1.2. Matemaattisia peruskäsitteitä

Loogisia symboleita

- Olkoon P ja Q *propositioita* eli totuusarvoisia lauseita, jotka kuvaavat jonkin asiaintilan toteutumisen.
- 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. Totuusarvo riippuu nyt x:n arvosta. Jos x:n arvoa ei ole kiinnitetty, em. lauseet ovat *predikaatteja*.
- *Negaatio* $\neg P$: tosi, kun P on epätosi (ohjelmointikielissä esim. `not P` tai `!P`).
- *Disjunktio* $P \vee Q$: tosi, kun P tai Q on tosi (sisältää vaihtoehdon, että molemmat ovat tosia) (`P or Q` tai `P || Q`).
- *Konjunktio* $P \wedge Q$: tosi, kun sekä P että Q ovat tosia (`P and Q` tai `P && Q`).
- *Implikaatio* $P \Rightarrow Q$: "jos P, niin Q", tosi kun $\neg P \vee Q$ on tosi.
- *Ekvivalenssi* $P \Leftrightarrow Q$: "P jos ja vain jos Q", tosi kun $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$ on tosi.
- *universaalikvantifiointi* $\forall x : P(x)$ on tosi, kun x:stä riippuva predikaatti P on tosi kaikilla mahdollisilla x:n arvoilla.

- *eksistentiaalikvantifiointi* $\exists x : P(x)$ on tosi, kun P on tosi jollakin x:n arvolla.
- 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.
- Määritellään vielä loogiset vakiot F, joka on aina epätosi, sekä T, joka on aina tosi.

Joukot

- *Joukko* on kokoelma olioita eli joukon *alkioita*, esim. $A = \{a_1, a_2, ..., a_n\}$ tai $\Sigma = \{a, b, c, ..., z\}$. Merkitään $a_i \in A$ sitä, että a_i kuuluu joukkoon A.
- Jokainen alkio lasketaan vain kerran eikä järjestyksellä ole väliä, siis esim. $\{a, a, b\} = \{b, a\}$.
- *Tyhjää joukkoa* merkitään $\emptyset$.
- *Perusjoukko* on tarkastelun kohteeksi valittu kokoelma olioita, esimerkiksi luonnolliset luvut $\mathbb{N}$, reaaliluvut $\mathbb{R}$, aakkoset $\{a, \ldots, z\}$.
- merkinnällä $\{x \in \mathbb{N} \mid x > 5 \wedge x < 10\}$ tarkoitetaan joukkoa $\{6, 7, 8, 9\}$, eli kaikista kokonaisluvuista $x \in \mathbb{N}$ valitaan joukkoon ne, joille pätee "viivan" oikealla puolella oleva valintapredikaatti.

- Joukkojen väliset suhteet:
  - $A \subseteq B$: A on B:n *osajoukko*

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

  - $A \cup B$: 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*

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

  - $\overline{A} = E \setminus A$ on 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*.
  - $\mathcal{P}(A)$ on joukon A *potenssijoukko*

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

    eli kyseessä on joukon kaikista mahdollisista osajoukoista muodostuva joukko. Joskus potenssijoukkoa merkitään 2^A.
  - $|A|$ on joukon A alkioiden lukumäärä.

- 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 $A \cap B \subseteq B$
  - $A \cup B = \{4, 6, 7, 8\}$
  - $A \setminus B = \{7\}$
  - $\overline{B} = \{x \in \mathbb{N} \mid x \neq 4 \wedge x \neq 6 \wedge x \neq 8\}$
  - $A \times B = \{(6,4), (6,6), (6,8), (7,4), (7,6), (7,8), (8,4), (8,6), (8,8)\}$
  - Huom: $(6, 8)$ ja $(8, 6)$ ovat eri alkio. Jos olisi kyse joukoista niin $\{8, 6\} = \{6, 8\}$.
  - $\mathcal{P}(A) = \{\emptyset, \{6\}, \{7\}, \{8\}, \{6, 7\}, \{6, 8\}, \{7, 8\}, \{6, 7, 8\}\}$
  - Huom. alkio 6 **ei** kuulu joukkoon $\mathcal{P}(A)$, sen sijaan kylläkin $\{6\} \in \mathcal{P}(A)$, eli joukko, jonka ainoa alkio on 6, kuuluu 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 joukolta A joukolle B.
- Jos siis $a \in A$, niin $f(a) \in B$, eli f kuvaa joukon A alkion a joukon B alkioksi $f(a)$.
- Esim. määritellään $f : \mathbb{N} \to \mathbb{N}$ seuraavasti $f(x) = x + 1$. Nyt f on funktio, joka kuvaa kokonaisluvun x sen seuraajalle, eli 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.
  - Esimerkiksi $f(q_1, a_2) = q_4 \in Q$.
- Jos taas $f : Q \times \Sigma \to \mathcal{P}(Q)$, niin f kuvaa parin (q_i, a_i) alkioksi $f(q_i, a_i)$ joka kuuluu joukon Q potenssijoukkoon, eli toisin sanoen $f(q_i, a_i)$ on Q:n osajoukko.
  - Esimerkiksi $f(q_1, a_2) = \{q_2, q_4, q_5\} \subseteq Q$ tai $f(q_1, a_3) = \emptyset \subseteq Q$.

Todistusmenetelmiä

Halutaan todistaa, että oletuksista A seuraa väite B, eli että $A \Rightarrow B$ on tosi.

Todistuksen tekoon on olemassa useita erilaisia tekniikoita; mikä niistä kulloinkin sopii parhaiten, riippuu todistettavasta väitteestä.

- *Deduktiivinen todistus*: oletetaan A ja näytetään, että tästä seuraa B.
  - Esim. Olkoon $x \in \mathbb{N}$. Väite: x on pariton $\Rightarrow x^2$ on pariton.
    Tod.: Jos $x \in \mathbb{N}$ on pariton, niin $x = 2k + 1$ jollekin $k \in \mathbb{N}$. Silloin $x^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$, eli x^2 on pariton.
- *Kontrapositiivinen todistus*: näytetään, että olettamalla $\neg B$ seuraa $\neg A$.
  - Pätee, koska $(\neg B \Rightarrow \neg A) \Leftrightarrow$
    $(\neg\neg B \vee \neg A) \Leftrightarrow (\neg A \vee B) \Leftrightarrow (A \Rightarrow B)$
  - 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 olettamalla A ja $\neg B$ todeksi seuraa ristiriita.
  - 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: Olkoon V ääretön joukko ja $S \subseteq V$ äärellinen. Silloin $V \setminus S$ on ääretön.
    Tod.: Vastaväite: $V \setminus S$ on äärellinen. Tällöin $V = S \cup (V \setminus S)$ on kahden äärellisen joukon yhdiste ja siis äärellinen. Mutta V oli ääretön. Ristiriita. □
- *Induktio*
  - Voidaan soveltaa, kun halutaan todistaa, että jokin väite $P(n)$ pätee kaikille luonnollisille luvuille $n \in \mathbb{N}$. Väite todistetaan kahdessa osassa:
    1. Perustapaus $n = 0$: Todistetaan, että väite $P(0)$ pätee
    2. Induktioaskel: Tehdään induktio-oletus, että pätee $P(n)$, $n \geq 0$. Todistetaan, että tällöin pätee myös $P(n+1)$.

    Kohdista 1 ja 2 seuraa, että väite pätee kaikilla $n \geq 0$. Siis $P(0)$ pätee, josta saadaan induktioaskeleella $P(1)$, tästä puolestaan $P(2)$ jne.
  - Esimerkki induktiotodistuksesta. Väite:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \text{ kaikilla } n \geq 0$$

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 $n+1$: $\sum_{i=0}^{n+1} i = \sum_{i=0}^{n} i + (n+1) =^{*} \frac{n(n+1)}{2} + \frac{2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$, missä $*$ merkityssä yhtäsuuruudessa käytettiin induktio-oletusta.
   □

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

Numeroituvuus

- Miten voidaan vertailla kahden 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 X:n alkiot voidaan siis järjestää ja niille voidaan antaa yksikäsitteinen järjestysnumero: $X = \{x_0, x_1, x_2, \ldots\}$, missä $x_0 = f(0), x_1 = f(1), \ldots$.
- Jos X ei ole numeroituva, se on *ylinumeroituva*.
- Jos X ja Y ovat äärettömiä joukkoja siten, että X on numeroituva ja Y on ylinumeroituva, niin joukossa Y on "enemmän" alkioita, sillä joukko $\mathbb{N}$ riittää numeroimaan X:n alkiot mutta ei Y:n alkioita.
- Parittomien luonnollisten lukujen joukko $\{1, 3, 5, \ldots\}$ on numeroituva, sillä $f(n) = 2n+1$ on bijektio, joka kuvaa luonnolliset luvut kyseiselle joukolle. Jokaista lukua joukossa $\mathbb{N}$ siis vastaa luku joukossa $\{1, 3, 5, \ldots\}$, joten parittomia lukuja on tässä mielessä "yhtä monta" kuin kokonaislukuja.
- Esim. reaalilukujen joukko $\mathbb{R}$ on ylinumeroituva, eli "suurempi" kuin $\mathbb{N}$.

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 kokonaislukuparit
  - annettuun tapaukseen liittyvä vastaus: lukuparin tulo
  - ongelman ratkaisu: mikä tahansa yleinen kertolaskualgoritmi
- Kunkin yksittäisen tapauksen ja sen vastauksen oltava *äärellisesti esitettäviä*
- Laskennallinen ongelma = *kuvaus äärellisesti esitettävien tapausten joukosta äärellisesti esitettävien vastausten joukkoon*

Äärellinen esitys

- Kaikki tietokoneen käsittelemä tieto on viime kädessä voitava koodata bittijonoiksi.
- Luontevaa sallia koodaukseen käytettävän myös muita merkkejä kuin 0 ja 1, sillä 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* $\{$A,B,...,Z$\}$. Aakkosto merkitään yleensä isolla kreikkalaisella kirjaimella ja merkit pienillä latinalaisilla kirjaimilla. Aakkoston $\Sigma = \{a_1, \ldots, a_n\}$ koko on $|\Sigma| = n$.
- *Merkkijono* on äärellinen järjestetty jono jonkin aakkoston merkkejä. Esim. "01001" ja "000" ovat binääriaakkoston merkkijonoja, ja "LTE" ja "XYZZY" ovat latinalaisen aakkoston merkkijonoja.
- Merkkijonon x *pituus* on siihen sisältyvien merkkien määrä, merkitään $|x|$. Esim.

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

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

- Merkkijonojen välinen perusoperaatio on *katenaatio* eli jonojen peräkkäin kirjoittaminen. (Katenaation operaatiomerkkinä käytetään joskus selkeyden lisäämiseksi symbolia $\widehat{\ }$.) Esimerkkejä:
  - KALA$\widehat{\ }$KUKKO = KALAKUKKO
  - jos $x = 00$ ja $y = 11$, niin $xy = 0011$ ja $yx = 1100$
  - kaikilla x on $x\epsilon = \epsilon x = x$
  - kaikilla x, y on $|xy| = |x| + |y|$
- Merkin a toisto k kertaa merkitään a^k ja vastaavasti merkkijonon x toisto k kertaa x^k.
- Merkkijonon $x = x_1x_2 \ldots x_{k-1}x_k$ käänteinen merkkijono on $x^R = x_kx_{k-1} \ldots x_2x_1$.
- Merkitään k:n pituisia aakkoston Σ merkkijonoja

$$\Sigma^k = \{x_1x_2 \ldots x_k \,|\, \forall i = 1, \ldots, k : x_i \in \Sigma\}$$

  ja korkeintaan k:n pituisia aakkoston Σ merkkijonoja

$$\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i.$$

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

Formaali kieli

- Aakkoston Σ *(formaali) kieli* (engl. *formal language*) L on merkkijonojoukko $L \subseteq \Sigma^*$.
- Aakkoston Σ formaali kieli on siis joukko, joka koostuu Σ:n merkkijonoista.
- Esimerkkejä:
  - Aakkoston $\Sigma = \{a, b\}$ kieliä ovat esim. seuraavat:

    $$L_1 = \{\epsilon, aa, aba, abba, abbba\} \text{ ja } L_2 = \{ab^i | i \in \mathbb{N}\}.$$

  - Olk. $\Sigma = \{0, 1, \ldots, 9\}$; eräs Σ:n kieli on

    $$L_3 = \{w \in \Sigma^* \mid w \text{ on alkuluku}\} = \{2, 3, 5, 7, 11, 13 \ldots\}.$$

  - Olk. $\Sigma = \{a, \ldots, z, 0, 1, \ldots\}$, eli annetaan Σ:n sisältää kaikki ascii-merkit. Σ:n kieliä ovat nyt 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$\}$, tai
    - $L_6 = \{w \in \Sigma^* \mid$ merkkijono w esittää suomen kielen verbin perusmuodossa$\}$.

- Edellä määritelty kieli L_5 siis sisältää kaikki syntaktisesti oikeat olpe03-kieliset ohjelmat.
- Nyt annetun ohjelmalistauksen w' (missä w' siis on merkkijono, joka sisältää ohjelman, kuten sivun 3 "$x = read; i := 0; y := 1; \ldots$") syntaksitarkistus voisi siis tapahtua testaamalla päteekö $w' \in L_5$.
- Hyviä kysymyksiä ovat
  - miten formaaleja kieliä pystyttäisiin määrittelemään käytännöllisellä tavalla (sovelluksena ohjelmointikielen syntaksin määrittely) ja
  - miten testata (tehokkaasti) kuuluuko annettu merkkijono formaaliin kieleen (sovelluksena ohjelmalistauksen syntaksitarkastus).
- Kurssin edetessä opitaan 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 aakkoston $\Sigma = \{0,1,2,\ldots,9\}$ kuvauksena

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

- Päätösongelma "onko annettu merkkijono olpe03:n syntaksin mukainen ohjelma" voidaan esittää sivulla 19 määritellyn aakkoston $\Sigma = \{a,\ldots,z,0,1,\ldots\}$ kuvauksena:

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

- 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*) esitetään merkkijonojoukkoon liittyvänä päätösongelmana f_A.
  - $f_A(x)$ on 1, jos ja vain jos x kuuluu tarkasteltavaan kieleen.

Laskennallisten ongelmien ratkeavuus

- Sanotaan, että ohjelma P *ratkaisee* laskentaongelman f, jos kullakin syötteellä x ohjelma P laskee ja tulostaa arvon $f(x)$.
- Voidaanko kaikki mahdolliset laskentaongelmat ratkaista ohjelmilla?
- Osoittautuu, että ei voida. Tämä nähdään siitä, että kaikkien merkkijonojen (minkä tahansa ohjelmointikielen mahdollisten ohjelmien) joukko on numeroituva, mutta kaikkien päätösongelmien joukko on ylinumeroituva.
- Laskentaongelmia on siis 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: Selvästi Σ^* on ääretön. 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 ($= \epsilon$), sitten 1:n mittaiset ($= a_1, a_2, \ldots, a_n$), sitten 2:n mittaiset jne.;
  2. kunkin pituusryhmän sisällä merkkijonot luetellaan aakkosjärjestyksessä.

  Jokaiseen luonnolliseen lukuun n voidaan siis liittää yksikäsitteinen Σ^*:n merkkijono ja päinvastoin. Tästä seuraa, että Σ^* 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: (sovelletaan *diagonalisointia*)
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	$\not{\emptyset}^{1}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

- Eli 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 $f_2, f_3, \ldots$. Oletus, että $\mathcal{F}$ olisi numeroituva, johti ristiriitaan ja 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*.
- Myös monet mielenkiintoiset ja käytännölliset ongelmat ovat ratkeamattomia.
- 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 $\mathbf{1}$, jos merkkijonoparametrin p esittämä funktio pysähtyy syötteellä w ja $\mathbf{0}$ muuten. Kirjoitetaan tämän perusteella toinen C-funktio $\hat{H}$:

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

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

  $\hat{H}(\hat{h})$ pysähtyy $\Leftrightarrow$ $H(\hat{h},\hat{h}) = \mathbf{0}$ $\Leftrightarrow$ $\hat{H}(\hat{h})$ 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
  - miten formaaleja kieliä pystytään määrittelemään käytännöllisellä tavalla ja
  - miten voidaan ohjelmallisesti testata kuuluuko annettu merkkijono määriteltyyn formaaliin kieleen.
- Määritellään eräs aakkoston Σ_{ascii} kieli seuraavasti:

  $HALT = \{P, x \mid P$ on ohjelmalistaus ja x ohjelman sellainen syöte, jolla ohjelman suoritus pysähtyy$\}$
- Edellisessä luvussa todistettiin, että tämä ns. pysähtymisongelma ei ole ratkeava. Ei siis ole mahdollista tehdä tietokoneohjelmaa, joka annetusta syötteestä (P, x) pystyy päättelemään, kuuluuko se kieleen $HALT$.
- Havaitaan siis, että jos sallitaan yo. kaltaiset määritelmät, on helppoa määritellä kieli, jota ei voida tunnistaa tietokoneella.
- Seuraavassa tarkastellaan rajoitettuja tapoja formaalien kielten määrittelyyn, joiden avulla määritellyt kielet ovat vielä *ratkeavia*, ts. annetun merkkijonon kuuluvuus kieleen on ratkaistavissa algoritmisesti.
- Tarkastellaan kahta rinnakkaista formalismia, *äärellisiä automaatteja* sekä *säännöllisiä lausekkeita*.

