TIEP1020 Diskreetit rakenteet

Antti Valmari

Jyväskylän yliopisto
Informaatioteknologian tiedekunta

Symboleita

$=_M$	18	(tällä kurssilla) yhtäsuuruus modulo M eli $n \bmod M = m \bmod M$
$\div_M$	20	(tällä kurssilla) kertolaskun käänteistoiminto aritmetiikassa modulo M
$\neg$	53	negaatio eli "ei"
$\wedge$	53	konjunktio eli "ja"
$\vee$	53	disjunktio eli "tai"
$\rightarrow$	53	totuusarvojen implikaatio eli "jos ... niin"
$\leftrightarrow$	53	totuusarvojen ekvivalenssi eli "jos ja vain jos"
$\Rightarrow$	77	looginen implikaatio eli "jos ... niin"
$\Leftrightarrow$	77	looginen ekvivalenssi eli "jos ja vain jos"
$\lfloor \ldots \rceil$	85	(tällä kurssilla) "on määritelty"
$\forall$	127	universaalikvanttori eli "jokaisella"
$\exists$	127	eksistenssikvanttori eli "on olemassa"
Γ	77	(tällä kurssilla) yleisoletusten joukko

$\emptyset$	94	tyhjä joukko
$\{a_1, \ldots, a_n\}$	93	äärellinen joukko, jonka alkiot ovat $a_1, \ldots, a_n$
$\{a_1, a_2, \ldots\}$	93	ääretön joukko, jonka alkiot ovat $a_1, a_2, \ldots$
$\{a \in A \mid \varphi(a)\}$	94	niiden alkioiden joukko, jotka kuuluvat A:han ja toteuttavat φ:n
$\in$	94	kuuluu joukkoon
$\notin$	94	ei kuulu joukkoon
$\subseteq$	96	osajoukko
$\subset$	97	aito osajoukko
$\cup$	98	unioni
$\cap$	98	leikkaus
$\setminus$	99	joukkojen erotus
$\overline{A}$	100	joukon A komplementti
$\times$	101	joukkotulo
2^A	100	$\{X \mid X \subseteq A\}$ joukon A osajoukkojen joukko
B^A	118	funktioiden $A \mapsto B$ joukko
A^*	103	A:n alkioista muodostettujen äärellisten jonojen joukko
ε	103	tyhjä jono (jono, jossa on 0 alkiota)
Σ	28	aakkosto

$:=$		sijoitusoperaattori pseudokoodissa ja joissakin ohjelmointikielissä
&&	48	"ja"-operaattori, jonka oikea puoli lasketaan vain jos vasen tuottaa true
||	48	"tai"-operaattori, jonka oikea puoli lasketaan vain jos vasen tuottaa false
$\uparrow$	35	"katso osoittimen takaa" -operaattori
$\perp$	35	osoittimen arvo "tietoa ei ole"
$\mathbb{B}$	55	$\{\mathbf{F}, \mathbf{T}\}$ tavallisen logiikan totuusarvot
div	16	kokonaislukuosamäärä
$\mathbf{F}$	53	totuusarvo "epätosi" (false)
mod	16	jakojäännös
$\mathbb{N}$		$\{0, 1, 2, \ldots\}$ luonnolliset luvut
$\mathbb{Q}$		$\{\frac{n}{m} \mid n \in \mathbb{Z} \wedge m \in \mathbb{Z}^+\}$ rationaaliluvut
$\mathbb{Q}^+$		$\{x \in \mathbb{Q} \mid x > 0\}$ positiiviset rationaaliluvut
$\mathbb{Q}^-$		$\{x \in \mathbb{Q} \mid x < 0\}$ negatiiviset rationaaliluvut
$\mathbb{R}$		reaaliluvut (kaikki lukusuoran luvut)
$\mathbb{R}^+$		$\{x \in \mathbb{R} \mid x > 0\}$ positiiviset reaaliluvut
$\mathbb{R}^-$		$\{x \in \mathbb{R} \mid x < 0\}$ negatiiviset reaaliluvut
syt	21	suurin yhteinen tekijä
$\mathbf{T}$	53	totuusarvo "tosi" (true)

U	83	totuusarvo "määrittelemätön" (undefined)
$\mathbb{Z}$		$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ kokonaisluvut
$\mathbb{Z}^+$		$\{1, 2, 3, \ldots\}$ positiiviset kokonaisluvut
$\mathbb{Z}^-$		$\{\ldots, -3, -2, -1\}$ negatiiviset kokonaisluvut

1 Johdanto

1.1 Kurssin tavoite

Tällä kurssilla käsitellään *ohjelmoinnissa* ja *tietojenkäsittelytieteessä* tarvittavaa matematiikkaa

- vrt. sisällysluettelo
- suurin osa asioista on mukana tärkeytensä vuoksi
- itsessään vähemmän tärkeitä asioita on mukana mm. kertomassa sovelluksista tai siksi, että ne sopivat hyvin päättelytaidon kehittämiseen

Ohjelmoijien tuottavuudessa on moninkertaisia eroja

- monen mielestä tämä on tosielämässä ilmeistä
- tieteellinen kirjallisuus tukee väitettä
- väite on pyritty osoittamaan urbaaniksi legendaksi, mutta ei vakuuttavasti
- kirjallisuusviitteitä, keskustelua ym.: Steve McConnell (blogi 9.1.2011): Origins of 10X — How Valid is the Underlying Research?
- väitteestä ei seuraa, että pitää palkata vain huippuja
- väitteestä kyllä seuraa, että ohjelmoinnin opetusta ja opiskelua kannattaa yrittää muuttaa paremmaksi

Okei, mutta matematiikkako muka auttaa?

- oikeutettu kysymys!
  - yleisesti ajatellaan ohjelmistoalalla tarvittavan matemaattisia taipumuksia
  - silti monen ammattilaisen mielestä on tosielämässä ilmeistä, että vastaus on "ei"
  - tieteellinen kirjallisuus yms. on niukkaa, mutta seuraavien lainausten suuntaista
- T. C. Lethbridge, What knowledge is important to a software professional?, IEEE Computer Issue 5, May 2000: "*Because of the low importance and high forgetability of continuous mathematics and basic science, universities and colleges should either place less emphasis on these topics or they should teach them in a way that makes them more relevant to software engineering students.*"
- ACM Computing Curriculum 2013: "*We recognize that general facility with mathematics is an important requirement for all CS students. Still, . . . For example, an understanding of linear algebra plays a critical role in some areas of computing such as graphics and the analysis of graph algorithms. However, linear algebra would not necessarily be a requirement for all areas of computing (indeed, many high quality CS programs do not have an explicit linear algebra requirement). . . . we note that undergraduate CS students need enough mathematical maturity to have the basis on which to then build CS-specific mathematics . . . , which, importantly, does not explicitly require any significant college-level coursework in calculus, differential equations, or linear algebra.*"

$\Rightarrow$ ohjelmoijien matematiikkaa on olemassa, mutta perinteinen paketti ei ole sitä

- tärkeintä on osata ajatella matemaattis-loogisesti

Pohjana kurssin suunnittelulle olivat

- ACM Computing Curriculum: Software Engineering ja Computer Science
  - perusteellisesti ja pitkäjänteisesti mietitty kansainvälinen näkemys, mitä SWE- ja CS-tutkintojen tulee vähintään sisältää
  - maailmalla laajassa käytössä
- paikka opetusohjelmassa
- opettajan laaja kokemus ohjelmoinnista ja teoreettisesta tietojenkäsittelytieteestä
  - vilkaistaanpa "Papers I like (part 1) | The ryg blog", 12.8.2017
- kollegojen neuvot
- MathCheckin tarjoamat mahdollisuudet opiskelun tukena

MathCheck

- ohjelma, joka tarkastaa ratkaisuja vaihe vaiheelta, ei pelkästään lopputulosta
  $\Rightarrow$ mahdollistaa mm. ratkaisun korjaamisen omatoimisesti
- sisältää tämän kurssin kannalta hyödyllisiä tehtävälajeja, joita opetusohjelmissa ei tavallisesti ole
  - lausekepuutehtävät
  - logiikkatehtävät
  - modulaarinen aritmetiikka
- keskeneräinen
  $\Rightarrow$ kaikki ei toimi parhaalla mahdollisella tavalla

Kurssin suoritustapa

- harjoituksilla hankitaan lupa tulla tenttiin
  - kerättävä vähintään minimipistemäärä
  - pisteiden saanti perustuu vastausten hyväntahtoiseen tarkastukseen tai (automatisoiduissa tehtävissä) jopa vain opiskelijan omaan ilmoitukseen
- arvosana määräytyy yksinomaan tentin perusteella
- opettaja on kokeillut vuosien saatossa monenlaisia harjoituspistejärjestelmiä
  - kannustusvaikutus on vaikea saada juuri oikeaksi
  - päähuomio ei saa ohjautua siihen, onko vai eikö jokin vastaus pisteen arvoinen
  - on mahdotonta selvittää, missä määrin opiskelija teki kotitehtävät itse
  - ei ole hyvä kannustaa unohtamaan asia heti kun siitä on pisteet saatu!
- automatiikka mahdollistaa omatoimisen harjoittelun
  $\Rightarrow$ ehdotus: harjoittele, kunnes koet osaavasi riittävästi!
  - toki opettaja ymmärtää, että kaikki eivät tähtää arvosanaan 5 (juuri siksi aineistossa on helppoja ja vaikeita osuuksia)
- opettaja pyrkii siihen, että työmäärä olisi oikea
  - 5 op on virallisesti 133 tuntia mediaaniopiskelijan työtä kaikki työ huomioiden
  - se tarkoittaisi n. 14 tuntia / viikko, kohtuuttoman paljon!
  - olisiko 10 tuntia / viikko kohtuullinen eli $4 + 2 + 4$?

Yksi tehtävä tentissä tulee olemaan esseetehtävä (painoarvo 10% . . . 15%)

- aiheen pitää olla jokin kurssilla käsitelty aihe, jota ei käsitelty laskuharjoituksissa
- saat valita aiheesi ihan itse (ym. rajauksella)
- voit valmistella esseetä etukäteen, mutta se pitää tuoda tenttiin päässä eikä paperilla
- pisteitä voi saada proosatekstilläkin, mutta täydet pisteet vaativat edes hieman matematiikkaa
- sopiva pituus on puolesta sivusta yhteen sivuun

Taustatiedot

- ei oleteta, että muita matematiikan kursseja on jo suoritettu
- oletetaan, että opiskelija tarvittaessa ottaa eri asenteen kuin lukiossa
  - tavoitteena on asioiden *ymmärtäminen*, ei mekaaninen tekeminen
- ohjelmointiosaamisesta on paljon hyötyä, mutta välttämätöntä se ei ole

Kurssi pyrkii hyödyntämään omaa sisältöään sekä etu- että jälkikäteen

- esim. symboleita $\Rightarrow$ ja $\Leftrightarrow$ käytetään ennen systemaattista esittelyä
  - niiden merkitys selitetään "kevyesti" ensimmäisen käytön yhteydessä

  $\Rightarrow$ niistä tulee alustavasti tuttuja systemaattiseen käsittelyyn mennessä
- esim. kielioppeja hyödynnetään logiikan kielten esittelemisessä
- monenlaisia päättelyitä on pitkin matkaa

Esityksessä käytettävä värikoodaus

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

Muutama neuvo

- täsmällisen määrittelemisen taito on hyvin tärkeää!
  - opitaan harjoittelemalla, aloittamalla sopivan helpoista tehtävistä
  - esim. autojonon viimeisen mustan auton jälkeen tulee vain punaisia autoja
  - sin mus val mus pun pun toteuttaa ja sin mus val mus pun sin rikkoo
  - entä jos viimeinen auto on musta, tai entä jos jonossa ei ole mustia autoja?
- sama tieto voidaan esittää tietokoneessa monin eri tavoin
  - jotkin ovat ihmiselle luontevia
  - jotkin mahdollistavat tehokkaan käsittelyn
  - esim. viikkolukujärjestys + poikkeukset vs. oppimistapahtumien luettelo

  $\Rightarrow$ on hyvin tärkeää pohtia vaihtoehtoisia esitystapoja!
- päättelytaito on hyvin tärkeää!
  - sekin opitaan harjoittelemalla, aloittamalla sopivan helpoista tehtävistä
- älä keskity yksityiskohtien ulkoaopetteluun, vaan niiden taustalla olevien periaatteiden ymmärtämiseen!
  - esim. onko helppo muistaa tämä salakirjoitus? (videolta C. Brabrand ja J. Andersen: Teaching Teaching & Understanding Understanding, 2006)

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

1	2	3
4	5	6
7	8	9

1	2	3
4	5	6
7	8	9

1.2 Esimerkki

Epätyhjästä taulukosta `korkeus` on löydettävä mahdollisimman pitkän osuuden, jonka loppukohta on vähintään yhtä korkealla kuin alkukohta, pituus

- esim. pitkän kuntolenkin sisältä tilastointikelpoinen pituus

Helppo algoritmi C++:lla

- kokeillaan kaikki alku- ja loppukohdat, ja valitaan paras tulos

```
unsigned helppo( unsigned korkeus[], unsigned nn ){
  unsigned paras = 0;
  for( unsigned ii = 0; ii < nn; ++ii ){
    for( unsigned jj = ii; jj < nn; ++jj ){
      if( korkeus[ jj ] >= korkeus[ ii ] && jj - ii > paras ){
        paras = jj - ii;
      }
    }
  }
  return paras;
}
```

- helppo vakuuttautua, että toimii oikein
- hidas

Ajan kulutus sekunteina

nn	1 000	3 162	10 000	31 623	100 000	316 228	1 000 000
helppo	0	0	0	2	23	238	2 344
nopea	0	0	0	0	0	0	0

Nopea algoritmi C++:lla

```
unsigned nopea( unsigned korkeus[], unsigned nn ){
```

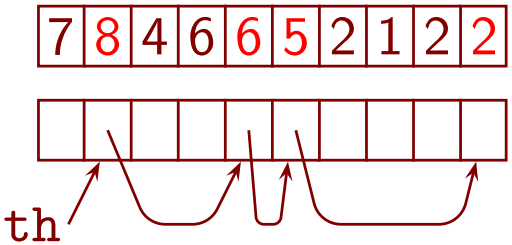
- etsitään "takahuiput"

```
  unsigned th = nn-1, *seuraava_th = new unsigned[ nn ];
  for( unsigned ii = nn; ii--; ){
    if( korkeus[ ii ] > korkeus[ th ] ){
      seuraava_th[ ii ] = th; th = ii;
    }
  }
```

  - kohdat, joiden jälkeen ei enää tule vähintään yhtä korkeita kohtia

⇒ *mahdollisimman pitkän osuuden loppukohta on takahuippu*

  - `th` on ensimmäisen takahuipun kohta
  - `seuraava_th[ ii ]` on seuraavan takahuipun kohta
  - *seuraava takahuippu on edellistä alempana*
  - *viimeisessä kohdassa on takahuippu*



- etsitään paras tulos

```
unsigned paras = 0;
for( unsigned ii = 0; ii < nn; ++ii ){
  while( korkeus[ th ] >= korkeus[ ii ] ){
    if( th - ii > paras ){ paras = th - ii; }
    if( th == nn-1 ){ delete[] seuraava_th; return paras; }
    th = seuraava_th[ th ];
  }
}
```

  - on helppo nähdä, että algoritmi kokeilee vain korkeusehdon toteuttavia osuuksia ja valitsee kokeilemistaan parhaan

- voiko (jokainen) kaikkein paras osuus $a \ldots \ell$ jäädä kokeilematta?
  - `ii` kokeilee jokaisen alkukohdan

  $\Rightarrow$ `ii` kokeilee parhaan osuuden alkukohdan a
  - parhaan osuuden loppukohta ℓ on takahuippu
  - jos `ii` $< a$, niin `korkeus[`ℓ`]` $<$ `korkeus[ii]` (miksi?)

  $\Rightarrow$ `th` ei ohita kohtaa ℓ kun `ii` $< a$

  $\Rightarrow$ suorituksen aikana on hetki, jolloin `ii` $= a$ ja `th` $= \ell$

  $\Rightarrow$ ohjelma tutkii jokaisen kaikkein parhaan osuuden

- tähän ei koskaan tulla, mutta C++-kääntäjäni suuttuu ilman sitä

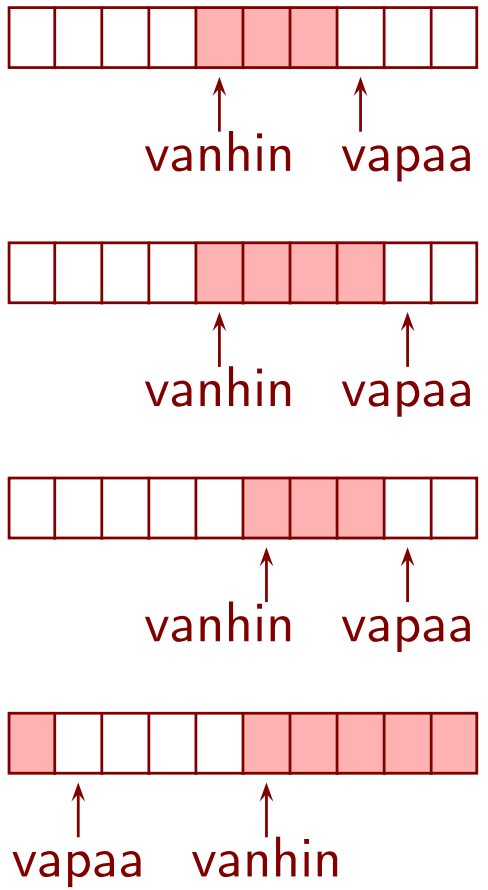
```
  delete[] seuraava_th; return paras;
}
```

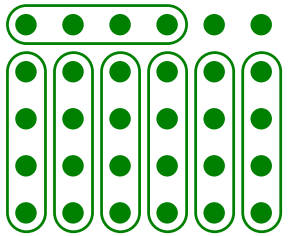
2 Lukujen ja merkkien esitys tietokoneessa

2.1 Modulaarinen aritmetiikka

Miksi tärkeä?

- tietokoneet eivät laske kokonaisluvuilla, vaan esim. modulo $4\,294\,967\,296 = 2^{32}$
- rengasmaiset rakenteet ovat yleisiä, esim. rengaspuskuri
- pedagoginen syy tällä opintojaksolla: tarjoaa mahdollisuuden antaa automaattisesti palautetta päättelytehtävistä
- tärkeä salausmenetelmä RSA perustuu siihen





Kokonaislukujen jakolaskuja on kahta lajia

- tulos on reaaliluku
  - esim. $\frac{30}{7} = 4\frac{2}{7} \approx 4{,}2857$
- tulos muodostuu *osamäärästä* ja *jakojäännöksestä*, jotka molemmat ovat kokonaislukuja
  - merkitsemme osamäärää $n \operatorname{div} m$ ja jakojäännöstä $n \bmod m$
  - esim. $30 \operatorname{div} 7 = 4$ ja $30 \bmod 7 = 2$
  - osamäärä on pyöristetty alaspäin, eli $n \operatorname{div} m = \lfloor \frac{n}{m} \rfloor$
  - $0 \leq$ jakojäännös $<$ |jakaja|
  - nollalla ei voi jakaa

Jälkimmäiselle kokonaislukujen jakolaskulle pätee *jakoyhtälö*

$$n = m(n \operatorname{div} m) + (n \bmod m)$$

- $n \operatorname{div} m$ on osamäärä — n on jaettava
- $n \bmod m$ on jakojäännös — m on jakaja
- $0 \leq n \bmod m < |m|$

Jakojäännökselle pätee

$$\ldots = (n - m) \bmod m = n \bmod m = (n + m) \bmod m = (n + 2m) \bmod m = \ldots$$

Ohjelmointikielissä on yleistä, että

- `n/m` tarkoittaa $\approx$ kokonaislukujen jakoa, jos `n` ja `m` ovat kokonaislukutyyppiä
- esim. C++
  - `std::cout << "4/3  = " << 4/3  << '\n';` tulostaa `4/3  = 1`
  - `std::cout << "4/3. = " << 4/3. << '\n';` tulostaa `4/3. = 1.33333`

$\Rightarrow$ toisin kuin matematiikassa, kaikki kolmoset eivät ole samanlaisia!

- `n%m` tarkoittaa jakojäännöstä, ainakin jos `n` ja `m` ovat kokonaislukutyyppiä
- jakoyhtälö pätee muuten, mutta $0 \leq$ `n%m` $< |$`m`$|$ ei aina päde, jos `n` < 0 tai `m` < 0
  - Javassa `-9%4` tuottaa -1
  - ennen 2011 C++ jätti avoimeksi, tuottaako `-9%4` 3 vai -1
- katso Wikipedia Modulo operation

Aritmetiikassa modulo M

- $M \in \mathbb{Z}^+$ eli M on positiivinen kokonaisuluku
- jokaisen yhteen-, vähennys- ja kertolaskun lopuksi tuloksesta otetaan jakojäännös jaettuna M:llä

$\Rightarrow$ tulos on aina kokonaisluku väliltä $0, 1, \ldots, M-1$

- kertolaskun käänteisfunktio vaatii uudenlaisen jakolaskun (kolmas jakolasku tänään!)
  - $(3 \operatorname{div} 7) \bmod 10 = 0$, mutta $3 \div_{10} 7 = 9$, koska $(9 \cdot 7) \bmod 10 = 63 \bmod 10 = 3$

$\Rightarrow$ käsittelemme jakolaskun, potenssilaskun ja juuret myöhemmin

Esim. aritmetiikassa modulo 5

- $3 + 4 =_5 2$, koska $7 \bmod 5 = 2$
- $3 - 4 =_5 4$, koska $-1 \bmod 5 = 4$
- $3 \cdot 4 =_5 2$, koska $12 \bmod 5 = 2$

Toinen tapa ajatella: $\ldots, n-2M, n-M, n, n+M, n+2M, \ldots$ ajatellaan yhtäsuuriksi

- esim. jos $M = 5$, niin $\ldots =_M -12 =_M -7 =_M -2 =_M 3 =_M 8 =_M 13 =_M \ldots$
- siis $n =_M m \Leftrightarrow \exists k \in \mathbb{Z} : n = m + kM \Leftrightarrow n \bmod M = m \bmod M$
  - on olemassa kokonaisluku k siten, että $n = m + kM$
  - esim. $-12 =_5 18$, koska $-12 = 18 + (-6) \cdot 5$
- lukujen $m + kM$ edustajana käytetään sitä, joka on välillä $0, \ldots, M-1$

Sama toisesta näkökulmasta

- kokonaisluvun n edustajana käytetään sitä lukua muotoa $n + kM$ ($k \in \mathbb{Z}$), jolle $0 \leq n + kM < M$
  - se on olemassa ja yksikäsitteinen
- esim. kun $M = 5$, lukujen 3, 18 ja -27 edustaja on 3, koska esim. $-27 + 6 \cdot 5 = 3$
- se on $n \bmod M = n - (n \operatorname{div} M)M$
- se löytyy toistuvasti lisäämällä tai vähentämällä M kunnes tulos on välillä $0, \ldots, M - 1$

Vielä yksi tapa ajatella yhtäsuuruutta modulo M

- $n =_M m$ jos ja vain jos $n - m$ on jaollinen M:llä
- todistus: $n =_M m \;\Leftrightarrow\; \exists k \in \mathbb{Z} : n = m + kM \;\Leftrightarrow\; \exists k \in \mathbb{Z} : n - m = kM$

Yhteen-, vähennys- ja kertolaskun tapauksessa välivaiheissa ei tarvitse laskea $\bmod M$

- $(\ (n \bmod M) + (m \bmod M)\ ) \bmod M$ $\quad x \bmod M = x - (x \operatorname{div} M)M$
  $= (\ n - (n \operatorname{div} M)M + m - (m \operatorname{div} M)M\ ) \bmod M$ $\quad$ vaihdetaan värejä
  $= (\ n - (n \operatorname{div} M)M + m - (m \operatorname{div} M)M\ ) \bmod M$ $\quad$ ryhmitellään uudelleen
  $= (\ n + m - ((n \operatorname{div} M) + (m \operatorname{div} M))M\ ) \bmod M$ $\quad$ M:n monikerran poisto
  $= (\ n + m\ ) \bmod M$

- $$\begin{aligned} & (\ (n \bmod M)(m \bmod M) &&) \bmod M \\ = & (\ (n - (n \operatorname{div} M)M)(m - (m \operatorname{div} M)M) &&) \bmod M \\ = & (\ nm - n(m \operatorname{div} M)M - (n \operatorname{div} M)Mm + (n \operatorname{div} M)M(m \operatorname{div} M)M) &&) \bmod M \\ = & (\ nm &&) \bmod M \end{aligned}$$

Jakolasku aritmetiikassa $\bmod M$ ei aina onnistu, vaikka jakaja $\neq_M 0$

- merkitsemme (tällä kurssilla) $n \div_M m$, koska $\frac{n}{m}$ tarkoittaa muuta
- $x = n \div_M m$ pitää olla luku väliltä $0, \ldots, M-1$ siten, että $(mx) \bmod M = n \bmod M$
- jos $M = 5$, niin jokainen jakolasku onnistuu kun jakaja $\neq_M 0$

$\cdot$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

$\div_5$	0	1	2	3	4
0	–	0	0	0	0
1	–	1	3	2	4
2	–	2	1	4	3
3	–	3	4	1	2
4	–	4	2	3	1

- jos $M = 10$ ja $m = 2$, niin $mx = 2x$ ja $(mx) \bmod M = (2x) \bmod 10$ ovat parillisia

$\Rightarrow$ $(2x) \bmod 10$ ei voi olla 3

$\Rightarrow$ $3 \div_{10} 2$ ei ole olemassa

- sekä $(2 \cdot 2) \bmod 10 = 4$ että $(7 \cdot 2) \bmod 10 = 4$, joten kumpi $4 \div_{10} 2$ on, 2 vai 7?
- päätös: puhumme jakolaskusta vain kun osamäärä on olemassa ja yksikäsitteinen

Yhtäsuuruutta $=_M$ käytettäessä yhteen-, vähennys- ja kertolaskuissa, mutta ei jakolaskuissa, voi laskea aivan kuten kokonais- ja reaaliluvuillakin

Tekijä

- olkoot $n \in \mathbb{Z}$ ja $m \in \mathbb{Z}$
- n on m:n *tekijä*, jos ja vain jos on olemassa sellainen kokonaisluku k, että $m = nk$

$\Leftrightarrow$ $n = m = 0$ tai jakolasku m jaettuna n:llä menee tasan

$\Leftrightarrow$ $n = m = 0$ tai $m \bmod n = 0$

Suurin yhteinen tekijä

- olkoot $n \in \mathbb{Z}^+$ ja $m \in \mathbb{Z}^+$ eli n ja m ovat positiivisia kokonaislukuja
- 1 on molempien tekijä
- mikään kokonaisluku, joka on suurempi kuin n, ei voi olla n:n tekijä
  $\Rightarrow$ ei voi olla molempien tekijä

$\Rightarrow$ on olemassa n:n ja m:n *suurin yhteinen tekijä*
  – suurin kokonaisluku, joka on sekä n:n että m:n tekijä

- merkitsemme sitä $\mathrm{syt}(n, m)$
- yleistys negatiivisille: $\mathrm{syt}(-n, m) = \mathrm{syt}(n, -m) = \mathrm{syt}(-n, -m) = \mathrm{syt}(n, m)$

Mitkä luvut voidaan muodostaa n:stä ja m:stä yhteen- ja vähennyslaskuilla?

- esimerkki: jos $n = 50$ ja $m = 80$, niin voidaan muodostaa mm.
  - $80 + 80 + 50 + 50 + 50 = 310$
  - $50 - 80 = -30$
  - $50 - 30 = 20 = 2 \cdot 50 - 80$
- voidaan muodostaa ainakin kaikki luvut muotoa $xn + ym$, missä $x \in \mathbb{Z}$ ja $y \in \mathbb{Z}$
- ei muita, koska $(x_1 n + y_1 m) \pm (x_2 n + y_2 m) = (x_1 \pm x_2)n + (y_1 \pm y_2)m$

Lause: Jos $n \in \mathbb{Z}^+$ ja $m \in \mathbb{Z}^+$, niin $\mathrm{syt}(n, m)$ on pienin positiivinen luku muotoa $xn + ym$, missä $x \in \mathbb{Z}$ ja $y \in \mathbb{Z}$

- esim. $\mathrm{syt}(50, 80) = 10$
  - voidaan muodostaa vain 10:llä jaollisia
  - $10 = -3 \cdot 50 + 2 \cdot 80$

Miksi on varmaa, että "pienin positiivinen ..." on olemassa?

- ei itsestään selvää
  - kokonaislukujoukossa $\{-6, -5, -4\}$ ei ole pienintä positiiviista
  - reaalilukujoukossa $\{\ldots, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}\}$ ei ole pienintä positiiviista
- luvuissa $xn + ym$ (missä $x \in \mathbb{Z}$ ja $y \in \mathbb{Z}$) on positiivisia, esim. $1n + 0m = n$
- jokaisessa epätyhjässä alhaalta rajoitetussa $\mathbb{Z}$:n osajoukossa on pienin alkio

Syt-lauseen todistus

- tarkastellaan kaikkia lukuja muotoa $xn + ym$, missä $x \in \mathbb{Z}$ ja $y \in \mathbb{Z}$
- merkitsemme pienintä positiivista $s = x'n + y'm$
- jos k on sekä n:n että m:n tekijä, niin k on jokaisen luvun $xn + ym$ tekijä
  - todistus: $n = kn'$ ja $m = km' \Rightarrow xn + ym = xkn' + ykm' = k(xn' + ym')$

$\Rightarrow$ jokainen n:n ja m:n yhteinen tekijä on myös s:n tekijä

$\Rightarrow$ (1) $\mathrm{syt}(n, m)$ on s:n tekijä

- myös $n \bmod s$ on tarkasteltavaa muotoa, koska $n \bmod s = n - (n \operatorname{div} s)s = n - (n \operatorname{div} s)(x'n + y'm) = (1 - (n \operatorname{div} s)x')n - ((n \operatorname{div} s)y')m$
- jakoyhtälössä luvataan, että $n \bmod s < s$

$\Rightarrow$ $n \bmod s = 0$, koska muutoin syntyisi ristiriita s:n valinnan kanssa

- jos pätisi $n \bmod s > 0$, niin $n \bmod s$ olisi pienempi positiivinen tarkasteltavaa muotoa oleva luku kuin s

$\Rightarrow$ (2) s on n:n tekijä

- samasta syystä (3) s on myös m:n tekijä

$\Rightarrow$ (1), (2) ja (3) $\Rightarrow$ $\mathrm{syt}(n, m) = s$

Jakolasku $n \div_M m$ onnistuu jos ja vain jos $m \neq_M 0$ ja $\mathrm{syt}(m, M) = 1$

- olkoon $s = \mathrm{syt}(m, M)$
  $\Rightarrow$ reaalilukujen jakolaskut $\frac{m}{s}$ ja $\frac{M}{s}$ menevät tasan
- jos $s \neq 1$ ja x toteuttaa ehdon $mx =_M n$ eli $(mx) \bmod M = n \bmod M$, niin
  - $s > 1$
  - $x \neq_M x + \frac{M}{s}$, koska $0 < \frac{M}{s} < M$
  - $x + \frac{M}{s}$ toteuttaa ym. ehdon, koska $m(x + \frac{M}{s}) = mx + \frac{m}{s}M =_M mx =_M n$

  $\Rightarrow$ osamäärä on monikäsitteinen

$\Rightarrow$ jos $s \neq 1$, niin osamäärä joko ei ole olemassa tai on monikäsitteinen

- jos $s = 1$, niin
  - on olemassa $z \in \mathbb{Z}$ ja $y \in \mathbb{Z}$ siten, että $zm + yM = 1$

  $\Rightarrow nzm + nyM = n$
  $\Rightarrow mnz =_M n$
  $\Rightarrow nz$ toteuttaa ehdon $mx =_M n$
  - kun n käy läpi luvut $0, \ldots, M-1$, niin $n \bmod M$ saa M eri arvoa

  $\Rightarrow$ myös $(mnz) \bmod M$ ja $(nz) \bmod M$ saavat M eri arvoa
  $\Rightarrow$ kaikki luvut $0, \ldots, M-1$ tulevat käytetyiksi
  $\Rightarrow$ ei voi olla siten, että $0 \leq x_1 < x_2 < M$ ja $mx_1 =_M mx_2$

$\Rightarrow$ jos $s = 1$, niin osamäärä on olemassa ja yksikäsitteinen
  - $n \div_M m = (n(1 \div_M m)) \bmod M$, missä $1 \div_M m$ on edellä löydetty z
  - ym. lukua z merkitään tavallisesti m^{-1}

Saako jakolaskussa jättää $\bmod M$ laskematta välivaiheissa?

- nimittäjän osalta kysymys ei nouse esiin, koska $n \div_M m$ ei lasketa tavallisen jakolaskun avulla vaan
  - etsimällä se luku m^{-1}, jolle $0 \leq m^{-1} < M$ ja $(mm^{-1}) \bmod M = 1$, ja
  - laskemalla $(nm^{-1}) \bmod M$
- osoittajassa saa: $(n + M)m^{-1} =_M nm^{-1} + Mm^{-1} =_M nm^{-1}$

Potenssilaskussa on sekaannuksen vaara

- esim. jos $M = 5$, niin $2^1 \bmod M = 2$ mutta $2^{1+M} \bmod M = 2^6 \bmod M = 64 \bmod M = 4 \neq 2$
- esim. jos $M = 2^8 = 256$, niin $10^3 \bmod M = 232$, mutta $10^m \bmod M = 0$ jos $m \geq 8$

$\Rightarrow$ eksponentissa ei voida soveltaa yhtäsuuruutta $=_M$

- kantaluvussa voidaan: $n^m = \underbrace{n \cdots n}_{m \text{ kpl.}} =_M \underbrace{(n \bmod M) \cdots (n \bmod M)}_{m \text{ kpl.}} = (n \bmod M)^m$

Juuri $\sqrt[n]{m}^M$ määritellään potenssilaskun käänteistoimintona

- siis jos $\sqrt[n]{m}^M$ on olemassa, niin $\left(\sqrt[n]{m}^M\right)^n =_M m$
- jos monta lukua täyttää tämän ehdon, niin valitaan vaikka pienin
  - reaaliluvuillakin valitaan monesta vaihtoehdosta jokin: $(\sqrt{x})^2 = (-\sqrt{x})^2$

2.2 Bitit, tavut ja sanat

Tiedon esittämisen perusyksikkö on **bitti** (**bit**)

- valinta kahdesta vaihtoehdosta
- esim.
  - 0 tai 1
  - päälla tai pois päältä
  - alhaalla tai ylhäällä
- n bittiä kykenee esittämään valinnan 2^n vaihtoehdosta

$2^3 = 8$ { 000, 001, 010, 011, 100, 101, 110, 111 }

Bitti on pieni yksikkö

$\Rightarrow$ usein käytetään yksikköä **tavu** (**byte**) kerrannaisineen

- tavu = 8 bittiä, kykenee esittämään valinnan 256 vaihtoehdosta
- esim.
  - luvut 0, 1, . . . , 255
  - luvut –128, –127, . . . , –1, 0, 1, . . . , 127
  - `A`, `B`, . . . , `Ö`, `a`, `b`, . . . , `ö`, `0`, `1`, . . . , `9`, `.`, `,`, `?`, `!`, `@`, . . .
  - koira, kissa, lehmä, hevonen, possu, kana, . . .

