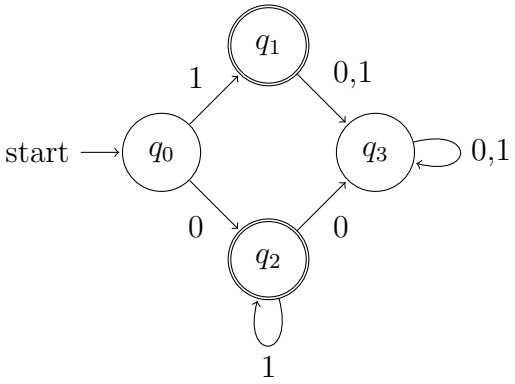
ÄÄRELLISTEN AUTOMAATTIEN MINIMOINTI

MIKKO KANGASMÄKI

1. Äärelliset automaatit

Äärellinen automaatti (DFA = deterministic finite automaton) on viisikko $(Q, \Sigma, s, \delta, F)$, missä

Q on äärellinen joukko *tiloja*
Σ on äärellinen kokoelma merkkejä, *aakkosto*
$s \in Q$ on *aloitustila*
$\delta : \Sigma \times Q \to Q$ on *siirtymäfunktio,* ja
$F \subset Q$ on *hyväksyvien* tilojen joukko.



Esimerkki 1.1.
Kuvan äärellisen automaatin tilojen joukko on

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

aakkosto $\{0, 1\}$, aloitustila q_0 ja hyväksyvien tilojen joukko $F = \{q_1, q_2\}$.

Siirtymäfunktio on ikävä kuvata sanallisesti, se on parhaiten ilmoitettu kuviolla: Tilasta q_0 menee 1:llä otsikoitu nuoli q_1:een ja 0:lla otsikoitu nuoli q_2:een, joten

$$\delta(1, q_0) = q_1 \text{ ja } \delta(0, q_0) = q_2.$$

Koska kuva täsmentää funktion δ yksikäsitteisesti, sen ilmoittaminen missään muussa muodossa on turhaa vaivannäköä.

Merkitään aakkostosta Σ muodostettuja äärellisen mittaisia sanoja Σ^*. Tässä joukossa on mukana myös tyhjä sana ε. Esimerkiksi aakkostolla $\{0, 1\}$ on

$$\Sigma = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots\}$$

Automaatti käsittelee sanan $w = w_1 w_2 \ldots w_k \in \Sigma^*$ siirtymällä ensin aloitustilasta tilaan $\delta(w_1, s)$, siitä tilaan $\delta(w_2, \delta(w_1, s))$ ja siitä edelleen

tilaan $\delta(w_3, \delta(w_2, \delta(w_1, s)))$ ja niin edelleen. Kun syöte loppuu, automaatti hyväksyy sanan, jos se on hyväksyvässä tilassa, ja hylkää muuten.[1]

Tämä on jälleen selkeämpi esimerkin kautta ajateltuna: Ylläolevalle automaatille syöte 01001 kulkee reittiä

$$q_0 \to q_2 \to q_2 \to q_3 \to q_3 \to q_3.$$

Automaatti hylkää sanan, koska se päätyy tilaan q_3, joka ei ole hyväksyvä.

Sanalle 0111 automaatti kulkee reittiä

$$q_0 \to q_1 \to q_1 \to q_1 \to q_1$$

ja hyväksyy sen, koska q_1 on hyväksyvä tila.

Ajatellaan tästedes funktiota δ rekursiivisest laajennettuna joukkoon Σ^* siten, että

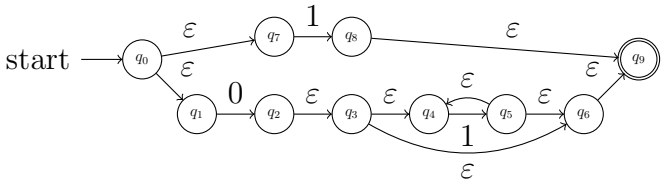
$$\delta(w_1 w_2 \dots w_k, q) := \delta(w_2 w_3 \dots w_k, \delta(w_1, q)).$$

Esimerkin automaatista huomaa pian, että se hyväksyy sanat

$$L := \{1, 01, 001, 0001, 00001, \dots\}.$$

Sanotaan, että L on automaatin hyväksymä *kieli*. Sitä voidaan merkitä säännöllisenä lausekkeena $1 \cup (0^*1)$.