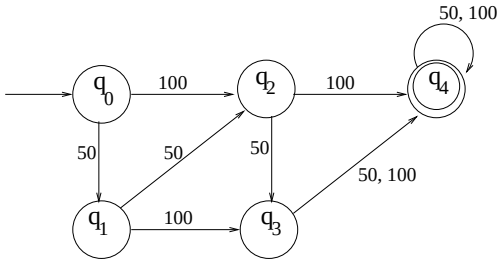
2.1. Äärelliset automaatit

- Ongelma: Kahviautomaatti, joka ei anna vaihtorahaa, hyväksyy vain 50 sentin ja yhden euron kolikoita ja minimimaksu on 2 euroa. Millaisia syötejonoja kahviautomaatti hyväksyy?
- Kelvollisia syötteitä ovat mm. seuraavankaltaiset kolikkojonot (yksikkönä snt):
  - 50, 50, 50, 50
  - 100, 100
  - 50, 100, 100
  - 100, 50, 50, 100
- Aakkostomme on nyt $\Sigma = \{50, 100\}$, ja automaatin hyväksymät kolikkojonot voidaan kuvata seuraavana kielenä:

$$\{a_1 \dots a_n \in \Sigma^* \mid a_1 + \dots + a_n \geq 200\},$$

  missä + tarkoittaa kolikkojen senttiarvojen yhteenlaskua.
- Kahviautomaatin toiminta voidaan kuvata *äärellisellä automaatilla*.
- Automaatin syötteitä ovat 50 sentin ja 1 euron kolikot, ja automaatti hyväksyy syötejonon, jos siihen sisältyvien rahojen arvo on yhteensä vähintään 2 euroa.

- Äärellinen automaatti voidaan esittää tilasiirtymäkaaviona

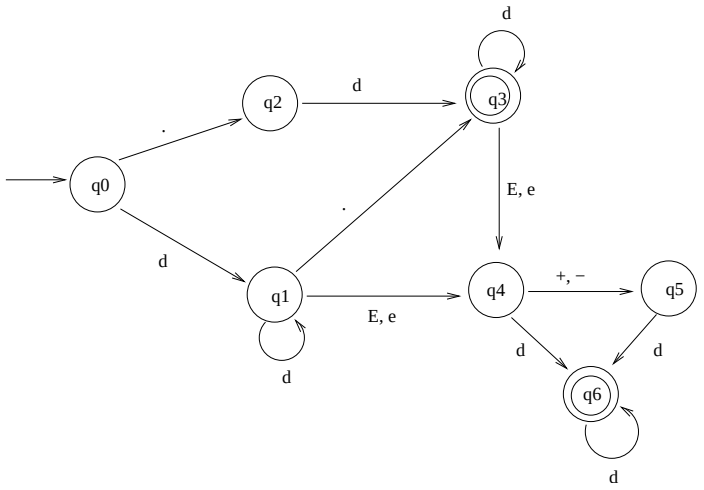


- tai tilasiirtymätaulukkona.

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

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

- Esim. C-kielen etumerkittömät liukuluvut (float, double, long double) määritellään seuraavasti:
  - (kokonaisosa).(desimaaliosa) (e tai E) [+ tai −] (eksponetti) [suffiksi]
  - kokonaisosa ja desimaaliosa koostuvat digiteistä
  - joko kokonaisosa tai desimaaliosa voi puuttua (mutta eivät molemmat)
  - joko (i) desimaalipiste tai (ii) (e tai E) ja eksponentti voivat puuttua (mutta eivät molemmat)
  - suffiksi: F tai f: float, L tai l: long double, muuten double
- Liukuluvun tunnistava automaatti:



- Tilasiirtymätaulukkona:

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

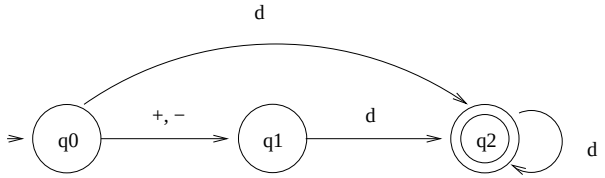
- Tässä d $= \{0, 1, \ldots, 9\}$. Taulukon puuttuvissa kohdissa on virhetila "Error", jota käytännössä jätetään merkitsemättä selkeyden takia

- Ohjelmana:

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

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


- Äärellisen automaatin pohjalta laadittuun ohjelmaan voidaan liittää myös semanttisia toimintoja.
- Sen lisäksi, että tarkistetaan kuuluuko jokin merkkijono kieleen, eli onko se syntaktisesti kielen mukainen, lasketaan merkkijonolle myös jokin arvo.
- Se, mitkä ovat mielekkäitä arvoja tietylle merkkijonolle, riippuu tarkasteltavasta kielestä. Esimerkiksi kahviautomaattikielessä merkkijonon arvo voisi olla palautettavan rahamäärän suuruus.
  - Siis kolikkojonon 50,50,50,100 voidaan todeta kuuluvan kieleen ja sen arvo on ym. tavalla laskettuna 50.
- Toinen esimerkki: Etumerkillisen kokonaisluvun tunnistaminen

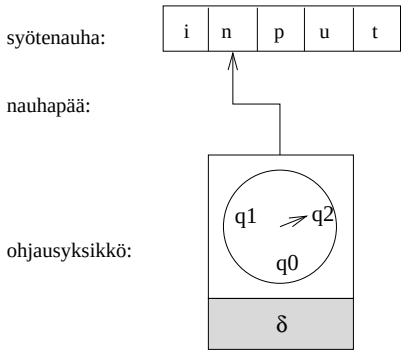


- Vastaava ohjelma, joka lisäksi laskee luetun luvun arvon:

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

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


Äärellisen automaatin formaali määrittely



- Äärellinen automaatti M
  - äärellistilainen *ohjausyksikkö*, jonka toimintaa säätelee automaatin *siirtymäfunktio* δ
  - merkkipaikkoihin jaettu *syötenauha*
  - *nauhapää*, joka kullakin hetkellä osoittaa yhtä syötenauhan merkkiä

- Automaatin "toiminta":
  - Automaatti käynnistetään erityisessä *alkutilassa* q_0 siten, että tarkasteltava syöte on kirjoitettuna syötenauhalle ja nauhapää osoittaa sen ensimmäistä merkkiä.
  - Yhdessä toiminta-askelessa automaatti lukee nauhapään kohdalla olevan syötemerkin, päättää ohjausyksikön tilan ja luetun merkin perusteella ohjausyksikön uudesta tilasta (soveltaa siis siirtymäfunktiota), ja siirtää nauhapäätä yhden merkin eteenpäin.
  - Automaatti pysähtyy, kun viimeinen syötemerkki on käsitelty. Jos ohjausyksikön tila tällöin kuuluu erityiseen *(hyväksyvien) lopputilojen* joukkoon, automaatti *hyväksyy* syötteen, muuten *hylkää* sen.
  - Automaatin *tunnistama kieli* on sen hyväksymien merkkijonojen joukko.

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

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

missä

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

- Esim. reaalilukuautomaatin formaali esitys:

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

missä δ on kuten aiemmin taulukossa; esim.

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

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

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

Ts. tilanne (q, w) johtaa suoraan tilanteeseen (q', w'), jos siitä tilanteesta, jossa automaatti on tilassa q ja syötteen käsittelemätön osa w on $aa_1 \ldots a_n$, automaatti siirtyy yhdellä laskenta-askeleella tilaan q' ja $w' = a_1 \ldots a_n$ jää syötteen käsittelemättömäksi osaksi.

- Tilanne (q, w) *johtaa tilanteeseen* (q', w') eli tilanne (q', w') on tilanteen (q, w) *seuraaja*, merk.

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

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

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

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

Ts. tilanne (q, w) johtaa tilanteeseen (q', w'), jos suorittamalla yksi tai useampia laskenta-askelia päästään tilanteesta (q, w) tilanteeseen (q', w') (tai jos $q = q'$ ja $w = w'$).

- Automaatti M *hyväksyy* merkkijonon $x \in \Sigma^*$, jos on voimassa

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

muuten M *hylkää* x:n. Ts. automaatti hyväksyy x:n, jos sen alkutilanne syötteellä x johtaa johonkin hyväksyvään lopputilanteeseen.

- *Määritelmä:* Automaatin M *tunnistama kieli*

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

Automaatin tunnistama kieli sisältää siis kaikki merkkijonot, joilla päästään alkutilasta hyväksyvään lopputilaan.

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

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

Koska $q_6 \in F = \{q_3, q_6\}$, on siis `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.
- Automaatti, jossa on enemmän tiloja kuin ekvivalentissa minimaalisessa automaatissa on *redundantti*.
- Automaatteja muodostavat algoritmit eivät aina tuota minimaalista automaattia.
- Ylimääräisten tilojen tallettaminen on turhaa.
- Minimaalisen automaatin algoritminen käsittely on tehokkaampaa kuin redundantin automaatin.
- Minimaalisesta automaatista nähdään helpoimmin, mikä on sen (ja tietysti sen kanssa ekvivalenttien redundanttien automaattien) tunnistama kieli.

Äärellisen automaatin minimointialgoritmi

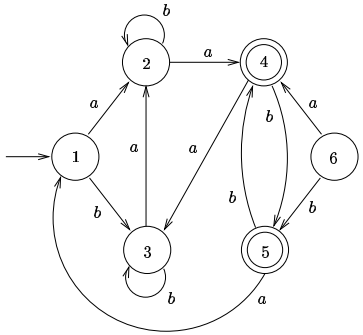
- Syöte: Äärellinen automaatti $M = (Q, \Sigma, \delta, q_0, F)$.
  1. Poista M:stä kaikki tilat, joita ei voida saavuttaa tilasta q_0 millään syötemerkkijonolla.
  2. Osita M:n jäljelle jääneet tilat kahteen luokkaan: ei-lopputiloihin ja lopputiloihin.
  3. **while not** (δ yhteensopiva luokkajaon kanssa) {
     jaa luokkien sisällä eri tavalla käyttäytyvät tilat eri luokkiin;
     }
  4. return $\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \widehat{F})$, missä
     - $\hat{Q}$=M:n tilaluokat,
     - $\hat{\delta}$=luokkien välinen siirtymäfunktio,
     - $\hat{q}_0$=M:n alkutilan luokka ja
     - $\widehat{F}$=M:n lopputilojen luokat.
- Lopputulos
  - M:n kanssa ekvivalentti äärellinen automaatti $\widehat{M}$, jossa on minimimäärä tiloja.
  - $\widehat{M}$ on tilojen nimeämistä vaille yksikäsitteinen.

- Huom: Tiloja on alun perin äärellinen määrä ja joka askeleessa 3 (paitsi viimeisessä) ositetaan vähintään yksi tilaluokka, joten algoritmin suoritus päättyy aina.
- Todistus siitä, että algoritmin tuottama automaatti $\widehat{M}$ on minimiautomaatti ja yksikäsitteinen sivuutetaan (Orposen moniste s. 17–18).

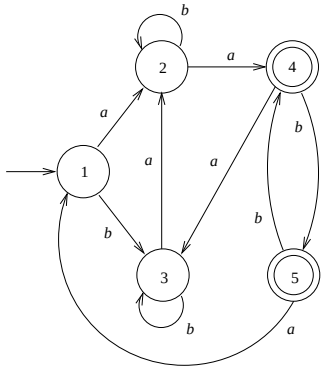
Esimerkki

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

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



- Askel 1 (turhien tilojen poisto):



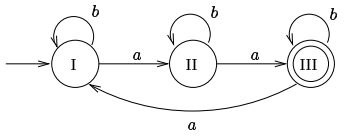
- Askel 2:
  - Osita M:n jäljelle jääneet tilat kahteen luokkaan: lopputiloihin ja muihin tiloihin.

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

  - Nyt osituksesta voidaan muodostaa automaatti, jossa
    - kutakin luokkaa vastaa yksi tila, ja
    - kustakin tilasta on kaikki erilliset siirtymät, jotka luokkaan kuuluvilla tiloilla on.
  - Tila on *epädeterministinen*, jos siitä voidaan siirtyä jollain merkillä useampaan kuin yhteen tilaan.
  - Esimerkissä tila I on epädeterministinen, koska merkillä a voidaan siirtyä tilaan I tai tilaan II.

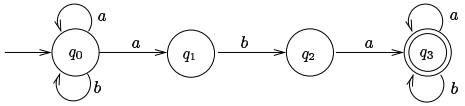
- Askel 3:
  - Jos $\widehat{M}$:ssä ei ole epädeterministisiä tiloja, niin algoritmi palauttaa $\widehat{M}$:n (askel 4) ja sen suoritus päättyy.
  - Muutoin hienonna kutakin $\widehat{M}$:n epädeterminististä tilaa vastaavan luokan ositusta edelleen:
    - Jaa sen sisällä alkuperäiset tilat eri luokkiin s.e. kustakin luokasta on vain samanlaisia siirtymiä.
    - Suorita askel 3 uudestaan.
  - Esimerkissämme jaetaan tila I kahtia.
  - Tämän jälkeen ei ole enää epädeterminisiä tiloja, joten palautetaan tuloksena saatu automaatti 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

- Esimerkiksi alimerkkijonon *aba* etsiminen aakkoston $\{a, b\}$ merkkijonoista on luontevasti kuvattavissa epädeterministisellä automaatilla. Epädeterminismi siis helpottaa automaatin konstruointia.
- Todistamme tässä luvussa, että deterministiset ja epädeterministiset automaatit tunnistavat täsmälleen samat kielet.
- Epädeterministisen automaatin siirtymäfunktio liittää vanhan tilan ja syötemerkin pariin (q, x) *joukon* mahdollisia seuraavia tiloja.
- Epädeterministinen automaatti hyväksyy merkkijonon jos *jokin* mahdollisten tilojen jono johtaa lopputilaan. Jos yhtään tällaista jonoa ei ole, niin epädeterministinen automaatti hylkää syötemerkkijonon.



- Esim. kuvan automaatti hyväksyy syötejonon *abbaba*, koska se voidaan käsitellä seuraavasti:

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

- Toisaalta voidaan myös päätyä hylkäävään tilaan:

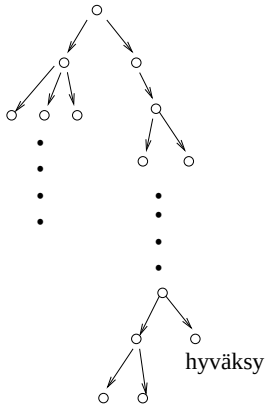
$$(q_0, abbaba) \vdash (q_0, bbaba) \vdash (q_0, baba)$$
$$\vdash (q_0, aba) \vdash (q_0, ba) \vdash (q_0, a) \vdash (q_0, \epsilon)$$

- Epädeterministisen automaatin voidaan ajatella suorittavan kaikki johdot rinnakkain.

Deterministinen laskenta

hyväksy tai hylkää

Epädeterministinen laskenta



hylkää

- *Määritelmä:* Epädeterministinen äärellinen automaatti (engl. *nondeterministic finite automaton*) on viisikko $M = (Q, \Sigma, \delta, q_0, F)$, missä
  - Q on äärellinen tilojen joukko,
  - Σ on syöteaakkosto,
  - $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ on (joukkoarvoinen) siirtymäfunktio,
  - $q_0 \in Q$ on alkutila ja
  - $F \subseteq Q$ lopputilojen joukko.
- Kuvan automaatin siirtymäfunktio:

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

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

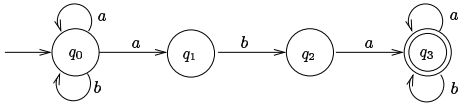
- Deterministiset automaatit ovat epädeterminististen erikoistapaus $\Rightarrow$ kaikki edellisillä tunnistettavat kielet ovat tunnistettavissa myös jälkimmäisillä.
- Mutta myös kääntäen: *deterministiset ja epädeterministiset äärelliset automaatit ovat yhtä vahvoja.*

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

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

- Esimerkki:



- 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
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$

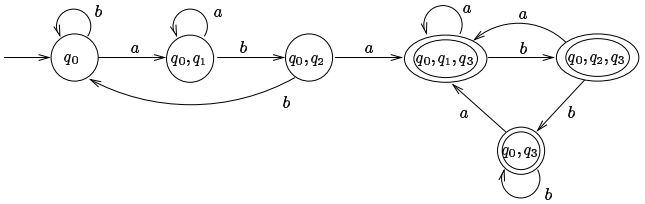
- Aina kun uusi tila löytyy, lisätään sitä varten uusi rivi taulukkoon:

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

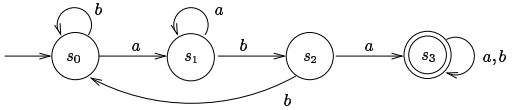
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$

...