Tavun kerrannaisista vallitsee sekaannus

SI-järjestelmä	usein käytössä	IEC 60027
1 kB = 1 kilobyte = 1000 B	1 KB = 1 Kilobyte = 1024 B	= 1 KiB
1 MB = 1 megabyte = 10^6 B	1 MB = 1 Megabyte = 2^{20} B	= 1 MiB
1 GB = 1 gigabyte = 10^9 B	1 GB = 1 Gigabyte = 2^{30} B	= 1 GiB
1 TB = 1 terabyte = 10^{12} B	1 TB = 1 Terabyte = 2^{40} B	= 1 TiB

⇒ sekä lukiessa että kirjoittaessa on syytä olla tarkkana

- IEC 60027 olisi yksikäsitteinen, mutta se on huonosti tunnettu
- KiB, MiB, GiB ja TiB luetaan kibibyte, mebibyte, gibibyte ja tebibyte

Tietokone käsittelee tietoa pääsääntöisesti **sana** (**word**) kerrallaan

- merkitys vaihtelee
- nykyisin usein 64 bittiä tai 32 bittiä

⇒ riittävän suuri esittämään käyttökelpoisen lukualueen yms.
  – 32 bittiä: –2 147 483 648, . . . , 2 147 483 647

3230	0	1	0	0	1	1	1	0	
3231	0	1	0	1	0	1	0	0	T
3232	0	1	0	0	1	0	0	1	I
3233	0	1	0	0	0	1	0	1	E
3234	0	1	0	1	0	0	0	0	P
3235	0	0	1	1	0	0	0	1	1
3236	0	0	1	1	0	0	0	0	0
3237	0	0	1	1	0	0	1	0	2
3238	0	0	1	1	0	0	0	0	0
3239	1	0	1	1	0	1	0	1	

Ohjelmien näkökulmasta tietokoneen muisti on iso taulukko tavuja

- tämä näkökulma auttaa ymmärtämään paljon asioita

⇒ kannattaa muistaa

Teoreettisesta näkökulmasta olennaista on vain, että perusvaihtoehtoja on äärellinen määrä ja vähintään 2

- perusvaihtoehtojen joukkoa
  - kutsutaan usein **aakkosto**ksi (**alphabet**)
  - merkitään melko usein Σ (joten $2 \leq |\Sigma| < \infty$)
- isompi määrä vaihtoehtoja saadaan laittamalla aakkosia peräkkäin
  - n aakkosen jono esittää enintään $|\Sigma|^n$ vaihtoehtoa
- käytännössä yleensä $\Sigma = \{0, 1\}$ eli bitit tai $\Sigma =$ 8-bittiset tavut

2.3 Lukujen esitys tietokoneessa

Etumerkittömät (unsigned) kokonaisluvut

- lukuarvot $0, 1, \ldots, 2^B - 1$, missä B on käytettävä bittien määrä
- bittijono $b_{B-1}b_{B-2}\cdots b_2b_1b_0$ edustaa lukuarvoa $\sum_{i=0}^{B-1} b_i 2^i$
  - esim. $01001011 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^3 + 1 \cdot 2^6 = 1 + 2 + 8 + 64 = 75$
- toisinpäin $b_i = (n \operatorname{div} 2^i) \bmod 2$
  - esim. jos $n = 75 = 01001011$, niin
    $b_4 = (75 \operatorname{div} 2^4) \bmod 2 = (75 \operatorname{div} 16) \bmod 2 = 4 \bmod 2 = 0$
- aritmetiikka tapahtuu **modulo** 2^B
  - yhteen-, vähennys- ja kertolaskun tulos pakotetaan välille $0, \ldots, 2^B - 1$ lisäämällä sopiva 2^B:n kerrannainen (jakolaskun tulos on automaattisesti oikealla välillä)
- esim. jos $|\Sigma| = 256$, niin

$$\begin{aligned} 200 + 100 &\rightsquigarrow 300 - 256 = 44 \\ 100 - 200 &\rightsquigarrow -100 + 256 = 156 \\ 10 \cdot 100 &\rightsquigarrow 1000 - 3 \cdot 256 = 232 \end{aligned}$$

- vaikutus on sama kuin laskemalla lasku oikein ja jättämällä liiat bitit pois
  - $1000 = 1111101000$, $232 = 11101000$

  $\Rightarrow$ riittää jättää liiat bitit laskematta
- $n >> i$ siirtää n:n bittikuviota i askelta oikealle
  - vasemmasta reunasta tulee nollia
  - esim. $00101101 >> 2 = 00001011$
  - vaikutus on sama kuin $n \operatorname{div} 2^i$
- vastaavasti $n << i$ siirtää vasemmalle
  - oikeasta reunasta tulee nollia
  - esim. $10001011 << 2 = 00101100$
  - vaikutus on sama kuin $(n2^i) \bmod 2^B$

Etumerkilliset (**signed**) kokonaisluvut

- tässä käsitellään vain esitystapaa nimeltä **kahden komplementti**
- lukuarvot $-2^{B-1}, \ldots, -1, 0, 1, \ldots, 2^{B-1} - 1$, missä B on bittien määrä
- b_{B-1} sisältää etumerkin, mutta etumerkki toimii toisin kuin koulussa
- jos $b_{B-1} = 0$, on lukuarvo $\sum_{i=0}^{B-2} b_i 2^i$, muutoin se on $\left(\sum_{i=0}^{B-2} b_i 2^i\right) - 2^{B-1}$

  $\Rightarrow$ lukuarvo on

$$\left(\sum_{i=0}^{B-2} b_i 2^i\right) - b_{B-1} 2^{B-1}$$

- luvun x vastaluku $-x$ saadaan kääntämällä bitit ja lisäämällä 1 modulo 2^B, paitsi luvulle -2^{B-1}

$$\begin{aligned} 13 &= 01101 \quad \text{ja} \quad -13 = 10010{+}1 = 10011 \\ -2 &= 11110 \quad \text{ja} \quad 2 = 00001{+}1 = 00010 \\ 0 &= 00000 \quad \text{ja} \quad -0 = 11111{+}1 = 00000 \\ -16 &= 10000 \quad \rightsquigarrow \quad 01111{+}1 = 10000 = -16 \end{aligned}$$

- lukualueen ylityksen ja >>-operaattorin toiminta vaihtelee
  - ajoaikainen virhe vai tulos leikattuna käytettäviin bitteihin?
  - osallistuuko etumerkkibitti toimintoon?

Liukuluvut (floating point numbers)

- monia mahdollisuuksia, esim.
  $x = (-1)^s(2^{23} + m)2^{e-150}$

- 0 on esitettävä erikseen, esim. $000 \cdots 0$
  - saattaa olla erikseen $+0$ ja -0
- nollaa lähellä tarvitaan poikkeavasti esitettyjä lukuja
  - muuten voitaisiin esittää $2^{-127} + 2^{-150}$ mutta ei $2^{-127} - 2^{-150}$
- osa bittiyhdistelmistä voidaan varata erikoistarkoituksiin
  - kertomaan, että on tapahtunut virhe kuten nollalla jako
  - $\pm\infty$

Keino tutkia esitystapoja

```
#include <iostream>                    // nn ja ff ovat samassa kohdassa muistia
int main(){
  union { unsigned nn; float ff; } uu;
  std::cin >> uu.ff;
  for( int ii = 32; ii--; ){
    if( ii % 8 == 7 ){ std::cout << ' '; }
    std::cout << (uu.nn >> ii) % 2;
  }
  std::cout << '\n';
}
```

Merkit (**characters**)

- pitkään tärkein standardi oli **ASCII**

0, ..., 31	näkymättömiä ohjausmerkkejä
32, ..., 47	!"# $%&' ()*+ ,-./
48, ..., 63	0123 4567 89:; <=>?
64, ..., 79	@ABC DEFG HIJK LMNO
80, ..., 95	PQRS TUVW XYZ[\]^_
96, ..., 111	`abc defg hijk lmno
112, ..., 126	pqrs tuvw xyz{ |}~
127	näkymätön ohjausmerkki

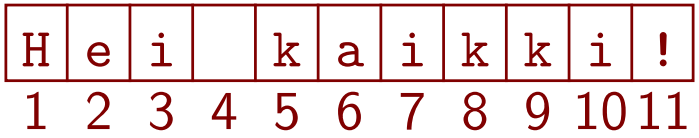
– 7 = äänimerkki, 10 = rivinsiirto, 13 = rivin alkuun

- 128, . . . , 255 otettiin eri tavoin käyttöön lisämerkeille
  - **ISO 8859-1** eli **Latin 1**: esim. 228 = ä
  - **Windows-1252** eli **CP-1252**: edellisen laajennos
  - **ISO 8859-15**: muutettu ISO 8859-1, esim. € mukana ja $^1/_2$ puuttuu
- **Unicode** riittää laajalti ihmisten kirjoitusjärjestelmille
  - 136 755 merkkiä kesäkuussa 2017
  - muutamia vaihtoehtoisia esitystapoja, joista UTF-8 valtasi maailman
- UTF-8-koodaus
  - merkin esittämiseen käytetään 1, . . . , 4 tavua
  - tavut 0xxxxxxx tulkitaan kuten ASCII:ssa
  - tavut 110xxxxx sisältävät 5 hyötybittiä ja niillä on 1 jatkotavu
  - tavut 1110xxxx sisältävät 4 hyötybittiä ja niillä on 2 jatkotavua
  - tavut 11110xxx sisältävät 3 hyötybittiä ja niillä on 3 jatkotavua
  - tavut 10xxxxxx ovat jatkotavuja, niissä on 6 hyötybittiä
  - esim. 195 164 = 11000011 10100100 = ä
- UTF-8 on todella hyvin suunniteltu!
  - ASCII-tiedostot kelpaavat sellaisinaan
  - monet ASCII:lle suunnitellut ohjelmat käsittelevät UTF-8-koodattuja tiedostoja ongelmitta (paitsi erehtyvät merkkien määrässä)
  - tavu ei voi tulla tulkituksi väärässä tehtävässä
  - jos tavuja katoaa välistä, tulkinta synkronoituu nopeasti uudelleen
  - on jätetty laajennusvaraa

3 Rakenteellisen tiedon esitys tietokoneessa

3.1 Taulukot, merkkijonot, tietueet ja osoittimet

Taulukko (array) sisältää nolla tai useampia rakenteeltaan samanlaisia tietoalkioita, joista valitaan haluttu **indeksoimalla**

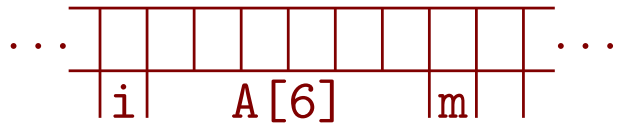


- esim. $A[i] := 0$
- indeksointiin käytetään tyypillisesti kokonaislukua
  - ensimmäinen (eli ensimmäisen alkion) indeksi on C++:ssa 0

  $\Rightarrow$ jos C++:n taulukossa on n alkiota, niin indeksit ovat 0, 1, ..., $n-1$
  - myös 1 esiintyy melko usein ensimmäisenä indeksinä
  - esim. Pascal antaa ohjelmoijan valita ensimmäisen indeksin
  - ensimmäinen indeksi on C++:ssa helppo muuttaa halutuksi: jos haluttu ensimmäinen indeksi on e, niin korvataan kaikkialla $A[i] \rightsquigarrow A[i-e]$
- tietokone löytää nopeasti indeksiä vastaavan taulukon alkion

Kiinteän kokoiset taulukot

- tietokoneen muisti voidaan ajatella isoksi taulukoksi, josta annetaan pätkiä eri tarkoituksiin

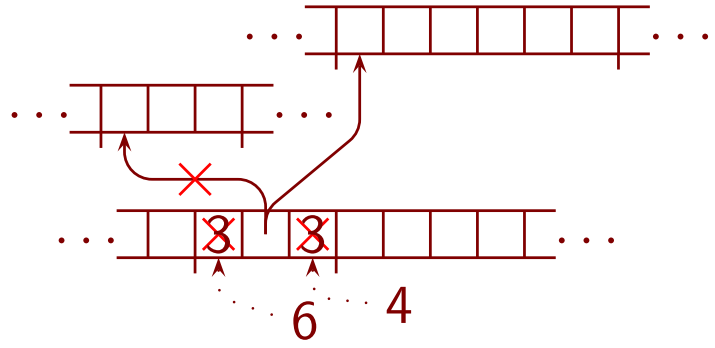


- usein taulukon perästä on varattu muistia johonkin muuhun

$\Rightarrow$ taulukkoa ei yleensä voi kasvattaa alkuperäisellä paikallaan

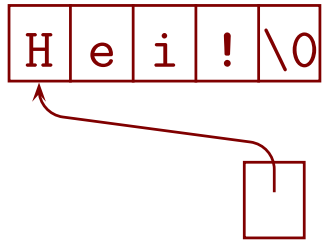
$\Rightarrow$ ohjelmointikielissä oli alkuun vain kiinteän kokoisia taulukoita
  – koko määräytyi käännösaikana tai kun taulukko otettiin käyttöön

Joustavan kokoiset taulukot

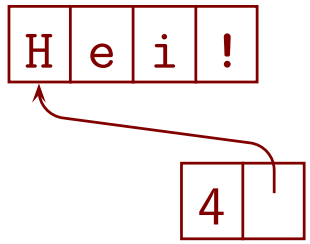


- esim. C++:n `vector` voi kasvaa käytön aikana
- kun sille varattu tila loppuu, varataan muualta suurempi tila ja siirretään koko sisältö sinne
- esimerkkikuvassa nuoli edustaa **osoitinta (pointer)**
  – sisältää **osoitteen (address)** eli kertoo, missä kohdassa muistia jokin on
  – osoite voidaan ajatella luonnolliseksi luvuksi
  – osoittimen arvo voi olla myös $\perp$ eli "tietoa ei ole" (muita nimiä nil, null ja 0, ei tarkoita osoitetta 0 vaan "tietoa ei ole")
  – jos p on osoitin, niin $p\uparrow$ on sen takaa löytyvä tieto
- vanha tila voidaan ottaa myöhemmin muuhun käyttöön
  – uuden tilatarpeen on oltava tarpeeksi pieni mahtuakseen sinne
  – siellä täällä olevien vapaiden muistialueiden hallinnointi on jossain määrin työlästä, mutta käyttöjärjestelmät selviävät siitä ainakin tyydyttävästi
- jos uusi koko on $2 \cdot$ (vanha koko), niin koska $\frac{1}{2} + (\frac{1}{2})^2 + \cdots + (\frac{1}{2})^n < 1$,
  – hukkamuistia on aina vähemmän kuin (käytettyä ja vapaata) hyötymuistia
  – alkio siirretään keskimäärin < 2 kertaa
  $\Rightarrow$ siedettävän vähän hukkatyötä ja -muistia
- jos uusi koko olisi esim. vanha koko + 10, niin hukkamuistia ja -työtä tulisi paljon!

Merkkijonot (**character strings**)



- C:n merkkijono on taulukko, jonka viimeisenä alkiona on 0
  - ei numeromerkki 0, vaan se näkymätön merkki, jonka koodi on 0
  - kokoa ei ole talletettu erikseen, vaan se tunnistetaan loppu-0:sta
  - sisällössä ei voi olla 0-merkkiä, mutta sehän ei olekaan näkyvä merkki
  - **ohjelmoijan on varattava tilaa myös loppu-0:lle!**

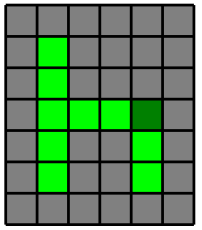


- C++:n `string` on taulukko (tai monimutkaisempi rakenne), jonka koko on talletettu erikseen

Tietue (**record**, **struct**) sisältää tietoalkioita, joista valitaan haluttu nimellä

- ei tarvitse olla rakenteeltaan samanlaisia
- esim.
```
struct henkilo{ int synt_vuosi; string nimi; } hh;
std::cout << "ikä on " << nyt_vuosi - hh.synt_vuosi;
```
- C++:ssa `tietue.kentta` ja `osoitin_tietueeseen->kentta`
- **olio** ja **luokka** (**object** ja **class**) ovat tietueen käsitteen laajennos

Lisää esimerkkejä taulukoista



- CD-levylle tallennettu ääni on taulukko signaalitasoja
  - $2 \cdot 44\,100$ kpl 16-bittistä näytettä / s
- bittikarttakuva on 2-ulotteinen taulukko esim. lukukolmikoita RGB

3.2 Esimerkkejä rakenteellisista tekstuaalisista kielistä

HTML kuvaa www-sivun muodostumisen (loogisista) osista

- **HyperText Markup Language**

HTML-tiedoston yleisrakenne

```
<!DOCTYPE html>
<html lang=fi>
<head>
<meta charset=UTF-8>
<style>
body { background: white; color: black; font-family: sans-serif }
     ...
p.blarm { background: pink; border: solid maroon; padding:1ex }
</style>
<title>TIEP1020 Diskreetit rakenteet</title>
</head>
<body>
     ...
</body>
</html>
```

<body>-osan rakenne

```
<h2>Näin tehdään otsikko</h2>
<p>Tekstiä voi kirjoittaa enimmäkseen normaalisti.
Riveille jako ja   tyhjän   käyttö

    eivät merkitse paljoa.
<em>Näin korostetaan</em> tai <strong>näin</strong>.
Pienempi kuin -merkki tehdään esim. &lt; ja et-merkki &amp;.
<p><a href="http://validator.w3.org/unicorn/">Tällä
sivulla</a> voi tarkastaa, onko www-sivu ehjä.
<p class=loota>Katso esimerkki myöhemmin
```

- seuraavat merkit eivät tekstissä edusta itseään: <, >, & ja "
  - siksi täytyy tehdä muilla keinoin

CSS sisältää komentoja tekstin muotoilemiseksi

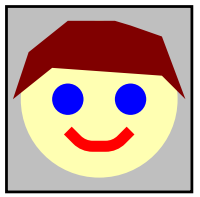
- **Cascading Style Sheets**

```
h2 { color: green; background: white; font: bold
  small-caps 20px normal }
p.loota { border: solid thick; color: maroon;
  background: yellow; text-align: right }
```

- vanhaa perua myös HTML sisältää muotoilukomentoja kuten <b>

Vektorigrafiikkaa voi piirtää esim. LaTeXin `pstricks`-pakkauksella

```
\begin{pspicture}(10,10)
\newrgbcolor{iho}{1 1 0.7}\newrgbcolor{ruskea}{.5 0 0}
\psframe[fillstyle=solid,fillcolor=lightgray](-1,-1)(11,11)
\psset{linecolor=blue}
\pscircle*[linecolor=iho](5,5){5}
\pscircle*(3,5){1}\pscircle *(7,5){1}
\psline [linecolor=red, linearc=4pt,linewidth=2pt
]  ( 3 , 3 ) (4,2)(6,2)(7,3)
\pspolygon*[linecolor=ruskea](-.5,5)(.5,8)(3,
10)(6,10)(7,9.7)(9,9)(10.5,5)(9,6.5)(2,7)
\end{pspicture}
```



Ohjelmointikielet sisältävät monenlaisia rakenteita

- muutamia käsitellään jäljempänä

3.3 Syntaksi

Tekstuaalisissa kielissä on yleensä kaksi rakenteellista tasoa

- **leksikaalinen** taso
  - **tekstialkiot** (**token**): avainsanat, luvut, välimerkit, laskuoperaattorit yms. joissa ei saa olla sisällä tyhjää
  - mitä tyhjä tila on: välilyönnit, rivinsiirrot, sarkaimet, kommentit
- (varsinainen) **syntaksi**: miten tekstialkioita saa laittaa peräkkäin
  - vertaa `-(1+2)*3` ja `*1+)2(3-`

Luonnollisissakin kielissä on (kenties sumeita) leksikaalisia sääntöjä

- seuraavat on tehty arpomalla kirjaimia kolmen edellisen kirjaimen perusteella jakaumalla, joka on laskettu suomen tai englannin sanojen luettelosta
  - saleittantua kuus kea ta-ampuvuotoutiikka punsijäämällipoida poskisti
  - raimplativer tomon cochred tuffrancork anougglashesteng ing proad

Ilmauksien merkitystä kutsutaan **semantiikaksi**

- esim. täsä-o syn daxi päeMÄNTYä mutt, siilti shemandiickan t a j u u
- esim. kuusi on havupuu, joten seitsemän on havupuu plus yksi
- joskus on tärkeää tiedostaa ero ilmauksen ja sen merkityksen välillä
  - vrt. < ja & HTML:ssä; loppu-0, \" ja \\ C:n merkkijonoissa
  - mihin avaruus loppuu? s-kirjaimeen

BNF eli **Backus-Naur form** on laajalti käytetty tapa määritellä syntaksi

- esimerkki: lauseke

$$\begin{aligned}
\mathit{Lauseke} &::= \mathit{Termi} \mid \mathit{Lauseke}\ \text{"+"}\ \mathit{Termi} \mid \mathit{Lauseke}\ \text{"}-\text{"}\ \mathit{Termi} \\
\mathit{Termi} &::= \mathit{Tekijä} \mid \mathit{Termi}\ \text{"}\cdot\text{"}\ \mathit{Tekijä} \mid \mathit{Termi}\ \text{"/"}\ \mathit{Tekijä} \\
\mathit{Tekijä} &::= \mathit{Atomi} \mid \text{"+"}\ \mathit{Atomi} \mid \text{"}-\text{"}\ \mathit{Atomi} \\
\mathit{Atomi} &::= \mathit{Luku} \mid \mathit{Muuttuja} \mid \text{"("}\ \mathit{Lauseke}\ \text{")"}
\end{aligned}$$

- käsitteen tai apukäsitteen rakenne ilmaistaan säännöllä muotoa

$$\mathit{Nimi} ::= \mathit{vaihtoehto} \mid \mathit{vaihtoehto} \mid \cdots \mid \mathit{vaihtoehto}$$

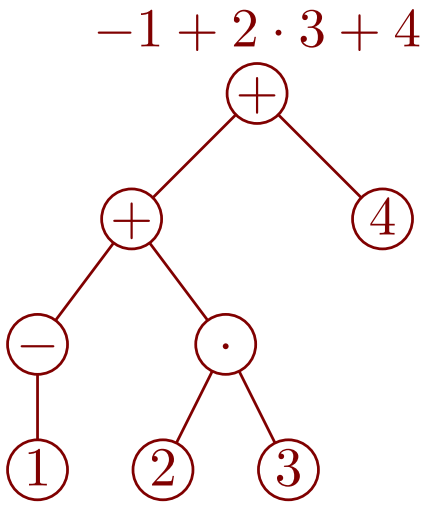
  missä vaihtoehto on ε tai jono käsitteiden nimiä ja/tai tekstialkioita
  - ε tarkoittaa, että ei laiteta mitään, esim. $\mathit{Etumerkki} ::= \varepsilon \mid \text{"+"} \mid \text{"}-\text{"}$

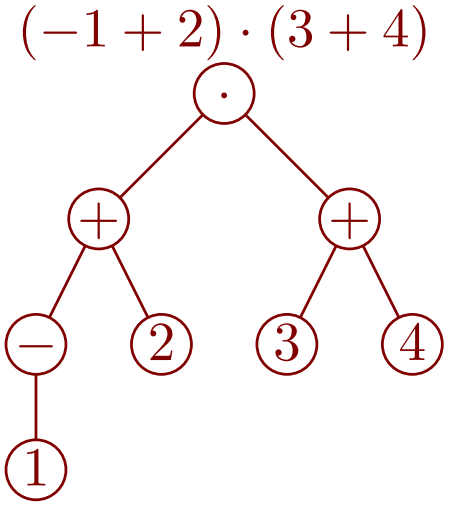
  $\Rightarrow$ ε ei esitä itseään
  - samanmuotoisten toistoa ei BNF:ssä esitetä $\cdots$, vaan kuten $\mathit{Lauseke}$ edellä

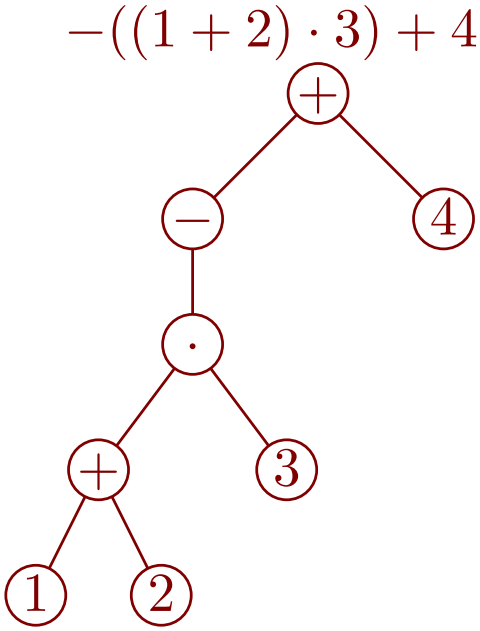
- BNF:stä on monenlaisia muunnelmia

Lausekepuu

- tietokonekielissä lauseke on tekstuaalinen keino ilmaista puumainen rakenne

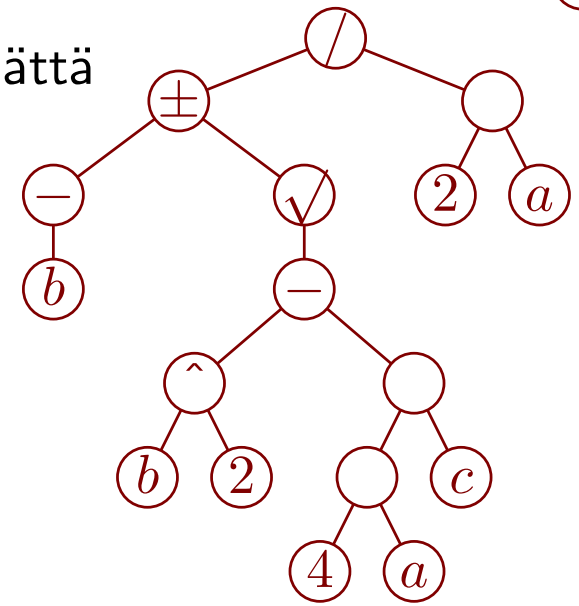






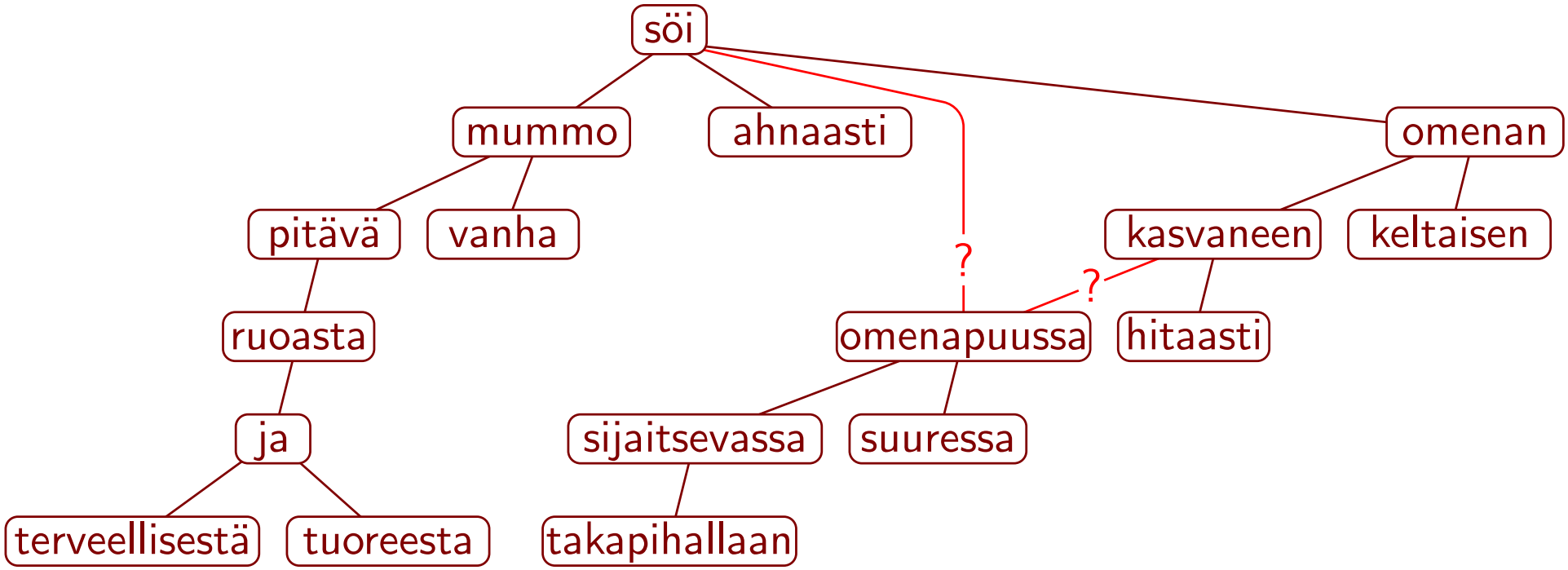
- tällainen puu on **lausekepuu**
- matematiikassa lauseke ei välttämättä ole lineaarista tekstiä

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



Luonnollisessakin kielessä on hierarkkisia rakenteita

terveellisestä ja tuoreesta ruoasta pitävä vanha mummo
söi ahnaasti takapihallaan sijaitsevassa suuressa omenapuussa
hitaasti kasvaneen keltaisen omenan



- joskus ne eivät jäsenny tarkoitetusti

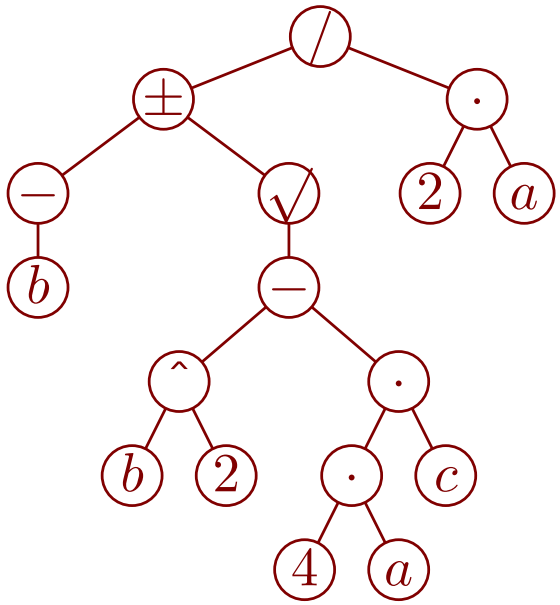
  Molemmat miehet vietiin putkaan. Siellä vuonna 1954 syntynyt mies jatkoi riehumista.

Lauseke (expression)

- kuten aikaisemmin todettiin,
  lauseke esittää puumaisen rakenteen

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- ilmaisee esim.
  - miten arvo lasketaan
  - miten kokonaisuus muodostuu osistaan



Salaisuus, älä kerro kenellekään: tentissä testataan kykyä piirtää lausekepuu

- tentissä älä tee suluista solmua lausekepuuhun
  - sulut ovat apuväline puurakenteen ilmaisemisessa, eivät osa puurakennetta
  - lauseketta käsittelevissä algoritmeissa saat tehdä sellaisen

Piirrettyinä puut vievät paljon tilaa ja piirtäminen on työlästä

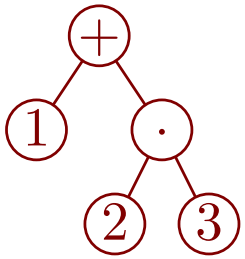
$\Rightarrow$ käytetään tiiviimpiä esitystapoja
- matemaattiset merkinnät
- ohjelmointikielten lausekkeet

- puumainen rakenne voidaan aina esittää siten, että lehdet esitetään sellaisinaan ja muut solmut muodossa (sisältö alipuu ... alipuu)
  - esim. $(/\ (\pm\ (-\ b)\ (\sqrt{}\ (-\ (\hat{}\ b\ 2)\ (\cdot\ (\cdot\ 4\ a)\ c))))\ (\cdot\ 2\ a))$
  - käytössä esim. LISP-kielessä
- matematiikassa on kuitenkin yleistä
  - sijoittaa solmun sisältö keskelle, jos solmulla on kaksi alipuuta
    $(((-\ b)\ \pm\ (\sqrt{}\ ((b\ \hat{}\ 2)\ -\ ((4\ \cdot\ a)\ \cdot\ c))))\ /\ (2\ \cdot\ a))$
  - jättää kertolaskut merkitsemättä
    $(((-\ b)\ \pm\ (\sqrt{}\ ((b\ \hat{}\ 2)\ -\ ((4\ a)\ c))))\ /\ (2\ a))$
  - osoittaa joidenkin alipuiden rajat muilla keinoilla kuin suluilla
    $$\frac{(-\ b)\ \pm\ \sqrt{(b)\ ^2\ -\ ((4\ a)\ c)}}{2\ a}$$
  - käyttää välistystä jossain määrin puurakenteen mukaisesti
    $$\frac{(-b) \pm \sqrt{b^2 - ((4a)c)}}{2a}$$
  - sopia tulkinnasta, jos sulkuja on jätetty pois $\dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- monissa ohjelmointikielissä tehdään samoin, paitsi että tekstin rivit rikkovia keinoja ei ole käytettävissä

$$\sum_{i=1}^{n} \frac{1}{\sqrt{i}}$$

Sitovuustasot (precedence)



- matematiikassa on käytäntö, että kertolasku sitoo voimakkaammin kuin yhteenlasku
  - $1 + 2 \cdot 3 \equiv 1 + (2 \cdot 3)$
- kielessä voi olla lukuisia operaattoreita ryhmiteltyinä sitovuustasoille
  - sitoo voimakkaammin = korkeampi sitovuustaso
  - jos kaksi operaattoria kilpailee välissään olevasta kohteesta, se liitetään voimakkaammin sitovaan: $1 + 2 \cdot 3$
- esim. C++ [Stroustrup 1997]: 68 operaattoria, 18 sitovuustasoa

`. [] ++ -- ...`
`++ -- ! - + ...`

`* / %`
`+ -`
`<< >>`
`< <= > >=`

Unaari ja binääri

- **unaarioperaattorilla** on yksi argumentti
  - esim. etumerkki--, logiikan $\neg$, etuliite-++, jälkiliite-++
- **binäärioperaattorilla** on kaksi argumenttia
  - esim. vähennyslasku--, kertolasku $\cdot$
- **ternäärioperaattorilla** on kolme argumenttia
  - esim. C++ `?: x < 0 ? -x : x`

Etumerkki-− on eri operaattori kuin vähennyslasku-−, vaikka kirjoitusasu on sama

Sitovuuden suunta

- sitovuustasot eivät ratkaise tilannetta, jos kohteen molemmin puolin on sama operaattori
  - onko $8-5-2 \equiv 8-(5-2) = 8-3 = 5$
    vai $8-5-2 \equiv (8-5)-2 = 3-2 = 1$?

$$\underbrace{8-\overbrace{5}-2}$$

$\Rightarrow$ operaattorille ilmoitetaan sitovuuden suunta

- jos operaattori sitoo vasemmalle (on **left associative**), niin
  - kohde liittyy vasemmalla puolellaan olevaan
  - laskenta etenee vasemmalta oikealle
- jos operaattori sitoo oikealle (on **right associative**), niin päinvastoin
- yleensä operaattorit sitovat vasemmalle
- esimerkkejä oikealle sitovista
  - potenssilasku matematiikassa: $2^{1^3} \equiv 2^{(1^3)} = 2^1 = 2$ eikä $(2^1)^3 = 2^3 = 8$
  - sijoitusoperaattorit C++:ssa: `i = n = 0;`
- liitännäisyys (associativity) on eri asia
  - operaattori $\diamond$ on liitännäinen, jos aina $(x \diamond y) \diamond z = x \diamond (y \diamond z)$
  - liitännäisyys liittyy operaattorin merkitykseen eikä lausekepuun muodostamissääntöihin
  - jos operaattori on liitännäinen, niin sen sitovuuden suunnalla ei ole väliä

Matematiikan sitovuussäännöt eivät ole aina hyvin määritellyt

- esim. kaksinkertaisen kulman sini: $\sin 2x = 2 \sin x \cos x$
  - jos $\sin$ sitoo voimakkaammin, saadaan $(\sin 2)x = 2(\sin x) \cos x$
  - jos $\cdot$ sitoo voimakkaammin, saadaan $\sin(2x) = 2 \sin(x \cos x)$
  - tarkoitetaan $\sin(2x) = 2(\sin x) \cos x$

Tarpeettoman osan laskematta jättäminen (**short-circuit evaluation**)

- jos puhtaasti loogisen operaation ensin laskettu puoli saa sopivan arvon, ei toista puolta tarvitse laskeakaan
  - $\mathsf{false} \wedge p = \mathsf{false}$ ja $\mathsf{true} \vee p = \mathsf{true}$

  $\Rightarrow$ työmäärä vähenee
- usein ohjelmoinnissa oikea puoli on määritelty vain, jos vasen saa sopivan arvon
  - **if** $i \geq 1 \wedge i \leq n \wedge A[i] = 0$ **then** ...

    vrt. **if** $i \geq 1 \wedge i \leq n$ **then if** $A[i] = 0$ **then** ...
  - **while** $p \neq \bot \wedge p\uparrow.n > 0$ **do** ...