Yleisemmin, jokaista äärellistä automaattia vastaa säännöllinen lauseke ja jokaista säännöllistä lauseketta äärellinen automaatti. Tämä voidaan todistaa määrittelemällä ensin epädeterministinen äärellinen automaatti (NDFA), jonka siirtymäfunktio ei määrää seuraavaa tilaa, vaan tilojen joukon, josta automaatti voi valita. Säännölliselle lausekkeelle voidaan rakentaa NDFA rekursiivisesti (ks. Sipser tai Hopcroft), mutta tyypillisesti tällaiset NDFAt ovat tarpeettoman isoja. Tässä esimerkki Hopcroftin NDFA:sta, joka tunnistaa saman kielen kuin esimerkkiautomaattimme:



Jokaisesta NDFA:sta puolestaan voidaan tehdä DFA. Tyypillisesti DFAn tilat ovat tällöin NDFAn kaikkien tilojen osajoukkoja, joten niiden määrä kasvaa n:stä 2^n:ään.

Jos siis laittaisimme tietokoneen muuntamaan säännöllisiä lausekkeita äärellisiksi automaateiksi Sipserin metodilla, saisimme esimerkin nelitilaisen automaatin sijasta 1024-tilaisen. Tämä on jo sinänsä hyvä syy miettiä, miten automaatteja voitaisiin minimoida, mutta toki se on yleisemminkin tutkimisen arvoinen asia. Lisäksi käyttämämme

[1]Tässä w_i:t ovat yksittäisiä aakkosia. Jätän tekstissä tällaiset asiat mainitsematta, jos lukijan voi olettaa tajuavan haetun merkityksen

metodi on hyödyllinen muutenkin, esimerkiksi kahden automaatin osoittamisessa ekvivalenteiksi keskenään.

Minimoinnissa on ajatuksena samaistaa tiloja, joista päädytään samaan lopputulokseen, riippumatta mikä käsiteltävän sanan loppuosa on. Katsotaan ensin, mitä samanlaisuus tässä tarkoittaa, sen jälkeen, millaisin keinoin tilojen samankaltaisuus pystytään löytämään, ja lopuksi vielä algoritmin oikeellisuutta ja aikavaatimuksia.

2. Tilojen ekvivalenssi

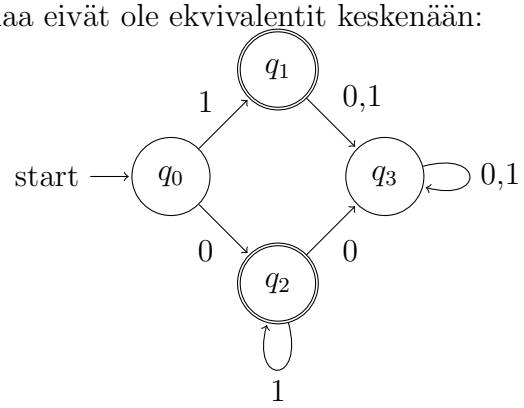
Määritelmä 2.1. Äärellisen automaatin $(Q, \Sigma, s, \delta, F)$ tilat q ja p ovat ekvivalentit keskenään, jos

$$\forall w \in \Sigma^* : \delta(q, w) \in F \iff \delta(p, w) \in F.$$

Tilat siis ovat ekvivalentit, jos niistä päädytään jokaisella sanalla molemmista joko hyväksyvään tai hylkäävään tilaan.

Jos tilat p ja q eivät ole ekvivalentit, on olemassa sana $w \in \Sigma^*$, joka *erottaa* ne, eli tasan toinen tiloista $\delta(w, p)$ ja $\delta(w, q)$ on hyväksyvä.

