TIEA241 Automaatit ja kieliopit

Antti Valmari

Jyväskylän yliopisto
Informaatioteknologian tiedekunta

Symboleita

Aritmetiikka

$n \operatorname{div} m$	osamäärä, kun n jaetaan m:llä kokonaislukujakona
$n \bmod m$	jakojäännös, kun n jaetaan m:llä kokonaislukujakona

Logiikka

$\varphi \Rightarrow \xi$	φ:stä päätellään ξ; jos φ on totta, niin myös ξ on totta (mutta ei välttämättä toisinpäin)
$\varphi \Leftrightarrow \xi$	φ on totta jos ja vain jos ξ on totta
$\varphi \wedge \xi$	*looginen ja*, sekä φ että ξ ovat totta
$\varphi \vee \xi$	*looginen tai*, φ tai ξ tai molemmat ovat totta
$\neg\xi$	*looginen ei*, ξ ei ole totta
$\forall x \in A : \xi$	ξ on totta jokaisella x:n arvolla joukosta A
$\exists x \in A : \xi$	ξ on totta ainakin yhdellä x:n arvolla joukosta A
$\forall x : \xi$	kuten edellä, mutta A oletetaan tunnetuksi asiayhteydestä
$\exists x : \xi$	kuten edellä, mutta A oletetaan tunnetuksi asiayhteydestä
$\forall x; \varphi : \xi$	ξ on totta jokaisella x:n arvolla, joka toteuttaa ehdon φ
$\exists x; \varphi : \xi$	ξ on totta ainakin yhdellä x:n arvolla, joka toteuttaa ehdon φ
$\square$	(oikeassa reunassa) todistuksen loppua osoittava merkki

Joukko-oppi

$\{2, 3, 5\}$		joukko, jonka alkiot ovat 2, 3 ja 5
$\emptyset$		*tyhjä joukko* eli joukko, jossa ei ole yhtään alkiota
$\{x \mid \varphi\}$		niiden alkioiden x joukko, jotka toteuttavat ehdon φ
$a \in A$		alkio a *kuuluu* joukkoon A
$a \notin A$		alkio a *ei kuulu* joukkoon A
$A \subseteq B$		*osajoukko*, jokainen A:n alkio kuuluu myös B:hen
$A \subset B$		*aito osajoukko*, $A \subseteq B$ mutta $A \neq B$
$A \cup B$		*unioni*, joukko jossa ovat sekä A:n alkiot että B:n alkiot (eikä muuta)
$A \cap B$		*leikkaus*, joukko jossa ovat A:n ja B:n yhteiset alkiot (eikä muuta)
$A \setminus B$		joukko, jossa ovat ne A:n alkiot jotka eivät ole B:ssä (eikä muuta)
$A \times B$		*tulojoukko*, niiden parien (a, b) joukko joille $a \in A$ ja $b \in B$
$A \to B$		*funktio* joukolta A joukolle B
$A \nrightarrow B$	18	*osittainen funktio* joukolta A joukolle B
$\lvert A \rvert$		joukon A alkioiden määrä
$\lvert A \rvert = \infty$		ilmoittaa, että joukko A on ääretön
A^*	20	joukosta A muodostettavien merkkijonojen joukko
A^+	20	joukosta A muodostettavien epätyhjien merkkijonojen joukko

Merkkijonot, kielet ja automaatit

ε	12	*tyhjä merkkijono* eli merkkijono, jossa ei ole yhtään merkkiä
$\|\sigma\|$	19	merkkijonon σ pituus eli merkkien määrä
$q \Rightarrow^{\sigma} q'$	20	tilasta q on polku tilaan q' merkkijonolla σ
$\mathcal{L}(X)$	21, 51	automaatin X hyväksymä kieli
$\sigma^{-1}K$	35	kielen K *loppuosa σ:n jälkeen*
σ^n	52	σ toistettuna n kertaa

Lukujoukot

$\mathbb{Z}$	$= \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ *kokonaisluvut*
$\mathbb{N}$	$= \{0, 1, 2, \ldots\}$ *luonnolliset luvut*
$\mathbb{Z}^+$	$= \{1, 2, 3, \ldots\}$ *positiiviset kokonaisluvut*
$\mathbb{Q}$	*rationaaliluvut* eli luvut, jotka voidaan esittää muodossa $\frac{n}{m}$, missä $n \in \mathbb{Z}$ ja $m \in \mathbb{Z}^+$
$\mathbb{Q}^+$	*positiiviset rationaaliluvut*

1 Johdanto

Kurssilla käydään läpi tärkeimmät automaattisen laskennan
rajoja ja mahdollisuuksia koskevat matemaattiset tulokset

- yleissivistystä, jonka "kaikki muutkin" tietävät
  - tyypillisesti osa syvällisiä ohjelmointipainotteisia tutkintoja
  - monien uudempien tulosten perusta esim. rinnakkaisuuden teoriassa
  - samankaltaisia ajattelutapoja käytetään muissakin asioissa tietojenkäsittelyssä

  $\Rightarrow$ helpottaa alan kehityksen seuraamista
  - osa asioista erityisen sopivia loogis-matemaattisen ajattelutavan harjoittelemiseen
- joillakin tuloksilla on suuri käytännön merkitys erityisesti
  - ohjelmointi- yms. kielten käsittelyssä
  - ratkottaessa vaikeita tehtäviä tietokoneella
- monilla tuloksilla on suuri filosofinen merkitys
  - tietokoneen universaalisuus (Church–Turingin teesi)
  - tietokoneen periaatteelliset rajoitukset
  - läheinen yhteys matemaattiseen logiikkaan ja matematiikan filosofiseen perustaan
  - tekoälyn mahdollisuuksien ja rajoitusten pohdinta
  - yritykset ymmärtää luonnollista älyä
- mukana on myös vähemmän tärkeitä asioita mm. esimerkkeinä ja havainnollistajina

Laskennan teoriassa on hyvin pitkälti kyse siitä

- mitä äärettömiä joukkoja voi ilmaista äärellisin keinoin
- mitä niille voi tehdä (esim. jäsenyystesti, unioni) ja kuinka nopeasti

Käsiteltäviä asioita

- äärelliset automaatit
  - yksinkertainen, heikko laskennan malli
  - melko tärkeä ohjelmointi- yms. kielten käsittelyssä
  - samankaltainen käsite tilakone on erittäin tärkeä mm. tietoliikenneprotokollissa
- yhteysriippumattomat kieliopit
  - keskivoimakas laskennan malli
  - tärkeä hierarkkisten rakenteiden käsittelyssä

  $\Rightarrow$ erittäin tärkeä ohjelmointi- yms. kielten käsittelyssä
  - jokseenkin sama asia kuin BNF eli Backus–Naur Format
- Turingin koneet ja laskettavuuden rajat
  - voimakkain tunnettu realistinen laskennan malli

  $\Rightarrow$ tavallisen tietokoneen matemaattinen vastine
  - laskettavuuden rajat ovat tärkeitä mm. tekoälyn kannalta sekä filosofisesti
- Chomskyn hierarkia
  - ym. kolme laskennan mallia sekä yhteysherkät kieliopit
  - sukua ohjelmointikielten nelitasomallille, joka auttaa hahmottamaan ohjelmointikieliin liittyviä asioita

- **NP**-täydelliset tehtävät
  - osataan ratkaista tietokoneella, mutta usein vain tolkuttoman hitaasti
  - mukana paljon käytännössä tärkeitä, kuten jakelureitin suunnitteluun liittyviä

  $\Rightarrow$ tärkeä ymmärtää, kun halutaan tehdä vaikeita asioita tietokoneella
  - on olemassa paljon muitakin samankaltaisia vaativuusluokkia

Kirjallisuus

- aihepiiristä on lukuisia kirjoja
  - esim. Kinber, Smith: Theory of Computing: A Gentle Introduction
- englanninkielinen Wikipedia on näissä asioissa varsin pätevä ja luotettava
- netistä löytyy joistakin aiheista opetusvideoita yms.
- eri lähteissä on eroja merkinnöissä yms. yksityiskohdissa
  - vaikeuttaa muun aineiston hyödyntämistä
- tavoitteena on, että tämä luentoesitys olisi helpompi ymmärtää kuin kirjat yms.

Tavoitteena on myös opiskella matemaattista luku- ja ajattelutaitoa yleensä

- erityisesti ohjelmoinnissa hyödyllisen matematiikan osalta
- siellä täällä kerrotaan, miten tai miksi jokin on tapana matematiikassa
- korostetaan päättelyitä ja syitä, miksi jokin asia pätee
  - yleensä hyödyllisintä ei ole tietää lopputulosta, vaan ymmärtää sen perustelut

- harjoitellaan kaavojen käyttöä
  - varsinkin aluksi mukana on sanallinen ilmaus varmistamassa ymmärtämistä
  - sanallisilla ilmauksilla on taipumusta olla pitkiä ja epätarkkoja

  $\Rightarrow$ jotkin asiat voi ilmaista kunnolla vain kaavoilla

  $\Rightarrow$ aina tukena ei ole sanallista ilmausta

  $\Rightarrow$ älä jättäydy sanallisten ilmausten varaan, vaan opettele lukemaan kaavoja
- matematiikassa on tapana käyttää sanallisten ilmausten ja kaavojen sekakieltä
  - tarkka tieto varmistetaan kaavoilla
  - kaavojen jatkuva tulkinta on rasittavaa matemaatikoillekin

  $\Rightarrow$ asioille annetaan havainnollisia nimiä ja symboleita, ja käytetään niitä
  - niiden merkitys ei kuitenkaan ole yleiskielen mukainen, vaan hyvin jämpti
  - "pisteen P ympäristössä ..." tarkoittaa esim. "on olemassa $r > 0$ siten, että jokaiselle pisteelle P', jonka etäisyys P:stä on alle r ..."
  - "tehtävä on ratkeamaton" tarkoittaa "jokaiselle algoritmille A on olemassa syöte siten, että ko. syötteellä A vastaa väärin tai laskee ikuisesti"
  - odottakaapas kun näette mitä "kieli" tarkoittaa tällä kurssilla!
- varsinkin alkuvaiheessa on usein yhtäaikaa monta eri suuntiin vievää asiaa, joista tarvitsee kertoa
  - vaarana on rönsyily ja tärkeimmän ajatuksen katoaminen muiden taakse

  $\Rightarrow$ joskus tärkeitä asioita jätetään kertomatta heti, mutta niihin palataan
- en tiedä kirjoja tms., joissa yritettäisiin opettaa matemaattista luku- ja ajattelutaitoa tällä tavalla

Esityksessä käytettävä värikoodaus

- erityisen tärkeitä asioita on osoitettu keltaisella taustalla
  - opiskele ainakin ne
  - usein niiden ymmärtäminen edellyttää myös ympäröiviä asioita
- himmeänkeltainen tausta osoittaa keskitärkeitä asioita
  - arvosanoihin 3, 4 ja 5 tähtääville
  - arvosanaan 5 tähtäävän kannattaa opiskella myös korostamattomat asiat
- tärkeitä mutta muun perusteella ilmeisiä asioita ei ole osoitettu väritaustalla
- *johtopäätöksiä yms. tärkeää* on korostettu violeteilla vinoilla kirjaimilla
- **määriteltävä sana** tai **käsite** on vahvennettuna sinisellä
  - ei välttämättä tärkeä, katso taustan tai ympäröivän tekstin väri
- esimerkeissä käytetään usein ruskeaa
  - esimerkkejä ei kysytä tentissä, vaikka olisivat mustallakin
  - esimerkit ovat vain asioiden ymmärtämisen tueksi
- jokin asia on osoitettu **hyväksi** tai **huonoksi** vihreällä tai punaisella
- harmaalla kirjoitettua pitkää osuutta ei varmasti kysytä tentissä
  - harmaata käytetään, kun asia muuten näyttäisi tentissä kysyttävältä
- lyhyt harmaa osuus voi olla tärkeä
  - esim. rönsy

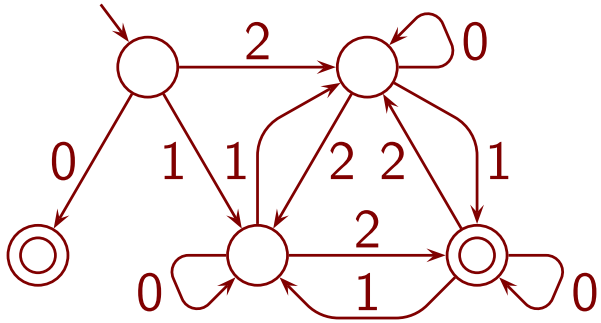
Pari neuvoa

- Älä keskity yksityiskohtien ulkoaopetteluun, vaan niiden taustalla olevien periaatteiden ymmärtämiseen!
- tehtäviä tehdään siksi, että opittaisiin *jatkossa tärkeitä* taitoja tai asioita!
  - tehtävän aihe voi olla ihan turha, mutta oppimistavoite ei ole
  - jos bluffaa alkuvaiheen tehtävissä, niin loppuvaiheen tehtävistä tulee ylivoimaisia
  - ”tentin (tai harjoituksen) jälkeen voin unohtaa” on huono asenne

2 Äärelliset automaatit ja säännölliset kielet

2.1 Deterministiset äärelliset automaatit

Deterministinen äärellinen automaatti koostuu viidenlaisista osista



- **tilat** (*states*) ◯ ◎
  - epätyhjä äärellinen joukko
  - tilojen nimet voidaan jättää kuvissa pois
- **aakkosto** (*alphabet*) $\{0, 1, 2\}$
  - äärellinen joukko
- **kaaret** eli **tilasiirtymät** (*transitions*) $\xrightarrow{2}$
  - kaari alkaa tilasta, päättyy tilaan ja on nimetty yhdellä aakkosella
  - samasta tilasta lähtee samalla aakkosella korkeintaan yksi kaari
  - käytämme useimmiten sanaa "kaari", koska "tilasiirtymä" on pitkä sana
- **alkutila** (*initial state*) ◯ tai ◎
  - yksi tila
- **lopputilat** (*final states*) eli **hyväksymistilat** (*accepting states*) ◎
  - nolla tai useampia tiloja

Englanniksi *deterministic finite automaton*

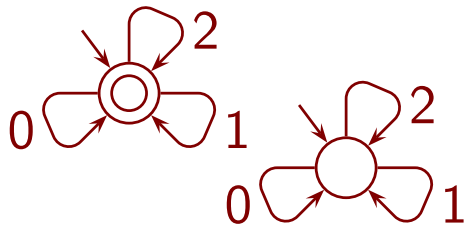
- lyhenne **DFA**

DFA:n tehtävä on **hyväksyä** (*accept*) ja **hylätä** (*reject*) aakkostosta muodostettuja merkkijonoja (*string*)

- aloitetaan alkutilasta ja kuljetaan merkkijonon ohjaamana
  - alkutilassa valitaan kaari merkkijonon ensimmäisen merkin mukaan
  - seuraavassa tilassa valitaan kaari merkkijonon toisen merkin mukaan
  - jne. kunnes sopivaa kaarta ei ole tai merkkijono loppuu
- jos vuorossa olevalle merkille ei ole kaarta, niin merkkijono hylätään
- kun merkkijono loppuu, niin
  - jos ollaan lopputilassa, niin merkkijono hyväksytään
  - jos ollaan muussa kuin lopputilassa, niin merkkijono hylätään
- kuvan DFA hyväksyy mm. 0, 12, 120, 201 ja 2001120
- kuvan DFA hylkää mm. 1, 012 ja 122

Kaksi esimerkkiä

- DFA, joka hyväksyy kaikki merkkijonot aakkostosta $\{0, 1, 2\}$
- DFA, joka hylkää kaikki merkkijonot aakkostosta $\{0, 1, 2\}$



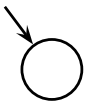
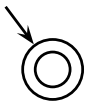
Aakkosto ei aina käy kuvasta ilmi

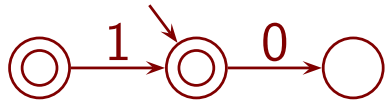
- sisältää *ainakin* kuvan kaarissa esiintyvät nimet
- voi sisältää muutakin, jottei tarvitsisi lisätä kaaria keinotekoisesti

$\Rightarrow$ mainittava tarvittaessa erikseen

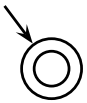
Havaintoja

- jos lopputiloja ei ole, niin DFA hylkää kaikki merkkijonot
- jos tilaan ei ole polkua alkutilasta, niin tila on turha
- jos tilasta ei ole polkua lopputilaan eikä tila ole alkutila, niin tila on turha
  - DFA:lla on aina oltava alkutila
- ainakin yksi merkkijono hyväksytään $\Leftrightarrow$ alkutilasta on polku lopputilaan
- jos aakkosto on tyhjä joukko, niin kaaria ei voi olla

  $\Rightarrow$ turhien tilojen poiston jälkeen DFA on joko  tai 

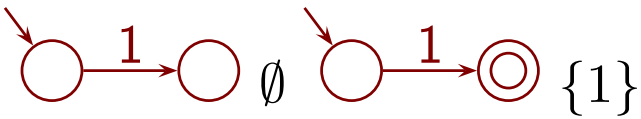


Tyhjä merkkijono

- se merkkijono, jossa ei ole yhtään merkkiä
- se merkkijono, jonka  hyväksyy (olipa aakkosto mikä tahansa)
- merkitään symbolilla ε
  - merkitseminen kirjoittamalla 0 merkkiä (siis ei mitään) aiheuttaisi sekaannusta
  - ei olisi tarpeen, jos merkkijonot rajat osoitettaisiin esim. ”lainausmerkeillä”: ””

$\Rightarrow$ *ε ei ole aakkostoon kuuluva merkki, vaan keino ilmaista merkkijono, jossa ei ole yhtään merkkiä*

- on eri asia hyväksyä ε kuin hyväksyä ei mitään



Äärellinen ja ääretön joukko

- *joukko* on kokoelma alkioita, jotka voivat olla melkein mitä vain
  - Viikonpäivät = {maanantai, tiistai, keskiv., torstai, perjantai, lauantai, sunnuntai}
  - Suomen_suurpedot = {karhu, ilves, susi, ahma}
  - Suomen_pussieläimet = $\emptyset$ (tyhjä joukko)
  - $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
  - Alkuluvut = $\{n \in \mathbb{Z} \mid n > 1 \wedge \neg\exists i \in \mathbb{Z} : \exists j \in \mathbb{Z} : i > 1 \wedge j > 1 \wedge n = ij\}$
- alkion ylimääräisillä esiintymillä ei ole merkitystä
  - esim. {karhu, ilves, susi} = {karhu, ilves, karhu, susi, ilves, karhu}
- alkioiden luettelointijärjestyksellä ei ole merkitystä
  - esim. {karhu, ilves, susi, ahma} = {ahma, ilves, karhu, susi}
- joukko on **ääretön**, jos ja vain jos yritys luetteloida se ei valmistu koskaan
  - esim. $\mathbb{N} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,$
  - yleensä tunnistaa siitä, että mikään alkio ei ole viimeinen
  - esim. mikään luonnollinen luku n ei ole viimeinen, koska $n + 1$ on suurempi
  - tämä kriteeri toimii vain lueteltaessa normaalisti vasemmalta alkaen, vrt. negatiiviset kokonaisluvut $\mathbb{Z}^- = \{\ldots, -3, -2, -1\}$
  - äärettömässä joukossa *voi* olla suurin alkio, esim. $\{\ldots, \frac{1}{3}, \frac{1}{2}, \frac{1}{1}\}$
- joukko on **äärellinen**, jos ja vain jos se ei ole ääretön
  - voidaan (ainakin periaatteessa) luetella kokonaan, jos aikaa ja tilaa riittää
  - esim. Viikonpäivät, Suomen_suurpedot ja $\{n \in \mathbb{N} \mid n \leq 100000000\}$
  - myös $\emptyset$ on äärellinen, vaikka siinä ei ole viimeistä alkiota

Kolme pistettä ...

- edustaa jotakin, jota ei voida tai jakseta kirjoittaa auki ja joka lukijan on helppo arvata
  - esim. $\{0, 1, \ldots, 9\} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - esim. $\{\ldots, -3, -2, -1\}$ negatiiviset kokonaisluvut
  - esim. $a_1 a_2 \cdots a_n$ merkkijono, jossa on n merkkiä, missä n voi vaihdella
- voi tarkoittaa äärellistä tai ääretöntä määrää alkioita
  - esim. $\{0, 1, \ldots, 9\}$
  - esim. $\{\ldots, -3, -2, -1\}$

Äärellinen ja ääretön merkkijono

- **merkkijono** on äärellinen määrä aakkoston merkkejä peräkkäin
  - alkaa vasemmalta ja loppuu oikealle
  - esim. aakkostosta $\{\mathsf{a}, \mathsf{b}, \ldots, \mathsf{ö}\}$: koira, koiranputki, h, ε
- **ääretön merkkijono** on ääretön määrä aakkoston merkkejä peräkkäin
  - alkaa vasemmalta mutta ei lopu
  - esim. kkk$\cdots$, toistotoisto$\cdots$
  - ei voida ilmaista ilman kolmea pistettä tai muuta epäsuoraa keinoa
  - useimpien reaalilukujen desimaaliesitykset ovat äärettömiä merkkijonoja
  - tällä kurssilla käytetään äärettömiä merkkijonoja hyvin vähän
- **äärellinen merkkijono** on sama kuin merkkijono

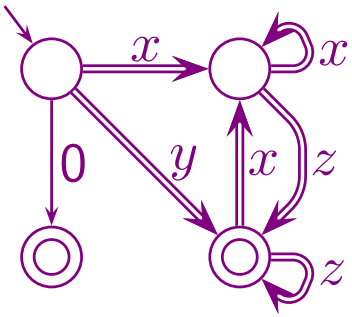
Kieli

- muodollisten kielten teoriassa kielen määritelmä on tyrmäävän yksinkertainen:

  kieli on joukko merkkijonoja

- määritelmän mukaan kieli tarkoittaa vain merkkijonojen jakoa sisä- ja ulkopuolisiin
  - esim. Koira on hieno eläin! kuuluu suomenkieleen
  - esim. Qwerty uiop asd? ei kuulu suomenkieleen
- määritelmä ei siis yritäkään tavoittaa
  - kielen ilmaisujen merkitystä, esim. computer vs. datamaskin vs. tietokone
  - kieliopillisia rakenteita, esim. `if (` *ehto* `)` *lause* `else` *lause*
- monimutkaisempia ilmiöitä tutkitaan muilla käsitteillä
  - esim. BNF:llä (myöhemmin tällä kurssilla) voi esittää kieliopillisia rakenteita
- ennen kuin mennään monimutkaisempiin ilmiöihin, on hyvä kyetä erottamaan ne merkkijonot, jotka esittävät jotain, niistä jotka eivät esitä mitään
  $\Rightarrow$ tyrmäävän yksinkertainen kielen käsitteemme tarvitaan lähtökohdaksi muulle
- ehkä yllättäen, kuuluu / ei kuulu kieleen onkin vaikea asia!
  - jokaista kyllä/ei-kysymystä vastaa kieli: ne merkkijonot, joille vastaus on "kyllä"
  - esim. Alkuluvut $= \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots\}$
  - esim. Tentin_tehtävän_1_täyden_pistemäärän_tuottavat_vastaukset
- tätä näkökulmaa käytetään paljon laskettavuuden teoriassa
  - syöte on aina viime kädessä merkkijono
  - esim. onko syöte alkuluku $\rightsquigarrow$ kuuluuko syöte kieleen Alkuluvut

Äärettömien joukkojen ilmaiseminen on mahdollista vain epäsuorasti



- ei voida luetteloida kaikkia alkioita!
- joskus on helppo kuvata ääretön kieli sanallisesti
  - esim. Parilliset = ne numerojonot, jotka loppuvat 0, 2, 4, 6 tai 8
- logiikan keinoin voidaan ilmaista esim. luonnollisten lukujen äärettömiä osajoukkoja
  - esim. Parilliset $= \{n \mid \exists k : n = 2k\}$
  - esim. Alkuluvut $= \{n \mid n > 1 \wedge \neg\exists i : \exists j : i > 1 \wedge j > 1 \wedge n = ij\}$
- kun ilmaisujärjestelmä on sovittu, on ilmaisu äärellinen merkkijono
  - esim. äärellinen suomenkielinen teksti
  - esim. äärellinen logiikan kaava
  - yritys kirjoittaa ääretön teksti tai kaava ei valmistu koskaan!

x: 1, 3, 5, 7, 9
y: 2, 4, 6, 8
z: 0, 2, 4, 6, 8

$\Rightarrow$ kun ilmaisujärjestelmä on sovittu, useimmille kielille ei ole *minkäänlaista* ilmausta!
  - kieliä on ylinumeroituva määrä
  - äärellisiä merkkijonoja on vain numeroituva määrä
- mikään keino ei pysty esittämään kaikkia kieliä

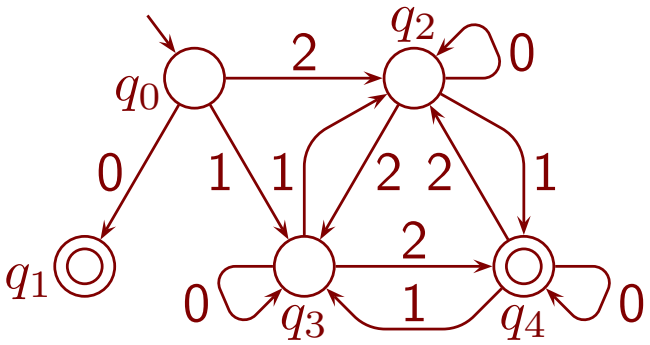
Äärelliset automaatit ovat *helppo* keino, joka pystyy ilmaisemaan vain *vähän* kieliä

- helppo $\Rightarrow$ kätevä silloin, kun niiden ilmaisuvoima riittää
- äärelliset automaatit pystyvät ilmaisemaan joitakin hyödyllisiä äärettömiä kieliä
- myöhemmin käsittelemme ilmaisuvoimaisempia, mutta vaikeampia keinoja

Deterministisen äärellisen automaatin matemaattinen määritelmä

DFA on viisikko $(Q, \Sigma, \delta, \hat{q}, F)$, missä

- Q on äärellinen joukko
- Σ on äärellinen joukko siten, että $\varepsilon \notin \Sigma$
- δ on osittainen funktio $Q \times \Sigma \nrightarrow Q$
- $\hat{q} \in Q$
- $F \subseteq Q$

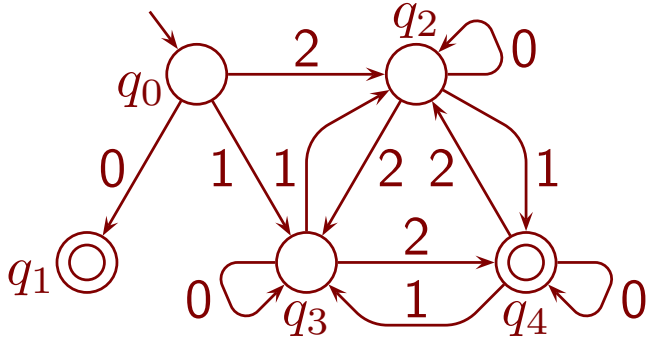


Huomautuksia

- Q on *tilojen joukko*, ja sen alkioita sanotaan *tiloiksi*
  - vaaditaan äärellisyys, koska puhumme *äärellisistä* automaateista
  - tilat piirretään yksin- tai kaksinkertaisina ympyröinä
  - aikaisemmin sanottiin erikseen, että $Q \neq \emptyset$ eli Q ei ole tyhjä joukko
  - sitä ei tarvitse sanoa erikseen, koska $\hat{q} \in Q$ takaa sen
  - esimerkissä $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- tilojen nimet tarvitaan vain apuna, kertomaan mistä kaari alkaa jne.
  $\Rightarrow$ voidaan jättää kuvista pois
- Σ on *aakkosto*, ja sen alkioita sanotaan *merkeiksi* tai *aakkosiksi*
  - myös sen on oltava äärellinen
  - jollei kuvan yhteydessä ole sanottu aakkostoa, se koostuu kuvassa esiintyvistä merkeistä
  - esimerkissä $\Sigma = \{0, 1, 2\}$

- δ on *osittainen tilasiirtymäfunktio*
  - $\delta : Q \times \Sigma \nrightarrow Q$ tarkoittaa, että δ ottaa tilan ja aakkosen
  - $\delta : Q \times \Sigma \nrightarrow Q$ tarkoittaa, että δ tuottaa tilan tai ei mitään
  - esimerkissä $\delta(q_0, 0) = q_1$, $\delta(q_0, 1) = q_3$ ja $\delta(q_0, 2) = q_2$
  - $\delta(q_1, 0)$, $\delta(q_1, 1)$ ja $\delta(q_1, 2)$ eivät ole määriteltyjä eli eivät tuota mitään
  - $\delta(q_2, 0) = q_2$, $\delta(q_2, 1) = q_4$ ja $\delta(q_2, 2) = q_3$
- *kaari* eli *tilasiirtymä* on kolmikko (q, a, q') siten, että $\delta(q, a)$ on määritelty ja on q'
  - piirretään nuolena $q \xrightarrow{a} q' = \delta(q, a)$
  - määrittelemätöntä $\delta(q, a)$ ei piirretä
  - kaaria on enintään $|Q||\Sigma|$ kappaletta: jokaisesta tilasta jokaisella aakkosella

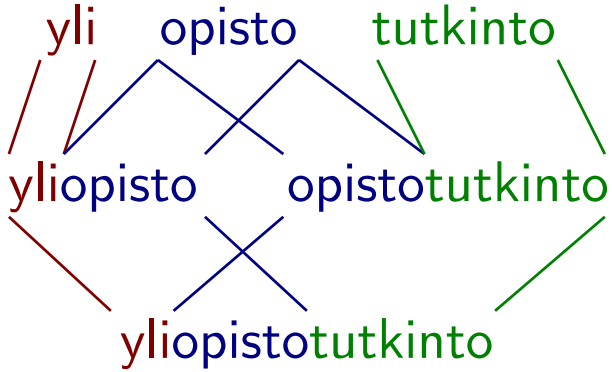
  $\Rightarrow$ kaaria on vain äärellinen määrä
  - esimerkissä $|Q| = 5$, $|\Sigma| = 3$ ja $|\delta| = 12$
  - määrittelemättä on loput $5 \cdot 3 - 12 = 3$ tapausta
- samasta tilasta samalla nimellä piirretään enintään yksi kaari
- $\hat{q}$ on *alkutila*
  - jokin tila
  - osoitetaan kuvissa tyhjästä alkavalla nuolella
- F on *lopputilojen* eli *hyväksymistilojen joukko*
  - saa olla $\emptyset$, Q tai mitä tahansa siltä väliltä (jos on $\emptyset$, niin hyväksytty kieli on tyhjä)
  - on äärellinen, koska Q on äärellinen
  - piirretään kaksinkertaisina ympyröinä



- tarvittaessa käytetään yläpilkkua, indeksejä tms. erottamaan eri DFA:t toisistaan
  - esim. $(Q', \Sigma', \delta', \hat{q}', F')$ ja $(Q_1, \Sigma_1, \delta_1, \hat{q}_1, F_1)$

Merkeistä ja merkkijonoista

- merkitsemme merkkejä pienillä aakkosjärjestyksen alkupään kirjaimilla a, b, c jne.
  - kuten lähes aina, voidaan käyttää yläpilkkua, alaindeksiä jne.
  - esim. a', b_2
- merkitsemme merkkijonoja kahdella tavalla:
  - merkkien jonoina, esim. $a_1 a_2 \cdots a_n$
  - pienillä kreikkalaisilla kirjaimilla σ, ρ jne.
- merkitsemme merkkijonon σ pituutta $|\sigma|$
  $\Rightarrow |a_1 \cdots a_n| = n$
  - esim. $|\mathsf{koira}| = 5$, $|\mathsf{h}| = 1$ ja $|\varepsilon| = 0$

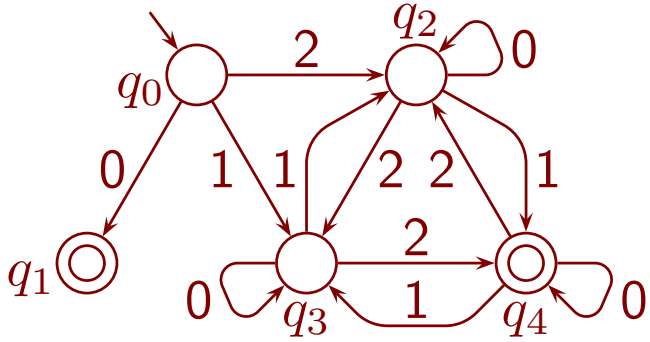


- merkkejä ja merkkijonoja saa laittaa peräkkäin, ja merkitys on ilmeinen
  - esim. jos $\sigma = \mathsf{koira}$ ja $\rho = \mathsf{koppi}$, niin $\sigma\mathsf{n}\rho = \mathsf{koirankoppi}$
  - aina $\sigma\varepsilon = \varepsilon\sigma = \sigma$
  - aina $|\sigma\rho| = |\sigma| + |\rho|$
- aina $(\sigma\rho)\beta = \sigma(\rho\beta)$
  - esim. jos $\sigma = \mathsf{yli}$, $\rho = \mathsf{opisto}$ ja $\beta = \mathsf{tutkinto}$, niin $(\sigma\rho) = \sigma\rho = \mathsf{yliopisto}$, $(\rho\beta) = \rho\beta = \mathsf{opistotutkinto}$, ja $(\sigma\rho)\beta = \mathsf{yliopistotutkinto} = \sigma(\rho\beta)$
  - koska näin on, yleensä jätetään sulut pois eli kirjoitetaan $\sigma\rho\beta$

Merkkijonojen joukkoja

- A^* on kaikkien joukosta A muodostettavissa olevien merkkijonojen joukko
  - esim. $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots\}$
  - esim. $\emptyset^* = \{\varepsilon\}$
- A^+ on kaikkien joukosta A muodostettavissa olevien epätyhjien merkkijonojen joukko
  - esim. $\{a, b\}^+ = \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots\}$
  - esim. $\emptyset^+ = \emptyset$
  - $A^+ = A^* \setminus \{\varepsilon\}$ ja $A^* = A^+ \cup \{\varepsilon\}$

Kaarista muodostuvat polut



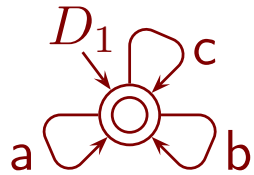
- $q =\sigma\Rightarrow q'$ tarkoittaa, että tilasta q on polku tilaan q' siten, että sen kaarien nimien jono on σ
  - esimerkissä mm. $q_0 =1\Rightarrow q_3$ ja $q_0 =20122\Rightarrow q_3$
  - jokaiselle $q \in Q$ pätee $q =\varepsilon\Rightarrow q$
  - $q =a\Rightarrow q'$ jos ja vain jos $\delta(q, a)$ on määritelty ja $\delta(q, a) = q'$
  - $q =a_1 \cdots a_n\Rightarrow q'$ missä $n > 1$ jos ja vain jos on olemassa q_1 siten, että $q =a_1\Rightarrow q_1$ ja $q_1 =a_2 \cdots a_n\Rightarrow q'$
- kun q ja σ on valittu, DFA:ssa on korkeintaan yksi q' siten, että $q =\sigma\Rightarrow q'$
  - "deterministinen" tarkoittaa tätä
  - myöhemmin tutustumme NFA:han, ja siellä niitä voi olla monta
- polkumerkintöjä saa ketjuttaa
  - $q =\sigma\Rightarrow q' =\rho\Rightarrow q_1 =\beta\Rightarrow q_2$ tarkoittaa, että $q =\sigma\Rightarrow q'$, $q' =\rho\Rightarrow q_1$ ja $q_1 =\beta\Rightarrow q_2$

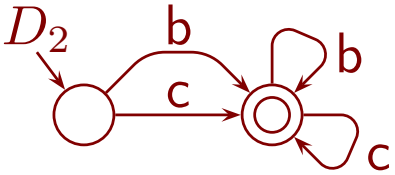
- $q \Rightarrow\sigma\rho\Rightarrow q'' \;\Leftrightarrow\; \exists q' : q \Rightarrow\sigma\Rightarrow q' \Rightarrow\rho\Rightarrow q''$
- $q \Rightarrow a_1 \cdots a_n\Rightarrow q'$ jos ja vain jos on olemassa $q_0, \ldots, q_n$ siten, että
  - $q = q_0$
  - $q_0 \Rightarrow a_1\Rightarrow q_1 \Rightarrow a_2\Rightarrow q_2 \Rightarrow a_3\Rightarrow \ldots \Rightarrow a_n\Rightarrow q_n$
  - $q_n = q'$

