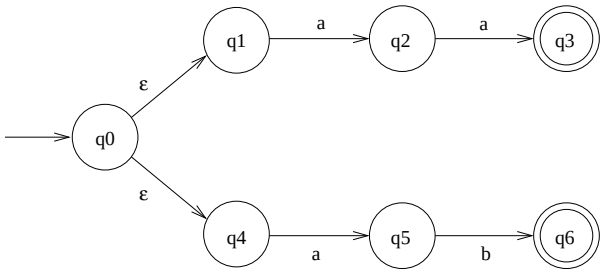
  ovat säännöllisiä.

2.5. Kaksi äärellisen automaatin laajennosta

Ennen kuin pystymme todistamaan että äärelliset automaatit ja säännölliset lausekkeet ovat yhtä ilmaisuvoimaisia, määrittelemme äärellisille automaateille kaksi laajennosta.

ϵ-automaatti

- Epädeterministinen äärellinen automaatti, jossa sallitaan *ϵ-siirtymät* — siirtymät, joissa automaatti ei lue yhtään syötemerkkiä, vaan siirtyy "itsestään" tilasta toiseen.



Kuva 1: Kielen $\{aa, ab\}$ tunnistava ϵ-automaatti.

- Kuvan automaatti siirtyy ensin joko tilaan $q1$ tai tilaan $q4$, ja vasta tämän jälkeen se alkaa lukea syötenauhansa sisältöä.
- *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.
- Esim. edellisen sivun automaatissa $(q0, aa) \vdash (q1, aa)$ koska $q0$:sta on ϵ-siirtymä tilaan $q1$, $(q1, aa) \vdash (q2, a)$, ja $(q2, a) \vdash (q3, \epsilon)$. Siispä $(q0, aa) \vdash^* (q3, \epsilon)$.
- *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

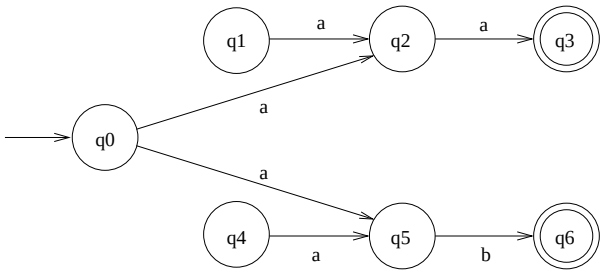
$$\hat{\delta}(q, a) = \{q' \in Q \mid (q, a) \underset{M}{\vdash}^* (q', \epsilon)\};$$

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

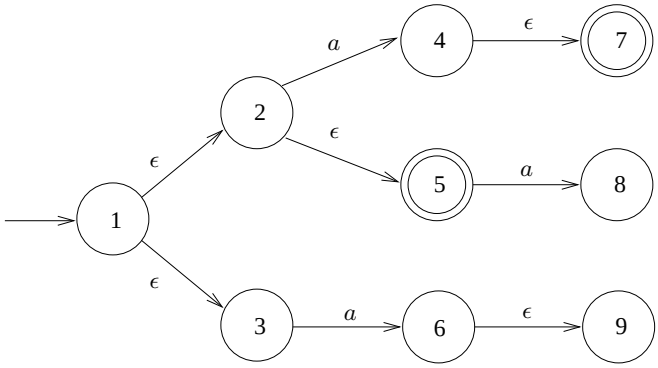
□

ϵ-siirtymien poisto käytännössä

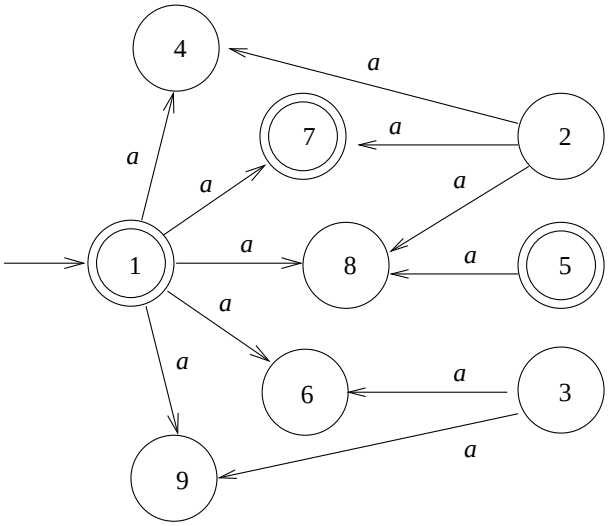
- Edellisen lemman todistuksen konstruktiossa:
  - Muodostetaan siirtymä $q \xrightarrow{a} q'$, jos $(q, a) \underset{M}{\vdash}^* (q', \epsilon)$.
  - Lisätään alkutila q_0 lopputiloihin, jos $(q_0, \epsilon) \underset{M}{\vdash}^* (q_f, \epsilon)$ jollakin tilalle $q_f \in F$.
- Eli jos haluamme muodostaa ϵ-automaattia M vastaavan epädeterministisen automaatin $\widehat{M}$, toimitaan seuraavasti:
  - Kopioidaan kaikki M:n tilat sekä normaalit siirtymät automaattiin $\widehat{M}$.
  - Jos M:ssä päästään alkutilasta johonkin lopputilaan pelkillä ϵ-siirtymillä, tehdään $\widehat{M}$:n alkutilasta q_0 lopputila.
  - Tämän jälkeen käydään läpi kaikki tilaparit $q, q' \in Q$ ja merkit $a \in \Sigma$. Jos automaatissa M on olemassa polku tilasta q tilaan q' siten että matkalla tehdään vain ϵ-siirtymiä sekä tasan kerran siirtymä merkillä a, lisätään automaattiin $\widehat{M}$ siirtymä tilasta q tilaan q' merkillä a; muuten siirtymää $q \xrightarrow{a} q'$ ei lisätä.
- Kuvan 1 ϵ-automaattia vastaava epädeterministinen automaatti:



Toinen esimerkki:



Kuva 2: ϵ-automaatti M.



Kuva 3: Epädeterministinen automaatti $\widehat{M}$.

Lausekeautomaatit

- *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

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

  (Kuvauksen δ äärellisyys tarkoittaa tässä, että $\delta(q, r) \neq \emptyset$ vain äärellisen monella parilla $(q, r) \in Q \times \text{RE}_\Sigma$).

- Yhden askelen tilannejohto määritellään:

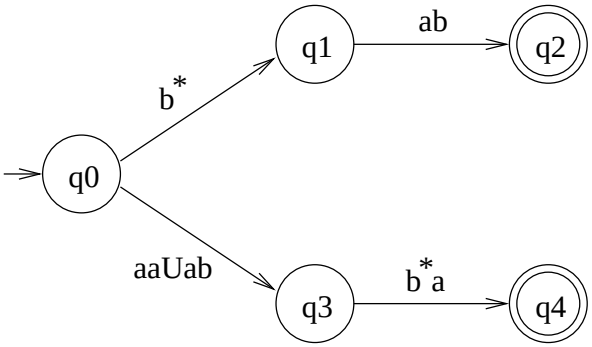
  $$(q, w) \underset{M}{\vdash} (q', w')$$

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

- Intuitio: lausekeautomaatti lukee jokaisessa laskenta-askeleessa *merkkijonon*, kun aiemmin määritellyt automaatit lukevat korkeintaan yhden merkin.

- Siirtymissä luettavat merkit määritellään säännöllisten lausekkeiden avulla.

- Tarkastellaan seuraavaa lausekeautomaattia:



- Siirtymässä $q0$:sta $q1$:n voidaan lukea minkä tahansa pituinen pelkistä b-merkeistä koostuva merkkijono (siis myös 0:n mittainen).

  Esim: $(q0, bbbab) \vdash (q1, ab)$ toisaalta myös $(q0, bbbab) \vdash (q1, bab)$, $(q0, bbbab) \vdash (q1, bbab)$ ja $(q0, bbbab) \vdash (q1, bbbab)$.

- Siirtymässä $q1$:stä $q2$:een voidaan lukea ainoastaan merkkijono ab, eli $(q1, ab) \vdash (q2, \epsilon)$.

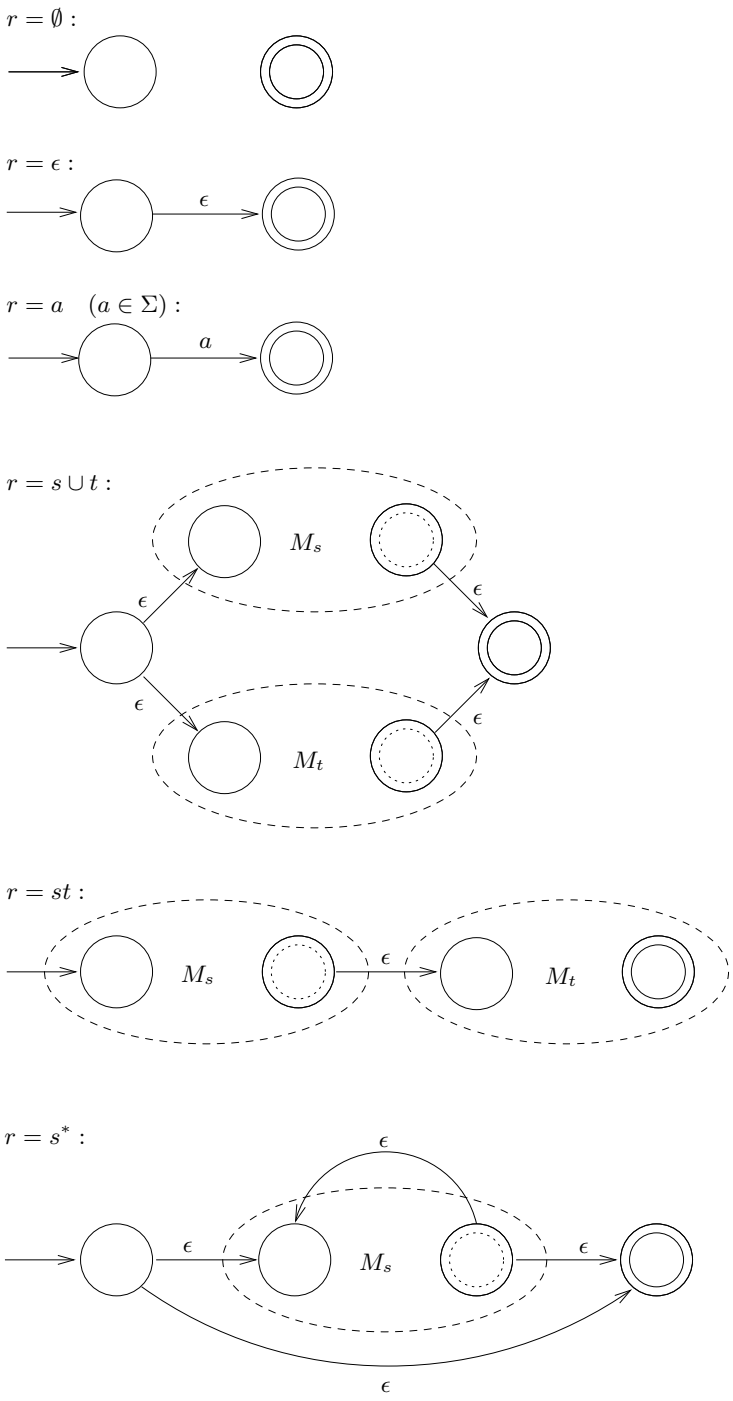
- Koska $(q0, bbbab) \vdash (q1, ab)$ ja $(q1, ab) \vdash (q2, \epsilon)$, pätee $(q0, bbbab) \vdash^* (q2, \epsilon)$. Koska $q2$ on hyväksyvä tila, hyväksyy laseautomaatti merkkijonon $bbbab$.

- Mieti miksi automaatti hyväksyy merkkijonot aaa ja $abba$!

2.6. 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
     - Poistetaan automaatista ϵ-siirtymät, tuloksena ekvivalentti eli saman kielen tunnistava 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 eli $L(M) = L(r)$ jollakin säännöllisellä lausekkeella r:
     - Tarkastellaan lausekeautomaatteja — jos väite pätee lausekeautomaateille, se pätee myös tavallisille äärellisille automaateille (jotka ovat lausekeautomaattien erikoistapaus).
     - Redusoidaan lausekeautomaatti M korkeintaan 2-tilaiseksi automaatiksi, josta voidaan lukea suoraan säännöllinen lauseke r, jolle $L(r) = L(M)$.

- Seuraava lause osoittaa, että jokainen säännöllinen lauseke voidaan muuntaa mekaanisesti äärelliseksi automaatiksi, joka tunnistaa saman kielen.
- *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).
  - M_r:stä voidaan poistaa ϵ-siirtymät edellisen lemman mukaisesti. Syntynyt epädeterministinen automaatti voidaan determinisoida aiemmin esitetyn konstruktion avulla. Tuloksena (deterministinen) äärellinen automaatti, joka tunnistaa kielen $L(r)$. $\Box$

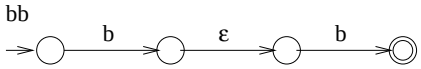


Kuva 4: Lauseketta r vastaavan ϵ-automaatin M_r muodostaminen.

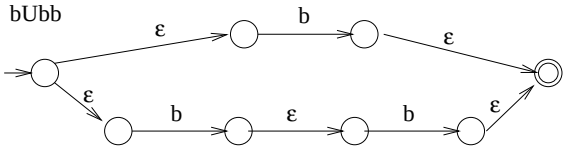
- Esimerkkinä säännöllisen lausekkeen $a(b \cup bb)^*$ muuntaminen ϵ-automaatiksi.

vaihe 1:

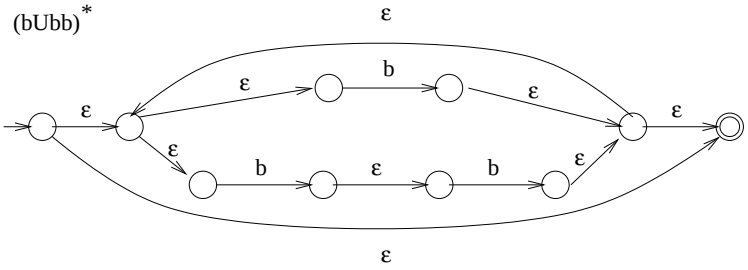
vaihe 2:



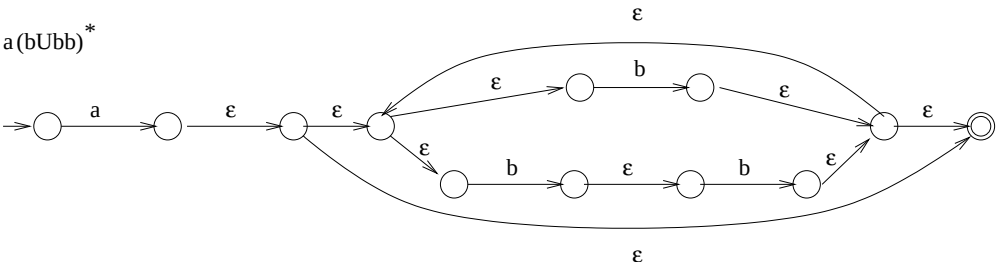
vaihe 3:



vaihe 4:



vaihe 5:



- Saatu ϵ-automaatti voidaan muuntaa opituin keinoin epädeterministiseksi automaatiksi ja edelleen deterministiseksi automaatiksi.