Esimerkki 2.2. Jo käyttämässämme esimerkissämme mitkään kaksi tilaa eivät ole ekvivalentit keskenään:

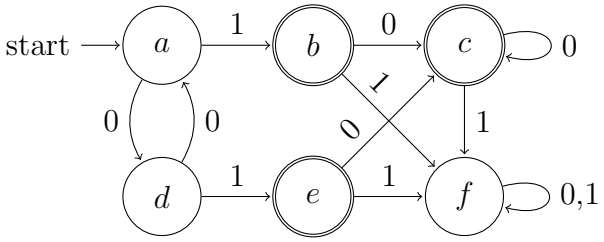


Tässä on taulukko sanoista, jotka erottavat eri tilat toisistaan:

	q_0	q_1	q_2	q_3
q_0		ε	ε	0
q_1			1	ε
q_2				ε

Taulukko on täytetty vain tarpeellisilta osiltaan, eivätkä kaikki kohdat ole yksikäsitteisiä. Esimerkiksi tilat q_0 ja q_3 erottaa toisistaan myös sana 1 tai 0000000001. Toisaalta tiloja q_1 ja q_3 ei erota toisistaan mikään muu sana kuin ε, joka erottaa toisistaan hylkäävät ja hyväksyvät tilat.

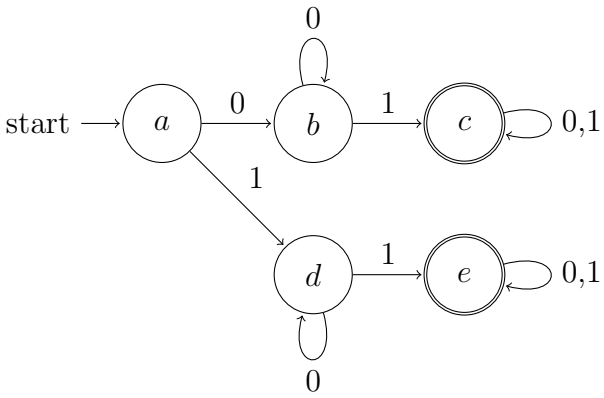
Esimerkki 2.3. Tässä automaatissa tilat c ja e ovat ekvivalentit:



Miten tämä nähdään? Käydään läpi kaikki mahdolliset sanat pituutensa mukaan: Sana ε hyväksytään molemmista tiloista. Sanalla 0 päädytään molemmista sanoista tilaan c, joka hyväksyy. Sanalla 1 päädytään molemmista sanoista tilaan f, joka hylkää. Pidemmillä sanoilla mennään molemmista tiloista samoihin paikkoihin: 1:llä c:hen ja 0:lla f:ään, joten reitit ovat loppusanalle samat.

Myös b ja c, sekä a ja d voidaan samantapaisella järkeilyllä osoittaa ekvivalenteiksi keskenään. Koska ekvivalenssimme todella on ekvivalenssirelaatio (helppo harjoitustehtävä), b, c ja e muodostavat yhden ekvivalenssiluokan, ja a ja d toisen. Tila f ei ole ekvivalentti minkään muun tilan kanssa, b:stä, c:stä ja e:stä sen erottaa ε ja a:sta ja d:stä 1.

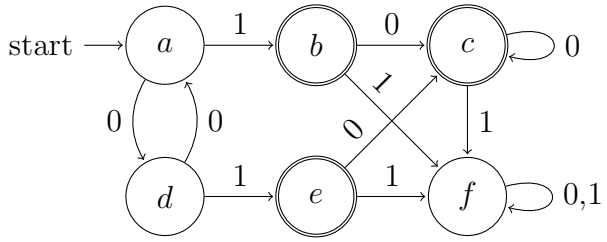
Esimerkki 2.4. Huomaa, että ekvivalenssi ei edellytä, että molemmista tiloista päädytään kaikilla sanoilla samaan *tilaan*. Tässä automaatissa tilat b ja d ovat ekvivalentit, mutta niistä ei voida koskaan päätyä samaan tilaan:



3. Algoritmi ekvivalenssien löytämiseksi

On helppoa huomata, että kaksi tilaa eroavat toisistaan (eivät ole ekvivalentit), mutta ekvivalenssin näyttäminen on hankalampaa. Seuraavassa algoritmissa on ajatuksena etsiä toisistaan eroavia tiloja kunnes se ei enää onnistu, ja tämän jälkeen loput tilat ovat ekvivalentit keskenään.

