582206 Laskennan mallit

luennot syksyllä 2009*, periodit I–II

Juha Kärkkäinen

- tietojenkäsittelytieteen aineopintokurssi, 6 op, pääaineopiskelijoille pakollinen
- esitietoina *Tietorakenteet* (ja sen esitiedot, erityisesti *Johdatus diskreettiin matematiikkaan*)
- opittuja asioita sovelletaan ja ymmärrystä syvennetään mm. algoritmeihin ja koneoppimiseen liittyvillä kursseilla

*Perustuu Jyrki Kivisen edellisvuosien luentoihin

Annettava opetus, kurssin suorittaminen

- luentoja 2 tuntia viikossa
- harjoituksia 2 tuntia viikossa; tarkemmin seuraavilla kalvoilla
- kaksi kurssikoetta (kummankin periodin tenttiviikolla)
- kurssikirja Sipser: *Introduction to the Theory of Computation*, luvut 1–4 (ja pieniä osia myöhemmistä luvuista)
- (ei luentoja eikä harjoituksia tenttiviikoilla eikä väliviikolla)
- **Huom.** kurssin arvioitu työmäärä on 160 tuntia (eli syksyn aikana keskimäärin yli 10 tuntia viikossa)
- maksimipisteet harjoituksista $10 + 2$ (tarkemmin seuraavilla sivuilla), tenteistä $24 + 24$, yhteensä 60; hyväksymisraja n. 30, arvosanan 5/5 raja n. 50

Laskuharjoitukset

Paritonnumeroisilla harjoituskerroilla (periodin I viikolta 1 alkaen joka toinen kerta) käsitellään *kotitehtäviä*:

- laitoksen "perinteinen" harjoitusmuoto: opiskelijat ratkaisevat tehtävät etukäteen, laskaritilaisuudessa ratkaisut kirjataan kalvolle ja esitetään ryhmätyönä
- 4 tehtävää per viikko, yhteensä siis 24 tehtävää
- kotitehtävien tekeminen on **pakollista**: ainakin 8 tehtävää vaaditaan
- tehdyistä kotitehtävistä saatavat pisteet (max. 10):

tehtäviä	alle 8	8	9	10	11	12	14	15	16	17	18	20
pisteitä	hylätty	0	1	2	3	4	5	6	7	8	9	10

Käytännössä kurssin menestyksekäs opiskeleminen edellyttää huomattavasti minimin ylittävää tehtävien ratkaisemista.

Parillisnumeroisilla harjoituskerroilla (periodin I viikolta 2 alkaen joka toinen kerta) *perustehtäviä* ja *yhteistehtäviä*:

Perustehtävät ovat suoria sovelluksia luennoilla esitetyille tekniikoille. Tarkoitus on, että kukin opiskelija *ratkaisee ne etukäteen* ja ne käsitellään laskaritilaisuudessa melko lyhyesti (ellei ilmene keskustelunaihetta).

Yhteistehtävien tarkoituksena on harjoitella laskarinpitäjän avustuksella ongelmanratkaisua. Tehtäviin on syytä *tutustua etukäteen* ja varmistaa, että muistaa asiaan liityvät määritelmät jne., mutta varsinainen ratkaisun miettiminen on tarkoitus jättää laskaritilaisuuteen.

Parillisnumeroisista harjoituskerroista saa pisteitä osallistumiskertojen mukaan: yksi osallistuminen 1 piste, kaksi tai useampia osallistumisia 2 pistettä.

Sisällysluettelo

0. Johdanto: yleiskatsaus, esitietojen kertaus
1. Säännölliset kielet: äärelliset automaatit, säännölliset lausekkeet
2. Yhteydettömät kielet: pinoautomaatit, yhteydettömät kieliopit
3. Ratkeavuus: Turingin koneet, ratkeavat ja ratkeamattomat ongelmat; laskennan vaativuus (lyhyesti)

- Äärelliset automaatit, pinoautomaatit js Turingin koneet ovat matemaattisia abstraktioita *laskentalaitteille*.
- Säännöllisyys, yhteydettömyys ja ratkeavuus ovat *laskentaongelmien vaikeusluokituksia*.

Kurssin peruskysymys on

> Mitkä periaatteelliset seikat rajoittavat automaattista laskentaa (ja siis erityisesti nykyisiä tietokoneita)?