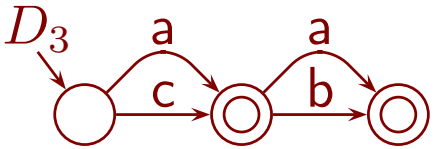
DFA:n $D = (Q, \Sigma, \delta, \hat{q}, F)$ hyväksymän kielen määritelmä

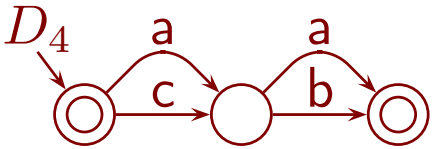
$$\mathcal{L}(D) = \{\sigma \in \Sigma^* \mid \exists q \in F : \hat{q} \Rightarrow\sigma\Rightarrow q\}$$

- niiden D:n aakkostosta muodostettavissa olevien merkkijonojen joukko, joille on olemassa polku, joka alkaa alkutilasta ja päättyy johonkin lopputilaan, ja jonka varrelta poimittujen merkkien jono on ko. merkkijono
- ts. D hyväksyy σ:n jos ja vain jos alkutilasta alkava, σ:n ohjaama kulkeminen päättyy johonkin lopputilaan
  - ei katkea kesken siksi, että vuorossa olevaa merkkiä vastaavaa kaarta ei ole
  - ei pääty muuhun kuin lopputilaan
- esimerkkejä aakkostolla {a, b, c}
  - $\mathcal{L}(D_1) = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}^*$
  - $\mathcal{L}(D_2) = \{\mathsf{b}, \mathsf{c}\}^+$
  - $\mathcal{L}(D_3) = \{\mathsf{a}, \mathsf{aa}, \mathsf{ab}, \mathsf{c}, \mathsf{ca}, \mathsf{cb}\}$
  - $\mathcal{L}(D_4) = \{\varepsilon, \mathsf{aa}, \mathsf{ab}, \mathsf{ca}, \mathsf{cb}\}$









Helppo ja havainnollinen lause

- tämän lauseen todistus kannattaa ymmärtää läpikotaisin!
  - helppo esimerkki ajattelutavasta, joka toistuu jatkossa useasti

$\mathcal{L}(D)$ on ääretön, jos ja vain jos on olemassa $\sigma \in \mathcal{L}(D)$ siten, että $|\sigma| \geq |Q|$

- suomeksi: D hyväksyy äärettömän monta eri merkkijonoa, jos ja vain jos se hyväksyy yhdenkin "tarpeeksi pitkän" merkkijonon
  - tässä "tarpeeksi pitkä" tarkoittaa, että pituus on vähintään D:n tilojen määrä
- lause ei ole keltaisella taustalla, koska sitä ei kannata muistaa ulkoa
  - nyt tavoitteena ei ole oppia ko. väittämää vaan syyt, miksi se pätee
- väitteen muotoa "möhköfantti jos ja vain jos tärppä" todistamisessa on kaksi vaihetta:
  - todistetaan, että jos möhköfantti niin tärppä
  - todistetaan, että jos tärppä niin möhköfantti

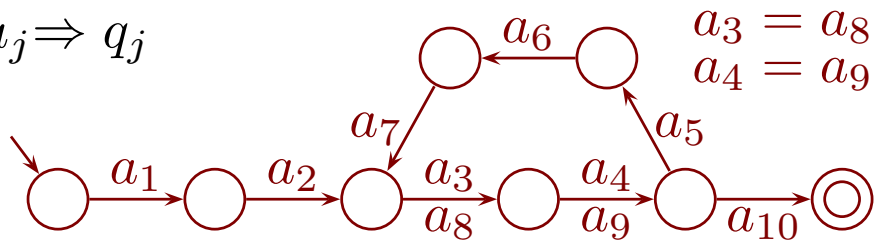
Todistamme ensin, että $\exists \sigma \in \mathcal{L}(D) : |\sigma| \geq |Q| \;\Rightarrow\; |\mathcal{L}(D)| = \infty$

- ts. jos on olemassa $\sigma \in \mathcal{L}(D)$ siten, että $|\sigma| \geq |Q|$, niin $\mathcal{L}(D)$ on ääretön
- oletamme, että on olemassa σ joka täyttää ym. ehdot, ja todistamme, että $\mathcal{L}(D)$ on ääretön
- olkoon $k = |\sigma|$ eli σ:n pituus
  - $k \geq |Q|$
  - voimme merkitä $\sigma = a_1 a_2 \cdots a_k$

- meillä on lupa merkitä näin, koska
  - k, a_1, ..., a_k eivät vielä ole varattu muuhun käyttöön
  - valintamme on σ:n rakenteen mukainen (k merkkiä peräkkäin)

  $\Rightarrow$ annamme vain uusia, aiemmin vapaita nimiä vanhoille asioille
- merkitsemme $q_0 = \hat{q}$, koska se helpottaa jatkossa
  - sallittua, koska q_0 ei vielä ole varattu muuhun käyttöön
- koska $\sigma \in \mathcal{L}(D)$, niin on olemassa q_1, ..., q_k siten, että $\hat{q} = q_0 =a_1\Rightarrow q_1 =a_2\Rightarrow \ldots =a_k\Rightarrow q_k$ ja q_k on lopputila
  - suomeksi: on olemassa polku alkutilasta lopputilaan, jonka nimien jono on σ
  - tämä tulee suoraan siitä mitä $\mathcal{L}(D)$ määriteltiin tarkoittamaan
  - sallittua, koska q_1, ..., q_n eivät vielä ole varattu muuhun käyttöön
- ym. polulla on $k+1$ tilaa: q_0, q_1, q_2, ..., q_k
- tiloja on kaikkiaan tätä vähemmän, koska $|Q| \leq k < k+1$

$\Rightarrow$ ainakin kahden ym. tiloista täytyy olla sama tila
  - ts. on olemassa i ja j siten, että $0 \leq i < j \leq k$ ja $q_i = q_j$
- ym. polun osa $q_i =a_{i+1}\Rightarrow q_{i+1} =a_{i+2}\Rightarrow \ldots =a_j\Rightarrow q_j$ muodostaa silmukan, jonka pituus on ainakin 1
  - sen pituus on $j-i$
  - koska $j > i$, pätee $j - i \geq 1$



- alkutilasta voidaan kulkea lopputilaan myös siten, että ym. silmukka kierretään kahdesti tai kolmesti tai miten monta kertaa tahansa

$\Rightarrow$ hyväksytään seuraavat merkkijonot

- $a_1 \cdots a_i a_{i+1} \cdots a_j a_{i+1} \cdots a_j a_{j+1} \cdots a_k$ pituus on $k + (j - i)$
- $a_1 \cdots a_i a_{i+1} \cdots a_j a_{i+1} \cdots a_j a_{i+1} \cdots a_j a_{j+1} \cdots a_k$ pituus on $k + 2(j - i)$
- $\ldots$

$\Rightarrow$ hyväksytään äärettömän monta toinen toistaan pitempää merkkijonoa $\square$

Todistamme sitten, että $|\mathcal{L}(D)| = \infty \Rightarrow \exists \sigma \in \mathcal{L}(D) : |\sigma| \geq |Q|$

- ts. jos $\mathcal{L}(D)$ on ääretön, niin on olemassa $\sigma \in \mathcal{L}(D)$ siten, että $|\sigma| \geq |Q|$
- erilaisia merkkijonoja, joiden pituus on 0, on 1 kpl, nimittäin ε
- erilaisia merkkijonoja, joiden pituus on 1, on $|\Sigma|$ kpl, nimittäin kukin merkki yksinään
- erilaisia merkkijonoja, joiden pituus on i, on $|\Sigma|^i$ kpl
  - kukin i merkistä voi saada $|\Sigma|$ eri arvoa

$\Rightarrow$ erilaisia merkkijonoja, joiden pituus on alle $|Q|$, on $|\Sigma|^0 + |\Sigma|^1 + \ldots + |\Sigma|^{|Q|-1}$ kpl

- osaisimme tuon laskeakin, mutta nyt riittää havainto, että se on äärellinen

$\Rightarrow \forall \sigma \in \mathcal{L}(D) : |\sigma| < |Q| \Rightarrow |\mathcal{L}(D)| < \infty$

- ts. jos jokainen $\mathcal{L}(D)$:n alkio on pituudeltaan alle $|Q|$, niin $\mathcal{L}(D)$ on äärellinen
- tämä on sama väite kuin mikä piti todistaa, mutta ilmaistuna "nurinpäin" $\square$

Matematiikassa on tapana muotoilla tulokset täsmällisiksi, selvästi rajatuiksi lauseiksi:

Lause 2.1 Olkoon D DFA. $\mathcal{L}(D)$ on ääretön, jos ja vain jos on olemassa $\sigma \in \mathcal{L}(D)$ siten, että $|\sigma| \geq |Q|$.

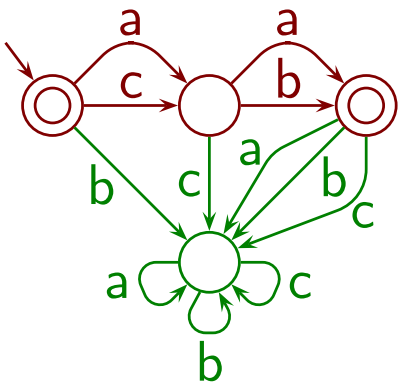
Pikakertaus miksi lause 2.1 pätee

- jos DFA:n hyväksymässä kielessä on tarpeeksi pitkä merkkijono σ, niin
  - σ:n hyväksyvä polku toistaa väkisin jonkin tilan

  $\Rightarrow$ polussa on silmukka
  - kulkemalla muuten sama polku mutta kiertämällä silmukka useasti saadaan loputtomasti toinen toistaan pitempiä merkkijonoja
- jos aakkosto on äärellinen, niin mitä tahansa annettua rajaa lyhyempiä merkkijonoja on vain äärellinen määrä
  - DFA:n aakkosto on äärellinen

  $\Rightarrow$ jos DFA:n hyväksymä kieli on ääretön, niin siinä on loputtomasti toinen toistaan pitempiä merkkijonoja

Tilasiirtymäfunktion täydentäminen

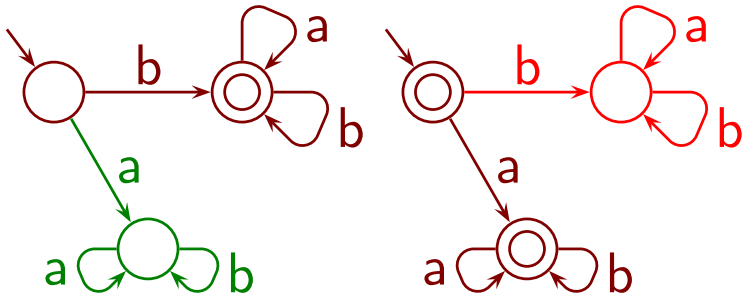


- toisinaan halutaan, että δ on *täysi* funktio
  - jokaisesta tilasta lähtee kaari jokaisella aakkosella
  - ts. $\delta(q, a)$ on määritelty jokaisella $q \in Q$ ja $a \in \Sigma$
- tarvittaessa tämä saadaan helposti voimaan kielen muuttumatta
  - lisätään yksi uusi tila $q_\dagger$
  - lisätään kaari $q_\dagger \Rightarrow_a q_\dagger$ eli asetetaan $\delta(q_\dagger, a) = q_\dagger$ jokaiselle $a \in \Sigma$
  - lisätään kaari $q \Rightarrow_a q_\dagger$ eli asetetaan $\delta(q, a) = q_\dagger$ jokaiselle q ja a, joille $\delta(q, a)$ oli määrittelemättä
- tämä kasvattaa tilojen määrää vain yhdellä

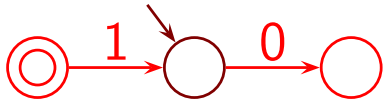
- se voi kasvattaa kaarien määrää paljon
  - juuri siksi sallimme δ:n olevan osittainen

Kielen komplementointi

- DFA on helppo muuntaa siten, että hylätyt merkkijonot muuttuvat hyväksytyiksi ja päinvastoin



1. tarvittaessa tilasiirtymäfunktio täydennetään
2. muutetaan lopputilat ei-lopputiloiksi ja päinvastoin
3. haluttaessa poistetaan turhat tilat ja kaaret
   - ne, jotka eivät ole millään alkutilasta lopputilaan vievällä polulla
   - alkutila ei kuitenkaan koskaan ole turha eikä sitä poisteta



Matematiikassa on tapana ilmaista käyttöön otettavien sanojen ja symbolien merkitykset täsmällisesti

$\Rightarrow$ otamme esimerkin vuoksi käyttöön täsmällisen merkityksen sanalle "turha"

Määritelmä DFA:n tila on **turha**, jos ja vain jos se ei ole alkutila eikä millään alkutilasta lopputilaan johtavalla polulla.

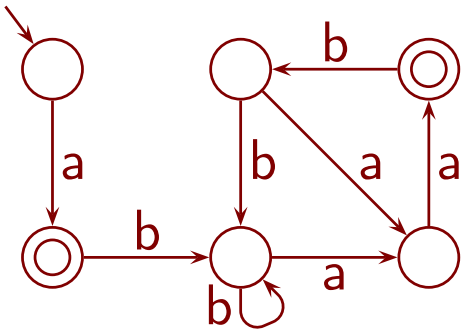
$\Rightarrow$ tästä eteenpäin sanalla "turha" ei ole sen yleiskielen mukaista merkitystä, vaan nimenomaan tämän määritelmän mukainen merkitys

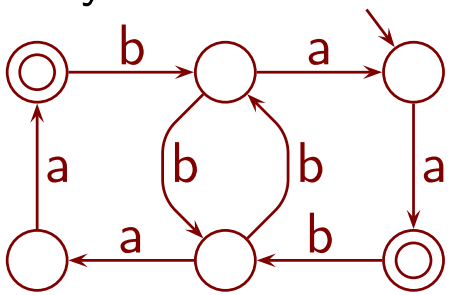
- kaavana $q \neq \hat{q} \wedge \neg\exists\sigma \in \Sigma^* : \exists\rho \in \Sigma^* : \exists q' \in F : \hat{q} =\sigma\Rightarrow q =\rho\Rightarrow q'$
- jos tilasiirtymäfunktion täydentämisessä luotiin uusi tila, niin se on turha

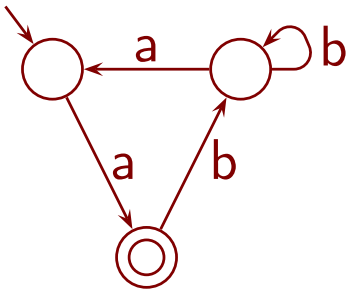
2.2 Deterministisen äärellisten automaattien minimointi

Tässä alaluvussa ratkaistaan hyvin tehokkaasti (mutta ei helposti) kaksi vaikeaa tehtävää:

- hyväksyvätkö kaksi DFA:ta saman vai eri kielen?
- mikä on pienin DFA, joka hyväksyy saman kielen kuin annettu DFA?
  - edellinen tehtävä ratkeaa tämän sivutuotteena
  - pienin DFA tulee osoittautumaan yksikäsitteiseksi







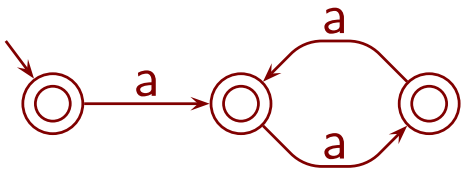
Minimoitava DFA on koko alaluvun ajan $D = (Q, \Sigma, \delta, \hat{q}, F)$

Apukäsite: tilan hyväksymä kieli

- jos $q \in Q$, niin $\mathcal{L}(q) = \{\sigma \in \Sigma^* \mid \exists q' \in F : q =\sigma\Rightarrow q'\}$
- tilan $\hat{q}$ hyväksymä kieli on sama kuin koko DFA:n hyväksymä kieli
  - vrt. $\mathcal{L}(D)$ sivulta 21: $\mathcal{L}(D) = \{\sigma \in \Sigma^* \mid \exists q \in F : \hat{q} =\sigma\Rightarrow q\}$

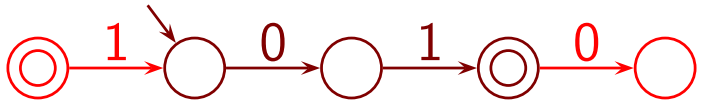
Apukäsite: siirtymä tilasta aakkosella tilajoukkoon

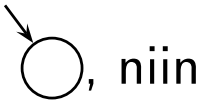
- olkoot $q \in Q$, $a \in \Sigma$ ja $B \subseteq Q$
- $q =a\Rightarrow B$ tarkoittaa, että $\exists q' \in B : q =a\Rightarrow q'$



Minimointialgoritmin vaihe 1: turhien tilojen poisto

- jokainen tila poistetaan, jolle pätee toinen tai molemmat seuraavista:
  - alkutilasta ei ole polkua siihen
  - tila ei ole alkutila ja siitä ei ole polkua lopputilaan

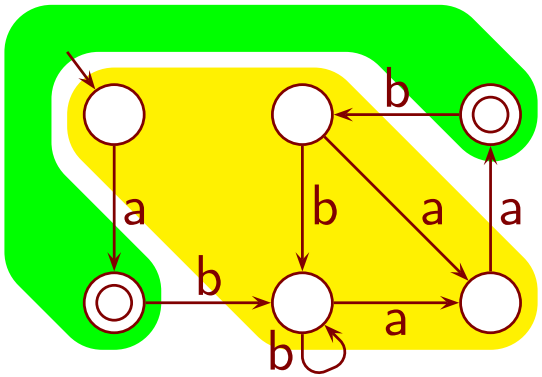


- onnistuu tavallisilla graafialgoritmeilla ajassa $O(|Q| + |\delta|)$ (esim. leveyteen ensin)
  $\Rightarrow$ emme käsittele tässä tämän tarkemmin
  - tästä eteenpäin $D = (Q, \Sigma, \delta, \hat{q}, F)$ on vaiheen 1 lopputulos
- jos D on  , niin koko minimointitehtävä on valmis
  - hyväksyy tyhjän joukon
  - on varmasti pienin mahdollinen!
- muussa tapauksessa $\forall q \in Q : \exists \sigma \in \Sigma^* : \exists \rho \in \Sigma^* : \exists q' \in F : \hat{q} =\sigma\Rightarrow q =\rho\Rightarrow q'$
  - ts. jokainen jäljellä oleva tila on jollakin alkutilasta lopputilaan vievällä polulla
  - voimme jatkossa olettaa näin, koska vastakkainen tapaus on loppuun käsitelty

$\Rightarrow$ ainakin yksi lopputila on jäljellä

Minimointialgoritmin vaihe 2: alkujako

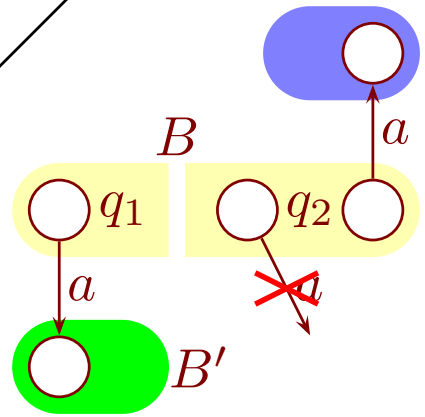
- jos myös ei-lopputiloja on jäljellä, tilat ryhmitetään kahdeksi **lohkoksi** (*block*): lopputilat ja muut tilat
- muussa tapauksessa muodostetaan yksi lohko: lopputilat
- varmistimme, että jokaisessa lohkossa on ainakin yksi tila
  - pidämme siitä huolta jatkossakin



- varmistimme myös, että jos kaksi tilaa on eri lohkoissa, niin ne hyväksyvät eri kielet
  - tila hyväksyy ε:n jos ja vain jos se kuuluu lopputilojen lohkoon
  - $\varepsilon \in \mathcal{L}(q) \Leftrightarrow q \in F$

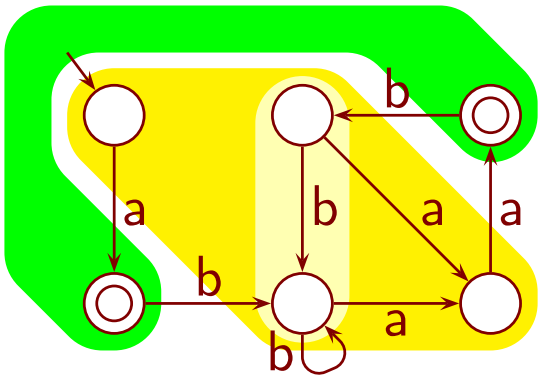
Minimointialgoritmin vaihe 3: lohkominen

- lohkoja pilkotaan pienemmiksi niin kauan kuin mahdollista siten, että koko ajan lohkot ovat epätyhjiä ja tuo pätee



- etsitään merkki $a \in \Sigma$ ja lohkot B ja B' siten, että B:ssä on tilat q_1 ja q_2 siten, että $q_1 =a\Rightarrow B'$ mutta $\neg(q_2 =a\Rightarrow B')$
  - jos sellaisia ei ole olemassa, niin tämä vaihe on valmis
- jaetaan B kahdeksi lohkoksi:
  - $B_1 = \{q \in B \mid q =a\Rightarrow B'\}$ epätyhjä, koska sisältää ym. q_1:n
  - $B_2 = \{q \in B \mid \neg(q =a\Rightarrow B')\}$ epätyhjä, koska sisältää ym. q_2:n

Katsotaanpa ennen väitteen "eri kielet" todistusta esimerkki



- vihreä lohko
  - a:lla ei pääse minnekään mistään tilasta
  - b:llä pääsee keltaiseen lohkoon jokaisesta tilasta

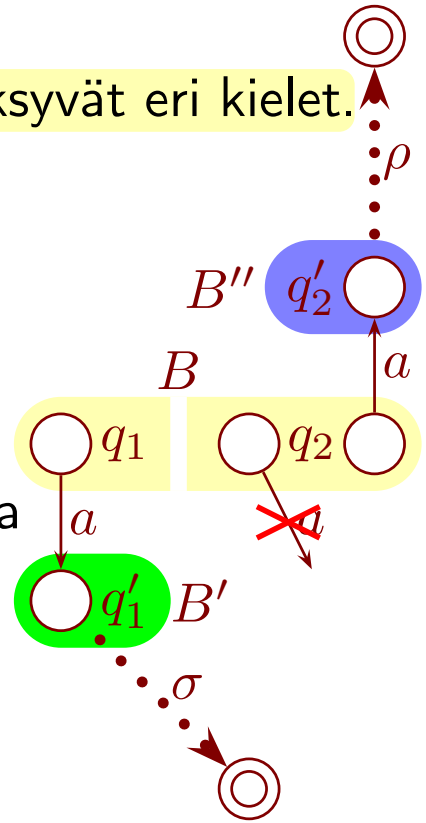
  $\Rightarrow$ ei halkea
- keltainen lohko
  - a:lla pääsee osasta tiloja keltaiseen, osasta muuhun (eli vihreään) lohkoon

  $\Rightarrow$ halkeaa

- keltainen lohko, toinen vaihtoehto
  - b:llä pääsee osasta tiloja keltaiseen lohkoon, osasta ei mihinkään

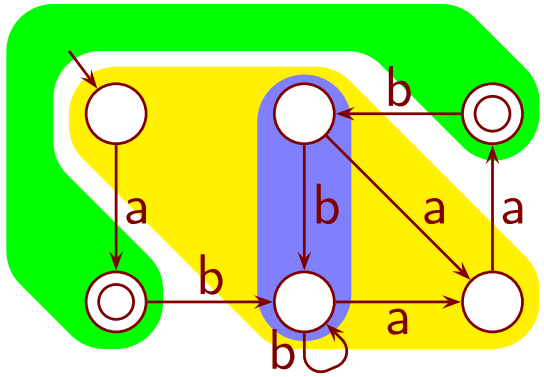
  $\Rightarrow$ halkeaa samalla tavalla

Apulause 2.2 Lohkomisessa säilyy se, että eri lohkoissa olevat tilat hyväksyvät eri kielet.



- täytyy todistaa, että jos $q_1 \in B_1$ ja $q_2 \in B_2$, niin $\mathcal{L}(q_1) \neq \mathcal{L}(q_2)$
  - $B_1 = \{q \in B \mid q =a\Rightarrow B'\}$
  - $B_2 = \{q \in B \mid \neg(q =a\Rightarrow B')\}$
- ennen lohkomista vallinneesta tilanteesta saa olettaa, että jos kaksi tilaa on eri lohkoissa, niin ne hyväksyvät eri kielet
  - todistamme, että algoritmi ei voi toimia väärin *ensimmäistä* kertaa
  - **invariantti** eli ominaisuus, jota algoritmi *ylläpitää* voimassa
  - vrt. induktio matematiikassa
- olkoon q_1' se tila, jolle $q_1 =a\Rightarrow q_1'$
- koska jokaisesta tilasta pääsee lopputilaan, $\exists\sigma : \sigma \in \mathcal{L}(q_1')$
- koska $\neg(q_2 =a\Rightarrow B')$, pätee joko $\neg(q_2 =a\Rightarrow)$ tai $q_2 =a\Rightarrow B''$, missä $B'' \neq B'$
- jos $\neg(q_2 =a\Rightarrow)$ niin $a\sigma \in \mathcal{L}(q_1)$ mutta $a\sigma \notin \mathcal{L}(q_2)$
- muutoin olkoon q_2' se tila, jolle $q_2 =a\Rightarrow q_2'$
  - koska $q_1' \in B' \neq B''$ ja $q_2' \in B''$, on $\mathcal{L}(q_1') \neq \mathcal{L}(q_2')$
  - jos $\exists\sigma \in \mathcal{L}(q_1') : \sigma \notin \mathcal{L}(q_2')$, niin $a\sigma \in \mathcal{L}(q_1)$ mutta $a\sigma \notin \mathcal{L}(q_2)$
  - muutoin $\exists\rho \in \mathcal{L}(q_2') : \rho \notin \mathcal{L}(q_1')$, joten $a\rho \in \mathcal{L}(q_2)$ mutta $a\rho \notin \mathcal{L}(q_1)$ □

Vaiheen 3 päättyminen ja lopputilanne



- jokainen lohkominen kasvattaa lohkojen määrää yhdellä

$\Rightarrow$ lohkominen päättyy viimeistään silloin, kun jokainen tila on yksin lohkossaan

$\Rightarrow$ lohkomisia voi olla enintään $|Q| - 1$

- lohkomisen päättymisehto takaa, että jokaiselle $a \in \Sigma$ ja lohkolle B pätee, että kaikki B:n tilat ovat samaa mieltä mihin lohkoon a vie
  - joko jokaiselle $q \in B$: $\delta(q, a)$ on määrittelemätön
  - tai jokaiselle $q \in B$: $\delta(q, a)$ on määritelty ja $\delta(q, a)$:t kuuluvat samaan lohkoon

lohkosta	a vie	b vie
vihreä	–	sininen
keltainen	vihreä	–
sininen	keltainen	sininen

Apulause 2.3 Jos q_0 ja q_0' kuuluvat samaan lohkoon, niin ne hyväksyvät saman kielen.

- tämä todistus on helppo ja kannattaa ymmärtää!
- olkoon $a_1 \cdots a_n \in \mathcal{L}(q_0)$

$\Rightarrow$ on olemassa $q_1, \ldots, q_n$ siten, että $q_0 =a_1\Rightarrow q_1 =a_2\Rightarrow \ldots =a_n\Rightarrow q_n \in F$

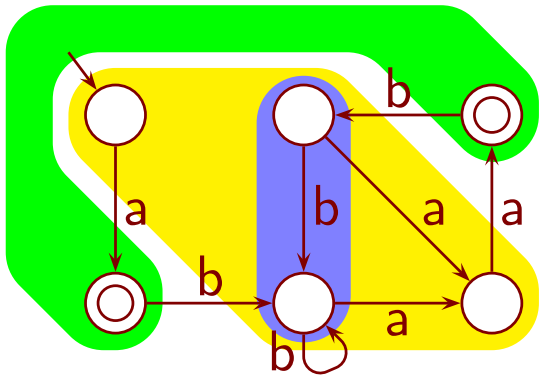
- koska q_0 ja q_0' kuuluvat samaan lohkoon, q_1:n lohkossa on q_1' siten, että $q_0' =a_1\Rightarrow q_1'$
- koska q_1 ja q_1' kuuluvat samaan lohkoon, q_2:n lohkossa on q_2' siten, että $q_1' =a_2\Rightarrow q_2'$

  $\ldots$
- koska q_{n-1} ja q_{n-1}' kuuluvat s. l., q_n:n lohkossa on q_n' siten, että $q_{n-1}' =a_n\Rightarrow q_n'$

- koska q_n ja q'_n kuuluvat samaan lohkoon ja $q_n \in F$, vaiheen 2 vuoksi $q'_n \in F$

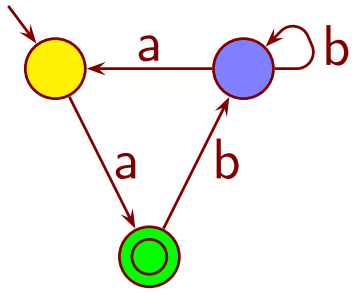
$\Rightarrow a_1 \cdots a_n \in \mathcal{L}(q'_0)$

- samoin $a_1 \cdots a_n \in \mathcal{L}(q'_0) \;\Rightarrow\; a_1 \cdots a_n \in \mathcal{L}(q_0)$ $\square$

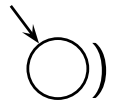


Minimointialgoritmin vaihe 4: lohkojen muuttaminen tiloiksi

- jokaisesta lohkosta muodostetaan tila lopputulokseen
- aakkosto säilyy muuttumattomana
- lohkosta lähtevät kaaret asetetaan lohkon jostakin tilasta lähtevien kaarien mukaan
  - lopputulos ei riipu siitä, mikä lohkon tiloista valitaan malliksi
- lohko merkitään lopputilaksi, jos ja vain jos sen tilat ovat lopputiloja
  - alkujaon ansiosta joko ne kaikki ovat tai yksikään ei ole
- alkutilaksi valitaan se lohko, johon alkuperäinen alkutila kuuluu



Kielen säilymisen ym. todistamiseksi olkoot

- $D = (Q, \Sigma, \delta, \hat{q}, F)$ vaiheen 1 lopputulos (muu kuin )
- $[D] = ([Q], \Sigma, [\delta], [\hat{q}], [F])$ lohkomisalgoritmin lopputulos
  - aakkosto ei tarvitse uutta symbolia, koska se ei muuttunut
- $[q]$ se lohko, johon q kuuluu
  - $[\hat{q}]$ tuli määriteltyä kahdesti, mutta samassa merkityksessä, koska vaiheessa 4 $[D]$:n alkutilaksi asetettiin $[\hat{q}]$

Jokaiselle $q \in Q$ ja $q' \in Q$ pätee

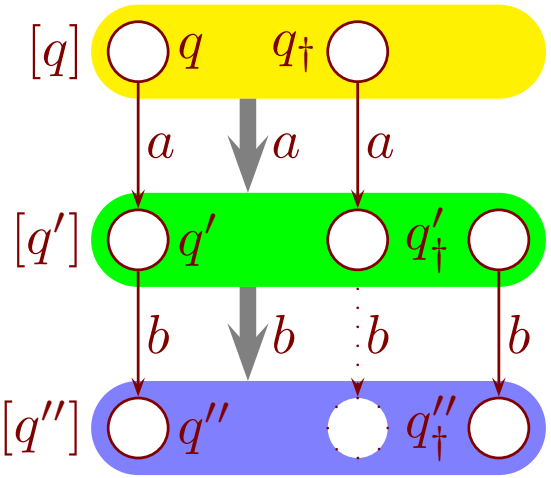
- $q \in [q]$
- jos $q' \in [q]$, niin $[q'] = [q]$

Apulause 2.4 Jos $q \mathrel{=}a\Rightarrow q'$, niin $[q] \; [\mathrel{=}a\Rightarrow] \; [q']$.

- $[q]$:ssa on jokin tila $q_\dagger$ jonka mukaan $[q]$:n lähtökaaret asetettiin vaiheessa 4
- $[q]$:n tilat ovat keskenään samaa mieltä lähtökaarien määränpäälohkoista

$\Rightarrow q_\dagger \mathrel{=}a\Rightarrow q'_\dagger$ jollekin $q'_\dagger \in [q']$

$\Rightarrow [q] = [q_\dagger] \; [\mathrel{=}a\Rightarrow] \; [q'_\dagger] = [q']$ □



Kielen säilyminen, osa 1: lopputulos hyväksyy kaiken minkä alkuperäinenkin

- olkoon $a_1 \cdots a_n \in \mathcal{L}(D)$

$\Rightarrow$ on olemassa $q_1, \ldots, q_n$ siten, että $\hat{q} \mathrel{=}a_1\Rightarrow q_1 \mathrel{=}a_2\Rightarrow \ldots \mathrel{=}a_n\Rightarrow q_n \in F$

- apulauseen 2.4 nojalla $[\hat{q}] \; [\mathrel{=}a_1\Rightarrow] \; [q_1] \; [\mathrel{=}a_2\Rightarrow] \ldots [\mathrel{=}a_n\Rightarrow] \; [q_n]$
- koska $q_n \in F$, vaiheessa 4 asetettiin $[q_n] \in [F]$

$\Rightarrow a_1 \cdots a_n \in \mathcal{L}([D])$ □

Apulause 2.5 Jos $[q]\ [{=}a{\Rightarrow}]\ B'$, niin on olemassa $q' \in B'$ siten, että $q\ {=}a{\Rightarrow}\ q'$.

- on olemassa $q_\dagger \in [q]$ ja $q'_\dagger \in B'$ joiden mukaan $[q]\ [{=}a{\Rightarrow}]\ B'$ asetettiin vaiheessa 4
- $[q]$:n kaikki tilat ovat samaa mieltä lähtökaarien määränpäälohkoista

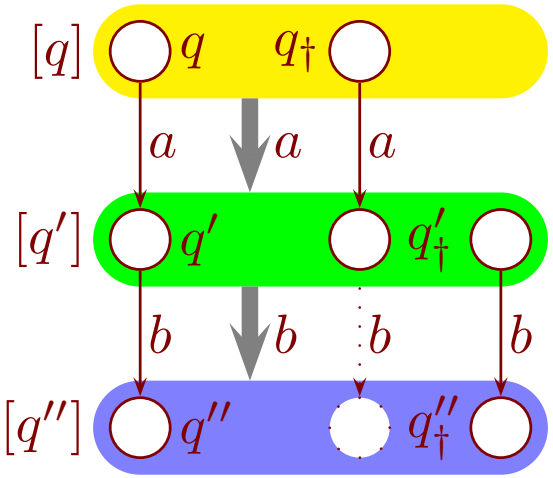
$\Rightarrow$ on olemassa $q' \in B'$ siten, että $q\ {=}a{\Rightarrow}\ q'$ □

Kielen säilyminen, osa 2: lopputulos ei hyväksy enempää kuin alkuperäinen

- olkoon $a_1 \cdots a_n \in \mathcal{L}([D])$

$\Rightarrow [\hat{q}]\ [{=}a_1{\Rightarrow}]\ B_1\ [{=}a_2{\Rightarrow}]\ \ldots\ [{=}a_n{\Rightarrow}]\ B_n \in [F]$

- apulauseen 2.5 nojalla
  - on olemassa $q_1 \in B_1$ siten, että $\hat{q}\ {=}a_1{\Rightarrow}\ q_1$
  - on olemassa $q_2 \in B_2$ siten, että $q_1\ {=}a_2{\Rightarrow}\ q_2$
  - $\ldots$
  - on olemassa $q_n \in B_n$ siten, että $q_{n-1}\ {=}a_n{\Rightarrow}\ q_n$
- koska $B_n \in [F]$, vaiheen 4 vuoksi jokainen sen tila on lopputila $\Rightarrow q_n \in F$



$\Rightarrow a_1 \cdots a_n \in \mathcal{L}(D)$ □

Olemme todistaneet seuraavan:

Lause 2.6 $\mathcal{L}([D]) = \mathcal{L}(D)$.

Lopputuloksen minimaalisuus ja yksikäsitteisyys ovat vielä todistamatta

Apulause 2.7 $[D]$:ssä ei ole turhia tiloja.