- ja jatketaan, kunnes uusia tiloja ei enää synny.
- Esimerkin automaatti loppuun asti determinisoituna:



- ja minimoituna:



- *Lause:* Olk. $A = L(M)$ jonkin epädeterminististen äärellisen automaatin M tunnistama kieli. Tällöin on olemassa deterministinen automaatti $\widehat{M}$, jolla $L(\widehat{M}) = A$.

*Todistus: Olk. $A = L(M)$, $M = (Q, \Sigma, \delta, q_0, F)$. Laaditaan edellisen sivun määrittelyn mukainen determinstinen automaatti $\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F})$. Tarkastetaan, että $L(\widehat{M}) = L(M)$. Kielten ekvivalenssi seuraa, kun todistetaan kaikilla $x \in \Sigma^*$ ja $q \in Q$:

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

Todistus induktiolla merkkijonon x pituuden suhteen:

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

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

□

2.4. Säännölliset lausekkeet ja kielet

- Ohjelmointikielten muuttujilla, vakioilla jne. on tietty muoto, jonka määrää ohjelmointikielen *syntaksin* (”oikeankirjoitusopin”) mukaisia. Esim. 0.123 on luku, mutta 0.1.2.3 ei ole. Miten voidaan määritellä oikeanmuotoiset merkkijonot?
- Ongelma voidaan ratkaista äärellisten automaattien sijaan myös *säännöllisten lausekkeiden* avulla.
- UNIXissa 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 nimenomaan säännöllisiä lausekkeita, joihin tutustumme tässä luvussa, ja onkin lyhenne sanoista ”Global search for Regular Expression and Print”.
- Esim. Hyväksy merkkijonot, joissa esiintyy sana ”teoria”. Ts. merkkijonot ovat muotoa

  [0 tai useampia kirjaimia]*teoria*[0 tai useampia kirjaimia]

- Esim. Hyväksy merkkijonot, joissa ei esiinny sanaa ”käytäntö”.
- Esim. Etsi tekstitiedostosta osoitteita, jotka ovat muotoa
  [$x$katu tai $x$tie][*numero*] [mahd. rapun kirjain]
  [mahd. asunnonnumero][postinumero][kunnan nimi]

- Ennen kuin esittelemme säännöllisten lausekkeiden käsitteen, tarkastellaan muutamia kieliin liittyviä apukäsitteitä.
- Aakkoston Σ formaali kieli siis on joukko, joka koostuu Σ:n merkkijonoista.
- Määritellään avuksi kolme operaatiota kielten yhdistämiseen: Olkoot A ja B aakkoston Σ kieliä. Tällöin
  - A:n ja B:n *yhdiste* on kieli

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

    Eli merkkijono kuuluu yhdistekieleen, jos se kuuluu vähintään toiseen kielistä A tai B.
  - A:n ja B:n *tulo* on kieli

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

    Eli merkkijono kuuluu kielten tuloon, jos se on saatu yhdistämällä jokin kielen A ja jokin kielen B merkkijono.
  - A:n *potenssit* A^k, $k \geq 0$, määritellään iteratiivisesti:

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

    Siis $A^1 = A$,
    $A^2 = AA^1 = AA$,
    $A^3 = AA^2 = AAA$,
    $A^4 = AA^3 = AAAA$, ...

    Eli merkkijono kuuluu kieleen A^k, jos se on saatu yhdistämällä k kielen A merkkijonoa.
  - A:n *sulkeuma* on kieli

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

    Eli merkkijono kuuluu kieleen A^*, jos se on tyhjä merkkijono, A:n merkkijono, tai saatu yhdistämällä jokin määrä kielen A merkkijonoja.

Esimerkki. 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 $AB \neq BA$!
- $A^0 = \{\epsilon\}$, $A^1 = A = \{00, 01\}$,
  $A^2 = AA = \{0000, 0001, 0100, 0101\}$,
  $A^3 = AA^2 = \{000000, 000001, 000100, 000101, 010000, 010001, 010100, 010101\}$,
  $A^* = \{\epsilon, 00, 01, 0000, 0001, 0100, 0101, 000000, 000001, 000100, 000101, 010000, 010001, 010100, 010101, \dots\}$
- $B^0 = \{\epsilon\}$, $B^1 = B = \{\epsilon, 1, 010\}$,
  $B^2 = BB = \{\epsilon, 1, 010, 11, 1010, 0101, 010010\}$,
  $B^3 = BBB$ ...

- Nyt olemme valmiit määrittelemään säännöllisen lausekkeen. Annetaan määritelmä kahdessa vaiheessa.
  - Ensin annetaan induktiivinen määritelmä säännöllisten lausekkeiden syntaksille.
  - Tämän jälkeen määritellään, minkä kielen kukin säännöllinen lauseke kuvaa. Määritellään siis säännöllisten lausekkeiden semantiikka (eli merkitys).
- *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;
  (iv) muita Σ:n säännöllisiä lausekkeita ei ole.
- Esimerkiksi seuraavat ovat aakkoston $\{a, b\}$ säännöllisiä lausekkeita:
$$r_1 = ((ab)b), \quad r_2 = (ab)^*,$$
$$r_3 = (ab^*), \quad r_4 = (a(b \cup (bb)))^*.$$
- Perustelu sille, miksi esim. r_1 on syntaktisesti oikea säännöllinen lauseke on seuraava. Koska $a, b \in \Sigma$, ne ovat itsessään säännöllisiä lausekkeita säännön (ii) nojalla. Säännön (iii) nojalla siis myös (ab) on säännöllinen lauseke. Edelleen saman säännön nojalla myös $r_1 = ((ab)b)$ on säännöllinen lauseke.

- Säännöllisen lausekkeen kuvaaman kielen määritelmä on myös induktiivinen:
- Kukin Σ:n säännöllinen lauseke r *kuvaa* kielen $L(r)$:
  (i) $L(\emptyset) = \emptyset$;
  (ii) $L(\epsilon) = \{\epsilon\}$;
  (iii) $L(a) = \{a\}$ kaikilla $a \in \Sigma$;
  (iv) $L((r \cup s)) = L(r) \cup L(s)$;
  (v) $L((rs)) = L(r)L(s)$;
  (vi) $L(r^*) = (L(r))^*$
- Tarkastellaan esimerkkinä lausekkeiden r_1, r_2, r_3 ja r_4 kuvaamien kielten määrittelemistä.
  - Kohdan (iii) perusteella $L(a) = \{a\}$ ja $L(b) = \{b\}$
  - siispä kohdan (v) perusteella $L(ab) = L(a)L(b) = \{a\}\{b\} = \{ab\}$, ja edelleen $L(r_1) = L((ab)b) = L(ab)L(b) = \{abb\}$.
  - Kohdan (vi) perusteella $L(r_2) = L((ab)^*) = \{ab\}^* = \{\epsilon, ab, abab, ababab, \ldots\}$.
  - Kohdan (vi) perusteella $L(b^*) = \{b\}^* = \{\epsilon, b, bb, bbb, \ldots\} = \{b^i \mid i \geq 0\}$, ja kohdan (v) perusteella $L(r_3) = L(ab^*)\{a\}\{b\}^* = \{ab^i \mid i \geq 0\}$.
  - Selvästi $L(b) = \{b\}$ ja $L(bb) = \{bb\}$, siispä säännön (iv) perusteella $L(b \cup bb) = L(b) \cup L(bb) = \{b\} \cup \{bb\} = \{b, bb\}$
  - Säännön (v) perusteella siis $L(a(b \cup bb)) = \{ab, abb\}$ joten säännön (vi) perusteella saamme $L(r_4) = L((a(b \cup (bb)))^*) = \{ab, abb\}^* = \{\epsilon, ab, abb, abab, ababb, abbab, abbabb, \ldots\}$.

- Sulkumerkkien vähentämissääntöjä:
  - Operaattoreiden prioriteetti:

$$* \quad \succ \quad \cdot \quad \succ \quad \cup$$

  - Yhdiste- ja tulo-operaatioiden assosiatiivisuus:

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

    $\Rightarrow$ peräkkäisiä yhdisteitä ja tuloja ei tarvitse suluttaa
  - Edelliset lausekkeet voidaan siis määritellä yksinkertaisemmin:

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

- Huom: Usein merkitään $r^+ = rr^* = r^*r$
- Kuten huomaamme, säännöllinen lauseke siis on merkkijono, joka kuvaa jonkin formaalin kielen. Säännöllinen lauseke on aina äärellinen, mutta lausekkeiden kuvaamat kielet ovat usein äärettömiä.
- *Määritelmä:* Kieli on *säännöllinen*, jos se voidaan kuvata säännöllisellä lausekkeella.
- Kuten kuitenkin tulemme huomaamaan, ei kaikkia formaaleja kieliä pystytä esittämään säännöllisten lausekkeiden avulla.

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

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

  missä l on lyhenne lausekkeelle $l = (\text{a} \cup \text{b} \cup ... \cup \text{ö})$ ja siis $l^* = \Sigma^*$.
- Esim.: Olkoon $\Sigma = \{A, B, \ldots, a, b, \ldots, 0, 1, 2, \ldots\}$. Osoite on muotoa

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

  missä d on lyhenne lausekkeelle

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

  ja l on lyhenne lausekkeelle

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

- Edellisiin esimerkkeihin säännölliset lausekkeet sopivat melko hyvin. Aina ei näin välttämättä ole.
- Kieli, johon kuuluvat kaikki ne merkkijonot, joissa ei esiinny sanaa "käytäntö", on kyllä säännöllinen kieli, mutta kielen määrittelevästä säännöllisestä lausekkeesta tulee pitkä ja epähavainnollinen.

- C-kielen etumerkittömät liukuluvut tunnistava kieli voidaan määritellä säännöllisellä lausekkeella (ilman suffikseja):
  number $= (d^+.d^* \cup .d^+)(\epsilon \cup ((e \cup E)(+ \cup - \cup \epsilon)d^+))$
  $\cup\, d^+(e \cup E)(+ \cup - \cup \epsilon)d^+$
- Kieleen kuuluvat esim. seuraavat merkkijonot: 12., .12, 1.2, 1.2E3, 1.2e3, 1.2E-3,1E2, 1e23
- Vaikka määrittely onnistuu, se on vaikeaselkoisempi kuin edellä esitelty saman asian ajanut äärellinen automaatti.

Säännöllisten lausekkeiden sieventäminen

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

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

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

Sievennyssääntöjä

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

- Kun lisätään seuraava päättelysääntö:
  - Jos $r = rs \cup t$, niin $r = ts^*$, kun $\epsilon \notin L(s)$.

  voidaan mikä tahansa säännöllisten lausekkeiden ekvivalenssi johtaa näistä laskulaeista.
- Näiden sääntöjen lisäksi voidaan soveltaa suorempaakin päättelyä, ts. järjen käyttöä.

- Näytetään, että pari sivua aiemmin esitetyt kolme säännöllistä lauseketta kuvaavat kaikki saman kielen. Käytetään hyväksi seuraavaa havaintoa:
- $L(r) = L(s) \Leftrightarrow L(r) \subseteq L(s) \wedge L(s) \subseteq L(r)$ eli $r = s \Leftrightarrow r \subseteq s \wedge s \subseteq r$
- Nyt kolmen lausekkeen r, s ja t ekvivalenssi voidaan osoittaa näyttämällä erikseen $s \subseteq r$, $r \subseteq t$ ja $t \subseteq s$, sillä nyt joukkojen sisältyvyyden transitiivisuuden (eli jos $A \subseteq B$ ja $B \subseteq C$ niin myös $A \subseteq C$) nojalla pätee myös $s \subseteq t$, $r \subseteq s$ ja $t \subseteq r$.
- 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
  4. Siispä $(a \cup b)^* = (a^*b^*)^* = a^*b^* \cup (a \cup b)^*ba(a \cup b)^*)$.
- Voidaan todistaa (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.

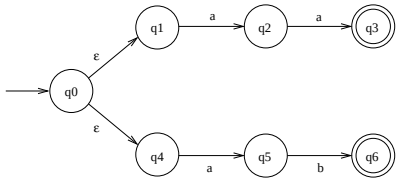
2.5. Kaksi äärellisen automaatin laajennusta

Seuraavassa halutaan näyttää, että äärelliset automaatit ja säännölliset lausekkeet ovat yhtä ilmaisuvoimaisia formalismeja, eli että niillä voidaan määritellä täsmälleen samat kielet.

Tässä tarkoituksessa määritellään äärelliselle automaatille ensin kaksi laajennusta.

ϵ-automaatti

- Epädeterministinen äärellinen automaatti, jossa sallitaan ϵ-*siirtymät*, eli siirtymät, joissa automaatti ei lue yhtään syötemerkkiä. ϵ-siirtymässä automaatti siis ei "katso" syötenauhaan siirtyessään tilasta toiseen.



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

- Yllä oleva automaatti siis siirtyy ensin joko ylempään (tilaan $q1$) tai alempaan haaraansa (tilaan $q4$) ja vasta tämän jälkeen 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')

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

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

- Muut määritelmät ovat kuten tavanomaisilla epädeterministisillä äärellisillä automaateilla.

- Esim. yllä olevassa automaatissa $(q0, aa) \vdash (q1, aa)$, koska $q0$:sta on ϵ-siirtymä tilaan $q1$ ja $(q1, aa) \vdash (q2, a)$ sekä $(q2, a) \vdash (q3, \epsilon)$. 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

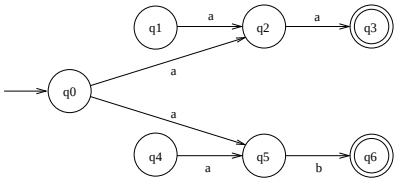
$$\hat{\delta}(q, a) = \{q' \in Q \mid (q, a) \underset{M}{\vdash}^* (q', \epsilon)\};$$

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

□

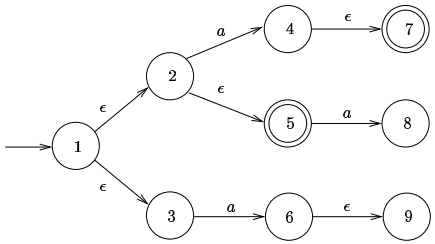
ϵ-siirtymien poisto

- Muodosta siirtymä $q \xrightarrow{a} q'$, jos $(q, a) \underset{M}{\vdash}^* (q', \epsilon)$.
- Lisää alkutila q_0 lopputiloihin, jos $(q_0, \epsilon) \underset{M}{\vdash}^* (q_f, \epsilon)$, s.e. $q_f \in F$.
- Eli jos halutaan muodostaa ϵ-automaattia M vastaava epädeterministinen automaatti $\widehat{M}$, niin toimitaan seuraavasti:
  - Kopioidaan kaikki M:n tilat sekä normaalit siirtymät automaattiin $\widehat{M}$.
  - Jos M ssä päästään lopputilaan pelkillä ϵ-siirtymillä, tehdään $\widehat{M}$:n alkutilasta lopputila.
  - Tämän jälkeen tutkitaan jokaista tilaparia $q, q' \in Q$ sekä jokaista aakkosta $a \in \Sigma$.
    - Jos automaatissa M on olemassa polku tilasta q tilaan q' siten, että matkalla tehdään vain ϵ-siirtymiä sekä tasan kerran siirtymä a, tulee automaattiin $\widehat{M}$ siirtymä tilasta q tilaan q' a:llä.

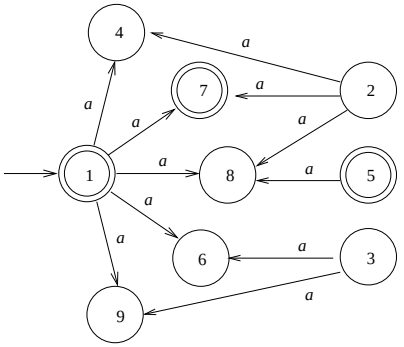


Kuva 2: Kuvan 1 ϵ-automaattia vastaava epädeterministinen automaatti. Tilat $q1$ ja $q4$ voidaan poistaa, sillä niitä ei saavuteta alkutilasta.

