

582206 Laskennan mallit

luennot syksyllä 2006, periodit I–II

Jyrki Kivinen

- tietojenkäsittelytieteen aineopintokurssi, 6 op, pääaineopiskelijoille pakollinen
- esitietoina *Tietorakenteet* (ja sen esitiedot)
- mahdollinen jatkokurssi *Laskennan vaativuus* (valinnainen aineopintokurssi, 4 op, periodi IV)

Annettava opetus, kurssin suorittaminen

- luentoja 2 tuntia viikossa
- harjoituksia 2 tuntia viikossa alkaen 11.9.; tarkemmin seuraavilla kalvoilla
- kaksi kurssikoetta (kummankin periodin tenttiviikolla)
- kurssikirja Sipser: *Introduction to the Theory of Computation*, suunnilleen luvut 1–5.
- (ei luentoja eikä harjoituksia tenttiviikoilla eikä väliviikolla)
- **Huom.** kurssin arvioitu työmäärä on 160 tuntia (eli syksyn aikana keskimäärin yli 10 tuntia viikossa)
- maksimipisteet harjoituksista $6 + 2$ (ks. seur. sivu), tenteistä $26 + 26$, yhteensä 60; hyväksymisraja n. 30, arvosanan 5/5 raja n. 51

Laskuharjoitukset

Paritonnumeroisilla harjoituskerroilla (viikosta 37 alkaen joka toinen kerta) käsitellään **kotitehtäviä**:

- laitoksen ”perinteinen” harjoitusmuoto: opiskelijat ratkaisevat tehtävät etukäteen, laskaritulaisuudessa ratkaisut kirjataan kalvolle ja esitetään ryhmätyönä
- 4 tehtävää per viikko, yhteensä siis 24 tehtävää
- kotitehtävien tekeminen on **pakollista**: ainakin 6 tehtävää vaaditaan
- tehdyistä kotitehtävistä saatavat pisteet (max. 6):

tehtäviä	alle 6	6	8	10	13	15	17	20
pisteitä	hylätty	0	1	2	3	4	5	6

Parillisnumeroisilla harjoituskerroilla (viikosta 38 alkaen joka toinen kerta) käsitellään **perustehtäviä** ja **yhteistehtäviä**:

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

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

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

(Laskarisysteemistä, kuten kurssista muutenkin, otetaan mielellään palautetta vastaan.)

Sisältö

0. Johdanto: yleiskatsaus, esitietojen kertaus
1. Säännölliset kielet: äärelliset automaattit, säännölliset lausekkeet
2. Yhteydettömät kielet: pinoautomaattit, yhteydettömät kieliopit
3. Ratkeavuus: Turingin koneet, ratkeavat ja ratkeamattomat ongelmat

Kurssin peruskysymykset ovat

- mitkä periaatteelliset seikat rajoittavat automaattista laskentaa (ja siis erityisesti nykyisiä tietokoneita) ja
- miten tällaiset asiat voidaan esittää täsmällisesti.

Näihin ei toki anneta mitään kovin yksiselitteisiä vastauksia.

Tavoitteet

Kurssin jälkeen opiskelija

- * tuntee automaattien ja kielioppien tärkeimmät luokat ja niiden yhteydet,
- + osaa muodostaa automaatteja ja kielioppeja annetuille yksinkertaisille kielille,
- + osaa soveltaa kurssilla esitettyjä muunnoksia automaatti- ja kielioppiformalismien välillä,
- * ymmärtää Churchin-Turingin teesin ja sen seuraukset,
- * tuntee keskeisimmät ratkeamattomuustulokset ja niiden seuraukset,
- + osaa laatia yksinkertaisia ratkeamattomuustodistuksia ja
- × osaa formalisoida laskentaan liittyviä ongelmia ja esittää formalismeihin perustuvia täsmällisiä argumentteja.

0. Johdanto

Erästä suosittua määritelmää mukailien tietojenkäsittelytiede tutkii,

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

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

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

Laskettavuuden teoria

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

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

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

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

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

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

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

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

Laskennan vaativuus

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

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

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

(Tällä kurssilla ei juuri käsitellä laskennan vaativuuskysemyksiä.)

Automaattiteoria

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

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

Yhteydettömät kieliopit ovat hieman äärellisiä automaatteja ilmaisuvoimaisempi mekanismi, jolla on tärkeitä sovelluksia esim. ohjelmointikielissä.

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

Matemaattisia perustietoja [Sipser luku 0.2]

Logiikan, joukko-opin ja verkkojen peruskäsitteet edellytetään tutuiksi kursseilta *Johdatus diskreettiin matematiikkaan* ja *Tietorakenteet*. Näistä asioista tulisi muistaa ainakin sen verran, että kurssikirjan *Exercises 0.1–0.9* (s. 25–26) eivät aiheuta vaikeuksia. Kurssilla tehdään jonkin verran induktiotodistuksia, mutta ne eivät ole keskeisessä roolissa.

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

Esimerkkejä tyypillisistä aakkostoista:

$$\Sigma_1 = \{0, 1\}$$

$$\Sigma_2 = \{a, b, c, \dots, z\}$$

$$\Gamma_1 = \{A, C, G, T\}$$

$$\Gamma_2 = \{ \text{HiiriVasen}, \text{HiiriKeski}, \text{HiiriOikea}, \\ \text{ShiftHiiriVasen}, \text{ShiftHiiriKeski}, \text{ShiftHiiriOikea} \}$$

$$\Gamma_3 = \{ \text{Renault}, \text{Ferrari}, \text{McLaren}, \text{muu} \}$$

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

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

Merkkijonon $w = w_1w_2 \dots w_n$ **käänteismerkkijono** on $w^R = w_nw_{n-1} \dots w_1$. Merkkijono v on merkkijonon w **osajono**, jos v esiintyy *yhtenäisenä* osana merkkijonossa w . Siis acad on merkkijonon abracadabra osajono, mutta rcd ei ole. Merkkijonojen $v = v_1 \dots v_n$ ja $w = w_1 \dots w_m$ **konkatenaatio** on $vw = v_1 \dots v_nw_1 \dots w_m$. Siis v on merkkijonon w osajono, jos $w = xvy$ joillakin merkkijonoilla x ja y . Merkkijonon w konkatenaatiota itsensä kanssa k kertaa merkitään w^k . Siis $|w^k| = k|w|$.

Leksikografisessa järjestyksessä merkkijonot ovat pituuden mukaan kasvavassa järjestyksessä, ja samanpituiset jonot aakkosjärjestyksessä. Siis **binääriaakkoston** Σ_1 merkkijonojen leksikografinen järjestys on

$\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \dots$

Mitä tahansa joukkoa annetun aakkoston merkkijonoja sanotaan (**formaaliksi**) **kieleksi**.

1. Säännölliset kielet

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

Tämän luvun jälkeen opiskelija

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

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

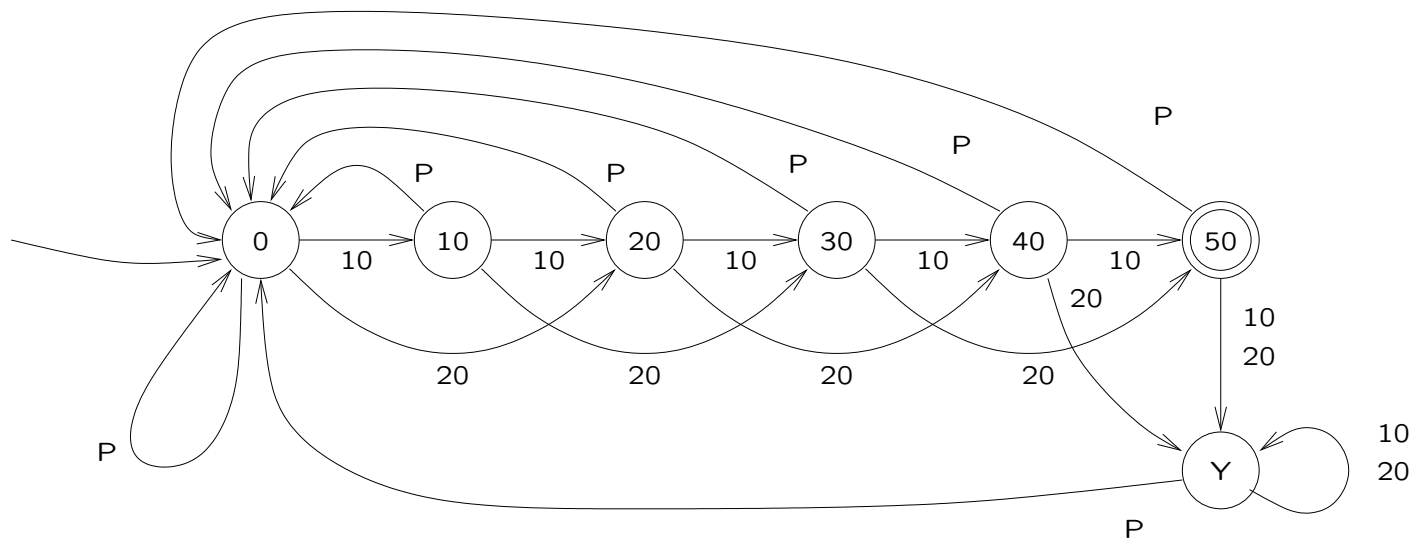
Havainnollistamme äärellistä automaattia tilanteessa, jossa kahviautomaatista voi ostaa kahvin 50 sentillä.

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

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

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

Saamme seuraavanlaisen äärellisen automaatin:



Automaatilla on

tiloja (7 kappaletta), jotka on esitetty ympyröinä ja nimetty 0, 10, 20, 30, 40, 50 ja Y;

siirtymiä, jotka on esitetty tilojen välisinä kaarina;

aakkosto, jonka symboleilla siirtymät on merkitty;

alkutila (tila 0), joka on merkitty tyhjästä tulevalla kaarella; ja

hyväksyvä tila (tila 50), joka on rengastettu.

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

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

Äärellinen automaatti: formaali määritelmä [Sipser s. 35–43]

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

1. Q on äärellinen tilajoukko; sen alkioita sanotaan tiloiksi,
2. Σ on äärellinen aakkosto,
3. $\delta: Q \times \Sigma \rightarrow Q$ on siirtymäfunktio,
4. $q_0 \in Q$ on alkutila ja
5. $F \subseteq Q$ on hyväksyvien tilojen joukko (joita toisinaan kutsutaan myös lopputiloiksi).

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

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

1. $Q = \{q_0, q_1, q_2\}$,

2. $\Sigma = \{0, 1\}$,

3. δ saadaan taulukosta

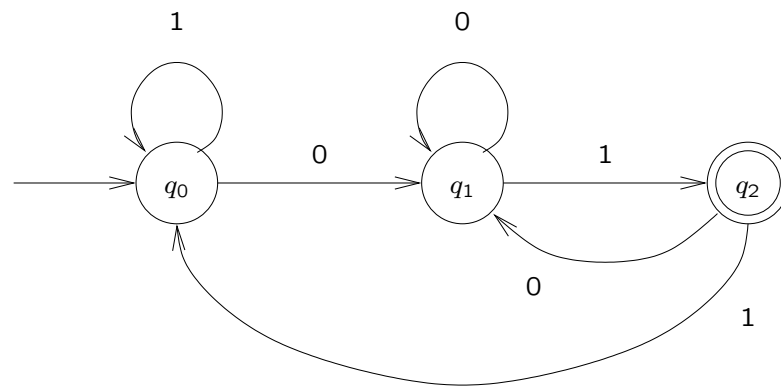
δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

(siis esim. $\delta(q_1, 0) = q_1$ ja $\delta(q_1, 1) = q_2$),

4. alkutila on q_0 ja

5. $F = \{q_2\}$.

Edellisen sivun automaatti M_1 voidaan esittää havainnollisemmin tilakaaviona



Automaatin toiminta syötteellä 1001101 voidaan kuvata siirtymäjonona

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

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

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

Formalisoimme nyt edellä kuvatun laskennan.

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

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

Siis laskenta alkaa alkutilasta, noudatta syötemerkkien mukaisia siirtymiä ja päättyy hyväksyvään tilaan.

Jos M ei hyväksy, se hylkää merkkijonon w .

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

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

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

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

Äärelliseen automaattiin voidaan helposti lisätä muutakin tulostusta kuin pelkkä hyväksyminen tai hylkääminen. Tämä ei kuitenkaan tämän kurssin kannalta toisi mitään oleellista uutta asiaan.

Hahmon etsiminen syötteestä (johdatteleva esimerkki)

Unix-komennolla

```
grep hahmo [ tiedosto ]
```

voidaan etsiä hahmon esiintymiä tiedostosta (tai syötevirrasta):

```
$ grep Kisaveikot SM-tulokset.txt  
$ ps aux | grep mmeikala  
$ ps aux | grep '\(mmeikala\|tteikala\)'
```

Tässä siis haetaan

- SM-tuloksista rivit, joilla esiintyy seura Kisaveikot
- prosessilistasta ne rivit, joilla esiintyy käyttäjätunnus mmeikala, ja
- prosessilistasta ne rivit, joilla esiintyy käyttäjätunnus mmeikala tai tteikala.

Eräs idea `grep`-toiminnon toteuttamiseksi olisi seuraava:

1. Muodostetaan äärellinen automaatti, joka hyväksyy tasan sellaiset merkkijonot, joissa esiintyy *hahmo*.
2. **Selataan** syöte rivi kerrallaan käyttämällä tätä automaattia, ja tulostetaan hyväksytyt rivit.

Selausvaihe toimii nopeasti eikä edellytä syötteen esiprosessointia, mikä on tässä tärkeää.

Kysymys: Kuinka monimutkaisia hahmoja tällä periaatteella voidaan käsitellä?

Esim. edellä muodostettiin hahmoista `mmeikala` ja `tteikala` uusi hahmo tai-operaattorilla `" | "`. Kuinka voimakkaat operaattorit voidaan siis sallia?

Operaatiot (säännöllisillä) kielillä [Sipser s. 44–47]

Kuten muistetaan, kielet ovat merkkijonojoukkoja.

Niillä voidaan siis suorittaa normaaleja joukko-opillisia operaatioita, kuten yhdiste, leikkaus ja komplementointi: jos A ja B ovat kieliä, niin

- kieli $A \cup B$ koostuu niistä merkkijonoista, jotka kuuluvat ainakin toiseen kielistä A ja B ,
- kieli $A \cap B$ koostuu niistä merkkijonoista, jotka kuuluvat sekä kieleen A että B , ja
- kieli \overline{A} koostuu niistä merkkijonoista, jotka eivät kuulu kieleen A .

(Komplementin \overline{A} määrittelyssä oletetaan, että puhutaan jonkin tietyn aakkoston Σ merkkijonoista, ja Σ selviää asiayhteydestä.)

Nimenomaan merkkijonjoukoille on kätevää määritellä myös konkatenaatio ja tähti:

- kieli $A \circ B$ koostuu merkkijonoista w , jotka voidaan esittää muodossa $w = xy$ joillakin $x \in A$ ja $y \in B$ ja
- kieli A^* koostuu merkkijonoista $w_1 \dots w_k$, missä $k \geq 0$ ja $w_i \in A$ kaikilla i .

Intuitiivisesti

$$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \dots,$$

missä A^k on kielen A konkatenaatio itsensä kanssa k kertaa (ja $A^0 = \{\varepsilon\}$).

Esimerkki Tarkastellaan aakkoston $\{a, \dots, z, 0, \dots, 9\}$ kieliä $A = \{aa, bb\}$ ja $B = \{01, 02\}$. Nyt

$$A \cup B = \{aa, bb, 01, 02\}$$

$$A \circ B = \{aa01, aa02, bb01, bb02\}$$

$$A^* = \{\varepsilon, aa, bb, aaaa, aabb, bbaa, bbbb, \\ aaaaaa, aaaabb, aabbaa, aabbbb, bbaaaa, \dots\}.$$



Seuraavana tavoitteenamme on osoittaa, että säännöllisten kielten luokka on **suljettu** operaatioiden \cup , \circ ja $*$ suhteen. Toisin sanoen jos A ja B ovat säännöllisiä, niin myös $A \cup B$, $A \circ B$ ja A^* ovat. Tapaus $A \cup B$ on melko suoraviivainen, ja aloitamme siitä.

Lause 1.1: [Sipser Thm. 1.25] Jos kielet A ja B ovat säännöllisiä, niin myös $A \cup B$ on.

Todistus: Oletetaan siis, että $A = L(M_1)$ ja $B = L(M_2)$ äärellisillä automaateilla $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ ja $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$. Muodostamme äärellisen automaatin M , jolla $L(M) = A \cup B$.

Automaatin M pitää siis hyväksyä w , jos ainakin toinen automaateista M_1 ja M_2 hyväksyy. Ongelmaa ei voi ratkaista simuloimalla automaatteja M_1 ja M_2 peräjälkeen, sillä äärellinen automaatti käsittelee kunkin syötemerkin vain kerran.

Ratkaisu on simuloida kumpaakin automaattia **samanaikaisesti**. Jos $|Q_1| = m$ ja $|Q_2| = n$, niin automaatin M tilajoukoksi valitaan $m \times n$ taulukko, jossa rivi esittää M_1 :n ja sarake M_2 :n tilaa. Vastaavasti δ_1 kertoo, mille riville mennään seuraavaksi, ja δ_2 mille sarakkeelle.

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

- $Q = Q_1 \times Q_2$,
- aakkosto Σ pysyy samana,
- kun $r = (r_1, r_2) \in Q_1 \times Q_2$ ja $a \in \Sigma$, niin $\delta(r, a) = (s_1, s_2)$ missä
$$s_1 = \delta_1(r_1, a) \quad \text{ja} \quad s_2 = \delta_2(r_2, a),$$
- $q_0 = (q_1, q_2)$ ja
- $F = \{ (r_1, r_2) \mid r_1 \in F_1 \text{ tai } r_2 \in F_2 \}$.

Tarkastellaan mielivaltaista merkkijonoa $w = w_1 \dots w_k$. On olemassa automaatin M_1 laskentaa esittävä (yksikäsitteinen) tilajono $(r_0, \dots, r_k) \in Q_1^{k+1}$ ja automaatin M_2 laskentaa esittävä (yksikäsitteinen) tilajono $(s_0, \dots, s_k) \in Q_2^{k+1}$, missä

1. $r_0 = q_1$ ja $s_0 = q_2$ ja
2. $r_{i+1} = \delta_1(r_i, w_{i+1})$ ja $s_{i+1} = \delta_2(s_i, w_{i+1})$ kun $i = 0, \dots, n - 1$.

Siis valitsemalla $p_i = (r_i, s_i)$ saadaan automaatin M laskentaa kuvaava tilajono, jolla $p_0 = q_0$ ja $p_{i+1} = \delta(p_i, w_{i+1})$. Nyt

M hyväksyy merkkijonon w

$$\Leftrightarrow (r_n, s_n) \in F$$

$$\Leftrightarrow r_n \in F_1 \text{ ja } s_n \in F_2$$

$$\Leftrightarrow M_1 \text{ hyväksyy merkkijonon } w \text{ ja } M_2 \text{ hyväksyy merkkijonon } w.$$

Siis $L(M) = A \cup B$. \square

Jatkossa osoitetaan myös

Lause 1.2: [Sipser Thm. 1.26] Säännöllisten kielten joukko on suljettu konkatenation suhteen; ts. jos kielet A ja B ovat säännöllisiä, niin myös $A \circ B$ on.

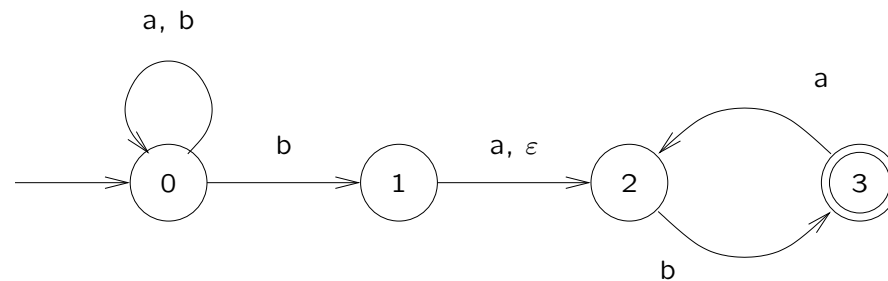
Todistaminen on kuitenkin edellistä lausetta hankalampaa. Oletaan $A = L(M_1)$ ja $B = L(M_2)$, ja halutaan muodostaa M , jolla $L(M) = A \circ B$. Luonnollinen ajatus olisi pistää automaattit M_1 ja M_2 peräkkäin. Tässä tulee ongelmaksi, milloin pitäisi siirtyä automaattista M_1 automaattiin M_2 .

Eryteisesti jollain $w \in A \circ B$ voi olla kaksi esitystä $w = uv = u'v'$, missä $u \in A$ ja $u' \in A$, ja $v \in B$ mutta $v' \notin B$. Jos tässä vielä u' on merkkijonon u alkuosa (eli $u = u'x$ jollain x), niin emme voi ilman muuta siirtyä automaattiin M_2 heti, kun M_1 pääsee hyväksyvään tilaan.

Jotta pääsemme eteenpäin, yleistämme äärellistä automaattia sallimalla **epädeterminismin**.

Epädeterministiset äärelliset automaattit [Sipser s. 47–54]

Tarkastellaan esimerkkinä seuraavaa automaattia:



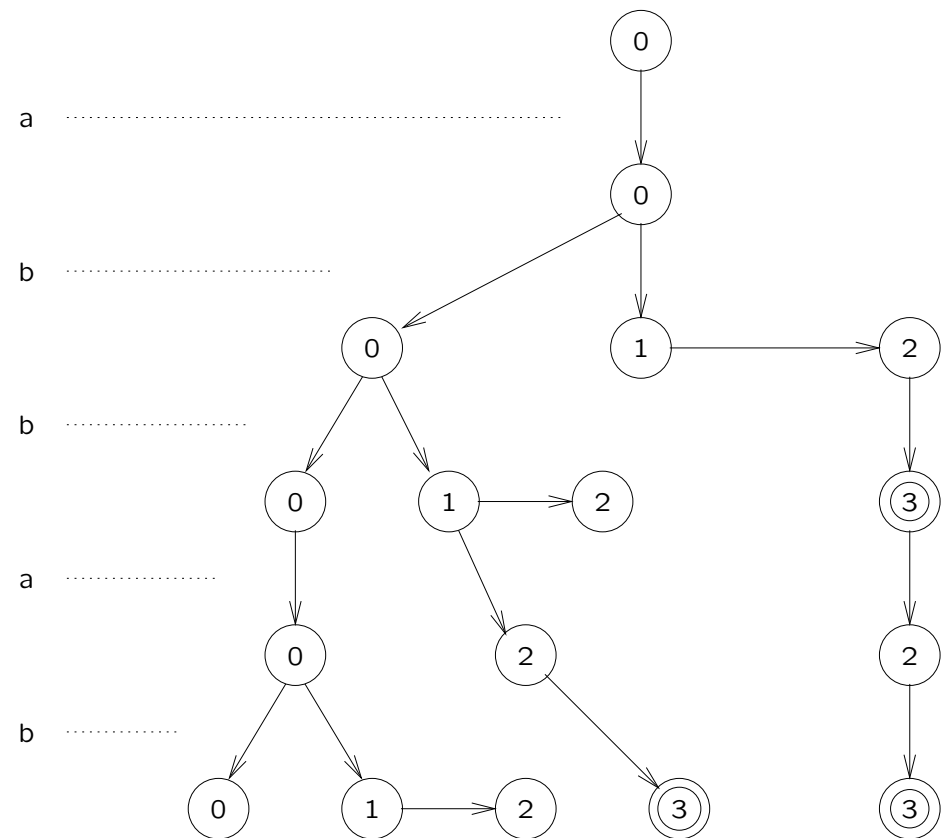
Tämä ei noudata edellä esitettyä formalismia:

- tilasta 0 pääsee merkillä b sekä tilaan 0 että tilaan 1,
- tilasta 1 on ϵ -siirtymä, jolla pääsee tilaan 2 "käyttämättä" yhtään syötemerkkiä ja
- joitain siirtymiä ei ole määritelty.

Aiottu tulkinta on, että annetulla syötteellä "kokeillaan" **kaikkia mahdollisia** siirtymävaihtoehtoja. Automaatti hyväksyy, jos **yksikin** näistä vaihtoehtoista päättyy hyväksyvään tilaan.

Tällaisen automaatin laskentaa voidaan havainnollistaa ryhmittämällä vaihtoehdot puuksi:

- esimerkkinä syöte abbab
- vaakanoilet kuvaavat ϵ -siirtymiä
- viimeisellä rivillä esiintyy kahteenkin kertaan hyväksyvä tila 3, joten automaatti hyväksyy merkkijonon abbab



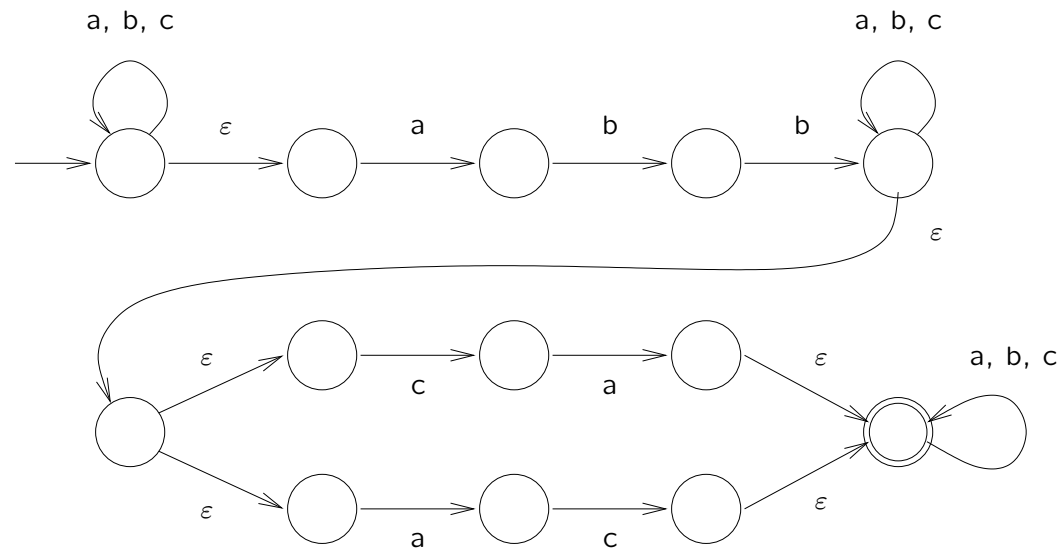
Kutsumme tällaista yleisempää automaattia **epädeterministiseksi äärelliseksi automaatiksi** (nondeterministic finite automaton, NFA). Aluksi esitetty perusversio on vastaavasti **deterministinen äärellinen automaatti** (deterministic finite automaton, DFA).

NFA:lle ei ole mitään samalla lailla ilmeistä fyysistä toteutusmallia kuin DFA:lle. Se kannattaa mieltää lähinnä **kuvausformalismiksi**.

Näemme jatkossa, että mille tahansa NFA:lle voidaan muodostaa DFA, joka tunnistaa saman kielen. Siis epädeterminismi **ei** anna tässä mielessä lisää **ilmaisuvoimaa**.

Epädeterminismiä käyttämällä esitystä voidaan kuitenkin usein selkeyttää ja yksinkertaistaa.

Seuraava NFA hyväksyy merkkijonot, joissa on osajonona ensin **abb** ja sen jälkeen **ca** tai **ac**. Huomaa konstruktion modulaarisuus. (Juuri tämän kielen voisi tunnistaa DFA:lla helpomminkin.)



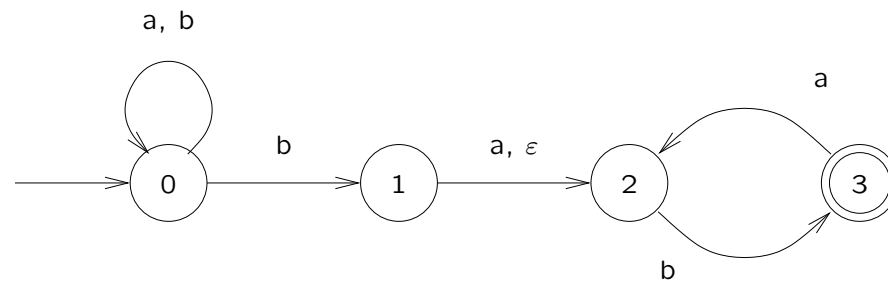
Mille tahansa aakkostolle Σ merkitsemme $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. Muistetaan, että joukon A **potenssijoukkoa** eli kaikkien osajoukkojen joukkoa merkitään $\mathcal{P}(A)$.

Muodollisesti NFA on viisikko $(Q, \Sigma, \delta, q_0, F)$, missä

1. Q on äärellinen tilajoukko,
2. Σ on äärellinen aakkosto,
3. $\delta: Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$ on siirtymäfunktio,
4. $q_0 \in Q$ on alkutila ja
5. $F \subseteq Q$ on hyväksyvien tilojen joukko.

Erotukseksi DFA:sta siirtymäfunktio siis antaa yhden seuraajatilan sijaan **joukon** tiloja. Nämä tulkitaan "mahdollisiksi" seuraajatiloina.

Esimerkkiautomaattimme



formaali esitys on siis $(\{0, 1, 2, 3\}, \{a, b\}, \delta, 0, \{3\})$, missä δ saadaan taulukosta

δ	a	b	ϵ
0	{0}	{0, 1}	\emptyset
1	{2}	\emptyset	{2}
2	\emptyset	{3}	\emptyset
3	{2}	\emptyset	\emptyset .

Määrittelemme nyt, että epädeterministinen äärellinen automaatti $(Q, \Sigma, \delta, q_0, F)$ hyväksyy merkkijonon w , jos jollain n voidaan valita jonot $(y_1, \dots, y_n) \in \Sigma_\varepsilon^n$ ja $(r_0, \dots, r_n) \in Q^{n+1}$, joilla

- $w = y_1 \dots y_n$,
- $r_0 = q_0$,
- $r_{i+1} \in \delta(r_i, y_{i+1})$ kun $i = 0, \dots, n - 1$ ja
- $r_n \in F$.

Edellisen sivun automaatilla ja merkkijonolla **abbab** voidaan valita esim.

$$\begin{aligned} n &= 6 \\ (y_1, y_2, y_3, y_4, y_5, y_6) &= (a, b, \varepsilon, b, a, b) \\ (r_0, r_1, r_2, r_3, r_4, r_5, r_6) &= (0, 0, 1, 2, 3, 2, 3), \end{aligned}$$

mistä nähdään, että automaatti hyväksyy merkkijonon.

NFA:n muuntaminen DFA:ksi

Lause 1.3: [Sipser Thm. 1.39] Mille tahansa NFA:lle on olemassa DFA, joka tunnistaa saman kielen.

Todistus: On siis annettu epädet. automaatti $N = (Q, \Sigma, \delta, q_0, F)$, ja halutaan muodostaa det. automaatti $M = (Q', \Sigma, \delta', q'_0, F')$, jolla $L(M) = L(N)$.

NFA:lla on kullakin laskennan hetkellä joukko mahdollisia tiloja. Toisaalta DFA on kullakin laskennan hetkellä jossain yksikäsitteisessä tilassa. Tämä perusongelma ratkaistaan valitsemalla $Q' = \mathcal{P}(Q)$. Automaatin M tila annetulla hetkellä tulkitaan automaatin N mahdollisten tilojen joukoksi. Huomaa, että $|Q'| = 2^{|Q|}$.

Jos $r \in Q$ on mahdollinen tila nykyhetkellä, seuraavaksi luetaan merkki a , ja $s \in \delta(s, a)$, niin s on eräs mahdollinen tila seuraavalla hetkellä. Siis jos $\delta'(R, a) = S$ (missä siis $R, S \subseteq Q$), niin joukossa S pitää olla kaikki ne tilat, jotka voivat seurata jotain joukon R tilaa merkillä a .

Esitetään sama formaalisti olettaen ensin, että automaatissa N ei ole ε -siirtymiä (eli $\delta(r, \varepsilon) = \emptyset$ kaikilla $r \in Q$). Siis $M = (Q', \Sigma, \delta', q'_0, F')$, missä

1. $Q' = \mathcal{P}(Q)$,

2. jos $R \in Q'$ ja $a \in \Sigma$, niin

$$\begin{aligned}\delta'(R, a) &= \bigcup_{r \in R} \delta(r, a) \\ &= \{s \in Q \mid s \in \delta(r, a) \text{ jollain } r \in R, \}\end{aligned}$$

3. $q'_0 = \{q_0\}$ ja

4. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$.