- olkoon $B \in [Q]$
- B on jokin Q:sta muodostettu lohko, ja lohkot eivät ole tyhjiä
  $\Rightarrow$ on olemassa $q \in B$
- vaiheen 1 ansiosta D:ssä ei ole turhia tiloja
  - on olemassa σ, ρ ja q' siten, että $\hat{q} =\sigma\Rightarrow q =\rho\Rightarrow q' \in F$
- apulauseen 2.4 nojalla $[\hat{q}]\ [=\sigma\Rightarrow]\ B\ [=\rho\Rightarrow]\ [q'] \in [F]$

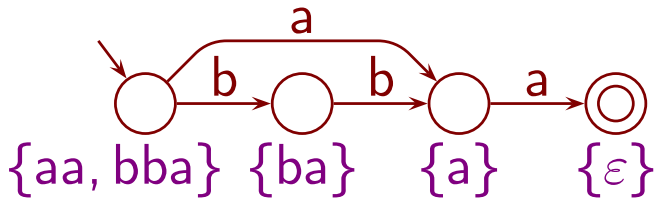
$\Rightarrow B$ ei ole turha $\square$

Apukäsite: kielen loppuosa

- olkoot $K \subseteq \Sigma^*$ ja $\sigma \in \Sigma^*$
  - ts. K on joukko merkkijonoja ja σ on yksi merkkijono Σ:n merkeistä
- määrittelemme: K:n **loppuosa σ:n jälkeen** on $\sigma^{-1}K = \{\rho \mid \sigma\rho \in K\}$
- esim. olkoon $K = \{\text{heppa, kissa, kirahvi, koi, koira, norsu}\}$
  - $(\text{ki})^{-1}K = \{\text{ssa, rahvi}\}$, $(\text{koi})^{-1}K = \{\varepsilon, \text{ra}\}$ ja $(\text{norppa})^{-1}K = \emptyset$
- K:n loppuosa ε:n jälkeen on $\varepsilon^{-1}K = K$
- $\sigma^{-1}K \neq \emptyset$ jos ja vain jos K:ssa on merkkijono, joka alkaa σ:lla
- jos $\sigma \in K$, niin $\varepsilon \in \sigma^{-1}K$
- määrittelemme: L on K:n **loppuosa**, jos ja vain jos $\exists\sigma : L = \sigma^{-1}K$
- em. esimerkkikielen loppuosia ovat lisäksi mm. {issa, irahvi, oi, oira}, {u} ja $\{\varepsilon\}$

Lopputuloksen minimaalisuus

- todistaaksemme, että $[D]$ on pienin mahdollinen, osoitamme, että jokaisessa DFA:ssa, joka hyväksyy $\mathcal{L}(D)$:n, täytyy olla ainakin "samat" tilat ja kaaret
  - "samat" ei voi nyt tarkoittaa saman nimiset, koska tilojen nimet ovat mielivaltaiset
  - toimiva "samuus" saadaan tilojen hyväksymistä kielistä kuten kohta näkyy
- olkoon $D' = (Q', \Sigma', \delta', \hat{q}', F')$ mikä tahansa DFA siten, että $\mathcal{L}(D') = \mathcal{L}(D)$



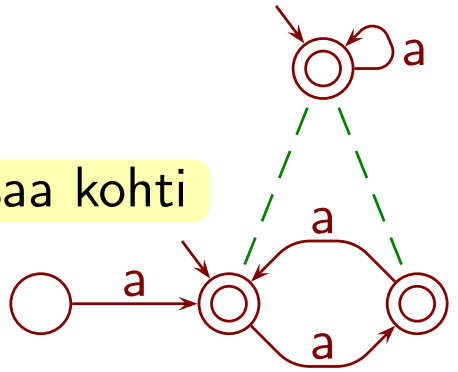
- jos $\hat{q}' =\sigma\Rightarrow' q'$, niin $\mathcal{L}(q') = \sigma^{-1}\mathcal{L}(D)$
- jos $\sigma^{-1}\mathcal{L}(D) \neq \emptyset$, niin on olemassa q' siten, että $\hat{q}' =\sigma\Rightarrow' q'$ ja $\mathcal{L}(q') = \sigma^{-1}\mathcal{L}(D)$
- jos $\sigma^{-1}\mathcal{L}(D) = \emptyset$, niin tila q' siten, että $\hat{q}' =\sigma\Rightarrow' q'$, on tai ei ole olemassa
  - jos on olemassa, niin siitä ei ole polkua mihinkään lopputilaan, joten se on turha
- olkoot σ_1 ja σ_2 mitkä tahansa siten, että $\sigma_1^{-1}\mathcal{L}(D) \neq \emptyset$ ja $\sigma_2^{-1}\mathcal{L}(D) \neq \emptyset$

$\Rightarrow$ on olemassa $q_1' \in Q'$ ja $q_2' \in Q'$ siten, että $\hat{q}' =\sigma_1\Rightarrow' q_1'$ ja $\hat{q}' =\sigma_2\Rightarrow' q_2'$

- vaikka olisi $\sigma_1 \neq \sigma_2$, voi silti olla $q_1' = q_2'$ (ja usein onkin)
- jos kuitenkin $\mathcal{L}(q_1') \neq \mathcal{L}(q_2')$, niin $q_1' \neq q_2'$

$\Rightarrow$ D':ssa on erillinen tila jokaista $\mathcal{L}(D)$:n erilaista epätyhjää loppuosaa kohti

- D':ssa voi olla samalle loppuosalle monta tilaa
- D':ssa voi olla myös turhia tiloja



- $[D]$:ssä on
  - vain epätyhjiä loppuosia vastaavia tiloja, koska siinä ei ole turhia (apulause 2.7)
  - kullekin $\mathcal{L}(D)$:n epätyhjälle loppuosalle vain yksi tila (apulause 2.2)

$\Rightarrow$ D':ssa on erillinen tila q' jokaista $[D]$:n tilaa $[q]$ kohti siten, että $\mathcal{L}(q') = \mathcal{L}([q])$

$\Rightarrow$ $[D]$:ssa on niin vähän tiloja kuin mahdollista kun hyväksyttynä kielenä on $\mathcal{L}(D)$

- tarkastellaan $[D]$:n mitä tahansa kaarta $[q_1]$ $[=a\Rightarrow]$ $[q_2]$
- D':ssa on oltava $\mathcal{L}([q_1])$:n hyväksyvä tila q'
- koska $[D]$:ssä ei ole turhia tiloja, on olemassa σ siten, että $\sigma \in \mathcal{L}([q_2])$
- $a\sigma \in \mathcal{L}([q_1])$, joten $\delta'(q', a)$:n on oltava määritelty

$\Rightarrow$ D':ssa on erillinen kaari jokaista $[D]$:n eri kaarta kohti

$\Rightarrow$ $[D]$:ssa on niin vähän kaaria kuin mahdollista kun hyväksyttynä kielenä on $\mathcal{L}(D)$

$\Rightarrow$ $[D]$ on niin pieni kuin mahdollista kun hyväksyttynä kielenä on $\mathcal{L}(D)$

Minimoidun DFA:n yksikäsitteisyys

- oletetaan, että $|Q'| = |[Q]|$

$\Rightarrow$ Q':ssa on yksi tila $\mathcal{L}(D)$:n jokaista epätyhjää loppuosaa kohti, eikä muita tiloja
  - merkitsemme epätyhjää loppuosaa K vastaavaa tilaa $q'(K)$

- Q':n alkutilan täytyy hyväksyä $\mathcal{L}(D)$
  $\Rightarrow \hat{q}' = q'(\mathcal{L}(D))$
- jos $a^{-1}\mathcal{L}(q'_1) \neq \emptyset$, niin $\exists q'_2 : q'_1 =a\Rightarrow q'_2$, ja jos $q'_1 =a\Rightarrow q'_2$, niin $\mathcal{L}(q'_2) = a^{-1}\mathcal{L}(q'_1)$

$\Rightarrow \delta'(q'(K), a)$ on
  - määrittelemätön, jos $a^{-1}K = \emptyset$ koska turhia tiloja ei mahdu koska $|Q'| = |[Q]|$
  - $a^{-1}K$, muutoin
- $q'(K) \in F'$ jos ja vain jos $\varepsilon \in K$

$\Rightarrow$ Q':n muut osat määräytyivät yksikäsitteisesti paitsi Σ'

$\Rightarrow$ jos $|Q'| = |[Q]|$, niin D' on tilojen nimiä ja aakkostoa vaille sama kuin $[D]$

- tämä ei ole mitäänsanomaton tulos, koska
  - vastaava ei päde NFA:ille
  - se takaa, että lopputulos ei riipu siitä, missä järjestyksessä lohkoja halkaistaan

Olemme todistaneet seuraavan:

Lause 2.8 Pienin mahdollinen DFA, joka hyväksyy saman kielen kuin syötteenä annettu DFA, on aakkostoa ja tilojen nimiä vaille yksikäsitteinen. Tässä alaluvussa kuvattu algoritmi tuottaa sen.

Mitä kaiken kaikkiaan todistimme, ja miksi?

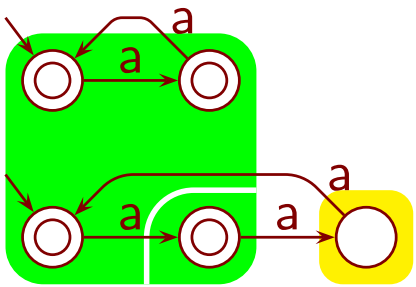
- turhien tilojen poiston tarve ja oikeellisuus on ilmeistä
  - jos alkutilasta ei pääse tilaan, se ei vaikuta kieleen
  - se voi vaikuttaa (mutta ei välttämättä vaikuta) tilojen määrään, jollei sitä poisteta
  - tyhjää kieltä vastaavia tiloja ei tarvita, koska δ saa olla osittainen

- lohkomisen valmistuttua
  1. saman kielen hyväksyvät tilat ovat samassa lohkossa
  2. eri kielen hyväksyvät tilat ovat eri lohkossa

  $\Rightarrow$ lohkot vastaavat yksi yhteen tilojen hyväksymiä kieliä
- 1. taattiin
  - rakentamalla lohkot aluksi sen mukaisesti (vaihe 2)
  - varmistamalla, että ominaisuus säilyy lohkomisen ajan (apulause 2.2)
- 2. taattiin vaiheen 3 loppuehdon avulla (apulause 2.3)
- tämä on tavallinen hahmo algoritmien pääsilmukoissa
  - sekä alku- että lopputilanne ovat jonkin yleisemmän väittämän erikoistapauksia
  - silmukan aikana pidetään huolta, ettei väittämää rikota
  - väittämästä ja silmukan lopetusehdosta yhdessä seuraa tavoite
  - tällaista väittämää kutsutaan *invariantiksi*
- täytyy varmistaa myös, että pääsilmukka lopettaa joskus
  - varmistimme, että lohko ei halkea tyhjäksi lohkoksi ja alkuperäiseksi lohkoksi

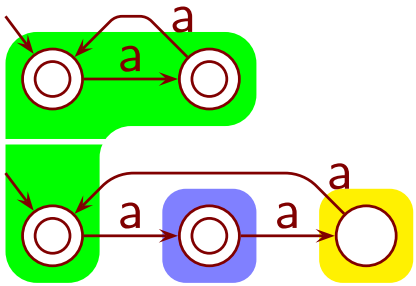
  $\Rightarrow$ jokainen kierros vie työtä eteenpäin
- tyhjiä lohkoja vältettiin myös siksi, että niistä tulisi turhia tiloja lopputulokseen
  - varmistettiin myös alkujaossa
  - tarvittiin apulauseen 2.7 todistuksessa

- apulauseet 2.4 ja 2.5 sekä lause 2.6 sanovat vain, että kieli ei muutu kun kaikkialla saman kielen hyväksyvät tilat yhdistetään yhdeksi tilaksi (vaihe 4)
  - yhdistetyn tilan lähtökaaret ja luonne (alku-, loppu-) määräytyy yksikäsitteisesti
  - intuitiivisesti ilmeistä
  - hieman kömpelöä ilmaista matemaattisesti
- minimaalisuus tarkoittaa, että pienempää saman kielen hyväksyvää DFA:ta ei ole
  - onko 5 tilaa ja 5 kaarta pienempi, yhtäsuuri vai suurempi kuin 4 tilaa ja 6 kaarta?
  - onneksi lopputulos on minimaalinen yhtäaikaa tilojen ja kaarien määrien suhteen
- yksikäsitteisyys tarkoittaa, että kaikki minimaaliset ovat rakenteeltaan samanlaisia
  - aakkoston samuus ei seuraa automaattisesti, mutta on tapana vaatia erikseen
  - tilojen nimet voivat olla mitä vain, niistä ei voi päätellä mitään
  - tällainen nimistä riippumaton samanlaisuus on *isomorfismi*



- niiden todistamiseksi luonnehdimme minimaalisen DFA:n, ja osoitimme, että $[D]$ on luonnehdinnan mukainen
  - tilat vastaavat yksi yhteen hyväksytyn kielen epätyhjiä loppuosia
  - kaaret, alkutila ja lopputilat määräytyvät tästä yksikäsitteisesti
  - aakkosto määräytyy vain sen verran, että sisältää ainakin kaikki käytetyt merkit

Apulauseiden 2.2 ja 2.3 ansiosta kahden DFA:n hyväksymien kielten samuus voidaan testata seuraavasti:



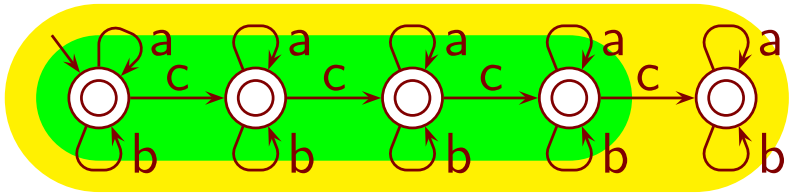
- annetaan ne yhtäaikaa lohkomisalgoritmin syötteeksi
- katsotaan, joutuvatko niiden alkutilat eri lohkoihin vaiheessa 2 tai 3

Miten lohkominen toteutetaan tehokkaasti?

- syötteen koon osatekijät ovat $|Q|$, $|\Sigma|$ ja $|\delta|$
- turhat tilat on poistettu
  $\Rightarrow$ jokaiseen tilaan paitsi ehkä $\hat{q}$ tulee ainakin yksi kaari
  $\Rightarrow |\delta| \geq |Q| - 1$
  $\Rightarrow$ jos Q on iso, myös δ on iso

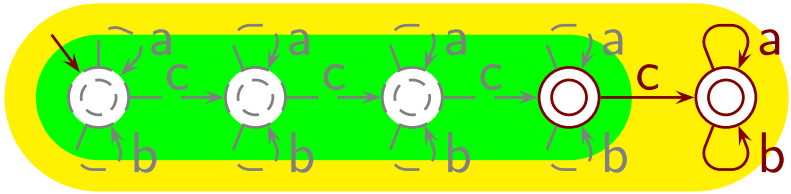
Helposti mieleen tuleva algoritmi

- käydään lohkoja läpi katsoen halkeavatko ne
  - kullekin aakkoselle a ja lohkon tilalle q tutkitaan $\delta(q, a)$
- yläraja työmäärälle
  - yhtä lohkomista varten tutkitaan (melkein) jokainen kaari
  - $|Q| - 1$ lohkomista
  $\Rightarrow O(|Q||\delta|)$
- yläraja voi toteutua
  - $|\delta| = |Q||\Sigma| - 1$
  - kaaria tutkitaan $|\Sigma|(|Q| + \ldots + 2 + 1) - 1$
    $= \frac{1}{2}|\Sigma||Q|(|Q| + 1) - 1 = \frac{1}{2}(|\delta| + 1)(|Q| + 1) - 1 = \Theta(|Q||\delta|)$ kpl

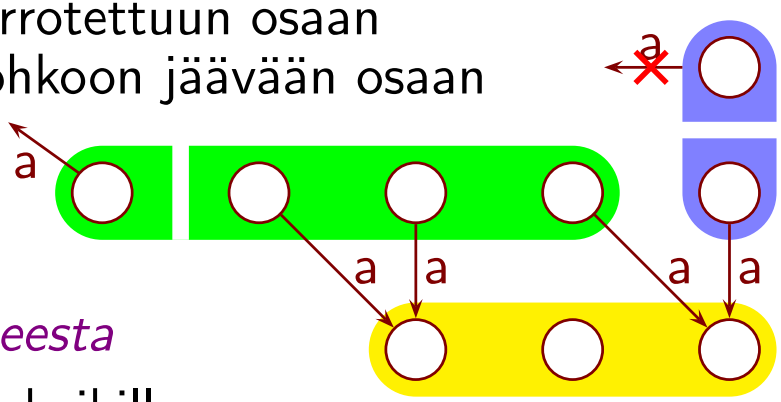


$\Rightarrow$ hidas, jos Q on iso

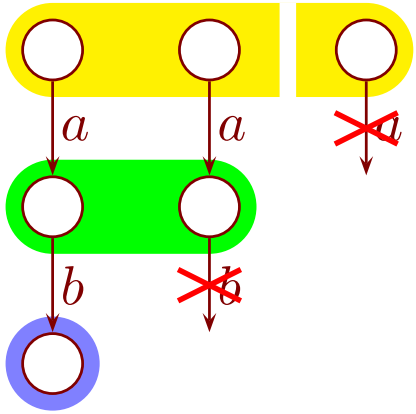
Hopcroft'n kuuluisat tehostuskikat vuodelta 1971



- kaaria kuljetaankin takaperin eikä etuperin
  - kaaren alkupään tila *merkitään*
- lohkominen hoidetaan siten, että
  - lohko jakaantuu lohkoon jäävään osaan ja siitä irrotettuun osaan
  - niihin tiloihin ei kosketa *lainkaan*, jotka jäävät lohkoon jäävään osaan



- yksi lohkomisvaihe
  - käyttää yhtä kärkipäälohkoa ja yhtä aakkosta
  - halkaisee nolla tai useampia lohkoja
  - koostuu *merkitsemisvaiheesta* ja *halkaisemisvaiheesta*
- jos lohko halkeaa kun se on jo käytetty lohkomiseen kaikilla aakkosilla, riittää käyttää jatkossa jompaa kumpaa osaa
  - helppo uskoa, (oli) vaikea todistaa kunnolla

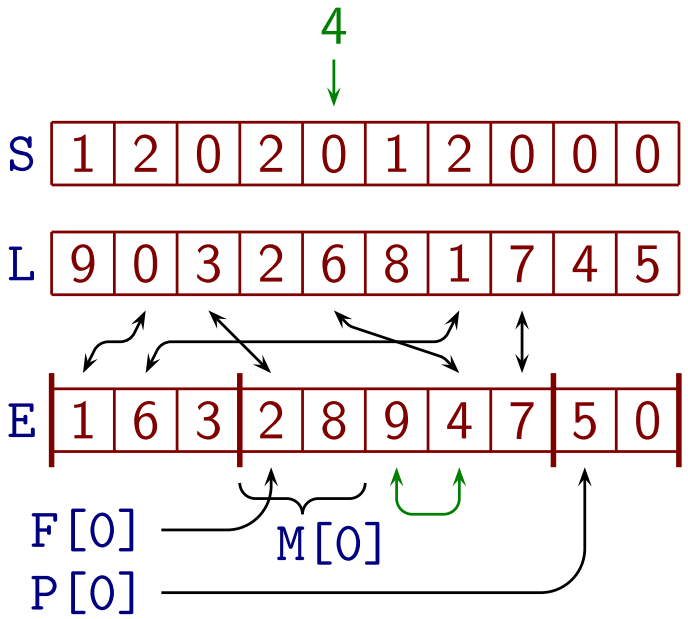


- jatkossa käytetään pienempää osaa
  - jos samaa kaarta käytetään lohkomiseen uudelleen, sen kärkitilan lohko on kooltaan enintään puolet aikaisemmasta

  $\Rightarrow$ kutakin kaarta voidaan käyttää enintään $\lfloor \log_2 |Q| \rfloor + 1$ kertaa

$\Rightarrow$ työmäärä $O(|\Sigma||Q| \log |Q|)$
- jos tilasiirtymäfunktio on täysi, niin on sama kuin $O(|\delta| \log |Q|)$
- muutoin on huonompi kuin $O(|\delta| \log |Q|)$
- Hopcroft oletti että, tilasiirtymäfunktio on täysi ja $|\Sigma| = 2$

Tietorakenne lohkojen esittämiseen, vuoden 2012 versio



- tilojen numerot ovat taulukossa `E` siten, että saman lohkon tilat ovat peräkkäin
- lohkon `s` tiedot ovat kolmessa taulukossa
  - `F[s]` alkukohta
  - `P[s]` loppukohta + 1
  - `M[s]` merkittyjen tilojen määrä
- `L[e]` kertoo paikan `E`:ssä, jossa tila `e` on
- `S[e]` kertoo lohkon, jossa tila `e` on
- pino `W` pitää kirjaa lohkoista, joissa on merkittyjä tiloja
  - `w` on pinossa olevien lohkojen määrä
- tila merkitään vaihtamalla se lohkonsa ensimmäisen merkitsemättömän tilan kanssa ja kasvattamalla merkittyjen määrää
  - tarvittaessa tilan lohko lisätään koskettujen lohkojen pinoon
  - vakioaikaista

```
void mark( int e ){
  int s = S[e], i = L[e], j = F[s]+M[s];
  E[i] = E[j]; L[E[i]] = i;
  E[j] = e; L[e] = j;
  if( !M[s]++ ){ W[w++] = s; }
}
```

- ei tarvitse varautua saman tilan merkitsemiseen kahdesti
  - tilasta lähtee kullakin aakkosella korkeintaan yksi kaari

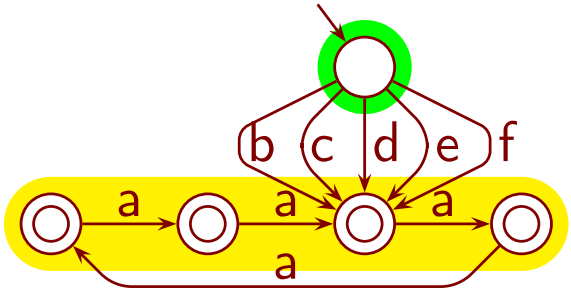
  $\Rightarrow$ tilaan tullaan takaperin korkeintaan kerran yhden lohkomisvaiheen aikana
- kunkin merkitsemisvaiheen jälkeen merkityt lohkot halkaistaan tarpeen mukaan
  - `z` on lohkojen määrä
  - jos kaikki tilat merkittiin, lohkoa ei halkaista, vaan merkittyjen määrä nollataan
  - muutoin pienemmästä osasta tehdään uusi lohko ja se saa numerokseen `z`
  - uuden lohkon tilat käydään läpi oikean lohkon asettamiseksi niille
  - vanhan ja uuden lohkon merkittyjen määrä nollataan
  - työmäärä on enintään verrannollinen saman lohkomisvaiheen `mark`-kutsujen määrään

  $\Rightarrow$ ei nosta koko algoritmin asymptoottista ajankäyttöä

```
void split(){
  while( w ){
    int s = W[--w], j = F[s]+M[s];
    if( j == P[s] ){ M[s] = 0; continue; }
    if( M[s] <= P[s]-j ){ F[z] = F[s]; P[z] = F[s] = j; }
    else{ P[z] = P[s]; F[z] = P[s] = j; }
    for( int i = F[z]; i < P[z]; ++i ){ S[E[i]] = z; }
    M[s] = M[z++] = 0;
  }
}
```

Ajankäytön parantaminen $O(|\Sigma||Q|\log|Q|) \rightsquigarrow O(|\delta|\log|Q|)$ vuonna 2012

- Hopcroft'n kikat eivät hyödynnä sitä, että usein $|\delta|$ on paljon pienempi kuin $|\Sigma||Q|$
- ongelma: niiden tilojen selaamiseen kuluva aika, joihin ei tule kaarta tutkittavalla aakkosella
- ratkaisu: *kimput*
  - joukko samannimisiä kaaria



- kaksi samannimistä kaarta pannaan eri kimppuihin, (vain) kun on varmaa, että ne päättyvät eri lohkoihin
- kimpuista muodostetaan samanlainen tietorakenne kuin lohkoista
  - `B` on lohkot (blocks)
  - `C` on kimput (cords)
- vuorotellen
  - kimppujen avulla halkaistaan lohkoja
  - lohkojen avulla halkaistaan kimppuja

  kunnes uusia lohkoja ja kimppuja ei enää synny
- loppuehdon takaavat invariantit: olkoon $a \in \Sigma$ ja kuulukoot q_1 ja q_2 samaan lohkoon
  - jos kaksi saman kimpun kaarta päättyy eri lohkoihin, ainakin toinen lohkoista on käyttämättä kimppujen halkaisemiseen
  - jos sekä q_1:stä että q_2:sta lähtee a-kaari ja ne kuuluvat eri kimppuihin, ainakin toinen kimpuista on käyttämättä lohkojen halkaisemiseen
  - jos q_1:stä lähtee ja q_2:sta ei lähde a-kaari, ko. kaaren kimppu on käyttämättä

Lohkominen ohjelmakoodina

- `b` kertoo vuorossa olevan lohkon numeron
- `c` kertoo vuorossa olevan kimpun numeron
- `T[k]` on sen tilan numero, josta kaari `k` alkaa
- tilaan `q` tulevien kaarten numerot ovat taulukon `A` kohdissa `F[q]`, `F[q]+1`, `F[q]+2`, ..., `F[q+1]-1`

```
int b = 1, c = 0, i, j;
while( c < C.z ){
  for( i = C.F[c]; i < C.P[c]; ++i ){
    B.mark( T[C.E[i]] );
  }
  B.split(); ++c;
  while( b < B.z ){
    for( i = B.F[b]; i < B.P[b]; ++i ){
      for( j = F[B.E[i]]; j < F[B.E[i]+1]; ++j ){
        C.mark( A[j] );
      }
    }
    C.split(); ++b;
  }
}
```

Lohkomisalgoritmin suoritusaika

- jos tilaa käytetään uudelleen halkaisemaan kimppua,
  sen lohko on kooltaan enintään puolet aikaisemmasta
  $\Rightarrow$ tilaa käytetään yhteensä enintään $O(\log |Q|)$ kertaa
- jos kaarta käytetään uudelleen halkaisemaan lohkoa,
  sen kimppu on kooltaan enintään puolet aikaisemmasta
  - kimpun maksimikoko on $|Q|$, koska kimpun kaarilla on sama nimi

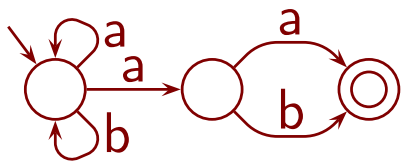
  $\Rightarrow$ kaarta käytetään yhteensä enintään $O(\log |Q|)$ kertaa
- tiloja ja kaaria on yhteensä $|Q| + |\delta| = \Theta(\delta)$
  - aiemmin todettiin, että $|\delta| \geq |Q| - 1$

$\Rightarrow$ suoritusaika on $O(|\delta| \log |Q|)$

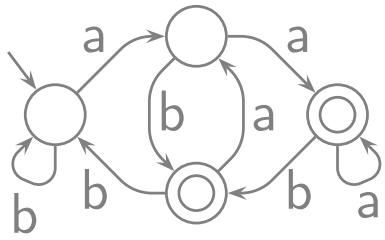
Huomautus suoritusajasta

- $O(|\delta| \log |Q|)$ olettaa, että kaaret saadaan syötteessä järjestettynä nimien mukaan
  - tämä järjestys tarvitaan alkuperäisten kimppujen muodostamiseen
- $O(|\delta| \log |Q|)$ ei riitä kaarten järjestämiseen nimien mukaan tutuilla algoritmeilla
- jos syötteeltä ei voida tätä olettaa, ajan kulutus nousee arvoon $O(|\delta| \log |\delta|)$
  - kaarten järjestäminen esim. heapsortilla vie näin kauan
  - koska myös $O(|\delta| \log |\delta|)$ on hyvin nopea, tämä ei ole ongelma
  - on teoriassa vältettävissä tietorakenteella, joka vie tolkuttomasti muistia
  - on käytännössä keskimäärin vältettävissä hajautustauluilla

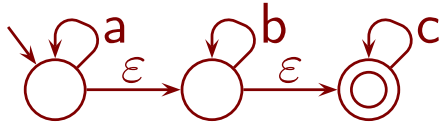
2.3 Epädeterministiset äärelliset automaatit



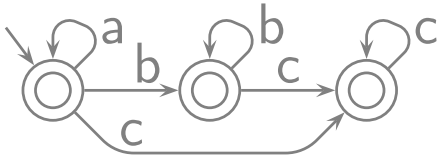
Epädeterministisen äärellisen automaatin käsitteessä on kaksi eroa deterministisen äärellisen automaatin käsitteeseen:



- samasta tilasta saa samalla nimellä lähteä monta kaarta
  - samasta tilasta samalla nimellä samaan tilaan enintään yksi
  - merkkijonoa luettaessa valitaan vaihtoehdoista jokin
  - esim. toiseksi viimeinen merkki on a, muut a tai b
- kaaren nimenä saa olla ε
  - muistamme, että ε ei ole merkki, vaan tyhjän merkkijonon symboli
  - kuljettaessa ε-kaari ei lueta merkkiä
  - esim. ensin ≥ 0 a:ta, sitten ≥ 0 b:tä, lopuksi ≥ 0 c:tä



- merkkijono hyväksytään, jos ja vain jos *ainakin yksi* tapa kulkea sen ohjaamana vie lopputilaan



Englanniksi *nondeterministic finite automaton*

- lyhenne **NFA**

Miksi NFA?

- NFA on usein helpompi laatia kuin saman kielen hyväksyvä DFA
- ”kuuluuko syöte kieleen” on DFA:lle hyvin helppo
- NFA:lle se on vaikeampi, mutta ei kohtuuttoman vaikea tietokoneelle

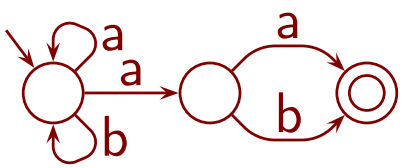
NFA antaa lisää vapauksia DFA:han verrattuna, mutta ei kiellä mitään entisiä

⇒ jokainen deterministinen äärellinen automaatti on epädeterministinen äärellinen automaatti
  - mutta ei toisinpäin
  - määritelmissä on tekninen ero, joka ei vaikuta oleelliseen sisältöön

⇒ termi "epädeterministinen" on harhaanjohtava

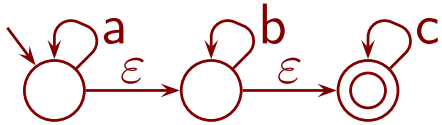
⇒ jotkut käyttävät NFA:sta termiä äärellinen automaatti, FA
  - jotta asia olisi oikein sekava, on myös heitä, joille FA = DFA

- tämän kaltaista esiintyy matematiikan kielenkäytössä jonkin verran, esim.
  - jokainen täysi funktio on osittainen funktio
  - jokainen täysi järjestys on osittainen järjestys
  - myös yhtäsuuruus on osittainen järjestys!

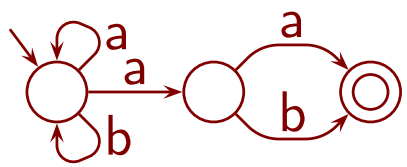


Määritelmä

NFA on viisikko $(Q, \Sigma, \Delta, \hat{q}, F)$, missä
- Q on äärellinen joukko
- Σ on äärellinen joukko siten, että $\varepsilon \notin \Sigma$
- $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$
- $\hat{q} \in Q$
- $F \subseteq Q$

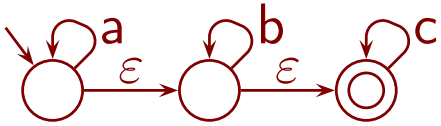


Ainoa ero DFA:n määritelmään on Δ



- Δ on **tilasiirtymärelaatio**
- sen alkiot ovat kolmikoita (q, a, q'), missä
  - q on se tila, josta tilasiirtymä alkaa
  - a on tilasiirtymän varrelle kirjoitettu merkki tai ε
  - q' on se tila, johon tilasiirtymä päättyy

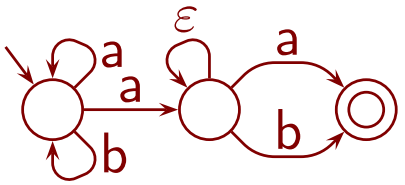
$q \Rightarrow^{\sigma} q'$ NFA:n tapauksessa



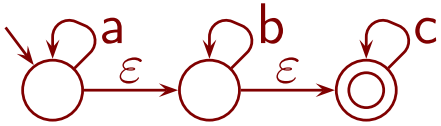
- $q \Rightarrow^{\varepsilon} q'$ jos ja vain jos on olemassa $q_0, \ldots, q_n$ siten, että $q = q_0$, $q_n = q'$ ja $\forall i; 1 \leq i \leq n : (q_{i-1}, \varepsilon, q_i) \in \Delta$
  - toisin sanoen, $q \Rightarrow^{\varepsilon} q'$ jos ja vain jos q:sta on q':uun polku, jonka jokaisen kaaren nimenä on ε
- kun $n > 0$, $q \Rightarrow^{a_1 \cdots a_n} q'$ jos ja vain jos on olemassa $q_1, q_1', \ldots, q_n, q_n'$ siten, että
  - $q \Rightarrow^{\varepsilon} q_1$
  - $q_n' \Rightarrow^{\varepsilon} q'$
  - $\forall i; 1 \leq i \leq n : (q_i, a_i, q_i') \in \Delta$
  - $\forall i; 2 \leq i \leq n : q_{i-1}' \Rightarrow^{\varepsilon} q_i$
- toisin sanoen, $q \Rightarrow^{a_1 \cdots a_n} q'$ jos ja vain jos q:sta on q':uun polku, jonka kaarien nimien jono ε:t pois lukien on $a_1 \cdots a_n$
- toisin kuin DFA:n tapauksessa, q:lle ja σ:lle voi olla monta q' siten, että $q \Rightarrow^{\sigma} q'$
  - epädeterminismi

NFA:n hyväksymä kieli määritellään samoin kuin DFA:n hyväksymä kieli

$$\mathcal{L}(N) = \{\sigma \in \Sigma^* \mid \exists q \in F : \hat{q} =\sigma\Rightarrow q\}$$



- NFA:n epädeterminismi on "ystävällistä"
  - riittää, että yksikin tapa lukea syöte vie lopputilaan
- ohjelmointikielten epädeterminismi on "vihamielistä"
  - ohjelmassa on virhe, jos yksikin tapa suorittaa se aiheuttaa virhetoiminnon
  - esim. `A[i] = i++;`



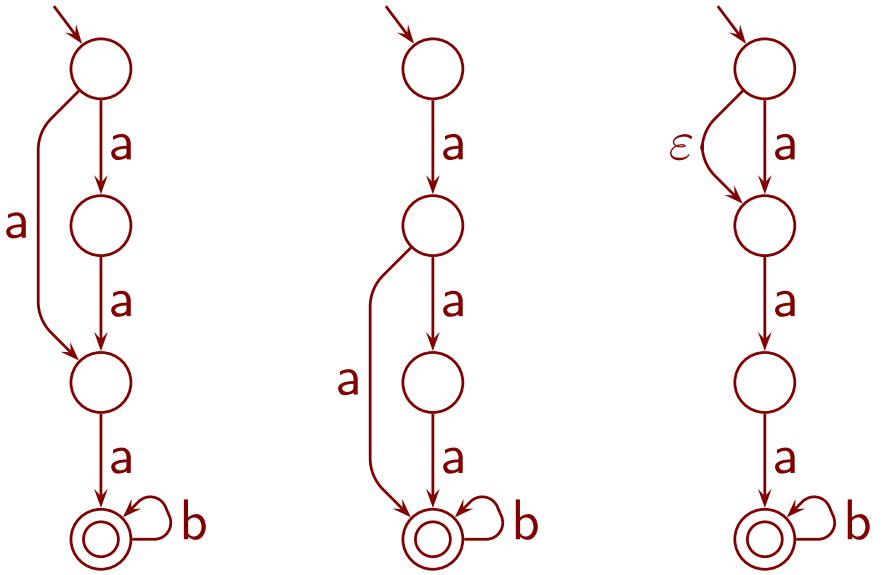
Pienin annetun kielen hyväksyvä NFA ei välttämättä ole yksikäsitteinen

- merkittävä ero DFA:in verrattuna!
- jos $aaa \in \mathcal{L}(N)$ mutta mikään pitempi a-jono ei kuulu, niin aaa:n hyväksyvä lukeminen ei saa kiertää ei-ε silmukkaa

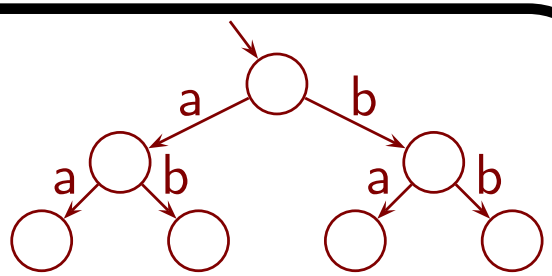
$\Rightarrow$ on oltava ainakin 4 eri tilaa ja 3 a-kaarta

- koska $aaab \in \mathcal{L}(N)$, on oltava b-kaari
- koska $aab \in \mathcal{L}(N)$, on oltava vielä jotakin

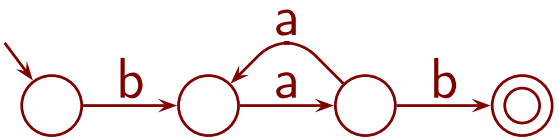
$\Rightarrow$ kuvan NFA:t ovat minimaalisia ko. kielelle



Mitä kieliä NFA:t hyväksyvät ja eivät hyväksy?