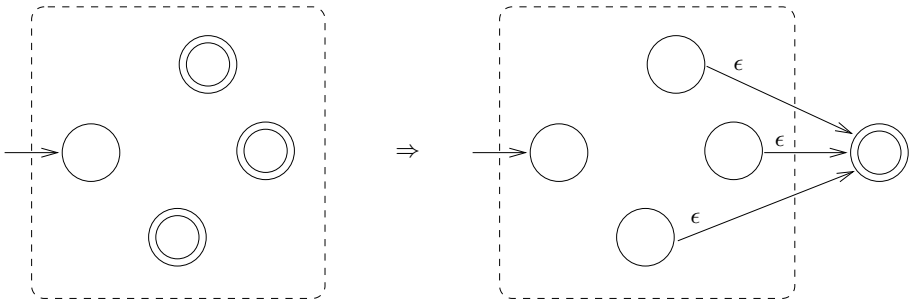
- Todistetaan sitten implikaatio toiseen suuntaan:
- *Lause:* Jokainen äärellisellä automaatilla tunnistettava kieli on säännöllinen.

  Tod.: Osoitetaan, että jokainen lausekeautomaatilla tunnistettava kieli on säännöllinen. Väite seuraa tästä, koska tavalliset äärelliset automaatit ovat erikoistapaus lausekeautomaateista.

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

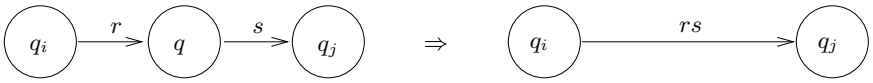
  – Yhdistetään M:n lopputilat yhdeksi lopputilaksi q_f ϵ-siirtymillä (ks. kuva 5).
  – while (muita kuin alku- ja lopputiloja) {
    * Valitse q, $q \neq q_0$, $q \neq q_f$.
    * Poista q reitiltä $q_i \rightarrow q \rightarrow q_j$ (missä q_i = q:n edeltäjätila ja q_j = q:n seuraajatila) reduktiosäännöillä. Muista käydä läpi kaikki tilan q sisältävät polut!
    * Yhdistä rinnakkaiset siirtymät.

  } (ks. kuvat 6 ja 7)



Kuva 5: Lausekeautomaatin lopputilojen yhdistäminen.

(i):



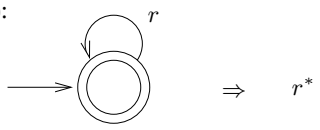
(ii):

Kuva 6: Tilan poistaminen lausekeautomaatista.

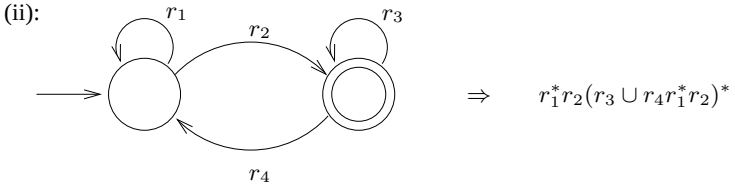
Kuva 7: Rinnakkaisten siirtymien yhdistäminen lausekeautomaatissa.

– Tiivistyksen päättyessä jäljellä olevaa enintään 2-tilaista automaattia vastaava säännöllinen lauseke muodostetaan kuten kuvassa 8. □

(i):



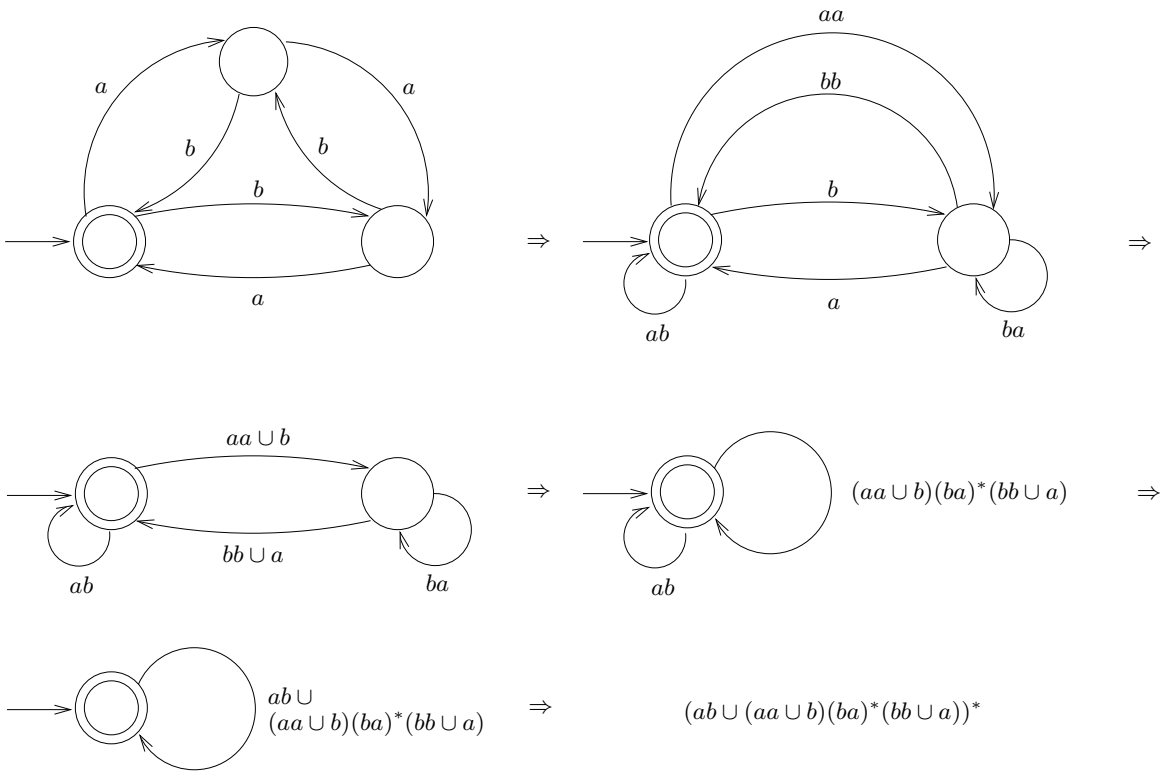
(ii):



Kuva 8: Säännöllisen lausekkeen muodostaminen redusoidusta lausekeautomaatista.

Esimerkki

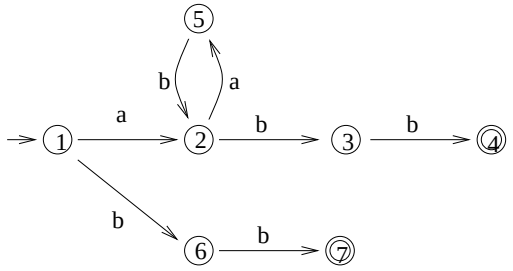
- Säännöllisen lausekkeen muodostaminen äärellisestä automaatista:



2.7. Säännöllisten kielten rajoituksista

- Formaaleja kieliä (päätösongelmia) on ylinumeroituva määrä, mutta säännöllisiä lausekkeita vain numeroituva määrä $\Rightarrow$ kaikki kielet eivät voi olla säännöllisiä. Miten osoittaa, että kieli ei ole säännöllinen?
- Olemme todistaneet, että kaikki säännölliset kielet voidaan tunnistaa äärellisillä automaateilla.
- Jokaisen äärellisen automaatin "muisti" on äärellinen: Ainoa asia, jonka automaatti muistaa syötteen jo käsitellystä osasta, on automaatin tila, joita on vain äärellinen määrä.
- Tästä seuraa perusrajoitus: Kieli on säännöllinen vain jos se voidaan tunnistaa vakiomäärällä muistia.
- Esim. tasapainoisten sulkujonojen muodostama kieli $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ ei ole säännöllinen, joten sitä ei voi tunnistaa äärellisellä automaatilla. Intuitiivisesti syynä on se, että ollessaan syötteen keskikohdassa tunnistavan automaatin pitäisi muistaa täsmälleen lukemiensa "$($"-merkkien lukumäärä, jotta se voisi tarkastaa "$)$"-merkkejä olevan yhtä monta. Tähän ei vakiomäärä muistia riitä, sillä "$($"-merkkejä voi olla mielivaltaisen monta.
- Seuraus: hyvinmuodostettuja aritmeettisia lausekkeita ei voida tunnistaa äärellisellä automaatilla.
- "Pumppauslemma" formalisoi tämän rajallisen muistin idean.

- Johdatuksena pumppauslemman ideaan tarkastellaan oheista automaattia joka hyväksyy saman kielen kuin säännöllinen lauseke $bb \cup a(ab)^*bb$.



- Automaatti hyväksyy merkkijonon $aabbb$, sillä
  $(1, aabbb) \vdash (2, abbb) \vdash (5, bbb) \vdash (2, bb) \vdash (3, b) \vdash (4, \epsilon)$.
- Huomionarvoista tässä on se että automaatti tekee “luupin” $2 \xrightarrow{a} 5 \xrightarrow{b} 2$ lukiessaan syötteen toisen ja kolmannen merkin.
- Automaatti luuppaa samaan tapaan myöskin käsitellessään muitakin pitempiä syötteitään jotka se tulee hyväksymään. Hyväksytyistä syötteistä ainoastaan abb ja bb eivät aiheuta luuppausta.
- Oikeastaan onkin niin, että jos automaatti hyväksyy merkkijonon $aabbb$, hyväksyy se minkä tahansa merkkijonon, jossa tilojen 2 ja 5 välillä on luupattu mielivaltaisen monta kertaa:
- Eli esim syöttellä $aababbb$ luupataan kahteen kertaan:
  $(1, aababbb) \vdash (2, ababbb) \vdash (5, babbb) \vdash (2, abbb) \vdash (5, bbb) \vdash (2, bb) \vdash (3, b) \vdash (4, \epsilon)$.
- Vastaavasti syötteellä $aababababbb$ luupataan kolmesti, ja ylei-

sessä tapauksessa syötteellä $a(ab)^i bb$ luupataan i-kertaa.

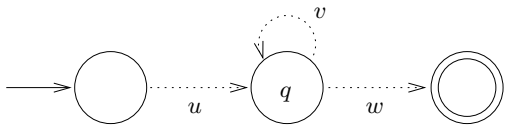
- Ts. merkkijonoa $aabbb$ voidaan “pumpata” keskeltä.
- Sama pätee mille tahansa säännölliselle kielelle L: Jokaista kielen “tarpeeksi pitkää” merkkijonoa voidaan pumpata jostain kohtaa merkkijonon keskeltä.
- Perusteena on sama kuin minkä äsken esitimme. Jos tarkastellaan kielen L hyväksyvää äärellistä automaattia, ovat “kaikki tarpeeksi pitkät” hyväksytyt merkkijonot sellaisia että niitä käsitellessään automaatti luuppaa, ja tätä luuppia voidaan toistaa halutessa miten monta kertaa tahansa

Pumppauslemma

- *Lemma*: 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 $i = 0, 1, 2, \ldots$
- Siis: Pumppauslemman nojalla jokaiselle säännölliselle kielelle löytyy luku n siten että kaikkia vähintään n:n mittaisia kielen merkkijonoja voidaan pumpata keskeltä.
- Merkkijonojen pumppauskohta on aina ensimmäisen n:n merkin sisällä.
- Huom: Lemma sanoo, että jokaista tarpeeksi pitkää merkkijonoa voidaan pumpata jostain kohtaa, mutta pumppauskohtaa ei saa valita mielivaltaisesti.

Pumppauslemman todistus:

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



Kuva 9: Merkkijonon $x = uvw \in A$ pumppaus.

- Pumppauslemma siis lupaa että kaikkien säännöllisten kielten riittävän pitkät merkkijonot pumppautuvat.
- Jos jokin kieli L ei toteuta pumppauslemmaa — millään n ei päde että n:ää pidemmät L:n merkkijonot pumppautuvat — tiedetään, että L ei ole säännöllinen kieli.
- Täten voimme todistaa pumppauslemmaa käyttäen että jokin kieli ei ole säännöllinen.
- Todistustekniikka sille että kieli L ei ole säännöllinen, on seuraava:
  - Tehdään vastaoletus: Kieli L on säännöllinen.
  - Pumppauslemman nojalla on olemassa luku n jota pitemmät merkkijonot pumppautuvat.

    Huom. Tiedetään ainoastaan että jokin tälläinen n on olemassa, (olettaen että L on säännöllinen), mutta luvun n suuruutta ei tiedetä.
  - Näytetään, että jokin merkkijono $x \in L$, jonka pituus on vähintään n, ei pumppaannu *millään* lemman mukaisella jakotavalla $x = uvw$.
  - Näin vastaoletus että L on säännöllinen kieli aiheuttaa ristiriidan pumppauslemman kanssa. Siispä kieli L ei ole säännöllinen.

Esimerkki 1

- Väite: Sulkulausekekieli

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

ei ole säännöllinen.

Tod.: Vastaoletus: L on säännöllinen. On siis olemassa luku $n \geq 1$, jonka pituisia (ja pitempiä) L:n merkkijonoja voidaan pumpata.

Valitaan pumpattavaksi merkkijonoksi $x = (^n)^n$, jolloin $|x| = 2n > n$. Vaikka emme tiedä konkreettista arvoa luvulle n, voimme käyttää sitä muodostaessamme merkkijonoa, sillä tiedämme että luku n on olemassa.

Lemman mukaan x voidaan jakaa pumpattavaksi osiin $x = uvw$, siten että $|uv| \leq n$, $|v| \geq 1$; siis on oltava

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

Siis pumpattava osa sisältää pelkästään (-merkkejä, ja on varma että pumpattavan osan pituus on vähintään yksi. Huom: tässä i:n ja j:n tarkkaa arvoa ei tiedetä. Pumppauslemma takaa että $i + j \leq n$ ja $j \geq 1$, mistä seuraa, että pumpattavan osan pituus on vähintään 1 ja enintään n.

Nyt esimerkiksi “0-kertaisesti” pumpattu merkkijono $uv^0w = (^i(^{n-(i+j)})^n = (^{n-j})^n$ ei kuulu kieleen L sillä (-merkkejä on vähintään 1 vähemmän kuin)-merkkejä.

Ristiriita. Siten L ei voi olla säännöllinen. □

- Huom: Pumppauslemma ei sano, että voimme valita u:n ja v:n haluamallamme tavalla!
- Edellä valitsimme pumpattavan jonon pituudeksi $2n$, jolloin ensimmäiset n "$($"-merkkiä muodostavat uv:n, vaikkei tiedetäkään tarkkaan miten.

Esimerkki 2

- Väite: $L = \{a^k b^l \mid l \geq k \geq 0\}$ ei ole säännöllinen.
  Tod.: Vastaoletus: L on säännöllinen. Pumppauslemma $\Rightarrow$ $\exists n \geq 1$, jonka pituisia merkkijonoja voidaan pumpata. Valitaan $x = a^n b^n$. Nyt $|x| = 2n > n$.

  Lemman mukaan x voidaan jakaa pumpattavaksi osiin $x = uvw$, siten että $|uv| \leq n$, $|v| \geq 1$; siis on oltava

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

  Pumpattava osa koostuu siis vähintään yhdestä a merkistä.

  Nyt $uv^0w = a^i a^{n-(i+j)} b^n = a^{n-j} b^n$ kuuluu kieleen L, joten 0-kertainen pumppaus onnistuu. Mutta 2-kertainen pumppaus $uv^2w = a^i a^{2j} a^{n-(i+j)} b^n = a^{n+j} b^n$ ei kuulu kieleen L. Ristiriita, eli L ei voi olla säännöllinen. $\square$

  Kuten yllä nähtiin, ristiriitaan säännöllisyyden kanssa riittää se, että joku määrä osan v toistoja johtaa ulos kielestä — pumppauslemman nojalla v kun pitäisi saada toistaa mielivaltaisen monta kertaa.