$\Rightarrow$ muunnetut loogiset operaattorit, joiden oikea puoli lasketaan vain jos tarpeen, ovat käteviä

- **tällä opintojaksolla niitä merkitään `&&` ja `||`**
- C++:n `&&` ja `||` ovat sellaiset
- Ada:ssa on erikseen `and` ja `and then` sekä `or` ja `or else`

Toisinaan pidetään ihanteena, että lauseke esittää
puhtaasti funktiota matemaattisessa mielessä

- ei saa olla **sivuvaikutuksia** (**side effect**)
  - sivuvaikutus = muu vaikutus kuin lausekkeen arvon tuottaminen
  - esim. `A[++i]` sivuvaikutuksenaan kasvattaa `i`:n arvoa
  - esim. `std::cout << i << ' ' << i*i` eli
    `((std::cout << i)<< ' ')<< i*i` palauttaa ns. viitteen
    tulostuskanavaan ja sivuvaikutuksenaan tulostaa
- myös tarpeettoman osan laskematta jättäminen rikkoo ihannetta
  - **while** $p \neq \bot \wedge p\uparrow.n > 0$ **do** ... kuuluu tuottaa virhe kun $p = \bot$, koska
    while $p\uparrow.n > 0 \wedge p \neq \bot$ **do** ... tarkoittaa samaa koska $\wedge$ on logiikan "ja"

$\Rightarrow$ ihanteen tinkimätön noudattaminen olisi epäkäytännöllistä

Siltä osin kuin puurakenne ei määrää laskujärjestystä, sillä ei saa olla väliä

- esim. $1 + 2 \cdot 3$: puurakenne määrää, että ensin lasketaan $2 \cdot 3$
- $(1 + 2) \cdot (3 + 4)$: puurakenne ei määrää, lasketaanko ensin $1 + 2$ vai $3 + 4$
- `i*A[i++]` **on vaarallinen**, koska ei tiedetä, kumpi sininen tapahtuu ensin

Saman osan käyttö monesti samassa lausekkeessa on kömpelöä

$$\left(\frac{1}{x^2+1}\right) \sin\left(\frac{1}{x^2+1}\right) + \left(1 - \frac{1}{x^2+1}\right)^3$$

3.4 Huomautuksia

0 Helsinki
108 Hämeenlinna
187 Tampere
347 Seinäjoki
680 Oulu
900 Rovaniemi

Sama tieto voi olla loogisesti organisoitu eri tavoin

- opiskelijoiden määrä, joista naisia $\leftrightarrow$ miesopiskelijoiden määrä ja naisopiskelijoiden määrä
- juna-aikataulun kilometrit pääteasemalta
  - osuuden Hämeenlinna–Oulu pituus saadaan yhdellä vähennyslaskulla
  - jos olisi annettu kunkin asemavälin pituus, tarvittaisiin monta yhteenlaskua

Tiedon sisäinen esittämistapa tietokoneessa voi olla kaukana siitä, mitä käyttäjä näkee

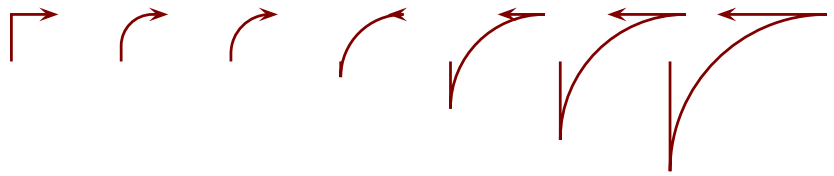
- esim. vektorigrafiikka
- (esim. HTML:n ja CSS:n yritys erottaa ulkoasu sisällöstä)

Ihmisillä on epäloogisia esitystapoja: 11:59 am, 12 noon, 12:01 pm, 12:59 pm, 1:00 pm

Tietokone tekee mitä käsketään, mutta se ei aina ole mitä halutaan

- paljolti kyse on ohjelmien ja ohjelmointikielten huonoista yksityiskohdista
  - esim. pieni kirjoitusvirhe muuttaa C++ loogisen ja && operaattoriksi &, joka käyttäytyy usein samoin mutta ei aina $\Rightarrow$ vaikeasti havaittava virhe
- samaa esiintyy luonnollisissakin kielissä
  - vauva juo virtsaa ja käy nukkumaan | vauva juo, virtsaa ja käy nukkumaan
  - panda eats, shoots and leaves vs. panda eats shoots and leaves

Toisinaan ohjelma on ok, mutta asia yllättää



- esim. kaari nuolessa kaaren säteen funktiona
- käskin tietokoneen etsiä Kotimaisten kielten keskuksen sanalistan sanoista pitkää sanajonoa, jonka voi jakaa kahdella tavalla yksittäisiksi sanoiksi

  šakki'ruutu'in'en.ää'ne'en.ää'ne'en.ää'ne'en.ää'ne'en.ää'ne'en.ää'ne'en.in

⇒ sama sana ei saa esiintyä useasti (huono pikakorjaus)

šakki'ruutu'in'en.ää'ne'en.empi'ä.äri'nä.es'imu'oto.s'ääli.ö'isi'n.ähkää.s'ääntö

⇒ in ja ähkää pois sanastosta

šinto'lain'en.ää'ne'en.empi'ä.äri'nä.es'imu'oto.s'ääli.ö'isi'n.äkki'ä.es'iva'ali

⇒ šinto ja lain pois sanastosta

äänenpitävä'sti.di'aari'o.as'e.cu'p.ää'ne'en.ää'nes.te'ak.alli.nen'ä.äri'nä.ämmä

⇒ vierasperäisten, vanhentuneiden jne. sanojen poisto olisi liian iso työ

Helppokäyttöisten luotettavien ohjelmien teko edellyttää sekä ihmisen sumean että tietokoneen pilkuntarkan ajattelutavan ymmärtämistä

- jos virhe on tarpeeksi hölmö, niin
  - ihminen korjaa sen automaattisesti, jopa tiedostamattaan
  - tietokone tekee jotain todella hölmöä
- esim. "kohta Suomessa on 100 000 alle 25-vuotiasta ilman tutkintoa"
- entä jos virhe on tarpeeksi hölmö kirjoittajan mutta ei lukijan näkökulmasta?

4 Propositiologiikka

Miksi tärkeä?

- propositiologiikka on kieli ja teoria
  - totuusarvojen funktioiden esittämiseen
  - totuusarvoista päättelemiseen
- yksinään se on aika tylsä ja keinotekoisen tuntuinen, mutta ...
- ... se on ilmaisuvoimaisempien logiikkojen perusta
- ... tietokoneiden peruspiirit ovat ymmärrettävissä sen avulla
- ... moni muukin aihealue on käännettävissä propositiologiikan kaavoiksi

$\Rightarrow$ sillä on ollut keskeinen merkitys sen tutkimisessa, mitä tehtäviä voidaan ja mitä ei voida ratkaista nopeasti tietokoneilla

- ja toki ohjelmoinnissa on hyödyksi osata loogista ajattelua ⌣̈

Kaikki tämän luvun asiat eivät ole itsessään tärkeitä, mutta ...

- ... on tärkeää touhuta propositiologiikan kanssa riittävästi, jotta siihen tulee tuntuma
- ... luvun päättelyt ovat hyödyllisiä esimerkkejä päättelemisestä

4.1 Kaksiarvoinen eli "tavallinen" propositiologiikka

Propositiologiikan kaava voidaan rakentaa seuraavista osista:

- totuusarvovakiot **F** eli *epätosi* (*false*) ja **T** eli *tosi* (*true*)
- totuusarvoiset muuttujat eli *propositiot* P, Q, P', P_1, P_i ...,
- *negaatio* $\neg$ eli "ei"
  - jos φ on kaava, niin $\neg\varphi$ tarkoittaa päinvastaista kuin φ
  - esim. reaaliluvuilla $\neg(x < y)$ tarkoittaa samaa kuin $x \geq y$
    ($x < y$ ja $x \geq y$ ovat predikaattilogiikan mutta eivät propositiologiikan kaavoja)
- *konjunktio* $\wedge$ eli "ja"
  - $\varphi \wedge \psi$ tarkoittaa, että sekä φ pätee että ψ pätee
  - esim. reaaliluvuilla $x \geq 0 \wedge x \neq 0$ tarkoittaa samaa kuin $x > 0$
- *disjunktio* $\vee$ eli "tai"
  - $\varphi \vee \psi$ tarkoittaa, että φ pätee tai ψ pätee tai molemmat pätevät
  - esim. reaaliluvuilla $x \geq 0 \vee x < 1$ tarkoittaa samaa kuin **T**
- *totuusarvojen implikaatio* (tai vain *implikaatio*) $\rightarrow$ eli "jos ... niin"
  - $\varphi \rightarrow \psi$ tarkoittaa, että φ ei päde tai ψ pätee
  - esim. reaaliluvuilla $x > 0 \rightarrow x \geq 0$ tarkoittaa samaa kuin **T**
- *totuusarvojen ekvivalenssi* (tai vain *ekvivalenssi*) $\leftrightarrow$ eli "jos ja vain jos"
  - $\varphi \leftrightarrow \psi$ tarkoittaa, että molemmat pätevat tai kumpikaan ei päde
  - esim. reaaliluvuilla $x > 0 \leftrightarrow x > 1$ tarkoittaa samaa kuin $x \leq 0 \vee x > 1$

- sulut (ja)
  - käytetään ohjaamaan laskujärjestystä kuten koulumatematiikassakin

Sitovuussäännöt

- kuten tavallista, sitovuussäännöillä vähennetään sulkujen tarvetta
  - esim. $(\neg P) \vee (Q \wedge R) \Leftrightarrow \neg P \vee Q \wedge R$
- propositiologiikan sitovuussäännöt ovat vain osittain vakiintuneet

$\Rightarrow$ alla olevat pätevät tällä kurssilla kokonaan ja muualla osittain

- voimakkaimmin sitoo $\neg$, sitten $\wedge$, sitten $\vee$, sitten $\rightarrow$ ja heikoimmin sitoo $\leftrightarrow$
- $\rightarrow$ sitoo oikealle ja muut binäärioperaattorit vasemmalle
  - muut ovat liitännäisiä, joten niiden suunnalla on vain vähän merkitystä

Propositiologiikan kielioppi (tällä kurssilla)

Kaava	::=	Ekvivalenssikaava
Ekvivalenssikaava	::=	Implikaatiokaava | Ekvivalenssikaava $\leftrightarrow$ Implikaatiokaava
Implikaatiokaava	::=	Disjunktiokaava | Disjunktiokaava $\rightarrow$ Implikaatiokaava
Disjunktiokaava	::=	Konjunktiokaava | Disjunktiokaava $\wedge$ Konjunktiokaava
Konjunktiokaava	::=	Negaatiokaava | Konjunktiokaava $\vee$ Negaatiokaava
Negaatiokaava	::=	Atomikaava | $\neg$ Negaatiokaava
Atomikaava	::=	**F** | **T** | Muuttuja | (Kaava)

- huomaa implikaation poikkeava sitovuuden suunta!

Operaattoreiden tulokset

$\neg$	
F	**T**
T	**F**

$\wedge$	**F**	**T**
F	**F**	**F**
T	**F**	**T**

$\vee$	**F**	**T**
F	**F**	**T**
T	**T**	**T**

$\rightarrow$	**F**	**T**
F	**T**	**T**
T	**F**	**T**

$\leftrightarrow$	**F**	**T**
F	**T**	**F**
T	**F**	**T**

- $P \diamond Q$ luetaan riviltä P sarakkeesta Q
- siis **F** $\rightarrow$ **T** $\Leftrightarrow$ **T** ja **T** $\rightarrow$ **F** $\Leftrightarrow$ **F**
- muut operaattorit kuin $\rightarrow$ ovat vaihdannaiset

Propositiologiikan kaava esittää *totuusfunktiota*

- merkitään totuusarvojen joukkoa $\mathbb{B} = \{\mathbf{F}, \mathbf{T}\}$
- totuusfunktio f on funktio n totuusarvolta totuusarvolle eli $f : \mathbb{B}^n \mapsto \mathbb{B}$
- n on funktion *ariteetti* (arity) eli argumenttien määrä
  - kullekin funktiolle kiinteä, mutta vaihtelee funktioiden välillä

Ariteetin 2 totuusfunktioita $f(P, Q)$ on kaikkiaan 16

P	Q	**F**
F	**F**	**F**
F	**T**	**F**
T	**F**	**F**
T	**T**	**F**

P	Q	$P \wedge Q$
F	**F**	**F**
F	**T**	**F**
T	**F**	**F**
T	**T**	**T**

P	Q	$P \wedge \neg Q$
F	**F**	**F**
F	**T**	**F**
T	**F**	**T**
T	**T**	**F**

P	Q	P
F	**F**	**F**
F	**T**	**F**
T	**F**	**T**
T	**T**	**T**

P	Q	$\neg P \wedge Q$
F	F	F
F	T	T
T	F	F
T	T	F

P	Q	Q
F	F	F
F	T	T
T	F	F
T	T	T

P	Q	$\neg(P \leftrightarrow Q)$
F	F	F
F	T	T
T	F	T
T	T	F

P	Q	$P \vee Q$
F	F	F
F	T	T
T	F	T
T	T	T

P	Q	$\neg(P \vee Q)$
F	F	T
F	T	F
T	F	F
T	T	F

P	Q	$P \leftrightarrow Q$
F	F	T
F	T	F
T	F	F
T	T	T

P	Q	$\neg Q$
F	F	T
F	T	F
T	F	T
T	T	F

P	Q	$P \vee \neg Q$
F	F	T
F	T	F
T	F	T
T	T	T

P	Q	$\neg P$
F	F	T
F	T	T
T	F	F
T	T	F

P	Q	$\neg P \vee Q$
F	F	T
F	T	T
T	F	F
T	T	T

P	Q	$\neg(P \wedge Q)$
F	F	T
F	T	T
T	F	T
T	T	F

P	Q	T
F	F	T
F	T	T
T	F	T
T	T	T

- tämä totuusfunktion esitystapa on *totuustaulu*
  - rivi jokaista muuttujien arvoyhdistelmää kohden
  - sarake lopputulosta ja jokaista syötemuuttujaa kohden

Yleisemmin

- ariteetin 0 totuusfunktioita on kaksi: **F** ja **T**
- ariteetin 1 totuusfunktioita on neljä: **F**, P, $\neg P$ ja **T**
- ariteetin n totuusfunktioita on 2^{2^n}
  - muuttujien arvoyhdistelmiä on 2^n
  - kullekin niistä funktio tuottaa yhden arvon 2 vaihtoehdosta: **F** tai **T**
- huom! x^{y^z} lasketaan $x^{(y^z)}$ eikä $(x^y)^z$
  $\Rightarrow$ ariteetin 3 totuusfunktioita on $2^{(2^3)} = 2^8 = 256$ eikä $(2^2)^3 = 4^3 = 64$

Jokaisella $n \in \mathbb{N}$, jokainen $f : \mathbb{B}^n \mapsto \mathbb{B}$ voidaan ilmaista
muuttujilla ja operaattoreilla **F**, **T**, $\neg$, $\wedge$ ja $\vee$

- (myöhemmin nähdään, että vielä pienempi joukko operaattoreita riittää)
- jos funktio tuottaa **F** kaikilla syötteillä, sen ilmaisee kaava **F**
- muutoin
  - tehdään $\wedge$-kaava "poimimaan" kukin syöte, jolla tulee **T**
  - yhdistetään ne $\vee$:lla
- esimerkki: **FFTFTFTT**
  - siis $f(\mathbf{F},\mathbf{F},\mathbf{F}) \Leftrightarrow \mathbf{F}$, $f(\mathbf{F},\mathbf{F},\mathbf{T}) \Leftrightarrow \mathbf{F}$, $f(\mathbf{F},\mathbf{T},\mathbf{F}) \Leftrightarrow \mathbf{T}$ jne.
  - esim. $\neg P_1 \wedge P_2 \wedge \neg P_3 \vee P_1 \wedge \neg P_2 \wedge \neg P_3 \vee P_1 \wedge P_2 \wedge \neg P_3 \vee P_1 \wedge P_2 \wedge P_3$
- usein tulee pitkiä kaavoja!
  - se on osittain, mutta vain osittain, tämän muodostustavan vika

- informaatioteoreettinen lainalaisuus:
  - totuusfunktioita on paljon
    (2^{2^n} $\Rightarrow$ totuusfunktion esittämiseksi tarvitaan keskimäärin ainakin 2^n bittiä)
  - lyhyitä kaavoja on vähän
    ("lyhyt" voi tarkoittaa vaikka $\leq cn + d$ joillekin vakioille c ja d)

  $\Rightarrow$ useimmille totuusfunktioille ei riitä lyhyttä kaavaa
  (2^n kasvaa paljon nopeammin kuin $cn + d$ tai vaikka $cn^3 + d$, kun n kasvaa)

Rekursiivinen tapa tuottaa totuusfunktiolle kaava

- tämän kaltainen (vaikka ei juuri tämä) konstruktio on laajassa käytössä
- ariteetti 0: **F** ja **T**
- kun $n > 0$,

$$f(P_1, \ldots, P_n) \Leftrightarrow \neg P_n \wedge f(P_1, \ldots, P_{n-1}, \mathbf{F}) \vee P_n \wedge f(P_1, \ldots, P_{n-1}, \mathbf{T})$$

  - $f(P_1, \ldots, P_{n-1}, \mathbf{F})$ ja $f(P_1, \ldots, P_{n-1}, \mathbf{T})$ ovat ariteettia $n - 1$, vaikka ovatkin määritelty f:n avulla
  - jos $n - 1 > 0$, niin $f(P_1, \ldots, P_{n-1}, \mathbf{F})$ ilmaistaan $f(P_1, \ldots, P_{n-2}, \mathbf{F}, \mathbf{F})$:n ja $f(P_1, \ldots, P_{n-2}, \mathbf{T}, \mathbf{F})$:n avulla jne.
- voidaan optimoida esim. käyttämällä kaavoja **F**, P_1, $\neg P_1$ ja **T** kun $n = 1$
  - yo. tuottaa
    $\neg P_1 \wedge \mathbf{F} \vee P_1 \wedge \mathbf{F}$, $\neg P_1 \wedge \mathbf{F} \vee P_1 \wedge \mathbf{T}$,
    $\neg P_1 \wedge \mathbf{T} \vee P_1 \wedge \mathbf{F}$ ja $\neg P_1 \wedge \mathbf{T} \vee P_1 \wedge \mathbf{T}$

- esimerkki on helpompi seurata, jos muuttujat käsitellään P_1 ensin:

  $\mathbf{FFTFTFTT} \Leftrightarrow \neg P_1 \wedge \mathbf{FFTF} \vee P_1 \wedge \mathbf{TFTT} \Leftrightarrow$

  $\neg P_1 \wedge (\neg P_2 \wedge \mathbf{FF} \vee P_2 \wedge \mathbf{TF}) \vee P_1 \wedge (\neg P_2 \wedge \mathbf{TF} \vee P_2 \wedge \mathbf{TT}) \Leftrightarrow$

  $\neg P_1 \wedge (\neg P_2 \wedge \mathbf{F} \vee P_2 \wedge \neg P_3) \vee P_1 \wedge (\neg P_2 \wedge \neg P_3 \vee P_2 \wedge \mathbf{T}) \Leftrightarrow$ sievennys

  $\neg P_1 \wedge P_2 \wedge \neg P_3 \vee P_1 \wedge (\neg P_2 \wedge \neg P_3 \vee P_2)$

- tuli eri kaava kuin viimeksi $\Rightarrow$ tarkastamme totuustaululla

P_1	P_2	P_3	$\neg P_1 \wedge P_2 \wedge \neg P_3$	$P_1 \wedge (\neg P_2 \wedge \neg P_3 \vee P_2)$	koko kaava
F	F	F	F	F	F
F	F	T	F	F	F
F	T	F	T	F	T
F	T	T	F	F	F
T	F	F	F	T	T
T	F	T	F	F	F
T	T	F	F	T	T
T	T	T	F	T	T

- oikein meni, mutta löytyisikö helpompi laskentakeino kuin totuustaulut?

Propositiologiikan lakeja

- harmaat lakkaavat pätemästä luvussa 4.4
- totuusarvoisten vakioiden eliminointi

$$\neg\mathbf{F} \Leftrightarrow \mathbf{T} \qquad \neg\mathbf{T} \Leftrightarrow \mathbf{F}$$
$$\mathbf{F} \wedge P \Leftrightarrow P \wedge \mathbf{F} \Leftrightarrow \mathbf{F} \qquad \mathbf{T} \wedge P \Leftrightarrow P \wedge \mathbf{T} \Leftrightarrow P$$
$$\mathbf{F} \vee P \Leftrightarrow P \vee \mathbf{F} \Leftrightarrow P \qquad \mathbf{T} \vee P \Leftrightarrow P \vee \mathbf{T} \Leftrightarrow \mathbf{T}$$
$$\mathbf{F} \leftrightarrow P \Leftrightarrow P \leftrightarrow \mathbf{F} \Leftrightarrow \neg P \qquad \mathbf{T} \leftrightarrow P \Leftrightarrow P \leftrightarrow \mathbf{T} \Leftrightarrow P$$
$$\mathbf{F} \rightarrow P \Leftrightarrow \mathbf{T} \qquad P \rightarrow \mathbf{F} \Leftrightarrow \neg P \qquad \mathbf{T} \rightarrow P \Leftrightarrow P \qquad P \rightarrow \mathbf{T} \Leftrightarrow \mathbf{T}$$

- toiston eliminointi

$$\neg\neg P \Leftrightarrow P \qquad P \wedge P \Leftrightarrow P \qquad P \vee P \Leftrightarrow P \qquad P \rightarrow P \Leftrightarrow \mathbf{T} \qquad P \leftrightarrow P \Leftrightarrow \mathbf{T}$$

- vaihdannaisuus

$$P \wedge Q \Leftrightarrow Q \wedge P \qquad P \vee Q \Leftrightarrow Q \vee P \qquad P \leftrightarrow Q \Leftrightarrow Q \leftrightarrow P$$

- implikaatiolle ei päde vaihdannaisuus, vaan

$$P \rightarrow Q \Leftrightarrow \neg Q \rightarrow \neg P$$

- liitännäisyys (ei päde implikaatiolle)

$$(P \wedge Q) \wedge R \Leftrightarrow P \wedge (Q \wedge R) \Leftrightarrow P \wedge Q \wedge R$$
$$(P \vee Q) \vee R \Leftrightarrow P \vee (Q \vee R) \Leftrightarrow P \vee Q \vee R$$
$$(P \leftrightarrow Q) \leftrightarrow R \Leftrightarrow P \leftrightarrow (Q \leftrightarrow R) \Leftrightarrow P \leftrightarrow Q \leftrightarrow R$$

- osittelulait

$$P \wedge (Q \vee R) \Leftrightarrow P \wedge Q \vee P \wedge R \qquad P \vee Q \wedge R \Leftrightarrow (P \vee Q) \wedge (P \vee R)$$

- ekvivalenssin ja implikaation eliminointi

$$P \leftrightarrow Q \Leftrightarrow (P \rightarrow Q) \wedge (Q \rightarrow P) \qquad P \rightarrow Q \Leftrightarrow \neg P \vee Q$$

- de Morganin lait

$$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q \qquad \neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$$

- muita

$$P \wedge (P \vee Q) \Leftrightarrow P \qquad P \vee P \wedge Q \Leftrightarrow P$$
$$P \wedge (\neg P \vee Q) \Leftrightarrow P \wedge Q \qquad P \vee \neg P \wedge Q \Leftrightarrow P \vee Q$$
$$P \wedge \neg P \Leftrightarrow \mathbf{F} \qquad P \vee \neg P \Leftrightarrow \mathbf{T}$$
$$P \rightarrow (Q \rightarrow R) \Leftrightarrow P \wedge Q \rightarrow R$$

Ei kannata opetella kaikkia ulkoa!

- usea on niin ilmeinen, ettei siinä ole mitään opettelemista
  - esim. totuusarvoisten vakioiden eliminointilait $\neg$:lle, $\wedge$:lle ja $\vee$:lle
  - esim. vaihdannaisuuslait
- moni jää itsestään mieleen kun sitä käyttää
- vaihdannaisuus ja liitännäisyys sallivat sulkujen poiston ja uudelleen järjestelyn
  - esim. $R \vee ((P \wedge S \vee \neg R) \vee Q) \Leftrightarrow Q \vee R \vee \neg R \vee P \wedge S$
  - $\wedge$, $\vee$ ja $\leftrightarrow$, mutta ei $\rightarrow$

- osittelulait ja de Morganin lait kannattaa opetella
  - hyödyllisiä, mutta vaikea hoksata itse
- ekvivalenssista ja implikaatiosta riittää muistaa eliminointi
  - sillä pääsee niistä eroon
  - sillä on yhteys päättelemisen lainalaisuuksiin
  - jos jaksat muistaa enemmän, niin muista $\leftrightarrow$:n liitännäisyys

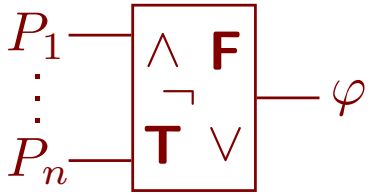
Usein kaavan tai lain voi tarkastaa näppärästi seuraavasti

- valitse jokin muuttuja
- sijoita sen arvoksi **F**, sievennä, ja tarkasta tulos
- sijoita sen arvoksi **T**, sievennä, ja tarkasta tulos
- jos tuloksen tarkastaminen on vaikeaa, valitse toinenkin muuttuja
- esim. $P \vee Q \wedge R \Leftrightarrow (P \vee Q) \wedge (P \vee R)$
  - sijoitetaan $P \Leftrightarrow \mathbf{F}$
  - vasen puoli tuottaa $\mathbf{F} \vee Q \wedge R \Leftrightarrow Q \wedge R$
  - oikea puoli tuottaa $(\mathbf{F} \vee Q) \wedge (\mathbf{F} \vee R) \Leftrightarrow Q \wedge R$
  - sijoitetaan $P \Leftrightarrow \mathbf{T}$
  - vasen puoli tuottaa $\mathbf{T} \vee Q \wedge R \Leftrightarrow \mathbf{T}$
  - oikea puoli tuottaa $(\mathbf{T} \vee Q) \wedge (\mathbf{T} \vee R) \Leftrightarrow \mathbf{T} \wedge \mathbf{T} \Leftrightarrow \mathbf{T}$

Implikaatio on valittu sitomaan oikealle, koska

- silloin $P_1 \to \cdots \to P_{n-1} \to P_n \;\Leftrightarrow\; P_1 \wedge \cdots \wedge P_{n-1} \to P_n$
  - vrt. Hornin klausuuli sivu 65
- päinvastainen valinta ei johda yhtä hyödylliseen kaavaan

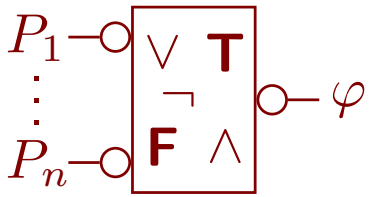
Kuinka monta operaattoria tarvitaan esittämään kaikki totuusfunktiot?



- de Morganin laista seuraa $P \vee Q \Leftrightarrow \neg(\neg P \wedge \neg Q)$
- lisäksi tiedetään $\mathbf{T} \Leftrightarrow \neg\mathbf{F}$

$\Rightarrow$ jokainen totuusfunktio voidaan esittää muuttujilla ja operaattoreilla $\mathbf{F}$, $\neg$ ja $\wedge$

- jos muuttujia on ainakin yksi, niin $\mathbf{F}$ on tarpeeton, koska $\mathbf{F} \Leftrightarrow P \wedge \neg P$
- operaattorien $\neg$ ja $\wedge$ sijaan riittää $\mathrm{NAND}(P, Q) \Leftrightarrow \neg(P \wedge Q)$
  - $\neg P \Leftrightarrow \mathrm{NAND}(P, P)$
  - $P \wedge Q \Leftrightarrow \mathrm{NAND}(\ \mathrm{NAND}(P, Q), \mathrm{NAND}(P, Q)\ )$



$\mathbf{F}$:n ja $\mathbf{T}$:n symmetria

- kuten edellä nähtiin, $\to$ ja $\leftrightarrow$ voidaan eliminoida kaavoista
- de Morganin laeista seuraa, että jos kaava φ ei sisällä $\to$ eikä $\leftrightarrow$, niin $\neg\varphi$ saadaan vaihtamalla $\mathbf{F} \rightleftharpoons \mathbf{T}$, $\wedge \rightleftharpoons \vee$ ja $P \rightleftharpoons \neg P$ jokaiselle muuttujalle P

$\Rightarrow$ $\mathbf{F}$:n ja $\mathbf{T}$:n välillä vallitsee varsin vahva symmetria

- se menee rikki luvussa 4.4

4.2 CNF, DNF, automatisoitu päättely ja P $\stackrel{?}{=}$ NP -ongelma

Tässä alaluvussa

- esitetään kaksi säännöllistä muotoa, joihin jokaisen kaavan voi muuntaa
  - muodoista on helppo nähdä tietyt asiat
  - hyödyllisiä automaattisessa päättelyssä
- pohditaan hieman propositiologiikan tehtävien laskennallista vaativuutta
- vilkaistaan propositiologiikan automatisoitua päättelyä

Sanastoa

- kaava φ on *tautologia*, jos ja vain jos $\varphi \Leftrightarrow \mathbf{T}$
- kaava φ on *ristiriita*, jos ja vain jos $\varphi \Leftrightarrow \mathbf{F}$
- kaava φ on *toteutettavissa*, jos ja vain jos se ei ole ristiriita
- *literaali* on propositiologiikan muuttuja tai sen negaatio
  - esim. P, $\neg Q$, P_1, $\neg P_i$
- *klausuuli* on kaava, joka koostuu vain literaaleista ja $\vee$:sta
  - esim. Q, $P \vee \neg Q \vee \neg P$, $\neg P_1 \vee P_2 \vee P_4 \vee \neg P_5 \vee P_8$ tai $P \vee P$
- *iluusualk* on kaava, joka koostuu vain literaaleista ja $\wedge$:sta
  - esim. Q, $P \wedge \neg Q \wedge \neg P$, $\neg P_1 \wedge P_2 \wedge P_4 \wedge \neg P_5 \wedge P_8$ tai $P \wedge P$
  - tälle piti keksiä oma nimi, koska kirjallisuudesta löytyneet aiheuttivat sekaannusta (keksittiinkin nimi, jota ei helposti erehdy luulemaan viralliseksi)

- *konjunktiivinen normaalimuoto* eli *CNF* (conjunctive normal form)
  on joko **F**, **T**, tai klausuuleista $\wedge$:lla muodostettu kaava
  - esim. **T**, $(P \vee \neg Q) \wedge \neg P \wedge (\neg P \vee R \vee \neg S)$, $\neg Q \wedge R$, $\neg Q \vee R$ tai Q
  - esim. seuraavat eivät ole CNF: $\neg(P \wedge Q)$, $\neg\neg P$, $P \vee Q \wedge R$ ja $P \to Q$
- *disjunktiivinen normaalimuoto* eli *DNF* (disjunctive normal form)
  on joko **F**, **T**, tai iluusualkeista $\vee$:lla muodostettu kaava
  - esim. **T**, $P \wedge \neg Q \vee \neg P \vee \neg P \wedge R \wedge \neg S$, $\neg Q \vee R$, $\neg Q \wedge R$ tai Q
  - esim. seuraavat eivät ole DNF: $\neg(P \vee Q)$, $\neg\neg P$, $P \wedge (Q \vee R)$ ja $P \to Q$
- Hornin klausuuli on klausuuli, jossa enintään yksi muuttuja on ilman $\neg$
  - esim. $\neg P \vee \neg Q \vee R \vee \neg S$
  - loogisesti yhtäpitävä kaava: $P \wedge Q \wedge S \to R$
  - logiikkaohjelmointikieli Prolog perustuu Hornin klausuuleihin

Kielioppina

$$
\begin{array}{rcl}
\text{Literaali} & ::= & \text{Muuttuja} \mid \neg\ \text{Muuttuja} \\
\text{Klausuuli} & ::= & \text{Literaali} \mid \text{Klausuuli} \vee \text{Literaali} \\
\text{Iluusualk} & ::= & \text{Literaali} \mid \text{Iluusualk} \wedge \text{Literaali} \\
\text{CNF} & ::= & \mathbf{F} \mid \mathbf{T} \mid \text{Klausuuli} \mid \text{CNF} \wedge \text{Klausuuli} \\
\text{DNF} & ::= & \mathbf{F} \mid \mathbf{T} \mid \text{Iluusualk} \mid \text{DNF} \vee \text{Iluusualk}
\end{array}
$$

Kaavan muunto CNF:ksi, joka esittää samaa totuusfunktiota

1. sovelletaan vakioiden eliminointilakeja niin monta kertaa kuin mahdollista
   - jokainen soveltaminen lyhentää kaavaa $\Rightarrow$ prosessi ei voi jatkua loputtomiin
   - kun se päättyy, kaava joko ei sisällä vakioita tai on **F** tai **T**
   - jos kaava on **F** tai **T**, niin lopetetaan
2. sovelletaan ekvivalenssin eliminointilakia niin monta kertaa kuin mahdollista
   - soveltaminen ei tuota vakioita
   - jokainen vähentää $\leftrightarrow$:ien määrää $\Rightarrow$ prosessi ei voi jatkua loputtomiin
   - kun se päättyy, kaavassa ei ole vakioita eikä $\leftrightarrow$
   - soveltaminen voi pidentää kaavaa paljonkin, koska osakaavoista tehdään kopioita
3. sovelletaan implikaation eliminointilakia niin monta kertaa kuin mahdollista
   - soveltaminen ei tuota vakioita eikä $\leftrightarrow$
   - ... $\Rightarrow$ prosessi ei voi jatkua loputtomiin
   - kun se päättyy, kaavassa ei ole vakioita, $\rightarrow$ eikä $\leftrightarrow$
4. sovelletaan de Morganin lakeja ja $\neg\neg$:n eliminointilakia niin monta kertaa kuin mahdollista
   - soveltaminen ei tuota ...
   - ... $\Rightarrow$ prosessi ei voi jatkua loputtomiin
   - kun se päättyy, kaavassa ei ole vakioita, $\rightarrow$ eikä $\leftrightarrow$, ja kaikki negaatiot kohdistuvat muuttujiin
   - de Morganin lain soveltaminen voi pidentää kaavaa, mutta vain vähän
   - $\neg\neg$:n poisto lyhentää kaavaa

5. sovelletaan osittelulakia $P \vee Q \wedge R \Leftrightarrow (P \vee Q) \wedge (P \vee R)$
   niin monta kertaa kuin mahdollista
   - soveltaminen ei tuota ...
   - ... $\Rightarrow$ prosessi ei voi jatkua loputtomiin
   - kun se päättyy, kaava on CNF
   - soveltaminen voi pidentää kaavaa paljonkin, koska osakaavoista tehdään kopioita

- lopputulosta voi optimoida mm.
  - poistamalla klausuulit, joissa sama muuttuja esiintyy sellaisenaan ja negatoituna (jos kaikki klausuulit poistuvat, kaava $\Leftrightarrow$ **T**)
  - poistamalla saman literaalin moninkertaiset esiintymät samassa klausuulissa
  - poistamalla saman klausuulin moninkertaiset esiintymät kaavassa

Kaavan muunto DNF:ksi, joka esittää samaa totuusfunktiota

- muuten sama kuin muunto CNF:ksi, mutta vaiheessa 5
  käytetään $P \wedge (Q \vee R) \Leftrightarrow P \wedge Q \vee P \wedge R$
- optimointikin on samankaltainen, mutta kaikkien iluusualkien poistuminen tuottaa **F**

Usein propositiologiikkaa halutaan käyttää käsin tai automaattisesti

- todistamaan, että jokin kaava φ on aina totta eli tautologia
  - sama tavoite toisin sanottuna: että $\neg\varphi$ on ristiriita
  - esim. tietokoneohjelma toimii aina oikein, matemaattinen väite pätee

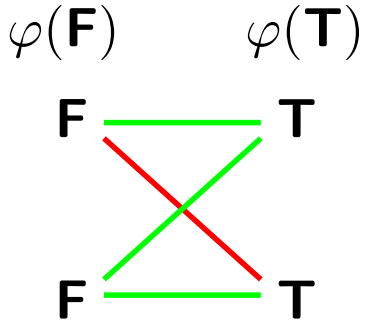
- löytämään muuttujille arvoyhdistelmä, jolla φ toteutuu
  - sama tavoite toisin sanottuna: löytämään arvoyhdistelmä, jolla $\neg\varphi$ tuottaa **F**
  - esim. syöte jolla ohjelma toimii väärin, optimointitehtävän (tai sudokun) ratkaisu
- jälkimmäinen tehtävä voi onnistua jos ja vain jos $\neg\varphi$ ei ole tautologia

Normaalimuotojen ominaisuuksia

- totuustaulusta saa helposti kirjoitettua DNF:n, joka esittää samaa totuusfunktiota
- optimoidusta CNF:stä näkee helposti, onko kaava tautologia
  - kaava **T** on tietenkin tautologia
  - muunlainen optimoitu CNF ei voi olla tautologia
    (tässä on tärkeää, että klausuulit joissa on sekä P että $\neg P$ on poistettu)
- jälkimmäisessä tapauksessa $\neg\varphi$:n toteuttava arvoyhdistelmä löytyy
  - valitsemalla mikä tahansa klausuuli
  - sijoittamalla sen muotoa P olevien literaalien muuttujiin **F**, sen muotoa $\neg P$ olevien literaalien muuttujiin **T**, ja muihin muuttujiin mitä huvittaa
  - esim. $(P \vee \neg R) \wedge (\neg P \vee \neg Q \vee S)$: **F** $\rightsquigarrow P$, **T** $\rightsquigarrow R$, **F** $\rightsquigarrow Q$ ja **F** $\rightsquigarrow S$
- optimoidusta DNF:stä näkee helposti, onko kaava ristiriita
  - kaava **F** on tietenkin ristiriita
  - muunlainen optimoitu DNF ei voi olla ristiriita
    (tässä on tärkeää, että iluusualkit joissa on sekä P että $\neg P$ on poistettu)
- jälkimmäisessä tapauksessa φ:n toteuttava arvoyhdistelmä löytyy ...
  - esim. $P \wedge \neg R \vee \neg P \wedge \neg Q \wedge S$: **T** $\rightsquigarrow P$, **F** $\rightsquigarrow R$, **F** $\rightsquigarrow Q$ ja **F** $\rightsquigarrow S$

Saadaanko siis mikä tahansa propositiologiikaksi käännetty tehtävä ratkaistua näppärästi muuntamalla kaava CNF:ksi tai DNF:ksi?