Kurssilla opitaan, miten tällaiset asiat voidaan esittää täsmällisesti.

- Automaatteja tarkastelemalla opimme, miten laskentaan liittyviä ilmiöitä *mallinnetaan* ja analysoidaan.
- Vaikeusluokitusten avulla voimme *täsmälleen* rajata, millaisia asioita esim. äärellinen automaatti osaa laskea.
- Erityisesti ratkeamattomat ongelmat ovat sellaisia, ettei niitä voi ratkaista *millään* nykytietämyksen mukaisella laskentalaitteella.

Kurssin asioilla on myös käytännön sovelluksia:

- Äärelliset automaatit ja säännölliset kielet ovat tärkeitä merkkijonoalgoritmeissa
- erilaisia automaatteja käytetään hajautettujen järjestelmien mallintamisessa
- yhteydettömät kielet ovat tärkeitä luonnollisen kielen ja ohjelmointikielten käsittelyssä
- ratkeamattomiin ongelmiin voi törmätä ainakin tekoälyssä ja formaalissa verifioinnissa (jos ei osaa varoa).

Tällä kurssilla sovelluksiin ei ehditä kuin hyvin pintapuolisesti.

Tavoitteet: kurssin jälkeen opiskelija

$*$ tuntee automaattien ja kielioppien tärkeimmät luokat ja niiden yhteydet

$+$ osaa muodostaa automaatteja ja kielioppeja annetuille yksinkertaisille kielille

$+$ osaa soveltaa kurssilla esitettyjä muunnoksia automaatti- ja kielioppiformalismien välillä

$*$ ymmärtää Churchin-Turingin teesin ja sen seuraukset

$*$ tuntee keskeisimmät ratkeamattomuustulokset ja niiden seuraukset

$+$ osaa laatia yksinkertaisia ratkeamattomuustodistuksia

$\times$ osaa formalisoida laskentaan liittyviä ongelmia ja esittää formalismeihin perustuvia täsmällisiä argumentteja.

Kurssilla on teknisiä ($+$), käsitteellisiä ($*$) ja metodisia ($\times$) tavoitteita. Listaan kannattanee palata kurssin aikana.

0. Johdanto

Erästä suosittua määritelmää mukaillen tietojenkäsittelytiede tutkii,

1. *millaiset* tietojenkäsittelytehtävät on *mahdollista* automatisoida ja
2. *miten* tämä automatisointi tulisi suorittaa.

Tietojenkäsittelytieteen peruskursseilla keskitytään yleensä konstruktiiviseen puoleen eli kysymykseen 2. Tällä kurssilla tarkastellaan kysymystä 1.

Osoittautuu (yllättäen? odotetusti?), että tosiaan on olemassa tehtäviä, joita ei *periaatteessakaan* voi automatisoida. Tämä tarkoittaa paljon enemmän kuin pelkästään, että automatisointia ei (vielä?) *osata* tehdä. Tällaisten väittämien perusteleminen edellyttää tietysti, että käsitteet määritellään huolellisesti.

Laskettavuuden teoria

Edellisellä sivulla mainittu automatisointi tarkoittaa tämän kurssin kannalta *algoritmin* esittämistä. Intuitiivisesti algoritmi kuvaa tietojenkäsittelyprosessin niin täsmällisesti, että se voidaan tämän kuvauksen perusteella suorittaa mekaanisesti (ilman "luovaa ajattelua").

Mekaanisen laskennan tarkemmaksi määrittelemiseksi, eli algoritmikäsitteen matemaattiseksi formalisoimiseksi, on kaksi lähestymistapaa:

1. Lähdetään liikkeelle tyhjästä ja mietitään, mitä voidaan pitää mekaanisena laskentana.
2. Otetaan lähtökohdaksi nykyiset tietokoneet, jotka selvästi suorittavat mekaanista laskemista, ja pelkistetään pois epäolennaisuudet.

Koska mekaaninen laskenta on keskeistä matematiikan perusteiden tarkastelussa, matemaatikot ja loogikot miettivät asiaa paljon 1930-luvulla. He sovelsivat luonnollisesti lähestymistapaa 1.