Jos automaatissa N on ε -siirtymiä, määritellään kaikilla $R \subseteq Q$ joukko $E(R)$ koostumaan niistä tiloista, joihin pääsee jostain tilasta $r \in R$ tekemällä nolla tai useampia ε -siirtymiä. (Siis aina $R \subseteq E(R)$.)

Siirtymäfunktion määritelmä muutetaan muotoon

$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a)).$$

Alkutilaksi asetetaan

$$\delta'_0 = E(\{\delta_0\}).$$

Konstruktion perusteella on selvää, että $L(M) = L(N)$. \square

Koska DFA:n muuntaminen NFA:ksi on triviaalia, saadaan

Korollari 1.4: [Sipser Cor. 1.40] Kieli on säännöllinen, jos ja vain jos se voidaan tunnistaa epädeterministisellä äärellisellä automaatilla. \square

Säännöllisten kielten sulkeumaominaisuudet

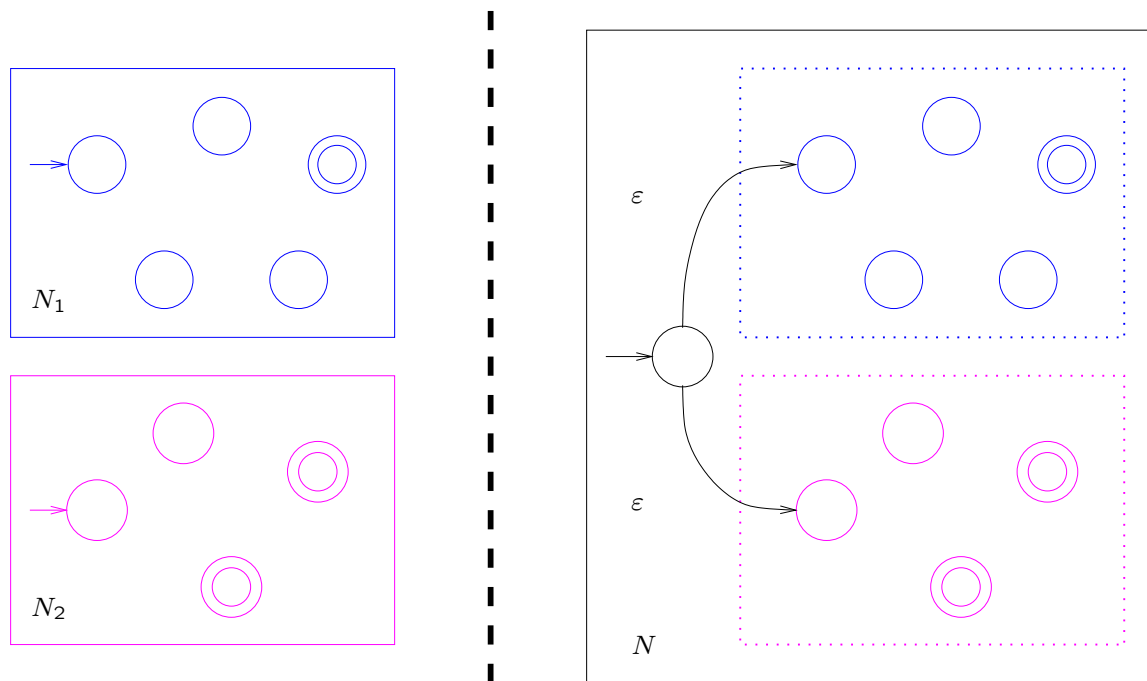
Osoitamme nyt, että säännöllisten kielten joukko on suljettu yhdisteen, konkatenation ja tähtioperaation suhteen. Toisin sanoen jos A ja B ovat säännöllisiä, niin myös $A \cup B$, $A \circ B$ ja A^* ovat.

Kieli on määritelmän mukaan säännöllinen, jos se voidaan tunnistaa deterministisellä äärellisellä automaatilla. Todistimme juuri (korollaari 1.4), että voimme yhtä hyvin käyttää epädeterministisiä automaatteja. Katsotaan ensin, miten tämä yksinkertaistaa jo esitetyn tapauksen $A \cup B$ (lause 1.1) todistusta.

Lause 1.5: [Sipser Thm. 1.45] Jos A ja B ovat säännöllisiä, niin $A \cup B$ on.

Todistus: Oletetaan siis annetuksi NFA:t N_1 ja N_2 , joilla $A = L(N_1)$ ja $B = L(N_2)$. Muodostetaan NFA N , jolla $L(N) = A \cup B$.

Konstruktion periaate on seuraava:



Muodollisemmin olkoon $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ ja $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
Voidaan tietysti olettaa, että $Q_1 \cap Q_2 = \emptyset$. Nyt $N = (Q, \Sigma, \delta, q_0, F)$, missä

1. $q_0 \notin Q_1 \cup Q_2$ on uusi tila,

2. $Q = \{q_0\} \cup Q_1 \cup Q_2$,

3. δ toteuttaa

$$\delta(q, a) = \delta_1(q, a) \quad \text{kun } q \in Q_1 \text{ ja } a \in \Sigma_\varepsilon$$

$$\delta(q, a) = \delta_2(q, a) \quad \text{kun } q \in Q_2 \text{ ja } a \in \Sigma_\varepsilon$$

$$\delta(q_0, \varepsilon) = \{q_1, q_2\}$$

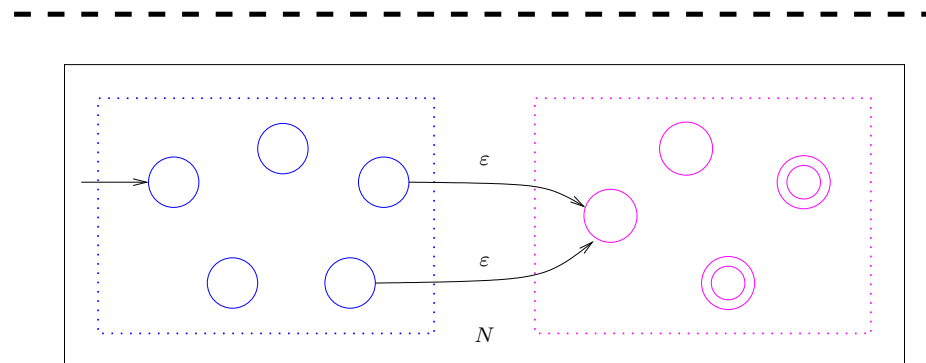
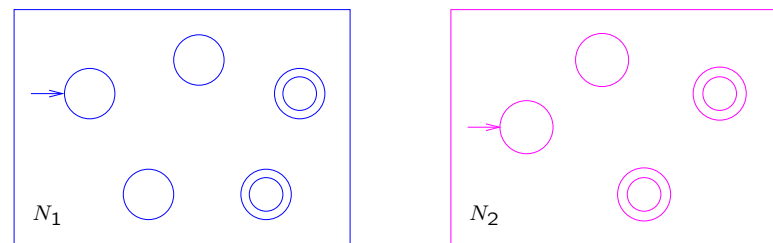
$$\delta(q_0, b) = \emptyset \quad \text{kun } b \neq \varepsilon.$$

Selvästi N tunnistaa kielen $A \cup B$. \square

Sovelletaan samaa ideaa konkatenaatioon.

Lause 1.6: [Sipser Thm. 1.47] Jos A ja B ovat säännöllisiä, myös $A \circ B$ on.

Todistus: Periaatepiirros on nyt seuraava:



Muodollisesti olkoon taas $A = L(N_1)$ ja $B = L(N_2)$, missä $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ ja $Q_1 \cap Q_2 = \emptyset$. Muodostetaan $N = (Q, \Sigma, \delta, q_0, F)$, missä

1. $Q = Q_1 \cup Q_2$,

2. $q_0 = q_1$,

3. $F = F_2$ ja

4. kun $q \in Q$ ja $a \in \Sigma_\varepsilon$, niin $\delta(q, a)$ on seuraava:

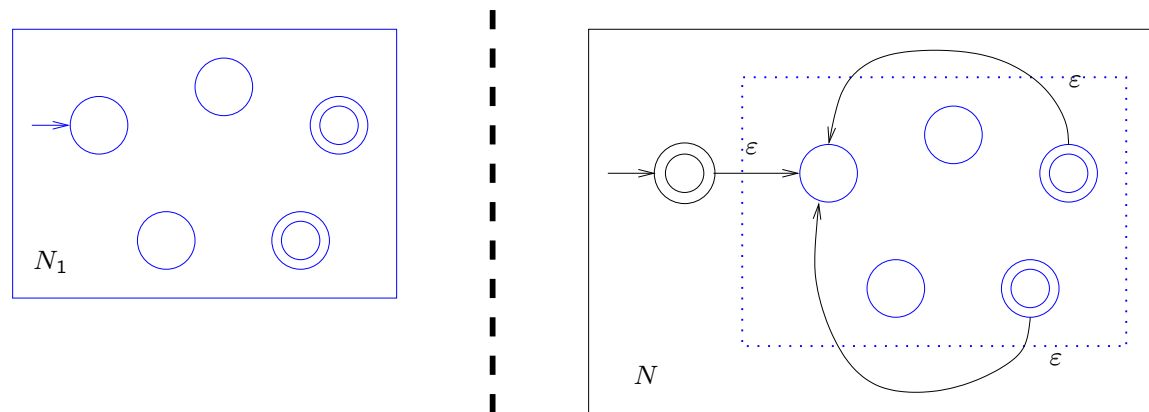
$$\begin{aligned} \delta(q, a) &= \delta_1(q, a) && \text{kun } q \in Q_1 - F_1 \\ \delta(q, a) &= \delta_1(q, a) && \text{kun } q \in F_1 \text{ ja } a \neq \varepsilon \\ \delta(q, \varepsilon) &= \delta_1(q, \varepsilon) \cup \{q_2\} && \text{kun } q \in F_1 \\ \delta(q, a) &= \delta_2(q, a) && \text{kun } q \in Q_2. \end{aligned}$$

Selvästi $L(N) = A \circ B$. \square

Vielä viimeiseksi tähtioperaatio:

Lause 1.7: [Sipser Thm 1.49] Jos A on säännöllinen, niin A^* on.

Todistus: Oletetaan taas $A = L(N_1)$, ja muodostetaan N jolla $L(N) = A^*$.
Periaate:



(Huomaa, että alkutilan vaihtaminen hyväksyväksi ei välttämättä toimisi.)

Olkoon siis $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$. Nyt $N = (Q, \Sigma, \delta, q_0, F)$, missä

1. $q_0 \notin Q_1$ on uusi tila,
2. $Q = Q_1 \cup \{q_0\}$,
3. $F = F_1 \cup \{q_0\}$ ja
4. δ määritellään

$$\begin{aligned}\delta(q_0, \varepsilon) &= \{q_1\} \\ \delta(q_0, a) &= \emptyset \quad \text{kun } a \neq \varepsilon \\ \delta(q, a) &= \delta_1(q, a) \quad \text{kun } q \in Q_1 - F_1 \\ \delta(q, a) &= \delta_1(q, a) \quad \text{kun } q \in F_1 \text{ ja } a \neq \varepsilon \\ \delta(q, \varepsilon) &= \delta_1(q, \varepsilon) \cup \{q_1\} \quad \text{kun } q \in F_1.\end{aligned}$$

Selvästi $L(N) = A^*$. \square

Säännölliset lausekkeet [Sipser luku 1.3]

Kiinnitetään jokin aakkosto Σ . Säännöllinen lauseke on kaava, jonka "arvona" on jokin aakkoston Σ kieli. Kun R on säännöllinen lauseke, sen "arvoa" eli sen esittämää kieltä merkitään $L(R)$. Perusajatuksena on muodostaa uusia kieliä edellä esitettyjen operaatioiden \cup , \circ ja $*$ avulla.

Esimerkki Säännöllisen lausekkeen $a(b \cup c)^*$ esittämään kieleen kuuluvat ne merkkijonot, joiden alussa on a ja sen jälkeen nolla tai useampia b - ja c -merkkejä. Siis

$$\begin{aligned} L(a(b \cup c)^*) &= \{a\} \circ (\{b, c\})^* \\ &= \{a, ab, ac, abb, abc, acb, acc, abbb, \dots\}. \end{aligned}$$

Osoitamme jatkossa, että tällä tavoin voidaan esittää (tai kuvata) tasan ne kielet, jotka voidaan tunnistaa äärellisellä automaatilla, eli säännölliset kielet (mistä nimi).

Määrittelemme säännölliset lausekkeet seuraavasti:

1. \emptyset ja ε ovat säännöllisiä lausekkeita, ja $L(\emptyset) = \emptyset$ ja $L(\varepsilon) = \{\varepsilon\}$.
2. a on säännöllinen lauseke kaikilla $a \in \Sigma$, ja $L(a) = \{a\}$.
3. Jos R_1 ja R_2 ovat säännöllisiä lausekkeita, niin myös $(R_1 \cup R_2)$ ja $(R_1 \circ R_2)$ ovat, ja $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$ ja $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$.
4. Jos R on säännöllinen lauseke, niin myös (R^*) on, ja $L((R^*)) = (L(R))^*$; ja
5. ei ole muita säännöllisiä lausekkeita, kuin mitä edelläolevasta seuraa.

Siis säännöllinen lauseke muodostetaan lähtemällä liikkeelle peruslausekkeista \emptyset , ε ja a , $a \in \Sigma$, ja soveltamalla yhdistettä, konkatenatiota ja tähteä äärellinen määrä kertoja.

Yksinkertaistamme säännöllisten lausekkeiden kirjoitusasua seuraavilla konventioilla:

- Konkatenaatio jätetään merkitsemättä; esim. $a \circ b \circ a$ kirjoitetaan aba .
- Operaatiot evaluoidaan presedenssijärjestyksessä "tähti, konkatenaatio, yhdiste" ja turhat sulut jätetään pois. Siis $a \cup bc^*$ tarkoittaa $(a \cup (b \circ (c^*)))$.
- Jos $\Sigma = \{a_1, \dots, a_k\}$, niin lauseketta $a_1 \cup \dots \cup a_k$ merkitään lyhyesti Σ .
- Lausekkeen R konkatenatiota itsensä kanssa k kertaa

$$\underbrace{R \circ \dots \circ R}_{k \text{ kertaa}}$$

merkitään R^k .

- Lauseke RR^* lyhennetään R^+ ; siis $R^* = \varepsilon \cup R^+$.

Jos sekaannuksen vaaraa ei ole, samastamme säännöllisen lausekkeen ja sen esittämän kielen (eli kirjoitamme R vaikka tarkoittammekin $L(R)$).

Esimerkkejä:

- Σ^* koostuu kaikista aakkoston Σ merkkijonoista ja Σ^+ kaikista aakkoston Σ ei-tyhjästä merkkijonoista.
- $1\Sigma^*$ esittää kaikkia ykkösellä alkavia merkkijonoja.
- $(0(0 \cup 1)^*1) \cup (1(0 \cup 1)^*0)$ esittää kaikkia aakkoston $\{0, 1\}$ merkkijonoja, joiden viimeinen merkki on eri kuin ensimmäinen.
- $R\emptyset = \emptyset$ kaikilla R , ja $\emptyset^* = \varepsilon$.

Operaatioilla \cup ja \circ on neutraali-alkiot \emptyset ja ε : kaikilla R pätee

$$R \cup \emptyset = \emptyset \cup R = R$$

$$R \circ \varepsilon = \varepsilon \circ R = R.$$

Säännöllisiä lausekkeita käytetään erilaisissa tekstinkäsittelytyökaluissa kuten Unixin `grep`.

Säännölliset lausekkeet ovat myös tärkeä formalismi ohjelmointikielten syntaktisten perusalkioiden esittämisessä. Esim. jossain ohjelmointikielessä sallitut tunnukset voisivat olla muotoa

$\text{kirjain}(\text{kirjain} \cup \text{numero} \cup \text{alaviiva})^*$,

kokonaislukupakiot muotoa

$(\epsilon \cup + \cup -)\text{numero}^+$

ja liukulukupakiot muotoa

$(\epsilon \cup + \cup -)(\text{numero}^*.\text{numero}^+ \cup \text{numero}^+.\text{numero}^*)$,

missä "kirjain" tarkoittaa $a \cup \dots \cup z$ ja "numero" tarkoittaa $0 \cup \dots \cup 9$. Tätä voidaan käyttää hyväksi kääntämisen ensimmäisessä vaiheessa muodostamalla sitä varten äärellinen automaatti (kuten seuraavaksi nähdään).

Äärellisten automaattien ja säännöllisten kielten ekvivalenssi

Osoitamme seuraavan keskeisen tuloksen:

Lause 1.8: [Sipser Thm. 1.54] Kieli on säännöllinen, jos ja vain jos jokin säännöllinen lauseke esittää sitä.

Siis säännöllisillä lausekkeilla voidaan esittää tasan ne kielet, jotka voidaan tunnistaa äärellisillä automaateilla.

Todistuksen helpompi suunta on seuraava:

Lemma 1.9: [Sipser Lemma 1.55] Säännöllisen lausekkeen kuvaama kieli voidaan tunnistaa äärellisellä automaatilla.

Todistus: Ennen yksityiskohtiin menemistä tarkastellaan yleisesti tällaisen todistuksen luonnetta. Säännöllisen lausekeen määritelmä on **induktiivinen** (eli rekursiivinen).

Vertaa induktiiviseen luonnollisen luvun määritelmään:

1. 0 on luonnollinen luku ja
2. jos n on luonnollinen luku, niin $n + 1$ on luonnollinen luku.

Luonnollisten lukujen induktioperiaatteesta seuraa, että jos

1. luvulla 0 on ominaisuus P ja
2. jos luvulla n on ominaisuus P niin luvulla $n + 1$ on ominaisuus P

niin kaikilla luonnollisilla luvuilla on ominaisuus P .

Todistamme lemmän suorittamalla analogisen ”induktion säännöllisten lausekkeiden yli”.

Haluamme siis osoittaa, että kaikilla säännöllisillä lausekkeilla R kieli $L(R)$ on säännöllinen. Teemme tämän osoittamalla, että

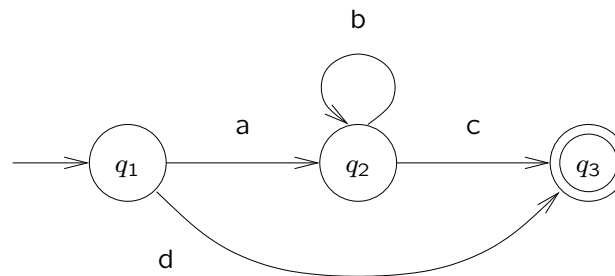
1. $L(\emptyset)$ ja $L(\varepsilon)$ ovat säännöllisiä,
2. $L(a)$ on säännöllinen kaikilla $a \in \Sigma$,
3. jos $L(R_1)$ ja $L(R_2)$ ovat säännöllisiä, niin $L(R_1 \cup R_2)$ ja $L(R_1 \circ R_2)$ ovat säännöllisiä ja
4. jos $L(R)$ on säännöllinen, niin $L(R^*)$ on säännöllinen.

Kohdat (1) ja (2) ovat ilmeisiä: on helppoa muodostaa äärellinen automaatti tunnistamaan kieli \emptyset , kieli $\{\varepsilon\}$ ja kieli $\{a\}$, missä $a \in \Sigma$.

Koska $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$ ja $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$, niin kohta (3) seuraa lauseista 1.5 ja 1.6. Samoin kohta (4) seuraa lauseesta 1.7. \square

Osoitetaan nyt kääntäen, että mille tahansa äärelliselle automaatille M voidaan muodostaa säännöllinen lauseke R , jolle $L(R) = L(M)$.

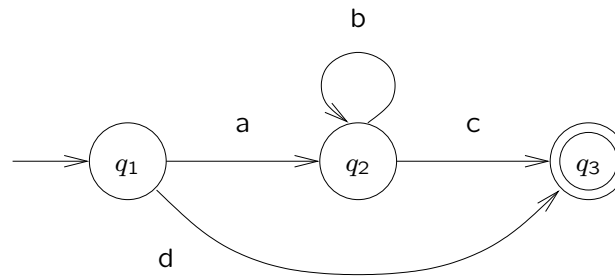
Jos M on automaatti



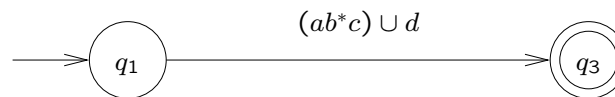
niin selvästi voidaan valita $R = (ab^*c) \cup d$.

Jotta voisimme hyödyntää tätä havaintoa osana monimutkaisemman automaatin analysoimista, otamme käyttöön [yleistetyt](#) NFA:t (generalized NFA, GNFA), joissa kaareen voi liittyä yksittäisen symbolin tai symbolijoukon sijasta kokonainen säännöllinen lauseke.

Ajatuksena on esim., että edellä esitetystä automaatista



voidaan eliminoida tila q_2 ja saada ekvivalentti GNFA



Meidän pitää nyt sopia tarkemmin, mitä GNFA tarkalleen on.

GNFA on siis kuten NFA, mutta kuhunkin kaareen voi liittyä mielivaltainen säännöllinen lauseke. Siirtyminen kaarta pitkin ”kuluttaa” syötteestä jonkin osamerkkijonon, joka kuuluu kyseisen säännöllisen lausekkeen kuvaamaan kieleen.

Yksinkertaisuuden vuoksi sovimme, että GNFA noudattaa seuraavia rajoitteita:

1. Siinä on tasan yksi hyväksyvä tila ja yksi alkutila, ja nämä eivät ole sama tila.
2. Alkutilaan ei tule siirtymiä.
3. Hyväksyvistä tilasta ei lähde siirtymiä.
4. Muuten minkä tahansa kahden tilan välillä on siirtymä, johon liittyy tasan yksi säännöllinen lauseke.

Muodollisesti GNFA on siis viisikko $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, missä

1. Q on äärellinen tilajoukko,
2. Σ on aakkosto,
3. δ on funktio $(Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \mathcal{R}$, missä \mathcal{R} on aakkoston Σ säännöllisten lausekkeiden joukko ja
4. $q_{\text{start}} \in Q$, $q_{\text{accept}} \in Q$ ja $q_{\text{start}} \neq q_{\text{accept}}$.

GNFA $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ hyväksyy merkkijonon $w \in \Sigma^*$, jos on olemassa esitys $w = w_1 \dots w_k$, missä $w_i \in \Sigma^*$ kaikilla $1 \leq i \leq k$, ja tilajono $(q_0, \dots, q_k) \in Q^{k+1}$, joilla

1. $q_0 = q_{\text{start}}$,
2. $w_i \in L(R_i)$ kaikilla $1 \leq i \leq k$, missä $R_i = \delta(q_{i-1}, q_i)$ ja
3. $q_k = q_{\text{accept}}$.

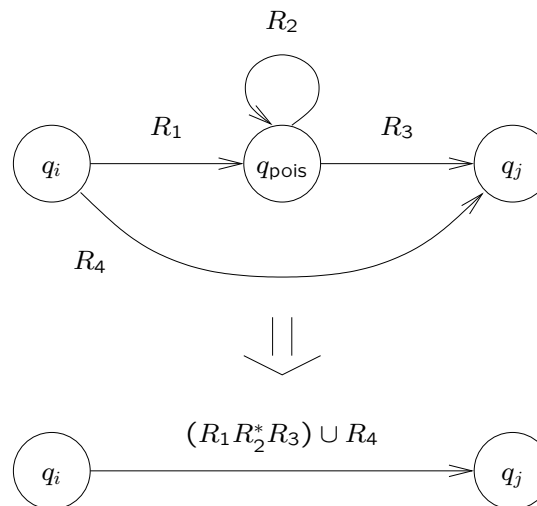
Mistä tahansa NFA:sta voidaan helposti muodostaa saman kielen tunnistava GNFA:

1. Lisää uudet tilat q_{start} ja q_{accept} .
2. Lisää ε -siirtymät tilasta q_{start} vanhaan alkutilaan ja vanhoista hyväksyvistä tiloista tilaan q_{accept} .
3. Kaikilla alkuperäisillä tiloilla q_1 ja q_2 aseta $\delta(q_1, q_2) = a_1 \cup \dots \cup a_n$, missä $\{a_1, \dots, a_n\}$ on niiden merkkien joukko, joilla vanhassa automaatissa on siirtymä tilasta q_1 tilaan q_2 .
4. Niillä q_1 ja q_2 , joilla $\delta(q_1, q_2)$ ei vielä tullut määritellyksi, aseta $\delta(q_1, q_2) = \emptyset$.

Korkean tason algoritmi NFA:ta M vastaavan säännöllisen lausekkeen muodostamiseksi on nyt seuraava:

1. Olkoon automaatin M tilojen lukumäärä k . Muodosta $k + 2$ -tilainen GNFA G_{k+2} , joka tunnistaa saman kielen.
2. Toista kun $i = k + 2, k + 1, \dots, 3$: muodosta GNFA:sta G_i saman kielen tunnistava GNFA G_{i-1} , jossa on yksi tila vähemmän.
3. Päättele kaksitilaisesta GNFA:sta G_2 , mikä säännöllinen lauseke esittää sen tunnistamaa kieltä.

Oleellisin kohta on tietysti yhden tilan eliminoiminen GNFA:sta. Se perustuu muunnokseen



Kootaan nyt edelläesitetty yhteen.

Lemma 1.10: [Sipser Lemma 1.60] Jos kieli on säännöllinen, se voidaan esittää säännöllisellä lausekkeella.

Todistus: Olkoon A säännöllinen kieli. Siis jokin äärellinen automaatti M tunnistaa sen. Muodostetaan edellä selitettyyn tapaan GNFA G , joka sekin tunnistaa kielen A .

Seuraavaksi lasketaan $G' = \text{Pienennä}(G)$, joka on 2-tilainen GNFA ja tunnistaa saman kielen kuin G . (Palaamme tähän pian.) Nyt G' koostuu pelkästään tiloista q_{start} ja q_{accept} sekä niiden välisestä siirtymästä. Automaatin G' hyväksymä kieli on määritelmän mukaan sama kuin $L(R)$, missä $R = \delta(q_{\text{start}}, q_{\text{accept}})$. Siis R on haluttu säännöllinen lauseke, joka esittää automaatin M tunnistamaa kieltä.

Proseduuri Pienennä(G) toimii seuraavasti:

Olkoon $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$. Jos automaatissa G on kaksi tilaa, palauta se sellaisenaan.

Muuten valitse mielivaltainen $q_{\text{pois}} \in Q - \{q_{\text{start}}, q_{\text{accept}}\}$. Muodosta $G' = (Q - \{q_{\text{pois}}\}, \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, missä kaikilla

$$\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4,$$

missä $R_1 = \delta(q_i, q_{\text{pois}})$, $R_2 = \delta(q_{\text{pois}}, q_{\text{pois}})$, $R_3 = \delta(q_{\text{pois}}, q_j)$ ja $R_4 = \delta(q_i, q_j)$. Suorita rekursiivinen kutsu Pienennä(G') ja palauta sen palauttama arvo.

Selvästi Pienennä(G) palauttaa 2-tilaisen GNFA:n. Pitää enää osoittaa, että edellä muodostettu G' tunnistaa saman kielen kuin G ; tästä seuraa induktiolla, että myös Pienennä(G) tunnistaa.

Väitämme siis, että G ja G' hyväksyvät tasan samat merkkijonot. Olkoon ensin w merkkijono, jonka G hyväksyy, ja tätä hyväksyntää vastaava tilajono

$$q_{\text{start}}, q_1, \dots, q_{k-1}, q_{\text{accept}}.$$

Jos q_{pois} ei esiinny tässä jonossa, se osoittaa samalla, että myös G' hyväksyy. Tämä seuraa siitä, että $\delta'(q_i, q_j)$ kaikilla q_i, q_j esittää ainakin yhtä laaja kieli kuin $\delta(q_i, q_j)$.

Oletetaan toisaalta, että q_{pois} esiintyy tilajonossa. Tarkastellaan jotain sen perättäisten esiintymien jonoa $q_i, q_{\text{pois}}, \dots, q_{\text{pois}}, q_j$. Siirtymäfunktion δ' määritelmästä seuraa, että automaatin G tähän siirtymäketjuun "kuluttama" pätkä merkkijonoa w voidaan automaatissa G' käyttää suoraan siirtymään $q_i \rightarrow q_j$.

Samoin on helppo nähdä, että G hyväksyy merkkijonon w , jos G' hyväksyy.

□

Tämä päättää myös lauseen 1.8 todistuksen. Olemme siis kahdesta täysin eri lähtökohdasta (automaatit, lausekkeet) päätyneet samaan säännöllisten kielten luokkaan. Seuraavaksi tarkastelemme, mitä jää tämän luokan ulkopuolelle.

Ei-säännöllisiä kieliä [Sipser luku 1.4]

Osoitamme, että joitain kieliä ei voi tunnistaa äärellisellä automaatilla. Tulos ei sinänsä ole erityisen yllättävä, koska äärellinen automaatti on äärimmäisen yksinkertainen laskennan malli. Ei kuitenkaan ole ilmeistä, miten annetusta kielestä osoitetaan, että **mikään** äärellinen automaatti ei tunnista sitä. Tätä ei välttämättä näe suoraan: esim. kieli

$$C = \{ w \in \{0, 1\}^* \mid w \text{ sisältää yhtä monta nollaa ja ykköstä} \}$$

ei ole säännöllinen (kuten pian näemme), mutta

$$D = \{ w \in \{0, 1\}^* \mid w\text{:ssä esiintyy } 01 \text{ ja } 10 \text{ yhtä monta kertaa} \}$$

itse asiassa on säännöllinen (HT).

Tarkastelemme ensin konkreettista esimerkkiä ja johdamme sitten yleisen säännön, joilla voidaan **tietyissä tapauksissa** todeta kielen ei-säännöllisyys.

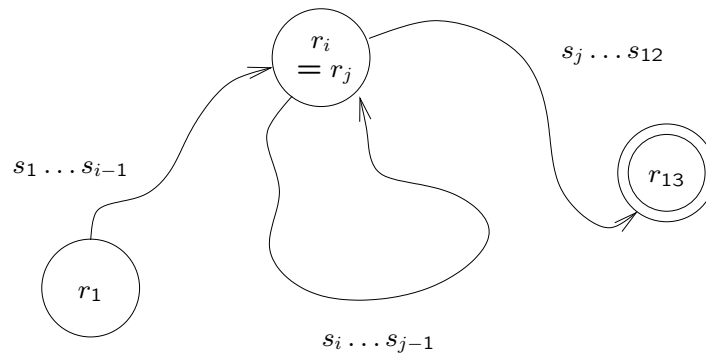
Väite: Kieltä

$$B = \{0^n 1^n \mid n \in \mathbf{N}\}$$

ei voi tunnistaa korkeintaan 12-tilaisella äärellisellä automaatilla.

Todistus: Tehdään vastaoletus, että $B = L(M)$ deterministisellä äärellisellä automaatilla $M = (Q, \Sigma, \delta, q_1, F)$, missä $|Q| \leq 12$. Siis erityisesti M hyväksyy merkkijonon $s = 000000111111 = 0^6 1^6$. Merkitään $s = s_1 \dots s_{12}$. Olkoon r_1, \dots, r_{13} vastaava tilajono, ts. $r_1 = q_1$ ja $r_{i+1} = \delta(r_i, s_i)$ kun $i = 1, \dots, 12$.

Keskeinen havainto: koska $|Q| \leq 12$ ja $r_i \in Q$ kaikilla i , niin jonossa r_1, \dots, r_{13} ainakin yksi tila esiintyy useamman kerran ("kyyhkyslakkaperiaate").
Olkoon $r_i = r_j$, missä $i < j$.



Merkkijonolla $s_i \dots s_{j-1}$ automaatti tekee **silmukan** tilasta r_i takaisin tilaan $r_i = r_j$.

Merkitään

$$x = s_1 \dots s_{i-1}$$

$$y = s_i \dots s_{j-1}$$

$$z = s_j \dots s_{12}.$$

Siis $s = xyz$ on alkuperäinen hyväksytty merkkijono, mutta automaatti hyväksyy myös merkkijonot $xy^0z = xz$, $xy^2z = xyyz$, $xy^3z = xyyyz$ jne.

On kolme mahdollisuutta:

1. $i < j \leq 6$: nyt $y = 0^{j-i+1}$.
2. $i \leq 6 < j$: nyt $y = 0^{6-i+1}1^{j-6}$.
3. $6 < i < j$: nyt $y = 1^{j-i+1}$.

Kaikissa tapauksissa xy^2z **ei** kuulu kieleen B :

- tapauksessa 1 siinä on liikaa nolliä,
- tapauksessa 2 se on muotoa $0 \dots 01 \dots 10 \dots 01 \dots 1$ ja
- tapauksessa 3 siinä on liikaa ykkösiä.