Esimerkki 3

- Väite: $L = \{c^l a^k b^k \mid k \geq 0 \wedge l \geq 1\}$ ei ole säännöllinen.
  Tod.: Vastaoletus: L on säännöllinen. Pumppauslemma $\Rightarrow$ $\exists n \geq 1$, jonka pituisia merkkijonoja voidaan pumpata.

  Valitaan $x = ca^n b^n$. Nyt $|x| = 2n + 1 > n$.

  Lemman mukaan x voidaan jakaa pumpattavaksi osiin $x = uvw$, siten että $|uv| \leq n$, $|v| \geq 1$. Huomattava ero edelliseen kahteen esimerkkiin on se ettei uv koostukaan pelkästään merkeistä a, vaan on tyyppiä $ca \ldots a$.

  Jakoja uvw on nyt olennaisesti kahta eri tyyppiä. Ensimmäisessä c sisältyy osaan u ja toisessa pumpattavaan osaan v (jolloin u:ssa ei ole yhtään merkkiä):

  1. $u = ca^i,\ v = a^j,\ w = a^{n-(i+j)}b^n,$ missä $i + j \leq n - 1,\ j \geq 1.$
  2. $u = \epsilon,\ v = ca^j,\ w = a^{n-j}b^n,$ missä $n \geq j \geq 0.$

  Kumpaakin tapausta on tarkasteltava erikseen, sillä haluammehan näyttää ettei x:n *mikään* jako uvw pumppaannu.

  1. $uv^0w = ca^i a^{n-(i+j)}b^n = ca^{n-j}b^n$ ei kuuluu kieleen L sillä merkkejä a on vähemmän kuin merkkejä b. Siis mikään jako jossa c sisältyy osaan u ei onnistu, sillä 0-kertainen pumppaus ei kuulu kieleen.
  2. $uv^0w = a^{n-j}b^n$ ei kuuluu kieleen L sillä alussa ei ole merkkiä c. Siis myöskään jako, jossa c sisältyy pumpattavaan

osaan, ei onnistu, sillä 0-kertainen pumppaus ei kuulu kieleen.

Ristiriita, eli L ei voi olla säännöllinen. □

- Edellinen esimerkki tuo selvemmin esiin sen että erilaisia jakomahdollisuuksia uvw on suuri määrä, ja ainoa asia jonka jaosta voi olettaa on se että $|uv| \leq n$ ja $|v| \geq 1$, ts. pumppattava osa on merkkijonon n:n ensimmäisen merkin sisällä ja pumpattavan osan pituus on vähintään 1 merkki.
- Esimerkiksi esimerkissä 2 merkitsimme merkkijonon $x = a^n b^b$ jakoja seuraavalla tavalla:

  $$u = a^i,\ v = a^j,\ w = a^{n-(i+j)}b^n.$$

  Tässä parametrit i, j toteuttavat ehdot $i + j \leq n,\ j \geq 1$, mutta niiden arvoja ei ole muuten kiinnitetty.
- Koska määrittelemme merkkijonot u, v, ja w avoimien parametrien i ja j avulla, tarkastelemme tavallaan montaa (jokaista lemman rajoitteiden sallimaa) jakoa rinnakkain. Siis

  $$u = a^i,\ v = a^j,\ w = a^{n-(i+j)}b^n$$

  tarkoittaa kaikkia seuraavia jakoja:

  $u = \epsilon$, $v = a$, $w = a^{n-1}b^n$,
  $u = a$, $v = a$, $w = a^{n-2}b^n$
  $u = \epsilon$, $v = aa$, $w = a^{n-2}b^n$
  $u = a$, $v = aa$, $w = a^{n-3}b^n$
  $u = aa$, $v = aa$, $w = a^{n-4}b^n$
  ...

Yhteenveto säännöllisistä kielistä

- Määritelmän mukaan kieli A on säännöllinen jos ja vain jos on olemassa säännöllinen lauseke r joka kuvaa kielen eli jolle $L(r) = A$.
- Toisaalta olemme näyttäneet että säännölliset lausekkeet ja äärelliset automaatit ovat täsmälleen yhtä ilmaisuvoimaisia kuvaustapoja: Jos kieli on kuvattavissa säännöllisellä lausekkeella, on se kuvattavissa (eli tunnistettavissa) myös äärellisellä automaatilla ja kääntäen.
- Seurauksena kieli A on säännöllinen jos ja vain jos on olemassa äärellinen automaatti M joka kuvaa kielen eli jolle $L(M) = A$.
- Konstruktiot, joilla säännöllisen lausekkeen voi muuntaa äärelliseksi automaatiksi ja äärellisen automaatin säännölliseksi lausekkeeksi on syytä osata.
- Jos pitää todistaa että jokin kieli on säännöllinen, riittää näyttää että kielen voi kuvata säännöllisellä lausekkeella tai äärellisellä automaatilla (deterministisellä, epädeterministisellä, ϵ-automaatilla, tai lausekeautomaatilla).
- Pumppauslemman avulla voimme osoittaa, että jokin kieli *ei* ole säännöllinen.
- Säännölliset kielet ovat vain pieni osajoukko kaikista formaaleista kielistä, mutta silti usein hyödyllisiä käytännössä.

2.8. Säännöllisten kielten sovelluksia

- Tässä luvussa esitellyt mekanismit tarjoavat työkaluja ohjelmointikielten syntaksin tarkastukseen. Säännöllisillä lausekkeilla voidaan määritellä esim. liukulukuliteraalien syntaksi, jolloin syntaksin tarkastus sujuu äärellisillä automaateilla.
- Luvun 2 mekanismeja käyttäen voidaankin toteuttaa ohjelmointikielen *selaaja* — se kääntäjän osa, joka erottelee koodista erilaiset ohjelmointikielen kannalta mielekkäät kokonaisuudet, kuten varatut sanat, muuttujanimet, luvut, ... Esim. selaajangeneroija flex perustuu näihin menetelmiin.
- Säännölliset lausekkeet ja kielet ovatkin käytössä monilla tietojenkäsittelyn osa-alueilla: niitä voi käyttää tekstihauissa (grep), kääntäjän selaajien toteuttamisessa (flex), joidenkin ohjelmointikielten (Perl, Java) merkkijono-operaatioissa, ...
- Äärelliset automaatit ja niiden laajennokset ovat käytössä myös ohjelmien ja tietoliikenneprotokollien määrittelyssä ja ohjelmien oikeellisuuden verifioinnissa.
- Tietojenkäsittelytieteen ulkopuolella säännölliset kielet ovat keskeisessä asemassa esimerkiksi kielitieteissä. Väitetään jopa, että luonnolliset kielet saattavat olla mallinnettavissa säännöllisinä kielinä (Fred Karlsson).
- Sovelluksia löytyy siis tietojenkäsittelyn teoriasta ja käytännöstä sekä myös monilta muilta aloilta.

3. Kontekstittomat kielet ja pinoautomaatit

- Edellä olemme todistaneet, että esim. tasapainoisten sulkulausekkeiden muodostama kieli

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

  ei ole säännöllinen.

- Kielen voi kuitenkin kuvata *kontekstittomalla kieliopilla* (engl. context-free grammars).

- Kuten edellisestä luvusta tiedämme, säännöllisten lausekkeiden kuvaamat kielet voidaan tunnistaa äärellisillä automaateilla. Vastaavasti kontekstittomien kielioppien kuvaamat kielet voidaan tunnistaa *pinoautomaateilla* (engl. push-down automaton).

- Tässä luvussa tarkastellaan kontekstittomia kielioppeja ja -kieliä, kontekstittomiin kieliin liittyviä kielentunnistusongelmia ja pinoautomaatteja.

3.1. Kontekstittomat kieliopit ja kielet

- Esim. sulkulausekekielellä L_{match} on yksinkertainen rekursiivinen kuvaus: 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. merkkijono $((()))$ voidaan tuottaa muunnossääntöjen avulla seuraavasti:

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

- Kontekstiton kielioppi on edellisen kaltainen muunnossysteemi eli lista muunnossääntöjä. Kuvattavat merkkijonot tuotetaan korvaamalla erityisiä muuttuja- tai *välikesymboleita* annettujen sääntöjen mukaan yksi kerrallaan, välikesymbolin kontekstista (eli sitä ympäröivän merkkijonon rakenteesta) riippumatta. Tästä kontekstittomien kielioppien nimi.

- Esim. Yksinkertainen kielioppi kielelle L_{expr}, joka koostuu kertoja yhteenlaskuja sisältävistä aritmeettisista lausekkeista:

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

  Tässä merkintä

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

  on lyhennysmerkintä merkinnälle

$$A \rightarrow \omega_1,\ A \rightarrow \omega_2,\ \ldots A \rightarrow \omega_k.$$

- Esim. lausekkeen $(a + a) * a$ tuottaminen em. kieliopin sääntöjen mukaan:

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

Kontekstittomien kielioppien formaali määritelmä

- 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* (engl. terminal symbol) joukko; sen komplementti $N = V \setminus \Sigma$ on kieliopin *välikemerkkien* tai -*symbolien* (engl. non-terminal symbol) joukko;
  - $P \subseteq N \times V^*$ on kieliopin *sääntöjen* tai *produktioiden* joukko;
  - $S \in N$ on kieliopin *lähtösymboli*.
- Produktiota $(A, \omega) \in P$ merkitään $A \to \omega$.

Esimerkkejä

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

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

- Edellä esitellyn aritmeettisten lausekkeiden kielen L_{expr} tuottaa kielioppi

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

missä

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

- Kielen L_{expr} tuottaa myös kielioppi

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

missä

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

- On siis mahdollista, että useat eri kontekstittomat kieliopit tuottavat saman kielen. Tosin vielä emme ole edes määritelleet, miten kontekstittomat kieliopit tuottavat kieliä — tehdään tämä seuraavaksi.

Kontekstittoman kieliopin kuvaama kieli

- 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 $A \to \omega$

- Esim. $(E)*F \Rightarrow (E+T)*F$ aritmeettisten lausekkeiden kieliopissa G_{expr}, sillä oikeanpuoleinen merkkijono on saatu soveltamalla vasemmanpuoleiseen kieliopin produktiota $E \to E+T$.

- 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 $\gamma_0, \gamma_1, \ldots, \gamma_n \in V^*$ ($n \geq 0$), s.e.

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

Esim. $(E) * F \Rightarrow^* (a + a) * a$ kieliopissa G_{expr}, sillä:
$(E) * F \Rightarrow (E + T) * F \Rightarrow (T + T) * F \Rightarrow (F + T) * F \Rightarrow$
$(a + T) * F \Rightarrow (a + F) * F \Rightarrow (a + a) * F \Rightarrow (a + a) * a$

- Erikoistapaus $n = 0$: $\gamma \underset{G}{\Rightarrow}^* \gamma$ millä tahansa $\gamma \in V^*$.

- Merkkijono $\gamma \in V^*$ on kieliopin G *lausejohdos*, jos pätee $S \underset{G}{\Rightarrow}^* \gamma$.
- Esim. Koska $E \Rightarrow^* (E) * F$, on $(E) * a$ kieliopin G_{expr} lausejohdos. Lausejohdos on siis välikkeistä ja päätemerkeistä koostuva merkkijono, jonka kielioppi tuottaa.
- G:n *lause* on pelkästään G:n päätemerkeistä koostuva G:n lausejohdos.

  Esim. $(a + a) * a$ on lause kieliopissa G_{expr}, koska $(a + a) * a \in \{a, +, *, (,)\}$, ja $E \Rightarrow^* (a + a) * a$.
- Kieliopin G *tuottama* tai *kuvaama kieli* on G:n lauseiden joukko:

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

- Siis: merkkijono $x \in \Sigma^*$ kuuluu kontekstittoman kieliopin G kuvaamaan kieleen $L(G)$ jos ja vain jos kieliopin G lähtösymboli S tuottaa merkkijonon x, ts. on olemassa jono merkkijonoja $\gamma_1, \ldots, \gamma_{n-1} \in V^*$ ($n \geq 0$), s.e.

$$S \underset{G}{\Rightarrow} \gamma_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} x.$$

- ***Määritelmä:***

  Formaali kieli $L \subseteq \Sigma^*$ on *kontekstiton*, jos ja vain jo se voidaan tuottaa jollakin kontekstittomalla kieliopilla.

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.

Esimerkki: olpe03-kielen syntaksin määrittely kontekstittoman kieliopin avulla

- $$\begin{aligned} S &\rightarrow S; S \mid \\ &\quad var := E \mid \\ &\quad \textbf{begin}\ S\ \textbf{end} \mid \\ &\quad \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ S \mid \\ &\quad \textbf{while}\ B\ \textbf{do}\ S \mid \\ &\quad \textbf{write}(E) \mid \\ E &\rightarrow int \mid var \mid (E) \mid E + E \mid E - E \mid E * E \mid \textbf{read} \\ B &\rightarrow E = E \mid E < E \mid \textbf{not}\ B \mid B\ \textbf{and}\ B \end{aligned}$$

- Päätesymbolit:

  $\textbf{if}, \textbf{then}, \textbf{else}, \textbf{while}, \textbf{do}, \textbf{begin}, \textbf{end}, \textbf{not}, \textbf{and}, \textbf{read}, \textbf{write}, ;, :=, +, -, =, <, (,), int, var$

  Tausta-ajatuksena on, että lähdekooditiedosto eli aakkoston Σ_{ascii} merkkijono muunnetaan ensin jonoksi päätesymboleja. Muunnoksen tekevää ohjelmaa kutsutaan *selaajaksi* (engl. scanner). Tyypillisesti selaaja generoidaan automaattisesti (esim. `flex`-ohjelman avulla) listasta säännöllisiä lausekkeita ja niihin liittyviä koodinpätkiä.

- Listassa int edustaa syntaktisesti korrektia kokonaislukua ja var syntaktisesti korrektia muuttujanimeä. Avainsanojen ja operaattorien vastineet aakkostossa $\Sigma_{\texttt{ascii}}$ ovat vastaavat merkkijonot.
- Oheinen kielioppi määrittelee **olpe03**-kielen syntaksin täsmällisesti: Aakkoston Σ_{acii} merkkijono on syntaktisesti korrekti **olpe03**-kielinen ohjelma joss se voidaan tuottaa ylläolevalla kieliopilla.
- Esim. merkkijono
  $i := 0;\ \text{while}\, i < 10\, \text{do}\ \ i := i + 1$
  on kieliopin mukainen, sillä ensin selaaja muuttaa sen muotoon
  $var := int; \mathbf{while}\ var < int\ \mathbf{do}\ var := var + int$
  ja
  $\underline{S} \Rightarrow \underline{S}; S \Rightarrow var := \underline{E}; S \Rightarrow var := int; \underline{S} \Rightarrow$
  $var := int; \underline{S} \Rightarrow$
  $var := int; \mathbf{while}\ B\ \mathbf{do}\ \underline{S} \Rightarrow$
  $var := int; \mathbf{while}\ B\ \mathbf{do}\ var := \underline{E} \Rightarrow$
  $var := int; \mathbf{while}\ B\ \mathbf{do}\ var := \underline{E} + E \Rightarrow$
  $var := int; \mathbf{while}\ B\ \mathbf{do}\ var := var + \underline{E} \Rightarrow$
  $var := int; \mathbf{while}\ \underline{B}\ \mathbf{do}\ var := var + int \Rightarrow$
  $var := int; \mathbf{while}\ \underline{E} < E\ \mathbf{do}\ var := var + int \Rightarrow$
  $var := int; \mathbf{while}\ var < \underline{E}\ \mathbf{do}\ var := var + int \Rightarrow$
  $var := int; \mathbf{while}\ var < int\ \mathbf{do}\ var := var + int$
  eli kielioppi tuottaa ko. symbolijonon.