Esimerkki 3.1. Etsitään tästä automattista keskenään ekvivalentit tilat:



Aluksi piirretään taulukko, jossa on kaikki tilaparit:

b		X	X	X	X
c			X	X	X
d				X	X
e					X
f					
	a	b	c	d	e

Taulukosta on rastittu yli toista kertaa esiintyvät tilaparit.

Sana ε erottaa hyväksyvät tilat muista, joten merkitään se taulukkoon tilapareihin, joista tasan yksi hyväksyy:

b	ε	X	X	X	X
c	ε		X	X	X
d		ε	ε	X	X
e	ε			ε	X
f		ε	ε		ε
	a	b	c	d	e

Nyt aletaan käydä tyhjiä ruutuja läpi yksi kerrallaan ja katsotaan, päästäänkö niistä yhdellä aakkosella jo eroavaksi merkittyyn tilapariin. Jos päästään, merkitään ruutuun se aakkonen, jolla päästään.

Tilaparista (a, d) päästään 0:lla $(d.a)$:han, jota ei ole vielä merkitty. 1:llä päästään pariin (b, e), jota sitäkään ei ole merkitty, joten tähän ruutuun ei merkitä ainakaan vielä mitään.

Seuraavasta tyhjästä ruudusta $(a.f)$ päästään 0:lla (d, f):ään, jota ei ole vielä merkitty eroavaksi. Aakkosella 1 päästään (b, f):ään, joka on merkitty eroavaksi. Merkitään se näin:

b	ε	X	X	X	X
c	ε		X	X	X
d		ε	ε	X	X
e	ε			ε	X
f	1	ε	ε		ε
	a	b	c	d	e

Jatketaan tilaparien läpikäyntiä: (b, c):stä, (b, e):stä ja $(e.c)$:stä ei päästä millään aakkosella eroavaksi merkittyyn ruutuun. Parista (f, d) sen sijaan päästään 1:llä (f, e):hen, joka on merkitty eroavaksi, joten

merkitään ruudukkoon:

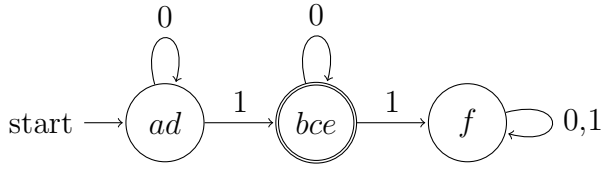
b	ε	X	X	X	X
c	ε		X	X	X
d		ε	ε	X	X
e	ε			ε	X
f	1	ε	ε		ε
	a	b	c	d	e

Nyt kaikki tilaparit on käyty kerran läpi, ja olemme ilmeisestikin löytäneet kaikki sellaiset tilaparit, jotka enintään yhden aakkosen mittainen sana erottaa toisitaan.

Seuraavaksi käydään taulukko uudelleen läpi, niin kauan, kunnes läpikäynnillä ei enää merkitä yhtään tilaparia eroavaksi. Sen jälkeen kaikki merkitsemättömät tilaparit ovat ekvivalentteja keskenään.

Tässä esimerkkitapauksessa toinen läpikäynti ei tee muutosta taulukkoon, joten tilojen joukko Q jakautuu ekvivalenssiluokkiin $\{a, d\}, \{b, c, e\} ja \{f\}$.