Toinen esimerkki:



Kuva 3: ϵ-automaatti M.



Kuva 4: 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

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

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

- Yhden askelen tilannejohto määritellään

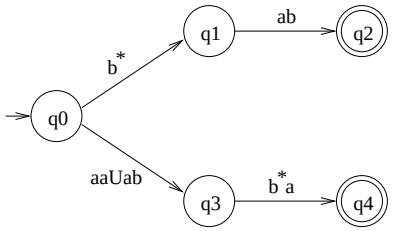
$$(q, w) \underset{M}{\vdash} (q', w'),$$

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

- Intuitiivisesti on siis kyse siitä, että lausekeautomaatti lukee yhdessä siirtymässään *merkkijonon*, siis mahdollisesti enemmän kuin yhden syötemerkin.

- Siirtymissä luettavat merkit määritellään säännöllisten lausekkeiden avulla.

- Tarkastellaan seuraavaa lausekeautomaattia:



- Siirtymässä $q0$:sta $q1$:n voidaan lukea minkä tahansa pituinen pelkästään merkistä b koostuva merkkijono (siis myös 0:n mittainen ϵ).
  - Esimerkiksi $(q0, bbbab) \vdash (q1, ab)$, toisaalta myös sekä $(q0, bbbab) \vdash (q1, bab)$, $(q0, bbab) \vdash (q1, bbab)$ että $(q0, bbbab) \vdash (q1, bbbab)$.

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

- Koska $(q0, bbbab) \vdash (q1, ab)$ ja $(q1, ab) \vdash (q2, \epsilon)$, on siis $(q0, bbbab) \vdash^* (q2, \epsilon)$, ja koska $q2$ on hyväksymätila, hyväksyy lauseautomaatti merkkijonon $bbbab$.

- Esim.: merkkijonot aaa ja $abba$?

2.6. Äärelliset automaatit ja säännölliset kielet

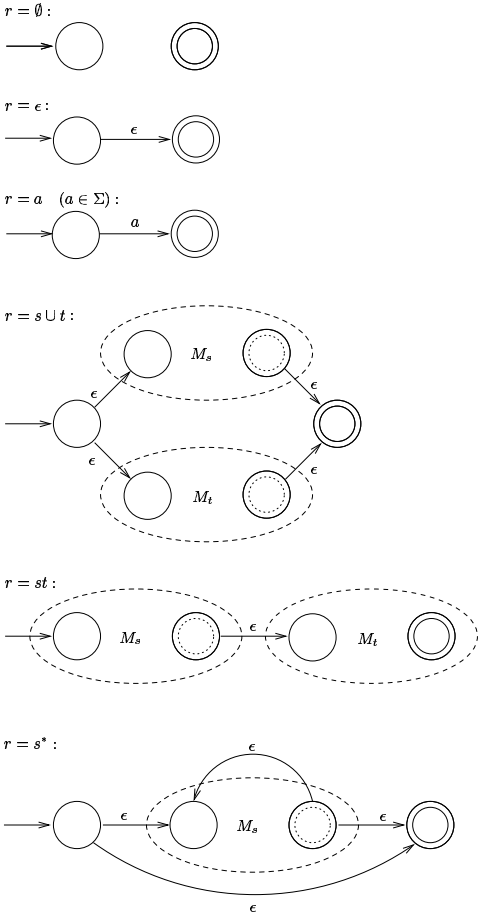
- Osoitetaan seuraava tulos:

  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.
     - Tällöin on olemassa vastaava ϵ-siirtymätön epädeterministinen automaatti.
     - Haluttaessa epädeterministinen automaatti voidaan vielä determinisoida (ja minimoida).
  2. Kieli $L(M)$ voidaan tunnistaa äärellisellä automaatilla M $\Rightarrow$ $L(M)$ on säännöllinen kieli:
     - Tarkastellaan lausekeautomaatteja - jos väite pätee lausekeautomaateille, se pätee myös tavallisille äärellisille automaateille (jotka ovat niiden erikoistapaus).
     - Redusoidaan lausekeautomaatti korkeintaan 2-tilaiseksi automaatiksi, josta voidaan lukea suoraan vastaava säännöllinen lauseke.

- *Lause:* Jokainen säännöllinen kieli voidaan tunnistaa äärellisellä automaatilla.
  Todistus:
  - Muodostetaan mielivaltaista säännöllistä lauseketta r vastaava ϵ-automaatti M_r, jolla $L(M_r) = L(r)$ (ks. kuva)
  - M_r:stä voidaan poistaa ϵ-siirtymät edellisen lemman mukaisesti, ja tarvittaessa voidaan syntyvä epädeterministinen automaatti determinisoida aiemmin esitetyn konstruktion avulla. $\square$

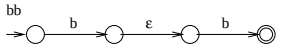


Kuva 5: Lauseketta r vastaavan ϵ-automaatin M_r muodostaminen.

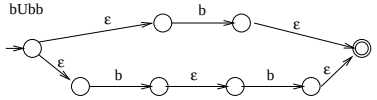
- Esimerkkinä säännöllisen lausekkeen $a(b \cup bb)^*$ muuntaminen ϵ-automaatiksi.

vaihe 1:

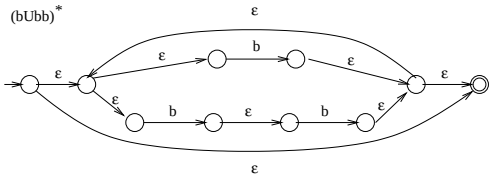
vaihe 2:



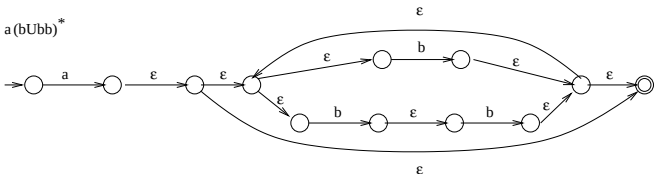
vaihe 3:



vaihe 4:



vaihe 5:



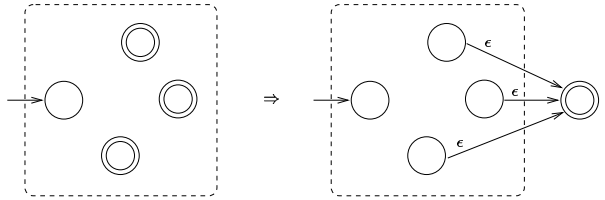
- Tuloksena saatu automaatti voidaan muuntaa opituin keinoin epädeterministiseksi ja edelleen deterministiseksi automaatiksi.

- Olemme siis todistaneet, että jokainen säännöllinen lauseke voidaan mekaanisesti muuttaa äärelliseksi automaatiksi siten, että automaatti hyväksyy saman kielen kuin säännöllinen lauseke.
- Tämä siis todistaa sen että kaikki mitä säännöllisillä lausekkeilla voidaan kuvata, voidaan kuvata myöskin äärellisillä automaateilla.
- Tulos on tietysti jo itsessään mielenkiintoinen, mutta todistuksen konstruktiot ovat hyödyllisiä myös käytännön kannalta: saamme niitä käyttäen helposti (ts. mekaanisesti) generoitua tunnistajaohjelman säännöllisinä lausekkeina määritellyille kielille.
- Näytetään seuraavaksi, että sama onnistuu myös toisin päin: mielivaltaisesta äärellisestä automaatista voidaan mekaanisesti tuottaa säännöllinen lauseke, joka tunnistaa saman kielen kuin ko. automaatti.

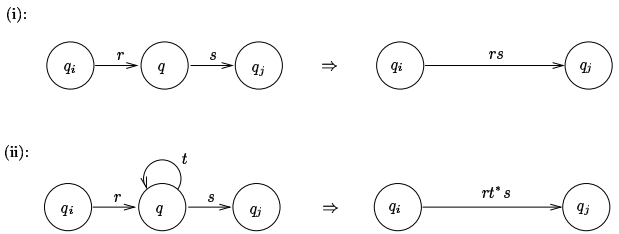
- *Lause:* Jokainen äärellisellä automaatilla tunnistettava kieli on säännöllinen.
  Todistus: Osoitetaan, että jokainen lausekeautomaatilla tunnistettava kieli on säännöllinen.

  Idea: Redusoidaan lausekeautomaatti siten, että siitä voidaan lukea vastaava säännöllinen lauseke:

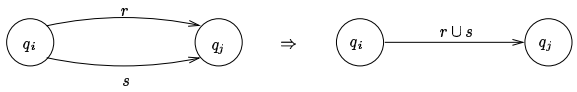
  – yhdistetään M:n lopputilat yhdeksi ϵ-siirtymillä (kts. kuva 6)
  – toistetaan seuraavaa kunnes automaatissa on vain alku- ja lopputiloja:
    * valitse q, $q \neq q_0$, $q \neq q_f$ $(q_f \in F)$
    * poista q reitiltä $q_i \rightarrow q \rightarrow q_j$, missä $q_i = q$:n edeltäjätila ja $q_j = q$:n seuraajatila soveltamalla kuvan 7 reduktiosääntöjä. Muista käydä läpi kaikki tilan q:n sisältävät polut!
    * yhdistä rinnakkaiset siirtymät (kts. kuva 8)



Kuva 6: Lausekeautomaatin lopputilojen yhdistäminen.

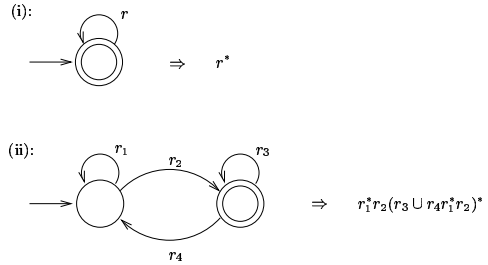


Kuva 7: Tilan poistaminen lausekeautomaatista.



Kuva 8: Rinnakkaisten siirtymien yhdistäminen lausekeautomaatissa.

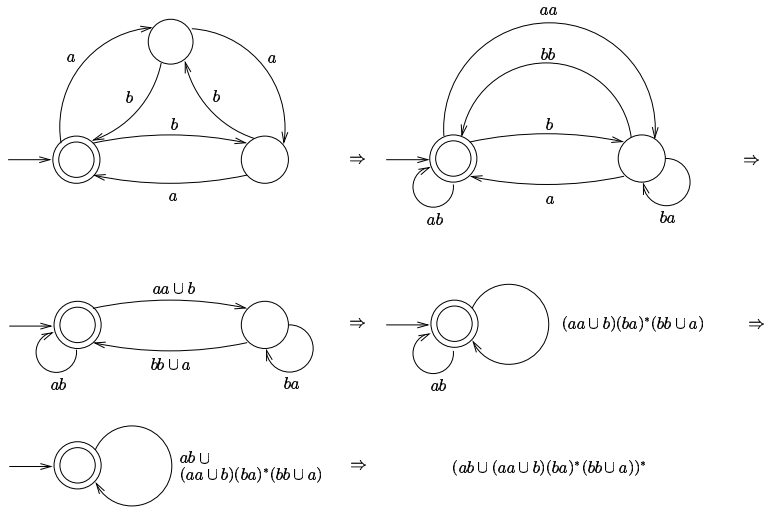
– Tiivistyksen päättyessä jäljellä olevaa enintään 2-tilaista automaattia vastaava säännöllinen lauseke muodostetaan kuten alla:



Kuva 9: Säännöllisen lausekkeen muodostaminen redusoidusta lausekeautomaatista.

□

• Esimerkki:

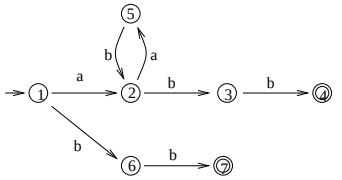


Kuva 10: Säännöllisen lausekkeen muodostaminen äärellisestä automaatista.

2.7. Säännöllisten kielten rajoituksista

- Minkä tahansa aakkoston formaaleja kieliä (päätösongelmia) on ylinumeroituva määrä, mutta sen säännöllisiä lausekkeita on vain numeroituva määrä $\Rightarrow$ kaikki kielet eivät ole säännöllisiä.
- Perusrajoitus: äärellisillä automaateilla on vain rajallinen "muisti"
- Milloin ääretön kieli on säännöllinen?
  - oltava jokin toistuva rakenne (sulkeuma)
  - automaatissa tätä vastaa silmukka
- Esim. tasapainoisten sulkujonojen muodostama kieli $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ ei ole säännöllinen, eli sitä ei voida tunnistaa äärellisellä automaatilla. Näin esimerkiksi hyvinmuodostettuja aritmeettisia lausekkeita ei voida tunnistaa äärellisellä automaatilla
- "Pumppauslemma" formalisoi tämän rajallisen muistin idean.
- Idea: mitä tahansa annetun säännöllisen kielen riittävän pitkää merkkijonoa voidaan "pumpata" keskeltä, ilman että kielen tunnistava äärellinen automaatti "huomaa" muutosta.

- Johdatuksena pumppauslemman ideaan tarkastellaan oheista automaattia, joka hyväksyy sen kielen, jonka säännöllinen lauseke $bb \cup a(ab)^*bb$ tuottaa.

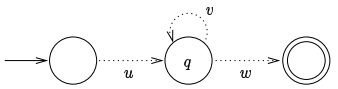


- Automaatti hyväksyy merkkijonon $aabbb$, sillä $(1, aabbb) \vdash (2, abbb) \vdash (5, bbb) \vdash (2, bb) \vdash (3, b) \vdash (4, \epsilon)$.
- Tässä kannattaa huomata se, että automaatti tekee silmukan $2 \xrightarrow{a} 5 \xrightarrow{b} 2$ lukiessaan syötteen toisen ja kolmannen merkin.
- Automaatti tekee silmukan samaan tapaan myös silloin, kun se käsittelee pidempiä syötteitään, jotka se tulee hyväksymään. Hyväksytyistä syötteistä ainoastaan abb ja bb eivät aiheuta silmukan suoritusta.
- Onkin siis olennaisesti niin, että automaatti hyväksyy minkä tahansa merkkijonon, jossa tilojen 2 ja 5 välillä on suoritettu silmukka mielivaltaisen monta kertaa:
- Eli esim. syötteellä $aababbb$ 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 yleisesti 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.
- Peruste on sama kuin mikä äsken esitettiin. Jos tarkastellaan kielen L hyväksyvää äärellistä automaattia, ovat "kaikki tarpeeksi pitkät" hyväksytyt merkkijonot sellaisia, että niitä käsitellessään automaatti suorittaa silmukan, ja tätä luuppia voidaan toistaa halutessa miten monta kertaa tahansa.
- *Lemma* (*Pumppauslemma*): Olkoon A säännöllinen kieli. Tällöin on olemassa $n \geq 1$ s.e. mikä tahansa $x \in A$, $|x| \geq n$, voidaan jakaa osiin $x = uvw$, $|uv| \leq n$, $|v| \geq 1$, ja $uv^i w \in A$ kaikilla $i = 0, 1, 2, \ldots$.
- Lemman mukaan siis 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; mitään tarkempaa pumppauskohdasta ei voida lemman perusteella sanoa.

Todistus (eli johdattelun päättely formaalimmin esitettynä):

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



Kuva 11: Merkkijonon $x = uvw \in A$ pumppaus.

- Pumppauslemma siis kertoo, että kaikilla säännöllisillä kielillä on se ominaisuus, että niiden tarpeeksi pitkät merkkijonot pumppautuvat.
- Jos jollekin kielelle L onnistutaan löytämään merkkijono, joka ei täytä pumppauslemman ehtoa, ts. merkkijono on tarpeeksi pitkä, mutta ei pumppautuva, tarkoittaa tämä että L ei ole säännöllinen kieli.
- Näinollen pumppauslemmaa käyttäen voidaan todistaa, ettei jokin kieli ole säännöllinen.
- Todistustekniikka sille, että kieli L ei ole säännöllinen, on seuraava:
  - Oletetaan, että kieli L on säännöllinen.
  - On siis olemassa luku n, jota pidemmät merkkijonot ovat pumppautuvia. Huom. Tiedetään ainoastaan, että tällainen n on olemassa, jos L on säännöllinen; ei kuitenkaan tiedetä mikä luvun n tarkka arvo on.
  - Näytetään, että jokin merkkijono $x \in L$, jonka pituus on vähintään n ei pumppaannu *millään* lemman mukaisella jakotavalla $x = uvw$, eli että jollakin $i \geq 0$ $uv^iw \notin L$.
  - Nyt oletus, että L on säännöllinen kieli on aiheuttanut ristiriidan. Siispä kieli L ei ole säännöllinen.