Säännöllisten kielten ja kontekstittomien kielioppien yhteys

- 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 aidosti laajempi kieliluokka kuin säännolliset kielet — jokainen säännöllinen kieli on kontekstiton, ja säännöllisten kielten lisäksi on olemassa ei-säännöllisiä kontektittomia kieliä.

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. Siksi lineaarisia kielioppeja nimitetään myös *säännöllisiksi* kieliopeiksi.
- Todistetaan seuraavassa väite vain oikealle lineaarisille kieliopeille:

- *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 äärellinen automaatti. Muodostetaan kielioppi G_M, jolla $L(G_M) = L(M) = L$.

  – 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}.

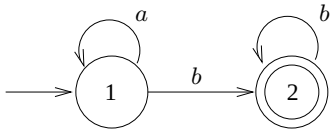
  – G_M:n produktiot vastaavat M:n siirtymiä:

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

    (ii) Kaikilla $q \in Q$ ja $a \in \Sigma$ lisätään kielioppiin produktio $A_q \rightarrow aA_{q'}$, missä $q' = \delta(q, a)$.

  Konstruktion oikeellisuuden voi todistaa induktiolla tunnistettavan/tuotettavan merkkijonon suhteen (sivuutetaan). □

- Esimerkki: Yksinkertainen äärellinen automaatti



  ja sitä vastaava kielioppi:

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

- *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 $A \rightarrow aB$ 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:

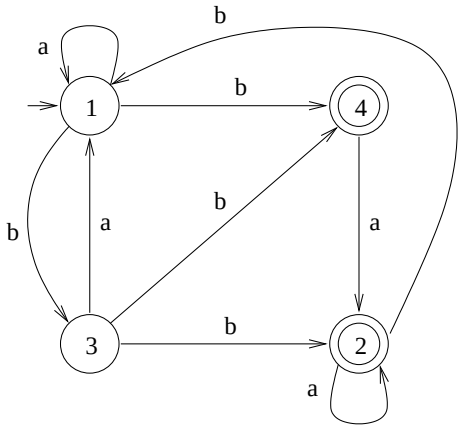
$$F = \{q_A \in Q \mid A \rightarrow \epsilon \in P\}$$

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

- Esimerkki: Tarkastellaan oikealle lineaarista kielioppia:

$$\begin{aligned} A_1 &\rightarrow aA_1 \mid bA_3 \mid bA_4 \\ A_2 &\rightarrow aA_2 \mid bA_1 \mid \epsilon \\ A_3 &\rightarrow aA_1 \mid bA_2 \mid bA_4 \\ A_4 &\rightarrow aA_2 \mid \epsilon \end{aligned}$$

Vastaava äärellinen automaatti on:



3.2. Kielioppien jäsennysongelma

- Tehtävä: “Annettu kontekstiton kielioppi G ja merkkijono x. Onko $x \in L(G)$?”
- Ratkaisumenetelmää kutsutaan *jäsennysalgoritmiksi*.
- Jäsennysongelman ratkaisemiseen on useita vaihtoehtoisia menetelmiä, erityisesti jos G on jotakin rajoitettua (käytännössä esiintyvää) muotoa.
- Esitellään ensin jäsentämiseen liittyviä peruskäsitteitä.

Johdot

- Olkoon 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

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

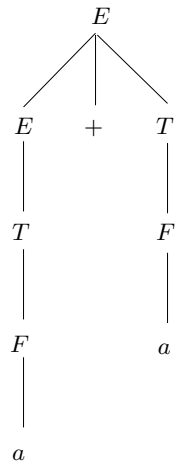
  Esim. lauseen $a + a$ johtoja kieliopissa G_{expr}:

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

- Johto $\gamma \Rightarrow^* \gamma'$ on
  1. *vasen johto*, merk. $\gamma \underset{\text{lm}}{\Rightarrow}^* \gamma'$, jos kussakin johtoaskelessa on produktiota sovellettu merkkijonon vasemmanpuoleisimpaan välikkeeseen (edellisessä esimerkissä 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 $\gamma \underset{\text{rm}}{\Rightarrow} \gamma'$.

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, josta nämä epäoleelliset erot on abstrahoitu pois.
- Jäsennyspuu kertoo ainoastaan, *miten* välikkeet on lavennettu, ei *missä järjestyksessä* lavennukset on suoritettu.
- Esim. kaikkia kolmea em. johtoa vastaa sama jäsennyspuu:



Kuva 10: 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 sen jälkeläisten nimet ovat $X_1, \ldots, X_k$ tässä 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”).

Jäsennyspuun muodostaminen

- Johtoa
$$S = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n = \gamma$$
vastaava jäsennyspuu muodostetaan seuraavasti:

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

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

  (iii) Jos seuraavassa askelessa on sovellettu produktiota $X_i \rightarrow Y_1Y_2 \ldots Y_l$, niin juuren i:nnelle jälkeläissolmulle tulee l jälkeläistä, joiden nimet vasemmalta oikealle ovat $Y_1, Y_2, \ldots, Y_l$;

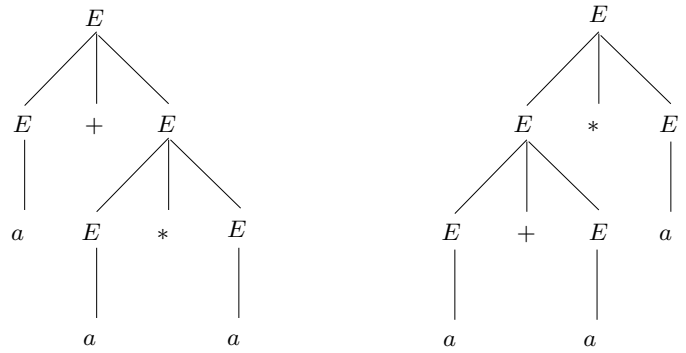
  (iv) Jatketaan samaan tapaan, kunnes ollaan johdon lopussa. Tällöin $\gamma_n = \gamma$ on luettavissa jäsennyspuun lehdistä vasemmalta oikealle.

- Jäsennysongelman ratkaisuksi katsotaan usein päätösongelman "Onko $x \in L(G)$?" ratkaisun lisäksi jäsennyspuun tai jonkin muun jäsennysesityksen tuottaminen x:lle.

- Esim. ohjelmointikielten kääntämisessä jäsennyspuita käytetään syntaksitarkistuksen lisäksi koodin generoinnissa.

Kieliopin moniselitteisyys

- Lauseella voi olla useita kieliopin mukaisia jäsennyspuita. Esim. lauseella $a + a * a$ on kieliopissa G'_{expr} seuraavat jäsennyspuut:



Kuva 11: Lauseen $a + a * a$ kaksi erilaista jäsennyspuuta.

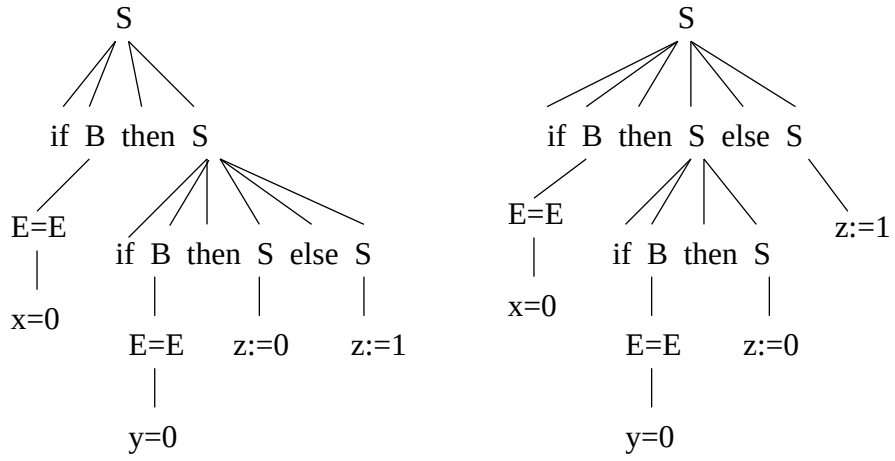
- Kontekstiton kielioppi G on *moniselitteinen*, jos jollakin G:n lauseella x on vähintään kaksi erilaista G:n mukaista jäsennyspuuta. Muuten kielioppi on *yksiselitteinen*.
- Jos kaikki kontekstittoman kielen tuottavat kieliopit ovat moniselitteisiä, sanotaan, että kieli on *luonnostaan moniselitteinen*.
- Esimerkkejä:
  - Kielioppi G'_{expr} on moniselitteinen.
  - Kieliopit G_{expr} ja G_{match} ovat yksiselitteisiä.
  - Kieli $L_{\text{expr}} = L(G'_{\text{expr}})$ ei ole luonnostaan moniselitteinen, koska sillä on myös yksiselitteinen kielioppi G_{expr}.
  - Kontekstiton kieli $\{a^i b^j c^k \mid i = j \text{ tai } j = k\}$ on luonnostaan moniselitteinen (todistus sivuutetaan).

Moniselitteiyyden aiheuttama ongelma:

- Oletetaan, että **olpe03**-kielessä olisi kaksi erityyppistä produktiota if-lauseiden tuottamiseen:

$$S \rightarrow \textbf{if } B \textbf{ then } S \textbf{ else } S \mid$$
$$\textbf{if } B \textbf{ then } S$$

- Tarkastellaan seuraavaa kieliopin mukaista **olpe03**:n lausetta: $\textbf{if } x = 0 \textbf{ then if } y = 0 \textbf{ then } z := 0 \textbf{ else } z := 1$
- Kumpaan $\textbf{if}$:iin $\textbf{else}$ liittyy?
- Lauseella on nyt kaksi erilaista jäsennyspuuta:



- Vasemmanpuoleisesta jäsennyspuusta generoitu ohjelma ei selvästikään toimisi samalla tavalla kuin oikeanpuoleisesta jäsennyspuusta generoitu!

  Eron huomaa esim. tarkastelemalla mitä koodinpätkät tekevät

kun $x = 1$: Vasemmanpuoleinen ei tee mitään, mutta oikeanpuoleinen suorittaa sijoituksen $z := 1$.

- Ongelma aiheutuu moniselitteisyydestä. Koska alkuperäinen **olpe03**:n määrittelevä kielioppi ei ole moniselitteinen (pelkät **if**:t eivät ole sallittuja), ei koko kuvattua ongelmaa esiinny.
- Esim. `C`:ssä ja `Java`ssa on kuitenkin sallittua käyttää **if**:iä ilman **else**:ä.
- Moniselitteisyyden aiheuttama ongelma voidaan kiertää joko

  (a) esittämällä kielelle yksiselitteinen kielioppi, jossa **else** liittyy aina lähimpään **if**:iin (`Java`).

  (b) käyttämällä moniselitteistä kielioppia, mutta "sopimalla", että **else** liittyy aina lähimpään **if**:iin. Tällöin kielen jäsentäjälle pitää erikseen kertoa, että edellisen sivun tilanteessa vain vasemmanpuoleinen jäsennyspuu on oikein (`C`).
- Yleensä pyrkimyksenä on yksiselitteinen kielioppi, mutta jos moniselitteisyys ei ole (yksinkertaisuutta uhraamatta) vältettävissä, on moniselitteisyydet aina selvitettävä ja ratkaistava (dokumentoidusti) jäsentäjätasolla. Muuten kielioppiin perustuvan kääntäjän käyttäjä ei voi tietää miten kääntäjä tulkitsee käyttäjän kirjoittaman ohjelman.

3.3. Rekursiivisesti etenevä jäsentäminen

LL(1)-kieliopit

- Tarkastellaan seuraavaa kielioppia G:

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

- Hankaluutena tarkastettaessa kuuluuko tietty merkkijono x kieleen $L(G)$ on se, ettei aina ole ilmeistä mikä produktio pitäisi valita.
- Esim. merkkijono $a + a$ voidaan tuottaa seuraavasti:

  $E \Rightarrow T + E \Rightarrow a + E \Rightarrow a + T \Rightarrow a + a$.

  Jos valitsemme "vahingossa" väärän produktion, voi olla, ettemme saa tuotettua merkkijonoa $a + a$ vaikka se kuuluukin kieleen:

  $E \Rightarrow T + E \Rightarrow T + T + E \Rightarrow a + T + E \Rightarrow \ldots$
- Miten siis testata (varmasti ja tehokkaasti) kuuluuko annettu merkkijono x kieleen $L(G)$?

- Muokataan kielioppia G “tekijöimällä” välikkeen E produktiot (se, mitä tekijöinti tarkkaan ottaen tarkoittaa selitetään myöhemmin). Tuloksena saadaan ekvivalentti (so. saman kielen tuottava) kielioppi G':

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

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

- Kielioppia, jolla on tämä ominaisuus, sanotaan *LL(1)-kieliopiksi* (”left to right scan, producing left parse with 1 symbol lookahead” — vast. LL(k)- ja LR(k)-kieliopit).

- LL(1)-kieliopeilla tuotettavissa olevat kielet ovat kontekstittomien kielten aito aliluokka — on olemassa kontekstittomia kieliä, joita ei voi tuottaa millään LL(1)-kieliopilla. Tämä seuraa esim. siitä, että LL(1)-kieliopit ovat aina yksiselitteisiä, joten mikään luonnostaan moniselitteinen kontekstiton kieli ei voi olla tuotettavissa LL(1)-kieliopilla.

- Tarkastellaan esimerkkinä miten merkkijonon $a+a$ *vasemman johdon* muodostaminen kieliopissa G' etenee.
  - Lähdetään liikkeelle lähtösymboolista E. Pidetään kirjaa siitä mikä osa merkkijonosta on vielä "tuottamatta". Alussa tuottamatta on $a + a$.
  - Koska välikkeeseen E liittyy vain yksi produktio, on etenemismahdollisuuksia vain yksi: $E \Rightarrow TE'$. Tuottamatta edelleen $a + a$.
  - TE':n vasemmanpuoleisimmalla välikkeellä T on kaksi produktiota. Koska tuotettavan merkkijonon ensimmäisenä merkkinä on a, on valittava produktioista ensimmäinen: $\ldots \Rightarrow TE' \Rightarrow aE'$. Tuottamatta $+a$.
  - E':lla on kolme produktiota, mutta tuotettavan merkkijonon seuraava käsittelemätön merkki + pakottaa valitsemaan produktioista ensimmäisen: $\ldots \Rightarrow aE' \Rightarrow a + E$. Tuottamatta a.
  - Nyt etenemismahdollisuuksia on vain yksi: $\ldots \Rightarrow a+E \Rightarrow a+TE'$. Tuottamatta edelleen a.
  - T:n produktioista valitaan ensimmäinen koska seuraava käsittelemätön merkki on a: $\ldots \Rightarrow a+TE' \Rightarrow a+aE'$. Koko merkkijono on nyt tuotettu.
  - Lopulta valitsemme E':n produktioista kolmannen: $\ldots \Rightarrow a + aE' \Rightarrow a + a$.
  - Yhdistämällä askelet saadaan etsitty vasen johto merkkijonolle $a + a$.

- Tarkastellaan toisena esimerkkinä merkkijonoa $(a-a)+a$.

seuraava tuottoaskel	tuottamatta
$E \Rightarrow TE'$	$(a-a)+a$
$TE' \Rightarrow (E)E'$	$a-a)+a$
$(E)E' \Rightarrow (TE')E'$	$a-a)+a$
$(TE')E' \Rightarrow (aE')E'$	$-a)+a$
$(aE')E' \Rightarrow (a-E)E'$	$a)+a$
$(a-E)E' \Rightarrow (a-TE')E'$	$a)+a$
$(a-TE')E' \Rightarrow (a-aE')E'$	$)+a$
$(a-aE')E' \Rightarrow (a-a)E'$	$+a$
$(a-a)E' \Rightarrow (a-a)+E$	a
$(a-a)+E \Rightarrow (a-a)+TE'$	a
$(a-a)+TE' \Rightarrow (a-a)+aE'$	
$(a-a)+aE' \Rightarrow (a-a)+a$	

- Huomaa yllä miten merkki $)$ tuotetaan “valmiiksi” jo kolmantena käytetystä produktiosta $T \rightarrow (E)$, mutta se poistuu tuottamatta olevista merkeistä vasta kun sulkujen sisäpuolella oleva merkkijono on saatu käsiteltyä.