- jokaiselle äärelliselle kielelle on olemassa DFA, joka hyväksyy sen
- NFA:illa hyväksyttäviä äärettömiä kieliä rajoittaa voimakkaasti se, että pitkän merkkijonon hyväksyvä suoritus sisältää silmukan, jossa luetaan ainakin yksi merkki
  - pitkä = vähintään yhtä monta merkkiä kuin NFA:ssa on tiloja
  - NFA hyväksyy myös kaikki ne merkkijonot, jotka saadaan kiertämällä ko. silmukka useasti tai nolla kertaa
  - esim. $\{bab, baaab, baaaaab, \ldots\}$



- jos σ on merkkijono ja $n \in \mathbb{N}$, niin $\sigma^n = \overbrace{\sigma \cdots \sigma}^{n \text{ kpl}}$

$\Rightarrow$ mikään NFA ei hyväksy kieltä $\{a^n b^n \mid n \in \mathbb{N}\}$
- ne merkkijonot, joissa on ensin jokin määrä a:ta ja sitten sama määrä b:tä
- oletetaan, että N hyväksyy ainakin ne merkkijonot

$\Rightarrow$ se hyväksyy merkkijonon $a^{|Q|}b^{|Q|}$
- osuutta $a^{|Q|}$ lukiessa täytyy kiertää jokin silmukka $q =a^j\Rightarrow q$, missä $j > 0$
- siis $\exists i : \hat{q} =a^i\Rightarrow q =a^j\Rightarrow q =a^k b^{|Q|}\Rightarrow q' \in F$, missä $k = |Q| - i - j$

$\Rightarrow$ N hyväksyy myös $a^{i+k}b^{|Q|} = a^{|Q|-j}b^{|Q|}$, joka ei kuulu ko. kieleen

$\Rightarrow$ N ei hyväksy kieltä $\{a^n b^n \mid n \in \mathbb{N}\}$

- edellisessä päättelyssä osoitetttin, että jos N hyväksyy tarpeeksi, niin se hyväksyy liikaa
- samoin voidaan monesta muusta kielestä osoittaa, että mikään NFA ei hyväksy sitä

Pumppauslemma

- yleistämme ym. päätelmän lauseeksi, joka tunnetaan nimellä pumppauslemma

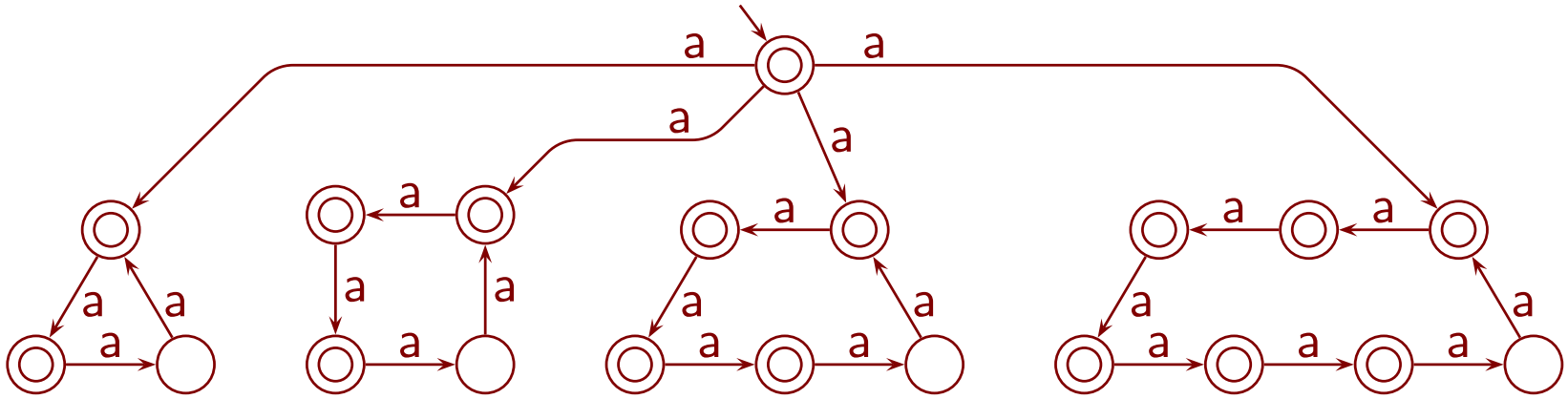
  Lause 2.9 Olkoon N NFA, jossa on n tilaa ja joka hyväksyy merkkijonon σ, jolle $|\sigma| \geq n$. On olemassa merkkijonot ρ, β ja γ siten, että $\sigma = \rho\beta\gamma$, $|\rho\beta| \leq n$, $|\beta| \geq 1$ ja N hyväksyy jokaisen merkkijonon muotoa $\rho\beta^i\gamma$, missä $i \in \mathbb{N}$.

- ρ tulee osuudesta ennen silmukkaa, β silmukasta ja γ osuudesta silmukan jälkeen

Jos $\mathcal{L}(N) \neq \emptyset$, niin lyhimmän N:n hyväksymän merkkijonon pituus on alle $|Q|$

- olkoon $\sigma \in \mathcal{L}(N)$ mahdollisimman lyhyt
- jos $|\sigma| \geq |Q|$, niin σ voidaan esittää muodossa $\sigma = \rho\beta\gamma$, missä $|\beta| \geq 1$ ja $\rho\gamma \in \mathcal{L}(N)$
  $\Rightarrow \mathcal{L}(N)$ sisältää lyhyemmän merkkijonon kuin σ

Mikä on lyhin merkkijono jonka oheinen NFA hylkää?



$\Rightarrow$ ei voida päätellä, että lyhin hylätty merkkijono on pituudeltaan alle $|Q|$

2.4 Saman kielen hyväksyvän DFA:n muodostaminen NFA:sta

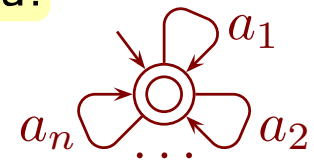
Lienee erittäin työlästä testata, hyväksyykö kaksi annettua NFA:ta saman kielen

- lienee erittäin työlästä testata, hylkääkö annettu NFA *yhtään* merkkijonoa!
- molemmat ovat **PSPACE**-täydellisiä tehtäviä (ks. luku **??**)

$\Rightarrow$ todennäköisesti työläämpiä kuin esim. kauppamatkustajan ongelma

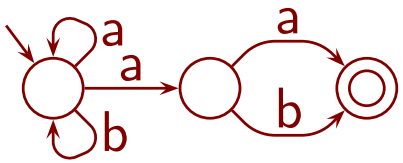
$\Rightarrow$ NFA:n muuntaminen mahdollisimman pieneksi lienee erittäin työlästä

- huom! tämä ei tarkoita työläyttä *kaikilla* syötteillä, vaan vain äärettömän useilla

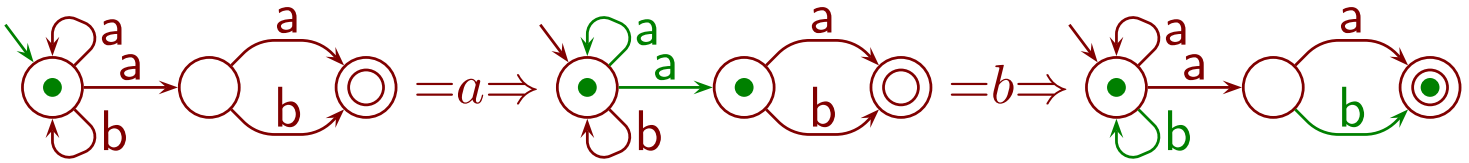


Merkkijonon lukeminen deterministisesti

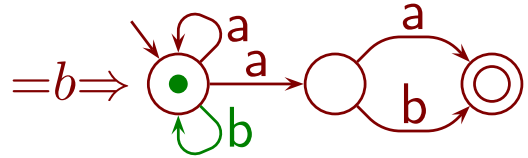
- tehtävän "päteekö $\sigma \in \mathcal{L}(N)$" voi ratkaista tehokkaasti pitämällä kirjaa kaikista tiloista, joissa N voi olla



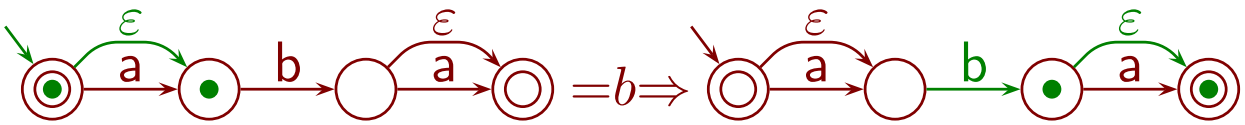
- esim. $ab \in \mathcal{L}(N)$



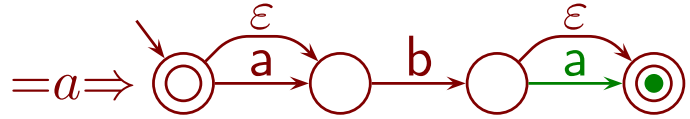
- esim. $abb \notin \mathcal{L}(N)$



- esim. $b \in \mathcal{L}(N)$

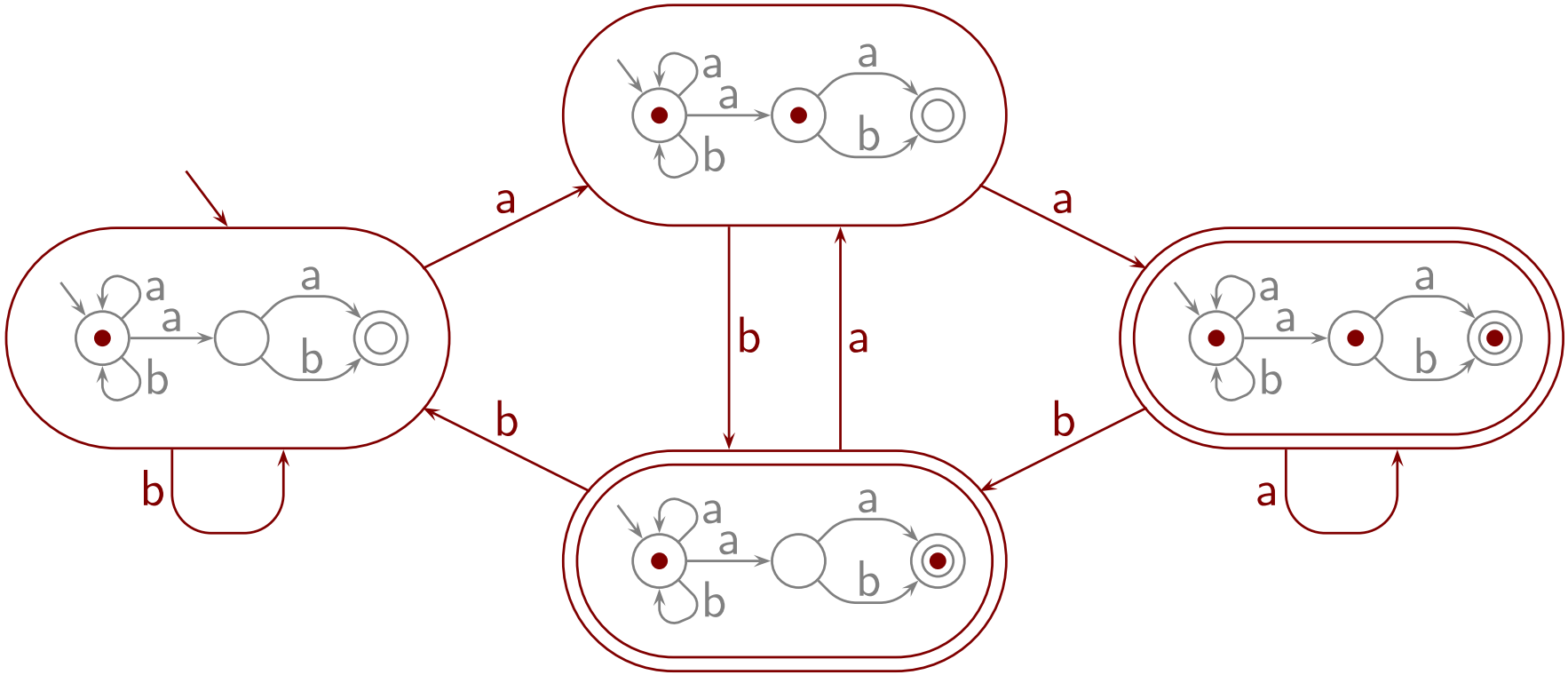


- esim. $ba \in \mathcal{L}(N)$



Muunnos NFA $\rightsquigarrow$ DFA

- tutkimalla samalla periaatteella aina kaikki merkit voidaan tehdä DFA, joka hyväksyy saman kielen kuin annettu NFA
- lopputulos voi olla iso!



- olkoon syöte $N = (Q_N, \Sigma, \Delta_N, \hat{q}_N, F_N)$ ja tulos $D = (Q_D, \Sigma, \delta_D, \hat{q}_D, F_D)$

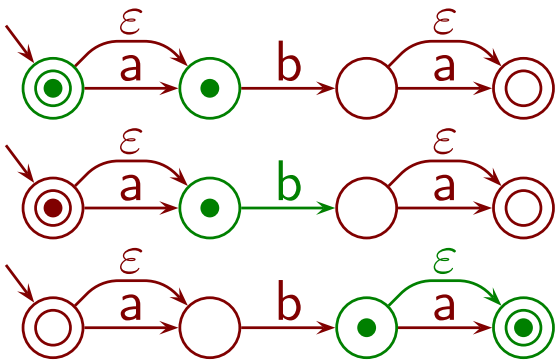
- algoritmi

  $\delta_D := \emptyset$; $\hat{q}_D := \{q_N \mid \hat{q}_N =\varepsilon\Rightarrow_N q_N\}$; $Q_D := \{\hat{q}_D\}$; $W := \{\hat{q}_D\}$
  if $\hat{q}_D \cap F_N \neq \emptyset$ **then** $F_D := \{\hat{q}_D\}$ **else** $F_D := \emptyset$
  while $W \neq \emptyset$ **do**
    valitse mikä tahansa $q_D \in W$; $W := W \setminus \{q_D\}$
    for $a \in \Sigma$ **do**
      $q'_D := \{q''_N \mid \exists q_N \in q_D : \exists q'_N \in Q_N : (q_N, a, q'_N) \in \Delta_N \wedge q'_N =\varepsilon\Rightarrow_N q''_N\}$
      $\delta(q_D, a) := q'_D$
      if $q'_D \notin Q_D$ **then**
        $Q_D := Q_D \cup \{q'_D\}$; $W := W \cup \{q'_D\}$
        if $q'_D \cap F_N \neq \emptyset$ **then** $F_D := F_D \cup \{q'_D\}$

- D:n tilat ovat N:n tilojen joukkoja, ts. $q_D \subseteq Q_N$
  $\Rightarrow$ niitä voi olla korkeintaan $2^{|Q_N|}$ kpl eli $|Q_D| \leq 2^{|Q_N|}$
- $\hat{q}_D$ on niiden N:n tilojen joukko, joihin pääsee N:n alkutilasta ε-kaaria pitkin
- W on niiden löydettyjen tilojen joukko, joista lähteviä kaaria ei ole vielä muodostettu
  - kun tila löytyy ensimmäisen kerran, se laitetaan sekä Q_D:hen että W:hen
  - jos sama tila löytyy uudelleen, sille ei tehdä mitään

  $\Rightarrow$ jokainen löydetty tila käy W:ssä täsmälleen yhden kerran
  $\Rightarrow$ **while**-silmukka kierretään enintään $2^{|Q_N|}$ kertaa
  $\Rightarrow$ algoritmi lopettaa

- **while**-silmukassa poistetaan yksi tila W:stä ja muodostetaan sen lähtökaaret tutkimalla jokainen aakkonen **for**-silmukassa
- tilan q_D ja aakkosen a käsittely
  - käsiteltävä tila q_D on joukko N:n tiloja
  - etsitään ne N:n tilat q_N, jotka kuuluvat q_D:hen ja joista lähtee ainakin yksi a-kaari
  - etsitään näiden kaarten lopputilat q'_N ja kaikki tilat q''_N joihin niistä pääsee ε-kaarilla
  - kootaan näin löytyneet tilat joukoksi q'_D
  - muodostetaan kaari $q_D \mathrel{=}a\Rightarrow_D q'_D$
  - jos q'_D on uusi, lisätään se joukkoihin Q_D ja W



- D:n tilasta tehdään lopputila, jos ja vain jos se sisältää ainakin yhden N:n tilan

Lopputulos kuvattuna matemaattisesti ilman muodostamistapaa

- merkitsemme $q_\sigma = \{q_N \mid \hat{q}_N \mathrel{=}\sigma\Rightarrow_N q_N\}$
  - niiden N:n tilojen joukko, joihin σ vie N:n alkutilasta
- kohta todistamme, että lopputuloksen osat ovat

$$\begin{aligned} Q_D &= \{q_\sigma \mid \sigma \in \Sigma^*\} \\ \delta_D(q_\sigma, a) &= q_{\sigma a}, \text{ kun } \sigma \in \Sigma^* \text{ ja } a \in \Sigma \\ \hat{q}_D &= q_\varepsilon \\ F_D &= \{q_\sigma \mid \sigma \in \mathcal{L}(N)\} \end{aligned}$$

Todistus, että algoritmin lopputulos on yllä mainittu

- $\hat{q}_D = q_\varepsilon$ seuraa suoraan määritelmistä
- jos $q''_N \in \delta_D(q_\sigma, a)$, niin
  - δ_D:n määritelmän mukaan $\exists q_N \in q_\sigma : q_N =a\Rightarrow_N q''_N$
  - q_σ:n määritelmän mukaan $\hat{q}_N =\sigma\Rightarrow_N q_N$

  $\Rightarrow \hat{q}_N =\sigma a\Rightarrow_N q''_N$

  $\Rightarrow q''_N \in q_{\sigma a}$
- jos $q''_N \in q_{\sigma a}$, niin
  - $q_{\sigma a}$:n määritelmän mukaan $\hat{q}_N =\sigma a\Rightarrow_N q''_N$
  - poimimalla tilat a-kaaren molemmin puolin saadaan

    $\exists q_N : \exists q'_N : \hat{q}_N =\sigma\Rightarrow_N q_N \wedge (q_N, a, q'_N) \in \Delta_N \wedge q'_N =\varepsilon\Rightarrow_N q''_N$
  - selvästi $q_N \in q_\sigma$

  $\Rightarrow \delta_D$:n määritelmän mukaan $q''_N \in \delta_D(q_\sigma, a)$

$\Rightarrow \delta_D(q_\sigma, a) = q_{\sigma a}$, kun $\sigma \in \Sigma^*$ ja $a \in \Sigma$

- induktiolla seuraa, että algoritmin muodostamat tilat ovat muotoa q_σ, missä $\sigma \in \Sigma^*$
- algoritmi muodostaa ne kaikki, koska se muodostaa $\delta_D(q_\sigma, a)$:n jokaiselle löytämälleen q_σ ja jokaiselle $a \in \Sigma$

  $\Rightarrow$ jos $\Sigma \neq \emptyset$, niin ainakin yksi q_D on q_σ äärettömän monelle eri σ:lle

$\Rightarrow Q_D = \{q_\sigma \mid \sigma \in \Sigma^*\}$

- $\sigma \in \mathcal{L}(N) \Leftrightarrow \exists q_N \in F_N : \hat{q}_N =\sigma\Rightarrow_N q_N \Leftrightarrow q_\sigma \cap F_N \neq \emptyset \Leftrightarrow q_\sigma \in F_D$

Induktiolla voidaan todistaa helposti, että $\forall \sigma \in \Sigma^* : \hat{q}_D =\sigma\Rightarrow_D q_\sigma$

- pohja
  - kaikkien DFA:ien kaikille tiloille pätee $q =\varepsilon\Rightarrow q$
  - edellä osoitettiin, että $\hat{q}_D = q_\varepsilon$

  $\Rightarrow \hat{q}_D =\varepsilon\Rightarrow_D q_\varepsilon$
- induktioaskel
  - induktio-oletus: $\hat{q}_D =\sigma\Rightarrow_D q_\sigma$
  - edellä osoitettiin, että $\delta_D(q_\sigma, a) = q_{\sigma a}$

  $\Rightarrow \hat{q}_D =\sigma\Rightarrow_D q_\sigma =a\Rightarrow_D q_{\sigma a}$

  $\Rightarrow \hat{q}_D =\sigma a\Rightarrow_D q_{\sigma a}$

Todistus, että kieli säilyy

- $\sigma \in \mathcal{L}(D) \Leftrightarrow \exists q_D \in F_D : \hat{q}_D =\sigma\Rightarrow_D q_D \Leftrightarrow q_\sigma \in F_D \Leftrightarrow \sigma \in \mathcal{L}(N)$

Olemme todistaneet seuraavan:

Lause 2.10 Jokaiselle NFA:lle N on olemassa DFA D siten, että $\mathcal{L}(D) = \mathcal{L}(N)$, ja päinvastoin.

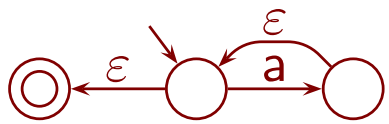
Pikavertailu lohkomisalgoritmiin

- myös lohkomisalgoritmissa lopputuloksen tilat ovat syötteen tilojen joukkoja
- siellä jokainen syötteen tila kuuluu *täsmälleen yhteen* lopputuloksen tilaan
- muunnoksessa NFA $\rightsquigarrow$ DFA syötteen tila voi kuulua useaan, jopa kaikkiin

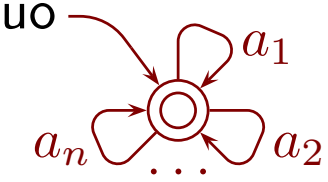
Suoritusajasta

- yhden D:n tilan muodostaminen on mahdollista saada ainakin kohtuullisen nopeaksi
  - Q_D kannattaa toteuttaa esim. hajautustauluna

  $\Rightarrow$ testi $q'_D \notin Q_D$ saadaan nopeaksi
  - $\{q''_N \mid q'_N =\varepsilon\Rightarrow_N q''_N\}$ voidaan laskea ajassa $O(|\Delta_N|)$, usein paljon nopeamminkin
  - F_D voidaan toteuttaa yhtenä bittinä tilaoliossa
  - W voi olla esim. taulukko osoittimia
  - kaarten (q_N, a, q'_N) löytymistä voi nopeuttaa esittämällä N sopivasti
- ongelmana on syntyvien D:n tilojen mahdollisesti suuri määrä!
  - yläraja $2^{|Q_N|}$
- jonkin verran voi tehostaa ottamalla D:n tiloihin vain ne N:n tilat, joista lähtee ei-ε-kaari
  - $\hat{q}_D$ ja F_D täytyy valita oikein



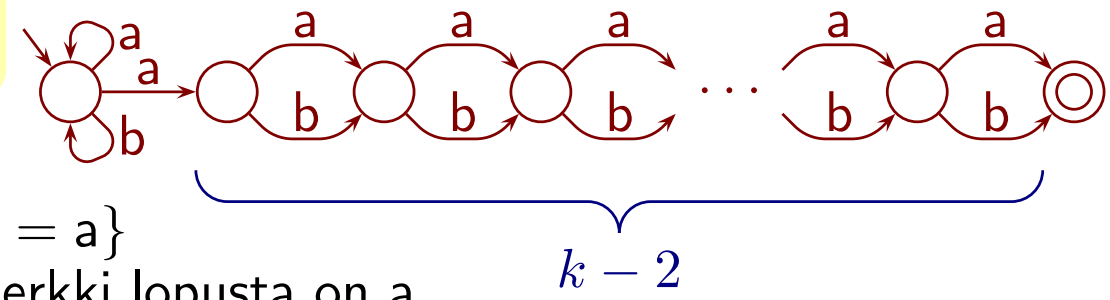
- jonkin verran voi tehostaa jättämällä pois tilan $\emptyset \subseteq Q_N$
- jos D:n muodostaminen olisi aina nopeaa, "onko $\mathcal{L}(N) = \Sigma^*$" voitaisiin ratkaista aina nopeasti muodostamalla D, minimoimalla se, ja katsomalla tuliko tuo



$\Rightarrow$ koska $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ ei voitane ratkaista aina nopeasti, D:n muodostaminen ei voine olla aina nopeaa

- ei tiedetä täysin varmasti, voidaanko $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ ratkaista aina nopeasti
- seuraavaksi näytämme, että on varmaa, että D:tä ei voi muodostaa aina nopeasti!

Äärettömän monta NFA:ta, joille jokainen DFA on eksponentiaalisesti isompi



- oheisessa NFA:ssa N on k tilaa
- $\mathcal{L}(N) = \{a_1 \cdots a_n \in \{\mathsf{a},\mathsf{b}\}^* \mid a_{n-k+2} = \mathsf{a}\}$
  - ne merkkijonot, joiden $(k-1)$:s merkki lopusta on a
- vertaamme D:n tilaa sen luettua merkkijonot $a_1 \cdots a_{k-1}$ ja $b_1 \cdots b_{k-1}$
- jos $a_1 \neq b_1$, niin tilat ovat erit
  - a:lla alkava merkkijono hyväksytään ja b:llä alkavaa ei

  $\Rightarrow$ toinen tila on lopputila ja toinen ei ole
- jos $a_2 \neq b_2$, niin tilat ovat erit
  - jos luetaan vielä yksi merkki, niin ollaan edellisen kohdan $a_1 \neq b_1$ tilanteessa
- ...
- jos $a_{k-1} \neq b_{k-1}$, niin tilat ovat erit
  - jos luetaan vielä yksi merkki, niin ollaan edellisen kohdan $a_{k-2} \neq b_{k-2}$ tilanteessa
  - ts. jos luetaankin vielä $k-2$ merkkiä, niin toisessa tapauksessa päädytään lopputilaan ja toisessa ei

$\Rightarrow$ jokaisessa kielen $\mathcal{L}(N)$ hyväksyvässä DFA:ssa D on ainakin $2^{k-1} = \frac{1}{2} \cdot 2^{|Q_N|}$ tilaa

Siksi *ei ole olemassa algoritmia*, jolla muunnos NFA $\rightsquigarrow$ DFA onnistuu polynomiajassa

- kuvan NFA:ille aikaa kuluu *ainakin* $\Theta(2^{|Q_N|})$ *riippumatta algoritmista*

Miksi vastaesimerkin täytyy olla ääretön joukko esimerkkejä?

- tämä on niin tärkeä asia, että se otetaan uusiksi luvussa 4!
- algoritmin suoritusaika riippuu yleensä syötteen koosta
  - mitä pitempi oja pitää kaivaa, sitä kauemmin aikaa menee
  - poikkeuksiakin on, esim. 10-järjestelmässä esitetyn luvun parillisuuden testaaminen
- aika voi vaihdella saman kokoisillakin syötteillä
  - ojan kaivamista hidastaa, jos maassa on paljon isoja kiviä
  - on paljon hitaampaa laskea $1414213562 \cdot 7$ kuin $1414213562 \cdot 0$
- jos syötteen koolla on yläraja, niin on olemassa aina nopea algoritmi
  - luetellaan valmiit vastaukset jollain nopeasti käytettävällä tavalla
  - esim. `bool on_alkuluku[] = {0,0,1,1,0,1,0,1,0,0,0,1,0,1,0,0,0,1};`
  - esim. kyllä / ei -kysymyksille deterministinen äärellinen automaatti
  - monimutkaisempia vastauksia voidaan saada liittämällä ne lopputiloihin
  - tämä algoritmi on joskus käytännöllinen
  - hyvin usein se on luettelon koon vuoksi täysin epärealistinen
- tämä ratkaisu voi tuntua huijaukselta!
  - se ei itse laske mitään vaan ainoastaan palauttaa sille valmiina annetun vastauksen
  - matemaattisesti se on kuitenkin ok niin kauan kuin algoritmi määritellään vain keinona tuottaa oikea vastaus
  - käytännössäkin on olemassa tällaisia algoritmeja, tai tällaista on isompien algoritmien osana

⇒ jos haluamme estää tämän ratkaisun, on meidän esitettävä jokin lisävaatimus
  - jos teorian tulokset eivät toimi käytännössä, niin vikana voi olla, että teoriassa ei ole otettu huomioon kaikkia olennaisia lähtökohtia
  - jos esim. lääkkeiden jakelun tietojärjestelmä ei toimi käytännössä (kuuklaa Q3 2017 - Oriola), niin ...

- vaatimus, että syötteitä voi olla äärettömän monta erilaista, hoitaa homman
  - ohjelman on oltava äärellinen
  - ⇒ tämä vaatimus estää kaikkien vastausten luetteloimisen valmiina
  - tässä hyvin todellisessa mielessä algoritmi tuottaa enemmän informaatiota kuin sen suunnittelija on siihen laittanut, vieläpä äärettömän paljon enemmän!
- siis oletamme, että äärellinen on mahdollista mutta ääretön ei ole
  - *mistä tahansa* kokonaisluvusta voidaan äärellisessä ajassa tutkia, onko se alkuluku
  - *kaikkia* kokonaislukuja ei voida tutkia, koska niitä on äärettömästi
  - ⇒ äärettömyys toteutuu äärettömänä joukkona mahdollisuuksia, joista voidaan toteuttaa mikä tahansa äärellinen osa (onpa löysäpäisen oloinen virke, älä pistä minun nimiini irrallaan asiayhteydestä!)
- jos algoritmi on hidas vain äärellisellä määrällä syötteitä, se voidaan korjata muuttamalla algoritmi hakemaan niille valmiit vastaukset luettelosta

⇒ jos halutaan osoittaa, että aina nopeaa algoritmia ei ole, niin
  - tarvitaan ääretön määrä vastaesimerkkejä
  - enempää ei tarvita

Lause 2.11 Syötteiden joukon osajoukko on ääretön, jos ja vain jos siinä on mielivaltaisen isoja syötteitä.

- "siinä on mielivaltaisen isoja syötteitä" tarkoittaa: valitaan mikä tahansa luonnollinen luku n, joukossa on syöte, jonka koko on vähintään n
  - $\forall n \in \mathbb{N} : \exists s \in$ Syötteet $: |s| \geq n$
- olkoon S puheena oleva syötteiden osajoukko
- suunnan $\Leftarrow$ todistus
  - olkoon $s_1 \in S$ jokin syöte ko. osajoukosta, jonka koko on vähintään 0
  - jokaiselle $i > 1$ olkoon $s_{i+1} \in S$ siten, että $|s_{i+1}| \geq |s_i| + 1$
  - s_1 ja s_{i+1} ovat olemassa, koska S:ssä on mielivaltaisen isoja alkioita
  - jos $i < j$ niin $s_i \neq s_j$, koska $|s_i| \neq |s_j|$, koska $|s_i| < |s_{i+1}| < \cdots < |s_j|$

  $\Rightarrow$ saatiin äärettön jono $s_1, s_2, \ldots$ keskenään erilaisia syötteitä
- suunnan $\Rightarrow$ todistus
  - syötteitä kooltaan enintään n on olemassa enintään $1 + |\Sigma| + |\Sigma|^2 + \ldots + |\Sigma|^n$ kpl
  - niistä S:ään kuuluu korkeintaan sama määrä
  - ne eivät riitä äärettömäksi määräksi syötteitä

  $\Rightarrow$ mille tahansa n on oltava syöte, joka on isompi kuin n
  - tuntuuko, että jotain tällaista on nähty ennenkin?

2.5 Tehtävän $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ muistin ja ajan kulutus

Olemme osoittaneet, että aina nopeaa algoritmia NFA $\rightsquigarrow$ DFA ei ole olemassa

- silti ei ole varmaa vaan ainoastaan todennäköistä, että $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ ei voida ratkaista aina nopeasti
- on varmaa, että NFA $\rightsquigarrow$ DFA $\rightsquigarrow$ minimoitu DFA $\stackrel{?}{=}$ 

ei ole aina nopea
- on varmaa, että mikään keino muotoa NFA $\rightsquigarrow$ DFA $\rightsquigarrow$ ... ei ole aina nopea eikä muistin käytöltään aina tehokas
  - DFA voi olla iso

  $\Rightarrow$ vie paljon muistia ja on hidas muodostaa
- silti keinoa NFA $\rightsquigarrow$ DFA käytetään paljon, uskon että ylivoimaisesti eniten
- kuitenkin muitakin ratkaisukeinoja saattaa olla olemassa
- sovimme, että "vähän muistia" tarkoittaa polynomimuistissa
  - muistissa $O(n)$ tai $O(n^2)$ tai $O(n^3)$ tai ...
  - ei salli esim. $O(2^n)$, $O(1{,}0001^n)$, eikä $2^{O(\log^2 n)}$ eli $n^{O(\log n)}$
- äänestys, mitä arvaatte, voidaanko $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ ratkaista aina vähällä muistilla?

  (a) tiedetään varmasti, että ei voida
  (b) on todennäköistä, että ei voida, mutta ei tiedetä varmasti
  (c) kysymys on täysin auki
  (d) on todennäköistä, että voidaan, mutta ei tiedetä varmasti
  (e) tiedetään varmasti, että voidaan

Mahdollisimman lyhyen hylätyn merkkijonon pituus NFA:n tapauksessa

- kuten ennenkin, olkoon $q_\sigma = \{q_N \mid \hat{q}_N =\sigma\Rightarrow_N q_N\}$
- $a_1 \cdots a_n \notin \mathcal{L}(N)$ jos ja vain jos $q_{a_1\cdots a_n} \cap F_N = \emptyset$
- muistanemme, että q_σ voi saada enintään $2^{|Q_N|}$ eri arvoa

$\Rightarrow$ jos $n \geq 2^{|Q_N|}$, niin on olemassa i ja j siten, että $0 \leq i < j \leq n$ ja $q_{a_1\cdots a_i} = q_{a_1\cdots a_j}$
  – $q_{a_1\cdots a_i a_{j+1}\cdots a_n} = q_{a_1\cdots a_n}$ ja $a_1 \cdots a_i a_{j+1} \cdots a_n \in \mathcal{L}(N) \Leftrightarrow a_1 \cdots a_n \in \mathcal{L}(N)$

$\Rightarrow$ jos $\sigma \notin \mathcal{L}(N)$ ja $|\sigma| \geq 2^{|Q_N|}$, niin on olemassa ρ siten, että $\rho \notin \mathcal{L}(N)$ ja $|\rho| < |\sigma|$

$\Rightarrow$ joko $\mathcal{L}(N) = \Sigma^*$ tai lyhin hylätty merkkijono on pituudeltaan alle $2^{|Q_N|}$

Siksi $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ ratkeaa tutkimalla kaikki merkkijonot pituudeltaan enintään $2^{|Q_N|}$

- teemme sen tutkimalla vastaavan DFA:n D polkuja muistia säästävällä tavalla

```
alku := q_ε
for maali ⊆ Q_N \ F_N do
  if on_polku( alku, maali, |Q_N| ) then return false
return true
```

- joukkoa maali voi kasvattaa kuten auton matkamittaria
- on_polku(lähtö, maali, k) kertoo, onko olemassa polkua N:n tilajoukosta eli D:n tilasta lähtö N:n tilajoukkoon eli D:n tilaan maali, jolla luetaan enintään 2^k merkkiä
  – lähtö $\subseteq Q_N$, maali $\subseteq Q_N$ ja $k \in \mathbb{N}$
  – kertoo, päteekö $\exists\sigma : |\sigma| \leq 2^k \wedge \text{maali} = \{q'_N \mid \exists q_N \in \text{lähtö} : q_N =\sigma\Rightarrow_N q'_N\}$

```
on_polku( lähtö, maali, k )
if k = 0 then
  if maali = {q'_N | ∃q_N ∈ lähtö : q_N =ε⇒_N q'_N} then return true
  for a ∈ Σ do
    if maali = {q'_N | ∃q_N ∈ lähtö : q_N =a⇒_N q'_N} then return true
  return false
for väli ⊆ Q_N do
  if on_polku(lähtö, väli, k − 1) ∧ on_polku(väli, maali, k − 1) then return true
return false
```

- jos $k = 0$, niin tutkitaan nollan ja yhden merkin mittaiset polut, koska $2^0 = 1$
- kun $k > 1$, niin tutkitaan kaikki mahdollisuudet muodostaa polku kahdesta osasta, joiden pituus on enintään 2^{k-1}
- tämä tekniikka on peräisin Savitch'n lauseen todistuksesta

Paljonko muistia kuluu?