Jokaisesta ekvivalenssiluokasta tulee uuden automaatin solmu. Aloitustila on se, joka sisältää alkuperäisen aloitustilan. Hyväksyviä ovat ne, joiden sisältämät solmut ovat hyväksyviä. Siirtymäsäännöissä valitaan jokin ekvivalnssiluokan solmu, katsotaan, mihin siitä siirryttiin alkuperäisessä automaatissa, ja merkitään uuteen siirtymä sen sisältävään ekvivalenssiluokkaan:



Algoritmi pähkinänkuoressa

- Tee taulukko, jossa on kaikki automaatin tilaparit.
- Merkitse ε:lla tilaparit, joissa tasan yksi tila on hyväksyvä.
- Käy taulukko läpi ja jokaisen merkitsemättömän tilaparin (p, q) kohdalla:
  - Jos tilaparista päästään jollain aakkosella $a \in \Sigma$ jo merkittyyn tilapariin $(\delta(a, p), \delta(a, q))$, niin merkitse ruutuun (p, q) merkki a.
- ...ja lopeta, kun taulukko on käyty läpi ensimmäisen kerran ilman, että yhtään uutta tilaparia on merkitty,

4. Algoritmin aikavaativuus ja oikeellisuus

Solmupareja[2] on $n(n-1)/2 \approxeq n^2$ kappaletta, ja jokaisella kierroksella vähintään yksi merkitään eroavaksi (tai muuten algoritmi pysähtyy),

[2]solmu = tila

joten parit käydään läpi enintään n^2 kertaa. Aikavaativuudella on siis yläraja $|\Sigma|n^4$.

Nettiä selaillessani näin väitettävän aikavaatimukseksi $O(n^3)$, mutta en missään perusteita tälle. Tämä pitäisi kyllä paikkansa, jos ei jokaista tilaparia varten tarvitse tarkistaa ensin, onko se merkattu, eli jos vaikkapa pidettäisiin yllä listaa merkitsemättömistä pareista.

Algoritmi voidaan tehostaa $O(n^2)$-aikaiseksi solmuparikohtaisilla listoilla, joihin merkataan, mikä solmu riippuu tästä parista. (Ks. Hopcroft s. 70-1).

Lisäksi on nopeampikin $O(n \log n)$-aikainen algoritmi, josta enemmän myöhemmin.

Algoritmi toimii oikein:

(1) *Jos kaksi solmua ovat ekvivalentit, niitä ei merkitä eroaviksi.* Tämä on täysin selvää, sillä kun solmupari (p, q) on merkitty eroavaksi merkillä a, voidaan katsoa, mikä merkki erottaa solmuparin $(\delta(a, p), \delta(a, q))$ ja niin edelleen, kunnes lopulta saadaan sana, joka erottaa solmut p ja q toisistaan.

(2) *Jos kaksi solmua eroavat toisistaan, ne merkitään eroaviksi.* Tämä on selvää samalla tavalla kuin edellinen. Oletetaan, että sana $w_1 w_2 \dots w_k$ erottaa solmut p ja q. Merkitään:

$$p_i := \delta(w_1 w_2 \dots w_i, p)$$

ja q_i vastaavasti. Tasan toinen tiloista p_k ja q_k hyväksyy, joten pariin (p_k, q_k) on ruudukossa merkitty ε.
Merkillä w_k päästäään tilaparista (p_{k-1}, q_{k-1}) tiloihin (p_k, q_k), joten se on merkitty vastaavaan paikkaan taulukosta. Näin jatketaan tilapareille $(p_{k-2}, q_{k-2}), (p_{k-3}, q_{k-3}) \dots$, kunnes lopulta ruutuun (p, q) merkitään w_1.

(3) *Uusi automaatti hyväksyy saman kielen kuin alkuperäinen eikä uusien siirtymäsääntöjen tekeminen aiheuta ristiriitoja.* Tämänkin näkee helposti induktiivisesti: Aloitetaan tilasta q_0 sanalla $w = w_1 w_2 \dots w_k$. Tästä päädytään hyväksyvään tilaan jos ja vain jos tilasta $q_1 := \delta(w_1, q_0)$ päädytään hyväksyvään tilaan sanalla $w_2 w_3 \dots w_k$. Se taas pätee jos ja vain jos tilan q_1 kanssa ekvivalenteista tiloista p päästään hyväkysvään tilaan sanalla $w_2 w_3 \dots w_k$.
Vuorottelemalla tällä tavalla ekvivalenssirelaatiota ja automaatissa eteenpäin kulkemista huomataan, että sanan hyväksymisen kannalta uudet siirtymäsäännöt eivät aiheuta ristiriitaa.