LL(1)-kieliopin rekursiivisesti etenevä jäsentäminen

- Edellä kuvatulla tavalla voidaan tuottaa mikä tahansa kielen $L(G')$ merkkijono — sovellettava produktio voidaan aina päätellä katsomalla mikä on tuotettavan merkkijonon seuraava käsittelemätön merkki.
- Yllä kuvattu tapa jäsentää kielen $L(G')$ merkkijonoja on kirjoitettavissa algoritmin muotoon. Saatua algoritmia kutsutaan *rekursiivisesti eteneväksi jäsentäjäksi*, ja se muodostuu välikkeitä vastaavista rekursiivisista funktioista.
- Funktiolla *getnext* jäsentäjä lukee seuraavan syötemerkin. Allaolevassa esimerkissä jäsentäjä ainoastaan tulostaa valitut produktiot, mutta samalla idealla voisi tehdä jotain järkevämpääkin (esim. generoida ohjelman, joka evaluoi aritmeettisen lausekkeen arvon).

```
function E'
   if (next == '+') then
        writeln("E' → +E");
        next = getnext;
        E;
   else if (next == '-') then
        writeln(E' → −E);
        next = getnext;
        E;
   else writeln(E' → ε);

function E
   writeln(E → T E');
   T;
   E';

function T
   if (next == 'a') then
        writeln(T → a);
        next = getnext;
        return;
   else if (next == '(') then
        writeln(T → (E));
        next = getnext;
        E;
        if (next ≠ ')') then
             ERROR;
        next = getnext;
   else ERROR;
```

Pääohjelmassa:

```
next = getnext;
E;
```

- Esim. syötejonon `a-(a+a)` jäsennys:

  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:

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

Kielioppien muokkaaminen LL(1)-muotoon

- Seuraavilla muunnoksilla voidaan joitakin “melkein’ LL(1)-muotoisia kielioppeja muuntaa LL(1)-muotoon.
- Vasen tekijöinti:
  - Kielioppi, jossa on produktiot

    $$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n, \qquad \alpha \neq \epsilon,\ \beta_i \neq \beta_j$$

    ei voi olla LL(1)-muotoinen (paitsi jos ainoa α:n tuottama päätejono on ϵ).
  - Parannus: Otetaan käyttöön uusi välike A' ja korvataan em. produktiot produktioilla:

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

    missä α siis on merkkijonojen $\alpha\beta_1, \alpha\beta_2, \ldots, \alpha\beta_n$ pisin yhteinen alkuosa.

- Esimerkki:

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

  – Tekijöidään välikkeen E produktiot. E:n produtioiden yhteinen alkuosa on T, joten valitaan $\alpha = T$ ja $\beta_1 = +E$, $\beta_2 = -E$, ja $\beta_3 = \epsilon$.

  – Otetaan yllä esitetyn mukaisesti käyttöön uusi välikesymboli E' sekä muokataan produktiot muotoon:

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

- Välittömän vasemman rekursion poisto:
  - Kielioppi, jossa on muotoa $A \rightarrow A\gamma$ olevia produktioita, ei voi täyttää LL(1)-ehtoa (paitsi jos välikkeeseen A ei liity muita produktioita).
  - *Välitön* vasen rekursio, so. suorat johdot $A \Rightarrow A\gamma$, voidaan välttää korvaamalla produktiot

    $$A \rightarrow A\alpha \mid \beta_1 \mid \ldots \mid \beta_n \qquad \alpha \neq \epsilon,$$

    produktioilla

    $$\begin{aligned} A &\rightarrow \beta_1 A' \mid \ldots \mid \beta_n A' \\ A' &\rightarrow \alpha A' \mid \epsilon \end{aligned}$$

  - Muotoa $A \rightarrow A$ olevat produktiot voidaan yksinkertaisesti jättää pois.
- Esimerkki:

  $$A \rightarrow Aa \mid b \mid c$$

  - Nyt $\alpha = a$, $\beta_1 = b$ ja $\beta_2 = c$.
  - Soveltamalla sääntöä saamme muokattua kieliopin muotoon:

    $$\begin{aligned} A &\rightarrow bA' \mid cA' \\ A' &\rightarrow aA' \mid \epsilon \end{aligned}$$

- Esimerkkinä sääntöjen käytöstä muokataan vielä kielioppi G'_{expr}

$$E \ \rightarrow \ E + E \mid E * E \mid (E) \mid a$$

  LL(1)-muotoon.

- Suoritetaan ensin vasen tekijöinti. Otetaan käyttöön uusi välike F, ja muokataan produktioita seuraavasti:

$$\begin{aligned} E &\rightarrow EF \mid (E) \mid a \\ F &\rightarrow +E \mid *E \end{aligned}$$

- Poistetaan vielä E:ssä esiintyvä vasen rekursio:

$$\begin{aligned} E &\rightarrow (E)E' \mid aE' \\ E' &\rightarrow FE' \mid \epsilon \\ F &\rightarrow +E \mid *E \end{aligned}$$

LL(1)-kielioppien tuolla puolen

- LL(1)-kieliopit ovat suhteellisen rajoittunut kontekstittomien kielten aliluokka.
- LL(1)-kieliopit ovat kiinnostavia lähinnä siksi, että niille on helppo kirjoittaa käsin tehokas rekursiivinen jäsentäjä edellä esitetyllä tavalla.
- Jos kielioppi ei ole valmiiksi LL(1)-muodossa, se voidaan yrittää muokata LL(1)-muotoiseksi edellä esitetyillä menetelmillä. Tosin tämä ei tietenkään aina onnistu, sillä kaikkia kontekstittomia kieliä ei voi tuottaa LL(1)-kieliopeilla.
- Koska edes kaikkia ohjelmointikielissä esiintyviä konstruktioita ei ole helppoa tai mahdollista kuvata LL(1)-kieliopeilla, turvaudutaan usein laajempaan LR(1)-kielioppien luokkaa. Tämäkään kielioppien luokka ei riitä kaikkien kontekstittomien kielten kuvaamiseen, mutta on kuitenkin riittävän rikas useimpiin käytännön tarpeisiin.
- Tehokkaan jäsentäjien käsin kirjoittaminen LR(1)-kieliopeille on hankalaa, mutta voidaan onneksi tehdä automaattisesti. Tästä syystä LR(1)-kielioppeja (ja niiden LL(1)-kielioppien luokkaa rikkaampia aliluokkia) käytetäänkin käytännössä paljon.
- Valitettavasti LR(1)-kielioppeihin ei tällä kurssilla voida perehtyä. Niiden kanssa saman kieliluokan tunnistavat deterministiset pinoautomaatit ehditään onneksi pikaisesti esitellä.

3.4. CYK-jäsennysalgoritmi

- Rekursiivisesti etenevä jäsentäminen on selkeä ja tehokas jäsennysmenetelmä LL(1)-kieliopeille: n merkin mittaisen syötemerkkijonon käsittely sujuu ajassa $O(n)$.
- Entäpä mielivaltaisen, ei LL(1) muotoisen kontekstittoman kieliopin tuottamien merkkijonojen tunnistaminen?
- Ratkaisu: *Cocke–Younger–Kasami-* eli *CYK-algoritmi*
  - Toimii ajassa $O(n^3)$, missä n on tutkittavan merkkijonon pituus.
  - Kieliopin on oltava *Chomskyn normaalimuodossa*.
- Tarkastellaan ensin, miten mielivaltainen kontekstiton kielioppi voidaan muuntaa Chomskyn normaalimuotoon.
- Chomskyn normaalimuoto ei ole kovin ihmisystävällinen. Ideana onkin, että ihminen käsittelee kontekstittomia kielioppeja itselleen havainnollisemmassa muodossa. Chomskyn normaalimuotoa käytetään ainoastaan tunnistusongelman ratkaisemiseen CYK-algoritmilla, ja muunnos Chomskyn normaalimuotoon voidaan suorittaa automaattisesti muunnosalgoritmilla.

Chomskyn normaalimuoto

- *Määritelmä:* Kontekstiton kielioppi $G = (V, \Sigma, P, S)$ on *Chomskyn normaalimuodossa*, jos
  - Välikkeistä enintään S on tyhjentyvä (engl. nullable). Ts. millään muulla välikkeellä A ei saa päteä $A \underset{G}{\Rightarrow}^* \epsilon$, mutta produktio $S \to \epsilon$ on sallittu.
  - 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:

$$\begin{aligned} S &\to AB|\epsilon \\ A &\to BA|a \\ B &\to b \end{aligned}$$

Muunnos Chomskyn normaalimuotoon

- Idea:
  1. Poistetaan lähtösymboli produktioiden oikealta puolelta.
  2. Poistetaan muihin kuin lähtösymboliin liittyvät ϵ-produktiot.
  3. Poistetaan yksikköproduktiot $A \rightarrow B$
  4. Poistetaan päätemerkit muiden kuin $A \rightarrow a$ muotoisten produktioiden oikealta puolelta.
  5. Poistetaan produktiot $A \rightarrow X_1X_2...X_k$, $k \geq 3$
- Poistot pitää tietysti tehdä siten, että kieliopin tuottama kieli ei muutu.

Lähtösymbolin poistaminen produktioiden oikealta puolelta

- Jos lähtösymboli S on jonkin produktion oikealla puolella, lisätään uusi lähtösymboli S' ja sille produktio $S' \to S$.

ϵ-produktioiden poistaminen

- *Määritelmä:* Välike A 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.

  Todistus: Olkoon $G = (V, \Sigma, P, S)$.

  1. Selvitetään ensin G:n tyhjentyvät välikkeet. Merkitään näiden joukkoa NULL $= \{A \in V \setminus \Sigma \mid A \underset{G}{\Rightarrow}^* \epsilon\}$. Nyt $A \in$ NULL, joss $A \underset{G}{\Rightarrow}^* \epsilon$, eli jos
     - A:lla produktio $A \to \epsilon$, tai
     - A tyhjentyy epäsuorasti, esim. $A \to BC$, ja $B \to \epsilon$ sekä $C \to \epsilon$.

NULL joukko voidaan muodostaa seuraavasti:

(i) Asetetaan aluksi

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

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

$$\begin{aligned}\text{NULL} \;:=\; &\text{NULL} \;\cup \\ &\{A \in V \setminus \Sigma \mid A \to B_1 \ldots B_k \in P \text{ ja} \\ &\quad B_i \in \text{NULL}\ \forall\, i = 1, \ldots, k\}.\end{aligned}$$

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

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

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 $S' \to \epsilon$. $\square$

Esimerkki

- Poistetaan ϵ-produktiot seuraavasta kieliopista:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aBa \mid \epsilon \qquad\qquad \Rightarrow \quad \text{NULL} = \{A, B, S\} \\ B &\rightarrow bAb \mid \epsilon \end{aligned}$$

  Koska $B \in$ NULL, joudumme korvaamaan A:n produktion $A \rightarrow aBa$ produktioilla $A \rightarrow aBa \mid a\epsilon a$ missä jälkimäinen siis on sama kuin $A \rightarrow aa$.

  Koska $A \in$ NULL, joudumme korvaamaan produktion $B \rightarrow bAb$ produktioilla $B \rightarrow bAb \mid bb$.

  Vastaavasti muuttuvat S:n produktiot, ja tuloksena on:

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

  Poistetaan ϵ-produktiot ja otetaan käyttöön uusi lähtösymbooli:

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

- Muokkasimme siis kielioppia askelen verran kohti Chomskyn normaalimuotoa: muokatussa versiossa on enää korkeintaan lähtösymboliin liittyviä ϵ-produktioita.

Yksikköproduktioiden poistaminen

- Produktio, joka on muotoa $A \rightarrow B$, 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 $G = (V, \Sigma, P, S)$. Edellisten muunnosvaiheiden nojalla voidaan olettaa, ettei G:ssä ole ϵ-produktioita (paitsi ehkä lähtösymbolilla S, joka ei esiinny minkään produktion oikealla puolella).

  1. Selvitetään ensin G:n kunkin välikkeen "yksikköseuraajat". Välikkeen A yksilöseuraajat kerätään joukkoon $F(A) = \{B \in V \setminus \Sigma \mid A \underset{G}{\Rightarrow}^* B, B \neq A\}$.

     Joukon $F(A)$ muodostamiseksi käydään läpi kaikki produktiot $B \in V \setminus \Sigma$. Joukon $F(A)$ määritelmän mukaan $B \in F(A)$ jos $B \neq A$ ja $A \underset{G}{\Rightarrow}^* B$, eli joko
     - A:lla produktio $A \rightarrow B$, tai
     - A voi tuottaa B:n epäsuorasti, esim. $A \rightarrow C$, ja $C \rightarrow B$.

Joukot $F(A)$ voidaan laskea seuraasti:

(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 seuraavaa F-joukkojen laajennusoperaatiota, kunnes joukot eivät enää kasva:

$$F(A) := F(A) \cup \{F(B) \mid A \to B \in P\}.$$

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

Esimerkki

- Poistetaan yksikköproduktiot edellä saadusta kieliopista

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

S':n yksikköseuraajia ovat A, B ja S, sillä $S' \rightarrow S$, ja koska $S \rightarrow A$ ja $S \rightarrow B$, niin $S' \Rightarrow^* A$ ja $S' \Rightarrow^* B$. Siis $F(S') = \{S, A, B\}$. Muiden välikkeiden yksikköseuraajat saadaan selvitettyä samaan tapaan, tuloksena $F(S) = \{A, B\}$, $F(A) = F(B) = \emptyset$.

Korvaamalla yksikköproduktiot edellä esitetyllä tavalla saadaan kielioppi

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

Tähän on päädytty, koska $A, B, S \in F(S')$, joten S':uun lisätään kaikki A:n, B:n ja S:n ei-yksikköproduktiot. Koska $A, B \in F(S)$, lisätään S:ään kaikki A:n ja B:n ei-yksikköproduktiot.

- Edellisten vaiheiden jälkeen kielioppin on jo melkein Chomskyn normaalimuodossa. Tyhjentyviä välikkeitä ja yksikköproduktioita ei enää ole. Ongelmana ovat vielä produktiot, joissa on oikealla puolella päätemerkkejä, kuten $B \rightarrow bb$, tai joiden oikea puoli koostuu useammasta kuin kahdesta välikkeestä, kuten $A \rightarrow BAC$.
- Ensimmäinen näistä ongelmakohdista poistuu seuraavalla menetelmällä:

Päätemerkkien poistaminen produktioiden oikealta puolelta

- Lisätään kielioppiin kutakin päätemerkkiä $a \in \Sigma$ kohden uusi välike C_a ja sille produktio $C_a \rightarrow a$.
- Korvataan sitten kunkin produktion oikealla puolella (poislukien $A \rightarrow a$ muotoiset produktiot) kaikki päätemerkit em. uusilla välikkeillä.