- ei, koska kaava voi paisua muunnoksessa liian suureksi



Voiko kaavan kahdentumisen $\leftrightarrow$:n eliminoinnissa estää?

- $\varphi(P)$ on *kasvava* P:n suhteen, jos ja vain jos jokaisella sen muiden muuttujien kuin P arvoyhdistelmällä pätee $\neg\varphi(\mathbf{F}) \vee \varphi(\mathbf{T})$
- $\varphi(P)$ on *vähenevä* P:n suhteen, jos ja vain jos jokaisella sen muiden muuttujien kuin P arvoyhdistelmällä pätee $\varphi(\mathbf{F}) \vee \neg\varphi(\mathbf{T})$
- jos φ ja ψ ovat kasvavia, niin
  - $\varphi \wedge \psi$ ja $\varphi \vee \psi$ ovat kasvavia
  - $\neg\varphi$ on vähenevä
  - (miten $\rightarrow$ käyttäytyy?)
- jos φ ja ψ ovat väheneviä, niin
  - $\varphi \wedge \psi$ ja $\varphi \vee \psi$ ovat väheneviä
  - $\neg\varphi$ on kasvava
  - (miten $\rightarrow$ käyttäytyy?)

$\Rightarrow$ jos φ:ssä ei esiinny $\leftrightarrow$, ja P esiintyy φ:ssä enintään kerran, niin φ on kasvava tai vähenevä P:n suhteen

- $P \leftrightarrow Q$ ei ole kasvava eikä vähenevä P:n suhteen
  - kun $Q \Leftrightarrow \mathbf{F}$, niin $P \leftrightarrow Q$ ei ole kasvava
  - kun $Q \Leftrightarrow \mathbf{T}$, niin $P \leftrightarrow Q$ ei ole vähenevä

$\Rightarrow$ ei voida rakentaa $P \leftrightarrow Q$ siten, että P esiintyy vain kerran

- voidaan siten, että P esiintyy kahdesti:
  $P \leftrightarrow Q \;\Leftrightarrow\; (P \rightarrow Q) \wedge (Q \rightarrow P) \;\Leftrightarrow\; (\neg P \vee Q) \wedge (\neg Q \vee P) \;\Leftrightarrow\; \neg P \wedge \neg Q \vee P \wedge Q$

Ehkä jokin parempi muunnos tuottaa aina lyhyestä syötteestä lyhyen lopputuloksen?

- tarkastellaan DNF:ää $P_1 \wedge Q_1 \vee \cdots \vee P_n \wedge Q_n$
- se tuottaa **T** jos ja vain jos $\exists i; 1 \leq i \leq n : P_i \wedge Q_i$ eli jollain i pätee $P_i \wedge Q_i$
- esitetään sama totuusfunktio optimoituna CNF:nä
- jokaisessa klausuulissa jokaisella $1 \leq i \leq n$ pitää olla literaali P_i tai Q_i
  - muutoin klausuuli tuottaa **F** kun $P_i \Leftrightarrow Q_i \Leftrightarrow$ **T**
    ja muiden muuttujien arvot on valittu sopivasti
  - silloin DNF tuottaa **T**
- jokaista symbolien $X_1, \ldots, X_n$, missä kukin X_i on joko P_i tai Q_i,
  yhdistelmää varten pitää olla klausuuli joka sisältää $X_1 \vee \cdots \vee X_n$
  - oletetaan, että jokin $X_1 \vee \cdots \vee X_n$ puuttuu
  - olkoon $Y_i = P_i$ jos $X_i = Q_i$ ja $Y_i = Q_i$ jos $X_i = P_i$

  $\Rightarrow$ jokaisessa klausuulissa on ainakin yksi Y_i
  - valitsemalla $X_1 \Leftrightarrow \cdots \Leftrightarrow X_n \Leftrightarrow$ **F** ja $Y_1 \Leftrightarrow \cdots \Leftrightarrow Y_n \Leftrightarrow$ **T**
    kaava tuottaa **T** vaikka DNF tuottaa **F**

$\Rightarrow$ lyhin saman totuusfunktion esittävä CNF sisältää ainakin 2^n
klausuulia, joista jokainen sisältää ainakin n muuttujaa

$\Rightarrow$ DNF, jossa on $4n - 1$ symbolia, vaatii CNF:n, jossa on ainakin $n2^{n+1} - 1$ symbolia

- on olemassa lyhyitä DNF:iä, joilla on vain eksponentiaalisesti pitempiä CNF:iä
- eksponentiaalinen kasvu voi tehdä työmäärästä valtavan jo melko pienellä syötteellä

Mitä tämä merkitsee päättelyn automatisoinnille ja, kun ihminen päättelee, vaikeudelle?

- tämä ei todista, että päättely vaatii joskus paljon tilaa
  - onko kaava tautologia voidaan selvittää myös kokeilemalla kaikilla muuttujien arvoyhdistelmillä
  - ei vie paljon tilaa tietokoneessa (eikä liitutaululla, jota pyyhitään usein)
  - vie paljon aikaa
- tämä ei todista, että päättely vaatii joskus paljon aikaa
  - kenties on olemassa jokin parempi päättelykeino?
- tämä todistaa vain, että muuntaminen CNF:ksi (tai DNF:ksi) on joskus hyvin tilaa vievää (ja siksi hyvin tehotonta)
- vuosikymmenten tutkimus ei ole ratkaissut, onko olemassa kaikilla syötteillä tehokasta tapaa selvittää, onko propositiologiikan kaava tautologia
  - tämä (täsmällisemmin muotoiltuna) on kuuluisa $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ -ongelma
- on kuitenkin paljon tuloksia, jotka viittaavat siihen, että vastaus on "ei"

P, **NP** ja **NP**-täydellisyys

- *päätöstehtävä* on kyllä/ei-kysymys
  - on sovittu jokin luonteva tapa esittää syöte tietokoneelle
  - tietokoneen pitää vastata "kyllä" tai "ei"
  - merkitsemme n:llä syötteen pituutta yksittäisinä merkkeinä laskettuna

- *polynomiaalinen aika* tarkoittaa, että on olemassa $k \in \mathbb{N}$ siten, että ohjelman ajan kulutus on $O(n^k)$ eli joillekin vakioille c ja d enintään $cn^k + d$
- **P** on niiden päätöstehtävien joukko, joille on olemassa polynomiaalisessa ajassa toimiva ratkaisualgoritmi
- **NP** on niiden päätöstehtävien joukko, joille on olemassa polynomiaalisessa ajassa toimiva *tarkastusalgoritmi*
  - tarkastusalgoritmi lukee syötteen lisäksi merkkijonon α
  - jokaiselle "kyllä"-syötteelle on ja millekään "ei"-syötteelle ei ole olemassa ainakin yksi *todiste*, joka on merkkijono
  - jos α on todiste ko. syötteelle, tarkastusalgoritmi vastaa "kyllä"
  - jos α ei ole todiste ko. syötteelle, tarkastusalgoritmi vastaa "ei kelpaa"

Mitä tapahtuu, jos kokeillaan joukolla satunnaisesti valittuja merkkijonoja?

- jos niiden joukossa on todiste
  - saadaan ainakin kerran vastaus "kyllä"

  $\Rightarrow$ tiedetään varmasti, että oikea vastaus on "kyllä"
- jos niiden joukossa ei ole todistetta
  - saadaan joka kerta vastaus "ei kelpaa"

  $\Rightarrow$ ei tiedetä, onko oikea vastaus "ei" vai kävikö niin, että todiste on olemassa mutta se ei sattunut kokeiltuihin merkkijonoihin

Esimerkkejä tehtävistä joukosta **NP**

- tarpeeksi lyhyen reitin kuvaus on todiste tehtävälle "onko olemassa kaikkien Suomen kaupunkien kautta kulkeva ajoreitti, jonka pituus on enintään 10 000 km?"
- toteuttava muuttujien arvoyhdistelmä on todiste tehtävälle "onko tämä propositiologiikan kaava toteutettavissa?"
  - tämä tehtävä tunnetaan lyhenteellä SAT
- sudokun ratkaisu on todiste tehtävälle "onko tälle sudokulle olemassa ratkaisu?"

Määritelmistä seuraa helposti, että $\mathbf{P} \subseteq \mathbf{NP}$

- tyhjä merkkijono kelpaa todisteeksi ja ratkaisualgoritmi kelpaa tarkastusalgoritmiksi, kun vastaus "ei" korvataan vastauksella "ei kelpaa"

Ei tiedetä, onko $\mathbf{NP} \subseteq \mathbf{P}$

Tunnetaan paljon "**NP**-täydellisiä" päätöstehtäviä

- ohitamme määritelmän
- jokainen niistä kuuluu joukkoon **NP**
- jos yksikin niistä kuuluu joukkoon **P**, niin $\mathbf{NP} \subseteq \mathbf{P}$ (ja niin ollen $\mathbf{P} = \mathbf{NP}$)

Esimerkkejä **NP**-täydellisistä ja muista päätöstehtävistä

- edellä mainitut kolme esimerkkiä joukkoon **NP** kuuluvista päätöstehtävistä
- SUBSET-SUM: on annettu luonnolliset luvut t ja $k_1, \ldots, k_m$; voidaanko jälkimmäisistä valita osa siten, että niiden summa on t?
- CNF-SAT: onko tämä CNF toteutettavissa?
- 3-CNF-SAT: onko tämä CNF, jossa jokaisessa klausuulissa on tasan 3 literaalia, toteutettavissa?
  - 2-CNF-SAT $\in$ **P**
- "etsi mahdollisimman lyhyt kaikkien Suomen kaupunkien ..." ei ole **NP**-täydellinen
  - se ei ole päätöstehtävä $\Rightarrow$ se ei ole **NP**:ssä
- "onko tämä kaava ristiriita" ei todennäköisesti ole edes **NP**:ssä
  - se on muutoin sama tehtävä kuin SAT, mutta vastaukset "kyllä" ja "ei" ovat vaihtaneet paikkaa
  - toteuttava arvoyhdistelmä on todiste SAT:ille
  - usein on vaikea keksiä todiste tilanteelle "ei ole toteutettavissa"

$\Rightarrow$ näyttää siltä, että **NP** ei ole symmetrinen "kyllä":n ja "ei":n suhteen (toisin kuin **P**)

Ensimmäiset **NP**-täydellisyystodistukset koskivat tehtäviä SAT ja 3-CNF-SAT

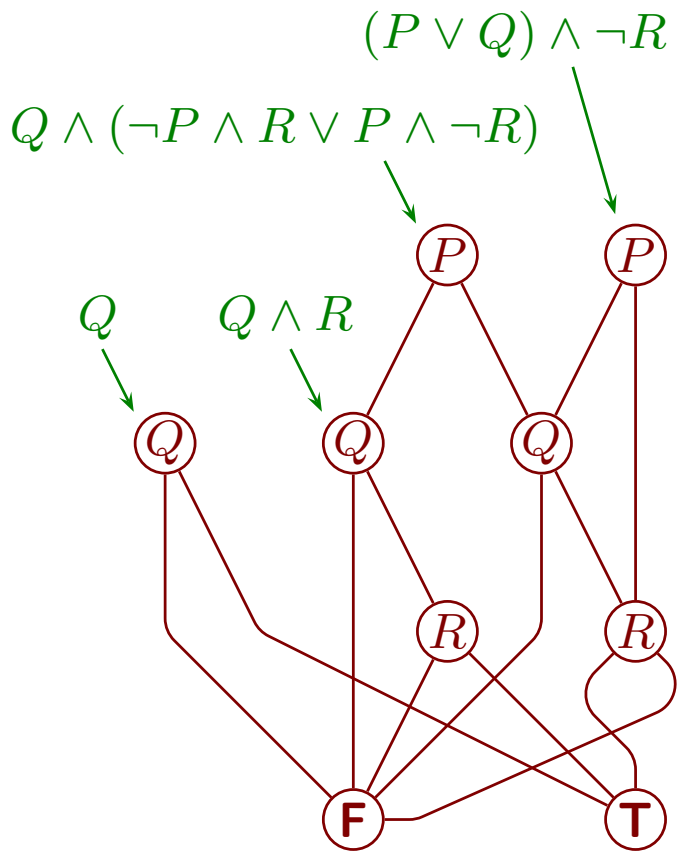
- Stephen Cook 1971
  - todistus perustuu tietokoneen (Turingin koneen) laskennan esittämiseen propositiologiikan kaavana
- samoihin aikoihin Leonid Levin keksi **NP**-täydellisyyden Neuvostoliitossa
  - julkaisu venäjäksi 1973, mutta ajatus oli esillä muutaman vuoden aikaisemmin
  - toisenlainen ongelma

Muita syitä uskoa, että $\mathbf{P} \neq \mathbf{NP}$

- moni päätöstehtävä on todistettu vaativammaksi kuin mikään **NP**:ssä
- tiedetään, että on olemassa päätöstehtäviä, jotka vaativat eksponentiaalisen ajan mutta ei enempää
- tunnetaan päätöstehtäviä, jotka ovat **NP**:ssä mutta näyttävät olevan olematta **P**:ssä tai **NP**-täydellisiä

Reduced ordered binary decision diagram eli ROBDD

- tietorakenne totuusfunktioiden esittämiseen
- laajassa käytössä erityisesti mikropiiriteollisuudessa
- suunnattu silmukaton graafi
  - kuvassa suunta on alas (ei nuolenkärkiä)
- vakiofunktioita **F** ja **T** varten on solmut
- jokaisessa muussa solmussa on
  - yksi muuttuja
  - kaksi kaarta eteenpäin: **F**-kaari ja **T**-kaari
- muuttujille on valittu järjestys (ordered)
  - solmun jälkeläisten muuttujat ovat järjestyksessä myöhemmin kuin solmun oma muuttuja
  - muuttujia saa jäädä välistä



- totuusfunktiota esittää osoitin BDD:n solmuun
  $\Rightarrow$ sama BDD voi esittää useita totuusfunktioita
  - totuusfunktion arvo saadaan kulkemalla muuttujien arvojen määräämä polku
- samanlaiset aligraafit on yhdistetty (reduced)
  - tehokasta ja melko helppoa hajautustaululla ja rekursiivisella algorimilla
  - onko $f(P_1, \ldots, P_n) \Leftrightarrow g(P_1, \ldots, P_n)$ selviää testillä `if( f_os == g_os )`
- jokaiselle perusoperaatiolle on tehokas algoritmi, mutta BDD:n koko voi kasvaa nopeasti

4.3 Päättelyoperaattorit

Olemme edellä käyttäneet $\Rightarrow$ ja $\Leftrightarrow$ ilman määritelmää

- niitä käytetään ilmaisemaan *päättelyitä*
- *looginen implikaatio* (tai vain *implikaatio*) $\varphi \Rightarrow \psi$ tarkoittaa, että kaikissa huomioon otettavissa tilanteissa joissa φ pätee, myös ψ pätee
  - propositiologiikassa tilanne = muuttujien arvoyhdistelmä
  - predikaattilogiikassa tilanteen käsite on monimutkaisempi, sivu 175
  - tilanne, jossa kaava pätee, on kaavan *malli*
    (sanaa "malli" käytetään siis jokseenkin päinvastoin kuin esim. tekniikassa)
- *looginen ekvivalenssi* (tai vain *ekvivalenssi*) $\varphi \Leftrightarrow \psi$ tarkoittaa, että kaikissa huomioon otettavissa tilanteissa joissa φ pätee, myös ψ pätee ja toisinpäin
- usein vain osa käsitteellisesti mahdollisista tilanteista otetaan huomioon
  - esim. kun päättely on jaettu tapauksiin $x < 0$ ja $x \geq 0$, ja käsitellään $x < 0$
  - esim. kun ristiriidan johtamiseksi oletetaan $x < 0$
  - esim. jos P tarkoittaa $x > 1$ ja Q tarkoittaa $x > 0$, niin $P \wedge \neg Q$ ei oteta huomioon
- merkitsemme tällä kurssilla Γ:lla kaikkea,
  mikä vähentää huomioon otettavia tilanteita
  - yleensä Γ on esitettävissä (mahdollisesti äärettömänä) joukkona kaavoja
- päättelyoperaattorit eivät palauta totuusarvoa, vaan päättelyaskel on tai ei ole pätevä

Päättelyoperaattoreiden kielioppi

$$\text{Päättely} \quad ::= \quad \text{Kaava} \mid \text{Päättely} \Rightarrow \text{Kaava} \mid \text{Päättely} \Leftrightarrow \text{Kaava}$$

Päättelyoperaattorit käyttäytyvät syntaktisesti kuten vertailut

- esim. $\varphi \Leftrightarrow \psi \Rightarrow \xi$ tarkoittaa, että $\varphi \Leftrightarrow \psi$ on pätevä askel ja $\psi \Rightarrow \xi$ on pätevä askel
  - vrt. $0 \leq x < 1$ tarkoittaa että $0 \leq x$ ja $x < 1$
- esim. $(x^2 > 0 \to x^2 < 0) \to x = 0$ tuottaa aina **T**...
  - jos $x = 0$, se tuottaa $(\mathbf{F} \to \mathbf{F}) \to \mathbf{T}$ joka tuottaa **T**
  - jos $x \neq 0$, se tuottaa $(\mathbf{T} \to \mathbf{F}) \to \mathbf{F}$ joka tuottaa $\mathbf{F} \to \mathbf{F}$ joka tuottaa **T**
- ... mutta $(x^2 > 0 \Rightarrow x^2 < 0) \Rightarrow x = 0$ on syntaksivirhe ...
  - kielioppi ei salli sulkuja päättelyn ympärille
  - vrt. $(0 \leq x) < 1$
  - esim. C++ sallii sekä `1<1<1` että `(1<1)<1`, ja sen mielestä molemmat ovat `true`!
- ... ja $x^2 > 0 \Rightarrow x^2 < 0 \Rightarrow x = 0$ on virheellinen päättely
  - $x^2 > 0 \Rightarrow x^2 < 0$ ei ole pätevä: tilanteessa $x = 1$ pätee $x^2 > 0$ mutta ei $x^2 < 0$

Päättelyoperaattoreiden ominaisuuksia

- $\Rightarrow$ ja $\Leftrightarrow$ ovat *refleksiivisiä*
  - toisin sanoen, jokaisella φ pätee $\varphi \Rightarrow \varphi$ ja $\varphi \Leftrightarrow \varphi$
- $\Rightarrow$ ja $\Leftrightarrow$ ovat *transitiivisia*
  - jokaisella φ, ψ ja ξ joille $\varphi \Rightarrow \psi$ ja $\psi \Rightarrow \xi$ pätevät myös $\varphi \Rightarrow \xi$ pätee
  - jokaisella φ, ψ ja ξ joille $\varphi \Leftrightarrow \psi$ ja $\psi \Leftrightarrow \xi$ pätevät myös $\varphi \Leftrightarrow \xi$ pätee

- $\Leftrightarrow$ on *symmetrinen* mutta $\Rightarrow$ ei ole
  - jokaisella φ ja ψ joille $\varphi \Leftrightarrow \psi$ pätee myös $\psi \Leftrightarrow \varphi$ pätee
  - $\mathbf{F} \Rightarrow \mathbf{T}$ pätee mutta $\mathbf{T} \Rightarrow \mathbf{F}$ ei päde
- $\varphi \Rightarrow \psi$ pätee jos ja vain jos $\neg\psi \Rightarrow \neg\varphi$ pätee (mutta huomaa luku 4.4)
  - esim. vertaa sataa $\Rightarrow$ kastuu ja $\neg$ kastuu $\Rightarrow \neg$ sataa
- $\varphi \Leftrightarrow \psi$ pätee jos ja vain jos sekä $\varphi \Rightarrow \psi$ että $\psi \Rightarrow \varphi$ pätevät

Mahdottomasta saa päätellä mitä tahansa!

- ehkä yllättäen, $x^2 < 0 \;\Rightarrow\; x = 0$ on pätevä päättelyaskel!
- "kaikissa huomioon otettavissa tilanteissa joissa φ pätee, myös ψ pätee"
- $x^2 < 0$ ei päde missään tilanteessa, joten $(x = 0)$:n ei tarvitse päteä missään tilanteessa
- tarkastelkaamme melkein kaikkien pätevyksi hyväksymää päättelyä $x > 1 \Rightarrow x > 0$
  - jos $x \leq 0$, se tuottaa $\mathbf{F} \Rightarrow \mathbf{F}$
  - jos $0 < x \leq 1$, se tuottaa $\mathbf{F} \Rightarrow \mathbf{T}$
  - jos $1 < x$, se tuottaa $\mathbf{T} \Rightarrow \mathbf{T}$
  - se ei voi tuottaa $\mathbf{T} \Rightarrow \mathbf{F}$
- tarkastelkaamme melkein kaikkien vääränä pitämää päättelyä $x > 0 \Rightarrow x > 1$
  - jos $x \leq 0$, se tuottaa $\mathbf{F} \Rightarrow \mathbf{F}$
  - jos $0 < x \leq 1$, se tuottaa $\mathbf{T} \Rightarrow \mathbf{F}$
  - jos $1 < x$, se tuottaa $\mathbf{T} \Rightarrow \mathbf{T}$
  - (se ei voi tuottaa $\mathbf{F} \Rightarrow \mathbf{T}$)

$\Rightarrow$ on järkevää hyväksyä päättelyaskel jos ja vain jos se ei missään huomioon otettavassa tilanteessa tuota **T** $\Rightarrow$ **F**

- sitäpaitsi tätä ilmiötä olisi vaikea estää vaikka yritettäisiin, koska useimpien hyväksymillä säännöillä todella voi johtaa väärästä lähtökohdasta monenlaista
  - oletetaan $1 = 2$
  - kertomalla molemmat puolet mielivaltaisella luvulla a saadaan $a = 2a$
  - vähentämällä molemmilta puolilta a saadaan $0 = a$
  - sama onnistuu toisellekin mielivaltaiselle luvulle b, joten $0 = b$
  - $a = 0 = b$, joten $a = b$

  $\Rightarrow$ kaikki luvut ovat yhtäsuuria!

Modus ponens; eräs muoto oletuksella, että Γ on äärellinen (huomaa luku 4.4)

- $\varphi \Rightarrow \psi$ on pätevä jos ja vain jos $\Gamma \wedge \varphi \rightarrow \psi$ on tautologia
- $\varphi \Leftrightarrow \psi$ on pätevä jos ja vain jos $\Gamma \wedge \varphi \leftrightarrow \Gamma \wedge \psi$ on tautologia

$\Rightarrow$ $\Rightarrow$:n ja $\rightarrow$:n välillä on läheinen yhteys, samoin $\Leftrightarrow$:n ja $\leftrightarrow$:n välillä
  - ainot erot: Γ, syntaktiset säännöt ja tuloksen tyyppi
  - (luvussa 4.4 tulee vielä yksi ero)

$\Rightarrow$ monessa kirjassa esitellään vain $\Rightarrow$ ja $\Leftrightarrow$ tai vain $\rightarrow$ ja $\leftrightarrow$, ja saatetaan antaa esitellyille edellisten ja jälkimmäisten ominaisuuksia ristiin

- me haluamme kätevän merkinnän esittämään päättelyitä

  $\Rightarrow$ Γ:n käsittely ym. tavalla on tarpeen

Sijoittaminen kaavan sisään

- varoitus: näihin lakeihin tulee tarkennoksia luvuissa 4.4 ja 6
- kaavan sisässä olevan luvun tuottavan lausekkeen minkä tahansa esiintymän saa korvata yhtäsuuren luvun tuottavalla lausekkeella (huom. luku 6)
  - siis jos $x = y$ niin $\varphi(x) \Leftrightarrow \varphi(y)$
  - esim. jos $x = 2$ niin $y > x(x-1) \Leftrightarrow y > 2(x-1) \Leftrightarrow y > 2(2-1) \Leftrightarrow y > 2$
  - esim. jos $2x + 1 = y$ niin
    $5x = 2y + 1 \Leftrightarrow 5x = 2(2x+1) + 1 \Leftrightarrow 5x = 4x + 3 \Leftrightarrow x = 3$
  - esim. jos $x^2 = 5x$ niin $x^2 > 0 \Leftrightarrow 5x > 0$
- alikaavan minkä tahansa esiintymän saa korvata loogisesti ekvivalentilla alikaavalla
  - siis jos $\varphi \Leftrightarrow \psi$ niin $\xi(\varphi) \Leftrightarrow \xi(\psi)$
  - esim. koska $x < 2 \vee x > 2 \;\Leftrightarrow\; x \neq 2$, pätee
    $(x < 2 \vee x > 2) \wedge y > x \;\Leftrightarrow\; x \neq 2 \wedge y > x$
- jos $\varphi \Rightarrow \psi$ ja $\xi(P)$ on kaava, jossa korvattava P:n esiintymä on parillisen määrän negaatioita vaikutuspiirissä eikä ole $\leftrightarrow$:n vaikutuspiirissä, niin $\xi(\varphi) \Rightarrow \xi(\psi)$
  - negaatioksi lasketaan myös $\rightarrow$ vasemman argumenttinsa suhteen
  - $\xi(P)$ on kasvava P:n suhteen
  - esim. koska $x > 1 \;\Rightarrow\; x > 0$, pätee $x \leq 2 \wedge x > 1 \;\Rightarrow\; x \leq 2 \wedge x > 0$
- ... parittoman määrän negaatioita ... niin $\xi(\psi) \Rightarrow \xi(\varphi)$
  - negaatioksi lasketaan myös ...
  - $\xi(P)$ on vähenevä P:n suhteen
  - esim. koska $x > 1 \;\Rightarrow\; x > 0$, pätee $x > 0 \rightarrow f(x) > 0 \;\Rightarrow\; x > 1 \rightarrow f(x) > 0$

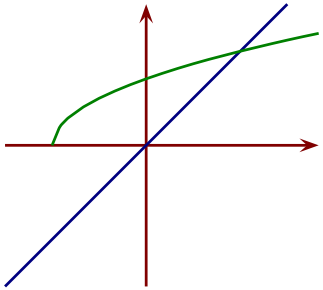
4.4 Totuusarvo ”määrittelemätön”

Haluamme käyttää $\Leftrightarrow$:a ja $\Rightarrow$:a yhtälöiden ratkaisemisessa

- esimerkki: $\sqrt{x+2} = x$
  $\Leftrightarrow x \geq 0 \wedge x + 2 = x^2$
  $\Leftrightarrow x \geq 0 \wedge x^2 - x - 2 = 0$
  $\Leftrightarrow x \geq 0 \wedge (x = 2 \vee x = -1)$
  $\Leftrightarrow x = 2$

  neliöidään puolittain, puolten pitää olla samanmerkkiset
  siirretään kaikki samalle puolelle, vaihdetaan puolet
  ratkaistaan toisen asteen yhtälö
  hylätään ehdon rikkova juuri

- onko siis $\sqrt{x+2} = x \Leftrightarrow x = 2$?
  - kun $x > 2$, tulee **F** $\Leftrightarrow$ **F**
  - kun $x = 2$, tulee **T** $\Leftrightarrow$ **T**
  - kun $-2 \leq x < 2$, tulee **F** $\Leftrightarrow$ **F**
  - kun $x < -2$, on $\sqrt{x+2}$ määrittelemätön, joten mitä $\sqrt{x+2} = x$ tarkoittaa?



yhtälön *juuri* on luku, joka toteuttaa yhtälön

Ongelma ei ratkea päättämällä, että jokainen määrittelemätön vertailu tuottaa **F**

- muutoin, koska $\neg(a \leq b) \Leftrightarrow a > b$, voitaisiin päätellä molemmat alla olevista:
  - **T** $\Leftrightarrow$ $\neg$**F** $\Leftrightarrow$ $\neg(\frac{1}{0} > 0)$, joten $\neg(\frac{1}{0} > 0)$
  - **T** $\Leftrightarrow$ $\neg$**F** $\Leftrightarrow$ $\neg(\frac{1}{0} \leq 0) \Leftrightarrow \frac{1}{0} > 0$, joten $\frac{1}{0} > 0$

$\Rightarrow \frac{1}{0}$ sekä on että ei ole suurempi kuin 0!

- ongelmana on, että $\neg$ voi muuttaa määrittelemättömän vertailun tuottaman **F**:n **T**:ksi
  - $\neg$(määrittelemätön) pitäisi olla määrittelemätön

Tietenkään ei voida sopia, että jokainen määrittelemätön vertailu tuottaa **T**

- muutoin $\sqrt{x} = \frac{2x}{x+1}$ saa juuren $x = -1$

$\Rightarrow$ ei voida edes sopia, että määrittelemätön on yhtä suuri kuin määrittelemätön

Ratkaisu

- jos aritmeettinen lauseke sisältää määrittelemättömän osan, niin lauseke kokonaisuudessaan katsotaan määrittelemättömäksi
- jos vertailun jompikumpi tai molemmat osapuolet ovat määrittelemättömiä, niin vertailu tuottaa uuden totuusarvon **U** eli määrittelemätön (undefined)
- sovimme, että **U** $\Leftrightarrow$ **F**
  - nyt $\sqrt{x+2} = x \Leftrightarrow x = 2$ myös kun $x < -2$
- $\neg$**U**, **U** $\wedge$ **U**, **U** $\vee$ **U**, **U** $\rightarrow$ **U** ja **U** $\leftrightarrow$ **U** tuottavat **U**
- **U** $\wedge$ **F** ja **F** $\wedge$ **U** tuottavat **F**
- **U** $\wedge$ **T** ja **T** $\wedge$ **U** tuottavat **U**
- **U** $\vee$ **F** ja **F** $\vee$ **U** tuottavat **U**
- **U** $\vee$ **T** ja **T** $\vee$ **U** tuottavat **T**
- $\wedge$:n ja $\vee$:n nämä ja alkuperäiset lait on helppo muistaa ajattelemalla, että
  - **F** on pienempi kuin **U** ja **U** on pienempi kuin **T**
  - $P \wedge Q$ tuottaa pienimmän
  - $P \vee Q$ tuottaa suurimman

$\Rightarrow$ kasvavuus ja vähenevyys toimivat kuten ennen **U**:n mukaantuloa

- edelleen $P \to Q$ tarkoittaa samaa kuin $\neg P \vee Q$
  - **F** $\to$ **U** ja **U** $\to$ **T** tuottavat **T**
  - **T** $\to$ **U** ja **U** $\to$ **F** tuottavat **U**
- edelleen $P \leftrightarrow Q$ tarkoittaa samaa kuin $(P \to Q) \wedge (Q \to P)$
  - **U** $\leftrightarrow$ **F**, **F** $\leftrightarrow$ **U**, **U** $\leftrightarrow$ **T** ja **T** $\leftrightarrow$ **U** tuottavat **U**
- näillä rakennetuilla totuusfunktioilla $\varphi(\mathbf{U})$ tuottaa **U** tai $\varphi(P)$ ei riipu P:stä
  - esim. $\varphi(P)$ on $P \wedge Q$: $P \wedge$ **F** ei riipu P:stä, **U** $\wedge$ **U** tuottaa **U** ja **U** $\wedge$ **T** tuottaa **U**

  $\Rightarrow$ ei saada aikaan lauseketta, joka tuottaisi em. ongelman $\neg$:n kanssa
  - esim. lauseketta $\varphi(P)$ siten, että $\varphi(\mathbf{F})$ ja $\varphi(\mathbf{U})$ tuottavat **F** ja $\varphi(\mathbf{T})$ tuottaa **T**
- päättelyssä **U** $\Leftrightarrow$ **F** em. ongelmaa $\neg$:n kanssa
  ei synny, koska $\neg(\mathbf{U} \Leftrightarrow \mathbf{F})$ on syntaksivirhe
- joitakin tuttuja lakeja menetetään, mutta ei kovin paljoa

Luvun 4.1 harmaat lait ja luvun 4.3 harmaat päättelysäännöt joudutaan muuttamaan

- $P \to P \Leftrightarrow \mathbf{T}$ ja $P \leftrightarrow P \Leftrightarrow \mathbf{T}$
- $P \vee \neg P \Leftrightarrow \mathbf{T}$ ja $P \vee \neg P \wedge Q \Leftrightarrow P \vee Q$
- $\varphi \Rightarrow \psi$ jos ja vain jos $\neg\psi \Rightarrow \neg\varphi$
- modus ponens
- sijoittaminen kaavan sisään

"On määritelty" -symboli

- edellä näimme, että käyttämillämme logiikan operaattoreilla ei voida kirjoittaa kaavaa $\varphi(P)$, joka tuottaa **T** jos ja vain jos P ei tuota **U**
  - se ei ole P:n suhteen vakio eikä sille $\varphi(\mathbf{U})$ tuota **U**
- silti jokaiselle lausekkeelle ja kaavalle erikseen voidaan kirjoittaa sellainen kaava
  - se ei ole funktio, jolle tutkittava lauseke tai kaava annetaan parametrina, vaan se täytyy kirjoittaa aina tapauskohtaisesti
  - merkitsemme sitä $\lfloor\varphi\rceil$
  - $\lfloor\varphi\rceil$ tuottaa aina **F** tai **T**, ei koskaan **U**
- muodostustapa aritmeettisille lausekkeille
  - jos x on lukuarvoinen muuttuja, niin $\lfloor x\rceil$ on **T**
  - $\lfloor f(x) + f(y)\rceil$ on $\lfloor f(x)\rceil \wedge \lfloor f(y)\rceil$, ja samoin vähennys- ja kertolaskulle
  - $\lfloor \frac{f(x)}{f(y)}\rceil$ on $\lfloor f(x)\rceil \wedge \lfloor f(y)\rceil \wedge f(y) \neq 0$
  - $\lfloor \sqrt{f(x)}\rceil$ on $\lfloor f(x)\rceil \wedge f(x) \geq 0$
  - ...
  - nämä on helppo keksiä tarvittaessa, kun muistaa periaatteen: jokaisen osan sekä niitä yhdistävän operaation täytyy olla määritelty

- muodostustapa loogisille lausekkeille
  - $\lfloor \mathbf{T} \rceil$ ja $\lfloor \mathbf{F} \rceil$ ovat $\mathbf{T}$, ja $\lfloor \mathbf{U} \rceil$ on $\mathbf{F}$
  - $\lfloor \neg\varphi \rceil$ on $\lfloor \varphi \rceil$
  - $\lfloor \varphi \wedge \psi \rceil$ on $\lfloor \varphi \rceil \wedge \neg\varphi \vee \lfloor \psi \rceil \wedge \neg\psi \vee \lfloor \varphi \rceil \wedge \lfloor \psi \rceil$
  - $\lfloor \varphi \vee \psi \rceil$ on $\lfloor \varphi \rceil \wedge \varphi \vee \lfloor \psi \rceil \wedge \psi \vee \lfloor \varphi \rceil \wedge \lfloor \psi \rceil$
  - $\lfloor \varphi \to \psi \rceil$ on ...
  - $\lfloor \varphi \leftrightarrow \psi \rceil$ on $\lfloor \varphi \rceil \wedge \lfloor \psi \rceil$
- nämäkin on helppo keksiä tarvittaessa
- usein voi käyttää helposti muistettavaa alalikiarvoa $\lfloor \varphi \rceil \wedge \lfloor \psi \rceil$

Ei saada "φ on määrittelemätön" sanomalla $\varphi \Leftrightarrow \mathbf{U}$

- $\varphi \Leftrightarrow \mathbf{U}$ pätee myös kun φ tuottaa $\mathbf{F}$
- ei myöskään saada "φ tuottaa $\mathbf{F}$" sanomalla $\varphi \Leftrightarrow \mathbf{F}$

$\Rightarrow$ käytämme tarvittaessa ilmauksia "tuottaa $\mathbf{U}$" ja "tuottaa $\mathbf{F}$"

- sen sijaan saadaan "φ tuottaa $\mathbf{T}$" sanomalla $\varphi \Leftrightarrow \mathbf{T}$ tai jopa vain φ

Luvun 4.1 harmaat lait korjattuina

- $P \vee \neg P \vee \neg\lfloor P \rceil \Leftrightarrow \mathbf{T}$ ja $P \vee \neg P \Leftrightarrow \lfloor P \rceil \Leftrightarrow P \to P \Leftrightarrow P \leftrightarrow P$
- $P \vee \neg P \wedge Q \vee \neg\lfloor P \rceil \wedge Q \Leftrightarrow P \vee Q$

Harmaat päättelysäännöt saadaan käyttöön lisäämällä sopivasti ”ja on määritelty”

- palauttaa tilanteen kahteen totuusarvoon **F** ja **T**
- esim. $\varphi \wedge \lfloor\varphi\rceil \Rightarrow \psi \wedge \lfloor\psi\rceil$ jos ja vain jos $\neg(\psi \wedge \lfloor\psi\rceil) \Rightarrow \neg(\varphi \wedge \lfloor\varphi\rceil)$
- valitettavasti se on toisinaan kömpelöä, koska lisättävää tulee paljon

$\Rightarrow$ tarkastelemme tilanteita, joissa lisäyksiä ei tarvita

Oletamme, että φ ja ψ on koottu vain edellä mainituilla operaattoreilla

- funktiomuotoinen ”on määritelty” on kielletty, koska sillä voi tuottaa vastaesimerkkejä alla oleville
- jos $\varphi \Rightarrow \psi$ ja $\xi(P)$ on kaava, jossa korvattava P:n esiintymä on parillisen määrän negaatioita vaikutuspiirissä eikä ole $\leftrightarrow$:n vaikutuspiirissä, niin $\xi(\varphi) \Rightarrow \xi(\psi)$
  - negaatioksi lasketaan myös $\rightarrow$ vasemman argumenttinsa suhteen
  - $\xi(P)$ on kasvava P:n suhteen
  - jos $\xi(P)$ ei riipu P:stä, niin triviaalisti $\xi(\varphi) \Leftrightarrow \xi(\psi)$
  - muussa tapauksessa $\xi(\mathbf{U})$ tuottaa **U** ja $\xi(\mathbf{F})$ tuottaa enintään **U**, eli **F** tai **U**, mikä hoitaa tapauksen jossa φ tuottaa **U** ja ψ tuottaa **F**; loput hoitaa kasvavuus
- sama pätee $\Leftrightarrow$:lle
- jos $\varphi \Leftrightarrow \psi$ ja φ ja ψ ovat määrittelemättömät täsmälleen samoilla muuttujien arvoyhdistelmillä, niin $\xi(\varphi) \Leftrightarrow \xi(\psi)$
  - φ tuottaa **F** jos ja vain jos ψ tuottaa **F**, ja samoin **U** ja **T**

- jos $\varphi \Rightarrow \psi$ ja $\xi(P)$ on kaava, jossa korvattava P:n esiintymä on parittoman määrän negaatioita vaikutuspiirissä eikä ole $\leftrightarrow$:n vaikutuspiirissä, ja φ ja ψ ovat määrittelemättömät täsmälleen samoilla muuttujien arvoyhdistelmillä, niin $\xi(\psi) \Rightarrow \xi(\varphi)$
  - negaatioksi lasketaan myös $\rightarrow$ vasemman argumenttinsa suhteen
  - $\xi(P)$ on vähenevä P:n suhteen, mikä antaa väitteen, koska $\varphi \Rightarrow \psi$ voi toteutua vain seuraavasti: **F** $\Rightarrow$ **F**, **F** $\Rightarrow$ **T**, **U** $\Rightarrow$ **U** ja **T** $\Rightarrow$ **T**

Entä määrittelemättömät luvun tuottavat lausekkeet?