- Esim. Väite: Sulkulausekekieli

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

on epäsäännöllinen.

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

Valitaan pumpattavaksi merkkijonoksi $x = (^n)^n$, jolloin $|x| = 2n > n$. Vaikka siis emme tiedä konkreettista arvoa luvulle n, sitä voidaan käyttää muodostaessamme merkkijonoa, sillä tiedämme että luku n on olemassa (sitä itseasiassa *täytyy* käyttää, koska muuten emme voi tietää, että merkkijono on tuntematonta n:ää pidempi!)

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

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

Pumpattava osa sisältää siis pelkästään (-merkkejä, ja on varma, että pumpattavan osan pituus on vähintään 1. Huom.: tässä i:n ja j:n tarkkaa arvoa ei tiedetä. Pumppauslemma takaa, että $i+j \leq n$ ja $j \geq 1$, eli tiedossa on, että pumpattavan osan pituus on vähintään 1, muttei enempää kuin n.

Nyt esimerkiksi 0-kertaisesti pumpattu merkkijono $uv^0w = (^i(^{n-(i+j)})^n = (^{n-j})^n$ ei kuulu kieleen L, sillä siinä on (-merkkejä ainakin 1 vähemmän kuin)-merkkejä. Tämä on ristiriita, eikä L siten ei voi olla säännöllinen. □

- Huom.: Pumppauslemma ei sano, että voimme valita u:n ja v:n haluamallamme tavalla!
- Todistusstrategia: Valitaan tarkasteltavan jonon pituudeksi esim. $2n$, jossa ensimmäiset n merkkiä muodostavat uv:n, muttei kiiinnitetä tarkkaan ottaen miten.
- Esim. Väite: $L = \{a^k b^l \mid l \geq k \geq 0\}$ ei ole säännöllinen.
  Tod.: Vastaoletus: L on säännöllinen $\Rightarrow \exists n \geq 1$, jonka pituisia merkkijonoja voidaan pumpata. Valitaan $x = a^n b^n$. 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 siis koostuu vähintään yhdestä a merkistä.
  $uv^0w = a^i a^{n-(i+j)} b^n = a^{n-j} b^n$, joka kuuluu kieleen L. Eli 0-kertainen pumppaus onnistui, mutta toisaalta
  $uv^2w = a^i a^{2j} a^{n-(i+j)} b^n = a^{n+j} b^n$, joka ei kuulu kieleen L. Ristiriita, eli L ei voi olla säännöllinen. □

  Havaittiin siis esimerkki siitä, että täytyy löytää vain (ja ainakin) yksi pumppausten määrä, jolla saatu merkkijono ei kuulu kieleen.

- Esim. Väite: $L = \{c^l a^k b^k \mid k \geq 0 \wedge l \geq 1\}$ ei ole säännöllinen.

  Tod.: Vastaoletus: L on säännöllinen $\Rightarrow \exists n \geq 1$, 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 merkistä a, vaan on tyyppiä $ca \ldots a$.
  Jakoja uvw on nyt (todistuksen kannalta olennaisesti) kahta eri tyyppiä; toisessa c sisältyy osaan u ja toisessa pumpattavaan osaan v (jolloin u:ssa ei ole yhtään merkkiä):

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

  Kumpaakin tapausta on nyt tarkasteltava erikseen, sillä halutaan näyttää, ettei x:n *mitään* jakoa uvw voida pumpata.

  1. $uv^0w = ca^i a^{n-(i+j)} b^n = ca^{n-j} b^n$ ei kuulu kieleen L, sillä merkkejä a on vähemmän kuin merkkejä b. Mikään jako, jossa c sisältyy osaan u ei siis onnistu, sillä 0-kertainen pumppaus ei kuulu kieleen.
  2. $uv^0w = a^{n-j} b^n$ ei kuulu kieleen L, sillä alussa ei ole merkkiä c. Mikään 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 mitä saamme jaosta olettaa on se, että $|uv| \leq n$ ja $|v| \geq 1$, eli että
  - pumpattava osa on merkkijonon n:n ensimmäisen merkin sisällä ja pumpattavan osan pituus on vähintään 1 merkki.
- Merkitsimmekin yllä jakoja seuraavalla tavalla:

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

missä $i + j \leq n$ ja $j \geq 1$. Tässä arvoja i ja j ei siis ole kiinnitetty; ne on ainoastaan rajoitettu lemman määräämällä tavalla.

- Koska siis määrittelemme merkkijonot avoimien parametrien i ja j avulla, tarkastelemme ikäänkuin useaa (jokaista lemman rajoitteiden sallimaa) pumppausta rinnakkain. Muotoilu

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

tarkoittaa siis kaikkia jakoja:

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

Kertausta: säännölliset kielet

- Kieli A on säännöllinen jos ja vain jos on olemassa säännöllinen lauseke r, joka kuvaa kielen, eli $L(r) = A$.
- Toisaalta näytimme, että säännölliset lausekkeet ja äärelliset automaatit ovat yhtä ilmaisuvoimaisia kuvaustapoja, eli
- jos kieli on kuvattavissa säännöllisenä lausekkeena, on se kuvattavissa (eli tunnistettavissa) myös äärellisellä automaatilla ja päinvastoin.
- Kieli A on siis säännöllinen jos ja vain jos on olemassa äärellinen automaatti M joka tunnistaa kielen, eli $L(M) = A$.
- Yllä olevasta seuraa, että jos haluamme todistaa, että jokin kieli on säännöllinen, riittää näyttää että kielen voi kuvata säännöllisellä lausekkeella tai äärellisellä automaatilla.
- Pumppauslemman avulla taas voidaan osoittaa, että jokin kieli *ei* ole säännöllinen.
- Havaittiin, että säännölliset kielet ovat vain osajoukko kaikista formaaleista kielistä.
- Ne muodostavat kuitenkin hyödyllisen osajoukon, sillä esim. merkkijonon x kuuluvuus kieleen L on ratkeava ongelma, ja kuuluvuuden testaus on vieläpä tehokasta, sillä kielen tunnistavaa äärellistä automaattia vastaava algoritmi toimii ajassa $O(n)$, missä n on syötemerkkijonon pituus.

2.8. Säännöllisten kielten sovelluksia

- Kuten olemme huomanneet, tarjoavat luvun 2 mekanismit apukeinoja ohjelmakoodien syntaksin tarkastukseen. Säännöllisten kielten avulla voidaan esim. määritellä minkälaisia muuttujanimet tai liukuluvut voivat olla.
- Luvun 2 mekanismeja käyttäen voidaankin toteuttaa ohjelmointikielen *selaaja*: se kääntäjän osa, joka erottelee koodista erilaiset ohjelmointikielen kannalta mielekkäät kokonaisuudet, kuten varatut sanat, muuttujanimet, luvut, yms.
- Säännölliset lausekkeet ovat käytössä monilla eri tietojenkäsittelyn osa-alueilla: niiden avulla määritellään ohjelmien parametrejä (grep), joidenkin ohjelmointikielten (Perl, Java) merkkijono-operaatiot sisältävät säännöllisiä lausekkeita, ym.
- Äärelliset automaatit ovat myös käytössä monin paikoin, esim. ohjelmien määrittelyssä (vrt. kahviautomaattiesimerkkimme).
- Eräs esimerkki on vielä ohjelmistojen oikeaksitodistaminen, jonka jotkut menetelmät perustuvat äärellisiin automaatteihin sekä niiden laajennuksiin.
- Sovelluksia löytyy siis sekä käytännöllisen- että teoreettisen tietojenkäsittelyn osa-alueilta.

3. Kontekstittomat kielet ja pinoautomaatit

- Kuten edellä (pumppauslemmaa soveltamalla) nähtiin, esim. tasapainoisten sulkulausekkeiden muodostama kieli

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

  ja if-else-parien muodostama kieli

$$L_{\text{if-else}} = \{\text{if}^k\text{else}^l | l \leq k\}$$

  eivät ole säännöllisiä,
- mutta näitä kieliä voidaan kuvata *kontekstittomilla kieliopeilla*.
- Säännölliset kielet voidaan tunnistaa äärellisillä automaateilla; kontekstittomien kielioppien määrittelemät kielet voidaan tunnistaa *pinoautomaateilla*.

3.1. Kontekstittomat kieliopit ja kielet

- Esim. yksinkertainen rekursiivinen kuvaus sulkulausekekielelle. Olkoon S mielivaltainen sulkumerkkijono. S on tasapainoinen sulkumerkkijono, jos

  (i) $S = \epsilon$ tai

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

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

  (i) $S \to \epsilon$,

  (ii) $S \to (S)$

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

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

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

- Merkintä

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

  on lyhennys merkinnästä

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

- Esim. yksinkertainen kielioppi tietyille aritmeettisille lausekkeille:

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

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

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

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

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

missä

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

- Produktio $(A, \omega) \in P$ merkitään $A \to \omega$.

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

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

- 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. $(E) * F \Rightarrow (E + T) * F$, sillä oikeanpuoleinen merkkijono on saatu soveltamalla vasemmanpuoleiseen 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 $\gamma_0, \gamma_1, \ldots, \gamma_n$ $(n \geq 0)$ siten, että

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

  - Esim. $(E)*F \Rightarrow^* (a+a)*a$, 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 on $S \underset{G}{\Rightarrow}^* \gamma$. Koska $E \Rightarrow^* (E) * a$, on $(E) * a$ aritmeettisia lausekkeita tuottavan kieliopin lausejohdos. Lausejohdos on siis jokin joko välikkeistä tai päätemerkeistä koostuva merkkijono, jonka kielioppi tuottaa.
- G:n *lause* on pelkästään päätemerkeistä koostuva G:n lausejohdos $x \in \Sigma^*$. Eli koska $E \Rightarrow^* (a + a) * a$, on $(a + a) * a$ lause aritmeettisia lausekkeita tuottavassa kieliopissa.
- Kieliopin G *tuottama* t. *kuvaama kieli* koostuu G:n lauseista:

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

- Eli merkkijono x kuuluu kontekstittoman kieliopin G kuvaamaan kieleen $L(G)$ jos ja vain jos x voidaan tuottaa lähtien kieliopin lähtösymbolista.
- *Määritelmä:* Formaali kieli $L \subseteq \Sigma^*$ on *kontekstiton* jos ja vain jo se voidaan tuottaa jollakin kontekstittomalla kieliopilla.

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

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

missä

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

- Toinen kielioppi kielen L_{expr} tuottamiseen on

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

missä

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

- Saman kontekstittoman kielen voi siis tuottaa useilla erilaisilla kieliopeilla.

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:

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

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

OLPE03-kielen syntaksi kontekstittomana kielioppina

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

- Päätesymbolit: **if**, **then**, **else**, **while**, **do**, **begin**, **end**, **not**, **and**, **read**, **write**, ;, " := ", +, −, =, <, (,), *var*, *int*,
- missä *int* edustaa (mitä tahansa) syntaktisesti korrektia kokonaislukua ja *var* (mitä tahansa) syntaktisesti korrektia muuttujanimeä (jotka ovat siis määriteltävissä säännöllisiksi kieliksi).
- Oheinen kielioppi määrittelee Olpe03-kielen syntaksin. Eli aakkoston Σ_{acii} merkkijono on syntaktisesti korrekti ope03-kielinen ohjelma, jos se voidaan tuottaa lähtien kieliopin lähtösymbolista S.
- Huom.: kääntäjän syötteenä on aakkoston Σ_{ascii} merkkijono; kielen *selaaja* pilkkoo tämän merkkijonon ennen syntaksitar-

kistusta oheisista päätesymboleista koostuvaksi ”päätesymbolimerkkijonoksi”.

- Onko seuraava merkkijono OLPE03:n syntaksin mukainen?
  $i := 0;$
  while $i < 10$ **do**
  $\quad i := i + 1$

- Kyllä, sillä
  $\underline{S} \Rightarrow$
  $\underline{S}; S \Rightarrow$
  $i := \underline{E}; S \Rightarrow$
  $i := \underline{E}; S \Rightarrow$
  $i := 0; \underline{S} \Rightarrow$
  $i := 0;$ **while** B **do** $\underline{S} \Rightarrow$
  $i := 0;$ **while** B **do** $i := \underline{E} \Rightarrow$
  $i := 0;$ **while** B **do** $i := \underline{E} + E \Rightarrow$
  $i := 0;$ **while** B **do** $i := i + \underline{E} \Rightarrow$
  $i := 0;$ **while** $\underline{B}$ **do** $i := i + 1 \Rightarrow$
  $i := 0;$ **while** $\underline{E} < E$ **do** $i := i + 1 \Rightarrow$
  $i := 0;$ **while** $i < \underline{E}$ **do** $i := i + 1 \Rightarrow$
  $i := 0;$ **while** $i < 10$ **do** $i := i + 1$

Säännölliset kielet ja kontekstittomat kieliopit

- Kontekstittomilla kieliopeilla voidaan kuvata joitakin ei-säännöllisiä kieliä (esimerkiksi kielet L_{match} ja L_{expr}).
- Osoitetaan, että *kaikki* säännölliset kielet voidaan kuvata kontekstittomilla kieliopeilla.
- Kontekstittomat kielet ovat siten säännollisten kielten aito yliluokka.
- Jokainen säännöllinen kieli on siis kontekstiton.

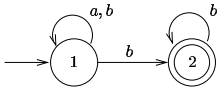
Oikealle ja vasemmalle lineaariset kieliopit

- *Määritelmä:* Kontekstiton kielioppi on *oikealle lineaarinen*, jos sen kaikki produktiot ovat muotoa $A \to \epsilon$ tai $A \to aB$, ja *vasemmalle lineaarinen*, jos sen kaikki produktiot ovat muotoa $A \to \epsilon$ tai $A \to Ba$.
- Osoittautuu, että sekä vasemmalle että oikealle lineaarisilla kieliopeilla voidaan tuottaa täsmälleen säännölliset kielet
- $\Rightarrow$ lineaarisia kielioppeja nimitetään myös yhteisesti *säännöllisiksi* kieliopeiksi.
- Todistetaan tässä 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 tai epädeterministinen) äärellinen automaatti. Muodostetaan kielioppi G_M, jolla on $L(G_M) = L(M) = L$.
  - G_M:n pääteaakkosto = M:n syöteaakkosto Σ
  - G_M:n välikeaakkostoon otetaan yksi välike A_q kutakin M:n tilaa q kohden
  - G_M:n lähtösymboli on A_{q_0}
  - G_M:n produktiot vastaavat M:n siirtymiä:
    (i) kutakin M:n lopputilaa $q \in F$ kohden kielioppiin otetaan produktio $A_q \to \epsilon$;
    (ii) kutakin M:n siirtymää $q \xrightarrow{a} q'$ (so. $q' \in \delta(q, a)$) kohden kielioppiin otetaan produktio $A_q \to aA_{q'}$.
  - Tarkastetaan konstruktion oikeellisuus:
    * Merk. A_q:sta tuotettavien päätejonojen joukkoa
    $$L(A_q) = \{x \in \Sigma^* \mid A_q \underset{G_M}{\Rightarrow}{}^* x\}$$
    * Induktiolla merkkijonon x pituuden suhteen voidaan osoittaa, että kaikilla q on
    $$x \in L(A_q) \;\Leftrightarrow\; (q, x) \underset{M}{\vdash}{}^* (q_f, \epsilon) \text{ jollakin } q_f \in F$$

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

- Esim.



Kuva 12: Yksinkertainen äärellinen automaatti.

Automaattia vastaava kielioppi on:

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

- *Lause:* Jokainen oikealle lineaarisella kieliopilla tuotettava kieli on säännöllinen.
  Todistus: Olkoon $G = (V, \Sigma, P, S)$ oikealle lineaarinen kielioppi. Muodostetaan kielen $L(G)$ tunnistava epädeterministinen äärellinen automaatti $M_G = (Q, \Sigma, \delta, q_S, F)$ seuraavasti:
  - M_G:n tilat vastaavat G:n välikkeitä:
  $$Q = \{q_A \mid A \in V - \Sigma\}$$
  - M_G:n alkutila on lähtösymbolia S vastaava tila q_S
  - M_G:n syöteaakkosto on G:n pääteaakkosto Σ

– M_G:n siirtymäfunktio δ jäljittelee G:n produktioita siten, että kutakin produktiota $A \to aB$ kohden automaatissa on siirtymä $q_A \xrightarrow{a} q_B$ (so. $q_B \in \delta(q_A, a)$)
M_G:n lopputiloja ovat ne tilat, joita vastaaviin välikkeisiin liittyy G:ssä ϵ-produktio:

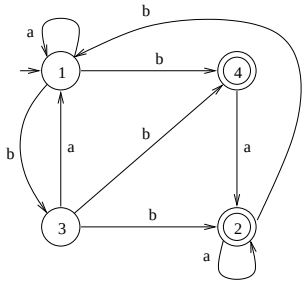
$$F = \{q_A \in Q \mid A \to \epsilon \in P\}$$

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

Tarkastellaan oikealle lineaarista kielioppia:

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

Vastaava äärellinen automaatti on:



3.2. Kielioppien jäsennysongelma

- Ratkaistava tehtävä:

  ”Annettu kontekstiton kielioppi G ja merkkijono x. Pätekö $x \in L(G)$?”