Esimerkki

- Esimerkkien kielioppi muokkautuu seuraavaan muotoon:

$$\begin{aligned} S' &\rightarrow C_aBC_a \mid C_aC_a \mid C_bAC_b \mid C_bC_b \mid \epsilon \\ S &\rightarrow C_aBC_a \mid C_aC_a \mid C_bAC_b \mid C_bC_b \\ A &\rightarrow C_aBC_a \mid C_aC_a \\ B &\rightarrow C_bAC_b \mid C_bC_b \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

Produktioiden $A \rightarrow X_1 \ldots X_k$, $k \geq 3$, poisto

- Produktiot, joiden oikealla puolella on enemmän kuin kaksi välikettä, voidaan pilkkoa pienemmiksi palasiksi. Avuksi otetaan uusia välikkeitä.
- Esim. produktio $A \rightarrow BCDE$ pilkotaan seuraavasti:

$$\begin{aligned} A &\rightarrow BA_1 \\ A_1 &\rightarrow CA_2 \\ A_2 &\rightarrow DE \end{aligned}$$

- Menetelmä yleisessä muodossa on seuraava:
  - Korvataan kukin muotoa $A \rightarrow X_1 \ldots X_k$, $k \geq 3$, oleva pro-

duktio produktiojoukolla

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

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

Esimerkki

- Sovelletaan edellä esitettyä produktioiden pilkkomismenetelmää esimerkkikielioppiimme.

$$
\begin{aligned}
S' &\rightarrow C_aS'_1 \mid C_aC_a \mid C_bS'_2 \mid C_bC_b \mid \epsilon \\
S'_1 &\rightarrow BC_a \\
S'_2 &\rightarrow AC_b \\
S &\rightarrow C_aS_1 \mid C_aC_a \mid C_bS_2 \mid C_bC_b \\
S_1 &\rightarrow BC_a \\
S_2 &\rightarrow AC_b \\
A &\rightarrow C_aA_1 \mid C_aC_a \\
A_1 &\rightarrow BC_a \\
B &\rightarrow C_bB_1 \mid C_bC_b \\
B_1 &\rightarrow AC_b \\
C_a &\rightarrow a \\
C_b &\rightarrow b
\end{aligned}
$$

- Kielioppi on viimein Chomskyn normaalimuodossa.
- On helppo huomata, että lopputuloksena saatu Chomskyn normaalimuotoinen kielioppi on epähavainnollisempi kuin alkuperäinen kielioppi. Chomskyn normaalimuoto onkin suunniteltu CYK-algoritmia eikä ihmistä silmälläpitäen.

Yhteenveto Chomskyn normaalimuotoon muuntamisesta

- *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 sekä korvataan muiden kuin $A \to a$ muotoisten produktioiden oikealla puolella olevat päätesymbolit välikesymboleilla. Tämän jälkeen kaikki G:n produktiot ovat muotoa $S \to \epsilon$, $A \to a$ tai $A \to X_1 \ldots X_k$, $k \geq 3$. Viimeksi mainitut poistetaan edellä esitetyn pilkkomiskonstruktion mukaisesti. $\Box$

Esimerkki

- Muunnetaan Chomskyn normaalimuotoon kielioppi

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

Tuloksena saadaan kielioppi

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

Välivaiheiden läpikäyminen jätetään vapaaehtoiseksi harjoitustehtäväksi...

Cocke–Younger–Kasami-algoritmi

- Perustuu dynaamiseen ohjelmointiin (välitulosten taulukointiin sopivassa järjestyksessä).
- Ongelma: Syötteenä Chomskyn normaalimuodossa oleva kielioppi $G = (V, \Sigma, P, S)$ ja syötemerkkijono $x \in \Sigma^*$. Ratkaistava, onko $x \in L(G)$ eli päteekö $S \underset{G}{\Rightarrow}^* x$?
- 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. Nyt $x \in L(G) \Leftrightarrow S \in N_{1,n}$. Taulukon muut alkiot ovat apuvälineitä alkion $N_{1,n}$ laskemiseksi.
- Joukkojen $N_{i,j}$ laskeminen
  - Muodostetaan kolmiomainen taulukko:

$$\begin{array}{lllll} N_{1,1} & & & & \\ N_{1,2} & N_{2,2} & & & \\ N_{1,3} & N_{2,3} & N_{3,3} & & \\ N_{1,4} & N_{2,4} & N_{3,4} & N_{4,4} & \\ N_{1,5} & N_{2,5} & N_{3,5} & N_{4,5} & N_{5,5} \end{array}$$

  - x-akseli vastaa merkkijonon x osajonon alkupositiota.

– Taulukon paikka $N_{i,j}$ $(1 \leq i \leq j \leq n) \sim$ joukko välikesymboleita A, joille $A \underset{G}{\Rightarrow}^* a_i a_{i+1} \ldots a_j$

– Jokainen diagonaali vastaa tietyn pituisia x:n osajonoja

  * 1. diagonaali $\sim$ välikkeet, jotka tuottavat x:n yhden merkin osajonot a_i, $1 \leq i \leq n$,
  * 2. diagonaali $\sim$ välikkeet, jotka tuottavat x:n kahden merkin pituiset osajonot $a_i a_{i+1}$, $1 \leq i \leq n-1$, jne.

(i) Lasketaan ensin kaikille $i \in \{1, \ldots, n\}$:

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

(ii) Täytetään taulukon muut diagonaalit seuraavasti:

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

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

$$N_{i,i+k} := \{A \in V - \Sigma \mid \text{jollakin } j,\ i \leq j < i + k, \text{ on välikkeet } B \in N_{ij} \text{ ja } C \in N_{j+1,i+k}, \text{ joille } A \to BC \in P\ \}$$

Idea:

– Johdon $A \underset{G}{\Rightarrow}^* a_i a_{i+1} \ldots a_j$ täytyy alkaa askeleella $A \underset{G}{\Rightarrow} BC$, missä B:stä voidaan johtaa $a_i \ldots a_j$:n prefiksi ja C:stä joku $a_i \ldots a_j$:n suffiksi:
$B \underset{G}{\Rightarrow}^* a_i a_{i+1} \ldots a_l$ ja $C \underset{G}{\Rightarrow}^* a_{l+1} \ldots a_j$ jollakin $i \leq l < j$.

– Esim. taulukon alkion $N_{3,7}$ ($k = 4$ ja $i = 3$) laskemiseksi läpikäydään alkiot
$(N_{3,3}, N_{4,7}), (N_{3,4}, N_{5,7}), (N_{3,5}, N_{6,7}), (N_{3,6}, N_{7,7})$

Esimerkki

- Sovelletaan CYK:iä allaolevaan Chomskyn normaalimuotoiseen kielioppiin G

$$\begin{aligned} S &\rightarrow AB \mid BA \mid \epsilon \\ A &\rightarrow AB \mid a \\ B &\rightarrow BA \mid b \end{aligned}$$

syötemerkkijonolla $x = abba$.

Tavoitteena siis laskea oheisen taulukon $N_{i,j}$ arvot:

	$1:a$	$2:b$	$3:b$	$4:a$
1	$N_{1,1}$			
2	$N_{1,2}$	$N_{2,2}$		
3	$N_{1,3}$	$N_{2,3}$	$N_{3,3}$	
4	$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$

Taulukon alkioon $N_{i,j}$ kerätään siis välikkeet, joista lähtien on mahdollista johtaa syötemerkkijonon positiosta i alkava ja positioon j päättyvä osajono.

  - $N_{1,1}$ ja $N_{4,4}$: välikkeet, joista voidaan johtaa merkkijono a,
  - $N_{2,2}$ ja $N_{3,3}$: välikkeet, joista voidaan johtaa merkkijono b,
  - $N_{1,2}$: välikkeet, joista voidaan johtaa merkkijono ab,
  - $N_{2,3}$: välikkeet, joista voidaan johtaa merkkijono bb,
  - $N_{3,4}$: välikkeet, joista voidaan johtaa merkkijono ba,
  - $N_{1,3}$: välikkeet, joista voidaan johtaa merkkijono abb,

– $N_{2,4}$: välikkeet, joista voidaan johtaa merkkijono bba, ja
– $N_{1,4}$: välikkeet, joista voidaan johtaa merkkijono $abba$.

Laskenta etenee diagonaaleja pitkin eli siten, että ensin täytetään taulukon alkiot jotka vastaavat x:n yhden pituisia osajonoja, sitten alkiot jotka vastaavat x:n kahden pituisia osajonoja, ...

Ensimmäinen vaihe on selkeä: koska ainoastaan $A \to a$, $N_{1,1} = N_{4,4} = \{A\}$. Vastaavasti ainoastaan $B \to b$, joten $N_{2,2} = N_{3,3} = \{B\}$. Siis saadaan:

	$1:a$	$2:b$	$3:b$	$4:a$
1	$\{A\}$			
2	$N_{1,2}$	$\{B\}$		
3	$N_{1,3}$	$N_{2,3}$	$\{B\}$	
4	$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$\{A\}$

Seuraavana vuorossa x:n kahden pituisia osajonoja vastaavat taulukon alkiot eli alkiot $N_{1,2}$, $N_{2,3}$ ja $N_{3,4}$.

$N_{1,2}$:n arvo saadaan selville alkioiden $N_{1,1}$ ja $N_{2,2}$ avulla seuraavasti:

$$N_{1,2} = \{X \mid \text{kieliopissa on produktio } X \to YZ, \text{ missä } Y \in N_{1,1} \text{ ja } Z \in N_{2,2}\}$$

Näin ollen $N_{1,2} = \{S, A\}$. Joukko $N_{1,2}$ sisältää siis välikkeet, joista pystytään tuottamaan merkkijono ab.

Samalla logiikalla $N_{2,3}$ arvo saadaan selville alkioiden $N_{2,2}$ ja $N_{3,3}$ avulla:

$$N_{2,3} = \{X \mid \text{kieliopissa on produktio } X \to YZ, \text{ missä } Y \in N_{2,2} \text{ ja } Z \in N_{3,3}\}$$

Näin ollen $N_{2,3} = \emptyset$, sillä kieliopissa ei ole produktiota, jonka oikeana puolena olisi BB.

Koska joukko $N_{2,3}$ sisältää ne välikkeet joista pystytään tuottamaan merkkijono bb ja joukko on tyhjä, seuraa, ettei kieliopin mistään välikkeestä pysty johtamaan merkkijonoa bb.

Joukon $N_{3,4}$ laskemiseksi menetellään samoin. Koska $S \to BA$ ja $B \to BA$, saadaan $N_{3,4} = \{S, B\}$. Eli $S \Rightarrow^* ba$ ja $B \Rightarrow^* ba$.

Toisen vaiheen jälkeen taulukko näyttää seuraavalta:

	$1:a$	$2:b$	$3:b$	$4:a$
1	$\{A\}$			
2	$\{S, A\}$	$\{B\}$		
3	$N_{1,3}$	$\emptyset$	$\{B\}$	
4	$N_{1,4}$	$N_{2,4}$	$\{S, B\}$	$\{A\}$

Seuraavana vuorossa alkioiden $N_{1,3}$ ja $N_{2,4}$ arvojen selvittäminen.

Alkion $N_{1,3}$ arvo lasketaan seuraavasti:

$$N_{1,3} = \{X \mid \text{jos kieliopissa produktio } X \to YZ, \text{ missä joko } Y \in N_{1,1} \text{ ja } Z \in N_{2,3} \text{ tai } Y \in N_{1,2} \text{ ja } Z \in N_{3,3}\}$$

Näin ollen $N_{1,3} = \{S, A\}$, mikä nähdään käymällä läpi G:n produktiot ja joukkojen $N_{1,1}, N_{2,3}$ ja $N_{1,2}, N_{3,3}$ sisältämät välikkeet.

$N_{2,4}$ arvo:

$$N_{2,4} = \{X \mid \text{jos kieliopissa produktio } X \to YZ \text{ missä,}$$
$$\text{joko } Y \in N_{2,2} \text{ ja } Z \in N_{3,4}$$
$$\text{tai } Y \in N_{2,3} \text{ ja } Z \in N_{4,4} \}$$

Nyt $N_{2,4} = \emptyset$, mikä taas nähdään läpikäymällä kaikki produktiot ja yllä esiintyvät välikejoukot. Siispä mikään kieliopin välike ei tuota merkkijonoa bba.

Kolmannen vaiheen jälkeen taulukko näyttää seuraavalta:

	$1:a$	$2:b$	$3:b$	$4:a$
1	$\{A\}$			
2	$\{S, A\}$	$\{B\}$		
3	$\{S, A\}$	$\emptyset$	$\{B\}$	
4	$N_{1,4}$	$\emptyset$	$\{S, B\}$	$\{A\}$

Lopultakin on vuorossa taulukon alkion $N_{1,4}$ arvon selvittäminen, joka lasketaan seuraavasti:

$$N_{1,4} = \{X \mid \text{jos kieliopissa produktio } X \to YZ \text{ missä,}$$
$$\text{joko } Y \in N_{1,1} \text{ ja } Z \in N_{2,4}$$
$$\text{tai } Y \in N_{1,2} \text{ ja } Z \in N_{3,4}$$
$$\text{tai } Y \in N_{1,3} \text{ ja } Z \in N_{4,4} \}$$

Pienen (ja toivottavasti tutuksi käyneen) puuhastelun jälkeen saamme $N_{1,4} = \{S, A\}$.

Taulukko on nyt valmis:

	$1:a$	$2:b$	$3:b$	$4:a$
1	$\{A\}$			
2	$\{S, A\}$	$\{B\}$		
3	$\{S, A\}$	$\emptyset$	$\{B\}$	
4	$\{S, A\}$	$\emptyset$	$\{S, B\}$	$\{A\}$

Joukko $N_{1,4}$ sisältää täsmälleen ne välikkeet, joista lähtien voidaan tuottaa merkkijono $a_1 \dots a_4 = abba = x$. Koska lähtösymboli S kuuluu joukkoon $N_{1,4}$, kuuluu $abba$ määritelmän mukaan kieliopin tuottamaan kieleen.

- Toisena esimerkkinä sovelletaan CYK:iä Chomskyn normaalimuotoiseen kielioppiin

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

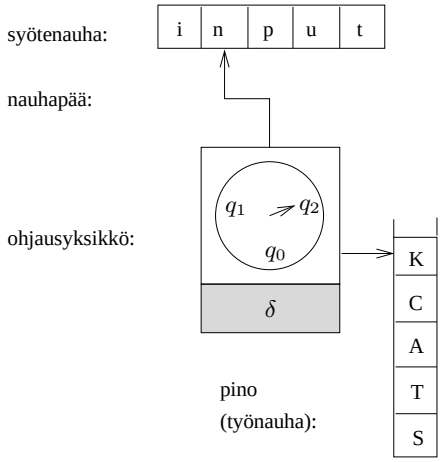
  syötemerkkijonolla $x = baaba$.

  Taulukon täyttäminen etenee samoin kuin edellisessä esimerkissä. Lopputuloksena saadaan:

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

- Lähtösymboli S kuuluu joukkoon $N_{1,5}$, joten x kuuluu kieliopin tuottamaan kieleen.

3.5. Pinoautomaatit



Kuva 12: Pinoautomaatti.

- Pinoautomaatti on äärellinen automaatti, johon on lisätty yksi *pino* työnauhaksi.
- Nauhaa sanotaan pinoksi, koska automaatti voi lukea ja kirjoittaa vain sen toiseen päähän, ja päästäkseen lukemaan aiemmin kirjoittamiaan merkkejä sen täytyy pyyhkiä niiden jälkeen kirjoittamansa merkit pois.
- Työnauha antaa automaatille “muistin”, jonka avulla voidaan tunnistaa joitakin kieliä joita äärellisillä automaateilla ei voi tunnistaa.

- *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.

- Siirtymän

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

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 vastaavasti 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, vaan painaa uuden merkin pinon päälle (push);
3. Jos $\gamma \neq \epsilon$ ja $\gamma_i = \epsilon$, niin ponnautetaan γ eikä paineta uutta merkkiä pinoon (pop).