Jos taas halutaan soveltaa tuloksia käytännön tietojenkäsittelyyn, lähestymistapa 2 tuntuisi lupaavammalta.

Onneksi osoittautuu, että lähestymistavat 1 ja 2 johtavat samaan algoritmikäsitteen formalisointiin.

Siis matemattista logiikkaa ja tietokoneita koskevilla periaatteellisilla rajoituksilla on syvällinen yhteys.

Laskettavuuden teoria tarkastelee näitä rajoituksia, eli sitä millaisille ongelmille on olemassa ratkaisualgoritmi.

Laskennan vaativuus

Laskettavuuden teoriassa algoritmeja tarkastellaan olettaen, että käytettävissä on mielivaltaisen paljon resursseja (laskenta-aikaa, muistia, ...).

Laskennan vaativuusteoriassa kysytään, millaisille ongelmille on olemassa *tehokas* algoritmi. Yleisimmin tämä tarkoittaa, että laskenta-ajan pitäisi skaalautua kohtuullisesti (esim. polynomisesti) syötteen koon suhteen.

Jos tehokasta algoritmia ei ole, joudutaan miettimään korvikkeita: *satunnaisalgoritmit*, *approksimointialgoritmit*, rajoittuminen helppoihin *erikoistapauksiin*, ...

(Tällä kurssilla ei juuri käsitellä laskennan vaativuuskysymyksiä.)

Automaattiteoria

Kun on saatu valmiiksi abstrakti malli tietokoneelle, voidaan kysyä, mikä muuttuu, jos mallista jätetään jokin piirre pois. Rajoitettujen mallien tarkasteleminen auttaa ymmärtämään yleisempiä malleja.

Äärellinen automaatti on hyvin yksinkertainen (abstrakti) laskentalaite, jolla kuitenkin voi tehdä mielenkiintoisia asioita. Teoreettisen mielenkiinnon lisäksi se on hyödyllinen käytännössä ohjelmointi- ja mallinnustekniikkana.

Yhteydettömät kieliopit ovat hieman äärellisiä automaatteja ilmaisuvoimaisempi mekanismi, jolla on tärkeitä sovelluksia esim. ohjelmointikielten määrittelemisessä ja kääntämisessä ja luonnollisen kielen mallintamisessa.

Kurssilla aloitamme äärellisistä automaateista, jotka yksinkertaisuutensa takia ovat hyvä paikka harjoitella tarvittavia ajattelutapoja. Etenemme sitten yhteydettömien kielioppien kautta yleiseen laskettavuuteen.

Matemaattisia perustietoja [Sipser luku 0.2]

Logiikan, joukko-opin ja verkkojen peruskäsitteet edellytetään tutuiksi kursseilta *Johdatus diskreettiin matematiikkaan* ja *Tietorakenteet*. Kurssilla tehdään jonkin verran induktiotodistuksia, mutta ne eivät ole keskeisessä roolissa.

Jatkossa ajattelemme yleensä, että algoritmin (tms.) syötteenä on jokin *merkkijono*. Tätä varten tarvitsemme ensin *aakkoston*, joka voi olla mikä tahansa äärellinen joukko.

Esimerkkejä tyypillisistä aakkostoista:

$$\begin{array}{rcl} & & \text{ASCII-merkistö} \\ \Sigma_1 & = & \{\,0,1\,\} \\ \Sigma_2 & = & \{\,\mathrm{a},\mathrm{b},\mathrm{c},\ldots,\mathrm{z}\,\} \\ \Gamma_1 & = & \{\,\mathrm{A},\mathrm{C},\mathrm{G},\mathrm{T}\,\} \\ \Gamma_2 & = & \{\,\mathrm{HiiriVasen},\mathrm{HiiriKeski},\mathrm{HiiriOikea}, \\ & & \quad \mathrm{ShiftHiiriVasen},\mathrm{ShiftHiiriKeski},\mathrm{ShiftHiiriOikea}\,\} \\ \Gamma_3 & = & \{\,\mathrm{Renault},\mathrm{Ferrari},\mathrm{McLaren},\mathrm{muu}\,\} \end{array}$$

Kuten yleensäkin, $|\cdot|$ tarkoittaa alkioiden lukumäärää; siis yllä $|\Gamma_1| = 4$.