Koska M hyväksyy merkkijonon xy^2z , tämä on **ristiriita** oletuksen $B = L(M)$ kanssa. \square

Ilmeisesti sama todistusrunko toimii muillakin tilojen lukumäärillä kuin 12.
(Esitämme kohta täsmällisen todistuksen.)

Siis kieltä B ei voi tunnistaa p -tilaisella deterministisellä äärellisellä automaatilla **millään** p , eli kieli B **ei ole säännöllinen**.

Edellisen todistuksen perusidea oli osoittaa, että jos M on 12-tilainen automaatti, niin kielellä $L(M)$ on **pumppauspituus** 12:

Määritelmä 1.11: Kielellä A on **äärellinen pumppausominaisuus**, jos on olemassa sellainen p , että mikä tahansa $s \in A$, jolla $|s| \geq p$, voidaan esittää muodossa $s = xyz$, missä

1. $xy^iz \in A$ kun $i = 0, 1, 2, \dots$,
2. $|y| > 0$ ja
3. $|xy| \leq p$.

Tällöin p on (eräs) kielen A **pumppauspituus**. \square

Siis $x = \varepsilon$ ja $z = \varepsilon$ ovat sallittuja, mutta ehto 2 sulkee pois mahdollisuuden $y = \varepsilon$ (muuten pumpattavuus olisi triviaalia).

Kun kielellä on äärellinen pumppausominaisuus, millä tahansa riittävän pitkällä merkkijonolla s on "keskiosa" y , jota "pumppaamalla" saadaan uusia kieleen kuuluvia merkkijonoja xy^2z, xy^3z, \dots .

Lause 1.12 (Pumppauslemma): [Sipser Thm. 1.70] Jokaisella säännöllisellä kielellä on äärellinen pumppausominaisuus.

Ennen pumppauslemman todistusta näytämme esimerkkinä, miten siitä johdetaan em. kielen B ei-säännöllisyys.

Esimerkki 1.13: Kieli

$$B = \{ 0^n 1^n \mid n \in \mathbb{N} \}$$

ei ole säännöllinen.

Väitteen todistamiseksi tehdään **vastaoletus**, että B on säännöllinen. Siis kielellä B on jokin pumppauspituus p .

Merkitään $k = \lceil p/2 \rceil$ (missä $\lceil x \rceil = \min \{ n \in \mathbb{N} \mid n \geq x \}$). Valitaan $s = 0^k 1^k$.

Koska $|s| \geq p$, voidaan kirjoittaa $s = xyz$, missä $y \neq \varepsilon$ ja $xy^i z \in B$ kaikilla $i \in \mathbb{N}$.

Tämä on **ristiriita** kielen B määritelmän kanssa (vrt. kohdat 1–3 sivulla 71).

□

Pumppauslemman todistus: Olkoon A säännöllinen kieli ja $M = (Q, \Sigma, \delta, q_1, F)$ sen tunnistava DFA. Valitaan $p = |Q|$.

Olkoon nyt $n \geq p$ ja $s = s_1 \dots s_n \in A$. Olkoon r_1, \dots, r_{n+1} tilajono, jonka kautta M hyväksyy merkkijonon s . Siis $r_1 = q_1$, $r_{i+1} = \delta(r_i, s_i)$ kun $i = 1, \dots, n$, ja $r_{n+1} \in F$.

Kyyhkyslakkaperiaatteen nojalla jonossa r_1, \dots, r_{p+1} esiintyy jokin tila kahteen kertaan. Olkoon $r_j = r_l$, missä $j < l \leq p + 1$. Jaetaan s osiin $s = xyz$, missä

$$x = s_1 \dots s_{j-1}$$

$$y = s_j \dots s_{l-1}$$

$$z = s_l \dots s_n.$$

Koska $|y| = l - j \geq 1$ ja $|xy| = l - 1 \leq p$, tämä täyttää pumppausmääritelmän ehdot 2 ja 3.

Automaatti M tekee merkkijonolla y silmukan tilasta r_j takaisin samaan tilaan. Silmukkaa voidaan toistaa nolla tai useampia kertoja ilman, että lopputulos muuttuu. Siis M hyväksyy kaikki merkkijonot xy^iz , $z \in \mathbb{N}$. \square

Esimerkki 1.14: [Sipser Ex. 1.74] Kieli

$$C = \{ w \in \{0, 1\}^* \mid w \text{ sisältää yhtä monta nollaa ja ykköstä} \}$$

ei ole säännöllinen.

Vastaoletus: C on säännöllinen, jolloin sillä on pumppauspituus p . Valitaan $s = 0^p 1^p$. Olkoon $s = xyz \in C$ kuten pumppausmääritelmässä. Siis $y \neq \varepsilon$ ja $|xy| \leq p$, joten y koostuu yhdestä tai useammasta nolasta. Koska $xyz \in C$, niin $xyyz$ sisältää nollia enemmän kuin ykkösiä, joten $xyyz \notin C$; **ristiriita**. \square

Vaihtoehtoinen todistus: Tehdään vastaoletus, että C on säännöllinen. Kieli 0^*1^* on säännöllinen, joten kieli $(0^*1^*) \cap C$ on säännöllinen (harjoitus 2). Tämä on ristiriita, koska $(0^*1^*) \cap C$ on sama kuin esimerkin 1.13 ei-säännöllinen kieli B . \square

Huomautus 1: Pumppauslemma ei siis ole ainoa tapa todistaa kieli ei-säännölliseksi. On olemassa myös **ei-säännöllisiä** kieliä, joilla on **äärellinen pumppausominaisuus**, joten niiden ei-säännöllisyys pitää todistaa muuta kautta.

Huomautus 2: Miten valitaan sopiva s ? Jos olisi valittu $s = (01)^p$, ei olisi saatu ristiriitaa, koska tätä s voidaan pumpata (esim. $x = \varepsilon$, $y = 01$, $z = (01)^{p-1}$).

Yksi lähestymistapa on miettiä, mitkä olisivat hankalia tapauksia, jos kieltä yritettäisiin tunnistaa äärellisellä automaatilla.

Jos haluamme päättää, päteekö $s \in C$, ja saamme käydä merkkijonon s läpi vain kerran, niin intuitiivisesti meidän täytyy pitää kirjaa siitä, mikä on tähän mennessä nähtyjen nollien ja ykkösten lukumäärän ero. Tämä on mahdotonta äärellisellä automaatilla, koska p -tilaisessa automaatissa ei voi esittää yli p :n meneviä laskurin arvoja.

Merkkijono $s = 0^p 1^p$ on valittu siten, että se aiheuttaa "laskurin ylivuodon" mahdollisimman nopeasti.

Esimerkki 1.15: [Sipser Ex. 1.75] Kieli $F = \{ ww \mid w \in \{0, 1\}^* \}$ ei ole säännöllinen.

Vastaoletus: Kieli F on säännöllinen. Siis sillä on pumppauspituus p . Valitaan $s = 0^p 10^p 1 \in F$.

Olkoon $s = xyz$, missä $|y| > 0$ ja $|xy| \leq p$ (pumppausehdot 2 ja 3). Siis $y = 0^k$ jollain $k > 0$. Nyt $xyyz = 0^{p+k} 10^p 1$, joten pumppausehto 1 ei toteudu; **ristiriita**. \square

Edellisiä esimerkkejä yhdistää idea, että äärellinen automaatti ei osaa todeta kahta asiaa yhtäsuuriksi. Samaa intuitioon liittyy edellä esitetty "laskurin ylivuoto" -idea.

Tässä kuitenkin esim. valinta $s = 0^p 0^p$ ei sellaisenaan toimisi. Tarvitaan pieni trikki, jolla jako "alkupuoliskoon" ja "loppupuoliskoon" tehdään selväksi.

Esimerkki 1.16: [Sipser Ex. 1.76] Kieli $D = \{ 1^{n^2} \mid n \in \mathbf{N} \}$ ei ole säännöllinen.

Kieli siis koostuu ykkösjonoista, joiden pituudet ovat

$$0^2 = 0, \quad 1^2 = 1, \quad 2^2 = 4, \quad 3^2 = 9, \quad 4^2 = 16, \quad \dots$$

Havaitaan, että perättäisten pituuksien väliset aukot kasvavat. Tarkemmin alkuiden r ja $r + 1$ pituuksien väli on

$$r^2 - (r - 1)^2 = 2r - 1.$$

Tehdään nyt **vastaoletus**: kieli D on säännöllinen. Olkoon pumppauspituus p . Valitaan mikä tahansa $s \in D$, jolla $|s| \geq p$, ja tälle pumppausominaisuuden mukainen esitys $s = xyz$. Olkoon $|y| = k \geq 1$ ja $|xz| = l$. Nyt merkkijonot $xy^i z$ kuuluvat kieleen D , ja niiden pituudet muodostavat aritmeettisesti kasvavan jonon

$$l, \quad l + k, \quad l + 2k, \quad l + 3k, \quad \dots$$

Tämä on **ristiriita** sen kanssa, että kielessä D voi olla korkeintaan $(k + 1)/2$ perättäistä merkkijonoa, joiden pituuksien erotus on korkeintaan k . \square

Esimerkki 1.17: [Sipser Ex. 1.77] Kieli $E = \{ 0^i 1^j \mid i > j \}$ ei ole säännöllinen.

Vastaoletus: Oletetaan, että E on säännöllinen. Olkoon sillä pumppauspituus p .

Valitaan $s = 0^{p+1}1^p \in E$. Olkoon $s = xyz$ pumppausehdon mukaisesti. Taas ehtojen $|y| > 0$ ja $|xy| \leq p$ takia $y = 0^k$ jollain $k \geq 1$. Nyt $xy^0z = xz = 0^{p+1-k}1^p \notin E$; **ristiriita**. \square

Tässä siis "pumppattiin alaspäin" eli imettiin s :stä viimeinenkin y pois.

—

Siis monet melko yksinkertaisetkin tunnistustehtävät vaativat enemmän laskentavoimaa, kuin äärelliset automaattit tarjoavat. Ryhdymme seuraavaksi tarkastelemaan laajempaa yhteydettömien eli kontekstittomien kielten luokkaa.

2. Yhteydettömät kielet

Yhteydettömät eli kontekstittomat kielet (context-free language, CFL) ovat säännöllisiä kieliä laajempi luokka formaaleja kieliä. Ne voidaan esittää yhteydettömällä kieliopella (context-free grammar, CFG). Täsmälleen tämän kieliluokan tunnistamiseen kykeneväksi laskentalaitteeksi osoittautuu pinoautomaatti (push-down automaton, PDA).

Tämän luvun jälkeen opiskelija

- osaa määritellä kontekstittoman kieliopin ja pinoautomaatin sekä selittää niiden välisen suhteen,
- osaa muodostaa kieliopin yksinkertaiselle kontekstittomalle kielelle ja
- tuntee joitain perusesimerkkejä ei-yhteydettömistä kielistä ja osaa yksinkertaisissa tapauksissa osoittaa tämän ominaisuuden.

Yhteydettömät kieliopit [Sipser luku 2.1]

Johdantoesimerkkinä tarkastelemme kieltä $L = \{a^n b^m a^n \mid n > 0, m > 0\}$, joka on yhteydetön (mutta ei säännöllinen). Vastaavan kieliopin ytimenä on säännöt eli produktiot

$$\begin{aligned} S &\rightarrow aSa \\ S &\rightarrow aBa \\ B &\rightarrow bB \\ B &\rightarrow b. \end{aligned}$$

Kieliopin muuttujat eli välikkeet (variable, nonterminal symbol) ovat S ja B . Näistä S on erityisasemassa lähtösymbolina. Päätemerkit eli -symbolit (terminal) ovat a ja b . Kieliopista saadaan esim. merkkijonolle $aabbaa \in L$ seuraava johto (derivation):

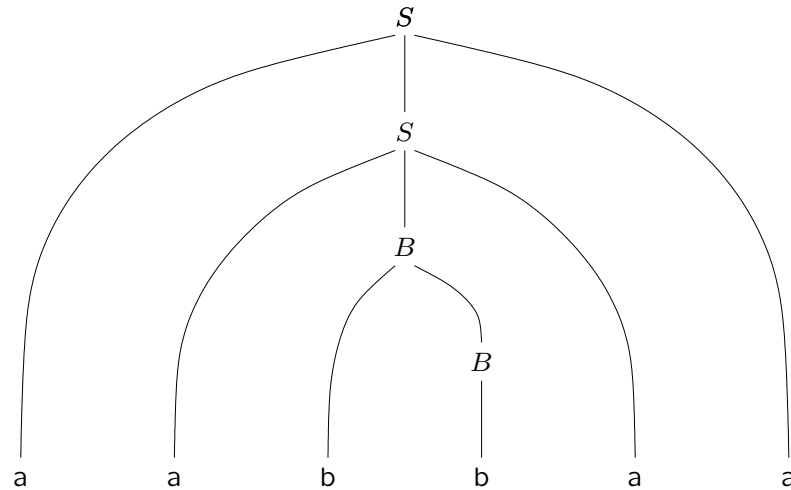
$$S \Rightarrow aSa \Rightarrow aaBaa \Rightarrow aabBaa \Rightarrow aabbaa.$$

Johdossa siis aloitetaan lähtösymbolista ja korvataan muuttujat yksi kerrallaan vastaavan produktion oikealla puolella olevalla merkkijonolla.

Merkkijonon aabbaa johtoa

$$S \Rightarrow aSa \Rightarrow aaBaa \Rightarrow aabBaa \Rightarrow aabbaa$$

vastaa seuraava **jäsennyspuu** (parse tree):



Lehtinä on johdetun merkkijonon merkit esijärjestyksessä ("vasemmalta oikealle"), juurena lähtösymboli ja sisäsolmuina välitteitä. Kunkin välitteeseen lapset vastaavat välitteeseen liittyvän säännön oikeaa puolta.

Muodollisesti yhteydetön eli kontekstiton kielioppi (context-free grammar, CFG) on nelikko (V, Σ, R, S) , missä

1. V on äärellinen muuttujien eli välikesymbolien joukko,
2. Σ on äärellinen päätesymbolien joukko, jolla $\Sigma \cap V = \emptyset$,
3. R on äärellinen joukko sääntöjä eli produktioita muotoa $A \rightarrow w$, missä $A \in V$ ja $w \in (V \cup \Sigma)^*$, ja
4. $S \in V$ on lähtösymboli.

Jos $x = uAv$ ja $y = uww$, missä $(A \rightarrow w) \in R$, niin x johtaa suoraan eli tuottaa suoraan (yields, derives directly) merkkijonon y . Tätä merkitään $x \Rightarrow y$.

Jos on olemassa merkkijonot $w_0, \dots, w_k \in (V \cup \Sigma)^*$, joilla

$$x = w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_k = y,$$

niin x johtaa eli tuottaa (derives) merkkijonon y . Tätä merkitään $x \xRightarrow{*} y$. Jos tällöin $k > 0$, voidaan merkitä $x \xRightarrow{\pm} y$.

Merkkijono $w \in (V \cup \Sigma)^*$ on **lausejohdos** (sentential form), jos $S \xRightarrow{*} w$. Jos lisäksi w sisältää vain päätesymboleja, se on **lause**.

Kieliopin G **tuottama** eli **kuvaama** kieli on sen lauseiden joukko

$$L(G) = \left\{ w \in \Sigma^* \mid S \xRightarrow{*} w \right\}.$$

Kieli on **yhteydetön**, jos jokin yhteydetön kielioppi tuottaa sen.

Jos $w \in L(G)$ ja

$$S \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_k = w,$$

tätä ketjua sanotaan merkkijonon w **johdoksi** (derivation), ja sen **pituus** on n . Tyypillisesti lauseella on useita eri johtoja; palaamme tähän pian.

Johdosta voidaan muodostaa jäsenyspuu ilmeisellä tavalla: kirjoitetaan ensin juureksi lähtösymboli, ja sitten laajennetaan solmuja soveltamalla johdon mukaisia sääntöjä.

Merkintäkonventioita

Yhteydettömiä kielioppeja käytetään mm. luonnollisten ja ohjelmointikielten syntaksin kuvaamiseen, jolloin niistä voi tulla hyvin laajoja.

Merkintöjen yksinkertaistamiseksi kieliopista riittää listata pelkkä sääntöjoukko:

- muuttujia ovat ne, jotka ovat jonkin säännön vasempana puolena,
- lähtösymboli on listassa ensimmäisen säännön vasen puoli ja
- muut symbolit ovat päätesymboleita.

Samaan muuttujaan liittyvät säännöt

$$A \rightarrow w_1, \quad A \rightarrow w_2, \quad \dots, \quad A \rightarrow w_n$$

voidaan tiivistää muotoon

$$A \rightarrow w_1 \mid w_2 \mid \dots \mid w_n.$$

Esimerkki 2.1: Yhteydetön kielioppi

$$S \rightarrow SS \mid (S) \mid \varepsilon$$

(jonka päätesymbolit ovat siis "(" ja ")") tuottaa kaikki oikein muodostetut sulkulausekkeet. \square

Esimerkki 2.2: Akkoston $\{a, b\}$ palindromien joukko $\{w \in \{a, b\}^* \mid w = w^R\}$ voidaan tuottaa yhteydettömällä kieliopilla

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb.$$

Vastaavasti kieli $\{ww^R \mid w \in \{a, b\}^*\}$ voidaan tuottaa yhteydettömällä kieliopilla

$$S \rightarrow \varepsilon \mid aSa \mid bSb.$$

Sen sijaan kieli $\{ww \mid w \in \{a, b\}^*\}$ ei ole yhteydetön, kuten jatkossa näemme. \square

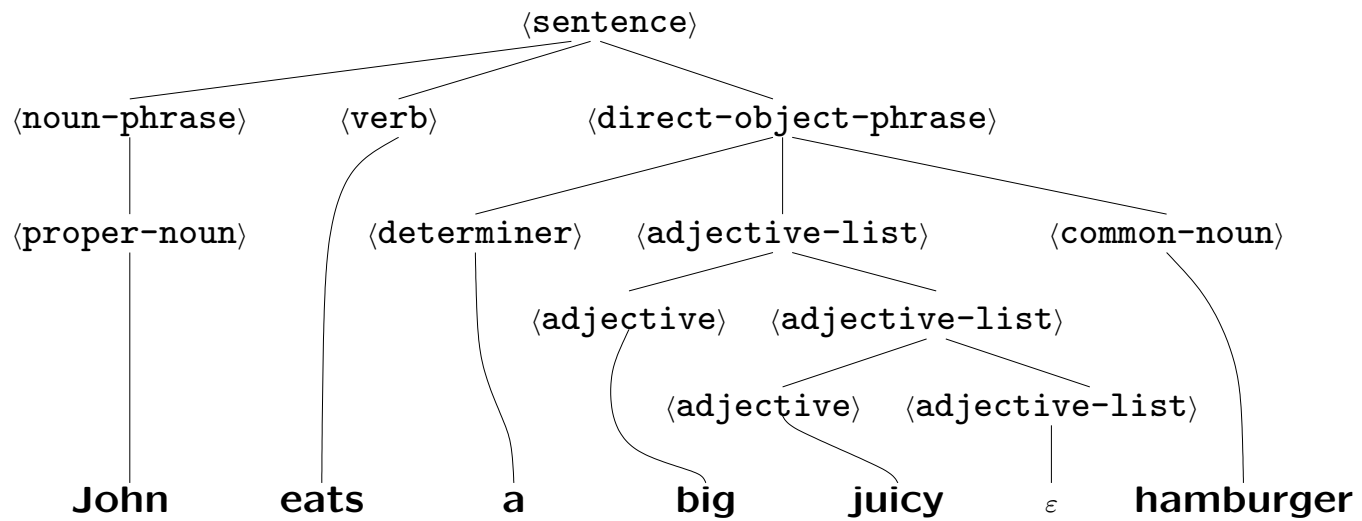
Huomautus: "Yhteydetön" viittaa siihen, että sääntöä $A \rightarrow w$ voidaan soveltaa riippumatta siitä, missä "yhteydessä" A esiintyy. Yleisemmissä yhteysherkissä kieliopissa sallitaan säännöt muotoa $uAv \rightarrow u w v$, jolloin A saadaan muuttaa w :ksi vain "kontekstissa" $u _ v$.

Edellisestä kieliopista saadaan esim. seuraavanlaisia johtoja:

- ⟨sentence⟩ ⇒ ⟨noun-phrase⟩⟨verb⟩⟨direct-object-phrase⟩
- ⇒ ⟨proper-noun⟩⟨verb⟩⟨direct-object-phrase⟩
- ⇒ **John** ⟨verb⟩⟨direct-object-phrase⟩
- ⇒ **John eats** ⟨direct-object-phrase⟩
- ⇒ **John eats** ⟨determiner⟩⟨adjective-list⟩⟨common-noun⟩
- ⇒ **John eats a** ⟨adjective-list⟩⟨common-noun⟩
- ⇒ **John eats a** ⟨adjective⟩⟨adjective-list⟩⟨common-noun⟩
- ⇒ **John eats a big** ⟨adjective-list⟩⟨common-noun⟩
- ⇒ **John eats a big** ⟨adjective⟩ ⟨adjective-list⟩⟨common-noun⟩
- ⇒ **John eats a big juicy** ⟨adjective-list⟩⟨common-noun⟩
- ⇒ **John eats a big juicy** ⟨common-noun⟩
- ⇒ **John eats a big juicy hamburger**

Tämä on esimerkki [vasemmasta johdosta](#) (leftmost derivation), mihin palataan pian.

Edellistä johtoa vastaava jäsenyspuu:



Esimerkki 2.4: Toinen sekä historiallisesti että nykykäytännön kannalta tärkeä yhteydettömien kielioppien sovellus on ohjelmointikielten syntaksin kuvaaminen. Tällöin käytetään usein ns. Backus-Naur-muotoa (BNF), joka on suunnilleen sama kuin yhteydetön kielioppi.

Kielioppi toimii paitsi ohjelmoijan muistilistana myös kääntäjän (tarkemmin jäsentäjän) laatimisen lähtökohtana.

Tarkastellaan esimerkkinä pientä Pascal-kielen osaa:

```

    <lause>   → <ehtolause> | <koottu-lause> | <sijoitus> | <kutsu>
<ehtolause> → if<ehto>then<lause>else<lause>
    <ehto>  → x=0
<koottu-lause> → begin<lausejono>end
<lausejono>  → <lause> | <lause> ; <lausejono>
<sijoitus>   → x:= 0
    <kutsu>   → a | b | c
```

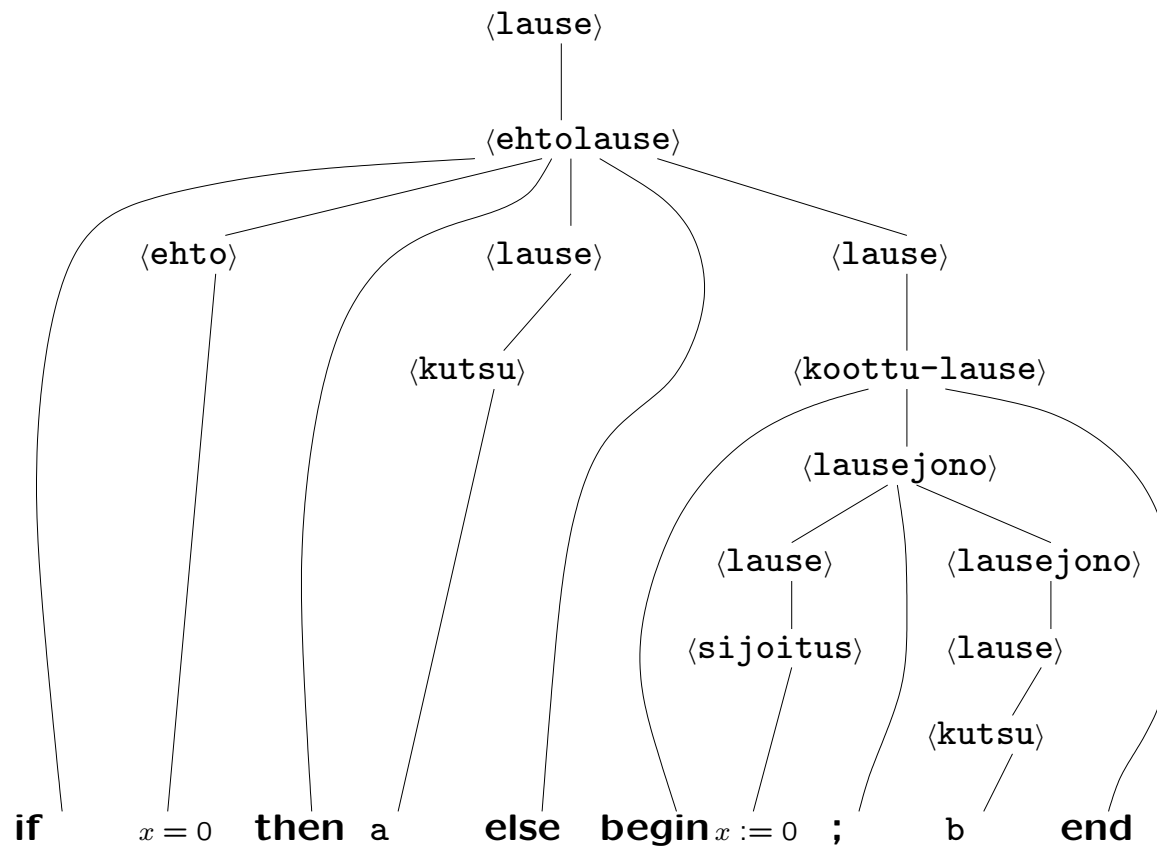
Esimerkkijohto:

`<lause>` ⇒ `<ehtolause>`
⇒ `if <ehto> then <lause> else <lause>`
⇒ `if x=0 then <lause> else <lause>`
⇒ `if x=0 then <kutsu> else <lause>`
⇒ `if x=0 then a else <lause>`
⇒ `if x=0 then a else <koottu-lause>`
⇒ `if x=0 then a else begin <lausejono> end`
⇒ `if x=0 then a else begin <lause> ; <lausejono> end`
⇒ `if x=0 then a else begin <sijoitus> ; <lausejono> end`
⇒ `if x=0 then a else begin x:=0 ; <lausejono> end`
⇒ `if x=0 then a else begin x:=0 ; <lause> end`
⇒ `if x=0 then a else begin x:=0 ; <kutsu> end`
⇒ `if x=0 then a else begin x:=0 ; b end`

Vastaava ohjelmanpätkä:

```
if x=0 then
    a
else begin
    x:= 0;
    b
end
```

Vielä jäsenyspuu samalle lauseelle:



Yhteydettömien kielten sulkeumaominaisuuksista

Yhteydettömien kielten luokka on suljettu yhdisteen, konkatenation ja tähtioperaation suhteen:

Lause 2.5: Jos A ja B ovat yhteydettömiä kieliä, niin myös $A \cup B$, $A \circ B$ ja A^* ovat.

Todistus: suoraviivainen harjoitustehtävä. \square

Tästä seuraa erityisesti

Korollari 2.6: Kaikki säännölliset kielet ovat yhteydettömiä. \square

Tulemme jatkossa todistaneeksi tämän korollarin myös toista kautta osoittamalla, että äärellisiä automaatteja voimakkaammat **pinoautomaatit** tunnistavat kuitenkin vain yhteydettömiä kieliä.

Jatkossa osoittautuu myös, että yhteydettömien kielten luokka **ei** ole suljettu leikkauksen ja komplementin suhteen, ts. $A \cap B$ ja \overline{A} eivät välttämättä ole yhteydettömiä.

Moniselitteisyys [Sipser s. 107–108]

Kontekstittoman kielen lauseella on tyypillisesti useita johtoja. Esim. kieliopissa

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

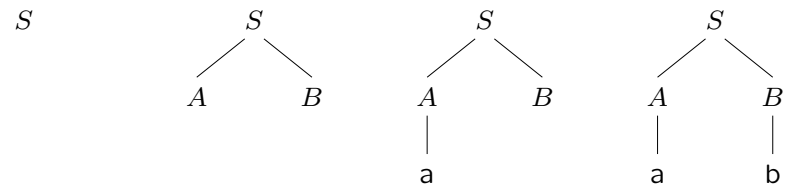
merkkijono ab voidaan johtaa kahdella eri tavalla: $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$ ja $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$. Intuitiivisesti näissä johdoissa on vain käytetty samoja sääntöjä eri järjestyksessä. Johdot myös vastaavat samaa jäsennyyspuuta.

Tällaisen triviaalin monijohtoisuuden poissulkemiseksi tarkastelemme **vasempia johtoja** (leftmost derivation). Vasemmassa johdossa sovelletaan aina produktiota merkkijonon vasemmanpuoleisimpaan muuttujaan. Siis em. kieliopissa merkkijonon ab ainoa vasen johto on

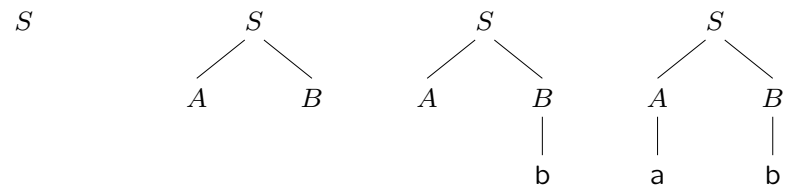
$$S \Rightarrow AB \Rightarrow aB \Rightarrow ab.$$

Kielioppi on **moniselitteinen** (ambiguous), jos siinä jollain lauseella on useampi kuin yksi vasemmanpuoleinen johto. Muuten kielioppi on **yksiselitteinen**. Tähänastiset esimerkkimme ovat kaikki olleet yksiselitteisiä.

Vasen johto $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$ esittää jäsennesspuun kasvattamista vasemmalta alkaen:



Samaan jäsennesspuuhun päästään myös johdolla $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$:

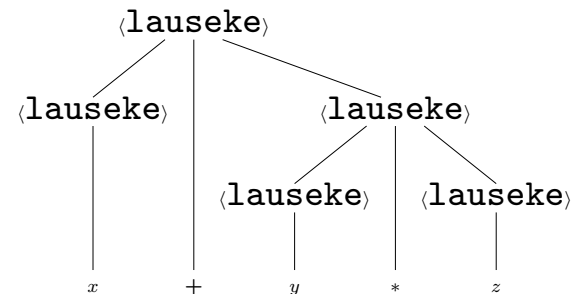


Esimerkki 2.7: Muuttujien x , y ja z aritmeettiset lausekkeet voidaan kuvata kieliopilla

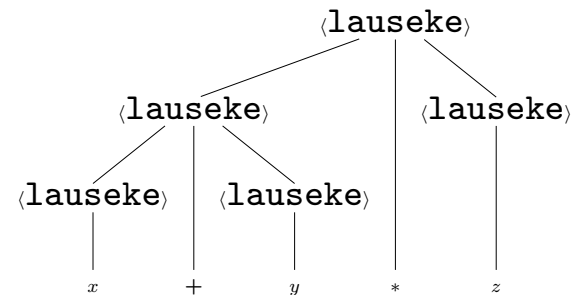
$\langle \text{lauseke} \rangle \rightarrow \langle \text{lauseke} \rangle + \langle \text{lauseke} \rangle \mid \langle \text{lauseke} \rangle * \langle \text{lauseke} \rangle \mid (\langle \text{lauseke} \rangle) \mid x \mid y \mid z.$

Kielioppi on moniselitteinen, sillä lauseella $x + y * z$ on kaksi vasenta johtoa ja jäsennyspuuta:

$\langle \text{lauseke} \rangle \Rightarrow \langle \text{lauseke} \rangle + \langle \text{lauseke} \rangle$
 $\Rightarrow x + \langle \text{lauseke} \rangle$
 $\Rightarrow x + \langle \text{lauseke} \rangle * \langle \text{lauseke} \rangle$
 $\Rightarrow x + y * \langle \text{lauseke} \rangle$
 $\Rightarrow x + y * z$



$\langle \text{lauseke} \rangle \Rightarrow \langle \text{lauseke} \rangle * \langle \text{lauseke} \rangle$
 $\Rightarrow \langle \text{lauseke} \rangle + \langle \text{lauseke} \rangle * \langle \text{lauseke} \rangle$
 $\Rightarrow x + \langle \text{lauseke} \rangle * \langle \text{lauseke} \rangle$
 $\Rightarrow x + y * \langle \text{lauseke} \rangle$
 $\Rightarrow x + y * z.$

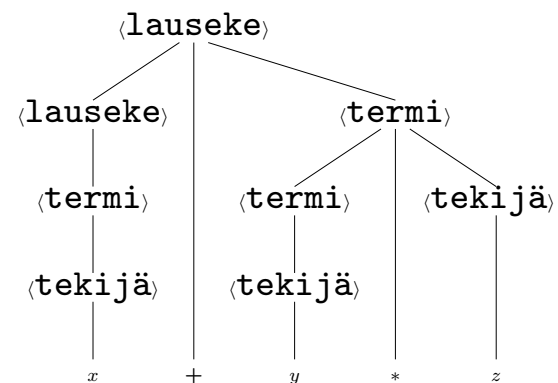


Sama kieli voidaan tuottaa seuraavalla yksiselitteisellä kieliopilla

$$\begin{aligned}\langle \text{lauseke} \rangle &\rightarrow \langle \text{lauseke} \rangle + \langle \text{termi} \rangle \mid \langle \text{termi} \rangle \\ \langle \text{termi} \rangle &\rightarrow \langle \text{termi} \rangle * \langle \text{tekijä} \rangle \mid \langle \text{tekijä} \rangle \\ \langle \text{tekijä} \rangle &\rightarrow (\langle \text{lauseke} \rangle) \mid x \mid y \mid z.\end{aligned}$$

Lauseen $x + y * z$ yksikäsitteiset vasen johto ja jäsennyspuu ovat

$$\begin{aligned}\langle \text{lauseke} \rangle &\Rightarrow \langle \text{lauseke} \rangle + \langle \text{termi} \rangle \\ &\Rightarrow \langle \text{termi} \rangle + \langle \text{termi} \rangle \\ &\Rightarrow \langle \text{tekijä} \rangle + \langle \text{termi} \rangle \\ &\Rightarrow x + \langle \text{termi} \rangle \\ &\Rightarrow x + \langle \text{termi} \rangle * \langle \text{tekijä} \rangle \\ &\Rightarrow x + \langle \text{tekijä} \rangle * \langle \text{tekijä} \rangle \\ &\Rightarrow x + y * \langle \text{tekijä} \rangle \\ &\Rightarrow x + y * z\end{aligned}$$



Aritmeettisen lausekkeen jäsenyspuun avulla voidaan helposti laskea lausekkeen arvo, kun muuttujien arvot tunnetaan. Yleisemmin kääntäjä voi jäsenyspuun avulla **generoida koodia** lausekkeen evaluoimiseksi. Tätä sovellusta silmällä pitäen edellisen sivun kielioppi noudattaa koulusta tuttua presedenssisääntöä, jonka mukaan kertolaskut lasketaan ennen yhteenlaskuja.