- Pinoautomaatit ovat siis yleisessä tapauksessa *epädeterminististisiä*.

- 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, α_0), …, (q_n, w_n, α_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

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

  (eli 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

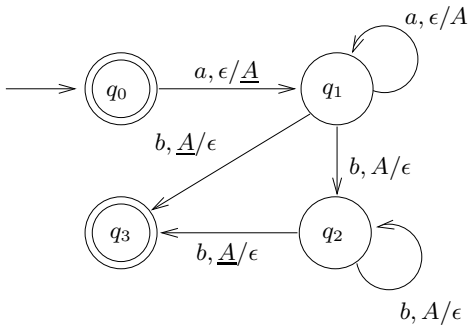
$$\begin{aligned}
\delta(q_0, a, \epsilon) &= \{(q_1, \underline{A})\}, \\
\delta(q_1, a, \epsilon) &= \{(q_1, A)\}, \\
\delta(q_1, b, A) &= \{(q_2, \epsilon)\}, \\
\delta(q_1, b, \underline{A}) &= \{(q_3, \epsilon)\}, \\
\delta(q_2, b, A) &= \{(q_2, \epsilon)\}, \\
\delta(q_2, b, \underline{A}) &= \{(q_3,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

Esim. syötteellä $aabb$ automaatti M toimii seuraavasti:

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

Koska $q_3 \in F = \{q_0, q_3\}$, on siis $aabb \in L(M) = \{a^k b^k \mid k \geq 0\}$.

- Pinoautomaatin voi esittää äärellisten automaattien tapaan myös kaaviomuodossa.
- Esim. Kielen $\{a^k b^k \mid k \geq 0\}$ tunnistava pinoautomaatti:



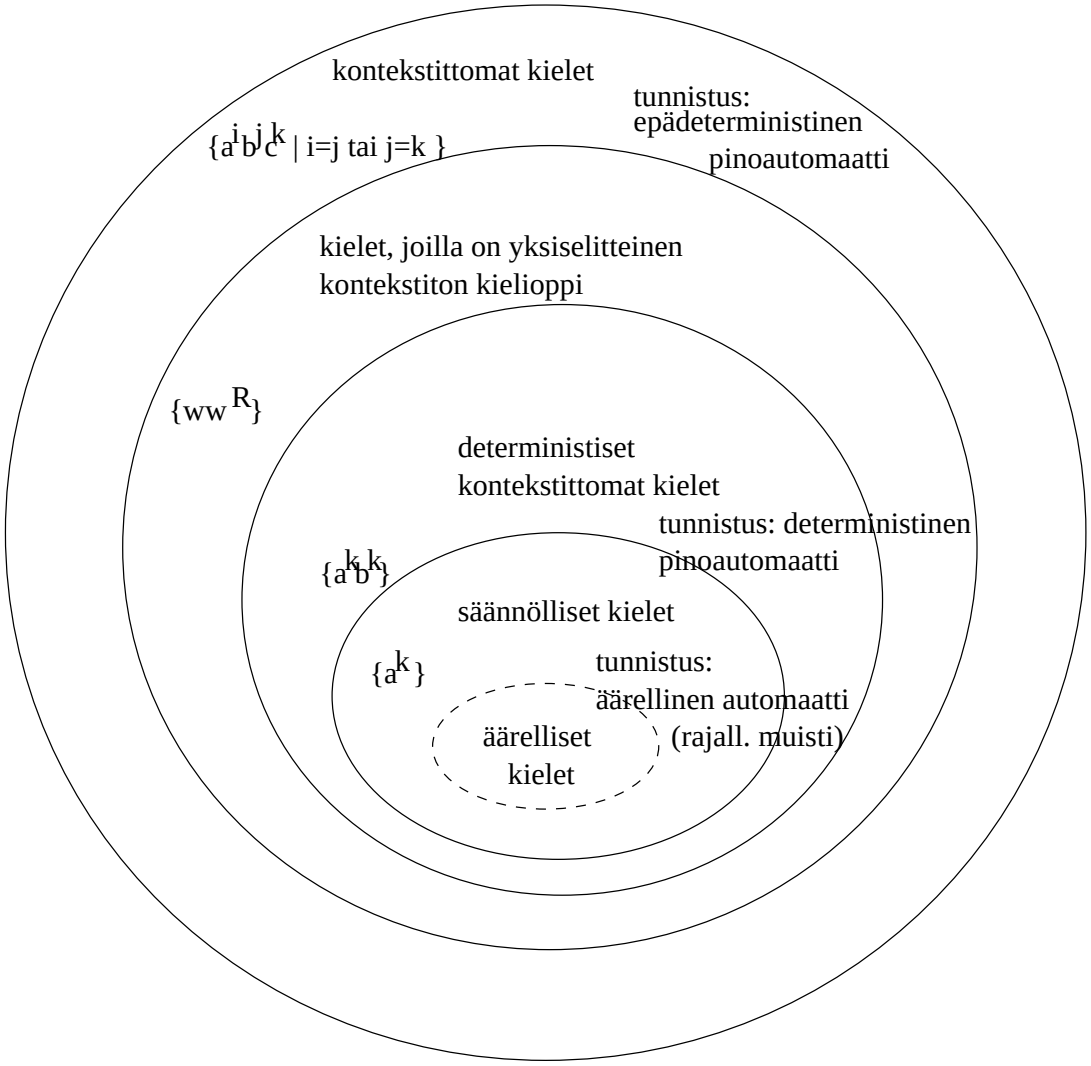
Kuva 13: Kielen $\{a^k b^k \mid k \geq 0\}$ tunnistava pinoautomaatti, jossa pinoaakkosto on $\{A, \underline{A}\}$.

- Semantiikka on seuraava:
  - Siirtymäehto muotoa $a, \gamma/\gamma'$, missä
    - automaatti lukee merkkijonosta merkin a ja
    - pinosta merkin γ sekä
    - kirjoittaa pinoon merkin γ'
  - ”Push”-operaatio: ei lue pinosta merkkiä, mutta kirjoittaa uuden merkin γ' pinon päälle: $a, \epsilon/\gamma'$.
  - ”Pop”-operaatio: lukee pinon päällimmäisen merkin γ, mutta ei kirjoita mitään pinoon: $a, \gamma/\epsilon$.
  - Jos syötemerkki $a = \epsilon$, niin siirtymä syötemerkkiä lukematta — automaatti voi silti lukea tai kirjoittaa pinomerkkejä!
- *Lause:* Kieli on kontekstiton, jos ja vain jos se voidaan tunnistaa pinoautomaatilla.
  Todistus: Sivuutetaan. □

Deterministiset ja epädeterministiset pinoautomaatit

- Pinoautomaatti M on *deterministinen*, jos jokaisella tilanteella (q, w, α) on enintään yksi mahdollinen seuraaja (q', w', α'), so. tila jolle
$$(q, w, \alpha) \underset{M}{\vdash} (q', w', \alpha').$$
- Huom: *Epädeterministiset pinoautomaatit ovat aidosti vahvempia kuin deterministiset*! (vrt. äärelliset automaatit)
- Esim. kieli $\{ww^R \mid w \in \{a, b\}^*\}$ voidaan tunnistaa epädeterministisellä, mutta ei deterministisellä pinoautomaatilla (todistus sivuutetaan).
- Kontekstiton kieli on *deterministinen*, jos se voidaan tunnistaa jollakin deterministisellä pinoautomaatilla. Voidaan osoittaa, että deterministisen kontekstittomien kielten luokka = aiemmin esiintynyt käytännön kannalta useimmiten riittävän rikas kieliluokka LR(k).
- Automaattiset jäsentäjängeneroijat kuten `yacc` ja `bison` muuntavat syötteenään saamansa (rajoitettua muotoa olevan) kontekstittoman kieliopin jäsennysalgoritmiksi. Tämä tapahtuu olennaisesti konstruoimalla kielioppia vastaavaa pinoautomaattia simuloivan ohjelman lähdekoodi.
- Esim. LL(1)-kieliopeilla tuotettavissa olevat kontekstittomat kielet ovat tunnistettavissa deterministisillä pinoautomaateilla, mutta myös monet monimutkaisemmat kielet.

- Kielten keksinäisiä suhteita kuvaa seuraava kuva:



3.6. Kontekstittomien kielten sovelluksia

- Jäsentäjät
  - Kuten olemme useaan otteeseen todenneet, ohjelmointikielten syntaksi voidaan määritellä kontekstittomien kielioppien avulla.
  - Kieliopista voidaan muodostaa kielen jäsentäjä eli kääntäjän osa, joka tutkii ohjelman rakenteen ja esittää sen jäsennyspuuna
  - Jäsentäjän voi joko kirjoittaa käsin (esim. rekursiiviset jäsentäjät LL(1)-kieliopeille) tai generoida automaattisesti esim. `yacc`- tai `bison`-työkaluilla.
  - Tyypillisesti jäsennyksen yhteydessä tehdään muutakin kuin generoidaan jäsennyspuu, esim. generoidaan koodia tms.
- Kontekstittomia kielioppeja voi käyttää myös muiden rakenteisten dokumenttien rakenteen määrittelyyn.
- Kontekstittomista kieliopeista on iloa myös luonnollisten kielten käsittelyssä ja yleensäkin sovelluksissa, joissa on tarpeen käsitellä merkkijonomuotoista, “rekursiivisesti rakenteista”, tietoa.

$*$ 3.7. 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]: Olkoon L kontekstiton kieli. Tällöin on olemassa sellainen $n \geq 1$, että mikä tahansa $z \in L$, $|z| \geq n$, voidaan jakaa osiin $z = uvwxy$ siten, että

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

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

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

Todistus:

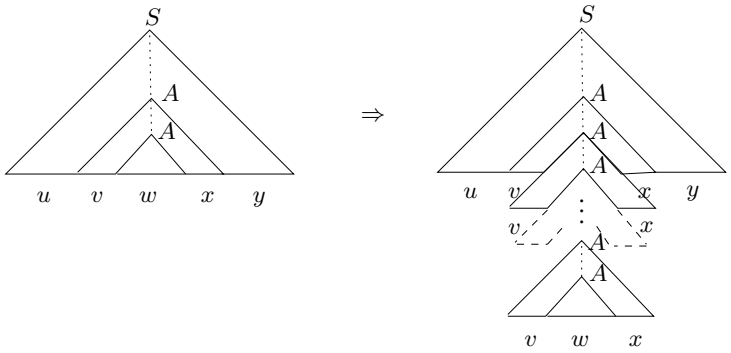
– Olkoon $G = (V, \Sigma, P, S)$ Chomskyn normaalimuotoinen kielioppi kielelle L. Tällöin missä tahansa G:n jäsennyspuussa, jonka korkeus (= pisimmän juuresta lehteen kulkevan polun pituus) on h, on enintään 2^h lehteä. Ts. minkä tahansa merkkijonon $z \in L$ jokaisessa jäsennyspuussa on polku, jonka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

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

– Luvun n valinnasta seuraa, että puun pisimmän polun pituus on vähintään $k + 1$. Tarkastellaan tämän polun "alinta" $k + 1$:n pituista osaa. Tällä polulla, jolla on $k + 1$ välikettä vastaavaa solmua, on jonkin välikkeen esiinnyttävä ainakin kahdesti (välikkeitä vain k kappaletta). Olkoon A jokin polun toistuva välike (ks. kuva 14).

– Merkkijono z voidaan nyt osittaa osiin $z = uvwxy$, missä w on A:n alimmasta ilmentymästä tuotettu osajono ja vwx seuraavaksi ylemmästä A:n ilmentymästä tuotettu osajono; osajonot saadaan johdosta

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



Kuva 14: Kontekstittoman kielen merkkijonon pumppaus.

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

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

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

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

– Koska kielioppi G on Chomskyn normaalimuodossa ja johdossa $A \Rightarrow^* vAx$ on vähintään yksi askel, on oltava $|vx| \geq 1$.

– Ylemmästä A:n ilmentymästä alkavan jäsennyspuun pisimmän polun pituus on enintään $k + 1 \Rightarrow$ vastaavan alipuun tuotokselle vwx pätee $|vwx| \leq 2^{k+1} = n$. $\square$

Esimerkki

- Väite: kieli $L = \{a^k b^k c^k \mid k \geq 0\}$ ei ole kontekstiton. Tod. Vastaväite: L on kontekstiton. Valitaan parametri n lemman mukaisesti ja tarkastellaan merkkijonoa $z = a^n b^n c^n \in L$. 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. □

4. Kontekstittomien kielten tuolla puolen ...

- Luvussa 3.7. osoitettiin kontekstittomien kielten pumppauslemmalla, että kieli $ABC = \{a^k b^k c^k \mid k \geq 0\}$ ei ole kontekstiton. Ei siis ole olemassa kontekstitonta kielioppia G, jolle $ABC = L(G)$.

- *Rajoittamattomissa kieliopeissa* produktiot ovat muotoa $\omega \to \omega' \in V^+ \times V^*$ — siis produktion vasemmallakin puolella saa olla mielivaltainen epätyhjä merkkijono.

  Esim: $Aa \to aab$ ja $CB \to A$ ovat kelvollisia produktioita rajoittamattomissa kieliopeissa.

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

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

  jos voidaan kirjoittaa $\gamma = \alpha\omega\beta$, $\gamma' = \alpha\omega'\beta$ ($\alpha, \beta, \omega, \omega' \in V^*$), ja kieliopissa G on produktio $\omega \to \omega'$.

  Esim: Jos G:ssä on yllämainitut esimerkkiproduktiot, niin $CBa \underset{G}{\Rightarrow} Aa \underset{G}{\Rightarrow} aab$.

- Relaatio $\underset{G}{\Rightarrow}^*$ ja kieliopin tuottama kieli määritellään samoin kuin kontekstittomien kielten tapauksessa.

- Rajoittamattomilla kieliopeilla tuotettavia kieliä sanotaan *rajoittamattomiksi kieliksi*.

Esimerkki

- Kieli $ABC = \{a^k b^k c^k \mid k \geq 0\}$ voidaan tuottaa seuraavalla rajoittamattomalla kieliopilla:

$$
\begin{array}{lcl}
S & \rightarrow & VT \mid \epsilon \\
T & \rightarrow & ABCT \mid ABC \\
BA & \rightarrow & AB \\
CB & \rightarrow & BC \\
CA & \rightarrow & AC \\
VA & \rightarrow & a \\
aA & \rightarrow & aa \\
aB & \rightarrow & ab \\
bB & \rightarrow & bb \\
bC & \rightarrow & bc \\
cC & \rightarrow & cc
\end{array}
$$

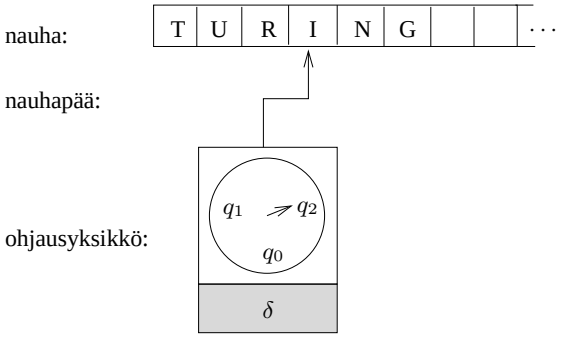
- Kielioppi tuottaa lauseen seuraavalla tavalla:
  1. Ensin johdetaan lähtösymbolista välikejono muotoa $V(ABC)^k$, missä $k \geq 1$.
  2. Sitten järjestetään välikkeet A, B ja C aakkosjärjestykseen, tuloksena $VA^k B^k C^k$,
  3. Lopuksi muutetaan välikkeet vastaaviksi päätemerkeiksi vasemmalta alkaen, tuloksena $a^k b^k c^k$.