Aakkoston Σ *merkkijono* on mikä tahansa *äärellinen* jono joukon Σ alkioita (eli merkkejä eli symboleita). Esim. 00010 on aakkoston Σ_1 merkkijono. Merkkijonon z pituutta merkitään $|z|$; siis $|00010| = 5$. Symbolilla ε merkitään minkä tahansa aakkoston *tyhjää merkkijonoa*, jonka pituus on nolla.

Merkkijonon $w = w_1 w_2 \ldots w_n$ *käänteismerkkijono* on $w^{\mathcal{R}} = w_n w_{n-1} \ldots w_1$.

Merkkijono v on merkkijonon w *osajono*, jos v esiintyy *yhtenäisenä* osana merkkijonossa w. Siis `acad` on merkkijonon `abracadabra` osajono, mutta `rcd` ei ole.

Merkkijonojen $v = v_1 \ldots v_n$ ja $w = w_1 \ldots w_m$ *konkatenaatio* on $vw = v_1 \ldots v_n w_1 \ldots w_m$. Siis v on merkkijonon w osajono, jos $w = xvy$ joillakin merkkijonoilla x ja y.

Kun $k \in \mathbb{N}$, niin merkkijonon w konkatenaatiota itsensä kanssa k kertaa merkitään w^k. Siis $|w^k| = k|w|$ ja erityisesti $w^0 = \varepsilon$.

Mitä tahansa (äärellistä tai ääretöntä) joukkoa annetun aakkoston merkkijonoja sanotaan (*formaaliksi*) *kieleksi*. Merkintä Σ^* tarkoittaa kieltä, jossa on *kaikki* aakkoston Σ merkkijonot.

Esimerkki 0.1: Olkoon R aakkoston $\{\,+,-,0,\ldots,9\,\}$ kieli, joka koostuu yksinkertaisista kokonaislukuvakioista (esim. hieman yksinkertaistetussa Java-kielessä). Siis $0 \in R$, $+7326 \in R$ ja $-32 \in R$, mutta $2+3 \notin R$.

Kieli R on esimerkki *säännöllisestä* kielestä (luku 1). □

Esimerkki 0.2: Olkoon S aakkoston $\{\,+,-,*,(,),0,\ldots,9\,\}$ kieli, joka koostuu laillisista kokonaislukulausekkeista. Esim. $1+1 \in S$ ja $(1+2+3)*4-5 \in S$, mutta $(1+2)) \notin S$ ja $3+ \notin S$.

Kieli S on esimerkki *yhteydettömästä* kielestä (luku 2). □

Esimerkki 0.3: Muodostukoon ASCII-aakkoston kieli T niistä C-kielisistä ohjelmista, jotka tyhjällä syöttötiedostolla joutuvat ikuiseen silmukkaan.

Kieli T on esimerkki *ratkeamattomasta* kielestä (luku 3). □

Jatkossa tarkastellaan laskentalaitteita, joiden syöte on mikä tahansa aakkoston merkkijono mutta tuloste on kyllä tai ei. Tällainen laskenta voidaan samaistaa formaalin kielen kanssa (niiden syötteiden joukko, joilla vastaus on "kyllä").

Pitäytyminen kyllä/ei-vastauksiin ei itse asiassa ole oleellinen rajoitus. Tarkastellaan yleisempää laskentaa, jossa myös tuloste on merkkijono. Syöte voidaan aina ajatella koodatuksi yhdeksi merkkijonoksi, vieläpä *binääriaakkostossa* $\{0,1\}$; samoin tuloste. Laskentaongelman prototyyppi on siis

> **Annettu:** syöte $x \in \{0,1\}^*$
> **Laskettava:** oikea tuloste $y \in \{0,1\}^*$.

Tällainen ongelma voidaan palauttaa joukoksi kyllä/ei-kysymyksiä:

> **Annettu:** syöte $x \in \{0,1\}^*$
> **Kysymys A:** onko oikeassa tulosteessa ainakin k bittiä?
> **Kysymys B:** onko oikean tulosteen k. bitti ykkönen?

1. Säännölliset kielet

Äärellinen automaatti on hyvin yksinkertainen laskennan malli (eli abstrakti laskentalaite). *Säännölliset kielet* on se luokka laskentaongelmia, jonka näin yksinkertaisella laitteella pystyy ratkaisemaan. Näillä tekniikoilla on sovelluksia esim. merkkijonoalgoritmeissa.