Jäsenyspuun hyödyntämiseksi pitää tietysti ensinnä osata muodostaa annetulle merkkijonolle jäsenyspuu (eli yhtäpitävästi johto) annetussa kieliopissa, tai todeta, että merkkijono ei kuulu kieleen. Palaamme jatkossa lyhyesti tähän yhteydettömän kieliopin **jäsenysoongelmaan**. Todetaan tässä vaiheessa, että ongelma on ei-triviaali, ja tehokkaat jäsenysmenetelmät jäävät tämän kurssin ulkopuolelle.

Todetaan vielä, että joillekin yhteydettömille kielille **ei ole olemassa** yksiselitteistä kielioppia. Esimerkki tällaisesta **luonnostaan moniselitteisestä** (inherently ambiguous) kielestä on $\{ a^i b^j c^k \mid i = j \text{ tai } j = k \}$ (todistus sivuutetaan).

Chomskyn normaalimuoto [Sipser s. 108–111]

Kielioppeja algoritmisesti käsiteltäessä on hyvä, jos ne ovat "siistejä".

Määritelmä 2.8: Yhteydetön kielioppi (V, Σ, R, S) on Chomskyn normaalimuodossa, jos joukon R jokainen sääntö on jotain seuraavista muodoista:

1. $A \rightarrow BC$, missä $A, B, C \in V$ ja $B \neq S$ ja $C \neq S$,
2. $A \rightarrow a$, missä $A \in V$ ja $a \in \Sigma$, tai
3. $S \rightarrow \varepsilon$.

Normaalimuodosta seuraa erityisesti, että jos $S \xRightarrow{*} w$ ja $|w| = n > 0$, niin johdon pituus on tasan $2n - 1$.

Lause 2.9: [Sipser Thm. 2.9] Mikä tahansa yhteydetön kieli voidaan tuottaa Chomskyn normaalimuodossa olevalla yhteydettömällä kieliopilla.

Tehdään todistus kahdessa vaiheessa. Tarkastellaan ensin ε -sääntöjen poistamista.

Lemma 2.10: Mikä tahansa yhteydetön kieli voidaan tuottaa yhteydettömällä kieliopilla, jossa

1. lähtösymboli ei esiinny minkään säännön oikealla puolella ja
2. kieliopissa ei ole sääntöä $A \rightarrow \varepsilon$ millään A , joka **ei ole** lähtösymboli.

Todistus: Olkoon $G = (V, \Sigma, R, S)$ yhteydetön kielioppi. Muodostamme ehdot toteuttavan kieliopin $G' = (V', \Sigma, R', S_0)$, jolla $L(G') = L(G)$.

Otetaan käyttöön uusi lähtösymboli $S_0 \notin V$ ja asetetaan $V' = V \cup \{S_0\}$. Laitetaan joukkoon R' sääntö $S_0 \rightarrow S$. Jos $\varepsilon \in L(G)$, lisätään myös sääntö $S_0 \rightarrow \varepsilon$. Muuttujalle S_0 ei tule muita sääntöjä, eikä se tule minkään säännön oikealle puolelle.

Määritellään nyt **nollautuvien** muuttujien joukko

$$\text{Null} = \left\{ A \in V \mid A \xrightarrow{*} \varepsilon \right\}.$$

Joukko Null voidaan helposti laskea iteratiivisella algoritmilla.

Perusidea on, että jos $B \in \text{Null}$, niin sääntö $A \rightarrow uBv$ korvataan säännöillä

$$A \rightarrow uv \mid uBv.$$

Vastaavasti jos $B, C \in \text{Null}$, niin sääntö $A \rightarrow uBvCw$ korvataan säännöillä

$$A \rightarrow uvw \mid uvCw \mid uBvw \mid uBvCw,$$

jne. Tämän jälkeen ε -säännöt ovat jääneet turhiksi ja voidaan poistaa.

Käydään siis läpi kaikki joukon R säännöt $A \rightarrow w$. Kirjoitetaan w muotoon

$$w = u_1 A_1 u_2 A_2 \dots u_k A_k u_{k+1},$$

missä kukin A_i on nollautuva muuttuja ja toisaalta mikään u_i ei sisällä nollautuvia muuttujia. Jos w ei sisällä nollautuvia muuttujia, niin $k = 0$. Lisätään nyt joukkoon R' kaikki 2^k sääntöä

$$A \rightarrow u_1 x_1 u_2 x_2 \dots u_k x_k u_{k+1},$$

missä x_i on joko A_i tai ε , **paitsi** ei sääntöä $A \rightarrow \varepsilon$ mikäli se esiintyy.

On helppo nähdä, että saatu kielioppi toteuttaa vaaditut ehdot. \square

Esimerkki 2.11: Sovelletaan konstruktiota kielioppiin

$$\begin{aligned} S &\rightarrow BSB \mid A \\ A &\rightarrow aA \mid aa \\ B &\rightarrow bB \mid \varepsilon. \end{aligned}$$

Nyt $\text{Null} = \{B\}$. Siis säännöstä $S \rightarrow BSB$ saadaan

$$S \rightarrow S \mid SB \mid BS \mid BSB,$$

ja säännöstä $B \rightarrow bB$ saadaan

$$B \rightarrow b \mid bB.$$

Lisätään vielä alkusymboli, ja otetaan mukaan kaikki vanhat säännöt lukuunottamatta ε -sääntöjä. Nyt $S \not\stackrel{*}{\rightarrow} \varepsilon$, joten kieliopiksi tulee

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow S \mid SB \mid BS \mid BSB \mid A \\ A &\rightarrow aA \mid aa \\ B &\rightarrow b \mid bB. \end{aligned}$$

□

Seuraava vaihe on poistaa yksikkösäännöt eli säännöt muotoa $A \rightarrow B$, missä $A, B \in V$.

Lemma 2.12: Mikä tahansa yhteydetön kieli voidaan tuottaa yhteydettömällä kieliopilla, jossa ei ole yksikkösääntöjä.

Todistus: Kaikilla $A \in V$ olkoon $\text{Unit}(A)$ niiden muuttujien joukko, jotka voidaan johtaa A :sta pelkillä yksikkösäännöillä, A itse mukaanlukien. Joukot $\text{Unit}(A)$ on helppo laskea iteratiivisesti.

Kieliopin muuttujat ja päätesymbolit pidetään ennallaan, samoin lähtösymboli. Sääntöihin laitetaan $A \rightarrow w$ kaikilla $A \in V$ ja $w \in (V \cup \Sigma)^*$, joilla

1. $B \in \text{Unit}(A)$,
2. $B \rightarrow w$ on sääntö alkuperäisessä kieliopissa ja
3. w ei ole muuttujasymboli.

Ehtojen 1 ja 2 perusteella on selvää, että uusien sääntöjen joukkoon ei tule mitään, mitä siellä ei saisi olla. Jos $B \in \text{Unit}(A)$ ja $B \Rightarrow w$, niin $A \xRightarrow{*} w$, joten $A \rightarrow w$ voidaan lisätä.

Ainoa ongelma näyttäisi olevan, että ehdon 3 nojalla pudotetaan pois muotoa $B \rightarrow C$ olevat säännöt. Kuitenkin jos $B \in \text{Unit}(A)$ ja $B \rightarrow C$ on sääntö, niin $C \in \text{Unit}(A)$, joten vuorostaan muotoa $C \rightarrow w$ olevat säännöt tuovat mukaan uudet säännöt $A \rightarrow w$. Siis uudet säännöt tuottavat samat merkkijonot kuin vanhat. \square

Esimerkki 2.13: Jatketaan edellisen esimerkin lopputuloksesta

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow S \mid SB \mid BS \mid BSB \mid A \\A &\rightarrow aA \mid aa \\B &\rightarrow b \mid bB.\end{aligned}$$

Nyt $\text{Unit}(S_0) = \{S_0, S, A\}$, $\text{Unit}(S) = \{S, A\}$, ja $\text{Unit}(X) = \{X\}$ muuten. Siis muuttujalle S_0 tulee omien alkuperäisten lisäksi kaikki muuttujien S ja A ei-yksikkösäännöt:

$$S_0 \rightarrow SB \mid BS \mid BSB \mid aA \mid aa.$$

Samoin S :lle lisätään A :n säännöt:

$$S \rightarrow SB \mid BS \mid BSB \mid aA \mid aa.$$

Ottamalla vielä kaikki vanhat ei-yksikkösäännöt saadaan kielioppi

$$\begin{aligned}S_0 &\rightarrow SB \mid BS \mid BSB \mid aA \mid aa \\S &\rightarrow SB \mid BS \mid BSB \mid aA \mid aa \\A &\rightarrow aA \mid aa \\B &\rightarrow b \mid bB.\end{aligned}$$

□

Lauseen 2.9 todistus: Olkoon $G = (V, \Sigma, R, S)$ yhteydetön kielioppi. Muodostamme Chomskyn normaalimuodossa olevan kieliopin, joka tuottaa saman kielen. Lemmojen 2.10 ja 2.11 nojalla voimme olettaa kieliopista G , että

1. S ei esiinny minkään säännön oikealla puolella,
2. $A \rightarrow \varepsilon \notin R$ kaikilla $A \in V - \{S\}$, ja
3. yksikkösääntöjä ei ole.

Siis sääntö $A \rightarrow u_1 \dots u_k$, missä $u_i \in V \cup \Sigma$, voi rikkoa normaalimuotoa kahdella tavalla:

1. joko $k = 2$ ja jokin u_i on päätesymboli tai
2. $k \geq 3$.

Otetaan jokaista päätesymbolia $a \in \Sigma$ kohti käyttöön uusi muuttuja X_a ja lisätään sääntö $X_a \rightarrow a$.

Korvataan jokainen normaalimuotoa rikkova sääntö $A \rightarrow u_1u_2$ säännöllä $A \rightarrow U_1U_2$, missä $U_k = X_a$ jos $u_k = a \in \Sigma$ ja $U_k = u_k$ jos $u_k \in V$.

Sääntö $A \rightarrow u_1 \dots u_k$, missä $k \geq 3$, korvataan $k - 1$ uudella säännöllä

$$\begin{aligned} A &\rightarrow U_1A_2 \\ A_2 &\rightarrow U_2A_3 \\ &\dots \\ A_{k-2} &\rightarrow U_{k-2}A_{k-1} \\ A_{k-1} &\rightarrow U_{k-1}U_k \end{aligned}$$

missä A_2, \dots, A_{k-1} ovat uusia muuttujia ja U_k kuten edellä. \square

Esimerkki 2.14: Jatketaan edellistä esimerkkiä

$$\begin{aligned}S_0 &\rightarrow SB \mid BS \mid BSB \mid aA \mid aa \\S &\rightarrow SB \mid BS \mid BSB \mid aA \mid aa \\A &\rightarrow aA \mid aa \\B &\rightarrow b \mid bB.\end{aligned}$$

Kielioppi tulee muotoon

$$\begin{aligned}S_0 &\rightarrow SB \mid BS \mid BA_1 \mid X_aA \mid X_aX_a \\A_1 &\rightarrow SB \\S &\rightarrow SB \mid BS \mid BA_2 \mid X_aA \mid X_aX_a \\A_2 &\rightarrow SB \\A &\rightarrow X_aA \mid X_aX_a \\B &\rightarrow b \mid X_bB \\X_a &\rightarrow a \\X_b &\rightarrow b.\end{aligned}$$

□

Yhteydettömän kieliopin jäsenysongelma

Yhteydettömän kieliopin jäsenysongelmalla tarkoitetaan laskentaongelmaa

Annettu: yhteydetön kielioppi G , merkkijono w

Kysymys: päteekö $w \in L(G)$.

Ongelma voidaan periaatteessa ratkaista seuraavasti:

1. Muodosta Chomskyn normaalimuodossa oleva G' , jolla $L(G') = L(G)$.
2. Jos $w = \varepsilon$ ja G' sisältää säännön $S \rightarrow \varepsilon$, vastaa **kyllä**.
3. Muuten luettele kaikki pituudeltaan $2|w| - 1$ olevat kieliopin G' johdot.
Jos jokin näistä on johto merkkijonolle w , vastaa **kyllä**;
muuten vastaa **ei**.

Ratkaisu perustuu havaintoon, että Chomskyn normaalimuodossa yhden päätemerkin tuottaminen vaatii tasan kaksi askelta. Koska pituutta $2|x| - 1$ olevia johtoja voi olla **paljon**, menetelmä on käytännössä liian tehoton.

Ohjelmointikielen kääntäminen

Vaiheet periaatteellisella tasolla:

1. Selaaminen (scanning, lexical analysis)
 - jakaa syötteen **tekstialkioiksi** (token); esim. muuttujanimi, liukulukuvakio
 - tekniikka DFA tai vastaava
2. jäsentäminen (parsing)
 - muodostaa tekstialkioille jäsennykspuun
 - perustuu yhteydettömään kielioppiin
3. semanttinen analyysi: tyyppitys jne.
4. koodin tuottaminen ja optimointi

Käytännössä vaiheet menevät ajallisesti limittäin; esim. missään vaiheessa koneen muistissa ei ole ohjelman koko jäsennykspuuta.

Ohjelmointikielen jäsentämiseen on useita tekniikoita. Lähtökohtana on tyypillisesti ohjelmointikielen yhteydetön kielioppi.

Tehokkaat jäsennessmenetelmät edellyttävät, että kielioppi on jossain rajoitetussa muodossa (tärkeimmät $LL(k)$ ja $LR(k)$). Emme tällä kurssilla puutu näihin menetelmiin.

Millä tahansa yhteydettömällä kieliopilla G kysymys "päteekö $w \in L(G)$?" voidaan ratkaista ajassa $O(|w|^3)$ **CYK-algoritmilla** (Cocke, Younger, Kasami). Esitämme algoritmin lyhyesti (huom. kurssikirjassa sivulla 267). Voidaan olettaa, että $G = (V, \Sigma, R, S)$ on Chomskyn normaalimuodossa.

Olkoon $w = w_1 \dots w_n \in \Sigma^n$. Perusajatus on muodostaa kaikilla $1 \leq i \leq j \leq n$ joukko

$$\text{table}(i, j) = \left\{ A \in V \mid A \xRightarrow{*} w_i \dots w_j \right\}.$$

Keskeinen havainto: jos $A \xRightarrow{*} w_i \dots w_j$, niin joko

1. $i = j$ ja $A \Rightarrow w_i$, tai
2. $i < j$ ja $A \Rightarrow BC \xRightarrow{*} w_i \dots w_j$.

Tapauksessa 1 pätee $A \rightarrow w_i \in R$.

Tapauksessa 2 pätee $A \rightarrow BC \in R$, ja lisäksi $B \xRightarrow{*} w_i \dots w_k$ ja $C \xRightarrow{*} w_{k+1} \dots w_j$ jollain $i \leq k \leq j - 1$.

Taulukko table annetulle w voidaan siis muodostaa seuraavasti:

alusta $\text{table}(i, j) := \emptyset$ kaikilla $1 \leq i \leq j \leq n$.

for $i := 1$ **to** n **do**

for $A \in V$ **do**

if $A \rightarrow w_i \in R$ **then** $\text{table}(i, i) := \text{table}(i, i) \cup \{A\}$

for $l := 2$ **to** n **do**

% laske joukot $\text{table}(i, j)$ missä $l = |w_i \dots w_j| = j - i + 1$

for $i := 1$ **to** $n - l + 1$ **do**

$j := i + l - 1$

for $k := i$ **to** $j - 1$ **do**

for $A \rightarrow BC \in R$ **do**

if $B \in \text{table}(i, k)$ **and** $C \in \text{table}(k + 1, j)$

then $\text{table}(i, j) := \text{table}(i, j) \cup \{A\}$

Joukot $\text{table}(i, j)$ kaikille $1 \leq i \leq j \leq n$ voidaan siis annetulle $w = w_1 \dots w_n$ muodostaa ajassa $O(n^3)$, kun kieliopin koko ajatellaan vakioksi.

Nyt

1. jos $n = 0$, niin $w \in L(G)$, jos ja vain jos $S \rightarrow \varepsilon \in R$, ja
2. jos $n \geq 1$, niin $w \in L(G)$, jos ja vain jos $S \in \text{table}(1, n)$.

Korollari 2.15: [Sipser Thm. 7.16] Millä tahansa yhteydettömällä kielellä L kysymys "pätee $x \in L$?" voidaan ratkaista ajassa $O(|x|^3)$. \square

Pinoautomaatit (pushdown automaton, PDA) [Sipser luku 2.2]

Pinoautomaatti on äärellinen automaatti, johon on lisätty rajoittamaton määrä muistia **pinon** (stack) muodossa. Osoitamme jatkossa, että kielen voi tunnistaa pinoautomaatilla, jos ja vain jos se on yhteydetön.

Esimerkki 2.16: Kielen $\{0^n1^n \mid n \in \mathbb{N}\}$ voi hyväksyä pinon avulla seuraavaan tapaan:

- 1:** jos seuraava merkki on 0 niin
Push(0)
mene kohtaan 1
- muuten**
Pop
mene kohtaan 2
- 2:** jos seuraava merkki on 1 niin
Pop
mene kohtaan 2

Automaatti hyväksyy, jos pino on tyhjä kun merkkijono loppuu. \square

Ryhdyimme nyt formalisoimaan pinoautomaattia. Kuten äärellisessä automaatissa, pinoautomaatissa on

- äärellinen tilajoukko Q ,
- syöteaakkosto Σ ,
- siirtymäfunktio δ ,
- alkutila $q_0 \in Q$ ja
- hyväksyvät tilat $F \subseteq Q$.

Lisäksi siinä on **pinoaakkosto** Γ , jonka alkioita voidaan tallentaa pinoon.

Tarkastelemme alusta alkaen **epädeterministisiä** pinoautomaatteja.

Siirtymäfunktio on nyt funktio $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ (muistetaan, että $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ ja \mathcal{P} tarkoittaa potenssijoukkoa).

Perustulkinta: jos $(q', c) \in \delta(q, a, b)$, niin tilassa q voidaan lukea syötemerkki a , poimia pinon päältä merkki b ja siirtyä tilaan q' . Samalla pinoon painetaan c .

Lisätulkinta siirtymälle $(q', c) \in \delta(q, a, b)$:

- jos $a = \varepsilon$, niin seuraavaa syötemerkkiä ei lueta;
- jos $b = \varepsilon$, niin pinosta ei poimita (eikä lueta) mitään; ja
- jos $c = \varepsilon$, niin pinoon ei paineta mitään.

Muodollisesti pinoautomaatti $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ hyväksyy merkkijonon $w \in \Sigma^*$, jos jollain $m \in \mathbb{N}$ on olemassa

- $w_1 \in \Sigma_\varepsilon, \dots, w_m \in \Sigma_\varepsilon$, joilla $w = w_1 \dots w_m$;
- tilat $r_0 \in Q, \dots, r_m \in Q$ ja
- merkkijonot $s_0 \in \Gamma^*, \dots, s_m \in \Gamma^*$ ("pinon sisällöt"),

missä

- $r_0 = q_0$ ja $s_0 = \varepsilon$ (aluksi pino tyhjä)
- kaikilla $i = 0, \dots, m - 1$ voidaan kirjoittaa $s_i = at$ ja $s_{i+1} = bt$, missä $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, ja
- $r_m \in F$.

Tuttuun tapaan automaatti **hylkää** merkkijonot, joita se ei hyväksy. Automaatin **tunnistama kieli** koostuu sen hyväksymistä merkkijonoista.

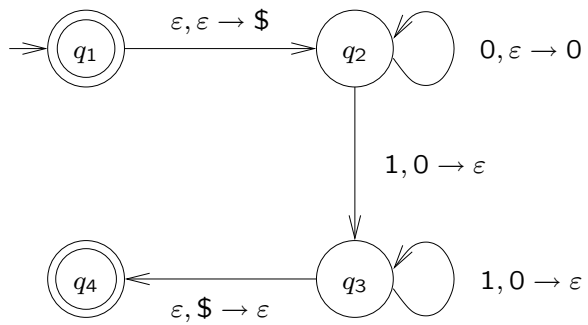
Esimerkki 2.17: [Sipser Ex. 2.14] Kieli $\{0^n 1^n \mid n \in \mathbf{N}\}$ voidaan tunnistaa pinoautomaatilla $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$, missä $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4\}$ ja δ on seuraava:

syöte:	0			1			ϵ		
pino:	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
q_3				$\{(q_3, \epsilon)\}$			$\{(q_4, \epsilon)\}$		
q_4									

Siis esim. $\delta(q_3, \epsilon, \$) = \{(q_4, \epsilon)\}$.

Koska formalismissa ei ole sisäänrakennettua pinon tyhjiystestiä, pinon pohjan merkiksi laitetaan **\$**. Sama tekniikka toistuu jatkossa.

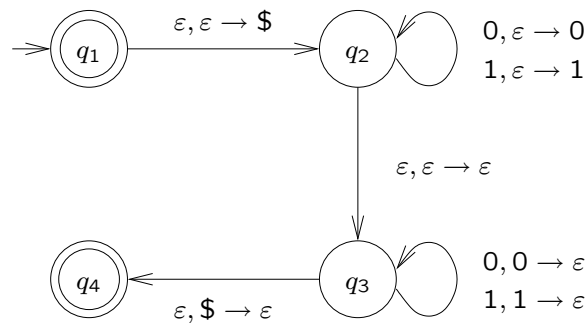
Esitetään sama havainnollisemmin tilakaaviona. Merkintä $q \xrightarrow{a,b \rightarrow c} q'$ tarkoittaa $(q', c) \in \delta(q, a, b)$.



Esimerkki 2.18: [Sipser Ex. 2.18] Kieli $\{ ww^R \mid w \in \{0, 1\}^* \}$ voidaan tunnistaa seuraavalla periaatteella:

1. Syötteen ensimmäisen puolikkaan ajan paina merkkejä pinoon.
2. Syötteen toisen puolikkaan ajan poimi merkkejä pinosta ja vertaa juuri luettuun.

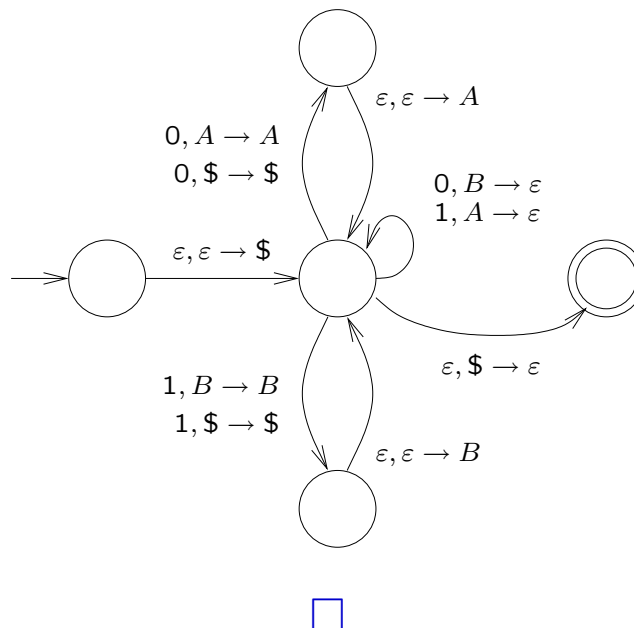
Syötteen keskikohta arvataan epädeterministisesti. Itse asiassa tätä kieltä ei edes ole mahdollista tunnistaa PDA:n deterministisellä variantilla.



□

Esimerkki 2.19: Seuraava PDA hyväksyy ne merkkijonot, joissa on yhtä monta nollaa kuin ykköistä.

Pinoaakkosto on $\Gamma = \{A, B, \$\}$. Pinossa pidetään kirjaa tähän mennessä luettujen nollien ja ykkösten määrien erosta. Esim. $AAA\$$ tarkoittaa, että nolliä on ollut kolme enemmän, ja $BBBBB\$$, että ykkösiä on ollut viisi enemmän.



Yhteydettömien kielioppien ja pinoautomaattien yhteys

[Sipser s. 117–124]

Todistamme, että yhteydettömien kielioppien tuottamat kielet ovat tasan samat kuin ne, jotka voidaan tunnistaa pinoautomaatilla. Aloitetaan helpommasta suunnasta.

Lemma 2.20: [Sipser Lemma 2.21] Jos kieli on yhteydetön, se voidaan tunnistaa pinoautomaatilla.

Todistus: Perusidea on laatia annetun kieliopin pohjalta pinoautomaatti, joka toteuttaa seuraavan algoritmin:

Generoi: Tuota epädeterministisesti pinoon merkkijono $w \in \Sigma^*$, jolla $S \xRightarrow{*} w$.

Testaa: Vertaa pinon merkkijono syötteeseen merkki kerrallaan. Jos löytyy ero, hylkää. Jos pino tyhjenee samaan aikaan, kun syöte loppuu, niin hyväksy.

Epädeterminismi on oleellista: valitsemalla generoimisvaiheessa sovellettavat säännöt epädeterministisesti varmistetaan, että jokaisella kieleen kuuluvalla merkkijonolla w on mahdollisuus tulla tuotetuksi.

Toteutusta rajoittaa, että automaatin tietorakenne on pino, josta vain huippu on kulloinkin näkyvässä. Siksi generointi- ja testausvaihe pitää lomittaa: aina kun pinon huipulle saadaan päätesymboleita, käydään vertaamassa niitä syötteeseen ennen generoinnin jatkamista.

Saadaan tarkennettu algoritmi:

1. Alusta pinon sisällöksi $S\$$, missä S on lähtösymboli ja $\$$ merkitsee pinon pohjaa.
2. Toista seuraavaa:
 - (a) Jos pinon huipulla on muuttujasymboli A , valitse epädeterministisesti sääntö $A \rightarrow w$. Korvaa A merkkijonolla w .
 - (b) Jos pinon huipulla on päätesymboli, poista se pinosta ja vertaa seuraavaan syötemerkkiin. Jos ne eroavat, hylkää.
 - (c) Jos pinon huipulla on $\$$, hyväksy jos syöte on loppu; muuten hylkää.

Jäljellä on enää em. algoritmin koodaaminen pinoautomaatiksi.

Jatkossa sallimme merkinnän

$$(r, u) \in \delta(q, a, s)$$

myös, kun $u = u_1 \dots u_l \in \Gamma^*$ (siis myös kun $l > 1$). Merkintä tarkoittaa

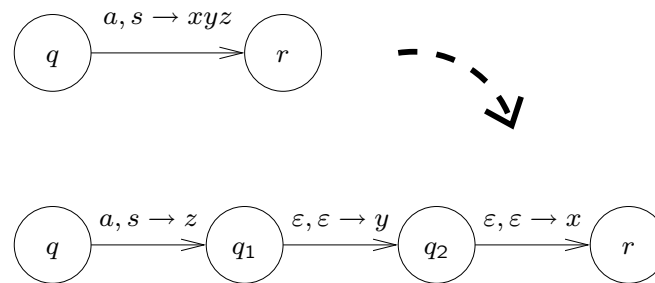
tilassa $q \in Q$ automaatti saa siirtyä tilaan $r \in Q$ lukemalla syötemerkin $a \in \Sigma$ ja poimimalla pinosta merkin $s \in \Gamma$ ja painamalla pinoon merkit u_l, \dots, u_1 .

Tapaukset $a = \varepsilon$ ja $s = \varepsilon$ sallitaan kuten ennenkin. Muodollisesti merkintä tarkoittaa, että automaatissa on tilat q_1, \dots, q_{l-1} , joilla

$$\begin{aligned} (q_1, u_l) &\in \delta(q, a, s) \\ \delta(q_1, \varepsilon, \varepsilon) &= \{ (q_2, u_{l-1}) \} \\ \delta(q_2, \varepsilon, \varepsilon) &= \{ (q_3, u_{l-2}) \} \\ &\dots \\ \delta(q_{l-1}, \varepsilon, \varepsilon) &= \{ (r, u_1) \} \end{aligned}$$

ja joihin ei liity muita siirtymiä.

Merkinnän $(r, xyz) \in \delta(q, a, s)$ kaavioesitys:



Olkoon nyt $G = (V, \Sigma, R, S)$ yhteydetön kielioppi. Konstruoimme pinoautomaatin, joka tunnistaa kielen $L(G)$.

Automaatin runkona on kolme tilaa q_{start} , q_{loop} ja q_{accept} . Alkutilana on q_{start} ja ainoa hyväksyvä tila q_{accept} .

Alustus hoidetaan siirtymällä

$$\delta(q_{\text{start}}, \varepsilon, \varepsilon) = (q_{\text{loop}}, S\$).$$

Tilasta q_{loop} lähtee kolmenlaisia siirtymiä:

- Kaikilla joukon R säännöillä $A \rightarrow w$ tulee $(q_{\text{loop}}, w) \in \delta(q_{\text{loop}}, \varepsilon, A)$.
- Kaikilla $a \in \Sigma$ tulee $(q_{\text{loop}}, \varepsilon) \in \delta(q_{\text{loop}}, a, a)$.
- Lopetusta varten $\delta(q_{\text{loop}}, \varepsilon, \$) = (q_{\text{accept}}, \varepsilon)$.

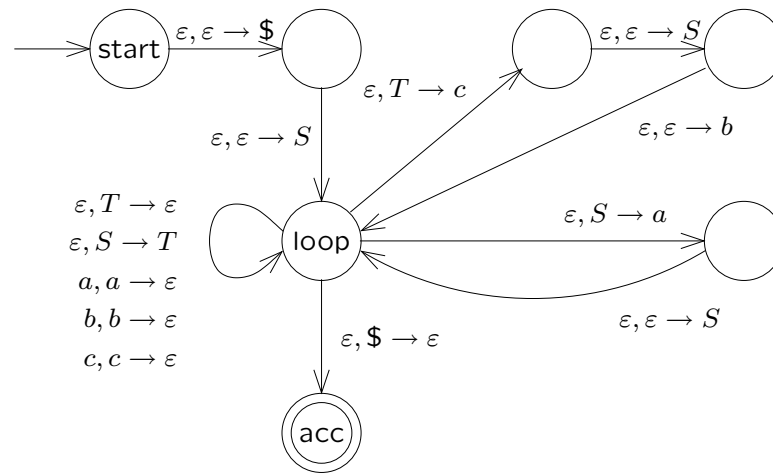
Muita siirtymiä ei ole. Selvästi automaatti toteuttaa esitetyn epädeterministisen algoritmin kielen $L(G)$ tunnistamiseksi. \square

Esimerkki 2.21: Soveltamalla konstruktiota kielioppiin

$$S \rightarrow Sa \mid T$$

$$T \rightarrow bSc \mid \varepsilon$$

saadaan seuraava pinoautomaatti:



Käänteinen suunta on hankalampi: meidän pitää osata "purkaa" mikä tahansa pinoautomaatti ja kuvata sen tunnistama kieli yhteydettömänä kielioppina.