- pääohjelmassa paljon muistia kuluttaa vain kaksi Q_N:n osajoukkoa $\Rightarrow 2|Q_N|$ bittiä
  - k:lle eli $|Q_N|$:lle riittää $\lceil \log_2 |Q_N| \rceil$ bittiä
  - lisäksi tarvitaan muutama muuttuja laskemaan σ_ε, toteuttamaan **for**-silmukka jne.
  - kullekin niistä riittää $\lceil \log_2 |Q_N| \rceil$ bittiä

- analyysimme epämääräisyys tarvittavien osoittimien, lukujen yms. määrän osalta ei vääristä tulosta, koska
  - yhden sellaisen muistin kulutus kasvaa $|Q_N|$:n funktiona paljon hitaammin kuin tilajoukon muistin kulutus
  - niiden määrä pääohjelmassa ja kullakin rekursiotasolla ei riipu N:stä
- jos tämä laskettaisiin algoritmiikasta tutulla tavalla, niin
  - joukkojen oletettaisiin vievän tilaa $\Theta(|Q_N|)$ bittiä (kuten nytkin oletetaan)
  - muiden muuttujien oletettaisiin (toisin kuin nyt) vievän vakion verran muistia
  - tulos olisi pätevä käytännön toteutukselle oikealla ohjelmointikielellä
  - teoreettisessa tietojenkäsittelytieteessä on kuitenkin tapana laskea toisin, koska siellä ei oleteta, että yksinkertaiset muuttujat ovat esim. 32- tai 64-bittisiä
- rekursion välitasolla tarvitaan joukko "väli" sekä muutamia osoittimia yms.
  - "lähtö" ja "maali" löytyvät osoittimien avulla
  - joukolle "väli" riittää $|Q_N|$ bittiä
  - kullekin osoittimelle, luvulle $k-1$ yms. riittää $\lceil \log_2 |Q_N| \rceil$ bittiä
  - kääntäjä tarvitsee rekursion toteuttamiseen jotain kertaa $\lceil \log_2 |Q_N| \rceil$ bittiä

  $\Rightarrow$ on olemassa vakio c_1 siten, että muulle kuin "väli" riittää $c_1 \lceil \log_2 |Q_N| \rceil$ bittiä
- rekursion pohjatasolla riittää
  - $|Q_N|$ bittiä joukkojen $\{q'_N \mid \ldots\}$ laskemista varten
  - muutama osoitin ja luku, kukin $\lceil \log_2 |Q_N| \rceil$ bittiä
- merkitsemme pääohjelmassa ja pohjatasolla yhteensä tarvittavien muiden muuttujien kuin joukkojen määrää c_2:lla

- rekursion välitasoja on $|Q_N|$ kpl

$\Rightarrow$ muistia tarvitaan kaikkiaan $|Q_N| \cdot (|Q_N| + c_1 \lceil \log_2 |Q_N| \rceil) + 3|Q_N| + c_2 \lceil \log_2 |Q_N| \rceil$
$= |Q_N|^2 + c_1 |Q_N| \lceil \log_2 |Q_N| \rceil + 3|Q_N| + c_2 \lceil \log_2 |Q_N| \rceil = \Theta(|Q_N|^2)$ bittiä

$\Rightarrow$ Onko $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$ voidaan ratkaista vähällä muistilla

$\Rightarrow$ edellä oikea vastaus on (e)

- esim. jos N:ssä on 1 000 tilaa, vähän yli 1 000 000 bittiä riittää
  - nykykoneiden muistille pikkujuttu

Paljonko aikaa kuluu?

- ym. algoritmi pilkkoo ym. 1 000 000 bittiä tuhanneksi 1000-bittiseksi laskuriksi, jotka se pyörittää toistuvasti ympäri
- kokonaisuus käyttäytyy kuten 1 000 000-bittinen laskuri, joka pyöritetään ympäri

$\Rightarrow$ ainakin $2^{1\,000\,000} \approx 10^{301\,030}$ toimenpidettä

- nykykoneet tekevät suuruusluokkaa 10^9 asiaa sekunnissa
- on äärimmäisen epärealistista olettaa, että päästäisiin suuruusluokkaan 10^{44}
  - kuuklaa Planckin aika
- maailmankaikkeuden arvioitu ikä on noin 10^{17} sekuntia

$\Rightarrow$ yhdessä maailmankaikkeuden iässä ehditään enintään noin 10^{61} toimenpidettä

$\Rightarrow$ sillä vauhdilla tarvitaan $\approx 10^{301\,030-61} = 10^{300\,969}$ maailmankaikkeuden ikää

- onnistuisiko rinnakkaistamalla?
  - hankitaan $10^{300\,000}$-ytiminen prosessori?
  - näkyvässä maailmankaikkeudessa arvioidaan olevan $\approx 10^{80}$ elektronia

  $\Rightarrow$ nanoteknologian täytyy kehittyä vielä aika paljon, ennen kuin onnistuu

Ym. algoritmia voi optimoida paljonkin

- esim. ei kannata tutkia joukkoa "väli", joka on jo syvemmällä rekursiopinossa
  - jos on olemassa polku lähdöstä maaliin, niin on olemassa silmukaton polku
- kenties onnistumme nopeuttamaan kertoimella $10^{100\,000}$?
  - huikea parannus, televisiokamerat paikalle ja juhlajuomat esiin!
  - sen jälkeen $10^{200\,969}$ maailmankaikkeuden ikää riittää ☹
- algoritmin tolkuton hitaus johtuu siitä, että se laskee samoja asioita toistuvasti

$\Rightarrow$ voidaan nopeuttaa hurjasti pitämällä kirjaa jo lasketuista vastauksista

bool laskettu$[\,2^{|Q_N|}, 2^{|Q_N|}, |Q_N|\,]$:= false, vastaus$[\,2^{|Q_N|}, 2^{|Q_N|}, |Q_N|\,]$
on_polku(lähtö, maali, k)
if $\neg$ laskettu$[\,$lähtö, maali, $k\,]$ **then**
  ... lasketaan vastaus kuten edellä ...
  vastaus$[\,$lähtö, maali, $k\,]$:= laskettu vastaus; laskettu$[\,$lähtö, maali, $k\,]$:= true
return vastaus$[\,$lähtö, maali, $k\,]$

- tätä voi vielä parantaa seuraavasti

  bool vastaus$[\,2^{|Q_N|}, 2^{|Q_N|}\,]$
  for lähtö $\subseteq Q_N$ **do for** maali $\subseteq Q_N$ **do** vastaus[lähtö, maali] := false
  for lähtö $\subseteq Q_N$ **do** vastaus[lähtö, lähtö] := true
  for lähtö $\subseteq Q_N$ **do**
    for $a \in \Sigma$ **do**
      maali := $\{q'_N \mid \exists q_N \in$ lähtö : $q_N =a\Rightarrow_N q'_N\}$
      vastaus[lähtö, maali] := true
  for väli $\subseteq Q_N$ **do**
    for lähtö $\subseteq Q_N$ **do**
      for maali $\subseteq Q_N$ **do**
        if vastaus[lähtö, väli] $\wedge$ vastaus[väli, maali] **then**
          vastaus[lähtö, maali] := true

  - kun uloin **for**-silmukka käsittelee tilajoukkoa väli, on jo löydetty kaikki polut, joissa väli ja sen jälkeen tulevat esiintyvät korkeintaan lähtönä ja maalina
  - ym. algoritmi on täysin käytännöllinen, vaikka sovelluksemme sille ei olekaan
  - lisätietoa: kuuklaa Floyd–Warshall

- mutta nyt muistin kulutus on eksponentiaalinen!
  - $2^{2|Q_N|}$ bittiä + jotain $\cdot\ |Q_N|$ bittiä + jotain $\approx 2^{|Q_N|} \cdot 2^{|Q_N|}$ bittiä

- meillä on nyt kolme algoritmia ratkaisemaan $\mathcal{L}(N) \stackrel{?}{=} \Sigma^*$

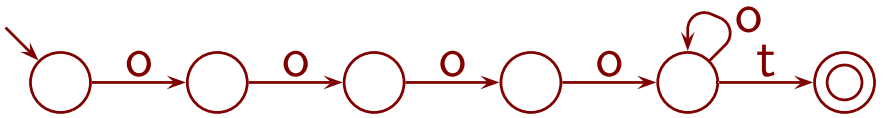
algoritmi	muistin kulutus	ajan kulutus
Savitch	aina melko pieni	aina tolkuttoman tolkuton
Floyd–Warshall	aina tolkuton	aina tolkuton
NFA $\rightsquigarrow$ DFA	pienestä tolkuttomaan	pienestä tolkuttomaan

  - "melko pieni" = $\Theta(|Q_N|^2)$
  - "tolkuton" = $\Theta(2^{2|Q_N|})$ tai pahempi
  - "tolkuttoman tolkuton" emme laske tarkasti, mutta paljon pahempi kuin tolkuton
  - "pieni" = esim. $\Theta(|\Sigma||\Delta_N|)$, jos kaikista tiloista pääsee kaikkiin ε-poluilla

$\Rightarrow$ muunnosta NFA $\rightsquigarrow$ DFA käytetään, koska se voi onnistua edes joskus!

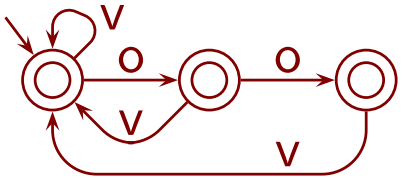
  - käytännössä esiintyvillä NFA:illa se onnistuu aika usein
  - parempaakaan keinoa ei tunneta

2.6 NFA:na esitettyjen kielten joukko-opillisia operaatioita



Johdanto

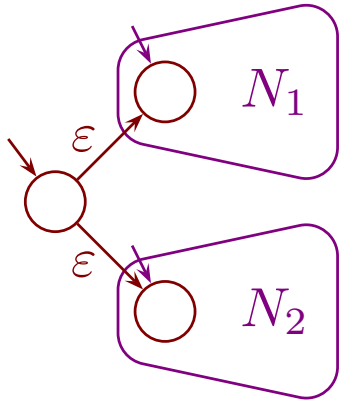
- usein sovelluksissa tarvitaan DFA tai NFA, jonka hyväksymä kieli on esim. syötteenä annettujen DFA:iden hyväksymien kielten leikkaus
- esimerkki yleistetystä kielten leikkauksesta
  - N_1 kertoo, miten pitää opiskella, että pääsee tentistä läpi
  - N_2 kertoo, miten pitää viettää vapaa-aikaa että pää ei räjähdä
  - $N_1 \,||\, N_2$ kertoo, miten molemmat ehdot toteutuvat



- teoreetikoita kiinnostaa, mitä voidaan ja ei voida tehdä nopeasti
- muilla laskennan malleilla joitakin asioita ei voida tehdä lainkaan

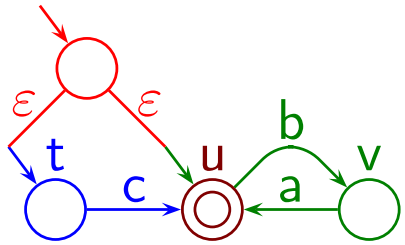
NFA N siten, että $\mathcal{L}(N) = \mathcal{L}(N_1) \cup \mathcal{L}(N_2)$, voidaan muodostaa seuraavasti

- olkoot N_1 ja N_2 NFA:ita
- varmistetaan, että $Q_1 \cap Q_2 = \emptyset$
  - tarvittaessa nimetään tiloja uudelleen
  - piirroksissa riittää, että N_1 ja N_2 ovat erillään
- valitaan $\Sigma = \Sigma_1 \cup \Sigma_2$
- luodaan uusi alkutila $\hat{q} \notin Q_1 \cup Q_2$
- asetetaan $Q = \{\hat{q}\} \cup Q_1 \cup Q_2$ ja $F = F_1 \cup F_2$



- lisätään kaaret $(\hat{q}, \varepsilon, \hat{q}_1)$ ja $(\hat{q}, \varepsilon, \hat{q}_2)$
  - ts. $\Delta = \{(\hat{q}, \varepsilon, \hat{q}_1), (\hat{q}, \varepsilon, \hat{q}_2)\} \cup \Delta_1 \cup \Delta_2$
- helppoa ja nopeaa!
- miksi täytyy varmistaa, että $Q_1 \cap Q_2 = \emptyset$?

$Q_1 = \{\mathsf{t}, \mathsf{u}\}$
$Q_2 = \{\mathsf{u}, \mathsf{v}\}$



NFA:n kielen komplementointi

- NFA $\overline{N}$ siten, että $\mathcal{L}(\overline{N}) = \Sigma^* \setminus \mathcal{L}(N)$, voidaan muodostaa muuntamalla N deterministiseksi ja komplementoimalla lopputulos, kuten edellä on kerrottu
- koska syntyvä DFA voi olla suuri, tämä tapa ei ole aina nopea
- laskuharjoituksissa osoitetaan, että aina nopeaa tapaa ei ole olemassa

Kahden NFA:n kielten leikkaus unionin avulla

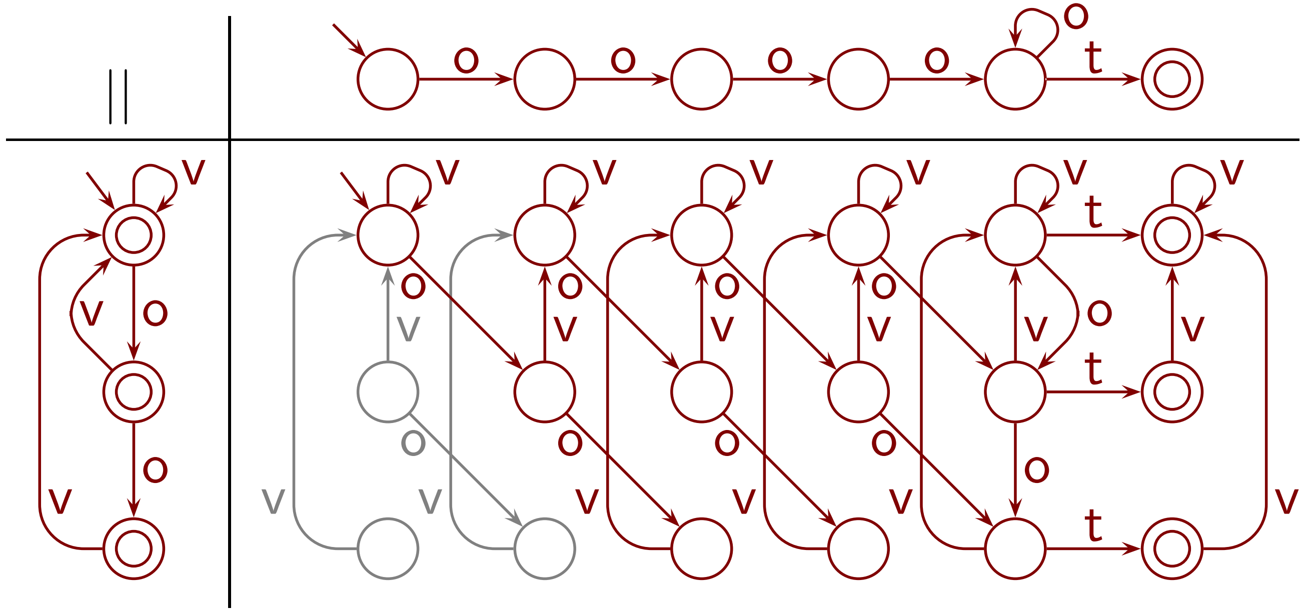
- Olkoon $N_1 \cup N_2$ edellä kuvattu NFA, jolle $\mathcal{L}(N_1 \cup N_2) = \mathcal{L}(N_1) \cup \mathcal{L}(N_2)$
- joukko-opin de Morganin kaavan vuoksi $\mathcal{L}(\overline{\overline{N}_1 \cup \overline{N}_2}) = \mathcal{L}(N_1) \cap \mathcal{L}(N_2)$
- tämä tapa voi olla kovin hidas, koska $\overline{N}_1$ voi olla kovin iso ja $\overline{N}_2$ myös

Tuloautomaatti

- aina melko tehokas keino muodostaa NFA siten, että tuloksen kieli on lähtökohtien kielten leikkaus
- jos lähtökohdat ovat deterministisiä, niin tuloskin on

- erityisen tärkeä rinnakkaisuuden (*concurrency*) tutkimuksessa
- jos $\Sigma_1 \neq \Sigma_2$, niin laskee muuta kuin leikkauksen
  - leikkaus tulee laajentamalla ensin kummankin syöte-NFA:n aakkostoksi $\Sigma_1 \cup \Sigma_2$
  - ilman laajennusta laskettu operaatio on yleensä hyödyllisempi kuin leikkaus
- ensin määritellään välitulos, jonka muut osat kuin Δ ovat
  - $Q = Q_1 \times Q_2$
  - jotta kaavoja olisi helpompi lukea, käytämme Q:n alkioille merkintää $\langle q_1, q_2 \rangle$ eikä (q_1, q_2)
  - $\Sigma = \Sigma_1 \cup \Sigma_2$
  - $F = F_1 \times F_2$
  - $\hat{q} = \langle \hat{q}_1, \hat{q}_2 \rangle$
- $(\langle q_1, q_2 \rangle, a, \langle q_1', q_2' \rangle) \in \Delta$ missä $\langle q_1, q_2 \rangle \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ ja $\langle q_1', q_2' \rangle \in Q$ jos ja vain jos
  - $a \in \Sigma_1 \cap \Sigma_2$, $(q_1, a, q_1') \in \Delta_1$ ja $(q_2, a, q_2') \in \Delta_2$,
  - $a \notin \Sigma_2$, $(q_1, a, q_1') \in \Delta_1$ ja $q_2' = q_2$ tai
  - $a \notin \Sigma_1$, $(q_2, a, q_2') \in \Delta_2$ ja $q_1' = q_1$
- tämä voidaan ajatella siten, että syöte-NFA:t lukevat samaa merkkijonoa yhtäaikaa
  - jos a kuuluu molempien aakkostoon, molempien on siirryttävä yhtäaikaa
  - jos a kuuluu vain toiseen aakkostoon, se NFA siirtyy ja toinen ei
  - kumpikin syöte-NFA suorittaa ε-siirtymänsä yksin
- tuloautomaatti $N_1 \,||\, N_2$ saadaan poistamalla välituloksesta turhat tilat ja kaaret

- esimerkki



  - mikä iso vika tässä esimerkissä on opettajan, mutta ei ehkä opiskelijan, mielestä?
  - mitkä kolme tilaa voi sulauttaa yhdeksi?
- usein välituloksen tiloista ja kaarista suurin osa on turhia
  - käytännössä yleensä tuloautomaattia muodostetaan ”tarpeen mukaan”, jotta turhia tiloja ja kaaria ei muodostettaisi
  - vrt. muunnos NFA $\rightsquigarrow$ DFA

2.7 Säännölliset lausekkeet

Säännölliset lausekkeet (*regular expression*) aakkostolle Σ, joka ei sisällä $\emptyset$, ε, $|$, *, (eikä), määritellään seuraavasti:

- $\emptyset$ on säännöllinen lauseke
- ε on säännöllinen lauseke
- jos $a \in \Sigma$, niin a on säännöllinen lauseke
- jos r ja s ovat säännöllisiä lausekkeita, niin myös $r \mid s$ on
- jos r ja s ovat säännöllisiä lausekkeita, niin myös rs on
- jos r on säännöllinen lauseke, niin myös r^* on

jos halutaan esim. ”$*$” $\in \Sigma$, niin korvataan ”$*$” metamerkeissä jollain muulla merkillä

Säännöllisen lausekkeen tuottama kieli määritellään ym. vaihtoehtojen mukaan:

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\mathcal{L}(a) = \{a\}$
- $\mathcal{L}(r \mid s) = \mathcal{L}(r) \cup \mathcal{L}(s)$
- $\mathcal{L}(rs) = \{\sigma\rho \mid \sigma \in \mathcal{L}(r) \wedge \rho \in \mathcal{L}(s)\}$
- $\mathcal{L}(r^*) = \{\varepsilon\} \cup \mathcal{L}(r) \cup \mathcal{L}(rr) \cup \mathcal{L}(rrr) \cup \ldots$

Siis

- $\mathcal{L}(r \mid s)$ sisältää kaikki $\mathcal{L}(r)$:n merkkijonot ja kaikki $\mathcal{L}(s)$:n merkkijonot, eikä muuta
- $\mathcal{L}(rs)$ sisältää täsmälleen ne merkkijonot, jotka saadaan ottamalla mikä tahansa merkkijono $\mathcal{L}(r)$:stä ja laittamalla sen perään mikä tahansa merkkijono $\mathcal{L}(s)$:stä
- $\mathcal{L}(r^*)$ sisältää täsmälleen ne merkkijonot, jotka saadaan valitsemalla nolla tai useampia merkkijonoja $\mathcal{L}(r)$:stä ja laittamalla ne peräkkäin
  - jokaisen valinnan saa tehdä vapaasti muista valinnoista riippumatta

Presedenssit ja liitännäisyys

- jotta ei tarvittaisi paljon sulkuja, sovimme että
  - * sitoo voimakkaimmin
  - peräkkäisyys sitoo toiseksi voimakkaimmin
  - $\mid$ sitoo heikoimmin
- esim. $ab^* \mid b^*a = (a(b^*)) \mid ((b^*)a)$
- koska $\mathcal{L}((rs)t) = \mathcal{L}(r(st))$, merkitään yleensä $\mathcal{L}(rst)$
- koska $\mathcal{L}((r \mid s) \mid t) = \mathcal{L}(r \mid (s \mid t))$, merkitään yleensä $\mathcal{L}(r \mid s \mid t)$

Esimerkkejä

- $\mathcal{L}(a \mid aa \mid ba) = \{a, aa, ba\}$
- $\mathcal{L}((a \mid aa \mid ba)(ab \mid b)) = \{aab, aaab, baab, ab, aab, bab\} = \{aaab, aab, ab, baab, bab\}$
- $\mathcal{L}((ab)^*) = \{\varepsilon, ab, abab, ababab, \ldots\}$

- $\mathcal{L}(ab^*) = \{a, ab, abb, abbb, \ldots\}$
- $\mathcal{L}((a \mid b)^*) = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$
- $\mathcal{L}(a^* \mid b^*) = \{\varepsilon, a, b, aa, bb, aaa, bbb, \ldots\}$

Joitakin melko tylsiä, mutta silti toisinaan tarpeellisia kaavoja

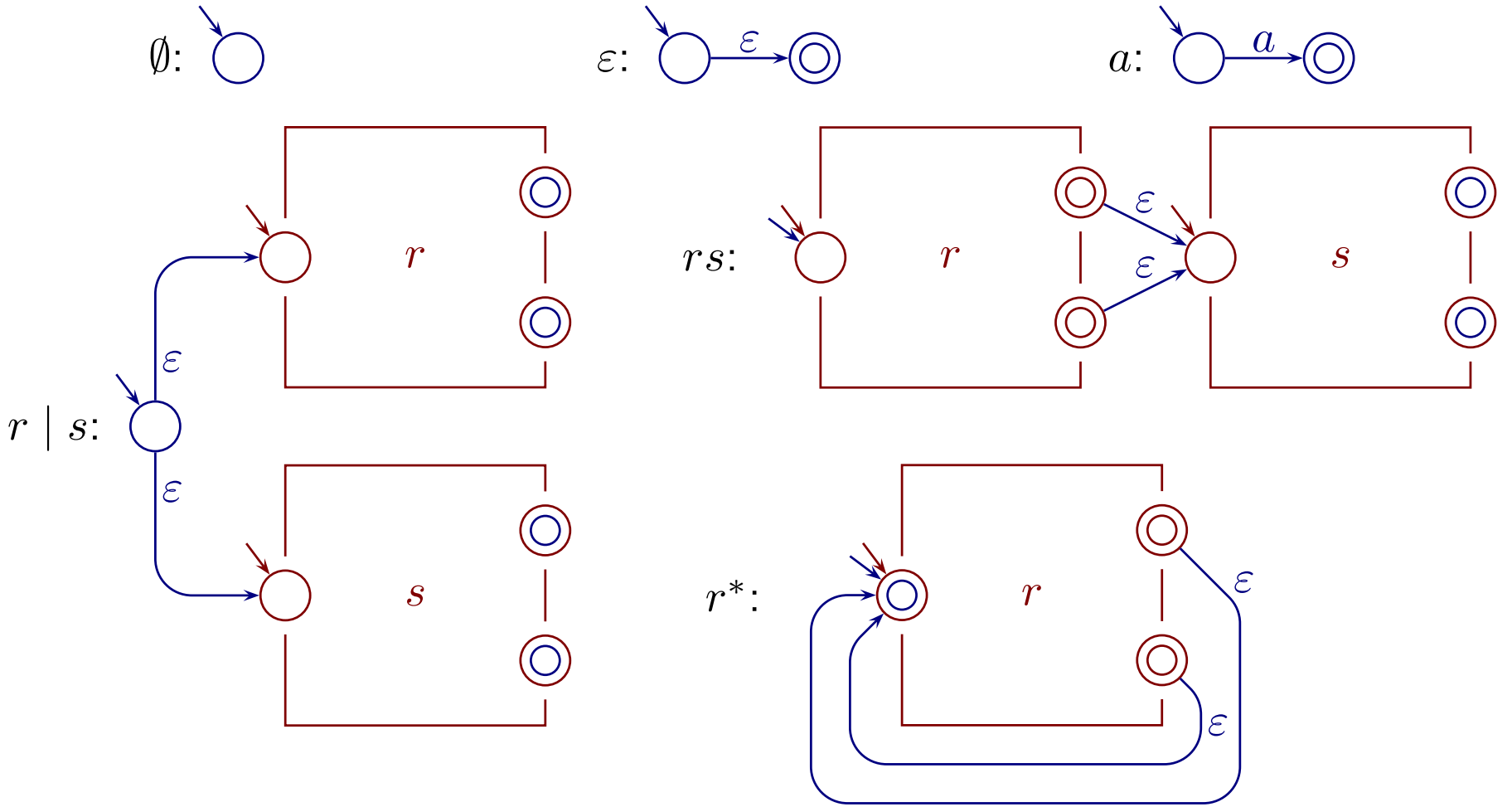
- $\mathcal{L}(r \mid s) = \mathcal{L}(s \mid r)$
- $\mathcal{L}(\varepsilon \mid r) = \{\varepsilon\} \cup \mathcal{L}(r)$
- $\mathcal{L}(\varepsilon r) = \mathcal{L}(r\varepsilon) = \mathcal{L}(r)$
- $\mathcal{L}(\varepsilon^*) = \{\varepsilon\}$
- $\mathcal{L}(\emptyset \mid r) = \mathcal{L}(r)$
- $\mathcal{L}(\emptyset r) = \mathcal{L}(r\emptyset) = \emptyset$
- $\mathcal{L}(\emptyset^*) = \{\varepsilon\}$

$\Rightarrow$ operaattoria $\emptyset$ ei muuten tarvittaisi, mutta se on ainoa keino saada $\mathcal{L}(r) = \emptyset$

Joitakin hieman mielenkiintoisempia kaavoja

- $\mathcal{L}(r(s \mid t)) = \mathcal{L}(rs \mid rt)$
- $\mathcal{L}((r \mid s)t) = \mathcal{L}(rt \mid st)$
- $\mathcal{L}((r \mid s)(t \mid u)) = \mathcal{L}(rt \mid ru \mid st \mid su)$
- $\mathcal{L}((r^*)^*) = \mathcal{L}(r^*)$
- $\mathcal{L}(r^* \mid s^*) \subseteq \mathcal{L}((r \mid s)^*)$

Säännöllisestä lausekkeesta voi muodostaa NFA:n, joka hyväksyy saman kielen

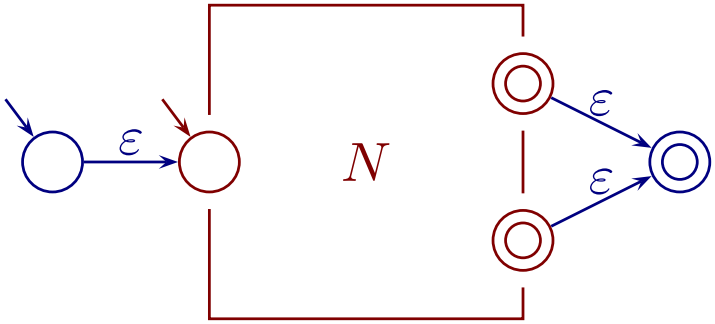


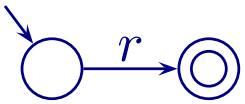
- helppoa ja nopeaa!

$\Rightarrow$ säännöllisillä lausekkeilla voi esittää enintään samat kielet kuin NFA:illa

NFA:sta voi muodostaa säännöllisen lausekkeen, joka tuottaa saman kielen

- muunnoksen välivaiheet ovat NFA:n kaltaisia olentoja, joiden kaarten niminä on säännöllisiä lausekkeita
  - kaarta kuljettaessa luetaan mikä tahansa ko. lausekkeen tuottama merkkijono
  - jokainen välivaihe hyväksyy saman kielen kuin alkuperäinen NFA
- aluksi lisätään uusi alku- ja lopputila
  - niihin ei kohdisteta jatkossa kuvattua tilan poistoa
  - vanhat lopputilat muutetaan ei-lopputiloiksi
  - tämä varmistaa, että alkutilaan ei tule eikä lopputilasta lähde kaaria

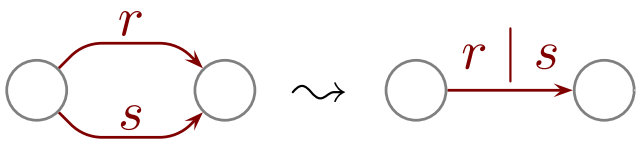


- sitten toistetaan seuraavia muunnoksia niin kauan kuin mahdollista
  - jokainen muunnos säilyttää muuntamansa osa-alueen merkkijonot
- lopulta jäljellä on joko   tai 

, missä $\mathcal{L}(r) = \mathcal{L}(N)$

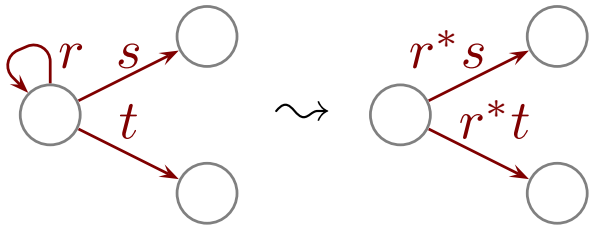
Tuplakaarten poisto

- tilojen määrä ei muutu
- kaarten määrä vähenee

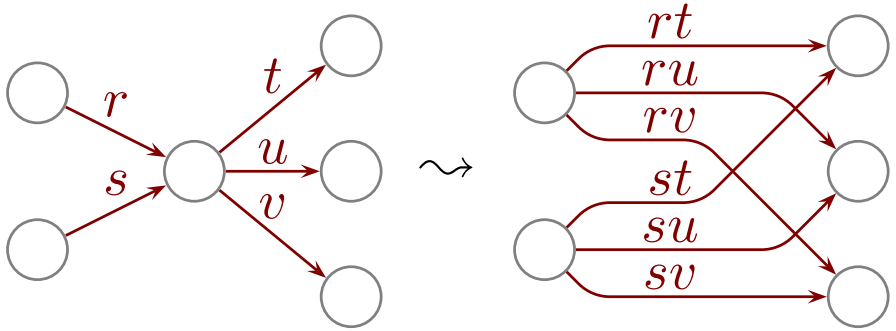


Paikallissilmukan poisto

- tilojen määrä ei muutu
- kaarten määrä vähenee



Tilan poisto



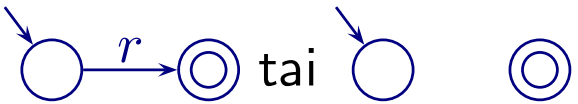
- saa soveltaa vain tilaan, jolla ei ole paikallissilmukoita
  - muutoin poistettava tila olisi samalla jokin kuvan muista tiloista
- ei sovelleta aikaisemmin lisättyihin alku- ja lopputilaan
- tilojen määrä vähenee
- kaarten määrä voi kasvaa, pysyä ennallaan tai vähentyä
- voi tuottaa tuplakaaria ja paikallissilmukoita

Algoritmin loppu

- tilojen väheneminen loppuu viimeistään $|Q|$ vähenemisen jälkeen
- sen jälkeen kaarten väheneminen loppuu joskus
  - miksi ei voida päätellä, että viimeistään $|\Delta| + |F|$ vähenemisen jälkeen?

$\Rightarrow$ algoritmi lopettaa

- mikään muunnos ei tuota lähtökaaria alussa lisätylle lopputilalle
- lopetusehdon vuoksi algoritmin lopetettua
  - paikallissilmukoita ei ole
  - muita tiloja ei ole, kuin alussa lisätyt alku- ja lopputila
  - tuplakaaria ei ole

$\Rightarrow$ lopputulos on muotoa 

tai

Huomioita muunnoksesta NFA $\rightsquigarrow$ säännöllinen lauseke

- helppoa ja melko nopeaa!
- lopputulos voi riippua muunnosten suorittamisjärjestyksestä

Olemme todistaneet seuraavan:

Lause 2.12 Jos kielen voi tuottaa säännöllisellä lausekkeella niin sen voi hyväksyä NFA:lla, ja päinvastoin.

Seuraava määritelmä kannattaa muistaa:

Määritelmä 2.13 Kieli on **säännöllinen** (*regular*), jos ja vain jos sen voi tuottaa säännöllisellä lausekkeella.

Edellä olevien perusteella

- kieli on säännöllinen, jos ja vain jos jokin NFA hyväksyy sen
- kieli on säännöllinen, jos ja vain jos jokin DFA hyväksyy sen
- $\{a^n b^n \mid n \in \mathbb{N}\}$ ei ole säännöllinen

Voidaan todistaa, että oikein käytettyjen sulkujen kieli ei ole säännöllinen

- esim. (()())() kuuluu siihen, mutta)(ei kuulu
- jos $\underbrace{(\cdots(}_{|Q|}\underbrace{)\cdots)}_{|Q|} \in \mathcal{L}(N)$, niin $\underbrace{(\cdots(}_{n}\underbrace{)\cdots)}_{|Q|} \in \mathcal{L}(N)$ jollekin $n < |Q|$

$\Rightarrow$ jos $\mathcal{L}(N)$ sisältää kaikki oikeat sulutukset, se sisältää myös vääriä

$\Rightarrow$ mikään $\mathcal{L}(N)$ ei ole täsmälleen oikeiden sulutusten joukko

Siksi tarvitaan voimakkaampi tapa ilmaista kieliä

2.8 Kuinka yleispätevä laskennan malli DFA, NFA ja säännölliset lausekkeet ovat?

Rajallisen muistin tapaukseen täysin yleispätevä

- mahdollisia muistin sisältöjä on vain äärellinen määrä
- mahdollisia (toisistaan erottuvia) herätteitä on vain äärellinen määrä

$\Rightarrow$ äärellinen luettelo riittää esittämään kaikki yhdistelmät ja sen, miten kuhunkin reagoidaan

  - ulos näkyvää vastetta on DFA:n perusmallissa vain hyväksyn / hylkään, mutta sen tilalla voisi olla monipuolinen joukko ulos näkyviä reaktioita
  - (pirteä, opettaja puhuu hyvin) $\rightarrow$ (pirteä, kuuntelen)
    (pirteä, opettaja puhuu vaikeita) $\rightarrow$ (väsyttää, kuuntelen)
    (väsyttää, opettaja puhuu sekavia) $\rightarrow$ (väsyttää, näprään kännykkää)
  - ihmisen toimintaa hyvin kuvaavan mallin pitäisi olla tätä esimerkkiä erittäin paljon yksityiskohtaisempi

- jokaiselle NFA on olemassa saman kielen hyväksyvä DFA
  $\Rightarrow$ epädeterminismi ei lisää laskentakykyä
- useimmissa tapauksissa luettelo olisi valtavan valtavan valtava

Sekä arki- että ohjelmointikokemuksen valossa melko vähään pystyvä

- ihminen osaa tarkastaa kuuluuko syöte kieleen $\{a^n b^n \mid n \in \mathbb{N}\}$, NFA ei osaa

- on helppo ohjelmoida tietokone tarkastamaan sama
- ihmisen kognitio mielletään paljon rikkaammaksi ilmiöksi kuin valtavan valtavasta luettelosta katsominen
- tietokoneohjelmammekin ovat enimmäkseen enemmän kuin luettelosta katsojia

Tietokoneen muistikapasiteetti on varmasti ja ihmisen aivojen kapasiteetti lähes varmasti rajallinen

$\Rightarrow$ äärellisen automaatin pitäisi olla mielekäs malli jopa ihmisen henkisille toiminnoille!