  Esim. lauseen $aabbcc$ johto: $\underline{S} \Rightarrow V\underline{T} \Rightarrow VABC\underline{T} \Rightarrow VAB\underline{CA}BC \Rightarrow VA\underline{BA}CBC \Rightarrow VAAB\underline{CB}C \Rightarrow \underline{VA}ABBCC \Rightarrow \underline{aA}BBCC \Rightarrow a\underline{aB}BCC \Rightarrow aab\underline{BC}C \Rightarrow aabb\underline{CC} \Rightarrow aabbc\underline{C} \Rightarrow aabbcc$

Turingin koneet

- Rajoittamattomien kielten luokkaa vastaava automaatti on *Turingin kone*.
- Turingin kone (1935–36, Alan M. Turing 1912–54) koostuu
  - Äärellisestä ohjausyksiköstä (vrt. äärellinen automaatti).
  - Rajoittamattoman pitkästä työnauhasta, jota voi lukea ja jolle voi kirjoittaa merkki kerrallaan vasemmalle ja oikealle liikuteltavan nauhapään avulla.
- Aluksi kone on alkutilassa q_0, nauhalla on syötemerkkijono (loppu nauhasta on tyhjää) ja nauhapää osoittaa ensimmäistä syötemerkkiä.
- Kullakin laskenta-askeleella kone lukee nauhapään kohdalla olevan merkin. Koneen tila ja luettu merkki määräävät koneen siirtymäfunktion mukaisesti
  - koneen uuden tilan,
  - nauhalle kirjoitettavan merkin, ja
  - siirretäänkö nauhapäätä vasemmalle vai oikealle.
- Koneella on hyväksyvä lopputila q_{yes} ja hylkäävä lopputila q_{no}. Näihin saapuessaan kone lopettaa laskennan.

- Turingin kone eroaa äärellisestä automaatista, koska
  - työnauhalle voidaan kirjoittaa
  - työnauhalla voi liikkua sekä vasemmalle että oikealle
  - työnauha on rajoittamattoman pitkä
  - kone pysähtyy vasta kun lopputila on saavutettu.



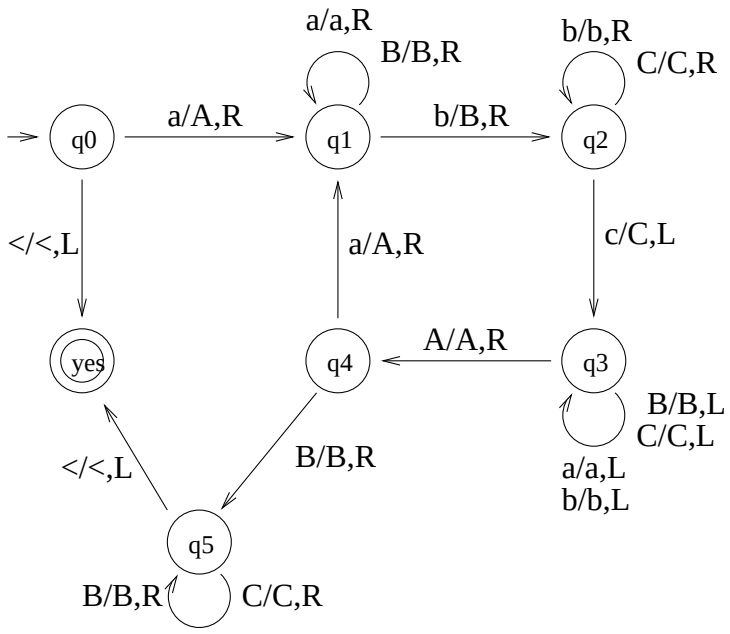
Kuva 15: Turingin kone.

- *Määritelmä: Turingin kone* on seitsikko $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{yes}, q_{no})$, missä
  - Q on koneen tilojen äärellinen joukko,
  - Σ on koneen syöteaakkosto,
  - Γ on koneen nauha-aakkosto, jolle $\Sigma \subseteq \Gamma$
  - $\delta : Q \times (\Gamma \cup \{>, <\}) \rightarrow Q \times \Gamma \times \{L, R\}$ on koneen siirtymäfunktio
  - q_0 on koneen alkutila
  - q_{yes} on koneen hyväksyvä lopputila
  - q_{no} on koneen hylkäävä lopputila

- Siirtymäfunktion tulkinta: $\delta(q, a) = (q', b, L)$ tarkoittaa, että ollessaan tilassa q ja lukiessaan nauhapään kohdalta syötemerkin a, Turingin kone siirtyy tilaan q', kirjottaa nauhapäänsä kohdalle merkin b ja siirtää nauhapäätä askeleen vasemmalle. Jos L:n tilalla on R, siirretään nauhapäätä vastaavasti oikealle.
- Symboli $>$ tarkoittaa syötenauhan vasenta ja $<$ syötenauhan käytössä olevan osan oikeaa reunaa ($<$ siirtyy askelen oikealle mikäli sen päälle kirjoitetaan).
- Turingin koneen tilanne on $(q, \alpha\underline{a}\beta)$, missä
  - q on koneen senhetkinen tila
  - a on nauhapään kohdalla oleva merkki
  - α nauhapään vasemmalla puolella oleva merkkijono, ja
  - β nauhapään oikealle puolella oleva merkkijono.

  Koneen alkutilanne syötteellä $x = x_1 \ldots x_n$ on siis $(q_0, \underline{x_1} x_2 \ldots x_n)$.
- Voidaan ajatella että x_1:n vasemmalla puolella on merkki $>$ ja x_n:n oikealla puolella on merkki $<$.
- Sivuutetaan formaali määritelmä sille, miten Turingin koneen tila johtaa seuraajatilaan. Määritelmän sisältö vastaa edellä esitettyä siirtymäfunktion tulkintaa.
- Jos kone pysähtyy syöttellä x tilaan q_{yes}, x kuuluu koneen M tunnistamaan kieleen $L(M)$. Jos taas kone päätyy tilaan q_{no} tai jää ikuiseen silmukkaan, niin $x \notin L(M)$.

- Huom: Turingin kone siis pysähtyy välittömästi lopputiloihin mennessään, mutta jos se ei koskaan päädy lopputilaan, laskenta jatkuu loputtomasti.
- *Lause:* Kieli on rajoittamaton, joss se voidaan tunnistaa Turingin koneella.
  Todistus: Sivuutetaan. $\square$
- Kielen $ABC = \{a^k b^k c^k \mid k \geq 0\}$ tunnistava Turingin kone:



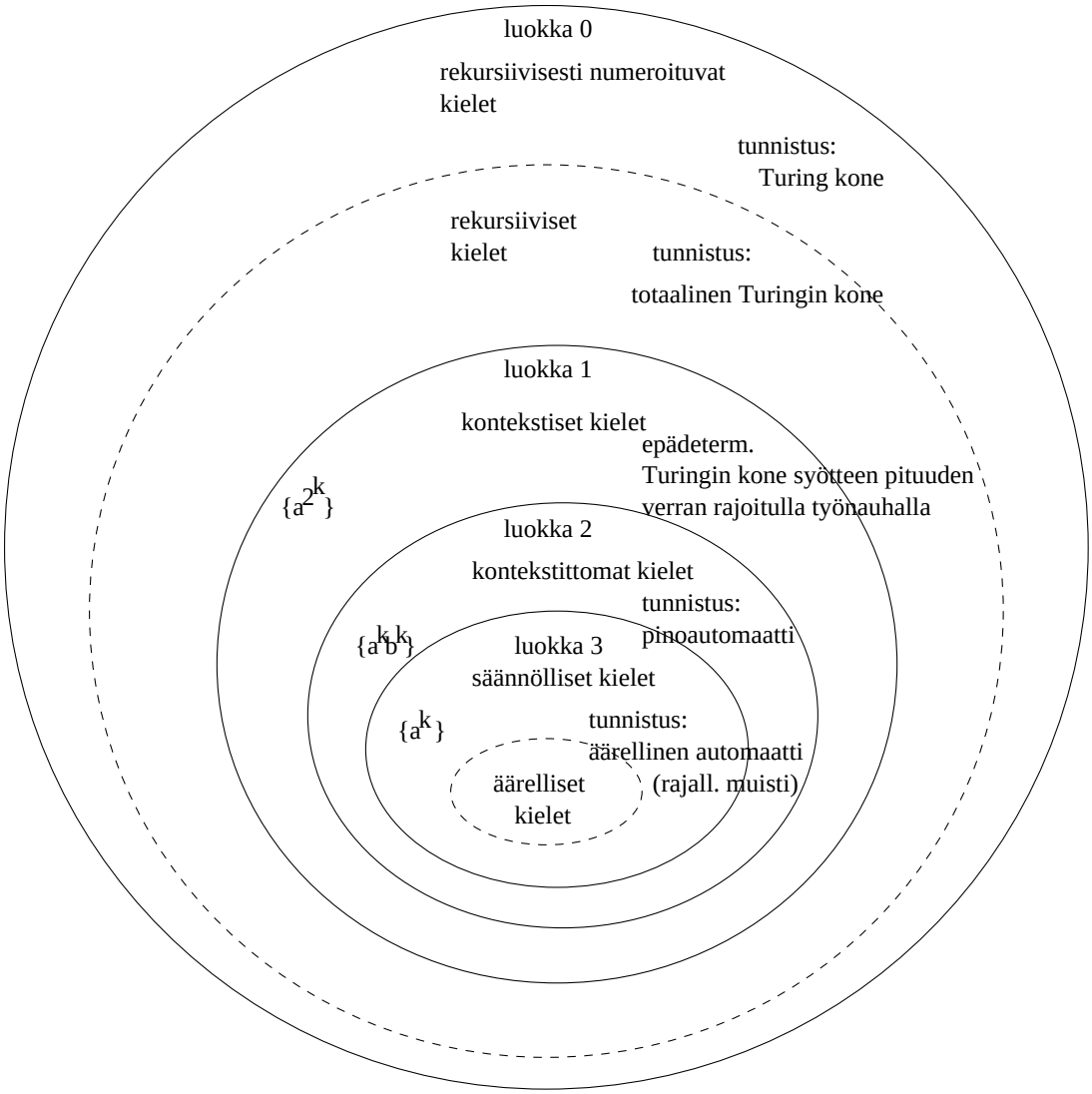
- Esim. syötteellä $aabbcc$ laskenta etenee seuraavasti

$$
\begin{aligned}
(q0, \underline{a}abbcc) &\vdash_M \\
(q1, A\underline{a}bbcc) &\vdash_M \\
(q1, Aa\underline{b}bcc) &\vdash_M \\
(q2, AaB\underline{b}cc) &\vdash_M \\
(q2, AaBb\underline{c}c) &\vdash_M \\
(q3, AaB\underline{b}Cc) &\vdash_M \\
(q3, Aa\underline{B}bCc) &\vdash_M \\
(q3, A\underline{a}BbCc) &\vdash_M \\
(q3, \underline{A}aBbCc) &\vdash_M \\
(q4, A\underline{a}BbCc) &\vdash_M \\
(q1, AA\underline{B}bCc) &\vdash_M \\
(q1, AAB\underline{b}Cc) &\vdash_M \\
(q2, AABB\underline{C}c) &\vdash_M \\
(q2, AABBC\underline{c}) &\vdash_M \\
(q3, AABB\underline{C}C) &\vdash_M \\
(q3, AAB\underline{B}CC) &\vdash_M \\
(q3, AA\underline{B}BCC) &\vdash_M \\
(q3, A\underline{A}BBCC) &\vdash_M
\end{aligned}
$$

$$
\begin{aligned}
(q4, AA\underline{B}BCC) &\underset{M}{\vdash} \\
(q5, AAB\underline{B}CC) &\underset{M}{\vdash} \\
(q5, AAB\underline{B}CC) &\underset{M}{\vdash} \\
(q5, AABB\underline{C}C) &\underset{M}{\vdash} \\
(q5, AABBC\underline{C}) &\underset{M}{\vdash} \\
(q5, AABBCC\underline{\leq}) &\underset{M}{\vdash} \\
(q_{yes}, AABBC\underline{C}) &
\end{aligned}
$$

Kielten luokittelua

- Turingin koneella tunnistettavia kieliä kutsutaan myös *rekursiiviseti lueteltaviksi kieliksi*.
- Jos Turingin kone M pysähtyy kaikilla syötteillään, sitä sanotaan *totaaliseksi*.
- Totaalisella Turingin koneella tunnistettavia kieliä kutsutaan *rekursiivisiksi kieliksi*.
- Päätösongelma on *osittain ratkeava* (engl. semidecidable), jos sitä vastaava formaali kieli on rekursiivisesti lueteltava ja *ratkeava* (engl. decidable), jos sitä vastaava formaali kieli on rekursiivinen.
- Ongelma, joka ei ole ratkeava, on *ratkeamaton* (engl. undecidable).
- Esimerkiksi pysähtymisongelma on ratkeamaton (mutta osittain ratkeava).



- Kieliopit ja niillä tuotettavat kielet ryhmitellään usein ns. *Chomskyn luokkiin* kuvan mukaisesti (Noam Chomsky 1928–)

  Kuvassa Chomskyn kielihierarkia täydennettynä rekursiivisten kielten luokalla. Huomaa että rekursiivisten kielten luokka on sama kuin rajoittamattomien kielten luokka. Kuvassa näkyy myös ns. kontekstisten kielten luokka.

Muita laskennan malleja

- Turingin koneesta on monta eri variaatiota, jotka tunnistavat samat kielet kuin tässä esitelty “perusmalli”.
- Tärkeä variaatio on *epädeterministinen Turingin kone*, jossa siirtymäfunktio on joukkoarvoinen (vrt. epädeterministiset äärelliset automaatit). Turingin koneeseen voidaan myös esim. lisätä nauhoja tai lukupäitä ilman että koneen ilmaisuvoima muuttuu.
- Moninauhaisessa koneessa yhden nauhoista voidaan ajatella olevan “tulostusnauha” — nauhalla laskennan pysähtyessä oleva merkkijono on koneen tuloste annetulla syötteellä. Turingin koneilla voidaan siis ratkaista päätösongelmien lisäksi yleisiä laskennallisia ongelmia, so. kuvauksia merkkijonoilta merkkijonoille.
- Myös monet muut formaalit laskennan mallit ovat osoittautuneet laskentavoimaltaan ekvivalenteiksi Turingin koneiden kanssa (mm. ohjelmointikielet).
- *Church-Turingin teesi* on seuraava hypoteesi: jokainen mekaanisesti ratkaistavissa oleva ongelma voidaan ratkaista Turingin koneella.

  Teesin mukaan kaikki riittävän ilmaisuvoimaiset (yhtä ilmaisuvoimaiset kuin Turingin kone) mekaanisen laskennan mallit ovat ekvivalentteja eli niillä voi tunnistaa täsmälleen samat kielet.

- Eri laskennan mallien tai ohjelmointikielten A ja B ekvivalenssi voidaan osoittaa kirjoittamalle näiden välille tulkit:
  - Kirjoitetaan/rakennetaan/... formalismissa A tulkki, jonka avulla A voi simuloida formalismin B mukaisten ohjelmien (tai automaattien tms) toimintaa.
  - Kirjoitetaan/rakennetaan/... formalismissa B tulkki, jonka avulla B voi simuloida formalismin A mukaisten ohjelmien (tai automaattien tms) toimintaa.

  Näin tulee näytettyä että mallissa A voi tehdä kaiken mitä mallissa B ja kääntäen.

Kurssin tuolla puolen...

- Luvun 4 asioita (ja muuta mielenkiintoista) käsitellään tarkemmin kursseilla *Laskennan teoria* (tietojenkäsittelytiede) ja *Laskettavuuden teoria* (matematiikka).