Lemma 2.22: [Sipser Lemma 2.27] Pinoautomaatin tunnistama kieli on yhteydetön.

Todistus: Olkoon $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ pinoautomaatti. Tehdään tekninen lisäoletus, että

1. automaatissa on tasan yksi hyväksyvä tila (merkitään sitä q_{accept}),
2. ennen hyväksymistä M aina tyhjentää pinonsa ja
3. siirtymiä on vain seuraavia lajeja:

pop-siirtymä: poimii pinosta yhden merkin, ei paina mitään
($a, s \rightarrow \varepsilon$ missä $a \in \Sigma_\varepsilon$ ja $s \in \Gamma$) ja

push-siirtymä: painaa pinoon yhden merkin, ei poimi mitään
($a, \varepsilon \rightarrow s$ missä $a \in \Sigma_\varepsilon$ ja $s \in \Gamma$).

Ehdot 1 ja 2 saadaan helposti voimaan lisäämällä pinon loppumerkki ja ylimääräiset pinontyhjennystila ja hyväksyvä tila.

Ehto 3 saadaan voimaan korvaamalla kukin siirtymä

$a, s \rightarrow s'$ pop-push-yhdistelmällä ja kukin siirtymä

$a, \varepsilon \rightarrow \varepsilon$ push-pop-yhdistelmällä (yksi uusi tila per korjattava siirtymä).

Olkoot $p, q \in Q$ tiloja ja $w \in \Sigma^*$ syötemerkkijono. Merkitään $p \xrightarrow{w} q$, jos seuraavanlainen tapahtumaketju on mahdollinen:

- aluksi automaatti on tilassa p ja pino on tyhjä,
- siirtymät kuluttavat syötettä merkkijonon w verran ja
- lopuksi automaatti on tilassa q ja pino on taas tyhjä.

Tämä on yhtäpitävää sen kanssa, että seuraava on mahdollista millä tahansa $n \in \mathbf{N}$:

- aluksi automaatti on tilassa p ja pinossa on **tasan** n merkkiä,
- siirtymät kuluttavat syötettä merkkijonon w verran ja pinossa on koko ajan **ainakin** n merkkiä ja
- lopuksi automaatti on tilassa q ja pinossa on taas **tasan** n merkkiä.

Ehto nimittäin sanoo, että siirtyessään tilasta p syötteellä w tilaan q automaatti ei "kurki" pinon vanhoja sisältöjä, joten pinon alkutilanteella ei ole merkitystä.

Tavoitteena on määritellä kielioppi, jossa on muuttuja A_{pq} kaikilla $p, q \in Q$ ja

$$A_{pq} \xRightarrow{*} w \iff p \xrightarrow{w} q.$$

Tarkastellaan tapahtumaketjua, kun automaatti siirtyy tilasta p syötteellä w tilaan q siten, että pino on aluksi ja lopuksi tyhjä. Toinen seuraavista pätee:

Pino on ainakin kerran tyhjä myös alku- ja lopputilanteen välillä. Jos r on välitila, jossa pino on tyhjä, voidaan kirjoittaa $w = uv$, missä

$$p \xrightarrow{u} r \xrightarrow{v} q.$$

Pino ei välillä tyhjene. Tällöin tapahtumaketju alkaa jonkin merkin $t \in \Gamma$ painamisella pinoon (push-siirtymä), ja loppuu **saman merkin** poimimiseen pinosta (pop-siirtymä). Jos r ja s ovat ketjun toinen ja toiseksi viimeinen tila, niin jollain $a, b \in \Sigma_\varepsilon$ voidaan kirjoittaa $w = avb$, missä

$$p \xrightarrow{a, \varepsilon \rightarrow t} r \xrightarrow{v} s \xrightarrow{b, t \rightarrow \varepsilon} q.$$

Edellisen motivoimana määrittelemme kieliopin $G = (V, \Sigma, R, S)$ seuraavasti:

- $V = \{ A_{pq} \mid p, q \in Q \}$,
- Σ on sama kuin automaatin syöteaakkosto,
- $S = A_{q_0, q_{\text{accept}}}$ ja
- joukossa R on seuraavat säännöt:
 - $A_{pq} \rightarrow A_{pr}A_{rq}$ kaikilla $p, q, r \in Q$,
 - $A_{pq} \rightarrow aA_{rs}b$ aina kun $(r, t) \in \delta(p, a, \varepsilon)$ ja $(q, \varepsilon) \in \delta(s, b, t)$, ja
 - $A_{pp} \rightarrow \varepsilon$ kaikilla $p \in Q$.

Osoitamme konstruktion oikeellisuuden kahdessa osassa:

Väite A: Jos $A_{pq} \xRightarrow{*} w$, missä $w \in \Sigma^*$, niin $p \xrightarrow{w} q$.

Väite B: Jos $p \xrightarrow{w} q$, missä $w \in \Sigma^*$, niin $A_{pq} \xRightarrow{*} w$.

Automaattia M koskevien oletusten nojalla $w \in L(M)$, jos ja vain jos $q_0 \xrightarrow{w} q_{\text{accept}}$. Siis väitteistä A ja B seuraa $S \xRightarrow{*} w$, jos ja vain jos $w \in L(M)$, eli haluttu tulos.

Väitteen A todistus: Oletetaan $A_{pq} \xRightarrow{*} w$, missä $w \in \Sigma^*$. Osoitetaan induktiolla johdon pituuden suhteen, että $p \xrightarrow{w} q$.

Johdon pituus 0 ei ole mahdollinen.

Jos johdon pituus on 1, niin ehdon $w \in \Sigma^*$ takia ainoa mahdollinen johto on $A_{pp} \Rightarrow \varepsilon$ jollain p . Selvästi $p \xrightarrow{\varepsilon} p$ kaikilla p .

Tehdään induktio-oletus, että jollain $k \geq 1$ väite pätee, kun johdon pituus on korkeintaan k . Tarkastellaan johtoa $A_{pq} \xRightarrow{*} w$, jonka pituus on $k + 1$. Jakaudutaan kahteen tapaukseen sen mukaan, onko johdon ensimmäinen sääntö muotoa $A_{pq} \rightarrow A_{pr}A_{rq}$ vai $A_{pq} \rightarrow aA_{rs}b$.

Tapaus I: $A_{pq} \Rightarrow A_{pr}A_{rq} \xRightarrow{*} w$. Siis $A_{pr} \xRightarrow{*} u$ ja $A_{rq} \xRightarrow{*} v$, missä $w = uv$. Koska kummankin johdon pituus on korkeintaan k , induktio-oletuksen nojalla $p \xrightarrow{u} r$ ja $r \xrightarrow{v} q$. Siis $p \xrightarrow{w} q$.

Tapaus II: $A_{pq} \Rightarrow aA_{rs}b \xRightarrow{*} w$. Nyt $w = avb$, missä $A_{rs} \xRightarrow{*} v$. Induktio-oletuksen nojalla $r \xrightarrow{v} s$. Toisaalta $(A_{pq} \rightarrow aA_{rs}b) \in R$, joten $(r, t) \in \delta(p, a, \varepsilon)$ ja $(q, \varepsilon) \in \delta(s, b, t)$ jollain $t \in \Gamma$. Seuraava tapahtumaketju on siis mahdollinen:

- Aluksi automaatti on tilassa p ja pino on tyhjä.
- Automaatti tekee yhden siirtymän $p \xrightarrow{a, \varepsilon \rightarrow t} r$. Nyt automaatti on tilassa r ja pinon syvyys on 1.
- Automaatti siirtyy tilasta r syötteellä v tilaan s niin, että pinossa on koko ajan ainakin yksi merkki ja lopuksi tasan yksi merkki (nimittäin t).
- Nyt automaatti on tilassa s , ja pinon sisältö on t . Automaatti tekee siirtymän $s \xrightarrow{b, t \rightarrow \varepsilon} q$.
- Lopuksi automaatti on tilassa q ja pino tyhjä.

Siis $p \xrightarrow{w} q$. \square (Väite A)

Väitteen B todistus: Oletetaan $p \overset{w}{\rightsquigarrow} q$. Osoitetaan $A_{pq} \overset{*}{\Rightarrow} w$ induktiolla tapahtumaketjun $p \overset{w}{\rightsquigarrow} q$ tilasiirtymien lukumäärän suhteen.

Jos ketjussa on 0 tilasiirtymää, niin $p = q$ ja $w = \varepsilon$. Määritelmän mukaan $A_{pp} \Rightarrow \varepsilon$.

Oletetaan, että väite pätee k tai vähemmän siirtymiä sisältävillä ketjuilla. Olkoon $p \overset{w}{\rightsquigarrow} q$, missä ketjussa on $k + 1$ siirtymää.

Tapaus I: Tapahtumaketjun aikana pino tyhjenee ainakin kerran ennen loppua. Olkoon r tila ja u merkkijonon w alkuosa, jolla laskenta tilasta p tyhjällä pinolla ja syötteellä u johtaa tilaan r ja tyhjään pinoon. Siis $p \overset{u}{\rightsquigarrow} r$, ja lisäksi pätee $r \overset{v}{\rightsquigarrow} q$, missä $w = uv$. Induktio-oletuksen mukaan $A_{pr} \overset{*}{\Rightarrow} u$ ja $A_{rq} \overset{*}{\Rightarrow} v$. Koska $A_{pq} \rightarrow A_{pr}A_{rq} \in R$, niin $A_{pq} \Rightarrow A_{pr}A_{rq} \overset{*}{\Rightarrow} uv = w$.

Tapaus II: Tapahtumaketjussa pino ei tyhjene ennen loppua.

Siis viimeisessä siirtymässä pinosta poimitaan sama merkki $t \in \Gamma$, joka sinne ensimmäisessä siirtymässä työnnettiin.

Olkoot siirtymäketjun toinen tila r ja toiseksi viimeinen tila s . Siis joillain $a, b \in \Sigma_\varepsilon$ pätee $(r, t) \in \delta(p, a, \varepsilon)$ ja $(q, \varepsilon) \in \delta(s, b, t)$ ja siis $A_{pq} \rightarrow aA_{rs}b \in R$.

Lisäksi tilasta r päästään tilaan s siten, että pinossa oleva yksi merkki säilyy ennallaan, eli $r \overset{v}{\rightsquigarrow} s$ missä $w = avb$. Induktio-oletuksen nojalla $A_{rs} \overset{*}{\Rightarrow} v$. Siis $A_{pq} \Rightarrow aA_{rs}b \overset{*}{\Rightarrow} avb = w$. \square (Väite B ja Lemma 2.22)

Yhdistämällä Lemmat 2.20 ja 2.22 saadaan

Lause 2.23: [Sipser Thm. 2.20] Kieli on yhteydetön, jos ja vain jos se voidaan tunnistaa pinoautomaatilla. \square

Huomaa, että tästä **ei** seuraa, että osaisimme millä tahansa yhteydettömällä kielellä A ja merkkijonolla w ratkaista tehokkaasti, päteekö $w \in A$. Pinoautomaatin toiminnassa epädeterminismi on oleellista, ja vastauksen saamiseksi voidaan joutua kokeilemaan hyvin suurta määrää erilaisia laskentoja. Kuten sivulla 112 mainittiin, tehokkaat jäsenyysalgoritmit tekevät lisäoletuksia kieliopista.

Tilanne on siis tässä suhteessa erilainen kuin säännöllisten kielten ja äärellisten automaattien tapauksessa.

Voimme kuitenkin todeta seuraavan tuloksen (jonka voi todistaa helpomminkin):

Korollari 2.24: Säännölliset kielet ovat yhteydettömiä.

Todistus: Jos kieli voidaan tunnistaa äärellisellä automaatilla, se voidaan tietysti tunnistaa pinoautomaatilla. \square

Ei-yhteydettömät kielet [Sipser luku 2.3]

Yhteydettömille kielille pätee samantapainen pumppauslemma kuin säännöllisille kielille. Siinä kuitenkin pumpataan **kahta** osamerkkijonoa samaan tahtiin.

Lause 2.25 (Yhteydettömien kielten pumppauslemma): [Sipser Thm. 2.34] Jos A on yhteydetön kieli, niin sille on olemassa pumppauspituus $p \in \mathbb{N}$, jolle seuraava pätee:

jos $s \in A$ ja $|s| \geq p$

niin voidaan kirjoittaa $s = uvxyz$, missä

1. $uv^i xy^i z \in A$ kaikilla $i \in \mathbb{N}$,
2. $|vy| > 0$ ja
3. $|vxy| \leq p$.

Toisin kuin säännöllisten kielten tapauksessa, pumppauslemma on helpoin ymmärtää kieliopin ja jäsennyspuiden avulla, ei automaattien. Idea on seuraava:

Jos s kielen A merkkijono, sillä on jäsennyspuu sopivassa kielen A kieliopissa.

Jos lisäksi s on kovin pitkä, jäsennyspuussa on oltava ainakin yksi pitkä haara.

Kun jäsennyspuun haara on riittävän pitkä, ainakin yhden muuttujan R on pakko esiintyä ainakin kaksi kertaa (kyyhkyslakkaperiaate).

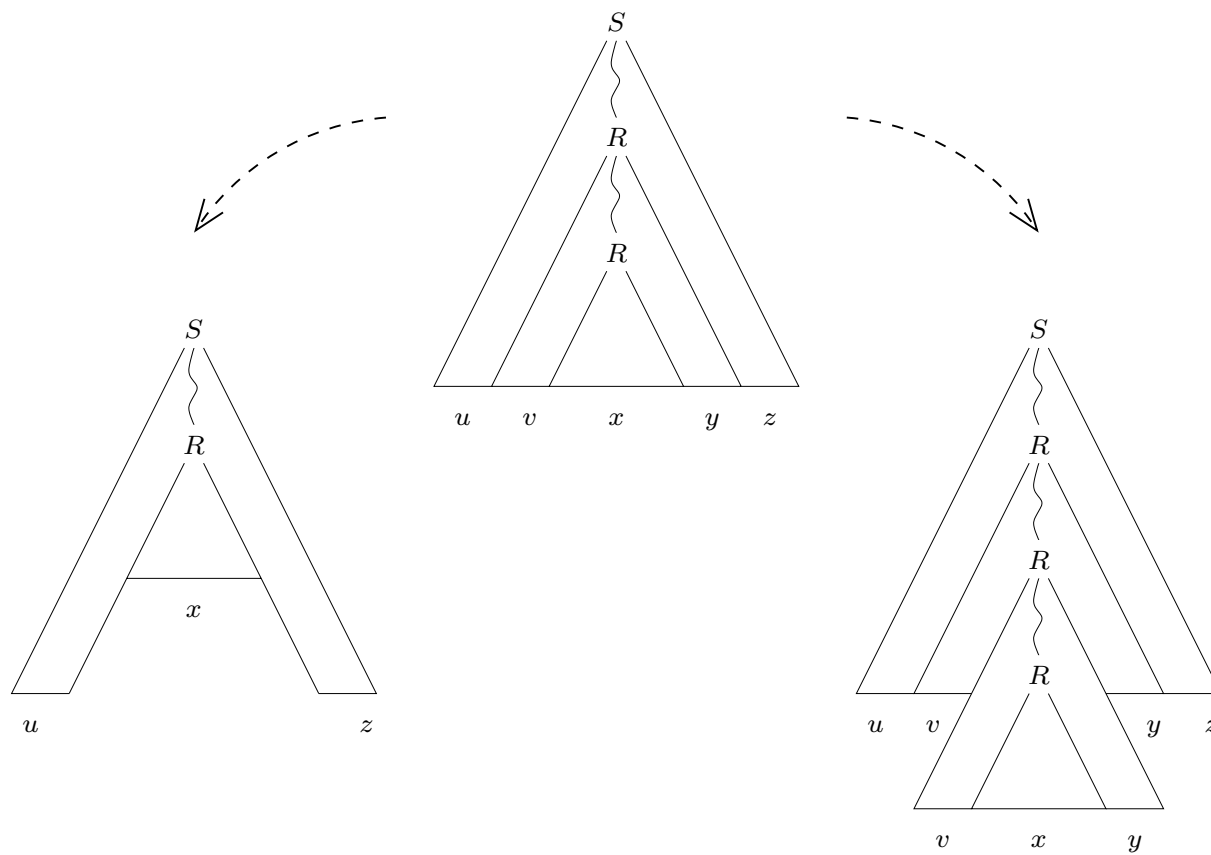
Jos asiaa ajatellaan johtojen kannalta, niin

$$S \xRightarrow{*} uRz \xRightarrow{*} uvRyz \xRightarrow{*} uvxyz,$$

missä sekä $R \xRightarrow{*} vRy$ että $R \xRightarrow{*} x$, ja $u, v, x, y, z \in \Sigma^*$. Näitä uudelleen yhdistelemällä saadaan johdot

$$\begin{aligned} S &\xRightarrow{*} uRz \xRightarrow{*} uxz \\ S &\xRightarrow{*} uRz \xRightarrow{*} uvRyz \xRightarrow{*} uvvRyyz \xRightarrow{*} uvvxyyz. \end{aligned}$$

Sama kuvana:



Lauseen 2.25 todistus: Edellä esitetystä ideasta täsmennetään ensin, mitä on "pitkä".

Olkoon $G = (V, \Sigma, R, S)$ yhteydetön kielioppi, jolla $A = L(G)$. Olkoon b suurin minkään säännön oikealla puolella olevien symbolien määrä. Siis jäsenyspuun millään solmulla ei ole yli b lasta. Jos jäsenyspuun syvyys on h , siinä on korkeintaan b^h lehteä. (Syvyys on kaarten lukumäärä pisimmällä polulla juuresta lehteen.)

Valitaan nyt $p = b^{|V|+1}$ ja oletetaan $s \in A$ ja $|s| \geq p$. Olkoon τ merkkijonon s jäsenyspuu; jos niitä on useita, valitaan vähiten solmuja sisältävä.

Valitaan puusta τ pisin polku juuresta lehteen. Edellisen perusteella polulla on enemmän kuin $|V|$ kaarta ja siis ainakin $|V| + 2$ symbolia. Näistä tasan yksi on päätesymboli, joten viimeisten $|V| + 2$ symbolin joukossa ainakin yksi muuttuja esiintyy ainakin kaksi kertaa. Olkoon R tällainen.

Nyt siis

$$S \xrightarrow{*} uRz \xrightarrow{*} uvRyz \xrightarrow{*} uvxyz,$$

joten edellä esitetyn päättelyn mukaisesti $uv^i xy^i z \in A$ kaikilla $i \in \mathbb{N}$ eli ehto (1) pätee.

Jos ehto (2) ei olisi voimassa, niin $v = y = \varepsilon$ eli em. johto on muotoa

$$S \xrightarrow{*} uRz \xrightarrow{+} uRz \xrightarrow{*} uvxyz.$$

Tässä on "turha" välivaihe $R \xrightarrow{+} R$. Tämä on ristiriita, koska τ oletettiin pienimmäksi mahdolliseksi.

Koska muuttujan R kaksi esiintymää ovat polun $|V| + 2$ alimman solmun joukossa ja polku oli puun pisin, niin ylempään R :n esiintymään juurtuva puu sisältää korkeintaan $b^{|V|+1}$ lehteä. Siis $|vxy| \leq b^{|V|+1} = p$ eli ehto (3) pätee. \square

Pumppauslemmaa sovelletaan samaan tapaan kuin säännöllisillä kielillä.

Esimerkki 2.26: [Sipser Ex. 2.36] Kieli $B = \{ a^n b^n c^n \mid n \in \mathbf{N} \}$ ei ole yhteydetön.

Vastaoletus: B on yhteydetön ja sillä on pumppauspituus p . Olkoon $s = a^p b^p c^p$. Siis $|s| \geq p$; olkoon $s = uvxyz$ missä $|vy| > 0$ ja $uv^i xy^i z \in B$ kaikilla $i \in \mathbf{N}$. Tarkastellaan kahta vaihtoehtoa:

1. Jos merkkijono vy sisältää kaikkia kolmea merkkiä a , b ja c , niin v tai y sisältää ainakin kahta eri merkkiä. Siis uv^2xy^2z sisältää merkkejä väärässä järjestyksessä.
2. Jos merkkijono vy ei sisällä jotakin merkkiä, niin uv^2xy^2z sisältää puuttuvaa/puuttuvia merkkiä vähemmän kuin mukana olevia.

Kummassakin tapauksessa $uv^2xy^2z \notin B$; ristiriita. \square

Korollari 2.27: Yhteydettömien kielten luokka ei ole suljettu leikkauksen eikä komplementin suhteen.

Todistus: Olkoon

$$B_1 = \{ a^m b^m c^n \mid m, n \in \mathbf{N} \} \quad \text{ja} \quad B_2 = \{ a^m b^n c^n \mid m, n \in \mathbf{N} \}.$$

Kielet B_1 ja B_2 ovat yhteydettömiä, mutta niiden leikkaus on edellisen esimerkin ei-yhteydetön kieli B . Siis yhteydettömien kielten luokka ei ole suljettu leikkauksen suhteen.

Tehdään nyt vastaoletus, että yhteydettömien kielten luokka on suljettu komplementin suhteen. Koska $A \cap B = \overline{\overline{A} \cup \overline{B}}$ ja yhteydettömien kielten luokka on suljettu yhdisteen suhteen, luokka on suljettu myös leikkauksen suhteen; ristiriita. \square

Esimerkki 2.28: [Sipser Ex. 2.37] Kieli $C = \{ a^i b^j c^k \mid 0 \leq i \leq j \leq k \}$ ei ole yhteydetön.

Vastaoletus: C on yhteydetön ja sillä on pumppauspituus p . Valitaan taas $s = a^p b^p c^p$ ja oletetaan $s = uvxyz$ kuten pumppauslemmassa. Koska $|vxy| \leq p$, niin vy ei voi sisältää sekä a:ta että c:tä. Kaksi tapausta:

1. Merkkijono vy ei sisällä yhtään a:ta. Nyt uxz sisältää b:tä tai c:tä vähemmän kuin a:ta; ristiriita.
2. Merkkijono vy sisältää ainakin yhden a:n mutta ei yhtään c:tä. Nyt uv^2xy^2z sisältää a:ta enemmän kuin c:tä; ristiriita.

□

Samantyyppinen päättely osoittaa, että kieli

$$\{ w \in \{ a, b, c \}^* \mid w \text{ sisältää yhtä monta a:ta, b:tä ja c:tä} \}$$

ei ole yhteydetön.

Esimerkki 2.29: [Sipser Ex. 2.38] Kieli $D = \{ ww \in \{0, 1\}^n \}$ ei ole yhteydetön.

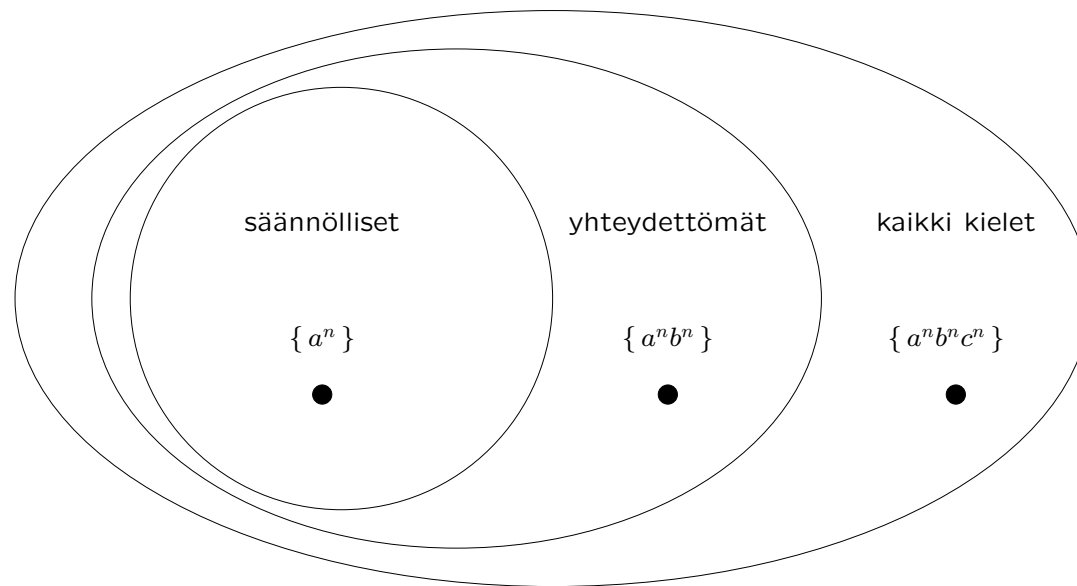
Vastaoletus: D on yhteydetön ja sillä on pumppauspituus p . Valitaan $s = 0^p 1^p 0^p 1^p \in D$. Koska $|s| \geq p$, voidaan kirjoittaa $s = uvxyz$ kuten pumppauslemmassa.

Jos vxy sisältyy kokonaan merkkijonon s alkupuoliskoon, niin $uxz = 0^i 1^j 0^p 1^p$, missä $i + j = 2p - |vy| < 2p$. Siis merkkijonon uxz alkupuolisko loppuu 0:aan, joten uxz ei ole muotoa ww ; ristiriita. Samoin nähdään, että vxy ei voi kokonaan sisältyä loppupuoliskoon.

Jäljelle jää mahdollisuus, että jonon $0^p 1^p 0^p 1^p = uvxyz$ keskikohta osuu pätkään uxz . Nyt $uxz = 0^p 1^i 0^j 1^p$, missä $i < p$ tai $j < p$. Taaskaan uxz ei ole muotoa ww ; ristiriita. \square

Tämän esimerkin perusteella voimme päätellä, että yhteydettömällä kieliopilla ei voi esittää sellaisia ohjelmointikielten rajoituksia kuin ”muuttuja pitää määritellä ennen käyttöä”.

Kurssi tähän asti: säännölliset ja yhteydettömät kielet



Kurssi lähti liikkeelle yksinkertaisista laskennan malleista: mitä niillä voidaan esittää, millaisia ilmiöitä havaitaan.

Laskennan formalisointi

- perustuu joukko-oppiin ja logiikkaan, mahdollistaa matemaattisen täsmälliset päättelyt
- edellyttää idealisointia (mielivaltaisen pitkät syötteet jne.)
- tärkeä erityiseikka: epädeterminismi

Proseduraalisuus ja deklarativisuus

- saman asian määrittely proseduraalisesti automaatilla tai deklarativisesti kieliopilla
- tärkeät ekvivalenssitulokset:
 - A tunnistettavissa DFA:lla $\Leftrightarrow A$ esitettävissä säänn. lausekkeena
 - A tunnistettavissa PDA:lla $\Leftrightarrow A$ tuotettavissa CFG:llä
- siirrytään tarpeen mukaan proseduraalisesta deklarativiseen esitykseen tai toisin päin

Ylärajat ongelmien vaikeudelle: esim. mikä tahansa säännöllinen kieli voidaan tuottaa CFG:llä

- todistukset tyypillisesti konstruktivisia (esim. $CFG \rightarrow PDA$)
- erikoistapaus konstruktioista: simulaatio (esim. $NFA \rightarrow DFA$)
- konstruktion oikeellisuus todistetaan induktiolla (jos ei muuten selvä)

Alarajat ongelmien vaikeudelle: esim. kieli $\{0^n1^n \mid n \in \mathbb{N}\}$ on "vaikeampi" kuin mihin äärellinen automaatti pystyy

- lähtökohtana formalisoitu laskentamalli
- todistetaan voimakas aputulos (esim. pumppauslemma) laskentamallin määritelmän perusteella
- sovelletaan aputulosta esimerkkitapauksiin (usein epäsuora todistus), laskentamallin yksityiskohdista ei enää tarvitse välittää
- teknisesti vaikeita (myös alan ammattilaisille)

Hierarkia: esim. säännölliset kielet ovat yhteydettömien kielten aito aliluokka

- simulaatio: säännölliset kielet ovat yhteydettömiä
- erottelu: kieli $\{0^n1^n \mid n \in \mathbf{N}\}$ on yhteydetön (yläraja), mutta ei säännöllinen (alaraja)

Siirrymme nyt tarkastelemaan **Turingin konetta**:

- edellä esitellyt ilmiöt toistuvat
- tekniset yksityiskohdat monimutkaisempia
- tulokset kiinnostavia, koska Turingin kone on malli ”oikealle” tietokoneelle (tai mille tahansa nykytietämyksen valossa mahdolliselle laskentalaitteelle)

3. Turingin koneet

Turingin kone on alkuaan matemaattisen logiikan tarpeisiin kehitelty laskennan malli. Tarkoituksena oli vangita mahdollisimman laajasti, millaisia asioita voidaan (periaatteessa) laskea ”mekaanisesti”. Malli on sittemmin osoittautunut sopivaksi myös ”oikeiden” tietokoneiden ymmärtämiseen.

Tämän luvun jälkeen opiskelija

- osaa esittää yksinkertaisia algoritmeja täsmällisesti käyttäen Turingin konetta ja sen muunnelmia,
- hieman monimutkaisemmille algoritmeille osaa kuvata periaatetasolla toteutuksen Turingin koneella ja
- tuntee Churchin-Turingin teesin ja osaa sen avulla selittää Turingin koneen yhteyden yleiseen algoritmin käsitteeseen.

Turingin koneen rajoituksia tarkastellaan lähemmin seuraavassa luvussa.

Turingin kone [Sipser luku 3.1]

Turingin kone (Turing machine, TM) on automaatti, jossa on rajoittamaton määrä muistia. Toisin kuin pinoautomaatissa, muistialkioita voi käsitellä mielivaltaisessa järjestyksessä.

Turingin koneen peruskomponentit ovat

1. äärellinen joukko **tiloja** (state), kuten DFA:ssa ja PDA:ssa,
2. rajoittamattoman pituinen **nauha** (tape), joka aluksi sisältää syötteen ja laskennan aikana toimii apumuistina ja
3. liikuteltava **nauhapää** (tape head, read/write head), joka osoittaa seuraavaksi vuorossa olevaa symbolia nauhalla.

Turingin koneen yksityiskohdat voidaan määritellä monella eri tavalla, ja laskentavoimaltaa samaan lopputulokseen voidaan päätyä myös aivan toisennäköisistä lähtökohdista. Otamme tässä perustaksi mahdollisimman yksinkertaisen mallin.

Kuten muutkin käsittelemämme automaattit, Turingin kone saa syötteenä merkkijonon, jonka sitten hyväksyy tai hylkää:

1. Aluksi syöte on nauhan alussa, ja nauhan loppu sisältää pelkkiä **tyhjämerkkejä** \sqcup (blank). Nauhapää osoittaa nauhan alkuun.
2. Yhdessä laskenta-askeleessa
 - kone lukee nauhapään alla olevan symbolin,
 - valitsee seuraavan tilan,
 - **kirjoittaa** nauhapään kohdalle uuden symbolin (vanha häviää) ja
 - siirtää nauhapäätä **vasemmalle tai oikealle**.
3. Tilojen joukossa on **hylkäävä** ja **hyväksyvä** lopputila, joihin päätyminen lopettaa laskennan välittömästi.

Jos laskenta ei koskaan päädy hyväksyvään tai hylkäävään tilaan, se on **silmukassa**.

Tarkastellaan esimerkkinä kielen $A = \{ a^n b^n c^n \mid n \in \mathbf{N} \}$ tunnistamista Turingin koneella. Periaate on seuraava:

1. Kela nauhaa oikealle, kunnes löytyy joko (yliviivaamaton) a, \sqcup tai (mahdollisesti yliviivattu) b tai c. Jos löytyi muuta kuin a, siirry kohtaan 5. Muuten viivaa yli löytynyt a.
2. Kela nauhaa oikealle, kunnes löytyy joko (yliviivaamaton) b, \sqcup tai (mahdollisesti yliviivattu) c. Jos löytyi \sqcup tai c, niin **hylkää**. Muuten viivaa yli löytynyt b.
3. Kela nauhaa oikealle, kunnes löytyy joko c tai \sqcup . Jos löytyi \sqcup , niin **hylkää**. Muuten viivaa yli löytynyt c.
4. Kela nauha alkuun ja siirry kohtaan 1.
5. Jos nauhalla on jäljellä yliviivaamattomia a-, b- tai c-symboleita, niin **hylkää**, muuten **hyväksy**.

Huomaa, että tähän ei pinoautomaatti pysty (kieli A ei ole yhteydetön).