- Ratkaistaan *jäsennysalgoritmilla*.
- Useita vaihtoehtoisia menetelmiä, erityisesti kun G on jotain rajoitettua (käytännössä esiintyvää) muotoa.
- Tarkastellaan 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

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

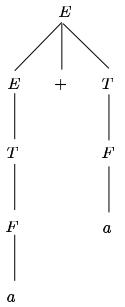
- Esim. lauseen $a + a$ johtoja kieliopissa G_{expr}:

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

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

Jäsennyspuut

- Suurin osa vaihtoehtoisten johtotapojen eroista muodostuu vain välikkeiden laventamisesta eri järjestyksessä (vrt. äskeiset johdot (i)–(iii)).
- Lausejohdoksen *jäsennyspuu* (syntaksipuu, johtopuu) (engl. *parse tree, syntax tree, derivation tree*) on esitystapa, jossa nämä epäoleelliset erot on abstrahoitu pois.
- Jäsennyspuu kertoo ainoastaan, *miten* kukin välike on lavennettu, ei *missä järjestyksessä* lavennukset on tehty.
- Esim. kaikkia kolmea em. johtoa vastaa sama jäsennyspuu:



Kuva 13: Lauseen $a + a$ jäsennyspuu kieliopissa G_{expr}.

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

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

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

- Jäsennyspuun τ *tuotos* on 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 $S \to X_1 X_2 \ldots X_k$, niin juurelle tulee k jälkeläissolmua, joiden nimet vasemmalta oikealle ovat $X_1, X_2, \ldots, X_k$;

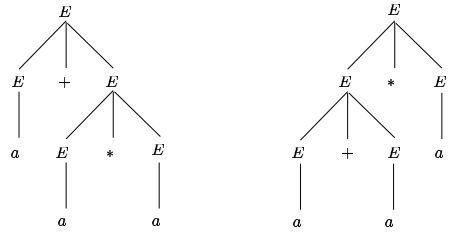
  (iii) jos seuraavassa askelessa on sovellettu produktiota $X_i \to Y_1 Y_2 \ldots Y_l$, niin juuren i:nnelle jälkeläissolmulle tulee l jälkeläistä, joiden nimet vasemmalta oikealle ovat $Y_1, Y_2, \ldots, Y_l$ ja niin edelleen.

- Jäsennysongelman ratkaisuksi katsotaan usein päätösongelman "Päteekö $x \in L(G)$?" ratkaisemisen lisäksi jonkin näistä jäsennysesityksistä tuottaminen x:lle.

- Esim. ohjelmointikielen kääntämisessä jäsennyspuuta käytetään syntaksitarkistusta seuraavissa vaiheissa, eli esim. koodin generoinnissa.

Kieliopin moniselitteisyys

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



Kuva 14: Lauseen $a + a * a$ kaksi erilaista jäsennystä.

- Kontekstiton kielioppi G on *moniselitteinen*, jos jollakin G:n lauseella x on kaksi erilaista G:n mukaista jäsennyspuuta.

- Muuten kielioppi on *yksiselitteinen*.

- Kontekstiton kieli, jonka tuottavat kieliopit ovat kaikki moniselitteisiä, on *luonnostaan moniselitteinen*.

- Esimerkkejä:
  - Kielioppi G'_{expr} on moniselitteinen,
  - kieliopit G_{expr} ja G_{match} ovat yksiselitteisiä.
  - Kieli $L_{\text{expr}} = L(G'_{\text{expr}})$ ei ole luonnostaan moniselitteinen, koska sillä on myös yksiselitteinen kielioppi G_{expr}.
  - Kieli $\{a^i b^j c^k \mid i = j \text{ tai } j = k\}$ on luonnostaan moniselitteinen (todistus sivuutetaan).

Moniselitteisyyden aiheuttama ongelma

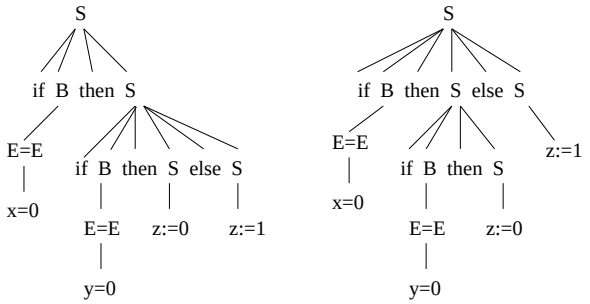
- Oletetaan, että Olpe03-kielessä olisi kaksi eri produktiota **if**-lauseen tuottamiseen, **else**n kanssa ja ilman:

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

- Tarkastellaan seuraavan koodinpätkää:

$$\textbf{if } x = 0 \textbf{ then if } y = 0 \textbf{ then } z := 0 \textbf{ else } z := 1$$

- Kumpaan **if**-lauseeseen **else** liittyy?
- Koodinpätkällä on nyt kaksi erilaista jäsennyspuuta:



- Vasemman puoleiseen jäsennykseen perustuva ohjelma ei selvästikään toimi samalla tavalla kuin oikeanpuoleiseen jäsennykseen perustuva.

  - Eron huomaa esim. tarkastelemalla mitä ohjelmat tekevät kun $x = 1$. Vasemmanpuoleinen ei tee mitään, mutta oikean puoleinen sijoittaa z:n arvoksi 1.

- Ongelma aiheutuu moniselitteisyydestä; alkuperäinen olpe03:n määrittelevä kielioppi ei ole moniselitteinen (pelkät **if**t eivät ole sallittuja) eikä yllä kuvattua **if**-ongelmaa esiinny.

3.3. Rekursiivisesti etenevä jäsentäminen

LL(1)-kielioppi

- Tarkastellaan seuraavaa kielioppia G:

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

- Merkkijono x siis kuuluu kieliopin tuottamaan kieleen, jos se pystytään tuottamaan kieliopin lähtösymbolista lähtien.
- Merkkijonon kuuluminen kieleen voidaan todeta tuottamalla kieliopin mukainen johdos merkkijonolle (tai toteamalla että sellaista ei ole).
- Ongelmana on se, että aina ei ole ilmeistä mikä produktio johdoksen laatimisen missäkin vaiheessa olisi valittava. Kaikkien mahdollisuuksien kokeileminen taas ei ole tehokasta.
- Tarvitaan siis menetelmä, joka osaa tehokkaasti päättää kuuluuko merkkijono kieleen vai ei.

- Muokataan G:stä välikkeen E produktiot "tekijöimällä" (selitetään myöhemmin miten tämä tapahtuu) ekvivalentti kielioppi G':

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

- G' lauseille voidaan helposti muodostaa *vasemmat johdot* suoraan lähtösymbolista E alkaen: Jäsennyksen joka vaiheessa tavoitteena olevan lauseen *seuraava merkki* määrää yksikäsitteisesti sen, mikä produktio valitaan lavennettavana olevaan välikkeeseen.
- 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)
- Huom.: kaikki kontekstittomat kieliopit eivät siis ole LL(1)-muodossa. On myös olemassa kontekstittomia kieliä (esim. luonnostaan moniselitteiset kielet), joille ei ole LL(1)-muodossa olevaa kielioppia.
- Määrittelimme aiemmin vasemmalle (ja oikealle) lineaaristen kontekstittomien kielioppien luokan. Samaan tapaan kuin LL(1), ovat lineaariset kieliopit eräs kontekstittomien kielioppien osajoukko. Lineaariset kieliopit ovat LL(1):n aito osajoukko, joka on aito osajoukko kontekstittomista kieliopeista.

- Tarkastellaan esimerkkinä miten merkkijonon $a+a$ *vasemman johdon* muodostaminen kieliopissa G' etenee.
  - Lähdetään siis liikkeelle lähtösymbolista E. Pidetään kirjaa siitä mikä osa merkkijonoa on vielä "tuottamatta", alussa siis $a + a$.
  - Etenemismahdollisuuksia on yksi: $E \Rightarrow TE'$ ja tuottamatta edelleen $a + a$.
  - T:llä on kaksi produktiota, mutta koska tuotettavan merkkijonon ensimmäisenä merkkinä on a, on valittava produktioista ensimmäinen: $\ldots \Rightarrow TE' \Rightarrow aE'$ ja tuottamatta $+a$.
  - E':lla on kolme produktiota, mutta tuotettavan merkkijonon seuraava käsittelemätön merkki $+$ ohjaa valitsemaan produktioista ensimmäisen: $\ldots \Rightarrow aE' \Rightarrow a + E$ ja tuottamatta a.
  - Nyt etenemismahdollisuuksia on vain yksi: $\ldots \Rightarrow a+E \Rightarrow a + TE'$ ja tuottamatta edelleen a.
  - T:n produktioista valitaan ensimmäinen, koska seuraava käsittelemätön merkki on a: $\ldots \Rightarrow a + TE' \Rightarrow a + aE'$ ja merkkijono tuotettu.
  - Koska koko merkkijono on tuotettu, valitsemme E':n produktioista kolmannen: $\ldots \Rightarrow a + aE' \Rightarrow a + a$

- 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 valmiiksi.

LL(1)-kieliopin rekursiivisesti etenevä jäsentäjä

- Samaan tapaan voidaan tuottaa mikä tahansa kielen $L(G')$ merkkijono: jokaisella askeleella voidaan valittava produktio päätellä tarkastamalla, mikä on tuotettavan merkkijonon seuraava käsittelemätön merkki.
- Tuottamisen idea on mekaaninen, ja se voidaan esittää yleisen algoritmin muodossa, kieliopin *rekursiivisesti etenevänä jäsentäjänä*. Tämä jäsentäjä muodostuu välikkeitä vastaavista rekursiivisista funktioista.
- Funktiolla *getnext* jäsentäjä lukee seuraavan syötemerkin, jokaisessa proseduurissa jäsentäjä tulostaa valitun produktion.