- kaikki vertailut, joissa jokin osa on määrittelemätön, tuottavat **U**
- $f(x) = g(x)$ ei päde, jos $f(x)$ tai $g(x)$ on määrittelemätön

$\Rightarrow$ sivun 81 sääntö ei anna lupaa korvata $\varphi(f(x))$:n $\varphi(g(x))$:llä

- toisaalta jos sekä $f(x)$ että $g(x)$ ovat määrittelemätömiä, niin
  - sekä $f(x) > 0$ että $g(x) > 0$ tuottaa **U**
  - samoin kaikille muillekin vertailuille, joissa $f(x)$ tai $g(x)$ on osapuolena

  $\Rightarrow \varphi(f(x)) \Leftrightarrow \varphi(g(x))$
- jos jokaisella x:n arvolla joko $f(x) = g(x)$ tai sekä $f(x)$ että $g(x)$ ovat määrittelemättömiä, niin $\varphi(f(x)) \Leftrightarrow \varphi(g(x))$
- kuitenkaan $f(x) = f(x)$ ei päde, jos $f(x)$ on määrittelemätön
  - silti silloin $\varphi(f(x)) \Leftrightarrow \varphi(f(x))$, koska myös **U** $\Leftrightarrow$ **U** on pätevä päättelyskel
- tietenkin $\lfloor f(x) \rceil \Rightarrow f(x) = f(x)$
- $f(x) \neq f(x)$ ei päde koskaan, vaan tuottaa aina **F** tai **U**

Kaikki vertailut, joissa jokin osa on määrittelemätön, tuottavat **U**

- $\frac{1}{0} > 0$ ei päde, mutta myöskään $\frac{1}{0} \leq 0$ ei päde
- $\frac{1}{0} > 0 \Leftrightarrow$ **U** ja $\frac{1}{0} \leq 0 \Leftrightarrow$ **U** , joten myös $\neg(\frac{1}{0} > 0) \Leftrightarrow$ **U** ja $\neg(\frac{1}{0} \leq 0) \Leftrightarrow$ **U**
- koska **U** $\Leftrightarrow$ **F**, saamme $\frac{1}{0} > 0 \Leftrightarrow \frac{1}{0} \leq 0 \Leftrightarrow \neg(\frac{1}{0} > 0) \Leftrightarrow \neg(\frac{1}{0} \leq 0) \Leftrightarrow$ **F**

$\Rightarrow$ monet tutut lait ovat voimassa määrittelemättömillekin lausekkeille, kuten
  - $f(x) \neq f(y) \Leftrightarrow \neg(f(x) = f(y))$
  - $f(x) \leq f(y) \Leftrightarrow \neg(f(x) > f(y))$
  - $f(x) \leq f(y) \Leftrightarrow f(x) < f(y) \vee f(x) = f(y)$
  - $f(x) < f(y) \Leftrightarrow f(x) \leq f(y) \wedge f(x) \neq f(y)$
- $f(x) - f(x) = 0$, $0 \cdot f(x) = 0$ ja $\frac{f(x)}{f(x)} = 1$ eivät päde kun $f(x)$ on määrittelemätön
  - silloin "=" tuottaa **U**
- esim. päättely $0 = 0 \Rightarrow 0 = 0 \cdot \frac{1}{0} \Rightarrow 0 = 1$ on kahdesti väärin, koska
  - 0:n korvaaminen $0 \cdot \frac{1}{0}$:lla on väärin, koska ne eivät ole yhtäsuuret eivätkä molemmat ole määrittelemättömiä
  - $0 \cdot \frac{1}{0} = 1$ ei päde

Ratkaistaan $x^2 + 2x + 2 = 0$ kaavalla $x^2 + px + q = 0 \Leftrightarrow x = -\frac{p}{2} \pm \sqrt{(\frac{p}{2})^2 - q}$

$\Leftrightarrow x = -\frac{2}{2} \pm \sqrt{(\frac{2}{2})^2 - 2} = -1 \pm \sqrt{-1} \Leftrightarrow x = -1 - \sqrt{-1} \vee x = -1 + \sqrt{-1}$

- siis $x^2 + 2x + 2 = 0 \Leftrightarrow x = -1 - \sqrt{-1} \vee x = -1 + \sqrt{-1}$
- minkä tahansa reaaliluvun sijoitus x:ksi tuottaa **F** $\Leftrightarrow$ **U** (mikä on oikein)
- kompleksilukuja käytettäessä tilanne muuttuu
  - neliöjuuri on määritelty kaikille kompleksiluvuille, myös negatiivisille reaaliluvuille
  - $(-1 - i)$:n ja $(-1 + i)$:n sijoitus x:ksi tuottaa **T** $\Leftrightarrow$ **T**
  - muiden kompleksilukujen sijoitus tuottaa **F** $\Leftrightarrow$ **F**

$\Rightarrow$ $f(x) = 0 \Leftrightarrow x = x_1 \vee \cdots \vee x = x_n$, missä $x_1, \ldots, x_n$ eivät sisällä x ja ovat määritelty tarkoittaa, että yhtälön $f(x) = 0$ juuret ovat $x_1, \ldots, x_n$

- **T** $\Leftrightarrow$ **T** saadaan jos ja vain jos x on juuri ja samalla jokin luvuista $x_1, \ldots, x_n$
- **T** $\Leftrightarrow$ **F** eli **T** $\Leftrightarrow$ **U** saadaan joss x on juuri mutta ei mikään luvuista $x_1, \ldots, x_n$
- **F** $\Leftrightarrow$ **T** eli **U** $\Leftrightarrow$ **T** saadaan joss x ei ole juuri mutta *on määritelty* ja jokin ...
- muulloin saadaan **F** $\Leftrightarrow$ **F** eli **F** $\Leftrightarrow$ **U** eli ...

$\Rightarrow$ päätelmä $0 \cdot x = 1 \Leftrightarrow x = \frac{1}{0}$ on loogisesti oikein

- olisi kuitenkin väärin sanoa, että $\frac{1}{0}$ on yhtälön $0 \cdot x = 1$ juuri, koska $\frac{1}{0}$ ei ole luku

Modus ponens (edellä esitetty muoto) muutetaan seuraavasti

- $\varphi \Rightarrow \psi$ on pätevä jos ja vain jos $\Gamma \wedge \varphi \wedge \lfloor \Gamma \wedge \varphi \rceil \rightarrow \psi$ on tautologia
- $\varphi \Leftrightarrow \psi$ on pätevä jos ja vain jos $\Gamma \wedge \varphi \wedge \lfloor \Gamma \wedge \varphi \rceil \leftrightarrow \Gamma \wedge \psi \wedge \lfloor \Gamma \wedge \psi \rceil$ on tautologia
- $P \wedge Q \wedge \lfloor P \wedge Q \rceil$ tuottaa saman kuin $P \wedge Q \wedge \lfloor P \rceil \wedge \lfloor Q \rceil$, joten:
- $\varphi \Rightarrow \psi$ on pätevä jos ja vain jos $\Gamma \wedge \lfloor \Gamma \rceil \wedge \varphi \wedge \lfloor \varphi \rceil \rightarrow \psi$ on tautologia
- $\varphi \Leftrightarrow \psi$ on pätevä jos ja vain jos $\Gamma \wedge \lfloor \Gamma \rceil \wedge \varphi \wedge \lfloor \varphi \rceil \leftrightarrow \Gamma \wedge \lfloor \Gamma \rceil \wedge \psi \wedge \lfloor \psi \rceil$ on tautologia

Oletetaan, että on annettu kokoelma Γ aina määriteltyjä kaavoja

- merkitsemme
  - päättely$_\Gamma$ tarkoittaa päättelyä, jossa Γ:n tietoja saa hyödyntää
  - pätee$_\Gamma$ tarkoittaa, että pätee kaikissa Γ:n sallimissa tilanteissa
- jos voi päätellä$_\Gamma$ $\varphi \Rightarrow \psi$, niin $\varphi \wedge \lfloor \varphi \rceil \rightarrow \psi \wedge \lfloor \psi \rceil$ pätee$_\Gamma$
- jos $\varphi \wedge \lfloor \varphi \rceil \rightarrow \psi \wedge \lfloor \psi \rceil$ pätee$_\Gamma$, niin saa päätellä$_\Gamma$ $\varphi \Rightarrow \psi$
- jos voi päätellä$_\Gamma$ $\varphi \Leftrightarrow \psi$, niin $\varphi \wedge \lfloor \varphi \rceil \leftrightarrow \psi \wedge \lfloor \psi \rceil$ pätee$_\Gamma$
- jos $\varphi \wedge \lfloor \varphi \rceil \leftrightarrow \psi \wedge \lfloor \psi \rceil$ pätee$_\Gamma$, niin saa päätellä$_\Gamma$ $\varphi \Leftrightarrow \psi$
- esim. kokonaisluvuilla $n > 0 \Leftrightarrow n \geq 1$, joten $n > 0 \leftrightarrow n \geq 1$ pätee aina kun $n \in \mathbb{Z}$
- esim. reaaliluvuilla $\frac{1}{x} < 0 \vee x = 0 \vee x > 0$ eli $\frac{1}{x} \geq 0 \wedge x \neq 0 \rightarrow x > 0$, joten $\frac{1}{x} \geq 0 \Rightarrow x > 0$ pätee aina kun $x \in \mathbb{R}$

Ohjelmointikielten ”ja” ja ”tai”

- ohjelmointikielissä ei yleensä ole totuusarvoa **U**, mutta ohjelma voi kaatua: †
- C:n loogisen operaattorin oikea puoli lasketaan vain jos tarpeen
- Pascalin loogisen operaattorin molemmat puolet lasketaan aina

⇒ ne toimivat näin:

&&	F	†	T
F	F	F	F
†	†	†	†
T	F	†	T

and	F	†	T
F	F	†	F
†	†	†	†
T	F	†	T

||	F	†	T
F	F	†	T
†	†	†	†
T	T	T	T

or	F	†	T
F	F	†	T
†	†	†	†
T	T	†	T

⇒ ohjelmointikielten loogiset operaattorit eivät ole samat kuin logiikan

- C:n tapa on kätevä esim. tilanteessa

```
for( i = 0; i < n && A[i] != x; ++i );
```

5 Joukot, relaatiot ja funktiot

5.1 Joukot

Joukko on kokoelma *alkioita*

- alkiot voivat olla melkein mitä vain
  - myös muita joukkoja: potenssijoukko, ositus, ...
  - joukko ei voi olla itsensä alkio
- alkio joko on tai ei ole joukossa; se ei ole moneen kertaan
  - toinen tapa ajatella: moneen kertaan joukossa oleminen on samanveroista kuin yhteen kertaan joukossa oleminen
- joukon kannalta millään muulla ei ole merkitystä kuin mitkä alkiot ovat siinä

Aaltosulkeet, tapaus 1

- äärellinen joukko voidaan ilmaista luettelemalla sen alkiot aaltosulkeiden välissä
  - esim. $\{2, 3, 5, 7\}$
  - esim. Pohjoismaat $= \{$Suomi, Ruotsi, Norja, Tanska, Islanti$\}$
- luettelointijärjestyksellä ei ole väliä
  - esim. $\{2, 3, 5, 7\} = \{3, 7, 2, 5\}$
- toistolla ei ole merkitystä
  - esim. $\{2, 3, 5, 7\} = \{3, 7, 7, 2, 3, 5\}$

Aaltosulkeet, tapaus 2

- jotkin äärettömät joukot voidaan ilmaista aaltosulkeiden ja kolmen pisteen avulla
  - $\mathbb{N} = \{0, 1, 2, \ldots\}$ luonnolliset luvut
  - $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ kokonaisluvut
- tämä vaatii, että tapa jatkaa luetteloa loputtomiin on lukijalle selvä
  - $\{1, 2, 4, 8, 15, \ldots\}$

Joukkoon kuuluminen

- $a \in A$ tarkoittaa, että alkio a kuuluu joukkoon A
  - $5 \in \mathbb{N}$
  - Suomi $\in$ Pohjoismaat
- $a \notin A$ tarkoittaa, että alkio a ei kuulu joukkoon A
  - $-5 \notin \mathbb{N}$
  - Latvia $\notin$ Pohjoismaat
- $a \notin A \;\Leftrightarrow\; \neg(a \in A)$
- kaksi joukkoa on samat, jos ja vain jos niissä on samat alkiot
  - $A = B \;\Leftrightarrow\; \forall x : x \in A \leftrightarrow x \in B$

$\emptyset$ on tyhjä joukko eli joukko, jossa ei ole yhtään alkiota

- $\forall x : x \notin \emptyset$
- myös merkintää $\{\}$ käytetään, mutta se on harvinainen

Joukon muodostaminen ehdon avulla (aaltosulkeet, tapaus 3)

- $\{a \in A \mid \varphi(a)\}$ on niiden a joukko, jotka kuuluvat A:han ja toteuttavat ehdon $\varphi(a)$
  - $\{n \in \mathbb{Z} \mid n > 0\}$ positiiviset kokonaisluvut
  - $\{n \in \mathbb{Z} \mid n \geq 0\} = \mathbb{N}$
  - $\{n \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : n = 2k\}$ parilliset kokonaisluvut
  - $\{n \in \mathbb{N} \mid n > 1 \wedge \neg\exists k \in \mathbb{N} : \exists h \in \mathbb{N} : 1 < k \leq h \wedge n = kh\}$ alkuluvut
- $x \in \{a \in A \mid \varphi(a)\} \Leftrightarrow x \in A \wedge \varphi(x)$
- jos A tiedetään asiayhteydestä, voidaan kirjoittaa $\{a \mid \varphi(a)\}$
  - tällöin A:ta kutsutaan *perusjoukoksi* eli *universumiksi*
- ns. Russellin paradoksin vuoksi ei voida sallia $\{a \mid \varphi(a)\}$ rajoituksetta
  - olkoon $R = \{a \mid a \notin a\}$

  $\Rightarrow$ jokaiselle x pätee $x \in R \Leftrightarrow x \notin x$

  $\Rightarrow$ R:lle pätee $R \in R \Leftrightarrow R \notin R$ 

Yhden alkion joukko on eri asia kuin ko. alkio

$$\forall a : a \neq \{a\}$$

Äärellisen joukon koko

- $|A|$ tarkoittaa äärellisen joukon A alkioiden määrää
  - $|\text{Pohjoismaat}| = 5$
  - $|\{3, 7, 7, 2, 3, 5\}| = 4$

- $|\{a_1, \ldots, a_n\}| \leq n$ ja $|\emptyset| = 0$
- äärettömien joukkojen koot ovat pitkä tarina!
  - puhutaan "mahtavuuksista"
- joskus merkitään $|A| = \infty$ tarkoittamaan, että A on ääretön

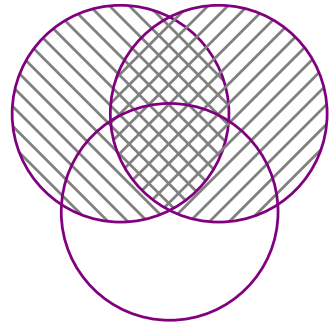
Osajoukko

- $A \subseteq B$ tarkoittaa, että A on B:n osajoukko
  - jokainen A:n alkio on myös B:n alkio (mutta ei välttämättä toisinpäin)
  - esim. $\{3, 7\} \subseteq \{2, 3, 5, 7\}$ ja $\{2, 3, 5, 7\} \subseteq \{2, 3, 5, 7\}$
  - esim. Pohjoismaat $\subseteq$ Euroopan_valtiot
- $A \subseteq B \Leftrightarrow \forall x : x \in A \rightarrow x \in B$
- $A \not\subseteq B \Leftrightarrow \neg(A \subseteq B)$
- jokainen joukko on itsensä osajoukko
  - $\forall A : A \subseteq A$
- tyhjä joukko on jokaisen joukon osajoukko
  - $\forall A : \emptyset \subseteq A$
- mikään muu joukko ei ole tyhjän joukon osajoukko
  - $\forall A : A = \emptyset \vee A \not\subseteq \emptyset$
- kaikille joukoille A ja B pätee $A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$
- äärellisille joukoille A ja B pätee $A \subseteq B \Rightarrow |A| \leq |B|$

Aito osajoukko

- $A \subset B$ tarkoittaa, että A on B:n aito osajoukko
  - jokainen A:n alkio on myös B:n alkio, mutta ei toisinpäin
  - esim. $\{3, 7\} \subset \{2, 3, 5, 7\}$ mutta $\{2, 3, 5, 7\} \not\subset \{2, 3, 5, 7\}$
  - esim. Pohjoismaat $\subset$ Euroopan_valtiot
- $(\forall x : x \in A \rightarrow x \in B) \wedge \exists x : x \notin A \wedge x \in B$
- mikään joukko ei ole itsensä aito osajoukko
  - $\forall A : A \not\subset A$
- mikään joukko ei ole tyhjän joukon aito osajoukko
  - $\forall A : A \not\subset \emptyset$ (mutta $\emptyset \subseteq \emptyset$)
- kaikille joukoille A ja B pätee
  - $A \subset B \Leftrightarrow A \subseteq B \wedge A \neq B$
  - $A \subseteq B \Leftrightarrow A \subset B \vee A = B$
- äärellisille joukoille A ja B pätee $A \subset B \Rightarrow |A| < |B|$
- joillakin matematiikan aloilla on tapana käyttää $\subset$ osajoukon merkkinä
  - sekaannuksen vaara!
  - tietojenkäsittelyssä $\subseteq$ on osajoukko ja $\subset$ on aito osajoukko

Unioni ja leikkaus



- joukkojen A ja B **unioni** sisältää sekä A:n että B:n alkiot (ja vain ne)

  $$x \in A \cup B \;\Leftrightarrow\; x \in A \vee x \in B$$

  - esim. $\{2,3,5,7\} \cup \{1,3,6,7,8\} = \{1,2,3,5,6,7,8\}$
  - esim. Pohjoismaat $\cup$ Suomalais-ugrilaiset_maat $=$ {Suomi, Ruotsi, Norja, Tanska, Islanti, Viro, Unkari}
- joukkojen A ja B **leikkaus** koostuu niistä, jotka ovat sekä A:n että B:n alkioita

  $$x \in A \cap B \;\Leftrightarrow\; x \in A \wedge x \in B$$

  - esim. $\{2,3,5,7\} \cap \{1,3,6,7,8\} = \{3,7\}$
  - esim. Pohjoismaat $\cap$ Suomalais-ugrilaiset_maat $= \{$Suomi$\}$
- kaikille joukoille A, B ja C pätee (älä opettele ulkoa, tärkeät oppii ilmankin)
  - $A \cup \emptyset = A$
  - $A \cap \emptyset = \emptyset$
  - $A \cup A = A \cap A = A$
  - $A \cup B = B \cup A$ ja $(A \cup B) \cup C = A \cup (B \cup C)$
  - $A \cap B = B \cap A$ ja $(A \cap B) \cap C = A \cap (B \cap C)$
  - $A \cup (A \cap B) = A \cap (A \cup B) = A$
  - $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
  - $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
  - $A \subseteq A \cup B$ ja $B \subseteq A \cup B$
  - $A \cap B \subseteq A$ ja $A \cap B \subseteq B$

näitä listoja käydään läpi siksi, että siten joukkoajattelu tulee tutuksi

- äärellisille joukoille A ja B pätee
  - $|A \cup B| \leq |A| + |B|$
  - $|A \cap B| \leq |A|$ ja $|A \cap B| \leq |B|$
  - $|A \cup B| = |A| + |B| - |A \cap B|$

Joukkojen erotus

- joukkojen A ja B erotus sisältää ne ja vain ne A:n alkiot, jotka eivät ole B:n alkioita

  $x \in A \setminus B \Leftrightarrow x \in A \wedge x \notin B$ myös merkintää $A - B$ käytetään

  - esim. $\{2,3,5,7\} \setminus \{1,3,6,7,8\} = \{2,5\}$
    mutta $\{1,3,6,7,8\} \setminus \{2,3,5,7\} = \{1,6,8\}$
  - esim. Pohjoismaat $\setminus$ Suomalais-ugrilaiset_maat $= \{$Ruotsi, Norja, Tanska, Islanti$\}$
    mutta Suomalais-ugrilaiset_maat $\setminus$ Pohjoismaat $= \{$Viro, Unkari$\}$
  - esim. $(\{1\} \setminus \{1\}) \setminus \{1\} = \emptyset \setminus \{1\} = \emptyset$ mutta $\{1\} \setminus (\{1\} \setminus \{1\}) = \{1\} \setminus \emptyset = \{1\}$

$\Rightarrow$ joukkojen erotus ei ole vaihdannainen eikä liitännäinen

- kaikille joukoille A, B ja C pätee (älä opettele ulkoa, tunnista de Morganin lait)
  - $A \setminus A = \emptyset$ ja $A \setminus B = A \setminus (A \cap B)$
  - $(A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$
  - $(A \cap B) \setminus C = (A \setminus C) \cap (B \setminus C)$
  - $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$
  - $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$
  - $A \setminus \emptyset = A \setminus (B \setminus A) = A$ ja $\emptyset \setminus A = (A \setminus B) \cap B = \emptyset$
  - $A \setminus B \subseteq A$

- äärellisille joukoille A ja B pätee
  - $|A \setminus B| \leq |A|$
  - $|A \setminus B| = |A| - |A \cap B|$

Komplementti

- jos perusjoukko U on tiedossa, voidaan $U \setminus A$ kirjoittaa lyhyemmin $\overline{A}$
  - kaikkien puheena olevien alkioiden oletetaan kuuluvan U:hun
  - esim. jos kaikki muuttujat edustavat reaalilukuja, niin $U = \mathbb{R}$
- $x \in \overline{A} \;\Leftrightarrow\; x \notin A$
- kaikille joukoille A ja B pätee (nämä voisi ehkä kenties opetella)
  - $\overline{A \cup B} = \overline{A} \cap \overline{B}$
  - $\overline{A \cap B} = \overline{A} \cup \overline{B}$
  - $\overline{A} \cap A = \emptyset$ ja $\overline{A} \cup A = U$
  - $\overline{\overline{A}} = A$
  - $\overline{\emptyset} = U$ ja $\overline{U} = \emptyset$

Potenssijoukko

- joukon *potenssijoukko* on sen kaikkien osajoukkojen joukko

$$2^A \;=\; \{X \mid X \subseteq A\}$$

  - esim. $2^{\{2,3,5\}} = \{\emptyset, \{2\}, \{3\}, \{5\}, \{2,3\}, \{2,5\}, \{3,5\}, \{2,3,5\}\}$
  - esim. $2^{\emptyset} = \{\emptyset\} \neq \emptyset$

- käytetään myös merkintää $\mathcal{P}(A)$
- potenssijoukko on määritelty, vaikka perusjoukkoa ei olisi sovittu
- $x \in 2^A \Leftrightarrow x \subseteq A$
- äärellisillä joukoilla $|2^A| = 2^{|A|}$
- potenssijoukkoja käytetään mm. merkkijonojen etsintäalgoritmeissa
  - muunnettaessa ns. epädeterminististen äärellinen automaatti deterministiseksi

Tulojoukko eli karteesinen tulo

- alkioiden a ja b muodostama *pari* merkitään (a, b)
  - esim. $(3, -7)$
  - esim. (Keski-Suomi, Jyväskylä)
  - parissa järjestys on olennainen: $(1, 3) \neq (3, 1)$
  - pari voitaisiin määritellä joukko-opillisesti, mutta emme tee niin, koska määritelmä on vaikeampi ymmärtää kuin parin idea ja vain esittää sen koodattuna
- A:n ja B:n *tulojoukko* eli *karteesinen tulo* on kaikkien niiden parien joukko, joiden ensimmäinen alkio on A:sta ja jälkimmäinen B:stä

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$$

  - esim. $\{2, 3, 5\} \times \{\mathsf{false}, \mathsf{true}\} =$
    $\{(2, \mathsf{false}), (2, \mathsf{true}), (3, \mathsf{false}), (3, \mathsf{true}), (5, \mathsf{false}), (5, \mathsf{true})\}$
  - esim. $\mathbb{R} \times \mathbb{R}$ on tuttu x-y–koordinaatisto
- tulojoukko on määritelty, vaikka perusjoukkoa ei olisi sovittu

- käytetään myös merkintöjä $A^2 = A \times A$, $A^3 = A \times A \times A$ jne.
  - x-y–koordinaatisto voidaan ilmaista myös $\mathbb{R}^2$
- tulojoukot ovat tärkeitä perusjoukkoina rakenteellisille kohteille
  - esim. navigointiohjelmassa tienpätkät ovat joukon Risteykset $\times$ Risteykset alkioita
- parametrilistat, tietueet ja oliot ovat karteesisten tulojen alkioita
  - tätä ajattelutapaa harrastetaan varsinkin funktionaalisessa ohjelmoinnissa
- karteesinen tulo ei ole vaihdannainen
  - parissa järjestys on olennainen

  $\Rightarrow \{1\} \times \{3\} = \{(1,3)\} \neq \{(3,1)\} = \{3\} \times \{1\}$
- karteesinen tulo ei ole liitännäinen, mutta usein $(A \times B) \times C$, $A \times (B \times C)$ ja $A \times B \times C$ samaistetaan jopa ilman mainintaa
- kaikille joukoille A, B, C ja D pätee
  - $A \times (B \cup C) = (A \times B) \cup (A \times C)$
  - $(A \cup B) \times C = (A \times C) \cup (B \times C)$
  - $(A \cap B) \times (C \cap D) = (A \times C) \cap (B \times D)$
  - $A \times (B \setminus C) = (A \times B) \setminus (A \times C)$
  - $(A \setminus B) \times C = (A \times C) \setminus (B \times C)$
  - $A \times \emptyset = \emptyset \times A = \emptyset$
- usein $(A \cup B) \times (C \cup D) \neq (A \times C) \cup (B \times D)$
  - esim. $(\{1\} \cup \{2\}) \times (\{a\} \cup \{b\}) = \{1,2\} \times \{a,b\} = \{(1,a),(1,b),(2,a),(2,b)\}$ mutta $(\{1\} \times \{a\}) \cup (\{2\} \times \{b\}) = \{(1,a)\} \cup \{(2,b)\} = \{(1,a),(2,b)\}$

- äärellisillä joukoilla pätee $|A \times B| = |A||B|$
- edellä otettiin käyttöön uusi merkintätapa: $\{(a, b) \mid \ldots\}$
  - "{":n ja "|":n välissä käytetään monenlaista
  - aina ei ole täysin selvää, mitä tarkoitetaan, esim. $A_i = \{(i + j)^2 \mid 0 \leq j \leq i\}$

Äärelliset jonot

- A^* tarkoittaa A:sta muodostettujen äärellisten jonojen joukkoa
- tietojenkäsittelyssä sitä käytetään paljon, koska syötteet ovat äärellisiä merkkijonoja
- merkkijonot merkitään usein pistämällä merkkejä peräkkäin ilman väli- yms. merkkejä

$\Rightarrow$ tarvitaan erityinen symboli tarkoittamaan tyhjää jonoa: ε

- esim. $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, \ldots\}$
- toisinaan käytetään $A^+ = A^* \setminus \{\varepsilon\}$
- äärellisiin merkkijonoihin ja niistä muodostettuihin joukkoihin liittyy paljon tärkeää asiaa, mutta niillä on oma kurssi

5.2 Relaatiot

Binäärirelaatio on symboli, joka ottaa kaksi alkiota ja tuottaa (tavallisesti) totuusarvon

- esim. aritmetiikassa "$=$", "$\neq$", "$<$", "$\leq$", "$>$" ja "$\geq$"
- esim. joukko-opissa "$\in$", "$\subseteq$" ja "$=$"
- esim. asuntomarkkinoilla tarkoittakoon $h \oint a$ että henkilö h omistaa asunnon a (osittain tai kokonaan)
- logiikassa totuusarvot ovat laskuoperaatioilla käsiteltävien suureiden roolissa ja relaatioilla muodostetaan päättelyitä, jotka pätevät tai eivät päde
  $\Rightarrow$ "$\Rightarrow$" ja "$\Leftrightarrow$" ovat relaatioita, vaikka nimenomaan ne eivät tuota totuusarvoa
- tuntematonta binäärirelaatiota merkitään toisinaan "$\sim$"

Usein on sovittu *lähtöjoukko* A ja *maalijoukko* B siten,
että ensimmäinen alkio on A:stä ja jälkimmäinen B:stä

- esimerkkejä
  - "$\leq$": molemmat alkiot ovat $\mathbb{R}$:stä
  - "$\Rightarrow$": molemmat alkiot ovat $\{\mathsf{false}, \mathsf{true}\}$:stä
  - "$\oint$": ensimmäinen alkio on joukosta Henkilöt ja toinen joukosta Asunnot
- tällöin on tapana merkitä $\sim \subseteq A \times B$ tai "$\sim$" $\subseteq A \times B$
  - esim. "$\leq$" $\subseteq \mathbb{R}^2$
  - esim. "$\oint$" $\subseteq$ Henkilöt $\times$ Asunnot

- relaatio ajatellaan niiden parien joukkona, joille relaatio pätee

  $$\text{“}\sim\text{”} = \{(x, y) \mid x \sim y\}$$

  $$x \sim y \Leftrightarrow (x, y) \in \text{“}\sim\text{”}$$

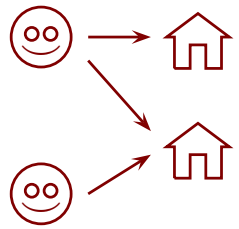
  - relaatiossa on kyse siitä, että $(A \times B)$:n alkiot jaetaan kahtia: parit, joille relaatio pätee ja parit, joille se ei päde
  - kahtiajaon voisi ilmaista joukko-opillisesti eri tavoilla
  - tämä on se tapa, jota yleisesti käytetään

$\Rightarrow$ päästään rakentamaan relaatioita relaatioista joukko-opin keinoilla

- esim. $\text{“}>\text{”} = \{(x, y) \mid y < x\}$
- esim. $\leq = < \cup =$
- oliko epäselvä? uusiksi: $\text{“}\leq\text{”} = \text{“}<\text{”} \cup \text{“}=\text{”}$

- filosofiassa erotetaan käyttö ja maininta
  - $x \sim y$ käyttää ja $(x, y) \in \text{“}\sim\text{”}$ mainitsee relaation “$\sim$”
  - `merkkijono = "lainausmerkki on \"";` käyttää, mainitsee ja käyttää ”"”:n
  - mihin avaruus loppuu? s-kirjaimeen
- relaation auki kirjoittaminen parien joukkona on tyypillisesti liian työlästä
  - esim. $\text{“}<\text{”} = \{(0,1), (0,2), (1,2), (0,3), (1,3), (2,3), (0,4), (1,4), (2,4), (3,4), \ldots\}$
- lähtö- ja maalijoukkoa ei ihan aina ole olemassa
  - esim. joukkojen ”$\subseteq$”:n kummallakin puolella pitäisi olla kaikkien joukkojen joukko, jota Russellin paradoksin vuoksi ei ole olemassa

Toisinaan relaatio on havainnollista esittää nuolina



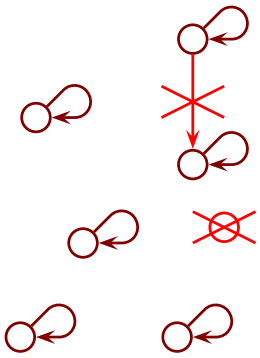
- esim. Isabelle omistaa yhden asunnon yksin ja toisen Clauden kanssa

Identiteettirelaatio

- *identiteettirelaatio* joukolle A on

$$\mathrm{id}_A \;=\; \{(x, x) \mid x \in A\}$$

- se on siis se, jota olemme tottuneet kutsumaan yhtäsuuruudeksi
- sitä merkitään mm. "$=$" ja "$\equiv$"
- identiteettirelaatiosta tarvitsee puhua erikseen, koska jatkossa tulee muitakin yhtäsuuruuden kaltaisia relaatioita
- "$=$" saattaa olla käytössä jollekin niistä
  $\Rightarrow$ tarvitaan toinen merkintä, esim. "$\equiv$"



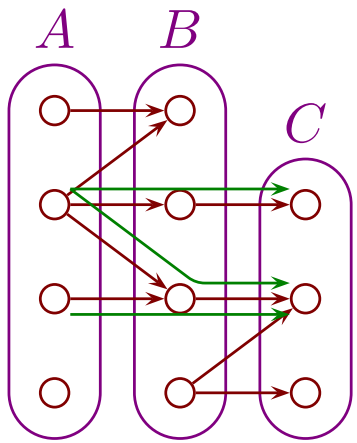
- tämä tilanne syntyy mm. kun "$=$" on ollut käytössä ja ilmenee tarve tehdä vielä tarkempia erotteluja
  - esim. jos $m \neq 0 \neq k$, niin $\frac{n}{m} = \frac{h}{k} \Leftrightarrow nk = mh$
  - millä merkinnällä ilmaistaan, että haluamme $n = h$ ja $m = k$, siis erottaa $\frac{2}{3}$ ja $\frac{4}{6}$?