Täsmällisemmin Turingin kone on seitsikko $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, missä

1. Q on äärellinen tilajoukko,
2. Σ on syöteaakkosto, joka **ei** sisällä tyhjäämerkkiä \sqcup ,
3. Γ on nauha-aakkosto, jolle $\sqcup \in \Gamma$ ja $\Sigma \subset \Gamma$,
4. $\delta: Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ on siirtymäfunktio, missä $Q' = Q - \{q_{\text{accept}}, q_{\text{reject}}\}$,
5. $q_0 \in Q$ on alkutila,
6. $q_{\text{accept}} \in Q$ on hyväksyvä tila ja
7. $q_{\text{reject}} \in Q$ on hylkäävä tila, jolla $q_{\text{reject}} \neq q_{\text{accept}}$.

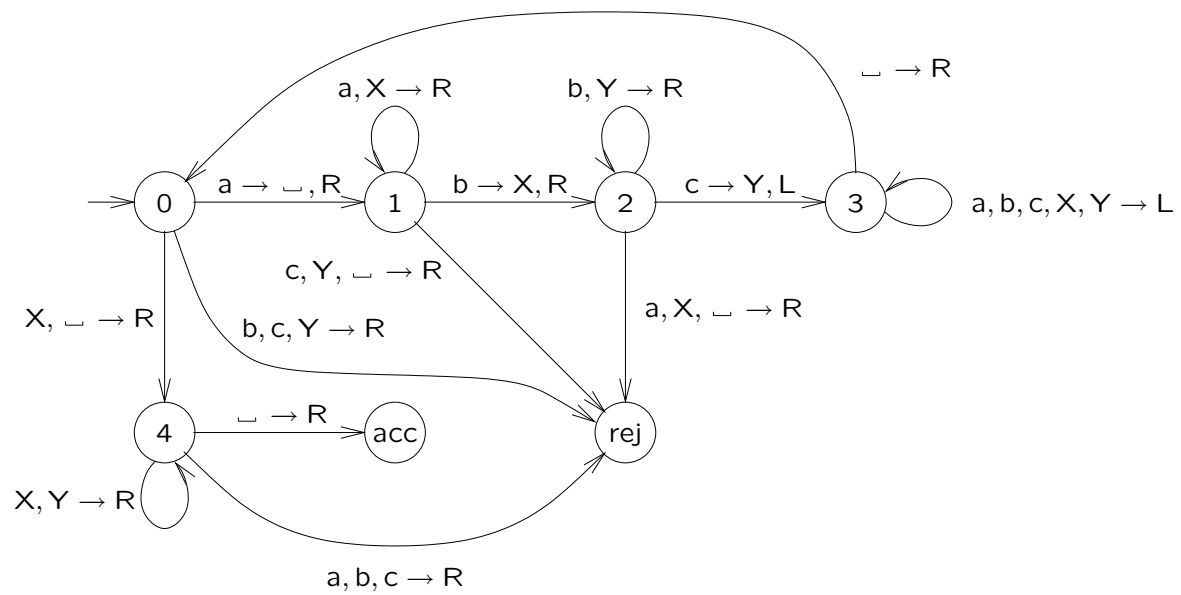
Perustulkinta: jos $\delta(q, a) = (q', b, D)$, niin tilasta q symbolillä a kone siirtyy tilaan q' , kirjoittaa symbolin b (joka korvaa vanhan symbolin a) ja siirtää nauhapäätä yhden askelen suuntaan D (missä L = vasen ja R = oikea).

Esitetään kielen $A = \{ a^n b^n c^n \mid n \in \mathbf{N} \}$ tunnistava kone tässä formalismissa. Syöteaakkosto on siis $\Sigma = \{ a, b, c \}$. Nauha-aakkostoksi valitsemme $\Gamma = \{ a, b, c, \sqcup, X, Y \}$, missä X ja Y esittävät yliviivattuja b - ja c -symboleja. Yliviivatut a :t esitetään tyhjällä \sqcup , mikä helpottaa nauhan alkukohdan käsittelyä.

Siirtymäfunktio voidaan esittää kaaviona, jossa

- $q \xrightarrow{a \rightarrow b, D} q'$ tarkoittaa $\delta(q, a) = (q', b, D)$ ja
- $q \xrightarrow{a, b, c, d \rightarrow D} q'$ tarkoittaa $\delta(q, x) = (q', x, D)$ kun $x \in \{ a, b, c, d \}$.

Kaaviota voidaan yksinkertaistaa jättämällä pois tila q_{reject} ja siihen johtavat siirtymät, mutta tässä näin ei ole tehty. (Kaavio seuraavalla sivulla.)



Turingin koneen **tilanne** (configuration) koostuu kolmesta komponentista:

1. koneen tila,
2. nauhan sisältö ja
3. nauhapään sijainti.

Koodaamme tilanteen merkkijonoksi uqv , missä $q \in Q$, $u = u_1 \dots u_m \in \Gamma^*$ ja $v = v_1 \dots v_n \in \Gamma^*$ joillain $m, n \in \mathbb{N}$. Tulkinta:

1. kone on tilassa q ,
2. nauhan sisältö on $u_1 \dots u_m v_1 \dots v_n \sqcup \sqcup \dots$ ja
3. nauhapää osoittaa symbolia v_1 .

Koneen **alkutilanne** syötteellä w on q_0w .

Koneen **lopputilanteita** eli **pysähtymistilanteita** (halting configuration) ovat

- hyväksyvät tilanteet $uq_{\text{accept}}v$ ja
- hylkäävät tilanteet $uq_{\text{reject}}v$

kaikilla $u, v \in \Gamma^*$.

Jos tilanteesta uqv kone menisi seuraavaksi tilanteeseen $u'q'v'$, niin tilanne uqv **johtaa suoraan** (yields) tilanteen $u'q'v'$, mitä merkitään

$$uqv \vdash u'q'v'.$$

Esitetään siirtymäfunktion semantiikka tätä formalismia käyttäen. Olkoot $u, v \in \Gamma^*$ ja $a, b, c \in \Gamma$.

- Jos $\delta(q, a) = (q', b, R)$, niin $uqav \vdash ubq'v$.
- Edellisen erikoistapaus: jos $\delta(q, _) = (q', b, R)$, niin $uq \vdash ubq'$ (sillä uq on sama tilanne kuin $uq_$).
- Jos $\delta(q, a) = (q', b, L)$, niin $ucqav \vdash uq'cbv$.
- Jos $\delta(q, a) = (q', b, L)$, niin $qav \vdash q'bv$ (ts. jos nauhapää yrittäisi siirtyä alkukohdasta vasemmalle, se jää paikalleen).

Esimerkkinä sivun 157 koneen hyväksyntä syötteelle aabbcc:

q_0 aabbcc	⊢	⊃	q_1 abbcc		⊢	⊃	⊃	q_3 XXYY	
	⊢	⊃	a q_1 bbcc		⊢	⊃	q_3 ⊃	XXYY	
	⊢	⊃	aX q_2 bcc		⊢	⊃	⊃	q_0 XXYY	
	⊢	⊃	aXb q_2 cc		⊢	⊃	⊃	X q_4 XY	
	⊢	⊃	aX q_3 bYc		⊢	⊃	⊃	XX q_4 YY	
	⊢	⊃	a q_3 XbYc		⊢	⊃	⊃	XXY q_4 Y	
	⊢	⊃	q_3 aXbYc		⊢	⊃	⊃	XXYY q_4 ⊃	
	⊢	q_3 ⊃	aXbYc		⊢	⊃	⊃	XXYY⊃	q_{accept} ⊃
	⊢	⊃	q_0 aXbYc						
	⊢	⊃	⊃	q_1 XbYc					
	⊢	⊃	⊃	X q_1 bYc					
	⊢	⊃	⊃	XX q_2 Yc					
	⊢	⊃	⊃	XXY q_2 c					
	⊢	⊃	⊃	XX q_3 YY					
	⊢	⊃	⊃	X q_3 XY					

Jos on olemassa tilanteet $u_1q_1v_1, \dots, u_nq_nv_n$, missä $u_iq_iv_i \vdash u_{i+1}q_{i+1}v_{i+1}$ kaikilla $1 \leq i \leq n - 1$, sanomme että tilanne $u_1q_1v_1$ **johtaa** tilanteeseen $u_nq_nv_n$ ja merkitsemme

$$u_1q_1v_1 \vdash^* u_nq_nv_n.$$

Jos $q_0w \vdash^* uq_{\text{accept}}v$ joillain $u, v \in \Gamma^*$, niin kone **hyväksyy** merkkijonon w .

Jos $q_0w \vdash^* uq_{\text{reject}}v$ joillain $u, v \in \Gamma^*$, niin kone **hylkää** merkkijonon w .

Jos kone hyväksyy tai hylkää merkkijonon w , sanomme, että se **pysähtyy** (halts) syötteellä w . Muuten se **jää silmukkaan** (loops), mikä voi ilmetä mielivaltaisen monimutkaisena loputtomana tilannejonona.

Turingin koneen M tunnistama kieli on

$$L(M) = \{ w \in \Sigma^* \mid M \text{ hyväksyy } w:n \}.$$

Jos $A = L(M)$, kieli A on **Turing-tunnistettava** tai lyhyesti **tunnistettava** (recognizable). Historiallisista syistä Turing-tunnistettavia kieliä sanotaan myös **rekursiivisesti numeroituviksi** (recursively enumerable); nimitys liittyy vain epäsuorasti ohjelmointikielissä käytettävään rekursioon.

Huom. komplementti $\overline{L(M)}$ sisältää ne merkkijonot, joilla M hylkää tai jää silmukkaan.

Jos M pysähtyy kaikilla syötteillä, se on **ratkaisija** (decider). Ratkaisijoita sanotaan myös **totaalisiksi** Turingin koneiksi. Jos $A = L(M)$, missä M on ratkaisija, niin A on **Turing-ratkeava** tai lyhyesti **ratkeava** (decidable). Ratkeavia kieliä sanotaan myös **rekursiivisiksi** (recursive).

Esimerkki 3.1: Kieli $B = \{ww \mid w \in \{0,1\}^*\}$ voidaan tunnistaa seuraavalla periaatteella:

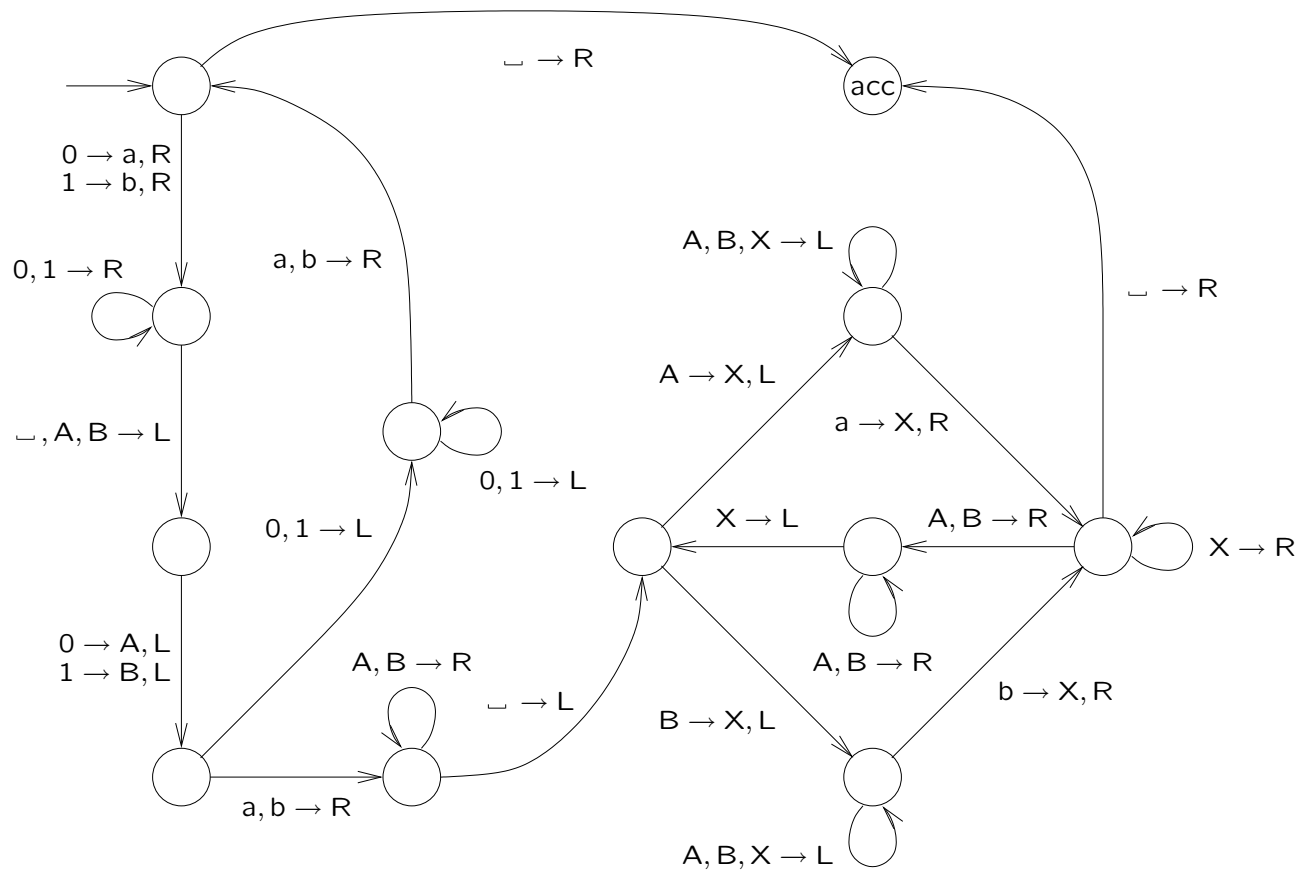
Vaihe I: Merkkijonon alkupuoliskossa korvaa $0 \mapsto a$ ja $1 \mapsto b$ ja loppupuoliskossa $0 \mapsto A$ ja $1 \mapsto B$. Esim. 00101100 tulee muotoon aabaBBAA. Parittoman mittaiset syötteen hylätään. Toteutus:

1. Vaihda nauhalta ensimmäinen 0- tai 1-symboli vastaavasti a:ksi tai b:ksi.
2. Vaihda nauhalta viimeinen 0- tai 1-symboli vastaavasti A:ksi tai B:ksi. Jos nollia tai ykkösiä ei löytynyt, **hylkää**.
3. Jos nollia tai ykkösiä vielä on, palaa kohtaan 1. Muuten siirry vaiheeseen II.

Vaihe II: Lopusta alkaen vertaa, että kutakin A-symbolia vastaa a-symboli ja B-symbolia vastaa b-symboli. Toteutus:

4. Etsi nauhan oikeanpuoleisin A tai B ja viivaa yli.
 - Jos se oli A, etsi oikeanpuoleisin a tai b. Jos se oli a, viivaa yli. Muuten hylkää.
 - Jos se oli B, etsi oikeanpuoleisin a tai b. Jos se oli b, viivaa yli. Muuten hylkää.
5. Jos A:t ja B:t loppuivat, **hyväksy**. Muuten palaa kohtaan 4.

Kaavioesitys em. koneelle (q_{reject} siirtymineen jätetty piirtämättä):



Kuten edellisestä esimerkistä havaitaan, Turingin koneista (kuten muistakin automaateista) tulee nopeasti hyvin monimutkaisia. Käytännössä esitämme koneet tilakaavion sijaan pseudokoodina.

Esimerkki 3.2: [Sipser Ex. 3.11] Tarkastellaan kertolaskun tarkastamista eli kielen

$$C = \{ a^i b^j c^k \mid i, j, k \geq 1 \text{ ja } k = i \cdot j \}$$

tunnistamista. Turingin kone toimii seuraavasti:

1. Selaa syöte vasemmalta oikealle ja tarkista, että se on muotoa $a^+b^+c^+$. Jos ei ole, niin **hylkää**.
2. Palauta nauhapää nauhan alkuun.
3. Viivaa yli vasemmanpuoleisin a ja selaa seuraavaan b:hen. Kela edestakaisin b- ja c-osien välillä merkaton aina yksi b ja yksi c kerrallaan. Jos c:t loppuvat, **hylkää**. Muuten kun b:t on kaikki merkattu, siirry kohtaan 4.
4. Poista merkit kaikista b:istä. Jos a:ita on jäljellä, mene kohtaan 3. Muuten jos kaikki c:t on merkattu, niin **hyväksy**, muuten **hylkää**.

Joitain toteutusteknisiä yksityiskohtia:

- Kohta 1 on helppo, koska koneen riittää toimia kuin äärellinen automaatti.
- Kohtaa 2 varten nauhan alkukohta pitää merkata esim. vaihtamalla ensimmäinen a tyhjämerkiksi (kuten edellä).
- Kohdassa 3 symbolin b ”merkkaaminen” tapahtuu esim. kirjoittamalla sen tilalle \check{b} , missä \check{b} on nauha-aakkostoon lisätty uusi symboli.

Jatkossa oletamme, että Turingin kone osataan tarvittaessa toteuttaa edellisen tapaisen (tai vielä korkeammantasoisien) kuvauksen perusteella. \square

Olemme siis todenneet, että edellä esitetyt kielet A , B ja C ovat Turing-tunnistettavia. Koska kaikki esitetyt koneet pysähtyvät kaikilla syötteillä, ne ovat vieläpä ratkeavia.

Turingin koneen muunnelmia [Sipser luku 3.2]

Turingin koneen yksityiskohdat voidaan määritellä monella tavalla, jotka antavat samat tunnistettavien ja ratkeavien kielten luokat.

Muunnelma 1: Sallitaan siirtymäfunktion olla muotoa

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$. "Suunta" **S** tarkoittaa, että nauhapää pysyy paikoillaan (stay put). Koska kukin S-siirtymä voidaan korvata peräkkäisillä R- ja L-siirtymällä (lisäten tarvittaessa uusi tila), niiden salliminen ei lisäisi mallin laskentavoimaa.

Muunnelma 2: Nauha jatkuu äärettömän pitkälle myös vasemmalle.

Syötteellä $w = w_1 \dots w_n$ nauhan sisältö on aluksi $\dots \sqcup \sqcup \sqcup w_1 \dots w_m \sqcup \sqcup \dots$ ja nauhapää osoittaa symbolia w_1 . Kahteen suuntaan äärettömän nauhan sisältö $\dots a_{-2} a_{-1} a_0 a_1 a_2 \dots$, missä $a_i \in \Gamma$, voidaan esittää "perusmallissa" muodossa $(a_1, a_0)(a_2, a_{-1})(a_3, a_{-2}) \dots$, missä nauha-aakkostona on Γ^2 . Tästä on helppo nähdä, että mallin laskentavoima ei taaskaan kasva.

Tärkeämpiä muunnelmia ovat **moninauhaiset** ja **epädeterministiset** koneet.

Moninauhainen Turingin kone (multitape TM)

Moninauhaisessa Turingin koneessa on useita nauhoja ja kullakin oma nauhapäänsä. Nauhoille voidaan kirjoittaa ja nauhapäitä siirtää toisistaan riippumatta.

Siirtymäfunktio k -nauhaiselle Turingin koneelle on muotoa

$$\delta: Q' \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k.$$

Siirtymä $\delta(q, a_1, \dots, a_k) = (q', b_1, \dots, b_k, D_1, \dots, D_k)$ tarkoittaa, että jos

- kone on tilassa q ja
- kaikilla $1 \leq i \leq k$ nauhapään i kohdalla on symboli a_i

niin

- kone siirtyy tilaan q' ja
- kaikilla $1 \leq i \leq k$
 - nauhalle i kirjoitetaan symboli b_i ja
 - nauhapää i siirtyy suuntaan D_i .

Alkutilanteessa ykkösnauha sisältää syötteen ja muut ovat tyhjiä.

Koneet ovat **ekvivalentit**, jos ne tunnistavat saman kielen.

Lause 3.3: [Sipser Thm. 3.13] Jokaiselle moninauhaiselle Turingin koneelle on olemassa ekvivalentti yksinauhainen Turingin kone.

Todistus: Olkoon M k -nauhainen kone, jonka nauha-aakkosto on Γ . Muodostamme ekvivalentin yksinauhaisen koneen S , jonka nauha-aakkostoksi tulee

$$\Gamma \cup \{\check{a} \mid a \in \Gamma\} \cup \{\#\}.$$

Tässä $\check{a} \notin \Gamma$ on symbolin a merkattu versio kaikilla $a \in \Gamma$, ja $\# \notin \Gamma$ uusi välimerkki.

Perusidea esimerkin valossa: Oletetaan, että kolminauhaisen koneen nauhojen sisällöt ovat

nauha 1: $a\underline{a}bB\underline{\quad}\underline{\quad}\underline{\quad}\dots$

nauha 2: $\underline{\quad}\underline{\quad}\underline{\quad}\dots$

nauha 3: $001\underline{0}01\underline{\quad}\underline{\quad}\underline{\quad}\dots$

missä nauhapään sijainti on alleviivattu. Tämä esitetään yhdellä nauhalla muodossa

nauha: $a\check{a}bB\#\check{\quad}\#001\check{0}01\#\check{\quad}\check{\quad}\check{\quad}\dots$

Tarkemmin S on kone, joka syötteellä $w = w_1 \dots w_n$ toimii seuraavasti:

1. Alusta nauhan sisällöksi $\# \check{w}_1 \dots w_n \# \check{} \# \check{} \# \dots \# \check{} \check{} \check{} \dots$ missä $\#$ -merkkejä on $k + 1$ kappaletta.
2. Simuloi yksi koneen M laskenta-askel seuraavasti:
 - (a) Selaa koko nauha ensimmäisestä viimeiseen $\#$ -merkkiin. Pane muistiin, mitkä merkatut symbolit ($\check{}$) esiintyivät ja missä järjestyksessä. Vaihtoehtoja on äärellinen määrä $|\Gamma|^k$, joten ne voidaan koodata koneen S tiloihin.
 - (b) Päätä koneen M uusi tila siirtymäfunktion mukaisesti.
 - (c) Selaa nauha takaisin alkuun ja samalla muuta merkittyjä symboleita koneen M siirtymäfunktion mukaisesti. Jos jokin nauhapään merkki siirtyy oikealle $\#$ -merkin päälle, tee tilaa siirtämällä koko nauhan loppuosa askel oikealle.
3. Jos koneen M uusi tila on hyväksyvä, niin hyväksy. Jos koneen M uusi tila on hylkäävä, niin hylkää. Muuten jatka kohdasta 2.

□

Korollaari 3.4: [Sipser Cor. 3.15] Kieli on Turing-tunnistettava (-ratkeava), jos ja vain jos jokin moninauhainen Turingin kone tunnistaa (vast. ratkaisee) sen.

Todistus: "Vain jos" -suunta on ilmeinen. "Jos" -suunta seuraa edellisestä konstruktiosta, jossa simuloiva S pysähtyy, jos ja vain jos simuloitava M pysähtyy. \square

Reunahuomautus: Oletetaan, että k -nauhainen kone M syötteellä w pysähtyy T askelella. Jos $T \geq w$, millekään nauhalle ei tule yli T symbolia. Siis simuloivassa koneessa S nauhalle tulee $O(kT)$ symbolia.

Yhden simulointiaskelen toteuttamiseksi S joutuu pahimmillaan siirtämään nauhan sisältöä oikealle k kertaa. Siis yksi koneen M askel vie pahimmillaan $O(k^2T)$ koneen S askelta, ja koko laskenta $O(k^2T^2)$ askelta.

Vaikka **laskennan vaativuusteoria** ei varsinaisesti kuulu tälle kurssille, toteamme, että kieli voidaan tunnistaa **polynomisessa ajassa** yksinauhaisella koneella, jos ja vain jos se voidaan tunnistaa polynomisessa ajassa moninauhaisella koneella.

Epädeterministinen Turingin kone

Kuten muidenkin automaattien tapauksessa, epädeterministisessä Turingin koneessa siirtymäfunktio antaa **joukon** mahdollisia seuraajia. Se siis on tyyppiä

$$\delta: Q' \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

Merkintä \vdash yleistetään vastaavasti, esim. $uqav \vdash ubq'b$ pätee, jos $(q', b, R) \in \delta(q, a)$. Epädeterministinen kone N hyväksyy syötteen, jos **jokin** mahdollinen laskenta johtaa hyväksyvään tilanteeseen.

Huom. Määritelmässä on oleellinen epäsymmetria. Kun $A = L(N)$, niin

- jos $w \in A$, niin **ainakin yksi** laskenta hyväksyy, mutta muut saavat johtaa hylkäämiseen tai silmukkaan;
- jos $w \notin A$, niin **mikään** laskenta ei johda hyväksymiseen, vaan kaikkien on johdettava hylkäämiseen tai silmukkaan.

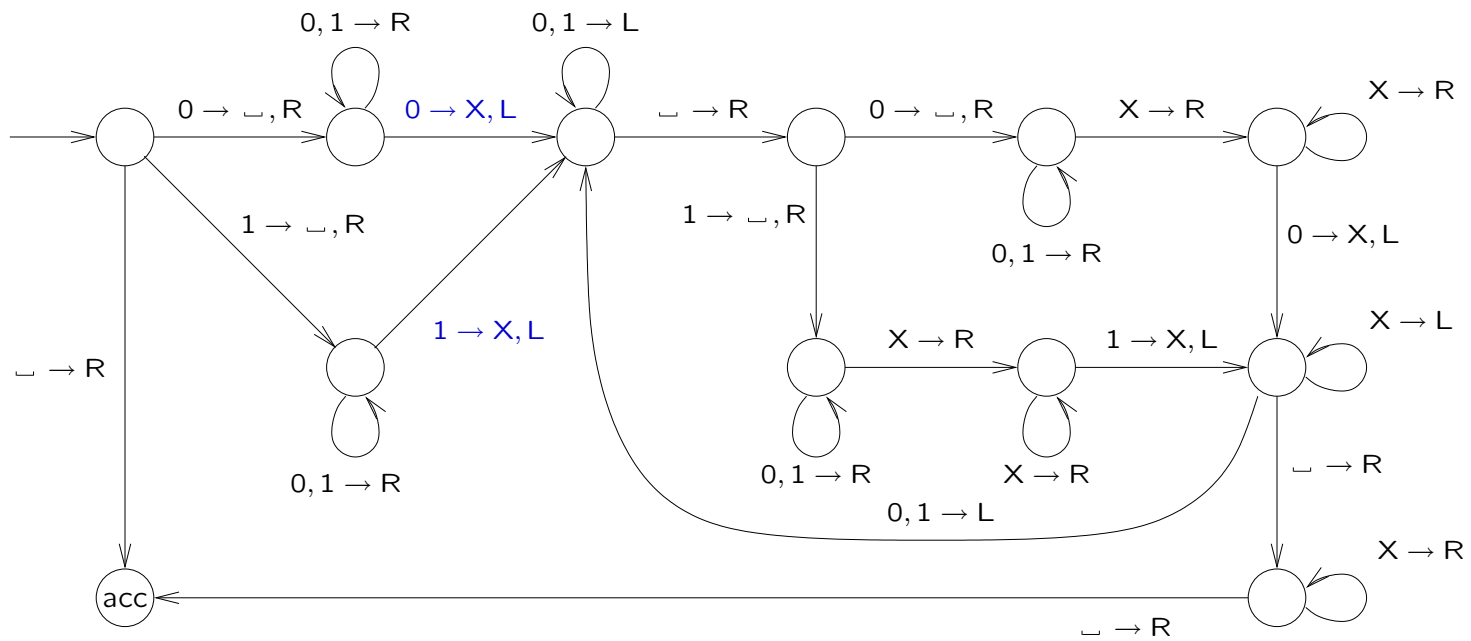
Jos määritelmä olisi löysästi "jonkin laskennan pitää johda oikeaan lopputulokseen", niin mikä tahansa kieli voitaisiin tunnistaa triviaalisti asettamalla kaikilla $a \in \Sigma$

$$\delta(q_0, a) = \{ (q_{\text{accept}}, a, R), (q_{\text{reject}}, a, R) \}.$$

Esimerkki 3.5: Tuttu kieli $B = \{ ww \mid w \in \{0,1\}^* \}$ voidaan tunnistaa epädeterministisellä Turingin koneella seuraavasti:

1. Laita syötteen ensimmäinen symboli muistiin ja pyyhi se pois.
2. Valitse epädeterministisesti jokin symboli syöttestä. Jos se ei ollut sama kuin ensimmäinen symboli, niin hylkää. Muuten viivaa se yli.
3. Palaa syötteen alkuun.
4. Laita syötteen seuraava merkki muistiin ja pyyhi se pois.
5. Selaa nauhaa oikealle, kunnes yliviivatut symbolit loppuvat. Jos niitä seuraava symboli on \sqcup , mene kohtaan 6. Muuten jos niitä seuraava symboli ei ole sama kuin muistissa oleva, hylkää. Muuten viivaa yli tämä symboli ja palaa kohtaan 3.
6. Jos kaikki symbolit on yliviivattu, hyväksy, muuten hylkää.

Erona deterministiseen ratkaisuun (s. 163) meidän ei tarvitse etsiä syötteen keskikohtaa. Riittää arvata **ja tarkistaa**.



Sama kaaviona. Hylkäävä tila siirtymineen jätetty merkitsemättä. Kaksi epädeterminististä siirtymää merkitty sinisellä. \square

Lause 3.6: [Sipser Thm. 3.16] Jokaiselle epädeterministiselle Turingin koneelle on olemassa ekvivalentti deterministinen Turingin kone.

Todistus: Olkoon N epädeterministinen Turingin kone. Muodostamme deterministisen Turingin koneen D , joka kokeilee järjestyksessä kaikkia mahdollisia koneen N laskentoja annetulla syötteellä. Jos hyväksyvä laskenta löytyy, D hyväksyy. Muuten D jää silmukkaan. Siis $L(D) = L(N)$.

Perusidea on kuvitella puu, jonka solmuina on koneen N tilanteita:

- juurena alkutilanne annetulla syötteellä ja
- solmun uqv lapsina kaikki tilanteet $u'q'v'$, joilla $uqv \vdash u'q'v'$.

Kone D käy puuta läpi leveyssuuntaisesti. Tätä varten haluamme indeksoida puun solmut.

Olkoon b yläraja solmun lapsien lukumäärälle:

$$b = \max_{q,a} |\delta(q, a)|.$$

Vakion b tarkalla arvolla ei ole tässä merkitystä, mutta selvästi $b \leq 3|Q||\Gamma|$.

Numeroidaan nyt jokaisen solmun lapset (eli jokaisen tilanteen seuraajatilanteet) numeroilla joukosta $\{1, \dots, b\}$ (joista osa voi jäädä käyttämättä). Jos uqv ja $u'q'v'$ ovat kaksi saman solmun lasta, voidaan esim. sopia, että uqv saa pienemmän numeron, jos

- jos q :n järjestysnumero on pienempi kuin q' :n tai
- $q = q'$ ja u on aakkosjärjestyksessä ennen kuin u' tai
- $q = q'$ ja $u = u'$ ja v on aakkosjärjestyksessä ennen kuin v' .

Tästä saadaan puun jokaiselle solmulle osoite joukosta $\{1, \dots, b\}^*$:

- juuren osoite on ε ja
- jos solmun osoite on $p_1 \dots p_k$, niin sen lapsi numero q saa osoitteen $p_1 \dots p_k q$.

Kaikki joukon $\{1, \dots, b\}^*$ alkioit eivät ole minkään solmun osoitteita.

Muodostamme nyt kolminauhaisen koneen D , joka toimii kuten edellä on esitetty. Lauseen 3.3 nojalla tästä saadaan edelleen yksinauhainen kone.

Nauha 1 sisältää syötteen. Nauhalle 1 ei koskaan kirjoiteta.

Nauha 2 on työnauha, jonka sisältö seuraa koneen N nauhan sisältöä laskennan eri vaiheissa.

Nauha 3 sisältää aakkoston $\{1, \dots, b\}$ merkkijonon, joka tulkitaan osoitteeksi nauhalla 1 olevaa syötettä vastaavaan koneen N laskentapuuhun. Aluksi nauha 3 on tyhjä eli osoittaa puun juurta.

Koneen D periaate on seuraava:

1. Etsi laskentapuusta nauhan 3 osoittama solmu. Jos se vastaa hyväksyvää tilannetta, niin hyväksy. Jos solmua ei ole tai se vastaa muuta kuin hyväksyvää tilannetta, siirry kohtaan 2.
2. Korvaa nauhan 3 sisältö leksikografisessa järjestyksessä seuraavalla aakkoston $\{1, \dots, b\}$ merkkijonolla. Palaa kohtaan 1.