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



```
function E
begin
  writeln(E → TE');
  T;
  E';
end

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

Pääohjelmassa:

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

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

`E` $\rightarrow$ `TE'`
`T` $\rightarrow$ `a`
`E'` $\rightarrow$ `-E`
`E` $\rightarrow$ `TE'`
`T` $\rightarrow$ `(E)`
`E` $\rightarrow$ `TE'`
`T` $\rightarrow$ `a`
`E'` $\rightarrow$ `+E`
`E` $\rightarrow$ `TE'`
`T` $\rightarrow$ `a`
`E'` $\rightarrow \epsilon$
`E'` $\rightarrow \epsilon$.

Tulostus vastaa vasenta johtoa:

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

LL(1)-kielioppien yleinen muoto

- LL(1)-kieliopeissa sallitaan rajoitetussa määrin myös
  - Produktioita, joiden oikeat puolet alkavat välikkeellä, sekä
  - *tyhjentyviä* välikkeitä A, joilla $A \Rightarrow^* \epsilon$.
- Esim. kielen $a^*b \cup c^*d$ tuottava kielioppi:

$$\begin{aligned} S &\rightarrow Ab \mid Cd \\ A &\rightarrow aA \mid \epsilon \\ C &\rightarrow cC \mid \epsilon. \end{aligned}$$

  Kielioppi on LL(1), vaikka ensimmäiseksi sovellettavaa produktiota ei voikaan päätellä pelkästään S:n produktioiden perusteella
- Määritellään tämän tarkentamiseksi seuraavat päätemerkkijoukot.

Apukäsite: päätemerkkijoukot FIRST ja FOLLOW

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

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

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

FIRST-joukkojen määritelmä laajennetaan mielivaltaisille merkkijonoille

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

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

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

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

ja edelleen merkkijonojoukkoihin:

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

- Kieliopin LL(1)-ehto voidaan nyt ilmaista täsmällisesti seuraavalla tavalla:

Määritelmä: Kielioppi on LL(1)-muotoinen, jos sen millä tahansa kahdella samaan välikkeeseen A liittyvällä produktiolla $A \to \omega_1$ ja $A \to \omega_2$, $\omega_1 \neq \omega_2$, on voimassa:

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

LL(1)-kieliopin rekursiivisesti etenevän jäsentäjän muodostaminen yleisesti:

Apurutiinit:

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

Välikettä A vastaava funktio:

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


Tässä lyhennemerkintä "$parse(\omega_i)$" tarkoittaa seuraavalla tavalla muodostettavaa käskyjonoa:
$parse(X_1 \ldots X_k) \equiv parse(X_1); \ldots; parse(X_k)$,
missä

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

*LIITE: FIRST- ja FOLLOW-joukkojen laskeminen

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

Ensin lasketaan FIRST-joukot:

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

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

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

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

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

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

FOLLOW-joukot määritetään FIRST-joukkojen avulla seuraavasti:

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

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

ja lähtösymbolille S lisäksi:

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

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

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

Kielioppien muokkaaminen LL(1)-muotoon

- LL(1)-kieliopit ovat varsin suppea kielioppiluokka
- Seuraavilla muunnoksilla voidaan joitakin "melkein' LL(1)-muotoisia kielioppeja muuntaa tähän 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
  - 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 $\alpha\beta_1$:n, $\alpha\beta_2$:n, ...ja $\alpha\beta_n$:n pisin yhteinen alkuosa
  - huom: esitimme säännön yleisessä muodossa, missä $n \ge 2$, esimerkiksi seuraavassa on kyseessä tapaus $n = 3$
- Esimerkki:
  $$\begin{aligned}E &\to T + E \mid T - E \mid T\\ T &\to a \mid (E)\end{aligned}$$
  - Tehdään välikkeen E produktioille vasen tekijöinti. Nyt E:n produktioilla on yhteinen alkuosa T, siis 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, jolla voidaan jostakin välikkeestä A lähtien tuottaa lauseke $A\gamma$, ei voi täyttää LL(1)-ehtoa.
  - *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.
  - Huom.: ohessa $n \geq 1$, eli "ei-rekursiivisia" produktioita voi olla yksi tai useampia.
- 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:

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

- Koska kaikkia kontekstittomia kieliä ei ole mahdollista saada LL(1)-muotoon, ylläolevat keinot toimivat siis vain joissain tapauksissa.
- Joskus sääntöjä on sovellettava useita kertoja peräkkäin ja eri järjestyksissä ennen kuin kielioppi on muokattu LL(1)-muotoon.
- Huom.: LL(1)-muoto siis tarvitaan rekursiivisesti etenevää jäsennystä varten, ihmisen kannalta saman kielen tuottava ei-LL(1)-muoto on usein paljon havainnollisempi, kuten L_{expr} tapauksessa.
- Voidaankin toimia siten, että esim. ohjelmointikielen määrittely tehdään ei-LL(1)-kieliopilla G, mutta kääntäjän toteutusta varten generoidaan ekvivalentti (eli saman kielen tuottava) LL(1)-muotoinen kielioppi G'.
- "Monimutkaisemman" näköisellä kieliopilla ei tarvitsekaan rasittaa ketään muita kuin kääntäjän tekijää; muut voivat käyttää havainnollisempaa kielioppia G.

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.
  - Kielopin on oltava *Chomskyn normaalimuodossa.*
- Tarkastellaan ensin, miten mv. kontekstiton kielioppi voidaan muuntaa Chomskyn normaalimuotoon.
- Huom.: Chomskyn normaalimuoto ei ole kovin havainnollinen esitystapa ihmisen kannalta; ideana onkin että ihminen käsittelee kontekstittomia kielioppeja muussa, havainnollisemmassa muodossa, mutta CYK-jäsennysalgoritmia varten kielioppi muokataan Chomskyn normaalimuotoon. Tämä muokkaus tapahtuu algoritmisesti.

Chomskyn normaalimuoto

- *Määritelmä:* Kontekstiton kielioppi $G = (V, \Sigma, P, S)$ on *Chomskyn normaalimuodossa*, jos
  - Välikkeistä enintään S on tyhjentyvä (engl. nullable), toisin sanoen, vain S:llä saa olla ϵ-produktio: $S \to \epsilon$
  - Muut produktiot ovat muotoa $A \to BC$ tai $A \to a$, missä A, B ja C ovat välikkeitä ja a on päätemerkki
  - Lisäksi vaaditaan yksinkertaisuuden vuoksi, että lähtösymboli S ei esiinny minkään produktion oikealla puolella.
- Esim. seuraava kielioppi on Chomskyn normaalimuodossa:

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

Muunnos Chomskyn normaalimuotoon

- Idea:
  1. Poistetaan lähtösymboli produktioiden oikealta puolelta
  2. Poistetaan ϵ-produktiot
  3. Poistetaan yksikköproduktiot $A \to B$
  4. Poistetaan päätemerkit muiden kuin $A \to a$ muotoisten produktioiden oikealta puolelta
  5. Poistetaan produktiot $A \to X_1X_2...X_k$, $k \geq 3$.

- Produktioiden oikealla puolella olevien lähtösymbolien poistaminen
  - Jos lähtösymboli S on jonkin produktion oikealla puolella lisätään uusi lähtösymboli S' ja sille produktio $S' \to S$
- ϵ-produktioiden poistaminen
  - Olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Välike $A \in V - \Sigma$ on *tyhjentyvä* (engl. nullable), jos $A \underset{G}{\Rightarrow}^* \epsilon$
  - *Lemma:* Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa enintään lähtösymboli on tyhjentyvä. (Huom.! Lähtösymbolin tyhjentymistä ei voida välttää, jos $\epsilon \in L(G)$.)
    Todistus: Olkoon $G = (V, \Sigma, P, S)$.
    1. Selvitetään ensin G:n tyhjentyvät välikkeet (NULL-joukko):
       Käydään läpi kaikki produktiot $A \in V \backslash \Sigma$, ja merkitään $A \in$ NULL jos $A \Rightarrow^* \epsilon$, eli joko
       - A:lla produktio $A \to \epsilon$, tai
       - A voi tuottaa ϵ:n epäsuorasti, esim. $A \to BC$, ja $B \to \epsilon$ sekä $C \to \epsilon$.

       NULL on siis kieliopin niiden välikkeiden joukko, joista voidaan tuottaa tyhjä merkkijono.
       NULL-joukon muodostamismenetelmä hieman formaalimmin esitettynä:
       (i) asetetaan aluksi

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

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

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

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

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

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

– Esim. Poistetaan ϵ-produktiot seuraavasta kieliopista:

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

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

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

Vastaavasti muuttuvat S:n produktiot, ja tuloksena on:

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

Poistetaan ϵ-produktiot ja otetaan käyttöön uusi lähtösymbooli:

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

– Muokkasimme siis kielioppia hiukan kohti Chomskyn normaalimuotoa: tuloksessamme ei ole enää ϵ-produktioita muilla kuin lähtösymbolilla.

- Yksikköproduktioiden poistaminen
  - Produktio muotoa $A \to B$, missä A ja B ovat välikkeitä, on *yksikköproduktio* (engl. *unit production*)
  - *Lemma:* Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa ei ole yksikköproduktioita.
    Todistus: Olkoon $G = (V, \Sigma, P, S)$.
    1. Selvitetään ensin G:n kunkin välikkeen "yksikköseuraajat".
       Välikkeen A yksilöseuraajat kerätään joukkoon $F(A)$:
       Käydään läpi kaikki produktiot $B \in V \setminus \Sigma$, ja merkitään $B \in$ F(A) jos $A \Rightarrow^* B$, eli joko
       - A:lla produktio $A \to B$, tai
       - A voi tuottaa B:n epäsuorasti, esim. $A \to C$, ja $C \to B$.

       Seuraavassa formaalimmin määritelty menetelmä joukkojen $F(A)$ laskemiseen:

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

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

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

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

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

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

    S':n yksikköseuraajia ovat A, B ja S, sillä ensinnäkin $S' \to S$ ja koska $S \to A$ ja $S \to B$, niin $S' \Rightarrow^* A$ ja $S' \Rightarrow^* B$, eli $F(S') = \{S, A, B\}$, muut yksikköseuraajat $F(S) = \{A, B\}$, $F(A) = F(B) = \emptyset$.
    Korvaamalla yksikköproduktiot edellä esitetyllä tavalla saadaan kielioppi

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

    Eli koska $A, B, S \in F(S')$, saa S' kaikki A:n, B:n ja S:n ei-yksikköproduktiot ja koska $A, B \in F(S)$, saa S kaikki A:n ja B:n ei-yksikköproduktiot.

- Edellisten vaiheiden jälkeen kielioppi on jo lähestulkoon Chomskyn normaalimuodossa, enää ei ole tyhjentyviä välikkeitä eikä yksikköproduktioita. Ongelmana ovat vielä produktiot, joissa on oikealla puolella päätemerkkejä muussa muodossa kuin $A \to a$, kuten $B \to bb$, $A \to aBa$, ja produktiot, joiden oikea puoli koostuu useammasta kuin kahdesta välikkeestä, kuten $A \to BAC$.
- Ensimmäinen näistä ongelmista ratkaistaan seuraavalla tavalla:
  - Lisätään kielioppiin kutakin päätemerkkiä a varten uusi välike C_a ja sille produktio $C_a \to a$
  - Korvataan sitten kussakin (paitsi $A \to a$ muotoisissa) produktiossa kaikki päätemerkit em. uusilla välikkeillä.
- Edellisen esimerkin kielioppi tulee muokatuksi muotoon

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

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

  Ideana seuraavassa on se, että liian pitkä produktio, esim. $A \to BCDE$ voidaan (ottamalla käyttöön uusia välikesymboleja) pilkkoa seuraavasti:

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

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

$$\begin{aligned} A &\to X_1A_1 \\ A_1 &\to X_2A_2 \\ &\vdots \\ A_{k-2} &\to X_{k-1}X_k, \end{aligned}$$

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

- Viimeistellään vielä esimerkkimme:

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

- Olemme viimein saaneet muokattua kieliopin Chomskyn normaalimuotoon.
- Kuten huomataan, tämä muoto on ihmisen kannalta epähavainnollisempi kuin alkuperäinen muoto, mutta kuten aiemmin on todettu, CYK-algoritmilla suoritettava jäsennys vaatii kieliopin muokkaamista Chomskyn normaalimuotoon.

- Vielä kertauksena:
- *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 $A \to a$ produktioiden oikealla puolella olevat päätesymboolit välikkeillä. Tämän jälkeen kaikki G:n produktiot ovat muotoa $A \to a$ tai $A \to X_1 \dots X_k$, $k \geq 3$ (tai $S \to \epsilon$). Viimeksi mainitut poistetaan ed. konstruktion mukaisesti. $\square$
- Huom.: Turhat välikkeet ja produktiot voidaan aina poistaa.

- Esim. Muunnetaan Chomskyn normaalimuotoon kielioppi

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

Tuloksena saadaan kielioppi

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

Cocke-Younger-Kasami-algoritmi

- Perustuu dynaamiseen ohjelmointiin (eli välitulosten taulukointiin).
- Ongelma: Olk. $G = (V, \Sigma, P, S)$ Chomskyn normaalimuodossa (ellei ole, muutetaan ensin Chomskyn normaalimuotoon). Onko $x \in L(G)$ eli $S \underset{G}{\Rightarrow}^* x$?
- Algoritmi:
  1. Jos $x = \epsilon$, niin $x \in L(G) \Leftrightarrow S \to \epsilon$ on G:n produktio.
  2. Muuten merk. $x = a_1 \dots a_n$ ja tarkastellaan x:n osajonot tuottavien välikkeiden joukkoja
  $$N_{i,j} = \{A \in V - \Sigma \mid A \underset{G}{\Rightarrow}^* a_i \dots a_j\}, \qquad 1 \le i \le j \le n$$
  3. $x \in L(G) \Leftrightarrow S \in N_{1,n}$
- Joukkojen $N_{i,j}$ laskeminen
  - muodostetaan kolmiomainen taulukko:

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

  - taulukon paikka $N_{i,j} \sim$ joukko välikesymboleita A, joille $A \underset{G}{\Rightarrow}^* a_i a_{i+1} \dots a_j$

– jokainen diagonaali vastaa tietyn pituisia x:n osajonoja

  * 1. diagonaali $\sim$ välikkeet, jotka tuottavat yhden merkin osajonot, 2. diagonaali $\sim$ välikkeet, jotka tuottavat kahden merkin osajonot jne.

(i) Lasketaan ensin 1. diagonaali, eli **for** $i = 1$ **to** n:

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

(ii) Sitten seuraavat diagonaalit:

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

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

  * Ideana on, että x:n osajono $y = a_i a_{i+1} \dots a_j$ saadaan yhdistämällä joko osajonot a_i ja $a_{i+1} \dots a_j$ tai osajonot $a_i a_{i+1}$ ja $a_{i+2} \dots a_j$ jne. Ne välikkeet, joista nämä osajonot saadaan tuotettua, on jo laskettu, ja ne löytyvät solupareista $(N_{i,i}, N_{i+1,j})$ ja $(N_{i,i+1}, N_{i+2,j})$ jne.
  * Jos siis jokin G:n välike A tuottaa koko osajonon y, täytyy G:ssä olla produktio $A \Rightarrow BC$ siten, että B tuottaa jonkin y:n alkuosan $a_i \dots a_{i+l}$ ja C jonkin y:n loppuosan $a_{i+l+1} \dots a_j$ jollekin $l \le j - i$. Tällöin pätee $B \in N_{i,i+l}$ ja $C \in N_{i+l+1,j}$. Olkoon $i = 3$ ja $j = 7$. Jotta tiedettäisiin onko tällainen välike A olemassa, täytyy tarkastella kaikkia pareja $(N_{3,3}, N_{4,7}), (N_{3,4}, N_{5,7}), (N_{3,5}, N_{6,7}), (N_{3,6}, N_{7,7})$.

- Algoritmin formaali kuvaus saattaa vaikuttaa vaikeaselkoiselta, tarkastellaan siksi seuraavassa algoritmin toimintaa esimerkkien kautta.
- Sovelletaan CYK:iä Chomskyn normaalimuotoiseen kielioppiin

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

syötemerkkijonolla $x = abba$.

Tavoitteenamme siis on 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 ne välikkeet, joista lähtien on mahdollista johtaa syötemerkkijonon positiosta i positioon j oleva osa.

– $N_{1,1}$ 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 vaiheittain siten, että ensin täytetään alkiot yhden pituisten merkkijonojen tuottamiseen, tämän jälkeen kahden pituiset, ...