Tämän luvun jälkeen opiskelija

- osaa selittää äärelliset automaatit, säännölliset lausekkeet ja niiden suhteen
- osaa muodostaa yksinkertaisia äärellisiä automaatteja ja säännöllisiä lausekkeita
- osaa tehdä muunnoksia determinististen ja epädeterminististen äärellisten automaattien ja säännöllisten lausekkeiden välillä
- osaa osoittaa kielen ei-säännölliseksi esim. *pumppauslemmalla*.

Äärellinen automaatti: johdattelua [Sipser s. 31–34]

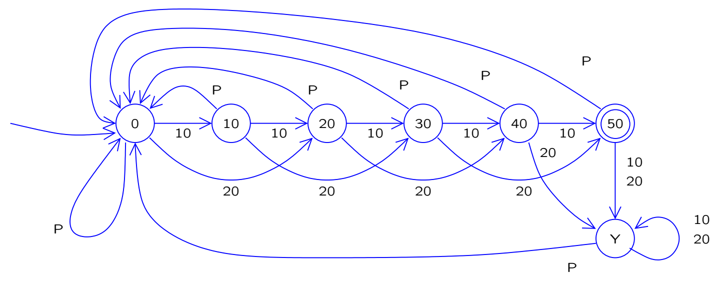
Havainnollistamme äärellistä automaattia tilanteessa, jossa kahviautomaatista voi ostaa kahvin 50 sentillä.

Laite hyväksyy 10 ja 20 sentin kolikoita. Kahvia saa vain maksamalla tasan 50 senttiä. Laitteessa on kuitenkin rahanpalautuspainike, jolla saa takaisin kaikki sinne laittamansa rahat.

Muodostamme *äärellisen automaatin*, jonka syötteenä on jono aakkoston $\{10, 20, \mathsf{P}\}$ symboleja. Nämä esittävät koneeseen laitettuja 10 ja 20 sentin kolikoita ja palautusnappulan painalluksia.

Automaatin täytyy muistaa sen verran, että se tietää, milloin sen saama rahamäärä viimeisen rahanpalautuksen jälkeen on tasan 50 senttiä. Tämän se tekee siirtymällä syötteen määräämällä tavalla *tilasta* toiseen. (Tiloja on äärellinen määrä, mistä mallin nimi.)

Saamme seuraavanlaisen äärellisen automaatin:



Automaatilla on

tiloja (7 kappaletta), jotka on esitetty ympyröinä ja nimetty 0, 10, 20, 30, 40, 50 ja Y;
siirtymiä, jotka on esitetty tilojen välisinä kaarina;
aakkosto, jonka symboleilla siirtymät on merkitty;
alkutila (tila 0), joka on merkitty tyhjästä tulevalla kaarella; ja
hyväksyvä tila (tila 50), joka on rengastettu.

Tilojen nimet on tässä valittu kuvaaviksi (ne kertovat, paljonko rahaa koneessa on), mutta automaatin toimintaan nimet eivät vaikuta. Tilassa Y rahaa on liikaa. Tilassa 50 rahaa on tasan oikea määrä.

Kaikissa tiloissa palautuspainike nollaa rahat. Kolikon laittaminen koneeseen taas "kasvattaa laskuria" vastaavalla määrällä. Tilassa Y kone pyörii silmukassa sekä 10 että 20 sentin syötteellä, kunnes tulee P.

Äärellinen automaatti: formaali määritelmä [Sipser s. 35–41]

Äärellinen automaatti on viisikko $(Q, \Sigma, \delta, q_0, F)$, missä

1. Q on äärellinen *tilajoukko*; sen alkioita sanotaan *tiloiksi*
2. Σ on äärellinen *aakkosto*
3. $\delta\colon Q \times \Sigma \to Q$ on *siirtymäfunktio*
4. $q_0 \in Q$ on *alkutila*
5. $F \subseteq Q$ on *hyväksyvien tilojen* joukko (joita toisinaan hieman harhaanjohtavasti kutsutaan *lopputiloiksi*).