Nauhan 3 osoittaman solmun etsiminen tapahtuu seuraavasti:

1. Olkoon nauhan 3 merkkijono $p_1 \dots p_n$.
2. Kopio nauhan 1 sisältö nauhan 2 sisällöksi ja aseta nauhapää nauhan 2 alkuun. Aseta koneen N simulaatio alkutilaan.
3. Simuloi koneen N laskentaa n askelta käyttäen nauhaa 2.
 - Laskenta-askelella numero i valitse uudeksi tilanteeksi nykyisen tilanteen seuraaja numero p_i .
 - Jos nykyisellä tilanteella on vähemmän kuin p_i seuraajaa, keskeytä laskenta; etsittyä solmua ei ole.



Korollaari 3.7: [Sipser Cor. 3.18] Kieli on Turing-tunnistettava, jos ja vain jos jokin epäterministinen Turingin kone tunnistaa sen.

Todistus: Suunta "vain jos" on ilmeinen. Suunta "jos" seuraa edellisestä lauseesta. \square

Epäterministinen Turingin kone on **ratkaisija** (eli **totaalinen** kone), jos sillä ei millään syötteellä ole päättymättömiä laskentoja.

Korollaari 3.8: [Sipser Cor. 3.19] Kieli on ratkeava, jos ja vain jos jokin epäterministinen Turingin kone ratkaisee sen.

Todistus: Edellisen lauseen konstruktioita on helppo täydentää siten, että D havaitsee, jos kaikki koneen N laskennat ovat pysähtyneet. \square

Luettelijakone (enumerator)

Luetteliija toimii tavallisen Turingin koneen tapaan, mutta

- syöte on aina tyhjä merkkijono ja
- aika ajoin kone voi tulostaa aakkoston Σ^* merkkijonon.

Tulostaminen voidaan mallintaa erillisellä nauhalla, jolle ainoat sallitut operaatiot ovat muotoa "kirjoita symboli a ja siirrä nauhapäätä oikealle". Tulostaminen siis on peruuttamatonta.

Tyypillisesti luetteliija ei pysähdy, jolloin se voi tulostaa äärettömän monta merkkijonoa.

Luettelijan **luettelema kieli** on niiden merkkijonojen joukko, jotka se joskus tulostaa. Tulostuksessa

- sama merkkijonon saa esiintyä monta kertaa ja
- merkkijonot saavat olla missä tahansa järjestyksessä.

Lause 3.9: [Sipser Thm. 3.21] Kieli on Turing-tunnistettava, jos ja vain jos jokin luettelijä luettelee sen.

Huom. tämä on termin ”rekursiivisesti numeroituva” alkuperä.

Todistus: Jos luettelijä E luettelee kielen, se voidaan tunnistaa Turingin koneella, joka syötteellä w toimii seuraavasti:

Simuloi luettelijaa E . Aina, kun E tulostaa jotain, vertaa tulostetta syötteeseen w . Jos ne ovat samat, niin hyväksy.

Olkoon A aakkoston Σ^* kieli ja s_1, s_2, s_3, \dots lista kaikista aakkoston Σ merkkijonoista (esim. leksikografisessa järjestyksessä). Jos M tunnistaa kielen A , se voidaan luetella seuraavasti:

1. Toista seuraavaa arvoilla $i = 1, 2, \dots$:
 - (a) Toista seuraavaa arvoilla $j = 1, \dots, i$:

Simuloi i askelta koneen M laskentaa syötteellä s_j . Jos M hyväksyi, tulosta s_j .

□

Algoritmin määritelmä [Sipser luku 3.3]

Mitä algoritmilla yleensä tarkoitetaan

periaatteessa: yksiselitteisesti kuvattu jono (tietojenkäsittely)operaatioita, jotka voidaan toteuttaa mekaanisesti

käytännössä: luonnollista kieltä, pseudokoodia yms. käyttävä esitys, jonka pätevä ohjelmoija osaa koodata ilman suurempia ongelmia.

Tämä tarkkuustaso ei ole riittävä, jos halutaan tutkia laskettavuuden rajoja.

Eryteisesti kun haluamme väittää jostain ongelmasta, että sille **ei ole olemassa** ratkaisualgoritmia, niin mitä oikeastaan väitämme? Huomaa, että tämä on oleellisesti eri asia kuin todeta, että ongelmalle ei ole **keksitty** algoritmia (toistaiseksi).

Tuntuu luontevalta vaatia, että algoritmissa

- yksi operaatio saa tehdä vain äärellisen määrän työtä.

Siis erityisesti yksi operaatio saa

- lukea äärellisen määrän tietoa,
- etsiä toimintaohjeen äärellisestä sääntöjoukosta ja
- kirjoittaa äärellisen määrän tietoa.

Toisaalta ei ole mitään syytä olla sallimatta, että

- algoritmin käytössä on rajattomasti apumuistia.

Tämän esittäminen matemaattisesti johtaa suoraan ajatukseen, että matemaattiselta kannalta

algoritmi on sama asia kuin **Turingin kone**.

Tämä on (karkeasti) alkuperäinen ajatus Turingin koneen takana.

Turingin kone ei syntynyt sattumalta.

Matematiikan perusteiden tutkimuksessa 1900-luvun alussa keskeistä oli matematiikan mekanisoiminen (voiko matemaatikon korvata algoritmilla). Tätä varten esitettiin useita formalisointeja mekaaniselle laskennalle:

- Turingin kone (Turing 1936)
- Postin sääntöjärjestelmät (Post 1936)
- μ -rekursiiviset funktiot (Gödel, Kleene 1936)
- λ -kalkyyli (Church 1936)

Merkittävä havainto oli, että nämä kaikki olivat [ekvivalentteja](#), ts. antavat tasan [saman](#) vastauksen kysymykseen ”Mitä ongelmia voidaan ratkaista algoritmisesti?” Tämä johti [Churchin-Turingin teesinä](#) tunnettuun väittämään:

Ongelma voidaan ratkaista algoritmilla, jos ja vain jos se voidaan ratkaista Turingin koneella.

Churchin-Turingin teesi ei tietenkään ole matemaattinen väittämä. Sitä voi (kenties) pitää luonnontieteellisenä väittämänä, jonka voisi periaatteessa falsifioida rakentamalla Turingin konetta voimakkaamman laskulaitteen.

Käytännössä Churchin-Turingin teesi lähinnä sanoo, että olemme tyytyväisiä Turingin koneeseen algoritmin määritelmänä. Kuten edellä ilmeni, tälle tyytyväisyydelle on tiettyjä filosofisia perusteita. Myöskään sellaiset modernit laskennan mallit kuin kvanttietokone eivät näyttäisi uhkaavan tätä teesiä.

Käytännön tietojenkäsittelyn kannalta tärkeä havainto on, että myös (idealisoitu) nykyaikainen tietokone on [Turing-ekvivalentti](#) eli pystyy ratkaisemaan tasan samat ongelmat kuin Turingin kone. Tarkastellaan tätä hieman lähemmin.

On selvää, että tietokone on ainakin yhtä laskentavoimainen kuin Turingin kone: on helppoa kirjoittaa Turingin kone -simulaattori esim. C-kielellä. Tietokoneen simuloiminen Turingin koneella vaatii hieman enemmän työtä.

Ajatellaan, että tietokoneessa on k rekisteriä ja lisäksi rajoittamaton määrä hajasaantimuistia (RAM).

Simuloidaan tietokonetta $k + 2$ -nauhaisella Turingin koneella. Nauhoille $1, \dots, k$ talletetaan rekisterien $1, \dots, k$ sisällöt. Nauhalla $k + 1$ esitetään tietokoneen muistin sisältö muodossa

$$\#a_1\#d_1\#\#a_2\#d_2\#\#\dots\#\#a_n\#d_n\#\sqcup,$$

missä n on käytössä olevan muistin määrä ja d_i on muistipaikan a_i sisältö (esim. binäärikoodattuna). Nauha $k + 2$ toimii apumuistina.

Aiempien esimerkkien perusteella pitäisi olla uskottavaa, että Turingin kone pystyy simuloimaan konekäskyjä kuten

- lataa rekisterin 2 osoittaman muistipaikan sisältö rekisteriin 3,
- lisää rekisterin 1 sisältöön rekisterin 2 sisältö jne.

Tosin muistiin kirjoitettaessa voidaan joutua siirtämään oikealle tai vasemmalle pitkä pätkä nauhan $k + 1$ sisältöä.

Turingin koneiden esittäminen

Olemme esittäneet Turingin koneita eri tarkkuustasoilla:

formaali kuvaus: tarkka tilakaavio

toteutustaso: selitetään nauhojen käyttö ja muut pääideat, mutta ei puututa koneen yksittäisiin tiloihin jne.

korkea taso: esitetään algoritmi sellaisenaan (esim. pseudokoodina) viittaamatta erityisesti Turingin kone -malliin.

Jatkossa tyydymme yleensä korkean tason kuvauksiin ja uskomme, että mikä tahansa algoritmi osataan kyllä tarvittaessa "koodata Turingin koneen konekielelle".

Seuraavassa esitetään joitain esitystä selkeyttäviä sopimuksia.

Syötteen koodaaminen: Turingin koneen syöte on merkkijono. Algoritmit tyypillisesti saavat syötteenään verkkoja, kokonaislukuja yms. olioita. Käytämme merkintää $\langle O \rangle$ merkkijonolle, joka esittää olion O jossain sopivassa aakkostossa. Merkintä $\langle O_1, \dots, O_n \rangle$ on vastaava merkkijonoesitys jonolle (O_1, \dots, O_n) .

Syötteen tarkistaminen: Usein sanomme esim. että Turingin koneen syöte on $\langle G, k \rangle$, missä G on suuntaamaton verkko ja k luonnollinen luku, Tällöin kyseinen kone aluksi tarkistaa, että syötemerkkijono todella on kelvallinen esitys parille (G, k) jollain verkolla G ja luonnollisella luvulla k . Jos syöte ei ole tätä muotoa, se hylätään.

Siirrymme nyt tarkastelemaan erilaisiin formaaleihin kieliin liittyviä ratkeavuusongelmia. Esimerkin vuoksi kiinnitämme aluksi tavallista enemmän huomiota esitystapakysymyksiin.

4. Ratkeavuus

Tarkastelemme formaaleihin kieliin liittyviä algoritmeja käyttäen Turingin konetta algoritmin täsmällisenä määritelmänä.

Tämän luvun jälkeen opiskelija

1. osaa kuvailla **universaaliin Turingin koneeseen** liittyvät peruskäsitteet
2. tuntee **universaalikielen ratkeamattomuuteen** liittyvät peruskonstruktiot
3. osaa perustella tärkeimpiin formaalien kielten tunnistamisongelmiin liittyvät ratkeavuustulokset.

Yleisemmällä tasolla tavoittena on ymmärtää, millaiset ongelmat ovat ratkeamattomia ja mitä tämä tarkoittaa.

Ratkeavuutta formaaleille kielille [Sipser luku 4.1]

Tarkastelemme hyväksymisongelmia (eli kieleenkuulumisongelmia), joiden perusformaatti on

Annettu: automaatti M (tai kielioppi G) ja merkkijono w

Kysymys: päteekö $w \in L(M)$ (tai vastaavasti $w \in L(G)$)

Esimerkiksi äärellisten automaattien hyväksymisongelma on kieli

$$A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ on DFA, joka hyväksyy } w:n \}.$$

Lause 4.1: [Sipser Thm. 4.1] Kieli A_{DFA} on ratkeava.

Todistus: Kieli A_{DFA} voidaan tunnistaa Turingin koneella, joka syötteellä $\langle B, w \rangle$ toimii seuraavasti:

1. Simuloi automaattia B syötteellä w .
2. Jos simulaation lopussa B on hyväksyvässä tilassa, hyväksy; muuten hylkää.

□

Koska jatkon kannalta on tärkeä ymmärtää, miten automaattia simuloidaan Turingin koneella, tarkastellaan hieman toteutuksen yksityiskohtia. Muodostetaan nelinauhainen Turingin kone M , joka tunnistaa kielen A_{DFA} .

Valitaan koneen M syöteaakkostoksi $\Sigma_M = \{0, \dots, 9\} \cup \{\#\}$. Automaatti $B = (Q_B, \Sigma_B, \delta_B, q_{\text{start}}, F_B)$ koodataan aakkoston Σ_M merkkijonoksi seuraavasti:

Oletetaan $Q_B = \{q_1, \dots, q_n\}$ ja $\Sigma_B = \{a_1, \dots, a_m\}$. Merkitään

$\langle q_i \rangle =$ indeksin i esitys kymmenjärjestelmässä.

Siis jokaisella $q \in Q$ koodi $\langle q \rangle$ on jokin aakkoston $\{0, \dots, 9\}$ merkkijono. Määritellään vastaavasti $\langle a_i \rangle$. Tilajoukko, aakkosto ja hyväksyvät tilat koodataan nyt merkkijonoilla

$$\begin{aligned}\langle Q_B \rangle &= \langle q_1 \rangle \# \langle q_2 \rangle \# \dots \# \langle q_n \rangle \\ \langle \Sigma_B \rangle &= \langle a_1 \rangle \# \langle a_2 \rangle \# \dots \# \langle a_m \rangle \\ \langle F_B \rangle &= \langle r_1 \rangle \# \langle r_2 \rangle \# \dots \# \langle r_k \rangle,\end{aligned}$$

missä $F_B = \{r_1, \dots, r_k\} \subseteq Q$.

Siirtymäfunktio δ_B koodataan merkkijonoksi

$$\langle \delta_B \rangle = \langle q_1 \rangle \# \langle a_1 \rangle \# \langle r_{11} \rangle \#\# \langle q_1 \rangle \# \langle a_2 \rangle \# \langle r_{12} \rangle \#\# \dots \#\# \langle q_n \rangle \# \langle a_m \rangle \# \langle r_{nm} \rangle,$$

missä $r_{ij} = \delta(q_i, a_j)$. Siis jokainen siirtymä esitetään omana kolmikkonaan.

Nyt pari (B, w) , missä $w = w_1 \dots w_l \in \Sigma_B^*$, koodataan merkkijonoksi

$$\langle B, w \rangle = \#\# \langle Q_B \rangle \#\#\# \langle \Sigma_B \rangle \#\#\# \langle \delta_B \rangle \#\#\# \langle q_{\text{start}} \rangle \#\#\# \langle F_B \rangle \#\#\# \langle w \rangle,$$

missä $\langle w \rangle = \langle w_1 \rangle \# \langle w_2 \rangle \# \dots \# \langle w_l \rangle$.

Simulointi tapahtuu nyt seuraavasti:

1. Kopioi nauhalta 1 (eli syötenauhalla)
 - koko $\langle w \rangle$ nauhalle 2,
 - ensimmäinen merkki $\langle w_1 \rangle$ nauhalle 3,
 - alkutila $\langle q_{\text{start}} \rangle$ nauhalle 4.
2. Etsi nauhalta 1 pätkä $\#\#\langle q \rangle\#\langle a \rangle\#\langle r \rangle$, missä $\langle q \rangle$ ja $\langle a \rangle$ ovat samat kuin nauhojen 4 ja 3 sisällöt. Kopioi $\langle r \rangle$ nauhan 4 uudeksi sisällöksi.
3. Jos nauhalla 2 on vielä käsittelemättömiä merkkejä, kopioi sieltä seuraava $\langle w_i \rangle$ nauhan 3 uudeksi sisällöksi ja palaa kohtaan 2. Muuten jatka kohdasta 4.
4. Jos nauhan 4 sisältö on sama kuin jokin syötteen $\langle F_B \rangle$ -osuudesta löytyvä koodi $\langle r_i \rangle$, hyväksy; muuten hylkää.

Edellisestä seuraa hyväksymisongelman ratkeavuus myös NFA:lle.

Lause 4.2: [Sipser Thm. 4.2] Kieli

$$A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ on NFA, joka hyväksyy } w:n \}$$

on ratkeava.

Todistus: Kieli A_{NFA} voidaan tunnistaa algoritmilla, joka syötteellä $\langle B, w \rangle$ toimii seuraavasti:

1. Muodosta B :n kanssa ekvivalentti DFA C lauseen 1.3 todistuksessa (s. 39–41) esitetyllä algoritmilla.
2. Jos $\langle C, w \rangle \in A_{\text{DFA}}$, niin hyväksy; muuten hylkää.

Koska A_{DFA} on ratkeava, tiedämme, että askelen 2 ehtotesti voidaan toteuttaa algoritmisesti. (Emme lähemmin puutu tämän algoritmin toteuttamiseen Turingin koneena.) \square

Samaa periaatetta noudattaen saadaan edelleen

Lause 4.3: [Sipser Thm. 4.3] Kieli

$$A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ on säännöllinen lauseke ja } w \in L(R) \}$$

on ratkeava.

Todistus: Syötteellä $\langle R, w \rangle$ muodostetaan ensin kielen $L(R)$ tunnistava NFA B (kuten lemmassa 1.9, s. 56). Hyväksytään, jos $\langle B, w \rangle \in A_{\text{NFA}}$; muuten hylätään. \square

Tämä tulos siis käytännön kannalta sanoo, että säännöllisiä lausekkeita on mahdollista tulkita tietokoneella.

Hieman vähemmän ilmeistä on kielen

$$E_{\text{DFA}} = \{ \langle B \rangle \mid B \text{ on DFA ja } L(B) = \emptyset \}$$

ratkeavuus. Siis $\langle B \rangle \in E_{\text{DFA}}$, jos ja vain jos $\langle B, w \rangle \notin A_{\text{DFA}}$ kaikilla merkkijonoilla w . Suora simulointi ei toimi, koska kokeiltavia w on äärettömästi, mutta silti

Lause 4.4: [Sipser Thm. 4.4] Kieli E_{DFA} on ratkeava.

Todistus: Syötteen $\langle B \rangle$ kuulumisen kieleen E_{DFA} voidaan ratkaista seuraavasti:

1. Muunna B oikealle lineaariseksi kieliopiksi G (harjoitus 6, tehtävä 7).
2. Tarkista, onko $L(G) = \emptyset$ (harjoitus 7, tehtävä 4(b)). Jos on, niin hyväksy; muuten hylkää.

(Kurssikirjassa on esitetty vaihtoehtoinen suora konstruktio.) \square

Samanhenkkinen on äärellisten automaattien **ekvivalenssiongelma** eli kieli

$$EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ ja } B \text{ ovat DFA:ita ja } L(A) = L(B) \}.$$

Lause 4.5: [Sipser Thm. 4.5] Kieli EQ_{DFA} on ratkeava.

Todistus: Syötteestä $\langle A, B \rangle$ voidaan tunnetuilla konstruktiolla muodostaa DFA C , jolla

$$L(C) = \left(L(A) \cap \overline{L(B)} \right) \cup \left(\overline{L(A)} \cap L(B) \right).$$

Nyt $L(A) = L(B)$, jos ja vain jos $L(C) = \emptyset$. Siis $\langle A, B \rangle \in EQ_{\text{DFA}}$, jos ja vain jos $\langle C \rangle \in E_{\text{DFA}}$. Väite seuraa nyt lauseesta 4.4. \square

Yhteydettömillä kielillä asiat sujuvat suunnilleen, mutta ei aivan, samaan tapaan.

Lause 4.6: [Sipser Thm. 4.7] Kieli

$$A_{CFG} = \{ \langle G, w \rangle \mid G \text{ on yhteydetön kielioppi ja } w \in L(G) \}$$

on ratkeava.

Todistus: Syötteellä $\langle G, w \rangle$ voidaan soveltaa seuraavaa algoritmia:

1. Muunna G Chomskyn normaalimuotoon.
2. Ratkaise CYK-algoritmilla (s. 112–115), päteekö $w \in L(G)$.

□

Tästä saadaan seuraava tärkeä seuraus:

Lause 4.7: [Sipser Thm. 4.9] Jokainen yhteydetön kieli on ratkeava.

Todistus: Olkoon A yhteydetön; siis $A = L(G)$ jollain yhteydettömällä kieliopilla G . Nyt A voidaan tunnistaa Turingin koneella M_G , joka syötteellä w toimii seuraavasti:

1. Muodosta koodi $\langle G, w \rangle$.
2. Hyväksy, jos $\langle G, w \rangle \in A_{CFG}$; muuten hylkää.

□

Myös yhteydettömien kielioppien tyhjiysongelma

$$E_{CFG} = \{ \langle G \rangle \mid G \text{ on CFG ja } L(G) = \emptyset \}$$

on ratkeava. Tämä on todettu harjoituksen 7 tehtävässä 4(b), johon jo edellä vedottiin.

Sen sijaan yhteydettömien kielioppien ekvivalenssiongelma

$$EQ_{CFG} = \{ \langle G, H \rangle \mid G \text{ ja } H \text{ ovat CFG:itä ja } L(G) = L(H) \}$$

ei ole ratkeava. Meillä ei tässä vaiheessa ole työkaluja tämän tuloksen todistamiseen. Todetaan vain, että kielen EQ_{DFA} tapauksessa (lause 4.5) käytetty menetelmä ei tässä toimi, koska yhteydettömät kielet eivät ole suljettuja leikkauksen suhteen (korollaari 2.27, s. 144).

Pysähtymisongelman ratkeavuus [Sipser luku 4.2]

Osoitamme nyt vihdoin, että

- jotkin Turing-tunnistettavat kielet ovat ratkeamattomia ja
- jotkin kielet eivät ole edes Turing-tunnistettavia.

Lisäksi toteamme, että ratkeamattomat ongelmat eivät välttämättä ole mitenkään "omituisia", vaan myös monet luonnolliset ja käytännössä kiinnostavat ongelmat ovat ratkeamattomia.

Perusesimerkki ratkeamattomasta ongelmasta on (sopivasti formuloitu) pysähtymisongelma

Annettu: (esim. C-kielinen) ohjelma P , syöte w

Kysymys: pysähtyykö P syötteellä w .

Määritellään ensin Turingin koneen hyväksymisongelma

$$A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ on Turingin kone joka hyväksyy } w:n \}.$$

Lause 4.8: [Sipser s. 176] Kieli A_{TM} on Turing-tunnistettava.

Todistus: Kieli A_{TM} voidaan tunnistaa Turingin koneella U , joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Simuloi konetta M syötteellä w .
2. Jos M menee tilaan q_{accept} , niin hyväksy.
Jos M menee tilaan q_{reject} , niin hylkää.

Syötteen koodaaminen ja simulointi voidaan hoitaa samaan tapaan kuin äärellisille automaateille (s. 191–193); sivuutamme yksityiskohdat. \square

Edellä esitettyyn laista konetta U sanotaan **universaalikoneeksi**. Sille pätee

$$\begin{aligned} U \text{ hyväksyy syötteen } \langle M, w \rangle &\Leftrightarrow M \text{ hyväksyy syötteen } w \\ U \text{ hylkää syötteen } \langle M, w \rangle &\Leftrightarrow M \text{ hylkää syötteen } w \\ U \text{ jää silmukkaan syötteellä } \langle M, w \rangle &\Leftrightarrow M \text{ jää silmukkaan syötteellä } w. \end{aligned}$$

Kieleen A_{TM} liittyy läheisesti Turingin koneen pysähtymisongelma

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ on Turingin kone joka pysähtyy syötteellä } w. \}$$

Karkea toimintasuunnitelma on seuraava:

1. Alkulämmittelynä todistetaan hieman abstrakti tulos, että ratkeamattomia (ja ei-tunnistettavia) kieliä **on olemassa**.
2. Samaa tekniikkaa soveltamalla osoitetaan, että **kieli A_{TM}** on ratkeamaton.
3. Edellisestä päätellään, että **myös $HALT_{TM}$** on ratkeamaton.

Koska Turingin koneet ovat yhtä ilmaisuvoimaisia esim. C-kielisten ohjelmien kanssa, tästä seuraa, että "C-kielen pysähtymisongelmaa" ei voi ratkaista C-kielillä.

Kertaus: Olkoon $f: A \rightarrow B$ funktio. Sanomme, että f on

- **injektio** (one-to-one), jos jokaisella $y \in B$ on korkeintaan yksi $x \in A$, jolla $f(x) = y$;
- **surjektio** (onto), jos jokaisella $y \in B$ on ainakin yksi $x \in A$, jolla $f(x) = y$; ja
- **bijektio**, jos jokaisella $y \in B$ on tasan yksi $x \in A$, jolla $f(x) = y$.

Jos f on bijektio, sillä on **käänteiskuvaus** $f^{-1}: B \rightarrow A$, jolla $f^{-1}(f(x)) = x$ ja $f(f^{-1}(y)) = y$ kaikilla $x \in A$ ja $y \in B$.

Joukot A ja B ovat **yhtä mahtavat**, merkitään $A \approx B$, jos on olemassa bijektio $f: A \rightarrow B$.

Siis äärelliset joukot ovat yhtä mahtavat, jos ja vain jos niissä on yhtä monta alkioita. Äärettömien joukkojen tapauksessa tilanne on monimutkaisempi.

Esimerkki 4.9: Joukot $\mathbf{N} = \{0, 1, 2, \dots\}$ ja $2\mathbf{N} = \{0, 2, 4, \dots\}$ ovat yhtä mahtavat, sillä $f(n) = 2n$ määrittelee bijektio $f: \mathbf{N} \rightarrow 2\mathbf{N}$. \square

Esimerkki 4.10: Joukot \mathbf{N} ja $\mathbf{N} \times \mathbf{N}$ ovat yhtä mahtavat, sillä $f(i, j) = \frac{1}{2}(i + j)(i + j + 1) + j$ määrittelee bijektio $f: \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$.

Selitys: f numeroi parit $(i, j) \in \mathbf{N} \times \mathbf{N}$ seuraavasti:

$f(i, j)$	0	1	j 2	3	4
0	0	2	5	9	14
1	1	4	8	13	
i 2	3	7	12		
3	6	11	...		
4	10	16			
5	15				

\square

Joukot voidaan jakaa mahtavuutensa perusteella kolmeen luokkaan. Joukko A on

- äärellinen (finite), jos $A \approx \{1, \dots, n\}$ (eli $|A| = n$) jollain $n \in \mathbf{N}$;
- numeroituvasti ääretön (countably infinite), jos $A \approx \mathbf{N}$; ja
- ylinumeroituva (uncountable) muuten.

Äärellisiä ja numeroituvasti äärettömiä joukkoja sanotaan **numeroituviksi**. Siis joukko on numeroituva, jos se on "korkeintaan yhtä mahtava" kuin \mathbf{N} .

Edellisen perusteella $\mathbf{N} \times \mathbf{N}$ on numeroituva. Tästä seuraa helposti, että rationaalilukujen joukko on numeroituva. Sen sijaan reaalilukujen joukko voidaan osoittaa ylinumeroituvaksi.

Lause 4.11: Joukko Σ^* on numeroituvasti ääretön millä tahansa äärellisellä aakkostolla Σ .

Todistus: Saamme bijektion $f: \mathbb{N} \rightarrow \Sigma^*$ määrittelemällä

$f(n) =$ leksikografisessa järjestyksessä $(n + 1)$:s merkkijono.

□

Lause 4.12: Millä tahansa aakkostolla Σ Turing-tunnistettavien kielten joukko

$$\text{RE} = \{ L(M) \subseteq \Sigma^* \mid M \text{ on Turingin kone} \}$$

on numeroituvasti ääretön.

Todistus: Määritellään kuvaus $f: \Sigma^* \rightarrow \text{RE}$ seuraavasti:

1. Jos $w = \langle M \rangle$ jollain M , niin $f(w) = L(M)$ tällä M .
2. Muuten $f(w) = \emptyset$.

Koska jokaisella Turingin koneella on koodi, funktio f on surjektio numeroituvasta joukosta Σ^* joukkoon RE. Tästä seuraa, että RE on numeroituva. □

Lause 4.13: Kaikkien kielten joukko $\mathcal{P}(\Sigma^*)$ on ylinumeroituva.

Todistus: Olkoon $f: \mathbb{N} \rightarrow \Sigma^*$ bijektio (kuten lauseessa 4.11). Tehdään vastaoletus, että g on bijektio $\mathbb{N} \rightarrow \mathcal{P}(\Sigma^*)$. Merkitään $w_i = f(i)$ ja $L_i = g(i)$, kun $i \in \mathbb{N}$. Määritellään kieli $L \in \mathcal{P}(\Sigma)$ siten, että jokaisella i kielet L ja L_i eroavat toisistaan merkkijonon w_i kohdalla:

$$L = \{ w_i \mid w_i \notin L_i \}.$$

Millä tahansa i pätee $w_i \in L$, jos ja vain jos $w_i \notin L_i$. Siis $L \neq L_i$ kaikilla i , joten $L \notin \{ L_i \mid i \in \mathbb{N} \}$ ja g ei ole surjektio; ristiriita. \square

Edellisen todistuksen tekniikkaa sanotaan **diagonalisoimiseksi**. (Oleellisesti samalla tekniikalla voidaan todistaa reaalilukujen joukko ylinumeroituvaksi.)

Korollari 4.14: On olemassa ei-tunnistettavia kieliä.

Todistus: Edellisen perusteella $RE \neq \mathcal{P}(\Sigma^*)$. \square

Ryhdyimme nyt todistamaan kielen A_{TM} ratkeamattomuutta. Kiinnitetään Turingin koneille koodaus jossain aakkostossa Σ^* . Valitaan koodaus siten, että jokaisella Turingin koneella M on tasan yksi koodi $\langle M \rangle \in \Sigma^*$ (mutta kaikilla $w \in \Sigma^*$ ei tarvitse olla olemassa konetta M , jolla $w = \langle M \rangle$).

Teknisenä valmisteluna sovimme ensin

1. Turingin koneille numeroinnin M_1, M_2, M_3, \dots ja
2. jonon aakkoston Σ merkkijonoja c_1, c_2, c_3, \dots

siten, että

3. annetusta merkkijonosta $w \in \Sigma^*$ voidaan päätellä, onko $w = c_i$ jollain $i \in \mathbb{N}$, ja mikä tällöin on vastaava M_i .

Tätä varten määritellään kaikkien koodien joukko

$$C = \{ \langle M \rangle \mid M \text{ on Turingin kone} \}.$$

Olkoot koodit leksikografisessa järjestyksessä

$$C = \{ c_1, c_2, c_3, \dots \}.$$

Kaikilla $i \in \mathbb{N}$ olkoon M_i se Turingin kone, jolla $\langle M_i \rangle = c_i$. Koodaus on helppo tehdä niin, että myös "dekkoodaus" (ehto 3 edellä) onnistuu.

Keskeisenä aputuloksena todetaan [Sipser s. 181–183]

Lemma 4.15: Diagonaalikieli

$$D = \{ c_i \mid c_i \notin L(M_i) \}$$

ei ole Turing-tunnistettava.

Todistus: Esitetään ensin tiivis todistus, ja sitten hieman selventäviä kommentteja.

Tehdään vastaoletus, että D on Turing-tunnistettava. Tällöin $D = L(M_i)$ jollain M_i edellä esitetyssä numeroinnissa. Nyt saadaan ristiriita tarkastelemalla merkkijonoa c_i :

$$\begin{aligned} c_i \in L(M_i) &\Leftrightarrow c_i \in D \\ &\Leftrightarrow c_i \notin L(M_i) \end{aligned}$$

M_i :n valinta
 D :n määritelmä.

□

Todistuksen diagonalisointi-idean valaisemiseksi määritellään ääretön taulukko $T(\cdot, \cdot)$, missä

$$T(i, j) = \begin{cases} 1 & \text{jos } c_j \in L(M_i) \\ 0 & \text{jos } c_j \notin L(M_i). \end{cases}$$

Taulukon rivi $T(i, \cdot) = (T(i, 1), T(i, 2), T(i, 3), \dots)$ kertoo, mitkä koodijoukon $\{c_1, c_2, c_3, \dots\}$ merkkijonot kuuluvat Turing-tunnistettavaan kieleen $L(M_i)$.

Määritellään $d(i) = 1$, jos $c_i \in D$, ja $d(i) = 0$ muuten. Siis jono $d(\cdot) = (d(1), d(2), d(3), \dots)$ kertoo, mitkä koodijoukon $\{c_1, c_2, c_3, \dots\}$ merkkijonot kuuluvat kieleen D . Koska

$$d(i) = \begin{cases} 0 & \text{jos } c_i \in L(M_i) \\ 1 & \text{jos } c_i \notin L(M_i), \end{cases}$$

jono $d(\cdot)$ on taulukon T diagonaalin "komplementti": $d(i) = 1 - T(i, i)$.

Siis $d(\cdot)$ ei ole taulukon rivi $T(i, \cdot)$, koska ne eroavat positiossa i . Tämä tarkoittaa, että D ei ole Turing-tunnistettava kieli $L(M_i)$.

Koska tämä pätee kaikilla i , niin D ei ole Turing-tunnistettava.