Ristiriidan osittainen ratkaisu

- jos ihan tarkkoja ollaan, niin $\sigma \stackrel{?}{\in} \{a^n b^n \mid n \in \mathbb{N}\}$ *vaatii* rajattomasti muistia
  - 32-bittinen luku ei riitä, kun syötteen pituus ylittää 4294967295
  - 64-bittinen luku ei riitä, kun syötteen pituus ylittää 18446744073709551615
- tässä esimerkissä pieni muisti riittää kaikille syötteille, joita voi käytännössä esiintyä
  - näkyvässä maailmankaikkeudessa on noin 10^{80} elektronia
  - syötteelle pituudeltaan 10^{80} riittää noin 266 bittiä
  - $\approx$ 0,000001 % nykyaikaisen kännykän muistista

$\Rightarrow$ ensimmäisenä vastaan tuleva käytännön rajoite on harvoin muistin loppuminen

$\Rightarrow$ äärelliset automaatit ovat käytännössä huono tietokoneen malli
  - tietokoneissa on niin paljon muistia, että ääretön on sille yleensä hyvä likiarvo

3 Yhteysriippumattomat kieliopit

Yhteysriippumaton kielioppi (*context-free grammar, CFG*) $G = (\Sigma, V, S, R)$ on keino määritellä enemmän kieliä kuin säännölliset kielet

- aakkosto Σ
  - äärellinen joukko
  - kuten säännöllisissä lausekkeissa: tuotetut merkkijonot muodostuvat aakkosista
  - aakkosia kutsutaan myös *loppusymboleiksi* (*terminal symbol*, *terminal*)
- välisymbolit V (*nonterminal symbol*, *nonterminal*)
  - äärellinen joukko
  - $\Sigma \cap V = \emptyset$
  - jokainen välisymboli $A \in V$ tuottaa jonkin kielen $\mathcal{L}(A) \subseteq \Sigma^*$
- alkusymboli S (*start symbol*)
  - $S \in V$
  - se välisymboli, jonka tuottama kieli on koko kieliopin tuottama kieli
- säännöt R (*rule*, *rewrite rule*, *production*)
  - äärellinen joukko pareja (A, σ), missä $A \in V$ ja $\sigma \in (\Sigma \cup V)^*$
  - sama A voi olla nollan, yhden tai useamman säännön vasen puoli
  - $\mathcal{L}(A)$ on unioni joukoista $\mathcal{L}(\sigma)$ niille σ, joille $(A, \sigma) \in R$
  - jokainen $X \in V$ saa esiintyä säännön (A, σ) σ:ssa, myös A itse

  $\Rightarrow$ $\mathcal{L}(A)$ ja $\mathcal{L}(\sigma)$ määräytyvät rekursiivisesti kuten myöhemmin kerrotaan

Käytännössä säännöt esitetään usein seuraavasti (**Backus-Naur Form** eli **BNF**)

$$
\begin{aligned}
A_1 &::= \cdots \mid \cdots \mid \cdots \\
A_2 &::= \cdots \mid \cdots \mid \cdots \\
&\vdots \\
A_n &::= \cdots \mid \cdots \mid \cdots
\end{aligned}
$$

- kukin $\cdots$ on merkkijono joukosta $\Sigma \cup V$
- kukin välisymboli esiintyy vasemmalla puolella enintään kerran
- osan "$X ::=$" oikealla puolella on lueteltu $|$:lla erotettuina ne σ, joille $(X, \sigma) \in R$
- jollei välisymbolien joukkoa ole sanottu, se on
  - joko vasemmilla puolilla esiintyvät merkit
  - tai kieliopissa esiintyvät isot kirjaimet mahdollisine koristeluineen (esim. A', A_1)
- jollei aakkostoa ole sanottu, se on ne merkit oikeilla puolilla, jotka eivät ole välisymboleita
- jollei alkusymbolia ole sanottu, se on ensimmäisen säännön vasen puoli
- esimerkki

$$
\begin{aligned}
S &::= T \mid S{+}T \mid S{-}T \\
T &::= 1 \mid 2 \mid (S)
\end{aligned}
$$

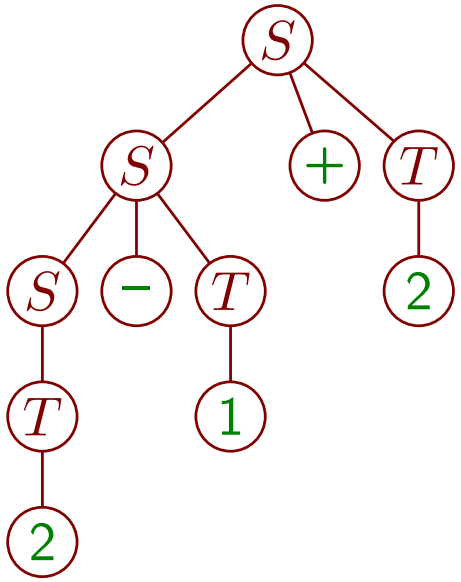
  - aakkosto on $\{+, -, 1, 2, (,)\}$
  - välisymbolit ovat $\{S, T\}$
  - alkusymboli on S

Ennen tuotetun kielen määritelmän esittämistä käsittelemme muutaman esimerkin

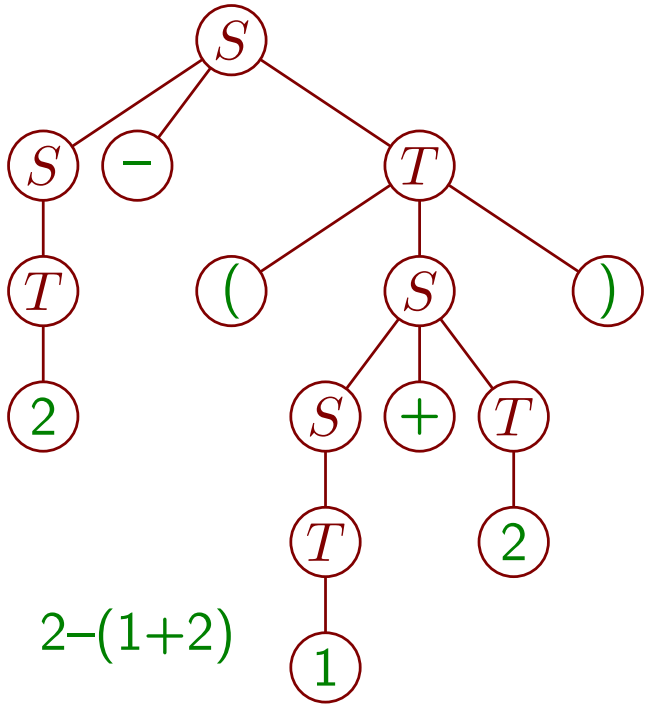
- isot kirjaimet ovat välisymboleita ja pienet aakkosia
- ε:n, peräkkäisyyden ja pystyviivan $|$ merkitys on sama kuin säännöllisissä lausekkeissa
- välisymbolin paikalla saa olla mikä tahansa sen kieleen kuuluva merkkijono
  $\Rightarrow$ kieleen kuuluvia merkkijonoja voidaan muodostaa lyhyistä alkaen
- jos $A ::= \varepsilon \mid aA$, niin $\mathcal{L}(A) = a^* = \{\varepsilon, a, aa, aaa, \ldots\}$
  - $R = \{(A, \varepsilon), (A, aA)\}$
  - koska $(A, \varepsilon) \in R$, saamme $\varepsilon \in \mathcal{L}(A)$
  - koska $(A, aA) \in R$ ja $\varepsilon \in \mathcal{L}(A)$, saamme $a\varepsilon \in \mathcal{L}(A)$ eli $a \in \mathcal{L}(A)$
  - koska $(A, aA) \in R$ ja $a \in \mathcal{L}(A)$, saamme $aa \in \mathcal{L}(A)$
  - koska $(A, aA) \in R$ ja $aa \in \mathcal{L}(A)$, saamme $aaa \in \mathcal{L}(A)$
  - koska $(A, aA) \in R$ ja $aaa \in \mathcal{L}(A)$, saamme $aaaa \in \mathcal{L}(A)$
  - $\ldots$
- jos $A ::= \varepsilon \mid aAb$, niin $\mathcal{L}(A) = \{\varepsilon, ab, aabb, aaabbb, \ldots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
  - $R = \{(A, \varepsilon), (A, aAb)\}$
  - koska $(A, \varepsilon) \in R$, saamme $\varepsilon \in \mathcal{L}(A)$
  - koska $(A, aAb) \in R$ ja $\varepsilon \in \mathcal{L}(A)$, saamme $a\varepsilon b \in \mathcal{L}(A)$ eli $ab \in \mathcal{L}(A)$
  - koska $(A, aAb) \in R$ ja $ab \in \mathcal{L}(A)$, saamme $aabb \in \mathcal{L}(A)$
  - koska $(A, aAb) \in R$ ja $aabb \in \mathcal{L}(A)$, saamme $aaabbb \in \mathcal{L}(A)$
  - koska $(A, aAb) \in R$ ja $aaabbb \in \mathcal{L}(A)$, saamme $aaaabbbb \in \mathcal{L}(A)$
  - $\ldots$

aakkoston sulku
metakielen sulku

- jos $S ::= T \mid S{+}T \mid S{-}T$ ja $T ::= 1 \mid 2 \mid (S)$, niin
  - $R = \{(S, T), (S, S{+}T), (S, S{-}T), (T, 1), (T, 2), (T, (S))\}$
  - $\mathcal{L}(S) = \{1, 2, 1{+}1, 1{+}2, 2{+}1, 2{+}2, 1{-}1, \ldots, (1), (2), \ldots, 2{-}1{+}2, (2{-}(1{+}2)), \ldots\}$
  - $\mathcal{L}(T) = \{1, 2, (1), (2), (1{+}1), (1{+}2), \ldots, (2{-}1{+}2), (2{-}(1{+}2)), \ldots\}$



2–1+2



2–(1+2)

- jos $A ::= \mathsf{a} \mid [B]$ ja $B ::= AA$, niin
  - $R = \{(A, \mathsf{a}), (A, [B]), (B, AA)\}$

A	a	[aa]	[a[aa]] [[aa]a] [[aa][aa]]	...
B	aa	a[aa] [aa]a [aa][aa]	a[a[aa]]	...

- jos $A ::= aA$, niin $\mathcal{L}(A) = \emptyset$
  - $R = \{(A, aA)\}$
  - yhtään merkkijonoa ei voida tuottaa, koska aA tuottaa jotain vain jos A:ssa on valmiiksi jotain
- jos $A ::= B$ ja $B ::= AA$, niin $\mathcal{L}(A) = \emptyset$
  - $R = \{(A, B), (B, AA)\}$
- jos $A ::= A$, niin $\mathcal{L}(A) = \emptyset$
  - $R = \{(A, A)\}$

Yhteysriippumattoman kieliopin määrittelemä kieli

- jos $V = \{A_1, \ldots, A_n\}$, niin $\mathcal{L}(A_1)$, ..., $\mathcal{L}(A_n)$ ovat pienimmät joukot, joille pätee
  - jos $x \in \Sigma$, niin $\mathcal{L}(x) = \{x\}$
  - jos $x_1 \cdots x_n \in (\Sigma \cup V)^*$, niin $\mathcal{L}(x_1 \cdots x_n) = \{\sigma_1 \cdots \sigma_n \mid \sigma_1 \in \mathcal{L}(x_1) \wedge \ldots \wedge \sigma_n \in \mathcal{L}(x_n)\}$
  - jos $(A, x_1 \cdots x_n) \in R$, niin $\mathcal{L}(x_1 \cdots x_n) \subseteq \mathcal{L}(A)$
- koko kieliopin määrittelemä kieli on sen alkusymbolin määrittelemä kieli
  - aivan kuten NFA:n hyväksymä kieli on sen alkutilan hyväksymä kieli

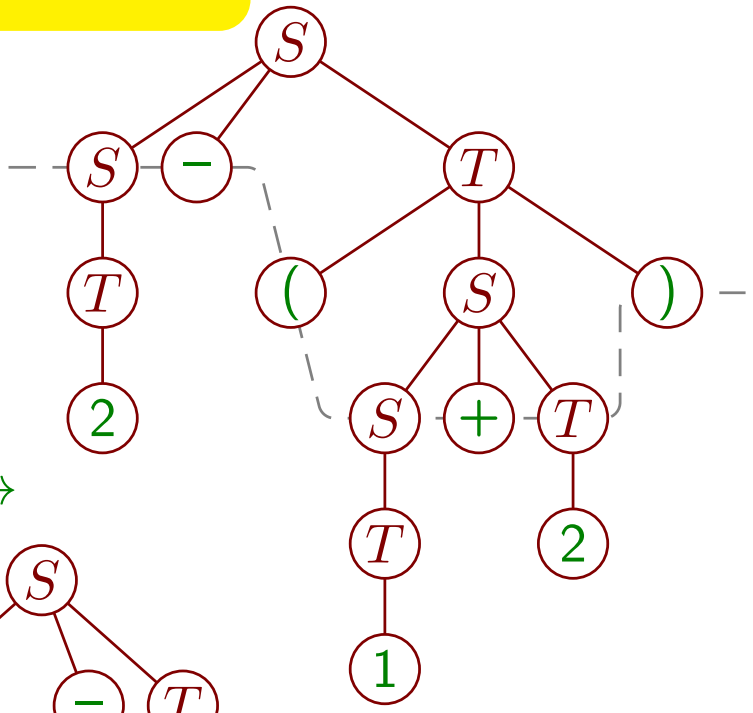
Määritelmän voidaan tulkita tuottavan merkkijonot "alhaalta ylös"

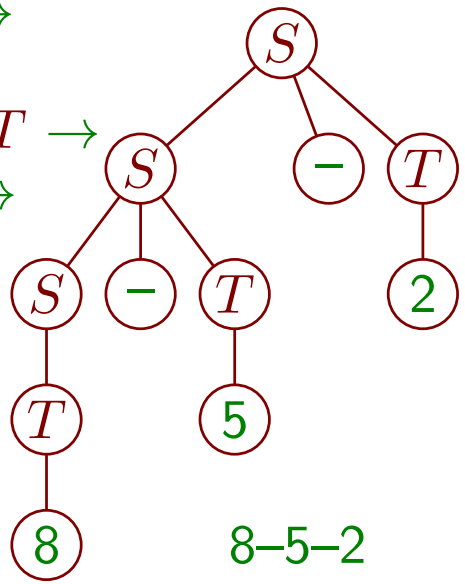
Merkkijono kuuluu yhteysriippumattoman kieliopin esittämään kieleen jos ja vain jos se voidaan tuottaa seuraavasti "ylhäältä alas"

- aloitetaan alkusymbolista
- mikä tahansa välisymboli korvataan sen minkä tahansa säännön oikealla puolella
- näin jatketaan, kunnes lopputuloksessa ei ole välisymboleita
- esim. $S ::= T \mid S{+}T \mid S{-}T$ ja $T ::= 0 \mid \ldots \mid 9 \mid (S)$
  - esim. oikeanpuoleisin ensin $S \to S{-}T \to S{-}(S) \to S{-}(S{+}T) \to S{-}(S{+}2) \to S{-}(T{+}2) \to S{-}(1{+}2) \to T{-}(1{+}2) \to 2{-}(1{+}2)$
  - esim. vasemmanpuoleisin ensin $S \to S{-}T \to T{-}T \to 2{-}T \to 2{-}(S) \to 2{-}(S{+}T) \to 2{-}(T{+}T) \to 2{-}(1{+}T) \to 2{-}(1{+}2)$

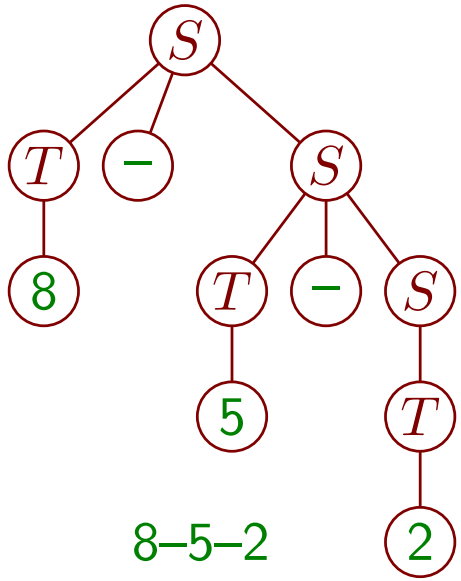
Jäsennyspuu esittää merkkijonon tuottamisen ottamatta kantaa vaiheiden järjestykseen

$\Rightarrow$ merkkijono kuuluu kieleen jos ja vain jos sillä on jäsennyspuu





8–5–2

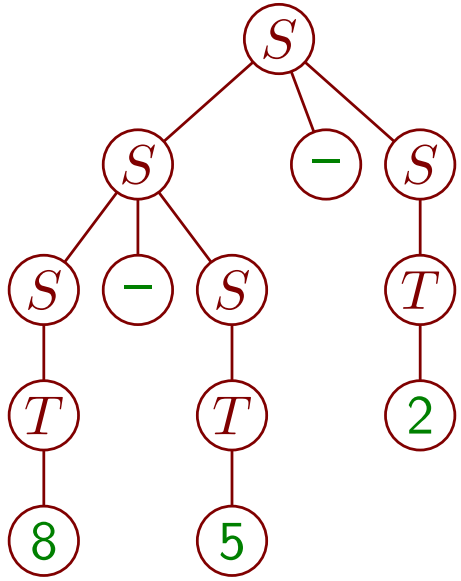


8–5–2

Eri kieliopit voivat tuottaa saman kielen eri jäsennyspuilla

- esim. $S ::= T \mid T{+}S \mid T{-}S$ ja $T ::= 0 \mid \ldots \mid 9 \mid (S)$

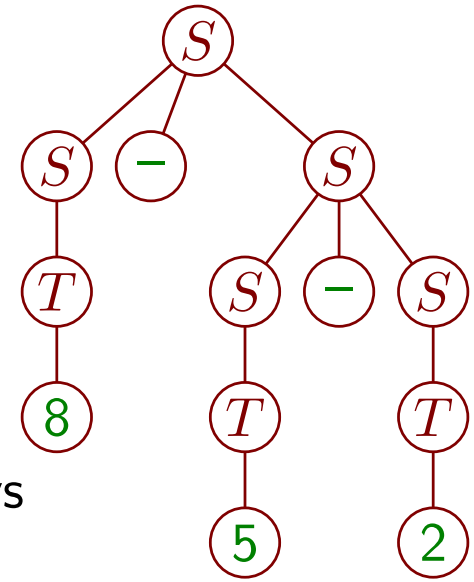
- mikäli mahdollista, kannattaa valita kielioppi, jonka tuottamat jäsennyspuut vastaavat kohteena olevia rakenteita
  - matematiikassa $8-5-2=(8-5)-2=3-2=1$
    eikä $8-5-2=8-(5-2)=8-3=5$
  - $\Rightarrow$ esim. $S ::= T \mid S{+}T \mid S{-}T$
    eikä $S ::= T \mid T{+}S \mid T{-}S$



- esim.
  - potenssilasku sitoo voimakkaammin kuin kertolasku
  - kertolasku sitoo vasemmalle ja potenssilasku sitoo oikealle
  - esim. $3ax^{2^n} = (3a)(x^{(2^n)})$

$$K ::= P \mid KP$$
$$P ::= A \mid A^P$$

$\Rightarrow$ kielioppi saadaan tukemaan rakenteen ymmärtämistä



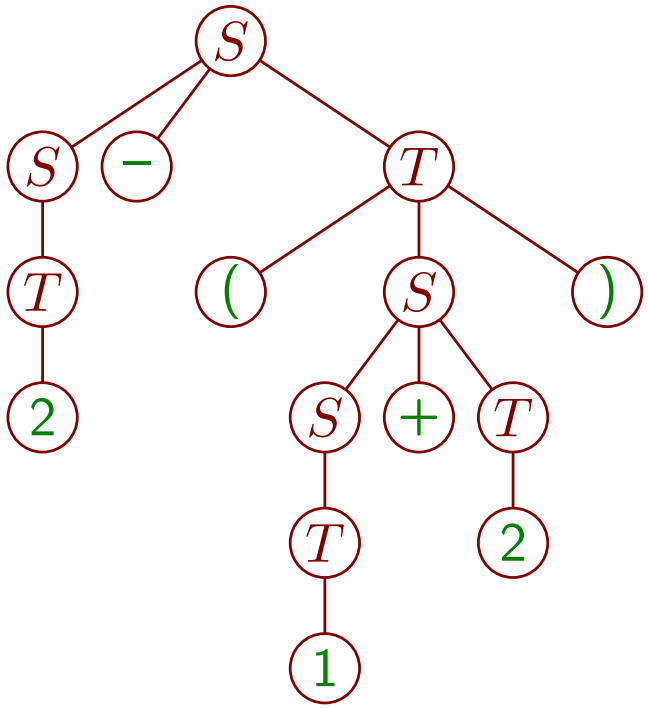
Jos ja vain jos jollekin merkkijonolle on kaksi eri jäsennyspuuta, niin kielioppi on *monikäsitteinen* (*ambiguous*)

- esim. $S ::= T \mid S{+}S \mid S{-}S$ ja $T ::= 0 \mid \ldots \mid 9 \mid (S)$
- tällaisiakin kielioppeja näkee, erityisesti kun on haluttu tiivis esitys
  - esim. $L ::= 0 \mid x \mid L+L \mid L-L \mid LL \mid L/L \mid L^L$
- on olemassa yhteysriippumattomia kieliä, joiden kaikki kieliopit ovat monikäsitteisiä

Kielioppi *saattaa* olla laadittu niin, että jäsennyspuut vastaavat asian rakennetta, mutta varmaa se ei ole

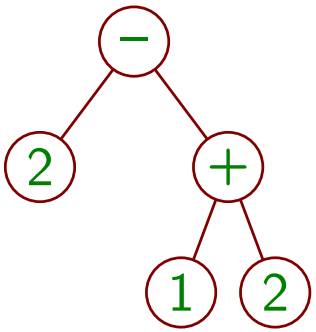
- kielen käsite ei ole mitään muuta kuin merkkijonojen jako kahtia: mukana ja ulkona
- kieliopin avulla *voidaan* esittää varsin paljon kielen syntaktisesta rakenteesta, *jos halutaan*
- on luvallista ja toisinaan järkevää käyttää kielioppeja ilmaisemaan kieliä, joiden merkkijonoilla ei ole tärkeää syntaktista rakennetta
  - millä perusteella $A ::= \varepsilon \mid Aa$ on oikeampi tai väärempi kuin $A ::= \varepsilon \mid aA$?
- toisinaan on helpompaa rakentaa jäsentäjä "väärälle" kieliopille ja korjata tulos myöhemmissä käsittelyvaiheissa
  - kohta kuvattava jäsennystapa onnistuu kieliopille $S ::= T \mid T{+}S \mid T{-}S$ mutta ei $S ::= T \mid S{+}T \mid S{-}T$

Jäsennyspuu on eri asia kuin lausekepuu



- jäsennyspuu esittää miten merkkijono saadaan kieliopista
  - sisältää kaikki välisymbolit, jäsennystä ohjaavat välimerkit yms.
  - ei välttämättä ole yhteyttä asian loogiseen rakenteeseen, koska voidaan käyttää "vääriä" jäsennyspuita tuottavaa kielioppia
- lausekepuu esittää asian loogisen rakenteen
- jotta voisi olla lausekepuu, tarvitaan lauseke!
  - $A ::= \varepsilon \mid aAb$ tuottaa mm. $aabb$, mikä olisi sen lausekepuu?

Rekursiivisesti laskeutuva jäsentäminen (**recursive descent parsing**)

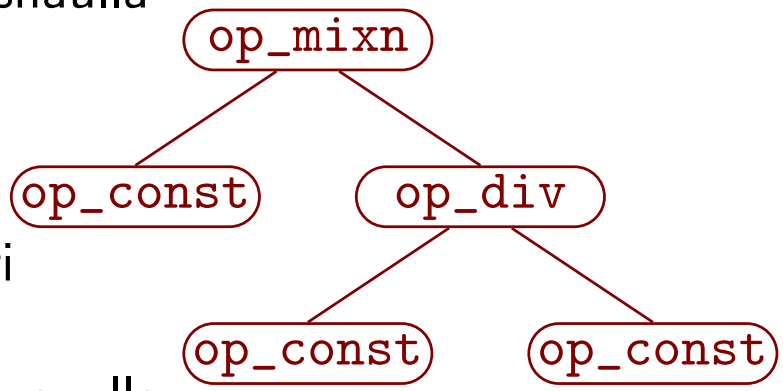


- toimii (tehokkaasti) vain osalle yhteysriippumattomia kielioppeja
- on poikkeuksellisen helppo toteuttaa
- esittelemme tuotantokäytössä olevan todellisen koodin avulla: MathCheckista lokakuussa 2018 kopioitu osuus
  - siitä ei kannata opiskella muuta kuin jäsentäminen
- muuttamaton kopio, paitsi
  - joidenkien osien tilalla on . . .
  - tyhjän tilan käyttöä on muutettu, jotta esimerkki jakautuisi ruuduille hyvin

- rakentaa lausekepuun, kukin aliohjelma palauttaa osoittimen solmuun
- `pm_now` on jäsennystä ohjaava bittijono (`pm_type` on `unsigned`)
  - kokonaislukuaritmetiikassa murtoluvut on kielletty
  - moduloaritmetiikassa eksponentti saa olla vain kokonaislukuvakio
  - ...
- logiikan ja aritmetiikan yhdistämiseen liittyi todella vaikea ongelma: minkä rakenteen $($ aloittaa
  - esim. $((x+1)(x-1) < 0 \vee x = -1) \wedge x \neq 0$
- ratkaisin ison työn jälkeen vippaskonstilla
  - tämä selitys on tässä koska tämä liittyy heti koodin alussa olevaan asiaan, mutta tämä kannattaa lukea myöhemmin (tai ei ollenkaan)
  - yrittää ensisijaisesti logiikkaa
  - logiikan sisällä voi olla aritmetiikkaa

  $\Rightarrow$ luettuaan $((x+1$ ei ole valmis aritmetiikan loppusulkuun, vaan odottaa löytävänsä esim. $+$ tai $\leq$
  - peruuttaa rekursiossa kohtaan jossa luki aritmetiikan alkusulun
  - tallettaa $(x+1)$:n jäsennystuloksen muuttujaan `par_expr`
  - tekee rekursiivisia kutsuja mitään lukematta logiikan, vertailun ja aritmetiikan mukaan, kunnes on aliohjelmassa `parse_arithm_atom`
  - nollaa `par_expr`:n ja jatkaa ikään kuin `parse_arithm_atom` olisi rakentanut jäsennystuloksen
- `tkn_number` on numerojono ja `tkn_val` sen edustama lukuarvo

- `parse_is` kirjaa että tätä tekstialkiota yritettiin ja kertoo, täsmäsikö
  $\Rightarrow$ syntaksivirheilmoitukseen saadaan luettelo tekstialkioista, joita yritettiin
- `new_expr` rakentaa lausekepuun solmun, parametreinaan
  - osoitin vasempaan alipuuhun (oletusarvo: ei alipuuta)
  - solmun sisältämä operaattori (oletusarvo: `op_const`)
  - osoitin oikeaan alipuuhun (oletusarvo: ei alipuuta)
  - etumerkitön kokonaisluku (solmun edustama numeroarvo, oletusarvo: 0)
- `get_token` lukee syötettä ja tulkitsee seuraavan tekstialkion
  - numerojono, esim. `12321`
  - numerojono.numerojono, esim. `3.14159`
  - yksittäinen kirjain, edustaa yleensä muuttujaa, esim. `x`
  - avainsana, esim. `log`, `end_of_answer` tai erikoismerkkialkio, esim. `<=`, `/\`
  - voisi periaatteessa olla DFA, mutta ei ole
  - uusien tekstialkioiden lisäämisen helpottamiseksi tunnistaa avainsanat ja erikoismerkkijonot etsimällä taulukosta puolitushaulla (tätä kirjoitettaessa taulukon koko on 245)
- `tkn_div` on `/`
- `parse_pass`
  - ohittaa tekstialkion jos se on sama kuin parametri
  - muutoin antaa virheilmoituksen
- sekaluvut kuten $5\frac{2}{3}$ esitetään `op_mixn`:n ja `op_div`:n avulla
  - halusin selvitä ilman, että solmulla voi olla yli kaksi lasta



```
expression *parse_arithm_atom( pm_type pm_now ){

  /* If parse_pred_or_expr delivered an expression, use it. */
  if( par_expr ){ expression *ee = par_expr; par_expr = 0; return ee; }

  /* Unsigned integer (plus fraction) number constant */
  if( parse_is( tkn_number ) ){
    expression *e1 = new_expr( tkn_val );
    get_token();
    if( pm_now & pm_int || !parse_is( tkn_number ) ){ return e1; }

    /* Fractional part */
    expression *e2 = new_expr( tkn_val );
    get_token(); parse_pass( tkn_div );
    expression *e3 = new_expr( tkn_val );
    parse_pass( tkn_number );
    return new_expr( e1, op_mixn, new_expr( e2, op_div, e3 ) );
  }

  /* Decimal number constant */
  if( !( pm_now & ( pm_int | pm_mod ) ) && parse_is( tkn_decimal ) ){
    expression *ee = new_expr( tkn_val ); get_token(); return ee;
  }
```

- muuttujien jäsentäminen
  - moduloaritmetiikalla on oma muuttujatyyppi
  - muuten muuttujat ovat reaalilukutyyppisiä
  - `pm_now` kertoo saako tässä esiintyä muuttuja ja saako se olla ennen esiintymätön
  - testien järjestys on valittu siten, että virheilmoitus olisi hyvä:
    jos ollaan moduloaritmetiikassa ja ei tule muuttujaa lainkaan,
    `parse_is_var( vtp_R, pm_now ) ) && !( pm_now & pm_mod )`
    tuottaisi virheilmoitukseen myös että odotettiin reaalilukumuuttujaa
- kovista suluista tulee ja tavallisista ei tule solmu lausekepuuhun
  - tarpeettomat tavalliset sulut eivät näy ja tarpeettomat kovat näkyvät kun MathCheck tulostaa lausekkeen
- `. . .`:n paikalla on mm. $\sqrt{\cdots}$, $\lceil \cdots \rceil$, π, trigonometriset funktiot ja derivaatta
- `parse_fact` jäsentää kertoman $n!$
- potenssi
  - joko kertoma tai kertoma potenssiin potenssi

  $\Rightarrow$ vastaa sitä, että potenssi sitoo oikealle
  - potenssin potenssi on toteutettu rekursiolla
  - moduloaritmetiikassa eksponentti saa olla vain kokonaisluku (saa olla negatiivinen)
  - `tkn_sup` on `^`
  - vasemman puolen sitovuustaso vastaa matematiikan käytäntöä, esim. $ax^2 = a(x^2)$
  - matematiikan käytäntö ei kerro oikean puolen sitovuustasoa, esim. $x^{n-i}y^i$
  - kuten LaTeX, valitsin oikealle puolelle saman sitovuustason

```
  /* Variable */
  if(
    ( !( pm_now & pm_mod ) && parse_is_var( vtp_R, pm_now ) ) ||
    ( pm_now & pm_mod && parse_is_var( vtp_mod, pm_now ) )
  ){ return parse_var( pm_now ); }

  /* Ordinary and hard parentheses */
  if( parse_is( tkn_lp ) ){
    get_token();
    expression *ee = parse_expression( pm_now & pm_not_in );
    parse_pass( tkn_rp ); return ee;
  }
  if( parse_is( tkn_lP ) ){
    get_token();
    expression *ee = new_expr(
      op_hpar, parse_expression( pm_now & pm_not_in )
    );
    parse_pass( tkn_rP ); return ee;
  }
  . . .
  parse_error(); return expr_dummy;
}
expression *parse_fact( pm_type pm_now ){ . . . }
```

```
/* Power ^ */
expression *parse_pow( pm_type pm_now ){
  expression *ee = parse_fact( pm_now );
  if( !parse_is( tkn_sup ) ){ return ee; }
  get_token();
  if( !( pm_now & pm_mod ) ){
    return
      new_expr( ee, op_pow, parse_pow( pm_now & pm_not_in ) );
  }
  if( parse_is( tkn_number ) ){
    number val = tkn_val;
    if( val.is_int_type() ){ get_token(); }
    else{ err_set_inp( "Here the number must be an integer" ); }
    return new_expr( ee, op_pow, new_expr( val ) );
  }
  parse_error(); return ee;
}
/* Division */
expression *parse_div( pm_type pm_now ){
  expression *ee = parse_pow( pm_now );
  if( ( pm_now & pm_int ) || !parse_is( tkn_div ) ){ return ee; }
  get_token(); return new_expr( ee, op_div, parse_pow( pm_now ) );
}
```

- jakolasku ei sido oikealle eikä vasemmalle harhaanjohtavuusvaaran vuoksi
  - pitäisikö `x/2/3` tarkoittaa $\frac{\frac{x}{2}}{3}$ vai $\frac{x}{\frac{2}{3}}$?
  - jos ollaan rajoituttu kokonaislukuihin, käsitellään vain yksi potenssi
  - muutoin, jos jakomerkkiä ei tule, käsitellään vain yksi potenssi
  - muutoin käsitellään potenssi jaettuna potenssilla
  - nytkin ehtojen järjestys on tärkeä virheilmoituksen kannalta
- $2\frac{1}{3}$ on käsiteltävä (ja käsitellään) aivan eri lailla kuin $2\frac{n}{3}$!
- näkymätön kertolasku
  - paljon vaihtoehtoja, mitä voi tulla siinä kohdassa missä muutoin olisi kertomerkki
  - matematiikan säännöt ovat epäjohdonmukaisia,
    esim. $\sin 2x = 2 \sin x \cos x$ ja $\frac{\partial}{\partial x} x \sin x + x^2$
    esim. tarkoittaako $|x|y|z|$ samaa kuin $(|x|)y(|z|)$ vai $|x(|y|)z|$?
  $\Rightarrow$ todella monimutkainen
  - yleisrakenne on sama kuin näkyvällä kertolaskulla
  - kutsuu alussa ja joka kierroksella `parse_div( pm_now )`
  - näkymättömälle kertolaskulle tarvittiin symboli mm. opettajan komentoon "tätä ei saa käyttää vastauksessa", valitsin `#*`
- `3 pi/2 n` tuottaa $3\frac{\pi}{2}n$ eikä esim. $\frac{3\pi}{2n}$
  - oikealla puolella vaikutus merkitykseen on olennainen
  - arvioin valitun vaihtoehdon vähemmän harhaanjohtavaksi kuin hylätyn
  - vasemmalla puolella ero ei vaikuta merkitykseen
  - valitsin samankaltaiseksi oikean puolen kanssa

```
/* Unwritten multiplication */
expression *parse_inv_prod( pm_type pm_now ){ . . . }

/* Star multiplication * */
expression *parse_prod( pm_type pm_now ){
  expression *ee = parse_inv_prod( pm_now );
  while( parse_is( tkn_ast, tkn_Usdot ) ){
    get_token(); ee = new_expr( ee, op_vprod, parse_inv_prod( pm_now ) );
  }
  return ee;
}

/* Unary + and - */
expression *parse_sign( pm_type pm_now ){
  if(
    !par_expr && ( parse_is( tkn_plus ) || parse_is( tkn_Uminus, tkn_minus ) )
  ){
    op_type opr = op_tkn( tkn_now ); get_token();
    return new_expr( opr, parse_sign( pm_now ) );
  }
  return parse_prod( pm_now );
}
```

- näkyvä kertolasku
  - noudattaa vasemmalle liitännäisen operaattorin perusrakennetta
  - toisto on toteutettu silmukalla
  - `tkn_ast` on `*` ja `tkn_Usdot` on $\cdot$ (Unicode-merkki, joka ei ole ASCIIssa)
  - $2n \cdot 3x$ tulkitaan kuten $(2n) \cdot (3x)$ eikä kuten $2(n \cdot 3)x$, ero ei vaikuta merkitykseen
- etumerkit
  - `!par_expr` estää tätä lukemasta etumerkkiä, jos ollaan toipumassa edellä kuvatusta sulkujen väärintulkinnasta
  - aikaisemmissa rakenteissa tätä ei tarvittu, koska ne alkavat rekursiivisella läpikutsulla
  - toisto on toteutettu rekursiolla
  - `tkn_plus`, `tkn_Uminus` ja `tkn_minus` ovat `+`, $-$ ja `-`
- $-2 \cdot y$ tulkitaan kuten $-(2 \cdot y)$ eikä kuten $(-2) \cdot y$ ja $-2y$ kuten $-(2y)$ eikä $(-2)y$
  - poikkeaa ohjelmointikielissä yleisestä käytännöstä
  - jälkimmäinen vastaa sitä, miten matematiikan lausekkeet yleensä ladotaan
  - olisi tuntunut hassulta laittaa sitovuusjärjestyksessä etumerkit kertolaskujen väliin
  - ero ei vaikuta merkitykseen
- yhteen- ja vähennyslasku
  - sama rakenne kuin näkyvällä kertolaskulla
  - etumerkkien pitää sitoa näitä voimakkaammin, jottei tulisi $-x + 2 = -(x + 2)$
- koko lauseke on summa

```
/* Addition + and subtraction - */
expression *parse_sum( pm_type pm_now ){
  expression *ee = parse_sign( pm_now );
  while( parse_is( tkn_plus ) || parse_is( tkn_Uminus, tkn_minus ) ){
    op_type opr = op_tkn( tkn_now ); get_token();
    ee = new_expr( ee, opr, parse_sign( pm_now ) );
  }
  return ee;
}

/* Expression */
expression *parse_expression( pm_type pm_now ){ return parse_sum( pm_now ); }
```

- onko tämä yksinkertaista vai monimutkaista?
- jäsennyksen toteuttaminen on hyvin yksinkertaista, opiskele se
- monimutkaisuus tuli
  - matematiikan syntaktisten sääntöjen sekavuudesta
  - halusta käyttää samaa jäsennintä samankaltaisissa tilanteissa, kuten kokonaisluvut, reaaliluvut ja moduloaritmetiikka
  - hieman myös halusta tuottaa poikkeuksellisen hyvät virheilmoitukset
- on tavallista, että asia on yksinkertainen, mutta sovelluskohde monimutkainen

- kokemuksieni mukaan liika ideologisuus ei toimi
  - osa kauniista jäsennysalgoritmeista ei kykene antamaan hyviä virheilmoituksia
  - valmis toteutus ei läheskään aina taivu tilanteen vaatimuksiin
  - rajapinnan opiskelu ja sen kanssa tappeleminen saattaa olla työläämpää kuin sen takana olevan asian tekeminen itse
- ei ole harvinaista, että mitättömän tuntuinen yksityiskohta tuhoaa hienon ajatuksen!
- kyky miettiä vakiintuneita "totuuksia" uudelleen on suureksi hyödyksi
  - harva on huomannut, että $\sin 2x = 2 \sin x \cos x$ ei noudata julkilausuttuja sääntöjä
  - ohjelmointikielten sitovuustasot eivät olleetkaan aina MathCheckille hyvät
  - voidaan sallia muitakin kuin ASCII-merkkejä
    $\Rightarrow$ kaavoja voi jossain määrin maalata, kopioida ja pudottaa MathCheckille

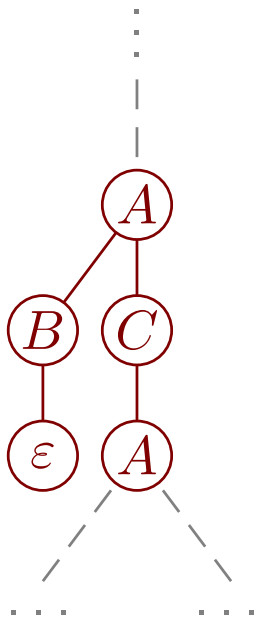
Mitä kieliä yhteysriippumattomat kieliopit voivat tuottaa?