Kuten jo näimme, parien joukkoihin voidaan soveltaa tuttuja joukko-operaatioita

- esim. luvuilla $\overline{\text{"}<\text{"}} = \text{"}\geq\text{"}$ ja $\text{"}<\text{"} \subseteq \text{"}\leq\text{"}$
- seuraavaksi tarkastelemme erityisen tärkeitä binäärirelaatioiden operaatioita

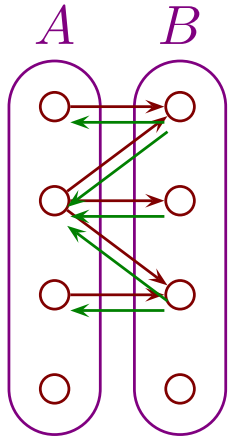
Binäärirelaatioiden yhdistäminen

- olkoon “$\frown$” $\subseteq A \times B$ ja “$\smile$” $\subseteq B \times C$
- niiden yhdistetty relaatio on “$\sim$” $= \{(a, c) \in A \times C \mid \exists b \in B : a \frown b \wedge b \smile c\}$
- toisin sanoen
  - “$\sim$” $\subseteq A \times C$
  - $a \sim c \;\Leftrightarrow\; \exists b : a \frown b \wedge b \smile c$
- toisin sanoen, “$\sim$” liittää joukon A alkion a joukon C alkioon c jos ja vain jos a:sta pääsee c:hen kahdella askeleella niin että välissä on jokin B:n alkio b
- esim. ”henkilön asunnon ikkuna” $\subseteq$ Henkilöt $\times$ Ikkunat
- esim. ”kissan omistajan serkku” $\subseteq$ Kissat $\times$ Ihmiset
- me ja jotkut merkitsemme “$\sim$” $=$ “$\smile$” $\circ$ “$\frown$”, toiset merkitsevät “$\sim$” $=$ “$\frown$” $\circ$ “$\smile$”
- “$\frown$” $\circ\, \mathrm{id}_A =$ “$\frown$” ja $\mathrm{id}_B \circ$ “$\frown$” $=$ “$\frown$”



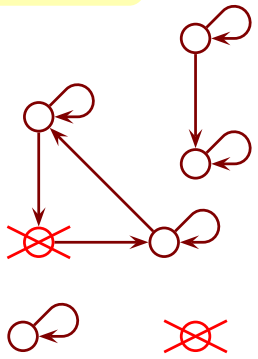
Binäärirelaation käänteisrelaatio

- relaatiota käytetään takaperin
- jos “$\sim$” $\subseteq A \times B$, niin “$\sim$”$^{-1} = \{(b, a) \in B \times A \mid a \sim b\}$
- toisin sanoen, $b \sim^{-1} a \;\Leftrightarrow\; a \sim b$
- esim. “$\leq$”$^{-1} =$ “$\geq$”
- esim. (Kissan omistaja)$^{-1}$ = Henkilön kissa



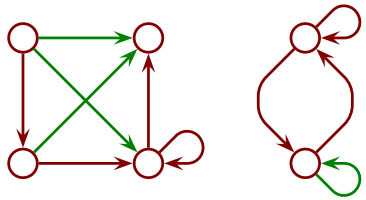
- jokainen binäärirelaatio on käänteisrelaationsa käänteisrelaatio
- relaation ja sen käänteisrelaation yhdistäminen ei välttämättä tuota identiteettiä
  - edellä A:n alin ei kuvaudu itselleen
  - edellä A:n toiseksi alin kuvautuu kahdelle alkiolle

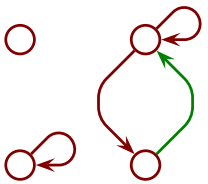
  $\Rightarrow$ nimitystä "käänteis" voidaan pitää huonona



Binäärirelaation refleksiivisyys, transitiivisuus ja symmetrisyys

- nyt oletetaan, että lähtöjoukko on maalijoukko $= A$
- "$\sim$" on *refleksiivinen*, jos ja vain jos $\forall x \in A : x \sim x$
  - toisin sanoen, "id_A" $\subseteq$ "$\sim$"
- "$\sim$" on *transitiivinen*, jos ja vain jos
  $\forall x \in A : \forall y \in A : \forall z \in A : x \sim y \wedge y \sim z \to x \sim z$
  - toisin sanoen, "$\sim$" $\circ$ "$\sim$" $\subseteq$ "$\sim$"
  - induktiolla seuraa $x_0 \sim x_1 \wedge \cdots \wedge x_{n-1} \sim x_n \to x_0 \sim x_n$
- "$\sim$" on *symmetrinen*, jos ja vain jos $\forall x \in A : \forall y \in A : x \sim y \to y \sim x$
  - toisin sanoen, "$\sim$" $=$ "$\sim$"$^{-1}$ (yhtäpitävästi "$\sim$" $\subseteq$ "$\sim$"$^{-1}$)





	$x < y$	$x \leq y$	$x = y$	$x \neq y$	$x + 1 \geq y$	$\lvert x - y \rvert < 1$	$x = x + 1$
refleksiivinen	–	$\surd$	$\surd$	–	$\surd$	$\surd$	–
transitiivinen	$\surd$	$\surd$	$\surd$	–	–	–	$\surd$
symmetrinen	–	–	$\surd$	$\surd$	–	$\surd$	$\surd$

Ekvivalenssirelaatio

- *ekvivalenssirelaatio* on mikä tahansa refleksiivinen transitiivinen symmetrinen relaatio
- tutuin esimerkki on identiteettirelaatio
- jokainen relaatio muotoa $x \simeq y \Leftrightarrow f(x) = f(y)$ on ekvivalenssi
  - samanvärinen
  - yhtä nopea
  - $x \bmod 7 = y \bmod 7$
- todistus: seuraavat pätevät kaikilla x, y ja z
  - $f(x) = f(x)$, joten $x \simeq x$
  - $x \simeq y \;\Rightarrow\; f(x) = f(y) \;\Rightarrow\; f(y) = f(x) \;\Rightarrow\; y \simeq x$
  - $x \simeq y \wedge y \simeq z \;\Rightarrow\; f(x) = f(y) \wedge f(y) = f(z) \;\Rightarrow\; f(x) = f(z) \;\Rightarrow\; x \simeq z$
- murtolukujen yhtäsuuruus
  - $n \in \mathbb{Z}$, $m \in \mathbb{Z}^+$, $h \in \mathbb{Z}$ ja $k \in \mathbb{Z}^+$
  - $\frac{n}{m} = \frac{h}{k} \Leftrightarrow nk = mh$
  - tässä siis ajatellaan, että $\frac{16}{56}$ ja $\frac{10}{35}$ ovat kaksi yhtäsuurta eri murtolukua
  - refleksiivisyys: $h = n$ ja $k = m$, joten vaaditaan $nm = mn$, täsmää
  - transitiivisuus: $\frac{n}{m} = \frac{h}{k} \wedge \frac{h}{k} = \frac{i}{j} \;\Rightarrow\; nk = mh \wedge hj = ki \;\Rightarrow$
    $nkj = mhj \wedge mhj = mki \;\Rightarrow\; nkj = mki \;\Rightarrow$
    (koska $k \neq 0$) $nj = mi \;\Rightarrow\; \frac{n}{m} = \frac{i}{j}$
  - symmetrisyys: $\frac{n}{m} = \frac{h}{k} \;\Rightarrow\; nk = mh \;\Rightarrow\; hm = kn \;\Rightarrow\; \frac{h}{k} = \frac{n}{m}$
- likimain yhtäsuuri "$\approx$" ei ole ekvivalenssi, koska se ei ole transitiivinen

Ekvivalenssiluokka

- alkion a ja ekvivalenssirelaation "$\simeq$" $\subseteq A^2$ määräämä *ekvivalenssiluokka* on

$$[\![a]\!] \;=\; \{x \in A \mid x \simeq a\}$$

- niiden alkioiden joukko, jotka ovat "samankaltaisia" a:n kanssa
- refleksiivisyys $\Rightarrow a \simeq a$

$\Rightarrow a \in [\![a]\!]$

$\Rightarrow [\![a]\!] \neq \emptyset$

- jos $b \in [\![a]\!]$, niin $b \simeq a$, joten symmetria $\Rightarrow a \simeq b$
- jos lisäksi $x \in [\![a]\!]$, niin $x \simeq a$, joten transitiivisuus $\Rightarrow x \simeq b$

$\Rightarrow x \in [\![b]\!]$

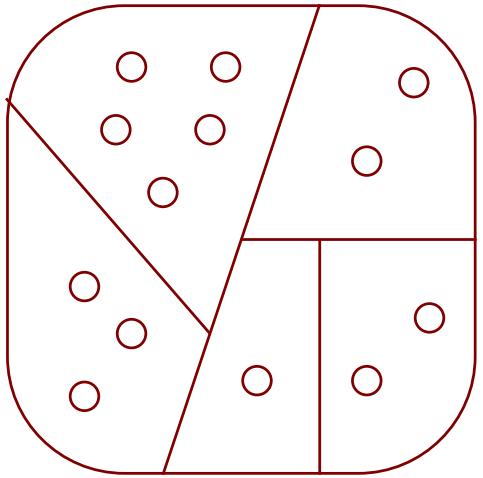
- jos $b \in [\![a]\!]$ ja $x \in [\![b]\!]$, niin $b \simeq a$ ja $x \simeq b$
- transitiivisuus $\Rightarrow x \simeq a$

$\Rightarrow x \in [\![a]\!]$

- osoitimme, että jos $b \in [\![a]\!]$, niin $[\![a]\!] = [\![b]\!]$

$\Rightarrow$ eri ekvivalenssiluokilla ei ole yhteisiä alkioita

$\Rightarrow$ ekvivalenssiluokat ryhmittelevät joukon siististi!



Joukon A *ositus*

- joukko joukkoja $\{A_1, \ldots, A_n\}$ siten, että
  - mikään A_i ei ole tyhjä
  - jokainen $a \in A$ kuuluu täsmälleen yhteen A_i
  - muuta niihin ei kuulukaan
- kaavamaisemmin:
  - $\forall i; 1 \leq i \leq n : A_i \neq \emptyset$
  - $A_1 \cup \cdots \cup A_n = A$
  - $\forall i : \forall j; 1 \leq i < j \leq n : A_i \cap A_j = \emptyset$
- tyhjällä joukolla on vain yksi ositus: $\emptyset$
- yhden alkion joukolla on vain yksi ositus: $\{\{a\}\}$
- kahden alkion joukolla on kaksi ositusta: $\{\{a\}, \{b\}\}$ ja $\{\{a, b\}\}$
- kolmen alkion joukolla on viisi ositusta:
  $\{\{a\}, \{b\}, \{c\}\}$, $\{\{a\}, \{b, c\}\}$, $\{\{b\}, \{a, c\}\}$, $\{\{a, b\}, \{c\}\}$ ja $\{\{a, b, c\}\}$
- A:n ositus B:n avulla tuottaa $\{A \setminus B, A \cap B\} \setminus \{\emptyset\}$
  - jos $A = \emptyset$, niin tulee $\{\emptyset \setminus B, \emptyset \cap B\} \setminus \{\emptyset\} = \{\emptyset, \emptyset\} \setminus \{\emptyset\} = \emptyset$
  - muutoin, jos $A \subseteq B$, niin tulee $\{\emptyset, A \cap B\} \setminus \{\emptyset\} = \{A \cap B\}$
  - muutoin, jos $A \cap B = \emptyset$, niin tulee $\{A \setminus B, \emptyset\} \setminus \{\emptyset\} = \{A \setminus B\}$
  - muutoin tulee $\{A \setminus B, A \cap B\}$

Ekvivalenssin ja osituksen suhde

- ekvivalenssiluokat muodostavat osituksen
  - ekvivalenssiluokat eivät ole tyhjiä, koska $a \in [\![a]\!]$
  - samasta syystä $A \subseteq \bigcup_{a \in A} [\![a]\!]$
  - ekvivalenssiluokat ovat A:n osajoukkoja, joten $\bigcup_{a \in A} [\![a]\!] \subseteq A$

  $\Rightarrow \bigcup_{a \in A} [\![a]\!] = A$
  - olemme jo osoittaneet, että eri ekvivalenssiluokilla ei ole yhteisiä alkioita
- osituksen joukot synnyttävät ekvivalenssin
  - $a \simeq b$ jos ja vain jos a ja b kuuluvat samaan osaan
  - on muotoa $a \simeq b \Leftrightarrow f(a) = f(b)$, joten on ekvivalenssi

$\Rightarrow$ ekvivalenssiluokat ja ositukset ovat sama asia ilmaistuna formaalisti eri keinoin

Kongruenssi

- olkoon sovittu perusjoukko A
- olkoon sovittu joukko $\mathcal{F}$ lasku- tai muita toimituksia
  - esim. $+$, $-$, $\cdot$ ja $/$
- selitämme idean binäärisillä laskutoimituksilla
  - se toimii myös muilla argumenttien määrillä

- A:n ekvivalenssi "$\simeq$" on *kongruenssi*, jos ja vain jos
  - kaikille $\mathcal{F}$:n laskutoimituksille $f(x, y)$ ja
  - kaikille A:n alkioille x_1, x_2, y_1 ja y_2

  pätee $x_1 \simeq x_2 \wedge y_1 \simeq y_2 \Rightarrow f(x_1, y_1) \simeq f(x_2, y_2)$
- $=_4$ ei ole kongruenssi jakolaskun suhteen
  - $2 =_4 6$, mutta $\frac{2}{2} = 1 \neq_4 3 = \frac{6}{2}$
- jos $M \in \mathbb{Z}^+$, niin $=_M$ on kongruenssi yhteen-, vähennys- ja kertolaskun suhteen
- jos M on alkuluku, niin $=_M$ on kongruenssi myös jakolaskun suhteen
- jos "$\simeq$" on kongruenssi joillekin laskutoimituksille, niin se on kongruenssi myös niiden yhdistelmille
  - olkoon $x_1 \simeq x_2 \wedge y_1 \simeq y_2 \wedge z_1 \simeq z_2 \wedge t_1 \simeq t_2$

  $\Rightarrow g(x_1, y_1) \simeq g(x_2, y_2) \wedge h(z_1, t_1) \simeq h(z_2, t_2)$

  $\Rightarrow f(g(x_1, y_1), h(z_1, t_1)) \simeq f(g(x_2, y_2), h(z_2, t_2))$
- ekvivalenssi ilman kongruenssiominaisuutta ei yleensä riitä samanveroisuudeksi
- äärimmäisen harvoin tarvitaan voimakkaampaa samanveroisuutta kuin kongruenssi
- kongruenssin käsite on tapauskohtainen
  - riippuu valituista lasku- tms. toimituksista

Muistathan, että ominaisuus pätee jos ja vain jos se pätee *jokaisessa* tapauksessa

- ominaisuus ei päde, jos sille on *yksikin* vastaesimerkki
- esim. "on äitinsä tytär" ei ole refleksiivinen (ei päde miehille)

Osittainjärjestykset

- A:n binäärirelaatio on *antisymmetrinen* jos ja vain jos
  $\forall x \in A : \forall y \in A : x \sim y \wedge y \sim x \rightarrow x = y$
  - kaikki alla mainitut (ja kaikki muutkin) osittainjärjestykset
  - $x = y \vee x = y + 1$
- osittainjärjestys on mikä tahansa refleksiivinen transitiivinen antisymmetrinen relaatio
  - identiteettirelaatio (kyllä, vaikka siinä mikään alkio ei ole toista pienempi!)
  - reaalilukujen "$\leq$" ja "$\geq$"
  - joukkojen "$\subseteq$"
  - suorakaiteiden "on korkeintaan yhtä pitkä ja korkeintaan yhtä leveä"
  - ihmisten "x on y:n jälkeläinen"
  - lounaiden "enintään yhtä kallis ja vähintään yhtä iso annos"
  - lukukolmikoiden vertailu komponenteittain:
    $(x, y, z) \leq (x', y', z') \Leftrightarrow x \leq x' \wedge y \leq y' \wedge z \leq z'$
- voi olla, että x ei ole pienempi, yhtäsuuri eikä suurempi kuin y
- osittainjärjestystä merkitään usein "$\preceq$"
- osittainjärjestyksen käänteisrelaatio on osittainjärjestys
  - vertailun suunta on vaihdettu
  - "$\preceq$":n käänteisrelaatiota merkitään usein "$\succeq$"

- usein merkitään $x \prec y \Leftrightarrow x \preceq y \wedge x \neq y$
  - "$\prec$" on transitiivinen
  - "$\prec$" on asymmetrinen: $\forall x : \forall y : \neg(x \prec y \wedge y \prec x)$

$\Rightarrow$ asymmetria on eri asia kuin antisymmetria
  - "$\sim$" on asymmetrinen jos ja vain jos se on antisymmetrinen ja $\forall x : x \not\sim x$

- "kivi–sakset–paperi" on asymmetrinen mutta ei transitiivinen

	kivi	sakset	paperi
kivi	–	$\surd$	–
sakset	–	–	$\surd$
paperi	$\surd$	–	–

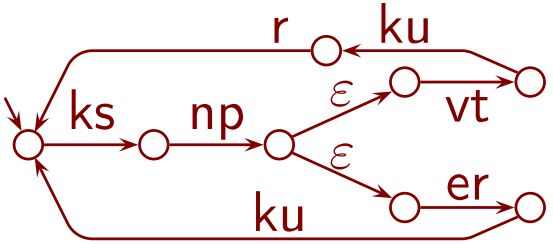
- transitiivisuus on se, joka antaa luvan päätellä "koska $x \sim y$ ja $y \sim z$, niin $x \sim z$"
  - $x \preceq y \wedge y \preceq z \Rightarrow x \preceq z$
  - $x \prec y \wedge y \prec z \Rightarrow x \prec z$
  - $x \preceq y \wedge y \prec z \Rightarrow x \prec z$
  - $x \prec y \wedge y \preceq z \Rightarrow x \prec z$

- lukukolmikoilla $(x, y, z) < (x', y', z')$ ei tarkoita $x < x' \wedge y < y' \wedge z < z'$ vaan $x \leq x' \wedge y \leq y' \wedge z \leq z' \wedge (x < x' \vee y < y' \vee z < z')$
  - tästä syystä "$\leq$" on kätevämpi kuin "$<$"

(Täydet) järjestykset

- osittainjärjestyksessä kullekin x ja y *enintään* yksi seuraavista pätee: $x = y$, $x \prec y$ ja $x \succ y$
  - voi olla, että x ei ole pienempi, yhtäsuuri eikä suurempi kuin y
- järjestyksessä eli täydessä järjestyksessä (total order) *täsmälleen* yksi niistä pätee
  - x on pienempi, yhtäsuuri tai suurempi kuin y
- reaalilukujen "$\leq$" ja "$\geq$"
- äärellisten merkkijonojen shortlex-järjestys
  - lyhyempi merkkijono on pienempi kuin pitempi
  - samanpituisten merkkijonojen järjestyksen ratkaisee ensimmäinen eroava merkki
  - $\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots$
- merkkijonojen leksikograafinen järjestys
  - verrataan vasemmalta alkaen kunnes löytyy eroava merkki tai jompikumpi loppuu
  - se, joka loppui ensin, on pienempi
  - se, jolla ensimmäinen eroava merkki on pienempi, on pienempi
  - $\varepsilon, a, aa, aaa, \ldots, aaab, aaaba, aaabaa, \ldots, aab, aaba, aabaa, \ldots$
  - puhelinluetteloissa, sanakirjoissa jne. käytettävä aakkosjärjestys (ainakin melkein)
- päiväysten järjestys $1.1.2000 < 2.1.2000 < \ldots < 9.1.2000 < 10.1.2000 < \ldots$
- lukukolmikoiden järjestys, jossa verrataan vasemmalta alkaen
  - $(x, y, z) < (x', y', z') \Leftrightarrow x < x' \vee x = x' \wedge y < y' \vee x = x' \wedge y = y' \wedge z < z'$
- lounaiden järjestys, jossa ensin verrataan hintaa ja jos hinta on sama niin annoskokoa

Muut kuin binäärirelaatiot



- kolmipaikkainen relaatio on kolmen joukon tulon osajoukko
  - esim. kaaret, joilla on nimet: $\Delta \subseteq V \times (\Sigma \cup \{\varepsilon\}) \times V$
- vastaavasti neli-, viisi- jne. paikkainen relaatio
- yksipaikkaisista relaatioista ei puhuta
  - olisi sama kuin osajoukko

5.3 Funktiot ja osittaiset funktiot

Funktio $f : A \mapsto B$ liittää jokaiseen A:n alkioon täsmälleen yhden B:n alkion

- esimerkkejä
  - $x^2 + 3x$
  - ympyrän pinta-ala
  - henkilön syntymävuosi
  - auton väri
- voidaan tulkita binäärirelaatioksi siten, että $\forall a \in A : \exists b \in B : (a, b) \in \text{“}f\text{”}$ ja $\forall a \in A : \forall b_1 \in B : \forall b_2 \in B : (a, b_1) \in \text{“}f\text{”} \wedge (a, b_2) \in \text{“}f\text{”} \rightarrow b_1 = b_2$
- A on lähtöjoukko ja B on maalijoukko
- kun a on valittu, niin sitä b, jolle $(a, b) \in \text{“}f\text{”}$, merkitään $f(a)$

Funktioiden yhdistäminen

- jos $f : A \mapsto B$ ja $g : B \mapsto C$, niin on olemassa $h : A \mapsto C$ siten, että $h(x) = g(f(x))$
- sama kuin relaatioiden yhdistäminen, tuottaa funktion
- esim. jos $f(x) = x + 1$ ja $g(x) = x^2$, niin $g(f(x)) = (x + 1)^2$ ja $f(g(x)) = x^2 + 1$

Funktioiden $A \mapsto B$ joukkoa merkitään B^A

- vrt. $B^3 = B \times B \times B = B^{\{1,2,3\}}$ ja 2^A voidaan tulkita $\{\text{false}, \text{true}\}^A$
- jos A ja B ovat äärellisiä, niin $|B^A| = |B|^{|A|}$

Injektio on funktio, joka ei saa samaa arvoa kahdesti

- $\forall x : \forall y : x \neq y \rightarrow f(x) \neq f(y)$
- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = x^3$ on injektio
- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = x^2$ ei ole injektio, koska $f(-1) = 1 = f(1)$
- $f : \mathbb{R}^+ \mapsto \mathbb{R} : f(x) = x^2$ on injektio
- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = 2^x$ on injektio
- $f :$ Henkilöt $\times$ Vuodet: $f(x) =$ henkilön x syntymävuosi ei ole injektio

Surjektio on funktio, joka saa jokaisen maalijoukon arvon ainakin kerran

- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = x^3$ on surjektio
- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = x^2$ ei ole surjektio, koska se ei saavuta arvoa -1
- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = 2^x$ ei ole surjektio, koska se ei saavuta arvoa 0
- $f : \mathbb{R} \mapsto \mathbb{R}^+ : f(x) = 2^x$ on surjektio

Bijektio on funktio, joka saa jokaisen maalijoukon arvon täsmälleen kerran

- yksi yhteen -vastaavuus
- samanaikaisesti injektio ja surjektio
- $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = x^3$ on bijektio
- $f : \mathbb{R} \mapsto \mathbb{R}^+ : f(x) = 2^x$ on bijektio

Bijektion käänteisrelaatio on funktio

- käänteisfunktio
- funktion $f : \mathbb{R} \mapsto \mathbb{R} : f(x) = x^3$ käänteisfunktio on $f^{-1} : \mathbb{R} \mapsto \mathbb{R} : f^{-1}(x) = \sqrt[3]{x}$
- funktion $f : \mathbb{R} \mapsto \mathbb{R}^+ : f(x) = 2^x$ käänteisf. on $f^{-1} : \mathbb{R}^+ \mapsto \mathbb{R} : f^{-1}(x) = \log_2(x)$
- jos $f : A \mapsto B$ on bijektio, niin
  - $f^{-1}(f(x))$ on se funktio $A \mapsto A$, jolle $f^{-1}(f(x)) = x$
  - f^{-1} on bijektio $B \mapsto A$

Monen argumentin funktiot

- on paljon funktioita, jotka ottavat monta argumenttia
  - "$+$" $: \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$
  - $\log_b : \mathbb{R}^+ \times \mathbb{R}^+ \mapsto \mathbb{R}$
- eri argumentit voivat olla peräisin eri joukoista
- merkitään $f : A_1 \times \cdots \times A_k \mapsto B$
- voidaan tulkita yksipaikkaiseksi funktioksi joukolta $A_1 \times \cdots \times A_k$ joukolle B
- voidaan tulkita myös funktioksi A_1:ltä funktioille $A_2 \times \cdots \times A_k \mapsto B$
  - jos $f : \mathbb{R}^+ \times \mathbb{R} \mapsto \mathbb{R}^+ : f(x, y) = x^y$, niin voidaan valita
    $g : \mathbb{R}^+ \mapsto \mathbb{R}^+ : g(x) = f(x, 2) = x^2$ ja
    $h : \mathbb{R}^+ \mapsto \mathbb{R}^+ : h(x) = f(x, \frac{1}{2}) = \sqrt{x}$
  - tämä tunnetaan nimellä currying

Osittainen funktio on muuten kuten funktio, mutta se liittää jokaiseen lähtöjoukon alkioon *enintään* yhden maalijoukon alkion

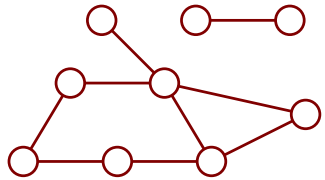
- esim. $\frac{1}{x}$ tuottaa tuloksen muille $x \in \mathbb{R}$ paitsi $x = 0$
- tietokoneohjelma laskee osittaisen funktion $\Sigma^* \mapsto \Sigma^*$
  - tietokoneohjelma ei välttämättä pysähdy kaikilla syötteillä

  $\Rightarrow$ osittainen funktio on tärkeä käsite laskettavuuden teoriassa

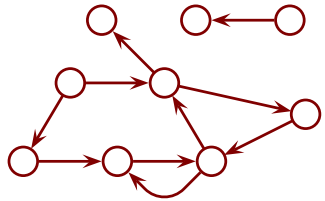
5.4 Graafit

Graafi (*graph*) on matemaattinen rakenne, jossa on kahdenlaisia osia

- *solmu* (*vertex*, *node*)
  - piirretään usein ympyröinä tai kiekkoina
- *kaari* (*edge*, *arc*)
  - piirretään usein viivana tai nuolena

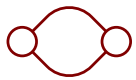


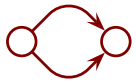
Graafit jakaantuvat kahteen ryhmään



- *suunnattu* (*directed*) graafi
  - kaari piirretään nuolena solmusta solmuun
- *suuntaamaton* (*undirected*) graafi
  - kaari piirretään viivana, joka yhdistää kaksi solmua

Monigraafi (*multigraph*) sallii samojen solmujen välillä (samaan suuntaan) monta kaarta





- useat yleiset esitystavat tietokoneessa sallivat moninkertaiset kaaret, jollei erikseen nähdä vaivaa niiden estämiseksi

$\Rightarrow$ monesti sanotaan graafi, vaikka esitystapa sallii monigraafin

Suunnatun graafin matemaattinen määritelmä

- solmujen joukkoa merkitään yleensä kirjaimella V
- kaarten joukkoa merkitään yleensä kirjaimella E
- suunnatussa graafissa vaaditaan, että $E \subseteq V \times V$
  - ts. kaari on kahden solmun järjestetty pari
  - $V \times V$ sisältää kaikki järjestetyt solmuparit
  - jokaisen solmuparin välillä joko on kaari tai ei ole
  - E kertoo, millä väleillä kaaret ovat
- jos $v \in V$, niin $(v, v) \in V \times V$
  $\Rightarrow$ solmusta voi olla kaari itseensä 

Suuntaamattoman graafin matemaattinen määritelmä

- V on kuten edellä
- E:n määritelmän tarkka sisältö ja muotoilu eivät ole vakiintuneet
  - joskus sallitaan kaari solmusta itseensä, joskus ei
- ei sallita: kaari on kahden solmun järjestämätön pari eli kahden solmun joukko
  - $E \subseteq \{V' \mid V' \subseteq V \wedge |V'| = 2\}$
- sallitaan: kaari on yhden tai kahden solmun joukko 
  - $E \subseteq \{\{u, v\} \mid u \in V \wedge v \in V\}$

Usein merkitään $u \to v$ tarkoittamaan $(u, v) \in E$

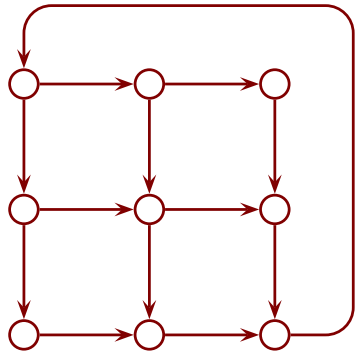
Polku

- polku on jono solmuja siten, että jokaisesta (paitsi viimeisestä) on kaari seuraavaan
  - $v_0 v_1 \cdots v_n$, missä $v_0 \in V \wedge \forall i; 1 \leq i \leq n : (v_{i-1}, v_i) \in E$ tai $\ldots \{v_{i-1}, v_i\} \in E$
  - myös yksi solmu muodostaa polun
  - polun pituus on sen kaarten määrä
- merkitään solmuille $u \rightarrow^* v$ jos ja vain jos u:sta on polku v:hen
  - myös nolla kaarta katsotaan poluksi

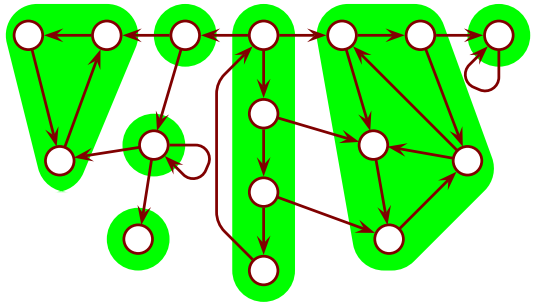
  $\Rightarrow$ jokaiselle solmulle v pätee $v \rightarrow^* v$

Silmukka

- silmukka on polku, jonka viimeisestä solmusta on kaari ensimmäiseen
  - $v_1 \cdots v_n$, missä $(v_n, v_1) \in E \wedge \forall i; 1 < i \leq n : (v_{i-1}, v_i) \in E$ tai $\ldots \{v_n, v_1\} \in E \ldots \{v_{i-1}, v_i\} \in E$
- jos graafissa on silmukka, siinä on äärettömästi silmukoita
  - jos $v_1 \cdots v_n$ on silmukka, niin myös $v_1 \cdots v_n v_1 \cdots v_n$, $v_1 \cdots v_n v_1 \cdots v_n v_1 \cdots v_n$ jne. ovat
- yksinkertainen silmukka on silmukka, jossa solmu toistuu vain lopussa
- usein graafissa on paljon yksinkertaisia silmukoita



Suunnatun graafin *vahva komponentti* (*strong component*)



- solmun v virittämä maksimaalinen vahvasti kytketty komponentti C_v on mahdollisimman iso kokoelma solmuja u siten, että $v \rightarrow^* u$ ja $u \rightarrow^* v$
  - $v \in C_v$
- usein käytetään lyhyempää nimeä "vahva komponentti" (strong component)
- kukin solmu kuuluu täsmälleen yhteen vahvaan komponenttiin
  - olkoot C_u ja C_v vahvoja komponetteja siten, että $w \in C_u$ ja $w \in C_v$
  - jos $u' \in C_u$, niin $u' \rightarrow^* u \rightarrow^* w \rightarrow^* v$ ja $v \rightarrow^* w \rightarrow^* u \rightarrow^* u'$

  $\Rightarrow u' \rightarrow^* v \wedge v \rightarrow^* u'$ eli $u' \in C_v$
  - samoin jos $v' \in C_v$, niin $v' \in C_u$

  $\Rightarrow C_u = C_v$
- vahva komponentti on käyttökelpoinen käsite silloinkin, kun silmukoita on liikaa lueteltaviksi

Isomorfismi

Bisimulaatiot ja bisimilaarisuus

6 Predikaattilogiikka

6.1 Syntaksi, semantiikka ja lakeja

Predikaattilogiikan kaava on muuten samanlainen kuin propositiologiikan, mutta

- propositioiden tilalla on monimutkaisempi käsite *predikaatti*
- kaavoissa voi käyttää *kvanttoreita* $\forall$ ja $\exists$

Predikaatti

- predikaattilogiikan kaava väittää jotain jostakin kohdemaailmasta
  - esim. reaaliluvut: $\forall x : x \geq 0 \rightarrow \exists y : y^2 = x$
  - esim. kokonaisluvut: $\forall n : \forall m : m \neq 0 \rightarrow \exists q : \exists r : n = qm + r \wedge 0 \leq r < |m|$
  - esim. merkkijonot: $\forall \alpha : \forall \beta : \text{takaperin}(\alpha\beta) = \text{takaperin}(\beta)\text{takaperin}(\alpha)$
  - esim. taulukot: $\forall i : 1 \leq i < n \rightarrow A[i] \leq A[i+1]$
- *termi* on kohdemaailman arvon tuottava lauseke
  - esim. kokonaisluvut: n, 0, qm, $|m|$ ja $(k+1)^2$
  - esim. merkkijonot: $\text{takaperin}(\beta)\text{takaperin}(\alpha)$
- k-paikkainen *predikaatti* ottaa k termiä ja tuottaa totuusarvon
  - esim. $=$, $>$ ja $\geq$ ovat 2-paikkaisia predikaatteja
  - propositiot ovat 0-paikkaisia predikaatteja
  - esim. kokonaisluvuilla $\text{parillinen}(n)$

- emme määrittele termeille ja predikaateille kielioppia, koska jokaisella kohdemaailmalla on omansa
- luettavuuden vuoksi käytämme sulkuja negatoiduissa vertailuissa, kuten $\neg(x > 0)$

Moni propositiologiikkaluvun esimerkki oli todellisuudessa predikaattilogiikan kaava

- esim. $x \geq 0 \wedge x \neq 0 \Leftrightarrow x > 0$ on havainnollisempi kuin $P \wedge Q \Leftrightarrow R$

Kvanttorit

- usein riippuu x:n arvosta, päteekö $\varphi(x)$
  - esim. $x^2 + x = 6$
  - ei kuitenkaan aina, esim. $x < 0 \wedge x > 0$ tai $1 = 2$
- $\forall x : \varphi(x)$ pätee jos ja vain jos $\varphi(x)$ pätee jokaisella x:n arvolla
  - universaalikvanttori
  - esim. $\forall x : x^2 \geq 0$ ja $\forall a : \forall b : a + b = b + a$ pätevät reaaliluvuilla
  - esim. $\forall x : x^2 > 0$ ei päde reaaliluvuilla
- $\exists x : \varphi(x)$ pätee jos ja vain jos $\varphi(x)$ pätee ainakin yhdellä x:n arvolla
  - eksistenssikvanttori
  - esim. $\exists x : x^2 > 0$ pätee reaaliluvuilla
  - esim. $\exists x : x^2 = -1$ ei päde reaaliluvuilla
- kirjallisuudessa kvanttorien syntaksi vaihtelee (myös välimerkit)

Kielioppi tällä kurssilla

$$
\begin{array}{rcl}
\text{Kaava} & ::= & \text{Kvanttorikaava} \\
\text{Kvanttorikaava} & ::= & \text{Ekvivalenssikaava} \\
 & & \mid \forall\ \text{Muuttujaosuus} : \text{Kvanttorikaava} \\
 & & \mid \exists\ \text{Muuttujaosuus} : \text{Kvanttorikaava} \\
\text{Muuttujaosuus} & ::= & \text{Muuttuja} \\
 & & \mid \text{Muuttuja} \in \text{Joukko} \\
 & & \mid \text{Muuttuja}\ ;\ \text{Rajain}
\end{array}
$$

- tapaukset Muuttuja $\in$ Joukko ja Muuttuja ; Rajain käsitellään s. 138

Jos kohdemaailmassa on vain äärellinen määrä eri arvoja $v_1, \ldots, v_n$, niin

- $\forall x : \varphi(x)$ tarkoittaa samaa kuin $\varphi(v_1) \wedge \cdots \wedge \varphi(v_n)$
- $\exists x : \varphi(x)$ tarkoittaa samaa kuin $\varphi(v_1) \vee \cdots \vee \varphi(v_n)$
- tämän avulla voi palauttaa mieleen monta kohta esitettävää lakia

Milloin kvanttorikaava tuottaa **U**?