Esimerkki: Oletetaan, että numerointi näyttää seuraavalta:

$$\begin{aligned}
 L(M_1) \cap C &= \{c_1, c_4, c_5, \dots\} \\
 L(M_2) \cap C &= \{c_3, c_4, c_6, \dots\} \\
 L(M_3) \cap C &= \{c_1, c_2, c_5, \dots\} \\
 L(M_4) \cap C &= \{c_1, c_2, c_4, \dots\} \\
 L(M_5) \cap C &= \{c_4, c_5, \dots\} \\
 L(M_6) \cap C &= \{c_2, c_3, c_4, \dots\} \\
 &\dots
 \end{aligned}$$

Saadaan seuraava taulukko (diagonaali alleviivattu):

T	c_1	c_2	c_3	c_4	c_5	c_6	\dots
M_1	<u>1</u>	0	0	1	1	0	\dots
M_2	0	<u>0</u>	1	1	0	1	\dots
M_3	1	1	<u>0</u>	0	1	0	\dots
M_4	1	1	0	<u>1</u>	0	0	\dots
M_5	0	0	0	1	<u>1</u>	0	\dots
M_6	0	1	1	1	0	<u>0</u>	\dots
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots

Diagonaalista $(1, 0, 0, 1, 1, 0, \dots)$ saadaan $d = (0, 1, 1, 0, 0, 1, \dots)$.

Lause 4.16: [Sipser Thm. 4.11] Kieli A_{TM} ei ole ratkeava.

Todistus: Tehdään vastaoletus, että jokin ratkaisija R tunnistaa kielen A_{TM} . Siis syötteellä $\langle M, w \rangle$ kone R hyväksyy, jos $w \in L(M)$, ja muuten hylkää.

Diagonaalikieli $D = \{c_i \mid c_i \notin L(M_i)\}$ voidaan tunnistaa (ja peräti ratkaista) Turingin koneella, joka syötteellä w toimii seuraavasti:

1. Jos w ei ole c_i millään i , niin **hylkää**.
2. Muodosta $\langle M_i, w \rangle$, missä i on se indeksi, jolla $w = c_i$.
3. Simuloi konetta R syötteellä $\langle M_i, w \rangle$. Jos R hyväksyi, niin **hylkää**. Jos R hylkäsi, niin **hyväksy**.

Numeroinnit valittiin edellä siten, että kohta 2 voidaan toteuttaa helposti. Mutta D ei ole Turing-tunnistettava; ristiriita. \square

Korollaari 4.17: [Sipser Cor. 4.23] Kieli $\overline{A_{TM}}$ ei ole Turing-tunnistettava.

Todistus: Tiedämme, että A_{TM} on Turing-tunnistettava. Jos myös $\overline{A_{TM}}$ olisi Turing-tunnistettava, niin A_{TM} olisi ratkeava (harjoitus 10.8; [Sipser Thm. 4.22]). \square

Korollaari 4.18: [Sipser Thm. 5.1] Kieli $HALT_{TM}$ on ratkeamaton.

Todistus: Tehdään vastaoletus, että $HALT_{TM}$ olisi ratkeava. Kieli A_{TM} voitaisiin ratkaista Turingin koneella, joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Jos $\langle M, w \rangle \notin HALT_{TM}$, niin **hylkää**.
2. Simuloi konetta M syötteellä w . Jos M hyväksyi, niin **hyväksy**. Jos M hylkäsi, niin **hylkää**.

Vastaoletuksen nojalla kohdan 1 ehto voidaan ratkaista, ja kohtaan 2 mennessä tiedetään, että simulaatio ei jää silmukkaan. Mutta A_{TM} tiedetään ratkeamattomaksi. \square

Edellä esitetyt kielten A_{TM} ja $HALT_{TM}$ ratkeamattomuustodistukset ovat esimerkkejä **palautuksesta** (reduction).

Intuitiivisesti ongelman A palauttaminen ongelmaan B tarkoittaa, että

- Oletetaan, että meillä on proseduurin P_B , joka ratkaisee ongelman B .
- Muodostetaan proseduurin P_A , joka ratkaisee ongelman A **käyttäen aliruutiinina** proseduuria P_B .

Kun ongelma A on palautettu ongelmaan B , voidaan tehdä päätelmiä **kahteen** suuntaan:

1. Jos proseduurin P_B **todella on olemassa**, olemme saaneet toimivan proseduurin myös ongelmalle A .
2. Jos ongelma A tiedetään ratkeamattomaksi, niin proseduuria P_B **ei voi olla olemassa**, joten myös B on ratkeamaton.

Käytännön tietojenkäsittelyssä palautusten soveltaminen suuntaan 1 on keskeistä (aliohjelmakirjastot).

Ratkeamattomuustulosten todistukset ovat usein **epäsuoria** ja perustuvat suunnan 2 käyttöön.

Jos ongelma A voidaan palauttaa ongelmaan B , merkitsemme

$$A \leq B.$$

Käytämme merkintää tässä epämuodollisesti ilman tarkkaa matemaattista määritelmää. Tarkka määritelmä voidaan tehdä eri tavoilla, jolloin puhutaan esim. kuvauspalautuksesta $A \leq_m B$ [Sipser luku 5.3] tai Turing-palautuksesta $A \leq_T B$ [Sipser luku 6.3]. Intuitiivinen tulkinta on joka tapauksessa, että

ongelma A voidaan ratkaista ongelman B avulla,

joten jossain mielessä

B on ainakin yhtä vaikea kuin A .

Kun halutaan tietää, onko jokin ongelma X ratkeava, voidaan siis yrittää kahdensuuntaisia palautuksia:

- Jos $A \leq X$, missä A on ratkeamaton, niin X on ratkeamaton.
- Jos $X \leq B$, missä B on ratkeava, niin X on ratkeava.

Edellä lauseen 4.16 todistuksessa $X = A_{\text{TM}}$ ja $A = D$. Vastaavasti korollarissa 4.18 $X = \text{HALT}_{\text{TM}}$ ja $A = A_{\text{TM}}$. Sama idea toistuu jatkossa.

Edellä todettiin äärellisten automaattien ja yhteydettömien kielten tyhjyysoongelmat E_{DFA} ja E_{CFG} ratkeaviksi. Kuitenkin

Lause 4.19: [Sipser Thm. 5.2] Turingin koneiden tyhjyysoongelma

$$E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ on Turingin kone ja } L(M) = \emptyset \}$$

on ratkeamaton.

Todistus: Tehdään palautus hyväksymisongelmasta A_{TM} . Osoitamme, että mistä tahansa Turingin koneesta M ja merkkijonosta w voidaan muodostaa Turingin kone M_1 , jolla on seuraava ominaisuus:

$$L(M_1) = \begin{cases} \{w\} & \text{jos } w \in L(M) \\ \emptyset & \text{muuten.} \end{cases}$$

Jos E_{TM} olisi ratkeava, kieli A_{TM} voitaisiin ratkaista algoritmilla, joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Muodosta edellä mainitun lainen kone M_1 .
2. Jos $\langle M_1 \rangle \in E_{\text{TM}}$, niin **hylkää**. Muuten **hyväksy**.

Koska todellisuudessa A_{TM} **ei** ole ratkeava, myös E_{TM} on ratkeamaton.

Kone M_1 toimii syötteellä x seuraavasti:

1. Jos $x \neq w$, niin **hylkää**.
2. Muuten simuloi konetta M syötteellä x . Jos M hyväksyisi, niin **hyväksy**. Jos M hylkäisi, niin **hylkää**.

Nyt koneella M_1 on haluttu ominaisuus

$$L(M_1) = \begin{cases} \{w\} & \text{jos } w \in L(M) \\ \emptyset & \text{muuten.} \end{cases}$$

Huomaa, että konetta M_1 rakennettaessa ei tarvitse tietää, kumpi vaihtoehto pätee.

Jos nyt R olisi ratkaisija ongelmalle E_{TM} , niin ongelma A_{TM} voitaisiin ratkaista koneella, joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Muodosta syötteestä $\langle M, w \rangle$ edellä kuvattu kone M_1 .
2. Simuloi konetta R syötteellä $\langle M_1 \rangle$. Jos R hyväksyisi, niin **hylkää**. Jos R hylkäisi, niin **hyväksy**.

Kone M_1 saadaan koneesta M lisäämällä suunnilleen $|x|$ tilaa, jotka tarkastavat syötteen x . Siis annetusta $\langle M, w \rangle$ osataan helposti muodostaa $\langle M_1 \rangle$. \square .

Kielen E_{TM} ratkeamattomuus on erikoistapaus **Ricen lauseesta**.

Kieli A on **semanttinen ominaisuus**, jos se voidaan esittää muodossa

$$A = \{ \langle M \rangle \mid L(M) \in \mathcal{S} \}$$

jollain **joukolla kieliä** $\mathcal{S} \subseteq \mathcal{P}(\Sigma^*)$. Esim. E_{TM} on semanttinen ominaisuus; siinä $\mathcal{S} = \{ \emptyset \}$ (joukko, jonka ainoa alkio on tyhjä kieli).

Siis jos A on semanttinen ominaisuus, kysymyksen ”päteekö $\langle M \rangle \in A$ ” vastaus riippuu vain koneen M hyväksymästä kielestä. Kääntäen kieli A **ei ole** semanttinen ominaisuus, jos joillain M_1 ja M_2 pätee $L(M_1) = L(M_2)$, mutta $\langle M_1 \rangle \in A$ ja $\langle M_2 \rangle \notin A$.

Semanttinen ominaisuus A on **triviaali**, jos joko kaikilla M pätee $\langle M \rangle \in A$ tai kaikilla M pätee $\langle M \rangle \notin A$.

Lause 4.20 (Rice): [Sipser Problem 5.28] Jokainen ei-triviaali semanttinen ongelma on ratkeamaton.

Todistus sivuutetaan, mutta perustuu oleellisesti samaan ideaan kuin lauseen 4.19 todistus. \square

Lause 4.21: Turingin koneiden epätyhjyysongelma

$$\tilde{E}_{\text{TM}} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

on Turing-tunnistettava.

Todistus: Kieli \tilde{E}_{TM} voidaan tunnistaa epädeterministisellä Turingin koneella, joka syötteellä $\langle M \rangle$ toimii seuraavasti:

1. Valitse epädeterministisesti merkkijono w .
2. Simuloi konetta M syötteellä w . Jos M hyväksyisi, niin hyväksy. Jos M hylkäisi, niin hylkää.

□

Korollaari 4.22: Turingin koneiden tyhjyysongelma E_{TM} ei ole Turing-tunnistettava.

Todistus: Tyhjyysongelman komplementti voidaan esittää muodossa

$$\overline{E_{\text{TM}}} = \tilde{E}_{\text{TM}} \cup B,$$

missä B koostuu merkkijonoista, jotka eivät ole muotoa $\langle M \rangle$ millekään M . Emme ole tarkemmin määritelleet koodausta $\langle \cdot \rangle$, mutta minimivaatimus järkevälle koodaukselle on, että B on ratkeava. Siis $\overline{E_{\text{TM}}}$ on Turing-tunnistettava. Koska E_{TM} ei ole ratkeava, se ei ole edes Turing-tunnistettava (vrt. korollaari 4.17). □

Lause 4.23: [Sipser Thm. 5.4] Turingin koneiden ekvivalenssiongelma

$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ ja } M_2 \text{ ovat Turingin koneita ja } L(M_1) = L(M_2) \}$
on ratkeamaton

Todistus: Tehdään vastaoletus, että EQ_{TM} on ratkeava. Olkoon M_\emptyset jokin Turingin kone, jolla $L(M_\emptyset) = \emptyset$. Nyt E_{TM} voidaan ratkaista Turingin koneella, joka syötteellä $\langle M \rangle$ toimii seuraavasti:

1. Muodosta $w = \langle M, M_\emptyset \rangle$.
2. Jos $w \in EQ_{TM}$, niin hyväksy; muuten hylkää.

Mutta E_{TM} on ratkeamaton; ristiriita. \square

Itse asiassa pätee voimakkaammin

Lause 4.24: [Sipser Thm. 5.30] Kumpikaan kielistä EQ_{TM} ja $\overline{EQ_{TM}}$ ei ole Turing-tunnistettava.

Todistus: melko suoraviivainen, mutta sivuutetaan. \square

Todetaan lopuksi ilman todistusta, että esim. seuraavat yhteydettömiin kielioppeihin liittyvät ongelmat ovat ratkeamattomia:

- onko G moniselitteinen,
- päteekö $L(G_1) \cap L(G_2) = \emptyset$,
- päteekö $L(G) = \Sigma^*$ ja
- päteekö $L(G_1) = L(G_2)$.

Täsmällisemmin esim. viimeinen kohta tarkoittaa, että kieli

$$EQ_{CFG} = \{ \langle G_1, G_2 \rangle \mid G_1 \text{ ja } G_2 \text{ ovat CFG:itä ja } L(G_1) = L(G_2) \}$$

on ratkeamaton.

5. Laskettavuus, kieliopit, logiikka

Lopuksi tarkastelemme Turingin koneeseen (tai yleisemmin algoritmeihin) perustuvaa laskettavuuden käsitettä suhteessa kahteen muuhun tärkeään tiedonkäsittelyformalismiin:

Kieliopit: Turing-tunnistettavat kielet liitetään säännöllisten ja yhteydettömien kanssa [Chomskyn hierarkiaan](#).

Logiikka: Ratkeamattomuus on läheisessä yhteydessä formaalista logiikasta tuttuun [epätäydellisyyteen](#).

Tavoitteena on vetää yhteen kurssin teemoja ja asettaa niitä laajempaan yhteyteen. Teknisiin yksityiskohtiin ei juuri kiinnitetä huomiota (eikä niitä kysytä kokeessa).

Chomskyn hierarkia

Rajoittamaton kielioppi on nelikko $G = (V, \Sigma, R, S)$, missä

1. V on äärellinen muuttujien joukko,
2. Σ on päätesymbolien joukko, jolla $\Sigma \cap V = \emptyset$,
3. R on äärellinen joukko sääntöjä muotoa $u \rightarrow v$, missä $u \in (\Sigma \cup V)^+$ ja $v \in (\Sigma \cup V)^*$ ja
4. $S \in V$ on lähtösymboli.

Tämä laajentaa yhteydettömiä kielioppeja siten, että säännön $u \rightarrow v$ vasen puoli u saa olla mikä tahansa epätyhjä merkkijono, ei pelkästään yksi muuttujasymboli.

Kuten yhteydettömien kielioppien tapauksessa, merkitsemme $w \Rightarrow w'$, ja sanomme että w johtaa suoraan merkkijonon w' , jos voidaan kirjoittaa $w = xuy$ ja $w' = xvy$, missä $(u \rightarrow v) \in R$. Jos $w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_{n-1} \Rightarrow w_n$, sanomme että w_0 johtaa merkkijonon w_n , ja merkitsemme $w_0 \xRightarrow{*} w_n$.
Kieliopin tuottama kieli on $L(G) = \{ w \in \Sigma^* \mid S \xRightarrow{*} w \}$.

Esimerkki 5.1: Kieli

$$A = \{ a^i b^i c^i \mid i \in \mathbf{N} \}$$

tunnetusti ei ole yhteydetön. Se voidaan kuitenkin tuottaa rajoittamattomalla kieliopilla

$$\begin{aligned} S &\rightarrow aAbc \mid abc \mid \varepsilon \\ A &\rightarrow aAbC \mid abC \\ Cb &\rightarrow bC \\ Cc &\rightarrow cc, \end{aligned}$$

missä on käytetty samoja merkintäkonventioita kuin yhteydettömille kieliopille.

Esim. merkkijonolle aaabbbccc saadaan johto

$$\begin{aligned} S &\Rightarrow aAbc \Rightarrow aaAbCbc \Rightarrow aaabCbCbc \Rightarrow aaabCbbCc \\ &\Rightarrow aaabbCbCc \Rightarrow aaabbbCCc \Rightarrow aaabbbCcc \Rightarrow aaabbbccc. \end{aligned}$$

□

Kielioppien ja Turingin koneiden yhteys on seuraava:

Lause 5.2: Kieli on Turing-tunnistettava, jos ja vain jos jokin rajoittamaton kielioppi tuottaa sen.

Todistushahmotelma: Kun on annettu rajoittamaton kielioppi G , kieli $L(G)$ voidaan luetella seuraavasti:

1. Käy läpi kaikki yhden pituiset johdot $S \Rightarrow w_1$ ja tulosta kaikki päättemerkkijonot $w_1 \in \Sigma^*$.
2. Käy läpi kaikki kahden pituiset johdot $S \Rightarrow w_1 \Rightarrow w_2$ ja tulosta kaikki päättemerkkijonot $w_2 \in \Sigma^*$.
3. Käy läpi kaikki kolmen pituiset johdot $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3$ ja tulosta kaikki päättemerkkijonot $w_3 \in \Sigma^*$.
4.

Siis $L(G)$ on Turing-tunnistettava (lause 3.9).

Mielenkiintoisempi suunta on muodostaa annetulle Turingin koneelle $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ kielioppi $G = (V, \Sigma, R, S)$, jolla $L(G) = L(M)$.

Perusidea on tulkita Turingin koneen tilanteet

$$u_1 u_2 \dots u_n q v_1 v_2 \dots v_n, \quad u_i, v_i \in \Gamma, q \in Q,$$

merkkijonoiksi. Tätä varten valitaan kieliopin muuttujiksi $V = Q \cup (\Gamma - \Sigma)$, jolloin tilanteet ovat suoraan aakkoston $V \cup \Sigma$ merkkijonoja.

Turingin koneen laskennan esittämiseksi liitetään kielioppiin sääntöjä seuraavasti:

- Jos $\delta(r, a) = (s, b, R)$, lisätään sääntö $ra \rightarrow bs$.
- Jos $\delta(r, a) = (s, b, L)$, lisätään sääntö $cra \rightarrow scb$ kaikilla $c \in \Gamma$.

Nyt Turingin koneen laskenta-askelta $uqv \vdash u'q'v'$ vastaa kieliopin suora johto $uqv \Rightarrow u'q'v'$.

Laskennan alun ja lopun vaatimat yksityiskohdat sivuutetaan (ks. esim. Sudkamp: Languages and Machines, tai Laskennan teoria luennot). \square

Yhteydettömien kielioppien ohella toinen tärkeä erikoistapaus on **yhteysherkkät** (context-sensitive) kieliopit. Tällaisessa kieliopissa kaikilla säännöillä $u \rightarrow v$ pitää olla $|u| \leq |v|$. Poikkeuksena sallitaan kuitenkin sääntö $S \rightarrow \varepsilon$ edellyttäen, että S ei esiinny minkään säännön oikealla puolella. Kieli A on yhteysherkkä, jos $A = L(G)$ jollain yhteysherkkällä G .

Esimerkki 5.3: Yhteydettömän kieliopin säännöissä $u \rightarrow v$ pätee aina $|u| = 1$. Poistamalla ε -säännöt (kuten Chomskyn normaalimuodon yhteydessä) yhteydetön kielioppi saadaan yhteysherkkään muotoon. Siis yhteydettömät kielet ovat yhteysherkkiä.

Toisaalta esimerkki 5.1 itse asiassa osoittaa kielen $\{ a^i b^i c^i \mid i \in \mathbf{N} \}$ yhteysherkkäksi. Siis kaikki yhteysherkkät kielet eivät ole yhteydettömiä. \square

Nimitys "yhteysherkkä" tulee siitä, että tällaisen kieliopin säännöt voidaan muuntaa muotoon $xAy \rightarrow xwy$, missä $A \in V$, $x, y \in (V \cup \Sigma)^*$ ja $w \in (V \cup \Sigma)^+$. Siis sääntöä $A \rightarrow w$ saadaan soveltaa vain "kontekstissa" $x _ y$.

Lause 5.4: Yhteysherkät kielet ovat ratkeavia.

Todistushahmotelma: Erikoistapausta $S \Rightarrow \varepsilon$ lukuunottamatta missä tahansa yhteysherkän kieliopin johdossa

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$$

pätee

$$1 \leq |w_1| \leq |w_2| \leq \dots \leq |w_n|.$$

Siis erityisesti jos kysytään jostakin merkkijonosta w , päteekö $S \xRightarrow{*} w$, niin riittää tarkastella johtoja, joiden välivaiheiden w_i pituudet $|w_i|$ ovat korkeintaan $|w|$. Koska tällaisia välivaiheita on äärellinen määrä, voidaan esim. käydä läpi kaikki niiden järjestykset ja katsoa, muodostuuko laillinen johto. \square

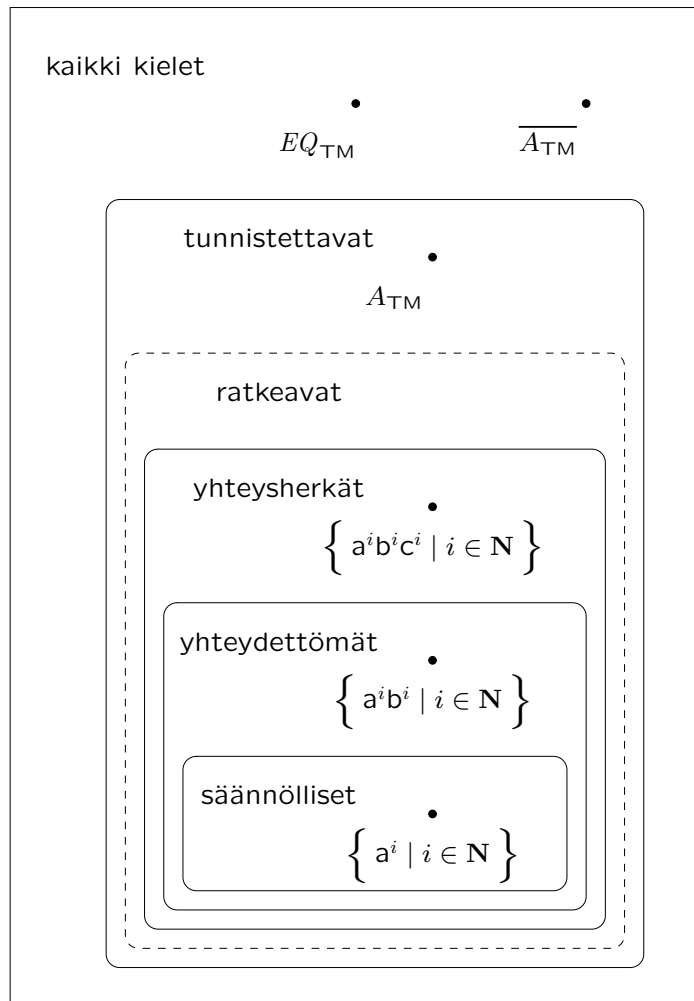
Tarkemmin voidaan osoittaa, että kieli on yhteysherkkä, jos ja vain jos se voidaan tunnistaa **lineaarisesti rajoitetulla automaatilla** (linear-bounded automaton, LBA). Tällainen automaatti on epädeterministinen Turingin kone, joka ei käytä nauhatilaa enempää kuin syötteen pituuden verran, ts. ei koskaan kirjoita mitään tyhjämärkin päälle.

Yleisiä, yhteysherkkiä, yhteydettömiä ja oikealle lineaarisia kielioppeja kutsutaan vastaavasti tyyppin 0, 1, 2 ja 3 kielioppeiksi. Saadaan seuraava Chomskyn hierarkia:

tyyppi	kieli	kielioppi	automaatti
0	tunnistettava	rajoittamaton	Turingin kone
1	yhteysherkkä	yhteysherkkä	lin. rajoitettu
2	yhteydetön	yhteydetön	pinoautom.
3	säännöllinen	oikealle lineaarinen	äärellinen autom.

(Oikealle lineaarista kielioppeista ks. harjoitus 6.) Ylempi taso sisältää myös kaikkien alempien tasojen kielet. Lisäksi tiedämme, että

- kaikki kielet eivät ole edes tunnistettavia,
- tasot ovat erillisiä (esim. on olemassa yhteysherkkiä ei-yhteydettömiä kieliä) ja
- yhteysherkkät \subset ratkeavat \subset tunnistettavat.



Chomskyn hierarkia ja eräiden kielten sijainti sen suhteen

Laskettavuus ja logiikka

Matematiikassa ja logiikassa **todistus** on keskeinen käsite. Yleisesti todistus on argumentti, jolla lukija vakuutetaan jonkin väitteen pätevyydestä.

Jotta todistukset todella olisivat vakuuttavia, niiden pitää perustua yhteisesti hyväksytyihin täsmällisiin sääntöihin.

Ääritapaus täsmällisyydestä on **formaali todistus**. Tällöin todistamisessa sallitut säännöt on kirjattu niin yksityiskohtaisesti, että annetun todistuksen oikeellisuus voidaan tarkistaa vaikka tietokoneella.

Täysin formaalit todistukset ovat yleensä hyvin hankalia. Käytännön matematiikassa tyydytään miltei aina vähäisempään täsmällisyyden asteeseen. Formaaliin todistamiseen liittyvät kysymykset ovat kuitenkin keskeisiä matematiikan perusteiden tarkastelussa.

Tarkastelemme seuraavassa logiikan formalisointia laskettavuuden näkökulmasta. **Varoitus:** yksityiskohdat sivuutetaan.

Tyypillinen tapa formalisoida päättelyä on määritellä **aksiomia** ja **päättelysääntöjä**.

Aksiomat ovat väittämiä, joita voidaan todistuksessa pitää annettuina. Tyypillinen esimerkki aksiomasta on

$$(\forall x)(\forall y)(\forall z)((x = y) \wedge (y = z)) \rightarrow (x = z),$$

joka esittää yhtäsuuruusrelaation transitiivisuuden.

Päättelysäännöt esitetään tyypillisesti muodossa

$$\frac{\psi_1, \dots, \psi_n}{\phi}.$$

Tämä sääntö tarkoittaa, että jos väittämät ψ_1, \dots, ψ_n on jo saatu todistetuksi, voidaan edelleen päätellä ϕ . Perusesimerkki päättelysäännöstä on **modus ponens**

$$\frac{\psi, \quad \psi \rightarrow \phi}{\phi}.$$

Väittämän ϕ todistus annetuista aksioomista ja päättelysäännöistä on jono ψ_1, \dots, ψ_n , missä

- $\psi_n = \phi$ ja
- kaikilla $1 \leq i \leq n$ pätee
 - ψ_i on aksiooma tai
 - on olemassa päättelysääntö

$$\frac{\theta_1, \dots, \theta_k}{\psi_i},$$

jolla $\{\theta_1, \dots, \theta_k\} \subseteq \{\psi_1, \dots, \psi_{i-1}\}$.

Oletetaan jatkossa, että aksioomien ja päättelysääntöjen joukot (sopivasti koodattuna) ovat ratkeavia. Tämä on edellä esitetyn motivaation valossa luonteva (mutta ei pakollinen) rajoitus.

Huomaa, että aksioomia ja päättelysääntöjä voi silti olla ääretön määrä.

Jos aksiomien ja päättelysääntöjen joukot ovat ratkeavia, myös kaikkien **todistusten** joukko

$$P = \{ \langle \psi_1, \dots, \psi_n \rangle \mid \psi_1, \dots, \psi_n \text{ on todistus kaavalle } \psi_n \}$$

on selvästi **ratkeava**.

Tästä seuraa edelleen, että **todistuvien väitteiden** joukko

$$T = \{ \langle \phi \rangle \mid \text{on olemassa } \psi_1, \dots, \psi_{n-1}, \text{ joilla } \langle \psi_1, \dots, \psi_{n-1}, \phi \rangle \in P \}$$

on **Turing-tunnistettava**. Kieli T voidaan luetella esim. seuraavasti:

1. Alusta $n := 1$.
2. Etsi leksikografisessa järjestyksessä ensimmäinen ϕ , jolla $\langle \psi_1, \dots, \psi_{n-1}, \phi \rangle \in P$.
3. Tulosta $\langle \phi \rangle$. Aseta $\psi_n := \phi$ ja $n := n + 1$. Palaa kohtaan 2.

Sen sijaan todistuvien väittämien joukko T ei välttämättä ole ratkeava. Lyhyenkin väittämän ϕ todistuksessa voi tarvita välivaiheina pitkiä väittämiä ψ_i . Tällöin ylläoleva luettelija ei tuota kieltä T leksikografisessa järjestyksessä.

Onko T todella ratkeava vai ei riippuu siitä, mitä nimenomaisia aksioomia ja päättelysääntöjä tarkastellaan.

Esim. luonnollisten lukujen Peanon aksioomien tapauksessa T ei ole ratkeava.

Toisaalta jos järjestelmä on **täydellinen** (complete) ja **ristiriidaton** (consistent), eli kaikilla ϕ pätee joko $\langle \phi \rangle \in T$ tai $\langle \neg\phi \rangle \in T$ mutta ei molemmat, niin T on ratkeava. Kysymys "pätee $\langle \phi \rangle \in T$ " voidaan näet ratkaista seuraavasti:

1. Simuloi edelläesitettyä kielen T luettelijaa, kunnes se tulostaisi $\langle \phi \rangle$ tai $\langle \neg\phi \rangle$.
2. Jos tulostettavana on $\langle \phi \rangle$, niin **hyväksy**.
Jos tulostettavana on $\langle \neg\phi \rangle$, niin **hylkää**.

(Asiat esitetään tarkemmin kurssilla Matemaattinen logiikka.)

Todetaan vielä seuraava kokonaislukujen aritmetiikkaan liittyvä keskeinen ratkeamattomuustulos:

Lause 5.5: Ei ole olemassa algoritmia, joka ratkaisee, onko annetulla kokonaislukukertoimisella polynomilla nollakohtia kokonaislukujen joukossa. Ts. kieli

$\{ \langle p \rangle \mid p(x_1, \dots, x_n) \text{ on kokonaislukukertoiminen polynomi}$
ja on olemassa $a_1, \dots, a_n \in \mathbf{Z}$ joilla $p(a_1, \dots, a_n) = 0 \}$

ei ole ratkeava. \square

Tästä seuraa yleisemmin, että ei ole olemassa menetelmää testata, onko jokin kokonaislukuja koskeva väittämä tosi. Huomaa, että tämä on eri asia kuin testata *toteutuvuutta* jossain formaalissa järjestelmässä (esim. Peanon aksioomat).

Lopuksi

Kurssin sisältöä voidaan tarkastella kahdesta näkökulmasta:

1. Perustiedot formaaleista kielistä ja niiden tunnistamisesta; esim.

- kielen määrittelemisen äärellisen automaatin, säännöllisen lausekkeen tai yhteydettömän kieliopin avulla,
- em. formalismien väliset yhteydet,
- Turingin kone yleisenä algoritmin mallina ja
- ratkeamattomuuden alkeet, kuten pysähtymisongelma.

2. Johdatus tietojenkäsittelyteoriaan ja sen metodiikkaan; erityisesti

- matematiikan soveltaminen laskennan mallintamiseen ja
- miten väitteet perustellaan täsmällisesti.

Käydään lyhyesti läpi kurssin sisältöä tältä kannalta.

Säännölliset kielet

Käytännössä tärkeä tietää:

- tilasiirtymäkone laskennan mallina
- säännölliset lausekkeet ja äärelliset automaattit

Teoreettisia ajatusmalleja:

- epäterministinen laskenta
- mallien väliset konversiot (NFA \rightarrow DFA)
- laskulaitteen ja kuvausformalismin ekvivalenssi (DFA vs. säännöllinen lauseke)
- luokan sulkeumaominaisuudet
- mahdottomuustodistukset (pumppauslemma)

Jatkoaiheita: äärellisen automaatin yleistykset; äärelliset automaattit merkkijonoalgoritmeissa

Yhteydettömät kielet

Käytännössä tärkeä tietää:

- kielen kuvaaminen kieliopilla
- jäsentämisen peruskäsitteet, erityisesti jäsennyypuu

Teoreettisia ajatusmalleja: samat kuviot kuin säännöllisillä kielillä, teknisesti haastavammassa tilanteessa

Teknistä:

- muunnos PDA \rightarrow CFG ja yhteydettömien kielten pumppaaminen esimerkkejä hieman vaikeammista konstruktioista
- CYK-algoritmi ja taulukointitekniikka

Jatkoihteita: sovellukset ohjelmointikielissä ja luonnollisessa kielessä

Turingin koneet ja laskettavuus

Käytännössä tärkeä tietää:

- Churchin-Turingin teesi; universaali Turingin kone
- ratkeamattomuuden käsite; pysähtymisongelma

Teoreettisia ajatusmalleja:

- palautustekniikka mahdottomuustodistuksissa
- kielioppeihin liittyvät ratkeavuustulokset

Teknistä: diagonalisointia tarvitsee, jos näiden asioiden parissa jatkaa

Jatkoaiheita: laskennan vaativuus; (matemaattinen) logiikka

Muista antaa kurssipalaute!

— Loppu —