Siirtymäfunktio on tässä kaikkein kiinnostavinta. Jos tiloilla q ja q' ja symbolilla $a \in \Sigma$ pätee $\delta(q, a) = q'$, niin intuitiivisesti tämä tarkoittaa, että jos ollaan tilassa q ja seuraavaksi tulee merkki a, niin siirrytään tilaan q'.

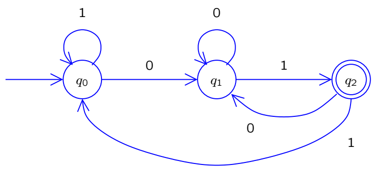
Esimerkki Tarkastellaan äärellistä automaattia $M_1 = (Q, \Sigma, \delta, q_0, F)$, missä

1. $Q = \{\, q_0, q_1, q_2 \,\}$,
2. $\Sigma = \{\, 0, 1 \,\}$,
3. δ saadaan taulukosta

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

   (siis esim. $\delta(q_1, 0) = q_1$ ja $\delta(q_1, 1) = q_2$),
4. alkutila on q_0 ja
5. $F = \{\, q_2 \,\}$.

Edellisen sivun automaatti M_1 voidaan esittää havainnollisemmin *tilakaaviona*



Automaatin toiminta syötteellä 1001101 voidaan kuvata siirtymäjonona

$$q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2.$$

Syötemerkkien loppuessa automaatti on hyväksyvässä tilassa q_2, ja sanomme, että se *hyväksyy* merkkijonon 1001101.

Toisaalta syötteen 00100 automaatti *hylkää*: $q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_1 \xrightarrow{0} q_1$ ja päädymme ei-hyväksyvään tilaan q_1.

Formalisoimme nyt edellä kuvatun laskennan.

Jos $M = (Q, \Sigma, \delta, q_0, F)$ on äärellinen automaatti ja $w = w_1 \dots w_n$ on n merkkiä pitkä aakkoston Σ merkkijono, niin automaatti M *hyväksyy* merkkijonon w, jos on olemassa $n + 1$ tilan jono $(r_0, r_1, \dots, r_n) \in Q^{n+1}$, jolla

- $r_0 = q_0$,
- $r_{i+1} = \delta(r_i, w_{i+1})$ kun $0 \le i \le n - 1$ ja
- $r_n \in F$.

Siis laskenta alkaa alkutilasta, noudattaa syötemerkkien mukaisia siirtymiä ja päätyy hyväksyvään tilaan.

Jos M ei hyväksy, se *hylkää* merkkijonon w.

Reunahuomatus: miksi formalismia

- On sinänsä arvokasta, että laskenta saadaan esitetyksi täsmällisessä matemaattisessa muodossa.
- Jatkossa haluamme esittää väitteitä muotoa

  Kaikki äärelliset automaatit toteuttavat ehdon p

  ja konstruktioita

  Mistä tahansa ehdon r toteuttavasta automaatista voidaan muodostaa automaatti, joka toteuttaa ehdon s.

  Tätä voidaan tiettyyn rajaan saakka tehdä piirtämällä kuvia, mutta uskottavuuden ja ymmärrettävyyden rajat tulevat melko pian vastaan.
- Matemaattinen formalismi on voimakas työkalu, jota on hyvä opetella käyttämään.

Automaatin M *tunnistama kieli* $L(M)$ on kaikkien niiden merkkijonojen joukko, jotka M hyväksyy. Esim. edellistä automaattia M_1 tarkastelemalla voidaan todeta, että se hyväksyy tasan ne merkkijonot, jotka loppuvat 01; siis

$$L(M_1) = \{ w_1 \dots w_n 01 \mid n \ge 0, w_i \in \{ 0, 1 \} \text{ kaikilla } i. \}$$

Sanomme, että kieli A on *säännöllinen*, jos jokin äärellinen automaatti tunnistaa sen, ts. $A = L(M)$ jollain M.

Huomaa, että säännöllisyys on *kielen* ominaisuus, ei yksittäisen merkkijonon. Ei ole mielekästä kysyä yksittäisestä merkkijonosta, onko se säännöllinen.

(Mistä tahansa merkkijonosta $w \in \Sigma^*$ voidaan toki muodostaa yksialkioinen kieli $\{ w \} \subseteq \Sigma^*$, joka on selvästi säännöllinen.)