- $\forall x : \varphi(x)$ tuottaa **U** jos ja vain jos
  - ainakin yhdellä x:n arvolla $\varphi(x)$ on määrittelemätön
  - muilla x:n arvoilla $\varphi(x)$ tuottaa **T**
- $\forall x : x \cdot \frac{1}{x} = 1$ tuottaa **U**
  - kun $x = 0$, on $x \cdot \frac{1}{x} = 1$ määrittelemätön, muulloin se tuottaa **T**

- $\exists x : \varphi(x)$ tuottaa **U** jos ja vain jos
  - ainakin yhdellä x:n arvolla $\varphi(x)$ on määrittelemätön
  - muilla x:n arvoilla $\varphi(x)$ tuottaa **F**
- $\exists x : \frac{1}{(x-3)^2} \leq 0$ tuottaa **U**
  - kun $x = 3$, on $\frac{1}{(x-3)^2} \leq 0$ määrittelemätön, muulloin se tuottaa **F**

$\forall x : \varphi(x)$ ja $\exists x : \varphi(x)$ ovat kasvavia $\varphi(x)$:n suhteen

- esim. jos $\varphi(x) \Rightarrow \psi(x)$ pätee jokaisella x, niin $\forall x : \varphi(x) \Rightarrow \forall x : \psi(x)$

$\Rightarrow \forall$ ja $\exists$ ei katsota negaatioiksi, kun tutkitaan, onko jokin parillisen vai parittoman negaatioiden määrän vaikutuspiirissä

Muuttujan vapaa ja sidottu esiintymä

- jokainen seuraavista väittää eri asiaa:
  - $\forall x : x > 0$ ei päde, koska ei toteudu esim. kun $x = 0$
  - $x > 0$ pätee tai ei, riippuen x:n arvosta
  - $\exists x : x > 0$ pätee, koska toteutuu esim. kun $x = 1$
- muuttujan x esiintymä on *sidottu*, jos ja vain jos se on jonkin $\forall x :$ tai $\exists x :$ vaikutuspiirissä
  - myös $\forall x$ ja $\exists x$ itse otetaan huomioon
- muut muuttujan x esiintymät ovat *vapaita*
- esim. $x = 0 \wedge (\exists x : x^2 = 2x) \vee x = 1$ sisältää kaksi vapaata ja kolme sidottua x:n esiintymää

- ohjelmoijalle on luontevaa ajatella, että vapaana esiintyvä muuttuja on eri muuttuja kuin sidottuna esiintyvä, vaikka sillä on sama nimi
  - myös eri kvantifioinneilla luodut muuttujat kannattaa ajatella eri muuttujiksi
  - esim. $(\exists x : (\forall x : x^2 \neq 2) \vee x^2 = 2) \wedge x \neq 1 \wedge (\forall x : x^2 > 0 \vee x = 0) \vee x = 1$ sisältää neljä eri muuttujaa, joiden nimi on x
  - mitä "Kauppakatu" tarkoittaa riippuu siitä, ollaanko Jyväskylässä vai Kuopiossa

Avoin kaava sisältää ja *suljettu* kaava ei sisällä ainakin yhden vapaan esiintymän

- esim.
  - $\forall x : x > 0$ on suljettu
  - $x > 0$ on avoin
  - $\exists x : x > 0$ on suljettu
  - $(\exists x : (\forall x : x^2 \neq 2) \vee x^2 = 2) \wedge (\forall x : x^2 \geq 0) \vee x = 1$ on avoin
- suljetulla kaavalla on yksikäsitteinen totuusarvo **F**, **U** tai **T**
  - tämä sillä oletuksella, että termien ja predikaattien merkitys on kiinnitetty
  - esim. reaaliluvuilla symbolien 0, -123, $+$ ja $\leq$ merkitys on kiinnitetty
- logiikkaa käytetään usein myös siten, että näin ei oleteta
  - esim. mikä on se kokonaislukujen predikaatti $n \mid m$, jolle pätee $n \mid m \Leftrightarrow \exists k : m = kn$?
  - esim. mikä on se positiivisten kokonaislukujen funktio $f(n, m)$, jolle pätee $f(n, m) \mid n \wedge f(n, m) \mid m \wedge \neg(\exists k : k > f(n, m) \wedge k \mid n \wedge k \mid m)$?

- avoimen kaavan totuusarvo voi riippua vapaana esiintyvien muuttujien arvoista
  - esim. $\sqrt{x} > 0$ tuottaa **U** kun $x < 0$, **F** kun $x = 0$ ja **T** kun $x > 0$
  - ei kuitenkaan välttämättä, esim. $x^2 > -1$
- varmista, että ymmärrät kaavojen $\varphi(x)$, $\forall x : \varphi(x)$ ja $\exists x : \varphi(x)$ eron

Termin sijoittaminen muuttujan vapaiden esiintymien paikalle

- jos x on muuttuja, t on termi ja $\varphi(x)$ on kaava, niin $\varphi(t)$ tarkoittaa kaavaa, joka saadaan korvaamalla x:n jokainen vapaa esiintymä $\varphi(x)$:ssä t:llä
- korvauksen tulee tapahtua lausekepuun, ei tekstin tasolla
  $\Rightarrow$ voi olla tarpeen lisätä sulkuja
  - esim. x:n korvaaminen $n + 1$:llä $x^2 \geq 0$:ssa: ei $n + 1^2 \geq 0$ vaan $(n + 1)^2 \geq 0$
- termissä ei saa olla muuttujia, jotka ovat sidottuja yhdessäkin korvauskohdassa
  - esim. jokaisella $x \in \mathbb{R}$ pätee $\exists y : y > x$
  - x:n korvaaminen siinä $y + 1$:llä tuottaisi $\exists y : y > y + 1$, joka ei päde
  - tämä ongelma ratkeaa kohta "sidotun muuttujan vaihdolla"
- esimerkkejä

$\varphi(x)$	t	$\varphi(t)$
$x(y + z) = xy + xz$	$a - 3$	$(a - 3)(y + z) = (a - 3)y + (a - 3)z$
$x \geq 0 \to \exists y : y^2 = x$	$z^2 + 1$	$z^2 + 1 \geq 0 \to \exists y : y^2 = z^2 + 1$
$x \geq 0 \to \exists y : y^2 = x$	$y^2 + 1$	$y^2 + 1 \geq 0 \to \exists y : y^2 = y^2 + 1$

Predikaattilogiikan lakeja

- de Morganin lait

$$\neg\forall x : \varphi(x) \;\Leftrightarrow\; \exists x : \neg\varphi(x)$$
$$\neg\exists x : \varphi(x) \;\Leftrightarrow\; \forall x : \neg\varphi(x)$$

- samanlaisten kvanttorien vaihto

$$\forall x : \forall y : \varphi(x,y) \;\Leftrightarrow\; \forall y : \forall x : \varphi(x,y) \qquad \exists x : \exists y : \varphi(x,y) \;\Leftrightarrow\; \exists y : \exists x : \varphi(x,y)$$

- erilaisten kvanttorien vaihto toimii vain toiseen suuntaan

$$\exists x : \forall y : \varphi(x,y) \;\Rightarrow\; \forall y : \exists x : \varphi(x,y)$$

  - toisinpäin ei päde, esim. $\forall y : \exists x : x \neq y$ mutta ei $\exists x : \forall y : x \neq y$

- kvantifioinnin jakaminen kahdeksi, huomaa jälkimmäisten suunnat!

$$\forall x : \varphi(x) \wedge \psi(x) \;\Leftrightarrow\; (\forall x : \varphi(x)) \wedge \forall x : \psi(x)$$
$$\exists x : \varphi(x) \vee \psi(x) \;\Leftrightarrow\; (\exists x : \varphi(x)) \vee \exists x : \psi(x)$$
$$\forall x : \varphi(x) \vee \psi(x) \;\Leftarrow\; (\forall x : \varphi(x)) \vee \forall x : \psi(x)$$
$$\exists x : \varphi(x) \wedge \psi(x) \;\Rightarrow\; (\exists x : \varphi(x)) \wedge \exists x : \psi(x)$$

- jos x ei esiinny vapaana φ:ssä, niin

$$\varphi \wedge \forall x : \psi(x) \;\Leftrightarrow\; \forall x : \varphi \wedge \psi(x) \qquad \varphi \wedge \exists x : \psi(x) \;\Leftrightarrow\; \exists x : \varphi \wedge \psi(x)$$
$$\varphi \vee \forall x : \psi(x) \;\Leftrightarrow\; \forall x : \varphi \vee \psi(x) \qquad \varphi \vee \exists x : \psi(x) \;\Leftrightarrow\; \exists x : \varphi \vee \psi(x)$$

  - ilman ehtoa voitaisiin erehtyä esim. $x < 1 \wedge \forall x : x^2 \geq 0 \overset{?}{\Leftrightarrow} \forall x : x < 1 \wedge x^2 \geq 0$
  - $(x < 1)$:n x tarkoittaa eri muuttujia virheellisen $\Leftrightarrow$:n eri puolilla

$\Rightarrow$ moneen lakiin liittyy ehtoja, jotka estävät eri muuttujia muuttumasta samaksi
  - kirjallisuudessa ehdot vaikuttavat sekavilta
  - valitettavasti ne vaikuttavat sekavilta myös näissä luennoissa ☹
  - niiden kanssa pärjää, kun ajattelee, että eri muuttujilla voi olla sama nimi, ja pitää huolta, että nimen esiintymä viittaa aina oikeaan muuttujaan
  - usein auttaa *sidotun muuttujan vaihto*, mutta sekin on toki tehtävä oikein!

- sidotun muuttujan vaihto: jos y ei esiinny $\varphi(x)$:ssä, niin

$$\forall x : \varphi(x) \;\Leftrightarrow\; \forall y : \varphi(y) \qquad \exists x : \varphi(x) \;\Leftrightarrow\; \exists y : \varphi(y)$$

  - esim. $\exists x : x > y$ mutta ei $\exists y : y > y$
  - esim. $\forall x : \exists y : x = y + 1$ mutta ei $\forall y : \exists y : y = y + 1$
  - yleistys: y saa esiintyä sidottuna $\varphi(x)$:ssä, kunhan x ei esiinny vapaana siellä missä y on sidottu

- jos $t_1 = t_2$ ja mikään t_1:ssä tai t_2:ssa esiintyvä muuttuja ei ole sidottu x:n kohdalla $\varphi(x)$:ssä, niin $\varphi(t_1) \Leftrightarrow \varphi(t_2)$
  - vaikka olisi päätelty $x = 1$, niin ei saa päätellä $\exists x : x^2 = 9 \Leftrightarrow \exists x : 1^2 = 9$

Avoimet kaavat päättelyssä

- $\varphi(x) \Rightarrow \psi(x)$ on pätevä jos ja vain jos niissä huomioon otettavissa tilanteissa, joissa $\varphi(x)$ pätee, myös $\psi(x)$ pätee
  - usein vain osa käsitteellisesti mahdollisista tilanteista otetaan huomioon, ks. s. 77
- esim. $x > 1 \Rightarrow x > 0$
- esim. käsiteltäessä tapausta $c \geq 2$: $x > 1 \Rightarrow cx > 2$

Kvanttorien luonti- ja eliminointilait

- universaalikvanttorin eliminointi: jos mikään termissä t esiintyvä muuttuja ei ole sidottu x:n kohdalla $\varphi(x)$:ssä, niin

$$\lfloor \varphi(t) \rceil \wedge \forall x : \varphi(x) \;\Rightarrow\; \varphi(t)$$

  - jos lisäksi t on määritelty kaikilla sisältämiensä muuttujien arvoyhdistelmillä, niin

$$\forall x : \varphi(x) \;\Rightarrow\; \varphi(t)$$

- eksistenssikvanttorin luonti: jos mikään termissä t esiintyvä muuttuja ei ole sidottu x:n kohdalla $\varphi(x)$:ssä, niin

$$\varphi(t) \;\Rightarrow\; \exists x : \varphi(x)$$

  - $\forall$-eliminoinnista saadaan $\varphi(t) \wedge \lfloor \varphi(t) \rceil \;\Rightarrow\; \exists x : \varphi(x)$, mutta se tarkoittaa samaa

- universaalikvanttorin luonti: jos x:stä ei ole oletettu mitään, niin

$$\varphi(x) \;\Rightarrow\; \forall x : \varphi(x)$$

  - x ei saa esiintyä vapaana sen päättelyn lähtökohdissa, jolla $\varphi(x)$ johdettiin

- eksistenssikvanttorin eliminointi: jos c ei esiinny aiemmin, $\varphi(x)$:ssä eikä ψ:ssä, niin

$$\text{jos } \varphi(c) \wedge \lfloor \varphi(c) \rceil \;\Rightarrow\; \psi \text{ niin } \exists x : \varphi(x) \;\Rightarrow\; \psi$$

  - usein ensin päätellään $\exists x : \varphi(x)$ ja sitten johdetaan $(\varphi(c) \wedge \lfloor \varphi(c) \rceil)$:stä ψ
  - c edustaa arvoa, jonka $\exists x : \varphi(x)$ lupaa olevan olemassa
  - jos $\varphi(x)$:n muut vapaat muuttujat kuin x ovat $x_1, \ldots, x_n$, niin c voi riippua niistä; tätä voi korostaa merkinnällä $c(x_1, \ldots, x_n)$ tai $c_{x_1,\ldots,x_n}$

Esimerkki pätevästä päättelystä:

- $\exists x : \forall y : \varphi(x, y)$ lähtökohta
  $\forall y : \varphi(c, y)$ $\exists$:n eliminointi alkaa pätevästi
  $\Rightarrow\ \varphi(c, z)$ pätevä $\forall$:n eliminointi, eikä z:sta oleteta mitään
  $\Rightarrow\ \exists x : \varphi(x, z)$ pätevä $\exists$:n luonti, $t = c$
  $\Rightarrow\ \forall z : \exists x : \varphi(x, z)$ pätevä $\forall$:n luonti, koska z ei ole vapaa $\forall y : \varphi(c, y)$:ssä
  $\Rightarrow\ \forall y : \exists x : \varphi(x, y)$ pätevä sidotun muuttujan vaihto
  $\Rightarrow\ \forall y : \exists x : \varphi(x, y)$ $\exists$:n eliminointi päättyy pätevästi, koska c ei esiinny tuloksessa
- todistimme, että $\exists x : \forall y : \varphi(x, y) \ \Rightarrow\ \forall y : \exists x : \varphi(x, y)$

Esimerkki virheellisestä päättelystä:

- $\forall x : \exists y : y > x$ tosi lähtökohta
  $\Rightarrow\ \forall z : \exists y : y > z$ pätevä sidotun muuttujan vaihto
  $\Rightarrow\ \exists y : y > x$ pätevä $\forall$:n eliminointi
  $c > x$ $\exists$:n eliminointi alkaa pätevästi, luotu c voi riippua x:sta
  $\Rightarrow\ \forall x : c > x$ virheellinen $\forall$:n luonti, koska oletettiin, että $x < c$
  $\Rightarrow\ \exists y : \forall x : y > x$ pätevä $\exists$:n luonti, mutta virheellisestä välituloksesta
- loppu uusiksi merkinnällä $c(x)$: alla $c(x)$-kaavat pätevät, jos esim. $c(x) = x + 1$
  $\Rightarrow\ \exists y : y > x$ tähän asti kuten edellä
  $c(x) > x$ $\exists$:n eliminointi alkaa pätevästi, c:n riippuvuus x:stä näkyy
  $\Rightarrow\ \forall x : c(x) > x$ pätevä $\forall$:n luonti, koska c:n riippuvuus x:stä näkyy
  $\Rightarrow\ \exists y : \forall x : y > x$ virheellinen $\exists$:n luonti sijoituksella $t = c(x)$, x on sidottu

Kuinka nämä lait voi muistaa?

- en tiedä, minä en muista niitä kaikkia!
- aika moni on ilmeinen, kun miettii symboleiden merkitystä
- esim. $\neg\forall x : \varphi(x) \;\Rightarrow\; \exists x : \neg\varphi(x)$
  - jos $\varphi(x)$ ei päde jokaisella x, niin on olemassa jokin x jolla $\varphi(x)$ ei päde
- esim. $\exists x : \forall y : \varphi(x, y) \;\Rightarrow\; \forall y : \exists x : \varphi(x, y)$
  - se x:n arvo, joka kelpaa vasemmalla, kelpaa oikeallakin
  - laki ei toimi toisinpäin, koska vaikka jokaisella y olisi sopiva x, *sama* x ei välttämättä kelpaa jokaiselle y
  - esim. $\forall y : \exists x : x = y$ pätee mutta $\exists x : \forall y : x = y$ ei päde

Tavoitteena *ei* ole oppia päättelemään yksityiskohtaisesti kuten sivun 134 esimerkissä

- jätämme sellaisen päättelemisen koneille!
- tavoitteena on oppia päättelemään ihmisille luontevalla tavalla
  - päättelyaskeleet saavat olla pitempiä kuin yksi lain soveltaminen
  - lähtökohtia, välituloksia ja lopputuloksia saa ilmaista (myös) suomeksi
  - päättelemisen oppimiseksi kurssilla on paljon päättelyesimerkkejä
- asioita suomeksi ilmaistessa tulee käyttää hyvin täsmällisiä ilmaisuja
  - esim. tyypillinen virhe on jättää epäselväksi, edustaako uusi symboli yhtä (kuten $\exists x$) vai jokaista (kuten $\forall x$) arvoa
  - aina ei ole helppoa sanoa suomeksi, esim. $P \to Q \to P \wedge Q$

- tarvitseeko sellaista kertoa, minkä voi olettaa olevan lukijalle selvää kertomattakin?
  - yleensä ei
  - opettaja voi kuitenkin käskeä kertomaan sellaista varmistuakseen, että se on kertojalle itselleen selvää!
  - on helppoa luulla, että jokin on lukijalle selvää vaikkei todellisuudessa olekaan
- päättelyn pitää perustua määritelmiin, luvattuihin lähtökohtiin sekä yleisesti tunnettuihin tosiasioihin ja päättelysääntöihin
  - esim. ristiriitatodistuksessa otetaan lähtökohdaksi tavoitteen negaatio
- matemaattisen todistuksen käsite on sosiaalinen
  - (nykynäkemyksen mukaan) todistuksen pitää aina olla ainakin periaatteessa palautettavissa pikkutarkoiksi, koneellisesti tarkastettaviksi askeliksi
  - sellaisesta palauttamisesta tulisi usein valtava määrä tylsää rutiinia
  - usein on itsestään selvää, että se onnistuisi, jos yritettäisiin

  $\Rightarrow$ ei ole järkevää käyttää aikaa moiseen

Kohdemaailman vakiot vs. muuttujat

- matematiikassa yleensä
  - laissa $x > 0 \wedge y > 0 \;\Rightarrow\; \ln xy = \ln x + \ln y$ x ja y ovat muuttujia (ne edustavat mitä tahansa lukuja)
  - laissa $\ln e = 1$ e on yksi nimenomainen luku, likimain 2,718

- predikaattilogiikassa vakio on muuttuja, jonka arvo tiedetään tarkalleen
  - erillistä vakion käsitettä ei ole

  $\Rightarrow$ jokainen kirjain edustaa muuttujaa
  - esim. e on muuttuja, jonka arvosta tiedetään, että $\ln e = 1$
  - vrt. $\exists e : \ln e = 1 \wedge \ldots$

Rajoitettu kvanttori, tapaus 1

Muuttujaosuus ::= Muuttuja | Muuttuja $\in$ Joukko | Muuttuja ; Rajain

- $\forall x \in A : \varphi(x)$ tarkoittaa "jokaiselle joukkoon A kuuluvalle x:lle pätee $\varphi(x)$"
  - $\forall x \in A : \varphi(x) \;\Leftrightarrow\; \forall x : x \in A \rightarrow \varphi(x)$
- $\exists x \in A : \varphi(x)$ tarkoittaa "jollekin joukkoon A kuuluvalle x:lle pätee $\varphi(x)$"
  - $\exists x \in A : \varphi(x) \;\Leftrightarrow\; \exists x : x \in A \wedge \varphi(x)$
- näissä A on yleensä jokin yksinkertaisesti ilmaistavissa oleva joukko, esim. $\mathbb{R}$ tai Σ^*
  - siis A:ta käytetään usein kuten tyyppimääreitä ohjelmoinnissa, vrt. `int n`
  - emme määrittele tarkkaa syntaksia
- Σ^* on kaikkien Σ:n alkioista muodostettavissa olevien äärellisten jonojen joukko
  - esim. jos $\Sigma = \{a, b\}$, niin $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

Rajoitettu kvanttori, tapaus 2

- Rajain on tarkoitettu rajoittamaan taulukon indeksit sallitulle alueelle
  - esim. $\forall i; 1 \leq i < n : A[i] \leq A[i+1]$
  - harvinainen tämän kurssin ulkopuolella
  - senkään syntaksia ei ole tarkoin määritelty
- $\forall x; \xi(x) : \varphi(x) \;\Leftrightarrow\; \forall x : \xi(x) \to \varphi(x)$
- $\exists x; \xi(x) : \varphi(x) \;\Leftrightarrow\; \exists x : \xi(x) \wedge \varphi(x)$
- $\forall x; \xi(x) : \varphi(x)$ voi aiheuttaa yllätyksen, kun $\xi(x)$ tuottaa false jokaisella x
  - jokainen tyhjää väliä koskeva sellainen väite tuottaa true
  - $\forall x : \varphi(x)$ ei voi koskea tyhjää joukkoa (jollei esim. jako tapauksiin ole tuottanut sellaisen)
- esim. $\forall x : \varphi(x) \Rightarrow \exists x : \varphi(x)$, muttei välttämättä $\forall x; \xi(x) : \varphi(x) \Rightarrow \exists x; \xi(x) : \varphi(x)$
  - $\forall x; \xi(x) : \varphi(x) \Leftrightarrow \forall x : \xi(x) \to \varphi(x) \Rightarrow \exists x : \xi(x) \to \varphi(x) \Leftrightarrow \exists x : \neg\xi(x) \vee \varphi(x)$
- esim. $A[i]$ on pienempi kuin mikään muu taulukon $A[1...n]$ alkio, missä i on laillinen indeksi
  - $1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n \wedge i \neq j : A[i] < A[j]$ sanoo sen oikein
  - kun $n = 0$, se tuottaa false, kuten pitääkin
  - j ei esiinny vapaana osakaavassa $1 \leq i \leq n$
  - $\varphi \wedge \forall x : \psi(x) \;\Leftrightarrow\; \forall x : \varphi \wedge \psi(x)$, jos x ei esiinny vapaana φ:ssä
  - $\forall j; 1 \leq j \leq n \wedge i \neq j : 1 \leq i \leq n \wedge A[i] < A[j]$ tuottaa true, kun $n = 0$

- vastaavasti $\exists x; \xi(x) : \varphi(x)$ voi tuottaa yllätyksen, kun $\xi(x)$ tuottaa false jokaisella x
  - jos x ei esiinny vapaana φ:ssä, niin $\varphi \vee \exists x : \psi(x) \Leftrightarrow \exists x : \varphi \vee \psi(x)$, mutta ei välttämättä $\varphi \vee \exists x; \xi(x) : \psi(x) \Leftrightarrow \exists x; \xi(x) : \varphi \vee \psi(x)$
- tapauksissa $\forall x; \xi(x) : \varphi(x)$ ja $\exists x; \xi(x) : \varphi(x)$ päteviä lakeja voi johtaa em. määritelmien avulla
  - esim. $\exists x; \xi(x) : \varphi(x) \vee \psi(x)$
    $\Leftrightarrow \exists x : \xi(x) \wedge (\varphi(x) \vee \psi(x))$
    $\Leftrightarrow \exists x : \xi(x) \wedge \varphi(x) \vee \xi(x) \wedge \psi(x)$
    $\Leftrightarrow (\exists x : \xi(x) \wedge \varphi(x)) \vee \exists x : \xi(x) \wedge \psi(x)$
    $\Leftrightarrow (\exists x; \xi(x) : \varphi(x)) \vee \exists x; \xi(x) : \psi(x)$
  - siis $\exists x; \xi(x) : \varphi(x) \vee \psi(x) \Leftrightarrow (\exists x; \xi(x) : \varphi(x)) \vee \exists x; \xi(x) : \psi(x)$

Esimerkki: kaksi tapaa sanoa, että taulukko $A[1 \ldots n]$ on kasvavassa järjestyksessä

- $\forall i; 1 \leq i < n : A[i] \leq A[i+1]$ ja $\forall i : \forall j; 1 \leq i < j \leq n : A[i] \leq A[j]$
- perustelemme, että jälkimmäisestä seuraa edellinen

  $\forall i : \forall j; 1 \leq i < j \leq n : A[i] \leq A[j]$ valitaan $j = i + 1$

  $\Rightarrow \forall i; 1 \leq i < i+1 \leq n : A[i] \leq A[i+1]$ $i < n \Leftrightarrow i < i + 1 \leq n$

  $\Rightarrow \forall i; 1 \leq i < n : A[i] \leq A[i+1]$
- perustelemme, että edellisestä seuraa jälkimmäinen
  - oletetaan $\forall i; 1 \leq i < n : A[i] \leq A[i+1]$
  - valitaan mielivaltaiset i' ja j' siten, että $1 \leq i' < j' \leq n$

  $\Rightarrow j' = i' + k$ jollekin $k \in \mathbb{Z}^+$
  - valitaan mielivaltainen h siten, että $i' \leq h < j'$

  $\Rightarrow 1 \leq h < n$
  - valitsemalla h lähtökohdan i:ksi saadaan $A[h] \leq A[h+1]$

  $\Rightarrow A[i'] \leq A[i'+1] \leq A[i'+2] \leq \cdots \leq A[j']$

  $\Rightarrow A[i'] \leq A[j']$
  - siis $\forall i' : \forall j'; 1 \leq i' < j' \leq n : A[i'] \leq A[j']$

$\Rightarrow$ tavat ovat loogisesti yhtäpitävät

Lisää taulukosta $A[1 \ldots n]$ puhuvia esimerkkejä

- jokainen taulukossa esiintyvä arvo esiintyy ainakin kahdesti
  $\forall i; 1 \leq i \leq n : \exists j; 1 \leq j \leq n \wedge j \neq i : A[i] = A[j]$
- monimutkainen tapa sanoa true
  $\forall i; 1 \leq i \leq n : \exists j; 1 \leq j \leq n : A[i] = A[j]$
- monimutkainen tapa sanoa, että taulukko on tyhjä
  $\forall i; 1 \leq i \leq n : \exists j; 1 \leq j \leq n : A[i] < A[j]$

Lisää taulukosta $A[1 \ldots n]$ puhuvia esimerkkejä

- taulukon pienin alkio esiintyy (ainakin) kohdassa i, joka on laillisella alueella
  $1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n : A[i] \leq A[j]$
  - i on parametri sanallisessa ilmauksessa

  $\Rightarrow$ kaavan totuusarvon tulee todennäköisesti riippua i:stä

  $\Rightarrow$ i:n tulee todennäköisesti esiintyä vapaana
  - i rajataan lailliselle alueelle, jotta kaava ei koskaa tuottaisi **U**
- taulukon pienin alkio esiintyy (vain) kohdassa i, joka on laillisella alueella
  $1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n : i = j \vee A[i] < A[j]$
- monimutkainen tapa sanoa false
  $1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n : A[i] < A[j]$

Verrataanpa vielä nämä:

- taulukon pienin alkio esiintyy (ainakin) kohdassa i, joka on laillisella alueella
  $$1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n : A[i] \leq A[j]$$
- monimutkainen tapa sanoa false
  $\forall i : 1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n : A[i] \leq A[j]$
- monimutkainen tapa sanoa, että taulukon kaikki alkiot ovat yhtäsuuret
  $\forall i; 1 \leq i \leq n : \forall j; 1 \leq j \leq n : A[i] \leq A[j]$
- monimutkainen tapa sanoa, että taulukko ei ole tyhjä
  $\exists i : 1 \leq i \leq n \wedge \forall j; 1 \leq j \leq n : A[i] \leq A[j]$
- sama hieman toisin muotoiltuna
  $\exists i; 1 \leq i \leq n : \forall j; 1 \leq j \leq n : A[i] \leq A[j]$

6.2 Äärellinen, ääretön, (yli)numeroituva ja rekursiivinen

Äärettömien asioiden äärelliset esitykset

- luonnollisten lukujen joukon luettelemiseksi kokonaan ei aika riitä
  - käytetään kolmea pistettä: $\{0, 1, 2, \ldots\}$
- silti jokaisella luonnollisella luvulla on esitys äärellisenä numerojonona
  - esim. 56, $28\,244$, $989\,765\,429$
- siis
  - luonnollisten lukujen joukko on ääretön
  - jokainen luonnollinen luku on äärellinen
  - jokaisella luonnollisella luvulla on esitys äärellisenä merkkijonona
- ∞ tarkoittaa ääretöntä
  $\Rightarrow$ äärettömällä on esitys äärellisenä merkkijonona
  (toki tämä esitys on vain pelkkä erisnimi)
  - jopa esitys tavallisilla merkeillä, esim. LaTeX: `\infty`
  - silti ääretön ei ole äärellinen
  - ääretön ei ole luonnollinen luku (eikä reaaliluku)
- $\mathbb{N}$ tarkoittaa luonnollisten lukujen joukkoa
  $\Rightarrow$ luonnollisten lukujen joukolla on esitys äärellisenä merkkijonona
  - jopa esitys tavallisilla merkeillä: `\mathbb{N}`
  - toki tämäkin esitys on vain pelkkä erisnimi

- luonnollisten lukujen esitysten joukko esitettynä äärellisesti BNF:llä

$$\begin{aligned} \mathit{Luonn\_luku} &::= \text{"0"} \mid \mathit{Muu\_nro}\ \mathit{Nro\_jono} \\ \mathit{Muu\_nro} &::= \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \text{"4"} \mid \text{"5"} \mid \text{"6"} \mid \text{"7"} \mid \text{"8"} \mid \text{"9"} \\ \mathit{Nro\_jono} &::= \varepsilon \mid \text{"0"}\ \mathit{Nro\_jono} \mid \mathit{Muu\_nro}\ \mathit{Nro\_jono} \end{aligned}$$

- monilla muillakin äärettömillä joukoilla on esitys äärellisenä merkkijonona
  - tapaukset, joissa joukolla on erisnimi: $\mathbb{R}$, $\mathbb{Q}^+$, ...
  - esim. parilliset luvut: $\{n \in \mathbb{Z} \mid \exists k \in \mathbb{Z} : n = 2k\}$
  - esim. BNF: $\mathit{Sulutus} ::= \text{"("}\ \text{")"} \mid \text{"("}\ \mathit{Sulutus}\ \text{")"} \mid \mathit{Sulutus}\ \mathit{Sulutus}$

Jono matemaattisena oliona numeroituvuuden teoriassa

- *jono* on luettelo, joka alkaa mutta ei välttämättä pääty, eikä sisällä äärettömiä osajonoja muualla kuin lopussa
  - esim. Suomi, Ruotsi, Norja, Tanska, Islanti
  - esim. $0, 1, 2, \ldots$
  - ei esim. $-1, -2, -3, \ldots, 0, 1, 2, 3, \ldots$
- jonon alkiot voidaan numeroida ykkösestä alkaen ...

0	-1	1	-2	2	-3	3	$\ldots$
1	2	3	4	5	6	7	$\ldots$

- ... tai nollasta alkaen

0	-1	1	-2	2	-3	3	$\ldots$
0	1	2	3	4	5	6	$\ldots$

$\Rightarrow$ päättymätön jono on funktio joukolta $\mathbb{Z}^+$ tai $\mathbb{N}$ joukolle, josta jonon alkiot on poimittu

- päättyvä jono on funktio joukolta $\{1, \ldots, p\}$ tai $\{0, \ldots, p-1\}$, missä p on jonon pituus, joukolle, josta jonon alkiot on poimittu

Numeroituvuus

- joukko on *numeroituva*, jos ja vain jos sen alkiot voidaan asettaa jonoon
  - alkio saa esiintyä jonossa useasti
  - jokaisen alkion täytyy esiintyä jonossa ainakin kerran
- luonnolliset luvut: $0, 1, 2, \ldots$
- kokonaisluvut: $0, -1, 1, -2, 2, \ldots$
- positiiviset rationaaliluvut: $\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{2}{2}, \frac{1}{3}, \frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}, \ldots$
- kaikki rationaaliluvut: $0, -\frac{1}{1}, \frac{1}{1}, -\frac{2}{1}, \frac{2}{1}, -\frac{1}{2}, \frac{1}{2}, -\frac{3}{1}, \frac{3}{1}, -\frac{2}{2}, \frac{2}{2}, \ldots$
- äärellisestä aakkostosta muodostetut äärelliset merkkijonot: esim. $\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \ldots$
- jokainen äärellinen joukko on numeroituva
  - jono päättyy
  - esim. Suomi, Ruotsi, Norja, Tanska, Islanti
- joukko on *ylinumeroituva* jos ja vain jos se ei ole numeroituva

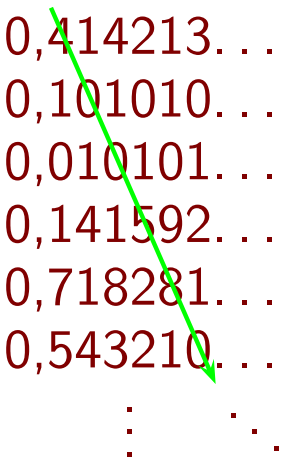
Reaalilukujen joukko on ylinumeroituva

- tämä voi tuntua yllättävältä, sillä
  - rationaalilukujen joukko on numeroituva
  - jokaisen kahden eri reaaliluvun välissä on rationaaliluku
    $\forall a \in \mathbb{R} : \forall b \in \mathbb{R} : a < b \rightarrow \exists q \in \mathbb{Q} : a < q < r$
- todistamme tämän neljällä eri tavalla

Todistus johtamalla ristiriita

- oletamme, että kaikkien $0 \leq x < 1$ desimaaliesitykset on laitettu jonoon
  - ne x, joilla on kaksi desimaaliesitystä, esitetään muodossa joka loppuu $000\ldots$
- muodostamme lävistäjää kulkemalla uuden desimaaliluvun seuraavasti:
  - luku aloitetaan 0,
  - jos lävistäjällä on desimaali 0, laitamme lukuun desimaalin 1
  - muutoin laitamme lukuun desimaalin 0

  $\Rightarrow$ saadaan luku, joka ei ole jonossa
- esim. 0,011001...



$\Rightarrow$ edes välin $0 \leq x < 1$ reaalilukuja ei voi laittaa jonoon

- tarvitsi varoa tapauksia, joissa kaksi eri desimaaliesitystä tarkoittaa samaa lukua
  - esim. 0,4999... = 0,5000...
  - muita tällaisia tapauksia ei ole kuin 999...-loppuiset ja 000...-loppuiset
- tämä todistusmenetelmä on keksijänsä kunniaksi nimeltään Cantorin diagonalisointi

Cantorin ensimmäinen todistus

- tarkastellaan mielivaltaista jonoa reaalilukuja
- valitaan jonon ensimmäinen luku luvuksi a_1
- edetään jonossa, kunnes tulee suurempi luku kuin a_1
  - jos sellaista ei tule, jono ei sisällä reaalilukua $a_1 + 1$
  - muutoin valitaan jonosta tullut luku luvuksi y_1
  - tähän mennessä ei ole tullut lukua väliltä $a_1 < x < y_1$
- edetään jonossa, kunnes tulee luku väliltä $a_1 < x < y_1$
  - jos sellaista ei tule, jono ei sisällä reaalilukua $\frac{a_1 + y_1}{2}$
  - muutoin valitaan jonosta tullut luku luvuksi a_2
  - tähän mennessä ei ole tullut lukua väliltä $a_2 < x < y_1$
- edetään jonossa, kunnes tulee luku väliltä $a_2 < x < y_1$
  - jos sellaista ei tule, jono ei sisällä reaalilukua $\frac{a_2 + y_1}{2}$
  - muutoin valitaan jonosta tullut luku luvuksi y_2
  - tähän mennessä ei ole tullut lukua väliltä $a_2 < x < y_2$

- ...
- edetään jonossa, kunnes tulee luku väliltä $a_i < x < y_i$
  - jos sellaista ei tule, jono ei sisällä reaalilukua $\frac{a_i + y_i}{2}$
  - muutoin valitaan jonosta tullut luku luvuksi a_{i+1}
  - tähän mennessä ei ole tullut lukua väliltä $a_{i+1} < x < y_i$
- edetään jonossa, kunnes tulee luku väliltä $a_{i+1} < x < y_i$
  - jos sellaista ei tule, jono ei sisällä reaalilukua $\frac{a_{i+1} + y_i}{2}$
  - muutoin valitaan jonosta tullut luku luvuksi y_{i+1}
  - tähän mennessä ei ole tullut lukua väliltä $a_{i+1} < x < y_{i+1}$
- jos tätä ei voida jatkaa loputtomasti, jono ei sisällä kaikkia reaalilukuja
- muussa tapauksessa
  - $a_1 < a_2 < a_3 < \ldots$
  - $y_1 > y_2 > y_3 > \ldots$
  - jos $i \leq j$, niin $y_i \geq y_j > a_j$, muutoin $j < i$ ja $a_j < a_i < y_i$, siis aina $a_j < y_i$

$\Rightarrow$ jonosta puuttuu lukujonon $a_1, a_2, a_3, \ldots$ raja-arvo
  - se on olemassa, koska $a_1, a_2, a_3, \ldots$ on kasvava ja ylhäältä rajoitettu
  - se on suurempi kuin a_i, koska se on vähintään a_{i+1}
  - se on pienempi kuin y_i, koska se on enintään y_{i+1}

  $\Rightarrow$ se jää jonojen $a_1, a_2, a_3, \ldots$ ja $y_1, y_2, y_3, \ldots$ väliseen aukkoon

$\Rightarrow$ mikään jono ei sisällä kaikkia reaalilukuja

Cantorin joukko-opillinen todistus

- joukon *potenssijoukko* on sen kaikkien osajoukkojen joukko

$$2^A = \{X \mid X \subseteq A\}$$

  - esim. $2^{\{1,2,3\}} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
  - esim. $2^\emptyset = \{\emptyset\} \neq \emptyset$
- olkoon $f : A \mapsto 2^A$
  - siis jos $x \in A$, niin $f(x) \subseteq A$
  - esim. $A = \{1,2,3\}$, $f(1) = \{2\}$, $f(2) = \emptyset$ ja $f(3) = \{1,3\}$
- Cantor todisti tyrmäävän yksinkertaisella tavalla, että ei voi olla, että jokaiselle $X \subseteq A$ on olemassa $x \in A$ siten, että $X = f(x)$
- olkoon $R = \{x \in A \mid x \notin f(x)\}$
  - esim. edellä $R = \{1,2\}$, koska $1 \notin \{2\} = f(1)$, $2 \notin \emptyset = f(2)$ ja $3 \in \{1,3\} = f(3)$
- siis $x \in R \Leftrightarrow x \notin f(x)$
- jos $R = f(x)$, niin $x \notin f(x) \Leftrightarrow x \notin R$, joten $x \in R \Leftrightarrow x \notin R$

$\Rightarrow$ joukon potenssijoukossa on aina enemmän alkioita kuin joukossa itsessään
  - äärettömien joukkojen kokoja ei voi verrata laskemalla alkiot
  $\Rightarrow$ $A \leq B$ tarkoittaa, että on olemassa $f : B \mapsto A$ siten, että $\forall a \in A : \exists b \in B : a = f(b)$
- erityisesti ei ole olemassa $f : \mathbb{N} \mapsto 2^\mathbb{N}$ siten, että se saa arvokseen jokaisen $X \subseteq \mathbb{N}$

Kuten edellä todettiin, jonot voidaan ajatella funktioiksi $f : \mathbb{N} \mapsto Y$

- $f(0)$, $f(1)$, $f(2)$, ...