- säännöllisten lausekkeiden $\sigma(\rho)^*\beta$ voidaan korvata
  - ottamalla käyttöön uusi välisymboli R
  - lisäämällä $R ::= \varepsilon \mid \rho R$
  - kirjoittamalla $\sigma(\rho)^*\beta$:n tilalle $\sigma R \beta$
- säännöllisten lausekkeiden $\emptyset$ voidaan korvata esim. lisäämällä O ja $O ::= O$
- muut säännöllisten lausekkeiden mahdollisuudet ovat yhteysriippumattomissa kieliopeissakin, ja samassa merkityksessä

$\Rightarrow$ yhteysriippumattomilla kieliopeilla voidaan ilmaista ainakin samat kielet kuin säännöllisillä lausekkeilla ja NFA:illa

- säännöllisillä lausekkeilla ja NFA:illa ei voi ilmaista $\{a^n b^n \mid n \in \mathbb{N}\}$
- yhteysriippumattomilla kieliopeilla voi: $A ::= \varepsilon \mid aAb$

$\Rightarrow$ yhteysriippumattomilla kieliopeilla voidaan ilmaista aidosti enemmän kieliä kuin säännöllisillä lausekkeilla ja NFA:illa

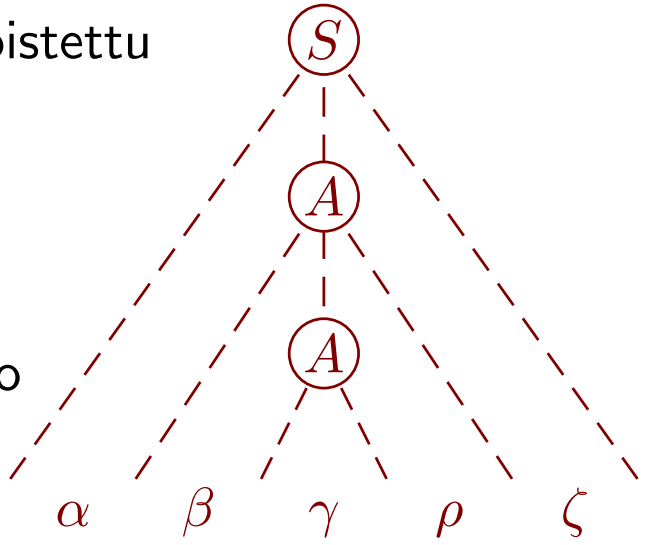


Yhteysriippumattomien kielten pumppauslemma

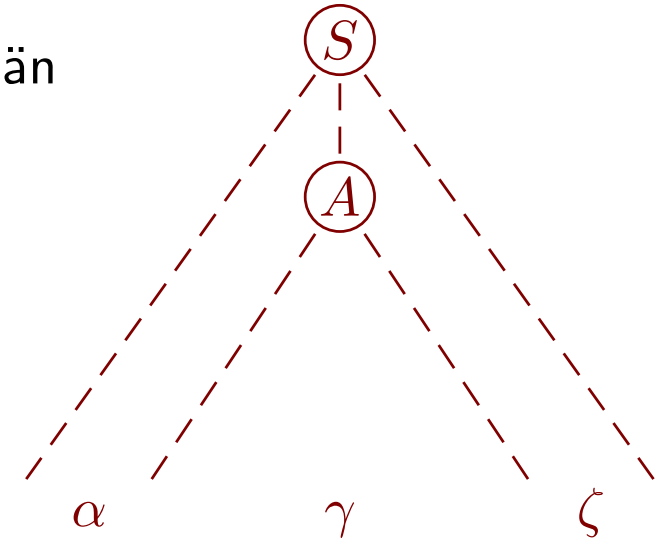
- tarkastellaan esimerkkiä

$$A ::= BC \mid \ldots$$
$$B ::= \varepsilon \mid \ldots$$
$$C ::= A \mid \ldots$$

- kuvassa näkyvä jäsennyspuun osa voidaan poistaa välistä, eikä kieli muutu
- jatkossa oletamme, että tällaiset jäsennyspuun osat on poistettu
- olkoon kielioppina (Σ, V, S, R)
- jos sama välisymboli A esiintyy jäsennyspuun ylhäältä alas -polulla kahdesti, niin ylemmästä esiintymästä peräisin oleva merkkijono on muotoa $\beta\gamma\rho$, missä
  - γ on alemmasta esiintymästä peräisin oleva merkkijono
  - β tai ρ tai kumpikaan ei ole ε
- koko tuotettu merkkijono on $\alpha\beta\gamma\rho\zeta$



- samaa kielioppia voidaan käyttää myös siten, että ylemmän A:n kohdalla jatketaan kuten alemman A:n kohdalla
  $\Rightarrow$ huomataan, että $\alpha\gamma\zeta \in \mathcal{L}(S)$
- sitä voidaan käyttää myös siten, että alemman A:n kohdalla jatketaan kuten ylemmän A:n kohdalla
  $\Rightarrow$ huomataan, että $\alpha\beta\beta\gamma\rho\rho\zeta \in \mathcal{L}(S)$
- samalla tavalla huomataan, että jokaisella $i \in \mathbb{N}$ pätee $\alpha\beta^i\gamma\rho^i\zeta \in \mathcal{L}(S)$



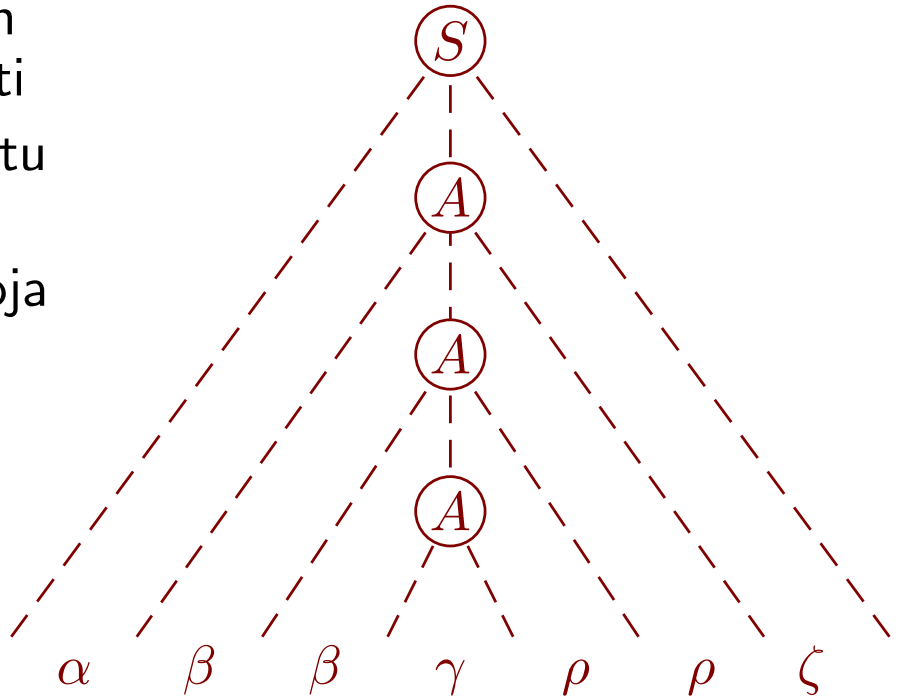
- jos jäsennyspuun polku on pituudeltaan ainakin $|V| + 1$, jokin välisymboli esiintyy siinä kahdesti

$\Rightarrow$ jäsennyspuita, joissa mikään välisymboli ei toistu millään polulla, on vain äärellinen määrä

$\Rightarrow$ ne tuottavat vain äärellisen määrän merkkijonoja

$\Rightarrow$ on olemassa raja n_1, jota pitempiä merkkijonoja ne eivät tuota

- myös sellaisia jäsennyspuita, joiden juuri on jokin välisymboli A, A toistuu enintään kerran, eikö mikään muu välisymboli toistu, on vain äärellinen määrä



$\Rightarrow$ on olemassa raja n_2, jota pitempiä merkkijonoja ne eivät tuota

- olkoon n maksimi luvuista $n_1 + 1$ ja n_2

- jos $\sigma \in \mathcal{L}(S)$ ja $|\sigma| \geq n$, niin
  - koska $n \geq n_1 + 1$, jollain polulla jokin välisymboli toistuu
  - valitsemme tarkasteluun toiston, jonka alla ei ole toistoa

  $\Rightarrow$ sen tuottama merkkijono $\beta\gamma\rho$ on pituudeltaan enintään n_2
  - koska $n \geq n_2$, pätee $|\beta\gamma\rho| \leq n$
  - σ jakautuu osiin kuten yllä, ts. $\sigma = \alpha\beta\gamma\rho\zeta$
- todistimme seuraavan lauseen:

Lause 3.1 Jokaisella yhteysriippumattomalla kieliopilla (Σ, V, S, R) on luku n siten, että jos $\sigma \in \mathcal{L}(S)$ ja $|\sigma| \geq n$, niin σ jakaantuu osiin $\sigma = \alpha\beta\gamma\rho\zeta$ siten, että

- $\beta \neq \varepsilon$ tai $\rho \neq \varepsilon$
- $|\beta\gamma\rho| \leq n$
- jokaisella $i \in \mathbb{N}$ pätee $\alpha\beta^i\gamma\rho^i\zeta \in \mathcal{L}(S)$

Miksi näimme vaivaa todistaaksemme, että $\beta \neq \varepsilon$ tai $\rho \neq \varepsilon$?

- ilman sitä tulos ei kertoisi mitään käyttökelpoista: jos $\beta = \rho = \varepsilon$, niin $\alpha\beta^i\gamma\rho^i\zeta = \alpha\beta\gamma\rho\zeta = \alpha\gamma\zeta$, joten lause ei kertoisi mitään mitä emme jo tietäisi
- ilman sitä tulos olisi triviaalisti totta kieliopeista yms. riippumatta: voitaisiin valita $\alpha = \sigma$ ja $\beta = \gamma = \rho = \zeta = \varepsilon$

$\Rightarrow$ ilman sitä lause olisi vain monimutkainen tapa sanoa ei mitään

Miksi näimme paljon vaivaa todistaaksemme, että $|\beta\gamma\rho| \leq n$?

- ilman sitä seuraava todistus ja muut samankaltaiset eivät onnistuisi

$\Rightarrow$ lauseesta olisi paljon vähemmän hyötyä

Kieli $\{a^n b^n c^n \mid n \in \mathbb{N}\} = \{\varepsilon, abc, aabbcc, aaabbbccc, \ldots\}$ ei ole yhteysriippumaton

- ne merkkijonot, joissa on ensin jokin määrä a:ta, sitten sama määrä b:tä, ja lopuksi sama määrä c:tä
- tarkastellaan mitä tahansa kielioppia, joka tuottaa ainakin kaikki ym. merkkijonot
- olkoon n pumppauslemman lupaama n

$\Rightarrow 3n \geq n$, joten $a^n b^n c^n = \alpha\beta\gamma\rho\zeta$ kuten pumppauslemmassa

- koska $|\beta\gamma\rho| \leq n$, osuus $\beta\gamma\rho$ ei ole niin pitkä, että se voisi alkaa a:lla ja loppua c:llä

$\Rightarrow$ siinä ei ole c:tä tai ei ole a:ta (tai ei kumpaakaan)

- jos siinä ei ole c:tä, niin kaikki c:t ovat ζ:ssa
  $\Rightarrow \alpha\gamma\zeta$ sisältää saman määrän c:tä kuin $\alpha\beta\gamma\rho\zeta$, mutta vähemmän a:ta ja / tai b:tä
  $\Rightarrow \alpha\gamma\zeta \notin \{a^n b^n c^n \mid n \in \mathbb{N}\}$ vaikka $\alpha\gamma\zeta \in \mathcal{L}(S)$
  $\Rightarrow \mathcal{L}(S) \neq \{a^n b^n c^n \mid n \in \mathbb{N}\}$
- jos siinä ei ole a:ta, niin samanlainen päättely tuottaa $\mathcal{L}(S) \neq \{a^n b^n c^n \mid n \in \mathbb{N}\}$

$\Rightarrow$ *jokainen kielioppi, joka tuottaa kaikki merkkijonot kielestä* $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, *tuottaa myös kieleen kuulumattomia merkkijonoja*

Kahden yhteysriippumattoman kielen leikkaus ei välttämättä ole yhteysriippumaton

- olkoot

$$\begin{array}{lcl@{\qquad}lcl} A & ::= & \varepsilon \mid aA & \mathcal{L}(A) & = & \{a^n \mid n \in \mathbb{N}\} \\ C & ::= & \varepsilon \mid cC & \mathcal{L}(C) & = & \{c^n \mid n \in \mathbb{N}\} \\ X & ::= & \varepsilon \mid aXb & \mathcal{L}(X) & = & \{a^n b^n \mid n \in \mathbb{N}\} \\ Y & ::= & \varepsilon \mid bYc & \mathcal{L}(Y) & = & \{b^n c^n \mid n \in \mathbb{N}\} \\ S & ::= & XC & \mathcal{L}(S) & = & \{a^n b^n c^m \mid n \in \mathbb{N} \wedge m \in \mathbb{N}\} \\ T & ::= & AY & \mathcal{L}(T) & = & \{a^n b^m c^m \mid n \in \mathbb{N} \wedge m \in \mathbb{N}\} \end{array}$$

- $\{a^n b^n c^n \mid n \in \mathbb{N}\} = \mathcal{L}(S) \cap \mathcal{L}(T)$

$\Rightarrow$ yhteysriippumattomilla kieliopeilla ilmaistuja rajoitteita ei välttämättä voi yhdistää siihen tapaan miten teimme tuloautomaateilla

- on kuitenkin todistettu, että yhteysriippumattoman ja säännöllisen kielen leikkaus on aina yhteysriippumaton

Kahden yhteysriippumattoman kielen unioni on aina yhteysriippumaton

- varmistetaan, että $V_1 \cap V_2 = \emptyset$ tarvittaessa muuttamalla välisymbolien nimiä
- olkoot kielten alkusymbolit S_1 ja S_2
- lisätään uusi alkusymboli S ja sille säännöt $S ::= S_1 \mid S_2$

Yhteysriippumattoman kielen komplementti ei välttämättä ole yhteysriippumaton

- muutoin leikkaukselle saataisiin kielioppi kaavan $\mathcal{L}(S) \cap \mathcal{L}(T) = \overline{\overline{\mathcal{L}(S)} \cup \overline{\mathcal{L}(T)}}$ mukaisesti
- tämä on harmillista, koska muuten merkkijonon jääminen kielen ulkopuolelle voitaisiin osoittaa näppärästi näyttämällä sen jäsennyspuu kielen komplementissa
- koska asiat ovat niin kuin ovat,
  - merkkijonon kuuluminen yhteysriippumattomaan kieleen on helppo havainnollistaa jäsennyspuun avulla
  - merkkijonon jääminen ulkopuolelle todistetaan ihmisille yleensä vaihtelevin, usein kömpelöin tapauskohtaisin keinoin
- tästä syystä MathCheck ei osaa antaa hyvää palautetta, jos kielioppisi tuottaa merkkijonon, jota se ei saisi tuottaa
- tämä on jonkinasteinen epäsymmetria vastausten "kyllä" ja "ei" välillä

Yhden pinon koneet

- kuvitellaan vahvistettu NFA, joka
  - lukee jonoja, joissa saa olla sekä merkkejä että välisymboleita
  - välisymbolin kohdatessaan se pistää talteen missä oli, ja kutsuu välisymbolia vastaavan vahvistetun NFA:n lukemaan välisymbolin kieleen kuuluvan merkkijonon
  - kun kutsuttu vahvistettu NFA lopettaa, kutsuja jatkaa siitä missä oli

- esimerkki: $S ::= \varepsilon \mid aSa \mid bSb$
- olkoon syötteenä $abba$
- aluksi kone lukee a:n, pistää pinoon että on tässä, ja kutsuu S:ää
- sitten kone lukee b:n, pistää pinoon että on tässä, ja kutsuu S:ää
- sitten kone käyttää sääntöä (S, ε) ja lukematta mitään palaa kutsusta eli nostaa pinosta ylimmän
- sitten kone käyttää syötteen seuraavan merkin (joka on b) bSb:n lukemiseksi loppuun ja nostaa pinosta ylimmän
- sitten kone käyttää syötteen seuraavan merkin (joka on a) aSa:n lukemiseksi loppuun ja nostaa pinosta ylimmän
- nyt koko syöte on luettu ja pino on tyhjä
  $\Rightarrow$ kone hyväksyi merkkijonon $abba$
- tarvittava "missä oli" kirjanpito hoidettiin pinolla
  - pinolla on oma äärellinen aakkostonsa
  - tässä käyttötavassa sen aakkonen kertoo välisymbolin, säännön ja kohdan säännössä, jossa oltiin, kun kutsuttiin vahvistettua NFA:ta
- tällainen kone on nimeltään **yhden pinon kone** (**pushdown automaton**)
- täsmällinen määritelmä sisältää paljon teknisiä yksityiskohtia, jotka eivät ole perusidean kannalta olennaisia
  - jätämme ne käsittelemättä

- jokaiselle yhteysriippumattomalle kielelle on sen hyväksyvä yhden pinon kone
- myös päinvastainen pätee: jokainen yhden pinon koneen hyväksymä kieli voidaan esittää yhteysriippumattomalla kieliopilla
  - jätämme todistuksen esittämättä (se on työläs)

Deterministiset ja epädeterministiset yhden pinon koneet

- olkoon S kuten edellä, mutta nyt syöte onkin $abbbba$
- kun on luettu ab, nyt koneen ei pidäkään käyttää sääntöä (S, ε) vaan lukea b säännöllä (S, bSb), ja vasta sitten käyttää (S, ε) ja alkaa purkaa pinoa
- kun on luettu ab, siihen asti luettu ja seuraavaksi vuorossa oleva merkki eivät riitä kertomaan, kumpaa merkkijonoa luetaan
- deterministinen kone ei selviä tällaisesta tilanteesta
- epädeterminististä konetta tämä ei haittaa, koska sille riittää, että jokin valinta eri vaihtoehdoista johtaa hyväksymiseen
- voidaan todistaa, että sama ongelma vaivaa kaikkia yhteysriippumattomia kielioppeja tälle kielelle

$\Rightarrow$ deterministiset yhden pinon koneen koneet eivät kata kaikkia yhteysriippumattomia kieliä

- tämä on toisin kuin äärellisillä automaateilla
  - jokaiselle NFA:lle on (kenties valtavasti isompi) DFA, joka hyväksyy saman kielen

Yhteysriippumattomien kielten jäsentämisestä

- ohjelmointikielten yms. jäsentämisessä
  - ajan kulutus suhteessa syötteen (= merkkijonon) pituuteen on tärkeä asia
  - jäsentimen muodostamisen vaikeus on toisarvoinen kriteeri, kunhan käytännössä onnistuu lopulta
- tunnetaan jäsennystekniikka, joka
  - toimii merkkijonon pituuden suhteen lineaarisessa ajassa
  - kattaa täsmälleen samat kielet kuin deterministiset yhden pinon koneet

  $\Rightarrow$ tärkeä ohjelmointikielten kääntämisessä
  - hakusana LR parser (tai $LR(1)$ parser)
  - ihmisen on hankala laatia $LR(1)$-jäsennin
  - on olemassa automaatteja sen laatimiseen, kuten `yacc` ja `bison`
- MathCheckin pääasiassa käyttämä jäsennystekniikka kattaa aidosti vähemmän kieliä
  - rekursiivisesti laskeutuva jäsentäminen $LL(1)$-kieliopille
  - toimii merkkijonon pituuden suhteen lineaarisessa ajassa
  - ihmisen on usein helppo laatia rekursiivisesti laskeutuva $LL(1)$-jäsennin
  - $LL(1)$-jäsennys ei selviä ym. sulkujen ongelmasta (logiikka vai aritmetiikka)

  $\Rightarrow$ MathCheck käyttää vippaskonstia `par_expr`
  - olisi *ehkä* ollut parempi käyttää esim. `bison`:ia
  - GCC aloitti `bison`:in tuottamilla LR-jäsentimillä, mutta siirtyi käsin tehtyihin rekursiivisesti laskeutuviin jäsentimiin (C++ 2004, C 2006)

- tunnetaan kaikki yhteysriippumattomat kieliopit kattavia $O(n^3)$ jäsennystekniikoita
  - tässä n on syötteen pituus
  - selvästi liian hidasta pitkille merkkijonoille, kuten ohjelmat
  - kielioppi voi olla tarpeen muuntaa erityiseen muotoon
  - MathCheck käyttää yhteysriippumattomien kielioppien tehtävissä CYK-algoritmin muunnelmaa, joka ei tarvitse kielioppien esikäsittelyä

On vaikeaa selvittää, esittääkö kaksi yhteysriippumatonta kielioppia saman kielen

- ei voi olla olemassa algoritmia, joka ei koskaan erehdy eikä jää ikuiseen silmukkaan
- MathCheck toimii muodostamalla kummankin kielen merkkijonoja lyhyistä alkaen kunnes löytyy ero, kielet valmistuvat tai työmäärälle asetettu yläraja saavutetaan

$\Rightarrow$ MathCheck voi erehtyä

Yhteysriippumattomatkin kielet osoittautuvat hyvin rajallisiksi

- $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ on ulkona, vaikka on helppo tehdä ohjelma, joka tarkastaa, kuuluuko syöte siihen

$\Rightarrow$ seuraavaksi tutkimme, missä ohjelmien rajat kulkevat

- yhteysriippumattomien kielten hyöty tulee siitä, että niitten avulla hallitaan osa kielten määrittelemisen ja jäsentämisen ongelmasta hyvin järjestelmällisesti ja tehokkaasti

4 Laskettavuus ja Turingin koneet

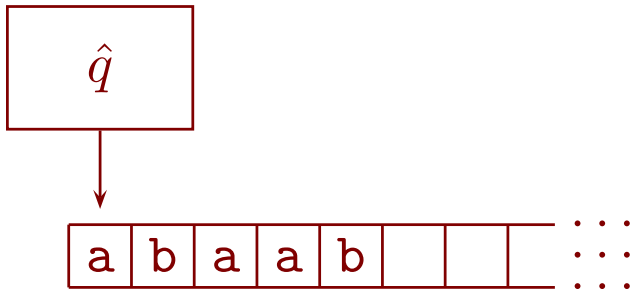
Turingin kone

- Alan Turingin vuonna 1936 keksimä teoreettinen laskennan malli
- käytetään sen tutkimiseen,
  - mitä (osittaisia) funktioita syöte $\mapsto$ tuloste voidaan ja ei voida laskea
  - kuinka nopeasti ne voidaan laskea
- todellisia tietokoneita vahvempi malli sikäli, että Turingin koneen muisti on rajaton
  - ei koskaan lopu kesken
  - voidaan ohjelmoida mm. rajattomat kokonaisluvut
- syöttö- ja tulostustapa ovat hyvin alkeelliset
  $\Rightarrow$ Turingin koneet eivät sovi esim. vuorovaikutuksen tai tietoverkkojen tutkimukseen
- perusoperaatiot ovat hyvin matalalla tasolla
  $\Rightarrow$ Turingin koneiden ohjelmointi on hyvin kömpelöä ja toiminta on hidasta
  - silti Turingin koneille voi ohjelmoida kaiken sen mitä todellisillekin tietokoneille voi ja missä ei haittaa, että syöttö ja tulostus ovat hyvin alkeelliset
  - oikeidenkin tietokoneiden perusoperaatiot ovat matalalla tasolla

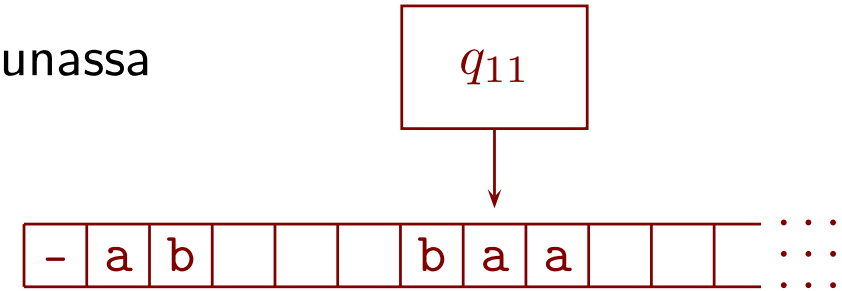
$\Rightarrow$ Turingin koneet ovat kuten todelliset tietokoneet seuraavin eroin:
- muistia on rajattomasti, toisin kuin todellisissa tietokoneissa
- kaikki syöte annetaan laskennan alussa ja kaikki tuloste saadaan vasta lopuksi
- Turingin koneet antavat nopeille algoritmeille huonohkon suoritusajan

Turingin koneen rakenne $(Q, \Sigma, \Gamma, \delta, \hat{q}, \sqcup, F)$

- muistina on oikealle ääretön, ruutuihin jaettu nauha
  - kukin ruutu sisältää yhden aakkosen äärellisestä aakkostosta
- koneella on yksi luku / kirjoituspää, joka on joka hetki jonkin ruudun kohdalla
- aakkostoon Γ kuuluu
  - syöteaakkoston Σ alkiot (nytkin Σ on äärellinen)
  - erikoismerkki $\sqcup$ eli tyhjä, $\sqcup \notin \Sigma$
  - äärellinen joukko muita merkkejä



- aluksi
  - syöte täyttää äärellisen määrän ruutuja nauhan vasemmasta reunasta alkaen
  - syöte ei sisällä $\sqcup$
  - muu nauha on täynnä $\sqcup$
  - luku / kirjoituspää on nauhan vasemmassa reunassa

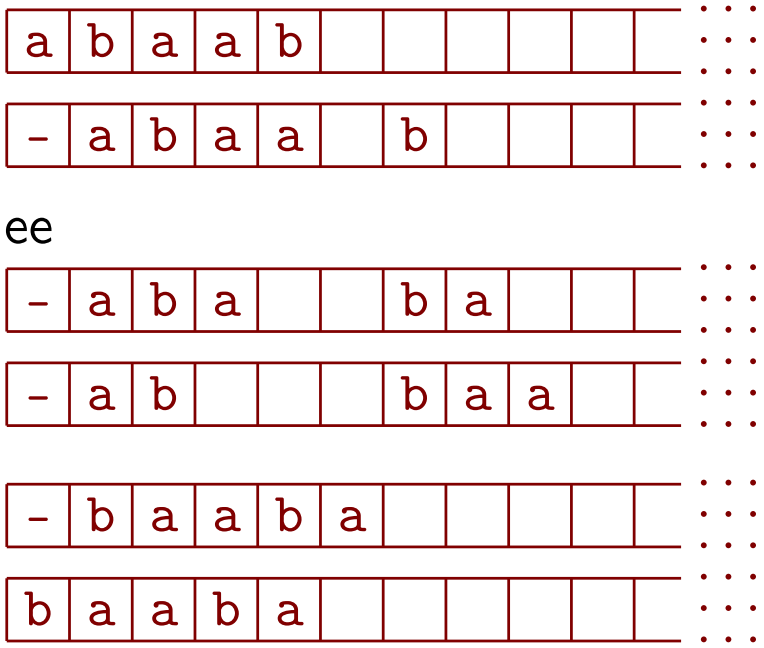


- koneella on ohjausyksikkö, jonka osat ovat
  - äärellinen joukko tiloja Q
  - alkutila $\hat{q} \in Q$
  - lopputilat $F \subseteq Q$
  - tilasiirtymät δ: osittainen funktio $Q \times \Gamma \nrightarrow Q \times \Gamma \times \{<, >\}$

- kone tekee seuraavanlaisia askelia kunnes alla mainittu pysähtymisehto toteutuu
  - katsoo, missä tilassa ollaan ja mitä luku / kirjoituspään alla on
  - siirtyy johonkin tilaan ja kirjoittaa luku / kirjoituspään alle merkin
  - siirtyy yhden ruudun verran vasemmalle < tai oikealle >
- kone pysähtyy jos ja vain jos
  - tilalle ja merkille ei ole määritelty siirtymää, tai
  - luku / kirjoituspää on nauhan vasemmassa reunassa ja siirtymän suunta on <

Esimerkki: merkkijonon $\sigma \in \{\mathtt{a}, \mathtt{b}\}^*$ kääntö takaperin

- jos syöte on tyhjä, lopeta heti siirtymällä lopputilaan ja vasemmalle, muutoin ...
- siirrä syöte pykälän verran oikealle lisäten eteen - ja jättäen viimeisen eteen tyhjän
- toista kunnes vasemmanpuoleinen merkkijono tyhjenee
  - peruuta sen loppuun
  - ota merkki mukaan ja korvaa se ␣:llä
  - kulje oikeanpuoleisen loppuun
  - jätä mukana ollut merkki siihen
- siirrä lopputulos vasemmalle kiinni merkkiin -
- siirrä lopputulos yhden ruudun verran vasemmalle
- - lisättiin, jotta tulos voitaisiin siirtää vasemmalle tuntematon matka



q_1	␣	q_{21}	␣	>	tyhjällä syötteellä lopeta heti
q_1	a	q_2	-	>	kirjoita reunamerkki, ota a mukaan
q_1	b	q_3	-	>	kirjoita reunamerkki, ota b mukaan
q_2	␣	q_4	␣	>	ohita yksi tyhjä a mukana
q_2	a	q_2	a	>	kirjoita a, ota a mukaan
q_2	b	q_3	a	>	kirjoita a, ota b mukaan
q_3	␣	q_5	␣	>	ohita yksi tyhjä b mukana
q_3	a	q_2	b	>	kirjoita b, ota a mukaan
q_3	b	q_3	b	>	kirjoita b, ota b mukaan
q_4	␣	q_6	a	<	kirjoita a käännetyksi merkkijonoksi, lähde paluumatkalle
q_5	␣	q_6	b	<	kirjoita b käännetyksi merkkijonoksi, lähde paluumatkalle
q_6	␣	q_6	␣	<	ohita kääntämättömän ja käännetyn osan välinen tyhjä alue
q_6	-	q_{12}	-	>	koko merkkijono on käännetty, siirrä se vasempaan reunaan
q_6	a	q_7	␣	>	lähde viemään a käännetyn loppuun
q_6	b	q_8	␣	>	lähde viemään b käännetyn loppuun
q_7	␣	q_7	␣	>	ohita tyhjä alue a mukana
q_7	a	q_9	a	>	aloita käännetyn merkkijonon ohittaminen a mukana
q_7	b	q_9	b	>	aloita käännetyn merkkijonon ohittaminen a mukana
q_8	␣	q_8	␣	>	ohita tyhjä alue b mukana
q_8	a	q_{10}	a	>	aloita käännetyn merkkijonon ohittaminen b mukana
q_8	b	q_{10}	b	>	aloita käännetyn merkkijonon ohittaminen b mukana
q_9	␣	q_{11}	a	<	kirjoita a käännetyn merkkijonon loppuun, lähde takaisin
q_9	a	q_9	a	>	ohita käännetty merkkijono a mukana
q_9	b	q_9	b	>	ohita käännetty merkkijono a mukana

q_{10}	␣	q_{11}	b	<	kirjoita b käännetyn merkkijonon loppuun, lähde takaisin
q_{10}	a	q_{10}	a	>	ohita käännetty merkkijono b mukana
q_{10}	b	q_{10}	b	>	ohita käännetty merkkijono b mukana
q_{11}	␣	q_6	␣	<	merkkijonojen välinen tyhjä löytyi, jatka kuten aiemmin
q_{11}	a	q_{11}	a	<	ohita käännetty merkkijono
q_{11}	b	q_{11}	b	<	ohita käännetty merkkijono
q_{12}	␣	q_{12}	␣	>	etsi käännetyn merkkijonon alku
q_{12}	a	q_{13}	␣	>	ota a mukaan, katso onko se viimeinen
q_{12}	b	q_{14}	␣	>	ota b mukaan, katso onko se viimeinen
q_{13}	␣	q_{19}	␣	<	a on käännetyn viimeinen, vie se paikalleen
q_{13}	a	q_{15}	a	<	a ei ole käännetyn viimeinen, vie se paikalleen
q_{13}	b	q_{15}	b	<	a ei ole käännetyn viimeinen, vie se paikalleen
q_{14}	␣	q_{20}	␣	<	b on käännetyn viimeinen, vie se paikalleen
q_{14}	a	q_{16}	a	<	b ei ole käännetyn viimeinen, vie se paikalleen
q_{14}	b	q_{16}	b	<	b ei ole käännetyn viimeinen, vie se paikalleen
q_{15}	␣	q_{15}	␣	<	etsi siirretyn loppu, a mukana, ei viimeinen
q_{15}	-	q_{17}	-	>	siirretyn loppu löytyi, a mukana, ei viimeinen
q_{15}	a	q_{17}	a	>	siirretyn loppu löytyi, a mukana, ei viimeinen
q_{15}	b	q_{17}	b	>	siirretyn loppu löytyi, a mukana, ei viimeinen
q_{16}	␣	q_{16}	␣	<	etsi siirretyn loppu, b mukana, ei viimeinen
q_{16}	-	q_{18}	-	>	siirretyn loppu löytyi, b mukana, ei viimeinen
q_{16}	a	q_{18}	a	>	siirretyn loppu löytyi, b mukana, ei viimeinen
q_{16}	b	q_{18}	b	>	siirretyn loppu löytyi, b mukana, ei viimeinen

q_{17}	␣	q_{12}	a	>	kirjoita a siirretyn jatkeeksi, lähde hakemaan seuraavaa
q_{18}	␣	q_{12}	b	>	kirjoita b siirretyn jatkeeksi, lähde hakemaan seuraavaa
q_{19}	␣	q_{19}	␣	<	etsi siirretyn loppu a mukana, on viimeinen
q_{19}	-	q_{21}	a	>	kirjoita a, mene lopettamaan
q_{19}	a	q_{19}	a	<	kirjoita a, ota a mukaan
q_{19}	b	q_{20}	a	<	kirjoita a, ota b mukaan
q_{20}	␣	q_{20}	␣	<	etsi siirretyn loppu b mukana, on viimeinen
q_{20}	-	q_{21}	b	>	kirjoita b, mene lopettamaan
q_{20}	a	q_{19}	b	<	kirjoita b, ota a mukaan
q_{20}	b	q_{20}	b	<	kirjoita b, ota b mukaan
q_{21}	␣	q_{22}	␣	<	mene vasempaan reunaan ja lopeta
q_{21}	a	q_{22}	a	<	mene vasempaan reunaan ja lopeta
q_{21}	b	q_{22}	b	<	mene vasempaan reunaan ja lopeta