Saatu automaatti on myös pienin, joka tunnistaa saman kielen kuin alkuperäinen.

Todistus. Oletetaan, että algoritmi tuottaa automaatin M, jossa on m tilaa. Olkoon N pienempi automaatti n:llä tilallaan, $n < m$.

Valitaan nyt jokaiselle M:n tilalle p sana $w_p \in \Sigma^*$ siten, että $\delta_M(w_p, s_M) = p$, eli sana jolla päästään aloitustilasta tilaan p. Näitä sanoja on m

kappaletta, joten N päätyy ainakin kahdella näistä sanoista samaan tilaan:

$$\delta_N(w_p, s_N) = \delta_N(w_q, s_N) \text{ joillakin } p, q \in Q_M.$$

Nyt, koska tilat p ja q eivät ole ekvivalentit, on oltava jokin sana $v \in \Sigma^*$, joka erottaa ne. Toisin sanoen, on oltava sana $v \in \Sigma^*$ siten, että automaatti M hyväksyy tasan toisen sanoista $w_p v$ ja $w_q v$.

Automaatilla N puolestaan tämä ei ole mahdollista, sillä jo sanojen alkuosissa ollaan päädytty samaan tilaan. Automaatti N siis hyväksyy tai hylkää yhtäaikaa molemmat sanat $w_p v$ ja $w_q v$, joten se tunnistaa eri kielen kuin M.□

5. Hopcroftin algoritmi

Hopcroftin artikkelissaan 1971 esittämä algoritmi toimii nopeammin $n \log n$-ajassa. En puutu tässä sen oikeellisuuteen tai aikavaativuuteen, mutta kerron lyhyesti, kuinka se toimii:

- Tee joukko S, johon laitetaan solmujoukkoja. Laita sinne joukot F ja $Q \setminus F$.
- Tee jono J, johon laitetaan solmujoukkoja. Laita sinne toinen joukoista F tai $Q \setminus F$.
- Niin kauan kuin jono J ei ole tyhjä, tee:
  - Ota jonosta J joukko A.
  - Jokaiselle aakkoselle $a \in \Sigma$, tee:
    - Määrittele joukko B_a niiden solmujen joukoksi, joista siirrytään aakkosella a joukkoon A:

      $$B_a := \{q \in Q : \delta(a, q) \in A\}.$$

    - Jokaiselle $X \in S$, jolle $X \cap B_a \neq \emptyset$ ja $X \setminus B_a \neq \emptyset$
      - poista X joukosta S ja laita tilalle joukot $X \cup B_a$ ja $X \setminus B_a$.
      - Jos $X \in J$, korvaa se joukoilla $X \cup B_a$ ja $X \setminus B_a$. Muussa tapauksessa laita toinen näistä joukoista jonoon.

Algoritmi ei ole oikeasti niin hankala kuin miltä se näyttää. Suurin vaikeus on viitsiä lukea niin monta sisäkkäistä luuppia. Osa algortimista on lisäksi aika teknistä tavaraa.

Ajatus lyhyesti on jakaa ensin kaikkien solmujen joukko Q hyväksyviin, F, ja ei hyväksyviin, $Q \setminus F$. Sitten jakoa hienonnetaan etsimällä joukot, josta päästään yhdellä merkillä hyväksyviin tiloihin. Sitten jakoa hienonnetaan etsimällä joukot, joista päästään yhdellä merkillä näihin tiloihin jne.

Pohjimmiltaan kyse on samasta ideasta kuin aiemmassa algoritmissa, mutta vain toisella tavalla toteutettuna.

References

[Hopcroft] Hopcroft, Ullman: *Introduction to Automata Theory, Languages and Computation* 1979

[Hopcroft1971] Hopcroft: *An n log n algorithm for minimizing the states in a finite automaton*
1971, The Theory of Machines and Computations pp. 189-96
`ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/71/190/CS-TR-71-190.pdf`

[Sipser] Sipser: *Introduction to the Theory of Computation.*