Äärellinen automaatti on hyvin rajoittunut laskennan malli. Esim. niinkin yksinkertainen kieli kuin $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ (eli merkkijonot joissa on ensin jono nollia ja sitten saman verran ykkösiä) **ei ole** säännöllinen. (Tähän palataan.)

Äärelliseen automaattiin voidaan helposti lisätä muutakin tulostusta kuin pelkkä hyväksyminen tai hylkääminen. Tällaisen modifikaatiot eivät tämän kurssin kannalta toisi mitään oleellista uutta asiaan. Muissa yhteyksissä ne voivat kuitenkin olla hyödyllisiä.

Jatkoa ajatelleen on hyödyllistä määritellä kuvaus $\delta^* \colon Q \times \Sigma^* \to Q$ seuraavasti:

> $\delta^*(q, w)$ on se tila, johon tilasta q päädytään merkkijonolla w.

Erityisesti $\delta^*(q, \varepsilon) = q$ kaikilla q. Toinen perustapaus on $\delta^*(q, a) = \delta(q, a)$ kaikilla $a \in \Sigma$.

Nyt automaatin tunnistama kieli voidaan luonnehtia seuraavasti:

$$w \in L(M) \quad \text{jos ja vain jos} \quad \delta^*(q_0, w) \in F.$$

Funktio δ^* voidaan määritellä siirtymäfunktion δ avulla rekursiivisesti:

$$\delta^*(q, w) = \begin{cases} q & \text{jos } w = \varepsilon \\ \delta(\delta^*(q, v), a) & \text{jos } w = va \text{ missä } a \in \Sigma. \end{cases}$$

Tämä tarkoittaa samaa kuin rekursiivinen proseduuri

```
DeltaStar(q, w)
1  olkoon w = w_1 ... w_n, missä w_i ∈ Σ kaikilla i
2  if n = 0
3     then return q                              ▷ w = ε
4     else  p ← DeltaStar(q, w_1 ... w_{n-1})     ▷ v = w_1 ... w_{n-1}
5           return δ(p, w_n)                      ▷ a = w_n
```

Rekursio aina päättyy, sillä

1. rivin 4 rekursiivisessa kutsussa parametrina olevan merkkijonon pituus aina pienenee ja
2. merkkijonon pituudella on ehdoton alaraja 0.

Äärellisen automaatin laatiminen [Sipser s. 41–43]

Halutaan laatia äärellinen automaatti, joka tunnistaa kielen

$$L_1 = \{\, 0^n 1^m \mid n, m \in \mathbb{N} \,\} \cup \{\, 1^n 0^m \mid n, m \in \mathbb{N} \,\}.$$

Siis joko nollat ovat ennen ykkösiä tai toisin päin; lukumäärällä ei ole väliä. Toisin sanoen vaihdoksia nollan ja ykkösen välillä saa olla korkeintaan 1.

Lähtökohta: seuraava algoritmi ratkaisee, kuuluuko syötejono kieleen L_1:

```
vaihtunut ← False
Get(nykyinen)
while syöte ei loppu
      do edellinen ← nykyinen
         Get(nykyinen)
         if nykyinen ≠ edellinen
           then if not vaihtunut
                   then vaihtunut ← True    ▷ ensimmäinen vaihdos
                   else  return False       ▷ toinen vaihdos
return True
```

Tässä *nykyinen* ja *edellinen* ovat merkkejä ja Get lukee seuraavan syötemerkin.

Koska algoritmi tarvitsee vain vakiomäärän muistia, toteutus äärellisenä automaattina on realistinen ajatus.

Äärellisen automaatin siirtymämekanismi huolehtii sellaisenaan uuden merkin lukemisesta. Algoritmin muisti pitää sen sijaan toteuttaa tilojen avulla.

Tässä meillä on kaksi muuttujaa, kummallakin kaksi arvoa, joten valitsemme neljä tilaa:

- $F0$: *vaihtunut* = False ja *edellinen* = 0
- $F1$: *vaihtunut* = False ja *edellinen* = 1
- $T0$: *vaihtunut* = True ja *edellinen* = 0
- $T1$: *vaihtunut* = True ja *edellinen* = 1

(Yleisemmin b bittiä muistia algoritmissa vastaa 2^b tilaa automaatissa.)