$\Rightarrow$ joukkoa $2^{\mathbb{N}}$ ei voi asettaa jonoon
  - puuttumaan jää ainakin $\{n \in \mathbb{N} \mid n \notin f(n)\}$

Tästä seuraa, että $\mathbb{R}$:ää ei voi asettaa jonoon

- jokaiselle luonnollisten lukujen osajoukolle A saadaan ikioma desimaaliluku asettamalla $d_{i+1} = 0$, jos $i \in A$ ja $d_{i+1} = 1$, jos $i \notin A$
  - esim. $\{1, 2, 3, 5\} \mapsto 0{,}011101000\ldots$
  - esim. $\{0, 2, 4, 6, \ldots\} \mapsto 0{,}1010101\ldots$
- eri joukoille tulee eri luvut
  - se, että $0{,}999\ldots = 1{,}000\ldots$ ei haittaa, koska konstruktio ei tuota numeroa 9
- useimmat reaaliluvut jäävät ilman omaa joukkoa, mutta se ei haittaa

$\Rightarrow$ jos $\mathbb{R}$:n voisi asettaa jonoon, niin poistamalla ne luvut, joilla ei ole omaa joukkoa, jäljelle jäisi $2^{\mathbb{N}}$:ää vastaava jono

$\Rightarrow$ joukkoa $\mathbb{R}$ ei voi asettaa jonoon

Jokainen numeroituva reaalilukujen osajoukko voidaan peittää miten lyhyellä tikulla tahansa

- olkoot osajoukon luvut x_1, x_2, x_3, ...
- otetaan tikku, jonka pituus on > 0

- katkaistaan se keskeltä, ja asetetaan toinen puolikas peittämään x_0
- katkaistaan jäljelle jäänyt tikunpätkä keskeltä,
  ja asetetaan toinen puolikas peittämään x_1
- katkaistaan jäljelle jäänyt tikunpätkä keskeltä,
  ja asetetaan toinen puolikas peittämään x_2
- näin voidaan jatkaa loputtomasti

$\Rightarrow$ miten lyhyt alkuperäinen tikku tahansa riittää peittämään kaikki jonon luvut

$\Rightarrow$ vaikka rationaaliluvut näyttävät peittävän lukusuoran
kokonaan, todellisuudessa ne peittävät mitättömän vähän

$\Rightarrow$ reaalilukuja on paljon enemmän kuin rationaalilukuja

Näimme edellä, että äärellisten merkkijonojen joukko on numeroituva

$\Rightarrow$ ei ole olemassa esitysjärjestelmää, joka esittäisi
jokaisen reaaliluvun äärellisenä merkkijonona

- siis reaalilukuja on niin paljon, että jokaiselle ei riitä omaa nimeä!
- siksi tietokoneissa on pakko käyttää likiarvoja, toisin kuin kokonaisluvuille
- asiaa voi ajatella myös seuraavasti
  - yhdessä reaaliluvussa voi olla ääretön määrä informaatiota, ja useimmissa onkin
  - yhdessä kokonaisluvussa on vain äärellinen määrä informaatiota
    (tosin ei ole ylärajaa sille, kuinka suuri tämä määrä voi olla)

Tietokoneissa ei voi käsitellä äärettömiä joukkoja sellaisinaan, mutta voi käsitellä

- niiden alkioita niin isoihin saakka kuin muistia riittää
  - esim. äärelliset merkkijonot (käsittely voi olla vaikka sanojen tavutus)
  - esim. luonnolliset luvut rajoittamattoman lukualueen kirjastolla
- niiden äärellisiä esitystapoja
  - esim. voidaan tehdä ohjelma, joka ottaa merkkijonon α ja BNF-kuvauksen B ja vastaa oikein kysymykseen "kuuluuko α B:n määrittelemään kieleen"
- alkion, BNF-kuvauksen tms. koolla on käytännössä yläraja, koska aikaa ja muistia ei ole rajattomasti
- jos aikaa ja muistia olisi rajattomasti, voitaisiin käsitellä tarkasti miten isoja alkioita ja BNF-kuvauksia tahansa
- vaikka aikaa ja muistia olisi rajattomasti, ei voitaisi käsitellä mielivaltaista reaalilukua tarkasti
  - jokaiselle luonnolliselle luvulle on esitys äärellisenä merkkijonona
  - jokaiselle BNF-kuvaukselle on esitys äärellisenä merkkijonona
  - jokaiselle reaaliluvulle ei riitä esitystä äärellisenä merkkijonona

  $\Rightarrow$ reaaliluvun esittäminen vaatii äärettömästi bittejä, paitsi poikkeustapauksissa, joita on mitättömän pieni osuus reaaliluvuista

Jatkossa oletamme, että aika ja muisti eivät lopu kesken

- emme oleta, että niitä on äärettömästi
- oletamme vain, että niitä on ainakin niin paljon kuin ohjelma tarvitsee
- jos ohjelma pysähtyy, jokin äärellinen määrä kumpaakin riittää!
  - nyt emme piittaa siitä, onko tarvittu määrä epärealistisen suuri

Mitä äärettömän joukon käsitteleminen tietokoneella tarkoittaa?

- käsitteellisesti yksinkertaisin tehtävä: esittääkö annettu merkkijono ennalta määrätyn joukon alkiota
  - esim. onko annettu merkkijono alkuluvun esitys numerojonona 27644437
  - esim. onko annettu merkkijono syntaktisesti virheetön kaava $x < 0 \vee \neg x^2 = x)$
  - esim. onko annettu merkkijono syntaktisesti virheetön kaava, joka pitää paikkansa kaikilla vapaiden muuttujien arvoyhdistelmillä $\forall x : \exists y : y > x$
- esimerkkejä joukkojen esityksiä käsittelevistä kyllä/ei-tehtävistä
  - kuuluuko annettu merkkijono annetun BNF-kuvauksen määrittelemään kieleen
  - onko annetun BNF-kuvauksen määrittelemässä kielessä yhtään merkkijonoa
  - ovatko kahden annetun BNF-kuvauksen määrittelemät kielet samat
- esimerkkejä muista kuin kyllä/ei-tehtävistä
  - laske annetun, luvuista ja aritmeettisista operaattoreista rakennetun merkkijonona esitetyn lausekkeen arvo $1 + 2(3 - 4)^5$
  - lisää annettuun merkkijonoon tavuviivat suomenkielen tavutussääntöjen mukaisesti li-sää an-net-tuun merk-ki-jo-noon ...

Voidaanko operaation tulos esittää sillä esitystavalla, jolla syötteet on annettu?

- helppo esimerkki: muodosta BNF-kuvaus B, jonka määrittelemä kieli on annettujen BNF-kuvausten B_1 ja B_2 määrittelemien kielten unioni
  - lisätään sääntö $B ::= B_1 \mid B_2$
- mahdoton esimerkki: muodosta BNF-kuvaus B, jonka määrittelemä kieli on annettujen BNF-kuvausten B_1 ja B_2 määrittelemien kielten leikkaus
  - $A ::= \varepsilon \mid "a"\ A$ $\{\varepsilon, a, aa, aaa, \ldots\}$
  - $C ::= \varepsilon \mid "c"\ C$ $\{\varepsilon, c, cc, ccc, \ldots\}$
  - $X ::= \varepsilon \mid "a"\ X\ "b"$ $\{\varepsilon, ab, aabb, aaabbb, \ldots\}$
  - $Y ::= \varepsilon \mid "b"\ Y\ "c"$ $\{\varepsilon, bc, bbcc, bbbccc, \ldots\}$
  - XC:n kieli sisältää täsmälleen ne merkkijonot, joissa on ensin jokin määrä a:ta, sitten sama määrä b:tä ja lopuksi jokin määrä c:tä
  - AY:n kieli sisältää täsmälleen ne merkkijonot, joissa on ensin jokin määrä a:ta, sitten jokin määrä b:tä ja lopuksi sama määrä c:tä
  - XC ja AY voidaan määritellä BNF:llä
  - niiden leikkaus sisältää täsmälleen ne merkkijonot, joissa on ensin jokin määrä a:ta, sitten sama määrä b:tä ja lopuksi sama määrä c:tä
  - voidaan todistaa, että sitä ei voi määritellä BNF:llä
- koska 2^Σ on ylinumeroituva ja BNF-kuvausten joukko on numeroituva, on selvää, että kaikille kielille ei riitä BNF-kuvausta

- sen sijaan ei ollut ennakolta selvää, että kaikille BNF-kielistä joukko-opin perusoperaatioilla muodostettaville kielille ei ole BNF-kuvausta
  - joukko-opin lausekkeet ovat äärellisiä merkkijonoja

  $\Rightarrow$ sellaisia kieliä on vain numeroituva määrä
  - ns. säännöllisillä lausekkeilla määriteltyjen kielten unioneille, leikkauksille ja joukkoerotuksille on kuvaukset säännöllisinä lausekkeina
- analogia lukujen maailmasta
  - tarkastellaan reaalilukuja, jotka voidaan esittää tarkasti desimaalilukuina
  - niiden summat, erotukset ja tulot ovat sellaisia reaalilukuja
  - osamäärät ovat sellaisia reaalilukuja vain tietyillä jakajilla

Palaamme takaisin merkkijonoja koskeviin kyllä/ei-kysymyksiin

- syöte on aina pohjimmiltaan merkkijono
- monimutkaisempien tehtävien vastaukset voidaan (ainakin periaatteessa) rakentaa kyllä/ei-vastauksista vaikka bitti kerrallaan (jos niille on esitystapa)

$\Rightarrow$ merkkijonoja koskevien kyllä/ei-kysymysten teoria kertoo hyvin paljon siitä, mitä tietokoneilla voi ja ei voi tehdä

- kyllä/ei-kysymyksissä ei voi käydä niin, että oikealle vastaukselle ei ole esitystapaa
- onkohan olemassa hyvin määriteltyjä kyllä/ei-kysymyksiä, joihin tietokone ei aina pysty vastaamaan, vaikka oletettaisiin, että muistia ja aikaa riittää?

Tietokoneohjelmia käsittelevät tietokoneohjelmat

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

$\Rightarrow$ voi olla mielekästä antaa ohjelmalle syötteeksi ohjelma

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

Miljardin suoritusaskeleen pysähtymistesteri

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

totuusarvo pysähtyy_miljardi(merkkijono x, merkkijono y)

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

  $\Rightarrow$ voiko vastauksen saada nopeammin?

Tarkastellaan seuraavaa ohjelmaa

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

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

  $\Rightarrow$ vastaus on väärä

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

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

  $\Rightarrow$ vastaus false on oikein

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

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

Miljoonan tavun pysähtymistesteri

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

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

  $\Rightarrow$ ei todellakaan onnistu käytännössä lähitulevaisuudenkaan tietokoneilla

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

Pysähtymistesteri

totuusarvo pysähtyy(merkkijono x, merkkijono y)

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

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

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

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

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

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

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

Merkkijonojen tuottaminen syötteettömällä pysähtyvällä ohjelmalla

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

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

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

$\Rightarrow$ ne tulostavat vain äärellisen määrän merkkijonoja

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

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

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

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

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

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

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

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

Todistus sille, että syötteettömien ohjelmien pysähtymistesteriä ei voida tehdä

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

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

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

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

Rekursiiviset funktiot ja joukot

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

Rekursiivisesti lueteltavat joukot

- joukko on *rekursiivisesti lueteltava* (*recursively enumerable*), jos ja vain jos on olemassa ohjelma, joka
  - pysähtyy lopulta, jos syöte on joukon alkio
  - laskee ikuisesti, jos syöte ei ole joukon alkio
- jos A on syötteiden osajoukko, niin A:n *komplementti* on $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$
  - syöte kuuluu $\overline{A}$:han jos ja vain jos se ei kuulu A:han
- A on rekursiivinen jos ja vain jos sekä A että $\overline{A}$ ovat rekursiivisesti lueteltavia
  - jos A on rekursiivinen, niin voidaan tehdä ohjelma, joka selvittää kuuluuko syöte A:han ja vastauksen perusteella pysähtyy tai aloittaa ikuisen silmukan
  - jos A ja $\overline{A}$ ovat rekursiivisesti lueteltavia, niin voidaan tehdä ohjelma, joka ajaa rinnakkain A:n ja $\overline{A}$:n ohjelmia kunnes jompikumpi pysähtyy
  - sitten se pysäyttää toisenkin ja vastaa sen mukaan, kumpi pysähtyi itsekseen
- $\{x \mid \text{syötteetön\_pysähtyy}(x)\}$ on rekursiivisesti lueteltava
  - jos x ei ole ohjelma tai ei ole syötteetön, aloitetaan ikuinen silmukka
  - muussa tapauksessa ajetaan x
- $\overline{\{x \mid \text{syötteetön\_pysähtyy}(x)\}}$ ei ole rekursiivisesti lueteltava
  - muutoin $\{x \mid \text{syötteetön\_pysähtyy}(x)\}$ olisi rekursiivinen

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

- joukko on rekursiivisesti lueteltava, jos ja vain jos on olemassa ohjelma, joka tulostaa sen alkiot eikä muuta
  - alkiot erotetaan toisistaan tulostuksessa esim. pilkulla
  - tulostus voi olla päättymätön

  $\Rightarrow$ kuten jonon käsite numeroituvuudessa
  - esim. seuraava ohjelma tulostaa parittomat kokonaisluvut:
    $i := 1$; **while** true **do** print i ", " $-i$ ", "; $i := i + 2$
- jos on olemassa tulostava ohjelma, testaava ohjelma saadaan seuraavasti:
  - aja tulostavaa ohjelmaa, kunnes se lopettaa tai tulostaa tutkittavan alkion
  - jos se lopetti, hyppää ikuiseen silmukkaan
  - jos se tulosti tutkittavan alkion, pysäytä se ja lopeta
  - jos se ei lopeta eikä tulosta tutkittavaa alkiota, suoritus jatkuu ikuisesti, kuten pitääkin
- jos on olemassa testaava ohjelma, tulostava ohjelma saadaan seuraavasti:
  - muodosta vuoronperään kaikki merkkijonot kasvavassa pituusjärjestyksessä
  - käynnistä kullekin niistä testi (monta testiä ajaa rinnakkain)
  - aina kun testi pysähtyy, tulosta testattu merkkijono
- koska testit valmistuvat mikä milloinkin, alkiot tulostuvat sekalaisessa järjestyksessä
- rekursiivisesti lueteltavan mutta ei rekursiivisen joukon alkiot voi tulostaa, mutta niitä ei voi tulostaa merkkijonojen mukaisessa kasvavassa järjestyksessä!
  - jos voisi, $x \in A$ ratkeaisi odottamalla kunnes tulostuu tutkittava tai suurempi

  $\Rightarrow$ A olisi rekursiivinen

Mikä on tällaisten tulosten käytännön merkitys?

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

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

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

6.3 1. kertaluvun teoriat

Miten aloitetaan teorian rakentaminen uudelle asialle?

- esim. on suunniteltu uusi, erikoinen tapa organisoida geenitietoa syöpätutkimusta varten ja toteutettu se monimutkaisena tietorakenteena
- halutaan varmistaa, että tietorakenne toimii oikein
- miten kerrotaan tarkastimelle, mitä tietorakenteen pitäisi milloinkin antaa ulos?

Mikä tässä on ongelma?

- toimiiko seuraava algoritmi oikein?

Hoida_homma(a_1, y_1)
if $a_1 \geq y_1$ **then return**
$x := A[\,\text{random}(a_1, y_1)\,].x$; $a := a_1 - 1$; $y := y_1 + 1$
while $a < y$ **do**
    $a := a + 1$; $y := y - 1$
    while $A[a].x < x$ **do** $a := a + 1$
    while $A[y].x > x$ **do** $y := y - 1$
    if $a < y$ **then** $apu := A[a]$; $A[a] := A[y]$; $A[y] := apu$
if $a = a_1$ **then** $a := a + 1$
Hoida_homma$(a_1, a - 1)$; Hoida_homma(a, y_1)

$\Rightarrow$ täytyy tietää koodia yksinkertaisemmalla tavalla, mitä tarkoittaa "oikein"!

Esimerkki: kumpi näistä on jono ja kumpi pino?

- jonosta alkiot tulevat ulos samassa järjestyksessä kuin menivät sisään
- pinosta tulee ensimmäisenä ulos se, joka on ollut siellä vähiten aikaa
- A on alkiot ja X on tietorakenteen tilat
- toimintojen nimikirjoitus (signature) on molemmilla sama

$\bot_X \in X$	tietorakenteen virhetila
$\bot_A \in A$	virhealkio
$\varepsilon \in X$	`const X tyhjä_X`
$i \in X \times A \to X$	`void X.lisää_alkio( Alkio )`
$d \in X \to X$	`void X.poista_alkio()`
$g \in X \to A$	`alkio X.anna_alkio()` (ei muuta X:n tilaa)

- eri olennot tarkoittavat eri asioita

  $\forall x \in X : \forall a \in A : a \neq x$ (toisin sanoen, $X \cap A = \emptyset$)

  $\varepsilon \neq \bot_X$
- virhetila tai virhealkio lähtökohtana tuottaa virhetilan tai virhealkion tuloksena

  $\forall a \in A : i(\bot_X, a) = \bot_X$

  $\forall x \in X : i(x, \bot_A) = \bot_X$

  $d(\bot_X) = \bot_X$

  $g(\bot_X) = \bot_A$

- tyhjästä rakenteesta ei voida poistaa eikä antaa alkiota
  $d(\varepsilon) = \bot_X$
  $g(\varepsilon) = \bot_A$
- muut operaatiot onnistuvat
  $\forall x \in X \setminus \{\bot_X\} : \forall a \in A \setminus \{\bot_A\} : i(x, a) \neq \bot_X$
  $\forall x \in X \setminus \{\varepsilon, \bot_X\} : d(x) \neq \bot_X$
  $\forall x \in X \setminus \{\varepsilon, \bot_X\} : g(x) \neq \bot_A$
- muut lainalaisuudet, vaihtoehto 1
  $\forall x \in X \setminus \{\bot_X\} : \forall a \in A \setminus \{\bot_A\} : d(i(x, a)) = x$
  $\forall x \in X \setminus \{\bot_X\} : \forall a \in A \setminus \{\bot_A\} : g(i(x, a)) = a$
- muut lainalaisuudet, vaihtoehto 2
  $\forall a \in A \setminus \{\bot_A\} : d(i(\varepsilon, a)) = \varepsilon$
  $\forall x \in X \setminus \{\varepsilon, \bot_X\} : \forall a \in A \setminus \{\bot_A\} : d(i(x, a)) = i(d(x), a)$
  $\forall a \in A \setminus \{\bot_A\} : g(i(\varepsilon, a)) = a$
  $\forall x \in X \setminus \{\varepsilon, \bot_X\} : \forall a \in A \setminus \{\bot_A\} : g(i(x, a)) = g(x)$
- käyttö aloitetaan tyhjästä tietorakenteesta, olennaista on mitä kulloinkin tulee ulos

$\Rightarrow$ kiinnostavat lausekkeet ovat $g(\ldots(\varepsilon)\ldots)$, missä $\ldots$ on jono i- ja d-kutsuja

- oletamme jatkossa $a_1 \neq \bot_A$, $\ldots$, $a_n \neq \bot_A$
  – siis oletamme, että tietorakenteeseen lisätään ehjiä alkioita, ei virhealkiota

$\Rightarrow$ on helppo osoittaa $i(\ldots i(\varepsilon, a_1)\ldots, a_n) \neq \bot_X$
- $\varepsilon \neq \bot_X \;\Rightarrow\; i(\varepsilon, a_1) \neq \bot_X \;\Rightarrow\; i(i(\varepsilon, a_1), a_2) \neq \bot_X \;\Rightarrow\; i(i(i(\varepsilon, a_1), a_2), a_3) \neq \bot_X \;\Rightarrow\; \ldots$

- sisimmän d-kutsun saa yllä olevilla kaavoilla pois
  - $d(\varepsilon) = d(\bot_X) = \bot_X$
  - jos $x = i(\ldots i(\varepsilon, a_1)\ldots, a_n)$ ja $A \in A \setminus \{\bot_A\}$, niin $d(i(x, a))$ on x tai $i(d(x), a)$

  $\Rightarrow$ d voidaan painaa sisäänpäin kunnes se katoaa

$\Rightarrow$ jokaisen d-kutsun saa yllä olevilla kaavoilla pois
- kun sisin on poistettu, poistetaan toiseksi sisin jne.
- jäljelle jää $g(i(\ldots i(\varepsilon, a_1)\ldots, a_n))$

- vaihtoehdossa 1 saadaan heti $\bot_A$ tai a_n

  $\Rightarrow$ vaihtoehto 1 on pino

- vaihtoehdossa 2 saadaan heti $\bot_A$ tai n askeleella a_1

  $\Rightarrow$ vaihtoehto 2 on jono

Esimerkissä joukko-opin käyttö ei olisi ollut tarpeen

esimerkissä	predikaateilla $X(x)$ ja $A(a)$
$\forall x \in X : \ldots$	$\forall x : X(x) \to \ldots$
$\forall a \in A \setminus \{\bot_A\} : \ldots$	$\forall a : A(a) \wedge a \neq \bot_A \to \ldots$

- on tavallista käyttää sitä lisäämään selkeyttä

Esimerkissä onnistuttiin kuvaamaan pinon ja jonon toiminta
tyhjentävästi predikaattilogiikan kaavoilla viittaamatta toteutukseen

- käytännön menetelmissä on esim. virheiden käsittelyyn valmiit mekanismit
  - esimerkissä sekin määriteltiin alusta asti predikaattilogiikalla
- on tavallista määritellä matemaattinen tai tietojenkäsittelytieteen käsite predikaattilogiikalla tai joukko-opilla ja predikaattilogiikalla
  $\Rightarrow$ käsitteen merkitys määräytyy täsmällisistä kaavoista eikä puheesta ja esimerkeistä
  - toki esimerkkejä voi olla havainnollistamassa
  - joukko-oppi voidaan määritellä predikaattilogiikalla
- käytännön ohjelmistotyössä käytetään vaihtelevasti mm. luonnollista kieltä, kaavioita, vertailutoteutuksia ja matemaattisia määrittelymenetelmiä

Käsitteen määritteleminen predikaattilogiikalla

- kieli
  - sovellusaluekohtaiset symbolit, esim. $0, 1, +, -$ tai $X, A, \bot_X, \bot_A, \varepsilon, i, d, g$
  - predikaattilogiikan symbolit $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists, =$ ja muuttujat
- aksioomat
  - joukko kohdemaailman ominaisuuksia kuvaavia kaavoja
  - esim. $\forall x : \forall y : \forall z : x(y + z) = xy + xz$
  - esim. $\forall x : \forall a : x \neq \varepsilon \wedge x \neq \bot_X \wedge a \neq \bot_A : d(i(x, a)) = i(d(x), a)$

- logiikan kaava, todistus, yms. ei saa olla ääretön
  - muuten sitä ei voi ihminen eikä tietokone lukea läpi asti edes periaatteessa

  $\Rightarrow$ kukin aksiooma on äärellinen
- muuttujia, sovellusaluekohtaisia symboleita ja aksioomia saa olla äärettömästi
  - esim. meillä on äärettömästi symboleita luvuille; 0, 5, 27, 338, ...
- tällainen määritelmä muodostaa *1. kertaluvun teorian*

Ääretön määrä aksioomia

- on tärkeitä ominaisuuksia, joita ei voi määritellä äärellisellä määrällä aksioomia

  $\Rightarrow$ sallitaan äärellinen määrä "aksioomaskeemoja"
  - aksioomaskeema on äärellinen esitystapa äärettömälle joukolle aksioomia
- yksinkertainen esimerkki hyödyllisestä äärettömästä aksioomajoukosta: kohdemaailmassa on äärettömästi alkioita

  $\{ \exists x_1 : \exists x_2 : \neg(x_1 = x_2),$
  $\exists x_1 : \exists x_2 : \exists x_3 : \neg(x_1 = x_2 \vee x_1 = x_3 \vee x_2 = x_3),$
  $\exists x_1 : \exists x_2 : \exists x_3 : \exists x_4 : \neg(x_1 = x_2 \vee x_1 = x_3 \vee x_1 = x_4 \vee x_2 = x_3 \vee x_2 = x_4 \vee$
  $x_3 = x_4),$
  $\ldots \}$
- tärkein, mutta ehkä ei vielä kovin tuttu esimerkki aksioomaskeemasta: luonnollisten lukujen induktioperiaate
  - jokaista muuttujaa n ja kaavaa $\varphi(n)$ kohti on aksiooma

    $(\varphi(0) \wedge \forall n : \varphi(n) \rightarrow \varphi(n+1)) \rightarrow \forall n : \varphi(n)$

- usein vaaditaan, että aksioomaskeemat ovat rekursiivisia
  tai ainakin rekursiivisesti lueteltavia
  - siis että on olemassa tietokoneohjelma, joka selvittää
    mistä tahansa merkkijonosta, onko se aksiooma vai ei
  - jos teoria ei ole rekursiivisesti lueteltava, niin ei voida tarkastaa todistuksia

  $\Rightarrow$ teoria olisi käytännössä kelvoton
- muilta osin predikaattilogiikka on rekursiivista
  - olettaen, että muuttujien jne. syntaksi on tolkullinen

$\Rightarrow$ on olemassa tietokoneohjelma, joka selvittää mistä
tahansa merkkijonosta, onko se pätevä todistus vai ei

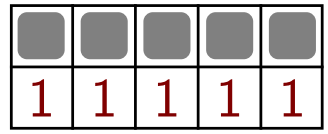
- tällainen määritelmä muodostaa *1. kertaluvun rekursiivisen teorian*
- 1. kertaluvun rekursiiviset teoriat ovat erittäin
  yleinen tapa muodostaa matemaattisia teorioita

Miksi logiikan sääntöihin on otettu tarkastettavuus tietokoneella?

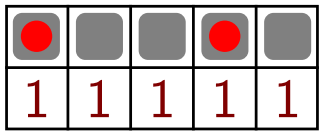
- miksi matematiikka on tällä tavalla sidottu ohjelmointiin?
- alun perin eli 1930-luvulla kokeiltiin useita vaihtoehtoja
- jälkikäteen paljastui, että ne kaikki johtivat samaan lopputulokseen
  - Wikipedia "Church–Turing thesis"
- ohjelmistoalan opiskelijoille "tarkastettavuus tietokoneella"
  on paljon helpompi ymmärtää kuin muut vaihtoehdot

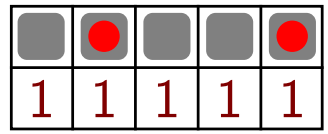
Aksioomajärjestelmän tulkinta ja malli

- aksioomajärjestelmän **tulkinta** on järjestelmä, josta voi puhua aksioomien käyttämillä symboleilla
- esim. $\forall x \in A : x \circ \sharp = \sharp \circ x = x$ ja $\forall x \in A : \forall y \in A : \forall z \in A : (x \circ y) \circ z = x \circ (y \circ z)$
  - voi olla $A = \mathbb{N}$, $\sharp = 0$ ja $\circ$ on $+$
  - voi olla $A = \mathbb{N}$, $\sharp = 1$ ja $\circ$ on $\cdot$
  - voi olla $A = \{\mathsf{false}, \mathsf{true}\}$, $\sharp = \mathsf{false}$ ja $\circ$ on $\vee$
  - voi olla A on Rubikin kuution kiertosarjojen vaikutukset, $\sharp$ on vaikutukseton kiertosarja (esim. täysi kierros) ja $\circ$ on kiertosarjan tekeminen edellisen perään
- olemme aikaisemmin käyttäneet sanaa "kohdemaailma"
- aksioomat eivät välttämättä päde tulkinnassa
- aksioomajärjestelmän **malli** on tulkinta, joka toteuttaa kaikki aksioomat
  - kaikki edellä esimerkkinä olleet tulkinnat ovat malleja

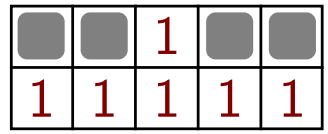


Aksioomien looginen seuraus





- esimerkki: miinapeli
- voi olla, että usea erilainen tilanne sopii tiedossa olevaan informaatioon
  - miinapeliesimerkillämme on kaksi eri mallia
- silti voi olla mahdollista päätellä jotakin varmasti



- aksioomajärjestelmän *looginen seuraus* on ko. järjestelmän kielellä ilmaistu väittämä, joka pätee jokaisessa mallissa
  - miinapelimme keskimmäinen peitetty ruutu on turvallinen molemmissa malleissa

Gödelin täydellisyyslause

- Kurt Gödel todisti vuonna 1929, että 1. kertaluvun rekursiivisten teorioiden tapauksessa jokainen looginen seuraus voidaan todistaa hänen käyttämällään päättelyjärjestelmällä
- päättelyjärjestelmät ovat sittemmin muuttuneet, mutta tulos pätee myös sille päättelyjärjestelmälle jonka esitimme luvussa 6.1
- tämä on valtavan voimakas tulos!
  - aksioomiin sisältyvä informaatio voidaan hyödyntää päättelyssä täydellisesti
  - jos voimme muotoilla kohdemaailmalle täydelliset aksioomat, saamme periaatteessa kaikki kohdemaailman totuudet
  - tosin tulos ei lupaa, että todistukset olisivat helppoja löytää

Mallin olemassaololause

- teoria on ristiriitainen, jos ja vain jos siinä voi todistaa jollekin φ sekä φ että $\neg\varphi$
- Leon Henkin todisti 1949, että jos 1. kertaluvun rekursiivinen teoria on ristiriidaton, niin sillä on malli
  - jopa äärellinen tai numeroituvasti ääretön malli
  - Henkinin todistus rakentaa mallin keinotekoisesti kielen kaikista lausekepuista

- Henkinin tuloksesta saadaan Gödelin täydellisyyslause helposti:
  - jos φ:tä ei voi todistaa, niin $\neg\varphi$:n lisääminen aksioomiin ei aiheuta ristiriitaa

  $\Rightarrow$ sillä tavalla täydennetyllä aksioomajärjestelmällä on malli

  $\Rightarrow$ alkuperäisellä aksioomajärjestelmällä on malli, jossa $\neg\varphi$ pätee

  $\Rightarrow$ φ ei ole alkuperäisten aksioomien looginen seuraus
- Henkinin todistus ei ole helppo, mutta on helpompi kuin Gödelin

$\Rightarrow$ nykyisin täydellisyyslause todistetaan usein Henkinin reittiä

Gödelin 1. epätäydellisyyslauseen moderni (tietojenkäsittelijöiden) versio

- oletamme kielen, jossa on 0, 1, $+$, $\cdot$, $=$, $\forall$, $\exists$, $\wedge$, $\vee$, $\neg$, $($, $)$ ja rajattomasti muuttujia
- oletamme tässä vaiheessa teoriasta lisäksi vain, että se on rekursiivisesti lueteltava
  - kuten edellä todettiin, ilman tätä oletusta teoria olisi käyttökelvoton
  - nyt emme oleta esim. edellä lueteltuja predikaattilogiikan sääntöjä
- Kurt Gödel todisti 1931, että
  - tällä kielellä esitetyt väittämät voidaan koodata luonnollisiksi luvuiksi
  - todistukset voidaan koodata luonnollisten lukujen laskutoimituksiksi
  - tätä kutsutaan Gödel-numeroinniksi
  - ei ollut helppoa, varsinkin kun potenssilasku ei ollut käytettävissä!
  - Gödel oletti rekursiivisuuden, mutta rekursiivinen lueteltavuus riittää
- Gödel muotoili numerointinsa avulla luonnollisia lukuja koskevan väittämän, joka intuitiivisesti tulkittuna sanoo "minä olen väittämä, jolle ei ole todistusta"

- jos sille on todistus, niin se on epätosi väittämä jolle on todistus
  $\Rightarrow$ teoria on rikki
- jos sille ei ole todistusta, niin se on tosi väittämä jolle ei ole todistusta
  $\Rightarrow$ teoria on epätäydellinen

$\Rightarrow$ ei ole olemassa rekursiivisesti lueteltavaa teoriaa, jossa voidaan todistaa kaikki luonnollisia lukuja koskevat em. kielellä ilmaistavissa olevat totuudet ja vain ne!

- 1. kertaluvun rekursiivisessa tapauksessa
  - päättelykyky on täydellinen

  $\Rightarrow$ aksioomien joukko ei voi olla täydellinen

$\Rightarrow$ jokainen luonnollisten lukujen 1. kertaluvun rekursiivinen aksiomatisointi on epätäydellinen

- yleisesti käytetään Peanon aritmetiikkaa
  - $\forall x : \neg(x + 1 = 0)$
  - $\forall x : \forall y : x + 1 = y + 1 \rightarrow x = y$
  - $\forall x : x + 0 = x$
  - $\forall x : x + (y + 1) = (x + y) + 1$
  - $\forall x : x \cdot 0 = 0$
  - $\forall x : x \cdot (y + 1) = x \cdot y + x$
  - $(\varphi(0) \wedge \forall n : \varphi(n) \rightarrow \varphi(n + 1)) \rightarrow \forall n : \varphi(n)$
- se koodaa luonnollisten lukujen ominaisuudet erittäin kattavasti, mutta ei täysin
- mallin olemassaololauseen vuoksi sille on olemassa epästandardeja malleja

Gödelin 1. epätäydellisyyslauseen alkuperäinen versio

- Gödelin aikana vallinnut filosofinen ajattelu ei olisi hyväksynyt yllä olevaa
  - jos väittämälle ei ole todistusta, niin millä perusteella sitä voi väittää todeksi?
- Gödel
  - oletti rekursiivisuuden ja jonkin verran tuttuja aksioomia ja päättelysääntöjä
  - todisti, että jokainen todistusjärjestelmä on joko "ω-ristiriitainen" tai siinä mielessä epätäydellinen, että jollekin φ se jättää todistamatta sekä φ:n että $\neg\varphi$:n
- J. Barkley Rosser onnistui 1936 korvaamaan ω-ristiriitaisuuden ristiriitaisuudella
- jopa laadukkaista kirjoista löytyy 1. epätäydellisyyslauseesta omituisia väitteitä, joissa ym. kaksi versiota ovat menneet sekaisin

Gödelin 2. epätäydellisyyslause

- menemättä yksityiskohtiin, Gödelin 1. epätäydellisyyslauseesta voidaan päätellä tulos, joka käytännössä tarkoittaa, että matematiikan voi todistaa ristiriidattomaksi vain jos se ei ole ristiriidaton

Gödelin epätäydellisyystulosten merkitys

- ne olivat vakava takaisku silloiselle tavoitteelle mekanisoida matematiikka
- ne näyttivät tietä mm. Turingin tulokselle, että pysähtymistesteriä ei ole
- meille ne ovat yksi tulos lisää kertomaan tekoälyn ja ohjelmoinnin rajoista

7 Lopuksi