Turingin koneen määritelmän yksityiskohtia voi muunnella laskentakyvyn muuttumatta

- joukon F sijaan riittää yksi lopputila vastausta "kyllä" varten
- voidaan lisätä kolmas siirtymäsuunta: pysyy paikallaan
- voidaan ottaa käyttöön merkki "vasen reuna"
  - se on nauhan alussa eikä muualla
  - sen päälle ei voi kirjoittaa muuta, eikä muun päälle voi kirjoittaa sitä
  - sen kohdalta voidaan siirtyä vain oikealle

Turingin kone ja tietokoneohjelma osittaisten funktioiden laskijana

- Turingin koneen laskeman osittaisen funktion tulos voidaan määritellä esim. nauhan sisältönä alusta ensimmäiseen $\sqcup$ asti se pois lukien, kun kone on pysähtynyt
  - ei piitata siitä, onko pysähtymisen jälkeinen tila lopputila
- Turingin kone voi jäädä ikuiseen silmukkaan
  $\Rightarrow$ funktiolla ei välttämättä ole arvoa jokaisella syötteellä
  $\Rightarrow$ funktio voi olla aidosti osittainen
- funktio on **rekursiivinen**, jos ja vain jos sen voi laskea Turingin koneella
  - se voi olla täysi tai aidosti osittainen
  - nimitys on peräisin kovasti toisennäköisestä määritelmästä, jonka on todistettu määrittelevän täsmälleen samat funktiot
- toisin sanoen, osittainen funktio on rekursiivinen jos ja vain jos sen voi laskea tietokoneohjelmalla
  - tulkitsemme, että muistin loppuminen kesken ei ole ohjelmointikielen vaan tietokoneen ominaisuus
  $\Rightarrow$ "voi laskea tietokoneohjelmalla" on eri väite kuin "voi laskea tietokoneella"
- toiseksi yksinkertaisin syötteestä riippuva tapaus on täydet funktiot, joiden arvo voi olla vain 0 tai 1
- yksinkertaisin on osittaiset funktiot, joiden arvo voi olla vain 0
  $\Rightarrow$ kyse on vain siitä, pysähtyykö kone annetulla syötteellä vai ei pysähdy

Turingin kone ja tietokoneohjelma merkkijonojen ja kielten hyväksyjänä

- Turingin kone **hyväksyy merkkijonon**, jos ja vain jos syötteen ollessa ko. merkkijono se lopulta pysähtyy lopputilaan
  - merkkijono hylätään pysähtymällä epälopputilaan tai jäämällä ikuiseen silmukkaan
- epälopputilaan pysähtyminen on helppo korvata ikuisella silmukalla
  - q_{65} ␣ q_{65} ␣ >, q_{65} - q_{65} - >, q_{65} a q_{65} a > ja q_{65} b q_{65} b >
- Turingin koneen **hyväksymä kieli** on sen hyväksymien merkkijonojen joukko
- kieli on **rekursiivisesti lueteltava** (**recursively enumerable**), jos ja vain jos jokin Turingin kone hyväksyy sen
- toisin sanoen, kieli on rekursiivisesti lueteltava jos ja vain jos on olemassa tietokoneohjelma, joka pysähtyy kielen alkioille ja laskee ikuisesti muille syötteille
- kieli on **rekursiivinen**, jos ja vain jos on olemassa Turingin kone, joka pysähtyy sen alkioille lopputilaan ja muille syötteille epälopputilaan
  - erona edelliseen on, että kaikilla hylätyillä syötteillä täytyy pysähtyä
  - tällainen kone voidaan ohjelmoida pyyhkimään nauha ja tulostamaan 0 tai 1 ennen epäloppu- tai lopputilaan pysähtymistä
  - funktion laskeva kone voidaan ohjelmoida lopuksi katsomaan mitä se tulosti ja sen mukaan pysähtymään epäloppu- tai lopputilaan

  $\Rightarrow$ kieli on rekursiivinen jos ja vain jos sen jäsenyysfunktio on rekursiivinen
- toisin sanoen, kieli on rekursiivinen jos ja vain jos on olemassa tietokoneohjelma joka vastaa kielen alkioille ”kyllä” ja muille syötteille ”ei”, ja sitten pysähtyy

- kieli on rekursiivisesti lueteltava jos ja vain jos se on jonkin rekursiivisen funktion määrittelyjoukko
- määritelmistä seuraa suoraan, että jokainen rekursiivinen kieli on rekursiivisesti lueteltava
- todistamme myöhemmin, että päinvastainen ei päde
- kieli on rekursiivinen, jos ja vain jos se itse ja sen komplementti ovat rekursiivisesti lueteltavia
  - $\Rightarrow$: suoraan määritelmästä, plus vaihdetaan loppu- ja epälopputilat
  - $\Leftarrow$: käynnistetään säikeet selvittämään jäsenyyttä kielessä ja komplementissa
  - niistä tasan yksi tulee lopettamaan lopputilaan
  - kun toinen lopettaa lopputilaan, toinen sammutetaan
  - sitten mennään loppu- tai epälopputilaan sen mukaan, kumpi lopetti lopputilaan
  - jos haluat tietää miten säikeet toteutetaan Turingin koneella, guuglaa dovetailing
- kieli on rekursiivisesti lueteltava, jos ja vain jos on olemassa Turingin kone / tietokoneohjelma, joka luettelee sen alkiot
  - nimitys on peräisin tästä
  - tulostetut alkiot erotetaan toisistaan esim. $\sqcup$:llä
  - $\Rightarrow$: käynnistetään jokaiselle merkkijonolle säie
  - kun säie pysähtyy lopputilaan, tulostetaan sen merkkijono
  - $\Leftarrow$: ajetaan luettelointikonetta kunnes se tulostaa odotetun merkkijonon (voi olla, että se ei tulosta sitä koskaan)
  - jos se pysähtyi sitä ennen, hypätään ikuiseen silmukkaan

- kieli on rekursiivinen, jos ja vain jos on olemassa Turingin kone / tietokoneohjelma, joka luettelee sen alkiot shortlex-järjestyksessä
  - $\Rightarrow$: muodostetaan kaikki merkkijonot shortlex-järjestyksessä
  - jokaisesta testataan jäsenyys kielessä
  - jäsenet tulostetaan
  - $\Leftarrow$: äärellinen kieli on triviaalisti rekursiivinen
  - äärettömän kielen tapauksessa ajetaan luettelointikonetta kunnes se tulostaa odotetun tai sitä isomman merkkijonon
  - odotetun tulostuttua pysähdytään lopputilaan
  - isomman tulostuttua pysähdytään epälopputilaan

$\Rightarrow$ rekursiivisesti lueteltavan mutta ei rekursiivisen kielen alkiot voidaan luetella, mutta vain epäjärjestyksessä!

Universaali Turingin kone

- sille annetaan kaksiosainen syöte: kuvaus Turingin koneesta ja sille tarkoitettu syöte
- laskee sen, mitä kuvattu Turingin kone laskisi annetulla syötteellä
- vastaa sitä, että tietokone voidaan ohjelmoida suorittamaan jotain ohjelmointikieltä tai simuloimaan toista tietokonetta
- tutkijat ovat löytäneet pieniä universaaleja Turingin koneita, mm.
  - Marvin Minsky 1962: 7 tilaa ja 4 aakkosta
  - Yurii Rogozhin 1996: useita, mm. 4 tilaa ja 6 aakkosta sekä 2 tilaa ja 18 aakkosta

Seuraavat koneet hyväksyvät täsmälleen samat kielet kuin Turingin koneet

- Turingin kone molempiin suuntiin äärettömällä nauhalla
- Turingin kone monella molempiin suuntiin äärettömällä nauhalla
  - siirtymäsuunnat eri nauhoilla voi valita vapaasti, m.l. paikallaan pysyminen
  - syöte on alussa nauhalla numero 1
  - lisänauhat voivat nopeuttaa laskentaa, koska tarve kulkea edestakaisin vähenee
  - esim. merkkijonon kääntäminen takaperin lineaarisessa ajassa: kopioi lisänauhalle, peruuta lisänauhan alkuun, kopioi käännetysti takaisin
- epädeterministinen Turingin kone
  - samalla tilalla ja merkillä voi olla vaihtoehtoisia kolmikoita (tila, merkki, suunta)
  - ei tiedetä varmasti, kuinka paljon voi nopeuttaa suoritusta
  - näyttää siltä, että eksponentiaalisesti
- ohjelmointikielet rajattomalla muistilla ja rajattoman lukualueen kokonaisluvuilla
  - Turingin konetta voi simuloida yksinkertaisella ohjelmalla
  - Turingin kone voi simuloida konekieltä, ja kääntäjä hoitaa loput
- kahden pinon kone
  - yhdellä pinolla voidaan simuloida luku / kirjoituspään vasemmalla puolella ja toisella pinolla oikealla puolella olevaa nauhan osaa
  - kahdella nauhalla voidaan simuloida kahta pinoa
  - luvussa 3 näimme, että yhden pinon kone kattaa vain yhteysriippumattomat kielet

  $\Rightarrow$ toinen pino tuo paljon lisää laskentavoimaa, mutta sen jälkeen lisäpinot eivät tuo yhtään

Neljän laskurin koneet hyväksyvät täsmälleen samat kielet kuin Turingin koneet

- laskuri sisältää luonnollisen luvun
- laskuria voi kasvattaa yhdellä, vähentää yhdellä ja testata onko nolla
- syöte on erillisellä nauhalla, jonka päissä on loppumerkit, jonka sisältöä ei voi muuttaa ja jota voi selata molempiin suuntiin
- kahden pinon koneen, jonka pinojen aakkosto on Γ, simuloimiseksi olkoot $k = |\Gamma| + 1$ ja kutakin merkkiä c vastatkoon luku väliltä $\{1, \ldots, |\Gamma|\}$
- pinon sisältö $a_1 \cdots a_n$ esitetään lukuna $\sum_{i=0}^{n-1} luku(a_{i+1})k^i$
- laskurissa c esitetyn pinon ylin merkki selvitetään tyhjän laskurin d avulla kiertämällä k-tilaista silmukkaa kunnes $c = 0$ siten, että joka askeleella vähennetään c:tä ja kasvatetaan d:tä
  - tila c:n tyhjennyttyä vastaa lukua $c \bmod k$
  - c:n alkuperäinen arvo siirtyi d:hen
- merkki a lisätään laskurissa d esitettyyn pinoon tyhjän laskurin c avulla
  - vähentämällä d nollaan siten, että joka vähennyksellä c:tä kasvatetaan k kertaa
  - kasvattamalla c:tä $luku(a)$ kertaa

  $\Rightarrow c$ saa arvon $kd + luku(a) = luku(a)k^0 + \sum_{i=1}^{n} luku(a_i)k^i$
- d:ssä esitetyn pinon ylin merkki poistetaan vähentämällä d nollaan siten, että aina kun k vähennystä tulee täyteen, kasvatetaan c:tä

  $\Rightarrow c$ saa arvon $\lfloor \frac{d}{k} \rfloor = \sum_{i=0}^{n-2} luku(a_{i+2})k^i$
- tolkuttoman hidasta, mutta toimii

Kahden laskurin koneet hyväksyvät täsmälleen samat kielet kuin Turingin koneet

- neljän laskurin koneen laskurit simuloidaan kahdella laskurilla
- neljän laskurin c, d, e ja f sisältö esitetään luvulla $2^c 3^d 5^e 7^f$
- $c \stackrel{?}{=} 0$, $d \stackrel{?}{=} 0$, $e \stackrel{?}{=} 0$ ja $f \stackrel{?}{=} 0$ testataan kopioimalla sisältö tyhjään laskuriin 210-tilaisella silmukalla
  - $210 = 2 \cdot 3 \cdot 5 \cdot 7$
  - $c = 0$ jos ja vain jos jäätiin silmukan tilaan 0, 2, 4, . . . , 208
  - $d = 0$ jos ja vain jos jäätiin silmukan tilaan 0, 3, 6, . . . , 207
  - $e = 0$ jos ja vain jos jäätiin silmukan tilaan 0, 5, 10, . . . , 205
  - $f = 0$ jos ja vain jos jäätiin silmukan tilaan 0, 7, 14, . . . , 203
- laskuria kasvatetaan ja vähennetään kertomalla ja jakamalla 2:lla, 3:lla, 5:llä tai 7:llä

Myös seuraavat hyvin köyhät koneet hyväksyvät samat kielet kuin Turingin koneet

- yksinauhaiset Turingin koneet, joissa on kolme tilaa (aakkostoa ei ole rajoitettu)
- yksinauhaiset Turingin koneet, joiden aakkosto on $\{\sqcup, 0, 1\}$
  - syöteaakkosto on $\{0, 1\}$
  - tämä ei ole olennainen rajoitus, koska isomman aakkoston alkiot voidaan esittää bittijonoina

Kvanttitietokoneet eivät laske edes samaa kuin Turingin koneet

- ne ehkä laskevat joitakin asioita olennaisesti nopeammin kuin Turingin koneet

Rinnakkaiskoneet eivät laske enempää kuin Turingin koneet

- yksi kone voi suorittaa useaa säiettä jakamalla aikaansa niiden kesken
- itse asiassa laskentakapasiteetista saadaan sitä suurempi hyöty, mitä pienemmässä määrässä prosessoreja se on

Koska hyvin laaja skaala koneita laskee täsmälleen saman kuin Turingin koneet, nykyisin uskotaan, että mikään realistinen laskentamalli ei laske enempää kuin Turingin koneet

- tämä väite tunnetaan nimellä Church–Turingin teesi
- se ei ole matemaattinen väittämä, vaan puhuu matematiikan ja tosimaailman välisestä yhteydestä

$\Rightarrow$ sitä ei voida todistaa

- se pitää paikkansa suunnilleen samassa mielessä kuin yleinen suhteellisuusteoria, ei edes suunnilleen samassa mielessä kuin "$\{a^n b^n c^n \mid n \in \mathbb{N}\}$ ei ole yhteysriippumaton"

Tietokoneohjelmia käsittelevät tietokoneohjelmat

- ohjelmointikielen kääntäjä vastaa kyllä/ei-kysymykseen ”noudattaako annettu merkkijono ko. ohjelmointikielen sääntöjä”
  - jos vastaus on ”kyllä”, se myös tuottaa käännöksen

$\Rightarrow$ voi olla mielekästä antaa ohjelmalle syötteeksi ohjelma

- on jopa olemassa ohjelmia, jotka tulostavat oman lähdekoodinsa!
  - jos et usko, katso Wikipedia ”Quine (computing)”

Miljardin suoritusaskeleen pysähtymistesteri

- miljardin suoritusaskeleen pysähtymistesteri on aliohjelma, joka ottaa parametreikseen kaksi merkkijonoa ja palauttaa totuusarvon

totuusarvo pysähtyy_miljardi(merkkijono x, merkkijono y)

- se tulkitsee x:n ohjelmaksi ja y:n ko. ohjelmalle tarkoitetuksi syötteeksi
- se palauttaa vastauksen kysymykseen ”pysähtyykö annettu ohjelma annetulla syötteellä korkeintaan miljardin suoritusaskeleen jälkeen”
  - jos ensimmäinen parametri ei ole ohjelma, se palauttaa false
- sellainen voidaan tehdä
  - tarkastetaan, onko ensimmäinen parametri ohjelma, kuten kääntäjässä
  - jos se on, simuloidaan sitä kunnes se pysähtyy tai miljardi askelta on täynnä
  - tämä ratkaisu kuluttaa jonkin verran enemmän aikaa ja muistia kuin ohjelma itse

  $\Rightarrow$ voiko vastauksen saada nopeammin?

Tarkastellaan seuraavaa ohjelmaa

ilkeily(merkkijono x)
if pysähtyy_miljardi(x, x) **then**
  for $i := 1$ **to** 1 000 000 001 **do**
    print "Hei!"

- mitä tapahtuu, jos sitä kutsutaan niin että sen parametrina on sen oma lähdekoodi?
  - siis kutsutaan ilkeily("**if** pysähtyy_miljardi(x, x) **then** ...")
  - merkitsemme sitä lyhennemerkinnällä ilkeily(ilkeily)
  - aluksi pysähtyy_miljardi(x, x) tutkii, mitä tapahtuu, jos ilkeily:ä kutsutaan niin että sen parametrina on sen oma lähdekoodi
  - jos pysähtyy_miljardi(x, x) vastaa true eli että ilkeily(ilkeily) pysähtyy miljardissa askeleessa, niin ilkeily(ilkeily) tulostaa yli miljardi kertaa "Hei!"

  $\Rightarrow$ vastaus on väärä

  $\Rightarrow$ oikein toimiva pysähtyy_miljardi(x, x) vastaa false
  - jotta vastaus olisi oikea, ilkeily(ilkeily):n suorituksen kokonaisuudessaan tulee käyttää yli miljardi askelta
  - kutsun pysähtyy_miljardi(x, x) ulkopuolella suoritetaan vain muutama askel

  $\Rightarrow$ pysähtyy_miljardi(x, x)*:n suoritukseen kuluu ainakin melkein miljardi askelta*
- simulointiin perustuva pysähtyy_miljardi(x, y) alkaa simuloida itseään kohdatessaan kutsun pysähtyy_miljardi(x, x)
  - syntyy simulaation sisäisiä simulaatioita kunnes miljardi askelta on täynnä

  $\Rightarrow$ vastaus false on oikein

- yllä oleva päättely todistaa, että millä tahansa muullakin periaatteella toimiva pysähtyy_miljardi(x, y) joutuu käyttämään melkein miljardi askelta ainakin yhdellä ohjelmalla x ja syötteellä $y = x$
- kun sanan "miljardi" tilalla on sopivasti valittu syötteen pituuden n funktio ja askelten määrän mittaamisen, simuloinnin yms. käsitteet on tarkennettu, tällä periaatteella voidaan todistaa, että
  - on olemassa tehtäviä, jotka ratkeavat ajassa $O(n^2)$ mutta ei $O(n)$
  - on olemassa tehtäviä, jotka ratkeavat ajassa $O(n^3)$ mutta ei $O(n^2)$
  - ...
  - (itse asiassa hierarkia on tiheämpi kuin tämä)
- "ei ratkea ajassa $O(n^2)$" tarkoittaa, että jos valitaan mielivaltainen $c > 0$, niin on olemassa äärettömän monta syötettä joille aika cn^2 ei riitä
  - se ei tarkoita, että millään syötteellä aika cn^2 ei riitä
  - ei välttämättä edes, että useimmilla syötteillä aika cn^2 ei riitä

Tämä ja muut samankaltaiset tulokset tarkoittavat käytännössä, että ohjelman käyttäytymistä ei välttämättä voi ennustaa olennaisesti tehokkaammin kuin suorittamalla ohjelma ja katsomalla mitä se tekee

- tämän voi tulkita sanomalla, että ohjelman käyttäytyminen voi olla ennustamatonta
  - ei kiellä käyttäytymisen saamista selville muuten kuin kokeilemalla
  - estää muita tapoja olemasta merkittävästi käyttökelpoisempia

Miljoonan tavun pysähtymistesteri

- palauttaa vastauksen kysymykseen ”pysähtyykö annettu ohjelma annetulla syötteellä käytettyään korkeintaan miljoona tavua muistia”
- käytetyksi muistiksi lasketaan myös tietokoneen rekisterit yms.
- viimeistään $256^{1\,000\,000} \approx 10^{2\,408\,240}$ askeleen jälkeen ohjelma on joko
  - lopettanut,
  - ottanut käyttöön yli miljoona tavua muistia tai
  - palannut tilaan jossa se on ollut aikaisemmin eli on ikuisessa silmukassa

$\Rightarrow$ miljoonan tavun pysähtymistesteri voidaan periaatteessa toteuttaa laskemalla suoritusaskelia miljoonatavuisella laskurilla
  - maailmankaikkeuden ikä on noin 10^{15} sekuntia
  - lyhyin fysiikan tuntema aika on suuruusluokkaa 10^{-44} sekuntia
  - sillä kellojaksolla simulointiin tarvittaisiin noin $10^{2\,408\,181}$ maailmankaikkeuden ikää

  $\Rightarrow$ ei todellakaan onnistu käytännössä lähitulevaisuudenkaan tietokoneilla

- ohjelman ilkeily periaatteella ja korvaamalla käsitteet täsmällisemmillä voidaan osoittaa, että
  - on olemassa tehtäviä, jotka ratkeavat muistissa $O(n^2)$ mutta ei $O(n)$
  - on olemassa tehtäviä, jotka ratkeavat muistissa $O(n^3)$ mutta ei $O(n^2)$
  - ...
  - (itse asiassa hierarkia on paljon tiheämpi kuin tämä)

Pysähtymistesteri

totuusarvo pysähtyy(merkkijono x, merkkijono y)

- vastaa kysymykseen, onko x ohjelma ja jos kyllä pysähtyykö se syötteellä y
- tarkastellaan seuraavaa ohjelmaa

  ilkeily(merkkijono x)
  if pysähtyy(x, x) **then**
  while true **do** print "Hei!"

- ilkeily(ilkeily) ennustaa tuleeko ilkeily(ilkeily) pysähtymään, ja tekee päinvastoin kuin ennuste sanoo

$\Rightarrow$ mitä tahansa pysähtyy(ilkeily, ilkeily) vastaakin, se on väärin

$\Rightarrow$ jokainen pysähtymistesteriksi tarkoitettu aliohjelma epäonnistuu ainakin tässä tilanteessa

$\Rightarrow$ pysähtymistesteriä ei voida tehdä — Alan Turing 1936

- tämä taustakäsitteineen on yksi ihmiskunnan suurimpia tieteellisiä keksintöjä viimeksi kuluneen 100 vuoden aikana, vaikka ei ole tämän vaikeampi todistaa!
- tulos voidaan helposti laajentaa koskemaan äärettömän monta syötettä
- käytännössä tämä tarkoittaa, että ohjelmasta ei välttämättä voi koskaan saada tietää, pysähtyykö se
  - jos sitä suoritetaan niin kauan että se pysähtyy, niin vastaus saatiin
  - jos se ei pysähdy sinä aikana joka sitä jaksetaan suorittaa, vastaus jää avoimeksi

Merkkijonojen tuottaminen syötteettömällä pysähtyvällä ohjelmalla

- print "Hei!" tulostaa itseään lyhyemmän merkkijonon
- **for** $i := 1$ **to** $1\,000\,000$ **do** print "Hei!" tulostaa itseään pitemmän merkkijonon

Pisin merkkijono, jonka voi tuottaa syötteettömällä pysähtyvällä ohjelmalla, jonka pituus on enintään n

- olkoon $n \in \mathbb{N}$ ja $\Sigma =$ tavut
- merkkijonoja, joiden pituus on enintään n, on $1 + 256 + 256^2 + \ldots + 256^n$ kpl
- syötteettömiä pysähtyviä ohjelmia pituudeltaan $\leq n$ on korkeintaan sama määrä

$\Rightarrow$ ne tulostavat vain äärellisen määrän merkkijonoja

$\Rightarrow$ niiden tulostamien merkkijonojen joukko on tyhjä tai siinä on ≥ 1 pisin

$\Rightarrow$ olkoon maxpit(n) funktio, joka palauttaa luvun m siten, että
  - jos syötteettömiä pysähtyviä ohjelmia pituudeltaan enintään n ei ole, niin $m = 0$
  - muutoin m on suurin sellaisen ohjelman tulostaman merkkijonon pituus

- maxpit(n) on matemaattisesti hyvin määritelty, mutta voiko sen laskea tietokoneella?
- johtaaksemme ristiriidan oletamme että voi ja tarkastelemme ohjelmia

  O_0: **for** $i := 0$ **to** maxpit(1) **do** print "a"
  O_1: **for** $i := 0$ **to** maxpit(10) **do** print "a"
  O_2: **for** $i := 0$ **to** maxpit(100) **do** print "a"
  O_3: **for** $i := 0$ **to** maxpit(1000) **do** print "a"
  ...

- jokainen O_j on syötteetön ja pysähtyvä
- olkoon k aliohjelman maxpit(n) määritelmän pituus
- O_j:n pituus on $k + j + 38$
- O_j tulostaa merkkijonon, jonka pituus on maxpit$(10^j) + 1$
- jos j on tarpeeksi iso, niin $k + j + 38 \leq 10^j$
  - Bernoullin epäyhtälön mukaan $(1 + 9)^j \geq 1 + 9j$, kun $j \in \mathbb{N}$

  $\Rightarrow$ jos $j \geq \frac{1}{8}(k + 37)$, niin $8j + 1 + j \geq k + 37 + 1 + j = k + j + 38$, joten $10^j = (1 + 9)^j \geq 1 + 9j = 8j + 1 + j \geq k + j + 38$

$\Rightarrow$ jos $j \geq \frac{1}{8}(k + 37)$, niin O_j:n pituus on $\leq 10^j$, mutta silti O_j tulostaa pitemmän merkkijonon kuin mikään enintään niin pitkä syötteetön pysähtyvä ohjelma!

$\Rightarrow$ maxpit(n) ei voi laskea tietokoneella

Todistus sille, että syötteettömien ohjelmien pysähtymistesteriä ei voida tehdä

- jos taulukko M sisältää merkkijonon pituudeltaan i, niin seuraava merkkijono voidaan muodostaa seuraavasti:
  - tässä "A" edustaa merkistön ensimmäistä ja "ö" viimeistä alkiota

```
j := i
while j > 0 && M[j] = "ö" do
   M[j] := "A"; j := j − 1
if j > 0 then M[j] := M[j] + 1
else i := i + 1; M[i] := "A"
```

- pysähtymistesterin avulla voitaisiin laskea maxpit(n) tietokoneella seuraavasti:
  - muodostetaan vuoronperään kaikki merkkijonot pituudeltaan enintään n
  - testataan, onko vuorossa oleva merkkijono syötteetön ohjelma
  - jos kyllä, niin testataan, pysähtyykö se
  - jos kyllä, niin ajetaan se ja katsotaan, kuinka pitkän merkkijonon se tulostaa
  - pidetään kirjaa suurimmasta havaitusta pituudesta

$\Rightarrow$ koska maxpit(n) ei voida laskea tietokoneella, syötteettömien ohjelmien pysähtymistestiä ei voida laskea tietokoneella

Rekursiiviset funktiot ja joukot uudelleen

- funktiota sanotaan *rekursiiviseksi* jos ja vain jos sen voi laskea tietokoneohjelmalla
  - siis tietokoneella olettaen, että aika ja muisti eivät lopu kesken
- koulumatematiikasta tutut funktiot kokonaisluvuilta kokonaisluvuille ovat rekursiivisia
- meillä on jo kolme esimerkkiä ei-rekursiivisista funktioista
  - pysähtyy(x, y)
  - syötteetön_pysähtyy(x)
  - maxpit(n)
- joukkoa A sanotaan rekursiiviseksi, jos ja vain jos testin $x \in A$ voi laskea tietokoneella
- $\{(x, y) \mid \text{pysähtyy}(x, y)\}$ ei ole rekursiivinen
- $\{x \mid \text{syötteetön\_pysähtyy}(x)\}$ ei ole rekursiivinen

Rekursiivisesti lueteltavat joukot uudelleen

- joukko on *rekursiivisesti lueteltava* (*recursively enumerable*), jos ja vain jos on olemassa tietokoneohjelma, joka
  - pysähtyy lopulta, jos syöte on joukon alkio
  - laskee ikuisesti, jos syöte ei ole joukon alkio
- A on rekursiivinen jos ja vain jos sekä A että A:n komplementti ovat rekursiivisesti lueteltavia
- $\{x \mid \text{syötteetön\_pysähtyy}(x)\}$ on rekursiivisesti lueteltava
  - jos x ei ole ohjelma tai ei ole syötteetön, aloitetaan ikuinen silmukka
  - muussa tapauksessa ajetaan x
- $\{x \mid \neg\ \text{syötteetön\_pysähtyy}(x)\}$ ei ole rekursiivisesti lueteltava
  - muutoin $\{x \mid \text{syötteetön\_pysähtyy}(x)\}$ olisi rekursiivinen

$\Rightarrow$ rekursiivisuus on symmetrinen vastausten kuuluu/ei kuulu joukkoon suhteen, mutta rekursiivinen lueteltavuus ei ole

Mikä on tällaisten tulosten käytännön merkitys?

- tulosten viesti on, että ohjelmien käyttäytyminen voi olla mahdotonta ennustaa
- jotkin ilmiöt ilmenevät vain niin pitkissä laskennoissa, että niillä ei ole käytännön merkitystä
  - vrt. $10^{2\,408\,181}$ maailmankaikkeuden ikää
- todistus, että pysähtymistesteriä ei ole, ei takaa, etteikö voisi olla melkein täydellistä pysähtymistesteriä joka epäonnistuu niin harvoin, että siitä ei tarvitse välittää
- käytännössä ohjelmien käyttäytymistä on vaikea ennustaa
  - usein ohjelma voidaan laatia ja sen toimintaperiaate dokumentoida siten, että on hyvin varmaa, että ohjelma toimii kuten pitääkin
  - se vaatii kuitenkin huolellisuutta ja korkeaa ammattitaitoa
  - ohjelmilla, joita ei ole suunniteltu miettimällä huolellisesti sekä toimintaperiaate että yksityiskohdat, on taipumus toimia joissain tilanteissa väärin

  $\Rightarrow$ ohjelmointitapa "tehdään muutoksia, kunnes testeissä ei enää paljastu vikoja" ei tuota luotettavia ohjelmia
- näillä tuloksilla on teoriassa vain teoreettista merkitystä, mutta käytännössä ne kertovat oikein hyvin mitä tapahtuu käytännössä!
- rastita oikeat vaihtoehdot, miksi et saa ohjelmaasi toimimaan kunnolla?

  ☐ olet tyhmä
  ☒ ohjelmointikielet ja työkalut ovat huonoja
  ☒ saamasi koulutus ei ole ollut niin hyvää kuin voisi toivoa
  ☒ ohjelmointi on aidosti vaikeaa syvällisistä syistä

5 Lopuksi

Chomskyn hierarkia

- säännölliset kielet
  - hyväksyminen: äärellinen automaatti
  - kielioppi: oikealle lineaarinen (right-linear): kaikki sääntöjen oikeat puolet ovat muotoa σB tai σ, missä σ ei sisällä välisymboleita
  - siis säännöt ovat muotoa $A \to \sigma B$ tai $A \to \sigma$
  - vasemmalle lineaarinen (left-linear) kelpaa yhtä hyvin: $A \to B\sigma$ tai $A \to \sigma$
  - kielioppi on säännöllinen, jos ja vain jos se on vasemmalle tai oikealle lineaarinen
- yhteysriippumattomat kielet
  - hyväksyminen: epädeterministinen yhden pinon kone
  - kielioppi: yhteysriippumattomat kieliopit (BNF eli Backus–Naur Format on käytännössä sama asia)
  - siis säännöt ovat muotoa $A \to \sigma$, missä σ saa sisältää väli- ja loppusymboleita
- yhteysriippuvat kielet (context-sensitive)
  - hyväksyminen: lineaarisesti rajoitettu automaatti (linear bounded automaton): epädeterministinen Turingin kone, joka saa käyttää muistia vain lineaarisesti suhteessa syötteen pituuteen
  - kielioppi: kuten rekursiivisesti lueteltavilla kielillä, mutta $|\sigma| \geq |\rho|$
  - vaihtoehtoinen kielioppi: säännöt muotoa $\alpha A \beta \to \alpha \sigma \beta$, missä $\sigma \neq \varepsilon$
  - suuri joukko kieliä, mutta silti vähemmän kuin rekursiiviset

- rekursiivisesti lueteltavat kielet
  - hyväksyminen: Turingin kone
  - kielioppi: säännöt muotoa $\rho \to \sigma$, missä $\rho \neq \varepsilon$ ja ρ saa sisältää sekä väli- että loppusymboleita
  - sääntöä saa soveltaa jonon sisällä, ts. $\alpha\rho\beta \to \alpha\sigma\beta$

Ohjelmointikielten nelitasomalli

- leksikaalitaso
- varsinainen syntaksi

} syntaksi

- staattinen semantiikka
- dynaaminen semantiikka

} semantiikka

Leksikaalitaso

- merkkien ryhmittyminen tekstialkioiksi (token)
  - varatut sanat: esim. `while`, `int`
  - ohjelmoijan määrittelemät nimet: esim. `x`, `oma_tyyppi`
  - literaalivakiot: esim. `3.14159`, `"tämä on merkkijono"`
  - merkeillä esitetyt: esim. `=`, `<=`, `++`
- tyypillisesti tekstialkioiden sisällä ei saa ja välissä saa olla ylimääräistä tyhjää
  - tyhjä tila = välilyönnit, rivinsiirrot, sarkaimet eli tabulaattorit ja kommentit
  - kuitenkin merkkijonoissa voi olla tyhjää tilaa, mutta siellä sillä on merkitys
  - esim. `while ( i > 0 )` ja `while(i>0)` kelpaavat, `wh ile ( i > 0 )` ei kelpaa

- jos tekstialkio voi olla toisen alkuosa, niin tyypillisesti valitaan pisin mahdollinen
- tunnistetaan DFA:lla (ainakin jos minimoidaan käännösaikaa)
- määritellään usein BNF:llä

Varsinainen syntaksi

- tekstialkioiden järjestystä koskevat muotoseikat
  - esim. jokaisella alkusululla `(` täytyy olla vastaava loppusulku `)`
  - esim. `int while < i 0` ei kelpaa kääntäjälle
- määritellään yleensä BNF:llä tai yhteysriippumattomalla kieliopilla
- jäsennetään usein rekursiivisesti laskeutuvalla tai LR-jäsentimellä
- tärkeä taso, koska
  - voidaan jäsentää tehokkaasti
  - voidaan ilmaista monimutkaisia hierarkkisia ihmisille luontevia rakenteita

Staattinen semantiikka

- muut käännösaikana tarkastettavat asiat kuin syntaksi
  - esim. muuttujat on määriteltävä ennen käyttöä $\rightsquigarrow$ symbolitaulu
  - esim. tyyppitarkastukset
- ei vastaa mitään Chomskyn hierarkian tasoa

Dynaaminen semantiikka

- ajoaikana tai ei edes silloin tarkastettavat asiat
  - esim. nollalla jako, `int *p = 0; p* = 3;`
  - esim. `A[i++] = i;`
- ohjelman merkitys
  - mitä se laskee
  - miten se sen laskee

Kurssiin ei mahtunut

- lisää ratkeamattomuustuloksia
  - kielet, jotka eivät ole rekursiivisesti lueteltavia eikä niiden komplementteja
  - Ricen lause: jokainen Turingin koneen hyväksymän kielen epätriviaali ominaisuus on ratkeamaton
  - ratkeamattomuustuloksia eri aloilta
- kompleksisuusteoria
  - kuinka nopeasti kieliä voi tunnistaa (ja rekursiivisia funktioita laskea)
  - kuinka vähällä muistilla kieliä voi tunnistaa (ja rekursiivisia funktioita laskea)
  - esim. **NP**-täydellisyys ja **PSPACE**-täydellisyys
- kaikkea tätä on tutkittu ahkerasti vuosikymmenet

$\Rightarrow$ tuloksia on valtavasti, ja joitakin käytännössäkin hyödyllisiä

Kiitos mielenkiinnosta, tsemppiä tenttiin!