Ensimmäinen vaiheessa on selkeä: koska ainoastaan $A \to a$, $N_{1,1} = N_{4,4} = \{A\}$ ja vastaavasti koska ainoastaan $B \to b$, $N_{2,2} = N_{3,3} = \{B\}$. Eli 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 alkioiden $N_{1,2}, N_{2,3}$ ja $N_{3,4}$ arvojen selvittäminen.

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

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

Koska $S \to AB$ ja $A \to AB$ saamme siis $N_{1,2} = \{S, A\}$.

Joukon $N_{1,2}$ alkiot vastaavat siis niitä välikkeitä, joista pystytään tuottamaan merkkijono ab, eli $S \Rightarrow^* ab$ ja $A \Rightarrow^* ab$.

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

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

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

Ja koska joukon $N_{2,3}$ alkiot vastaavat niitä välikkeitä, joista pystytään tuottamaan merkkijono bb, ja $N_{2,3} = \emptyset$, tarkoittaa tämä sitä, ettei kieliopin mistään välikkeestä pystytä johtamaan merkkijonoa bb.

Vastaavasti, $S \to BA$ ja $B \to BA$ joten $N_{3,4} = \{S, B\}$. 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.

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

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

Koska $S \to AB$ ja $A \to AB$, ja $A \in N_{1,2}$ $B \in N_{3,3}$ saamme siis $N_{1,3} = \{S, A\}$.

Joukon $N_{1,3}$ alkiot vastaavat siis niitä välikkeitä, joista pystytään tuottamaan merkkijono *abb*, eli $S \Rightarrow^* abb$ ja $A \Rightarrow^* abb$.

$N_{2,4}$ arvo:

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

$N_{2,4} = \emptyset$, sillä minkään produktion oikeana puolena ei ole välikkeitä BB tai BS. Kieliopin mistään välikkeestä ei pystytä siis johtamaan 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 alkioiden $N_{1,4}$ arvon selvittäminen.

$N_{1,4}$ arvo lasketaan seuraavasti:

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

Koska $S \to AB$ ja $A \to AB$, ja $A \in N_{1,2}$ $B \in N_{3,4}$, saamme siis $N_{1,4} = \{S, A\}$.

Taulukko on valmiina:

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

Joukon $N_{1,4}$ välikkeet vastaavat kieliopin niitä välikkeitä, joista pystytään johtamaan merkkijono *abba*, ja koska lähtösymboli S kuuluu joukkoon $N_{1,4}$, kuuluu *abba* kieliopin tuottamaan kieleen.

- Toisena esimerkkinä sovelletaan CYK:iä Chomskyn normaalimuotoiseen kielioppiin

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

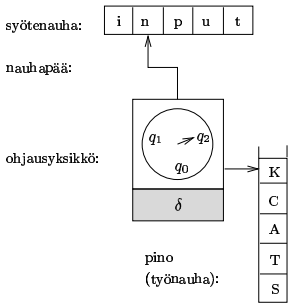
syötemerkkijonolla $x = baaba$:

Taulukon täyttäminen etenee samoin kuin edellisessä esimerkissä, ja 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 15: Pinoautomaatti.

- Pinoautomaatti on äärellinen automaatti, johon on lisätty yksi mielivaltaisen kokoinen pino työnauhaksi.
- Ts. automaatti voi lukea ja kirjoittaa vain työnauhan toista päätä, ja päästäkseen lukemaan aiemmin kirjoittamiaan merkkejä sen täytyy pyyhkiä viimeisin merkki pois.
- Työnauha antaa automaatille "muistin", jonka avulla voidaan välttää äärellisen automaatin (joitakin) rajoituksia.

- *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*;
  - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$ 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 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 $(q_0, w_0, \alpha_0), \ldots, (q_n, w_n, \alpha_n)$, $n \geq 0$ siten, että
  $(q, w, \alpha) = (q_0, w_0, \alpha_0) \underset{M}{\vdash} \cdots \underset{M}{\vdash} (q_n, w_n, \alpha_n) = (q', w', \alpha')$.
- Pinoautomaatti M *hyväksyy* merkkijonon $x \in \Sigma^*$, jos

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

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

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

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

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

  missä

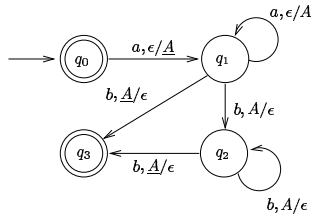
  $$\begin{aligned}
  \delta(q_0, a, \epsilon) &= \{(q_1, \underline{A})\}, \\
  \delta(q_1, a, \epsilon) &= \{(q_1, A)\}, \\
  \delta(q_1, b, A) &= \{(q_2, \epsilon)\}, \\
  \delta(q_1, b, \underline{A}) &= \{(q_3, \epsilon)\}, \\
  \delta(q_2, b, A) &= \{(q_2, \epsilon)\}, \\
  \delta(q_2, b, \underline{A})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

  Esimerkiksi 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 $aabb \in L(M)$.

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



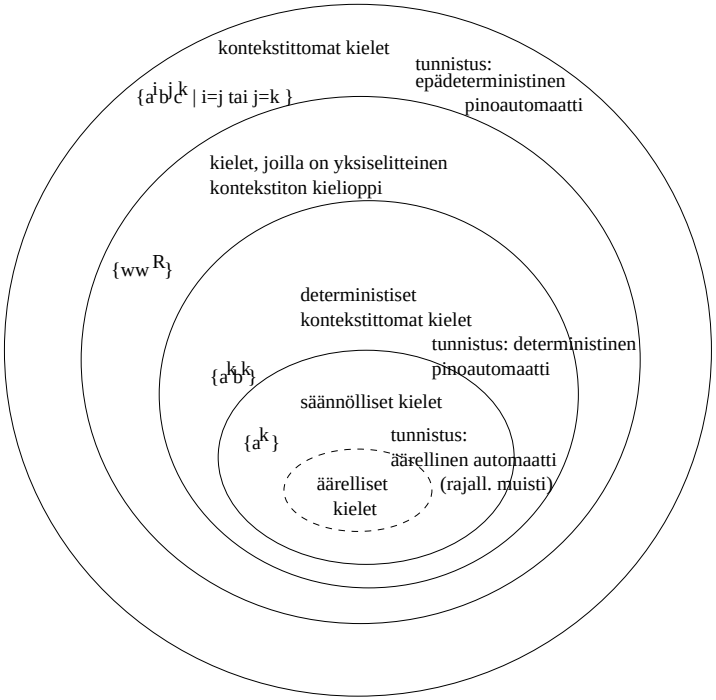
Kuva 16: Kielen $\{a^k b^k \mid k \geq 0\}$ tunnistava pinoautomaatti, jossa pinoaakkosto on $\{A, \underline{A}\}$.

- Tässä on käytetty tilasiirtymäkaaviotyyppistä esitystä:
  - Siirtymäehto muotoa $a_i, \gamma / \gamma'$, missä
    - automaatti lukee merkkijonosta merkin a_i ja
    - pinosta merkin γ sekä
    - kirjoittaa pinoon merkin γ'.
  - "Push"-operaatio: ei lue pinosta merkkiä, mutta kirjoittaa uuden merkin γ' pinon päälle: $a_i, \epsilon / \gamma'$.
  - "Pop"-operaatio: lukee pinon päällimmäisen merkin γ, mutta ei kirjoita mitään pinoon: $a_i, \gamma / \epsilon$.
  - Jos syötemerkki $a_i = \epsilon$, niin siirtymä syötemerkkiä lukematta - voi silti lukea tai kirjoittaa pinomerkkejä!
- *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'), jolla
$$(q, w, \alpha) \underset{M}{\vdash} (q', w', \alpha').$$
- 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.
- Tämä on tärkeä kieliluokka, sillä siihen kuuluvat kielet voidaan jäsentää oleellisesti tehokkaammin kuin yleiset, mahdollisesti epädeterministisen pinoautomaatin vaativat kontekstittomat kielet.
- Esim. LL(1)-muotoa olevat kontekstittomat kielet ovat tunnistettavissa deterministisillä pinoautomaateilla.

• Kielten keksinäisiä suhteita kuvaa seuraava kuva:



3.6. Kontekstittomien kielten sovelluksia

1. Jäsentäjät
   - Kuten jo totesimme olpe03:n määritelmän yhteydessä, ohjelmointikielten syntaksi voidaan määritellä kontekstittomana kielioppina,
   - ja kieliopista saadaan muodostettua kielen jäsentäjä eli kääntäjän osa, joka tutkii ohjelman rakenteen ja esittää sen jäsennyspuuna.
   - aputyökalu: YACC
     – generoi jäsentäjän annetusta kontekstittomasta kieliopista
2. Dokumentin rakenteen kuvaus
   - ns. ”mark-up”-kielet, kuten HTML ja XML
   - kielen merkkijonot dokumentteja, joissa tiettyjä kielen semantiikkaa kuvaavia merkkejä (tags) — esim. <OL> ja </OL> $\sim$ järjestetty lista
   - HTML: laillisen HTML-dokumentin kuvaus ja dokumentin prosessoinnin ohjaus
   - XML: kuvaa tekstin semantiikan — esim. <ADDR>Teollisuuskatu 23</ADDR> kertoo, että osajono on osoite

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

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

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

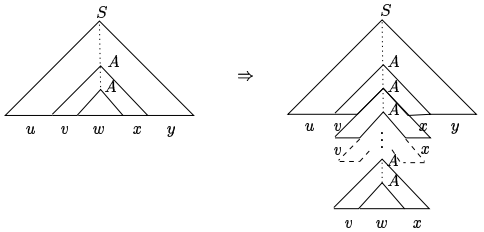
  (iii) $uv^iwx^iy \in L$ kaikilla $i = 0, 1, 2, \ldots$

  Huom.: Taas vain äärettömät kielet ovat kiinnostavia.

Todistus:

- Olk. $G = (V, \Sigma, P, S)$ Chomskyn normaalimuotoinen kielioppi L:lle. Tällöin missä tahansa G:n jäsennyspuussa, jonka korkeus (= pisimmän juuresta lehteen kulkevan polun pituus) on h, on enintään 2^h lehteä. Ts. minkä tahansa merkkijonon $z \in L$ jokaisessa jäsennyspuussa on polku, jonka pituus on vähintään $\log_2 |z|$.
- Olk. $k = |V - \Sigma|$ kieliopin G välikkeiden määrä. Asetetaan $n = 2^{k+1}$. Tarkastellaan jotakin $z \in L$, $|z| \geq n$, ja sen jotakin jäsennyspuuta
- $\Rightarrow$ puussa on polku, jonka pituus on $\geq k+1$. Tarkastellaan tämän polun "alinta" $k+1$:n pituista osaa. Tällä polulla, jossa on $k+1$ välikettä vastaavaa solmua ja välikkeitä on k kappaletta, on siis jonkin välikkeen toistuttava. Olkoon A joku polun toistuva välike (ks. kuva 17).
- Merkkijono z voidaan nyt osittaa $z = uvwxy$, missä w on A:n alimmasta ilmentymästä tuotettu osajono ja vwx seuraavaksi ylemmästä A:n ilmentymästä tuotettu osajono; osajonot saadaan johdosta

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



Kuva 17: Kontekstittoman kielen merkkijonon pumppaus.

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

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

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

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

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

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

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

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

Tällöin merkkijono vx ei voi sisältää sekä a:ta, b:tä että c:tä. Merkkijonossa $uv^0wx^0y = uwy$ on siten ylijäämä jotakin merkkiä muihin merkkeihin nähden eli $uwy \notin L$. Ristiriita. $\square$

4. Kontekstittomien kielten tuolla puolen

- Osoitimme luvussa 3.7 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 $L(G) = ABC$.
- Sallitaan produktiot muotoa $V^+ \times V^*$, eli produktion vasemmalla puolella saa olla mielivaltainen merkkijono.

  Esim: $Aa \Rightarrow aab$, $CB \Rightarrow A$.
- Merkkijono $\gamma \in V^*$ *tuottaa* t. *johtaa suoraan* merkkijonon $\gamma' \in V^*$ kieliopissa G, merk.

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

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

  Esim: $CBa \underset{G}{\Rightarrow} Aa \underset{G}{\Rightarrow} aab$, jos kieliopissa G yllä mainitut produktiot.
- Kielioppeja joissa on yleistä muotoa omaavia produktioita kutsutaan *rajoittamattomiksi kieliopeiksi.*
- Rajoittamattomilla kieliopeilla tuotettavia kieliä sanotaan *rajoittamattomiksi kieliksi.*

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

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

Kielioppi tuottaa lauseen seuraavalla tavalla:

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

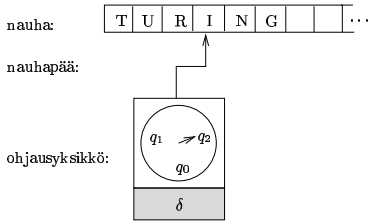
Esim. lauseen *aabbcc* johto:

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

Rajoittamattomien kielten tunnistaminen

- *Turingin kone* (1935–36, Alan M. Turing 1912–54) on kuin äärellinen automaatti, jolla on toiseen suuntaan rajoittamattoman pitkä työnauha, jota voi lukea ja kirjoittaa merkki kerrallaan.
- Aluksi nauhalla on syötemerkkijono (ja loppu nauhasta tyhjää), nauhapää osoittaa ensimmäistä nauhapään paikkaa ja kone käynnistetään alkutilassa q_0.
- Kullakin laskenta-askeleella se lukee nauhapään kohdalla olevan merkin, päättää siirtymäfunktion mukaisesti uuden tilansa, kirjoittaa nauhapään kohdalla uuden merkin ja siirtää nauhapäätä yhden askeleen vasemmalle tai oikealle (ensimmäisen paikan vasemmalle puolelle ei voi kuitenkaan mennä).
- Koneella on hyväksyvä lopputila q_{yes} ja hylkäävä lopputila q_{no} (kun kyseessä on kielen tunnistaminen, jota tässä käsittelemme). Kone pysähtyy, kun se saavuttaa lopputilan.

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



Kuva 18: 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 $\Sigma \subseteq \Gamma$
  - $\delta : Q \times (\Gamma \cup \{>, <\}) \to Q \times \Gamma \times \{L, R\}$ on koneen siirtymäfunktio
  - q_0 on koneen alkutila
  - q_{yes} on koneen hyväksyvä lopputila
  - q_{no} on koneen hylkäävä lopputila.

- Siirtymän
$$\delta(q, a) = (q', b, L)$$
tulkinta: Ollessaan tilassa q ja lukiessaan nauhapäänsä kohdalta syötemerkin a, Turingin kone siirtyy tilaan q', kirjottaa nauhapäänsä kohdalle merkin b ja siirtää nauhapäätä askeleen vasemmalle (vastaavasti, R tarkoittaa nauhapään siirtoa oikealle).

- Symboli $>$ tarkoittaa syötemerkin vasenta ja $<$ syötenauhan oikeaa reunaa.

- Turingin koneen tilanne on $(q, \alpha\underline{a}\beta)$ missä q on koneen senhetkinen tila, a on nauhapään kohdalla oleva merkki, α naupäästä vasemmalle olevat nauhan merkit ja β nauhapäästä oikealle oleva nauhan merkit.

- Koneen alkutilanne syötteellä $x = x_1 \dots x_n$ on $(q_0, \underline{x_1} x_2 \dots x_n)$. Voidaan ajatella, että x_1:n vasemmalla puolella on merkki $>$ ja x_n:n oikealla puolella on ääretön määrä merkkejä $<$.

- Ei anneta tässä formaalia määritelmää sille miten Turingin koneen tila johtaa seuraajatiloihin; asiaa valaisee seuraavan sivun esimerkki.

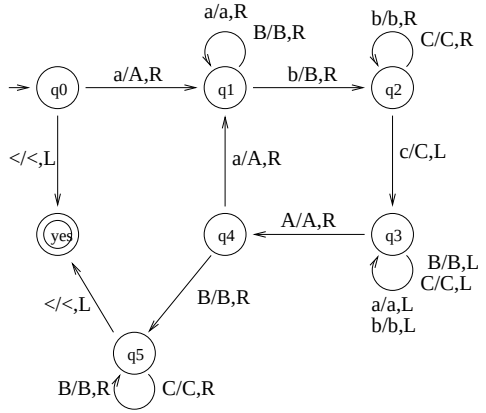
- Jos kone pysähtyy syöttellä x tilaan q_{yes}, x kuuluu koneen M tunnistamaan kieleen $L(M)$, jos päättyy tilaan q_{no} tai jää ikuiseen silmukkaan, niin $x \notin L(M)$.

- Huom.: Turingin kone siis pysähtyy välittömästi tilaan q_{yes} tai q_{no} mennessään, toisin kuin esim. äärellinen automaatti,

joka lukee aina syötteensä loppuun asti.

- *Lause:* Kieli on rajoittamaton, jos ja vain jos se voidaan tunnistaa Turingin koneella.
Todistus: Sivuutetaan. □

- Kielen $ABC = \{a^k b^k c^k \mid k \geq 0\}$ tunnistava Turingin kone:



- Esim, syötteellä $aabbcc$ laskenta etenee seuraavasti

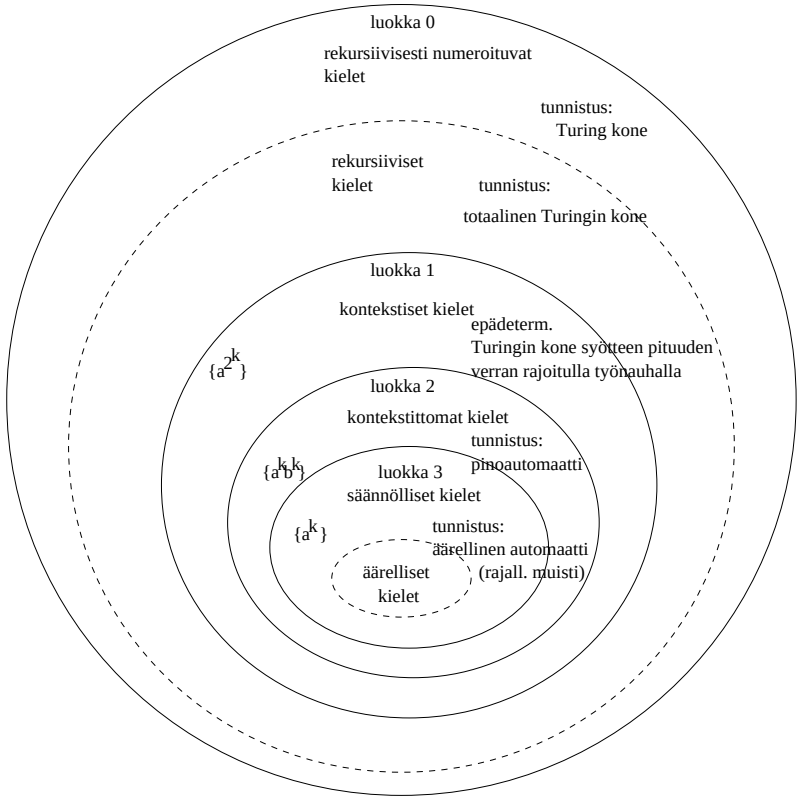
$$
\begin{aligned}
(q0, \underline{a}abbcc) &\vdash_M \\
(q1, A\underline{a}bbcc) &\vdash_M \\
(q1, Aa\underline{b}bcc) &\vdash_M \\
(q2, AaB\underline{b}cc) &\vdash_M \\
(q2, AaBb\underline{c}c) &\vdash_M \\
(q3, AaB\underline{b}Cc) &\vdash_M
\end{aligned}
$$

$$
\begin{aligned}
(q3, Aa\underline{B}bCc) &\vdash_M \\
(q3, A\underline{a}BbCc) &\vdash_M \\
(q3, \underline{A}aBbCc) &\vdash_M \\
(q4, A\underline{a}BbCc) &\vdash_M \\
(q1, AA\underline{B}bCc) &\vdash_M \\
(q1, AAB\underline{b}Cc) &\vdash_M \\
(q2, AABB\underline{C}c) &\vdash_M \\
(q2, AABBC\underline{c}) &\vdash_M \\
(q3, AABB\underline{C}C) &\vdash_M \\
(q3, AAB\underline{B}CC) &\vdash_M \\
(q3, AA\underline{B}BCC) &\vdash_M \\
(q3, A\underline{A}BBCC) &\vdash_M \\
(q4, AA\underline{B}BCC) &\vdash_M \\
(q5, AAB\underline{B}CC) &\vdash_M \\
(q5, AAB\underline{B}CC) &\vdash_M \\
(q5, AABB\underline{C}C) &\vdash_M \\
(q5, AABBC\underline{C}) &\vdash_M \\
(q5, AABBCC\underline{\leq}) &\vdash_M \\
(q_{yes}, AABBC\underline{C}) &
\end{aligned}
$$

Kielien luokittelua

- Turingin koneella tunnistettavien kielten luokkaa kutsutaan myös *rekursiivisesti lueteltaviksi kieliksi*.
- Turingin konetta M joka pysähtyy kaikilla syötteillään sanotaan *totaaliseksi*.
- Totaalisella Turingin koneella tunnistettavien kielten luokkaa kutsutaan myös *rekursiivisiksi kieliksi*.
- Rajoittamattomien kielten luokka on siis sama kuin rekursiivisesti lueteltavien kielten luokka.
- Päätösongelma on *osittain ratkeava* (engl. *semidecidable*), jos sitä vastaava formaali kieli on rekursiivisesti numeroituva 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–).
- Oheisessa kuvassa on Chomskyn kielihierarkia täydennettynä rekursiivisten kielten luokalla. Kuvassa näkyy myös ns.

kontekstisten kielten luokka, jota ei tällä kurssilla kuitenkaan käsitelty.



- Turingin koneesta on olemassa monta eri variaatiota, jotka tunnistavat samat kielet kuin tässä esitelty ”perusmalli”.
- Tärkeä tällainen variaatio on *epädeterministinen* Turingin kone, jossa siirtymäfunktio on joukkoarvoinen, kuten epädeterministisillä äärellisillä automaateilla.
- Turingin koneella voidaan sallia esimerkiksi useampia työnauhoja ilman että koneen ilmaisuvoima muuttuu. Turingin konetta voidaan myös ajatella päätösongelmia laajemmassa kontekstissa. Jonkin koneen nauhoista voidaan ajatella olevan ”tulostusnauha”, jolle kone on pysähtyessään laskenut syötettään vastaavan funktion arvon.
- On olemassa myös joukko muita laskennan formalisointeja, jotka ovat osoittautuneet laskentavoimaltaan ekvivalenteiksi (mm. ohjelmointikielet).
- *Church-Turingin teesi* on seuraava hypoteesi: kaikki mekaanisesti (siis algoritmilla eli tietokoneella) ratkeavat ongelmat = Turingin koneella ratkeavat ongelmat (Alonzo Church 1903–95).
- Eri laskennan mallien tai ohjelmointikielten ekvivalenssi voidaan todistaa kirjoittamalle näiden välille kääntäjät/tulkit, jotka on toteutettu yhtä ”vahvoilla” ohjelmointikielillä.
- Luvun 4 asioita käsitellään tarkemmin kurssilla *Laskennan teoria* sekä matematiikan laitoksen kurssilla *Laskettavuuden teoria*.