4.3. Pelit [RN95, luku 5][Nil98, luku 12]

Lautapelit ovat valmiiksi abstrahoitu ongelmaluokka, jossa älykkäät toimijat ottavat mittaa

- toisistaan ja/tai
- nopasta (joka edustaa ympäristön sattumanvaraisuutta),

ja siksi kauan ja paljon tutkittuja tekoälyssä.

Pelit ovat esimerkki luvun 3 varautumisongelmasta, jossa hyvää pelistrategiaa etsivän toimijan on *varauduttava siihen, että vastustaja ja/tai noppa eivät tottelekaan*.

Pelit ovat (yleensä) *laskennallisesti erittäin työläitä:*

Kussakin shakkitilanteessa on keskimäärin noin 35 eri siirtomahdollisuutta, ja peli saattaa jatkua 50 siirtoparia.

Siis etsintäpuussa on 35^{100} solmua (joista tosin "vain" noin 10^{40} keskenään erilaista).

Noppa ei siis ole ainoa epävarmuuden lähde, vaan toimija ei myöskään *käytännössä* voi kerätä kaikkea periaatteessa saatavilla olevaa informaatiota päätöksenteon tueksi — varsinkaan jos pelissä on aikaraja, eli jos se on semistaattinen ympäristö.

Rajoitutaan tarkastelemaan

kahden pelaajan. . . nimiltään $\max$ (joka tekee aloitus*siirron*) ja $\min$ (joka *vastaa* aloitukseen).

Useamman pelaajan peleissä joudutaan esimerkiksi määrittelemään, saavatko pelaajat sopia liittoutuvansa keskenään ja pitääkö sopimuksesta kertoa muille pelaajille,. . .

täydellisen informaation. . . eli kumpikin saa pelistä kaiken tarpeellisen tiedon (erityisesti sen säännöt).

Luvun 2.6 käsittein ympäristö on saavutettavissa.

nollasummapelejä. Jaossa oleva "palkintosumma" koostuu tällöin pelkistä pelaajien osallistumismaksuista. Kummankaan ei siis kannata tavoitella häviötä.

Vastakohtana on esimerkiksi nyrkkeilyottelu, jossa häviäjäkin tietää saavansa lipputuloista kipurahoja, joskin vähemmän kuin voittaja.

Silloin rationaalinen nyrkkeilijä saattaa tulla siihen tulokseen, että luovuttaminen heti kannattaa kun sairaalakulutkin lasketaan.

Määritellään tällainen peli seuraavana etsintäongelmana:

Pelitilanne käsittää sekä *laudan* L *tilanteen* että tiedon *kumman pelaajan vuoro* on siirtää seuraavaksi.

Alkutilanteessa siis L on alkuasetelmassa ja $\max$ vuorossa.

Operaattorit kertovat vuorossa olevan pelaajan P mahdolliset siirrot laudalla L:

Kaikki ne pelitilanteet, joissa

- pelivuoro vaihtuu
- lauta L' saadaan kun P siirtää jotakin "nappulaansa" laudalla L sääntöjen mukaan.

Lopputesti (terminal test) kertoo onko peli ohi annetussa pelitilanteessa.

Näitä pelitilanteita kutsutaan **lopputilanteiksi (terminal states)**.

Utiliteetti- eli tulosfunktio (utility/payoff function) antaa numeroarvon jokaiselle lopputilanteelle H.

Esimerkiksi shakissa

$$U(H) = \begin{cases} +1 & \text{kun } \max \text{ voitti lopputilanteessa } H \\ \pm 0 & \text{kun tuli tasapeli} \\ -1 & \text{kun } \min \text{ voitti.} \end{cases}$$

Tavoitteena on löytää **strategia** aloittajalle $\max$:

Sellainen pelitapa, jonka noudattaminen johtaa aina hänen kannaltaan parhaimpaan tarjolla olevaan lopputulokseen — sellaiseen H, jossa $U(H)$ on niin korkea kuin suinkin on mahdollista saavuttaa.

Vastaavassa etsintä- eli pelipuussa $\max$- ja $\min$-tasot vuorottelevat.

Strategia ei ole etsintäpuun polku vaan *kokonainen alipuu*:

Vastustajan $\min$ siirtoja ei tiedetä etukäteen, vaan niihin kaikkiin on varauduttava.

Pelaaja $\max$ pyrkii valitsemaan sellaisen alipuun, jonka lopputilannelehdet ovat hänen kannaltaan parhaimmat.

Käytännössä tämä valinta joudutaa siis tekemään *epätäydellisen informaation* pohjalta: eri alipuista voidaan tutkia vain jokin juurta lähellä oleva osa.

4.3.1. Minimax-periaate

Parhaimman alipuun valinnassa $\max$ voi järkeillä seuraavasti:

- "Nykyisessä lautatilanteessa L_1 minun kannattaa tehdä *jokin* sellainen siirto M_1 joka vie sellaiseen lautatilanteeseen L'_1, jossa. . .
- *jokainen* siirto M'_1 jonka vastustajani $\min$ voi tehdä lautatilanteessa L'_1 vie sellaiseen lautatilanteeseen L_2, jossa. . .
- minulla on *jokin* seuraava siirto M_2 lautatilanteessa L_2 joka vie sellaiseen lautatilanteeseen L'_2 jossa. . .
- *jokainen* siirto M'_2 jonka vastustajani $\min$ voi tehdä lautatilanteessa L'_2 vie sellaiseen lautatilanteeseen L_3, jossa. . .

$$\vdots$$

- (viimeisessä) lautatilanteessa L_n peli on päättynyt minun voittooni.

Vastustaja $\min$ ajattelee täsmälleen samoin.

Tarkastellaan sitten sitä *koko pelipuuta*, jonka juurena on nykyinen pelitilanne.

Järkeily johtaa seuraavaan tapaan arvottaa sen kaikki solmut:

Lehti saa arvokseen sitä vastaavan lopputilanteen H utiliteetin $U(H)$.

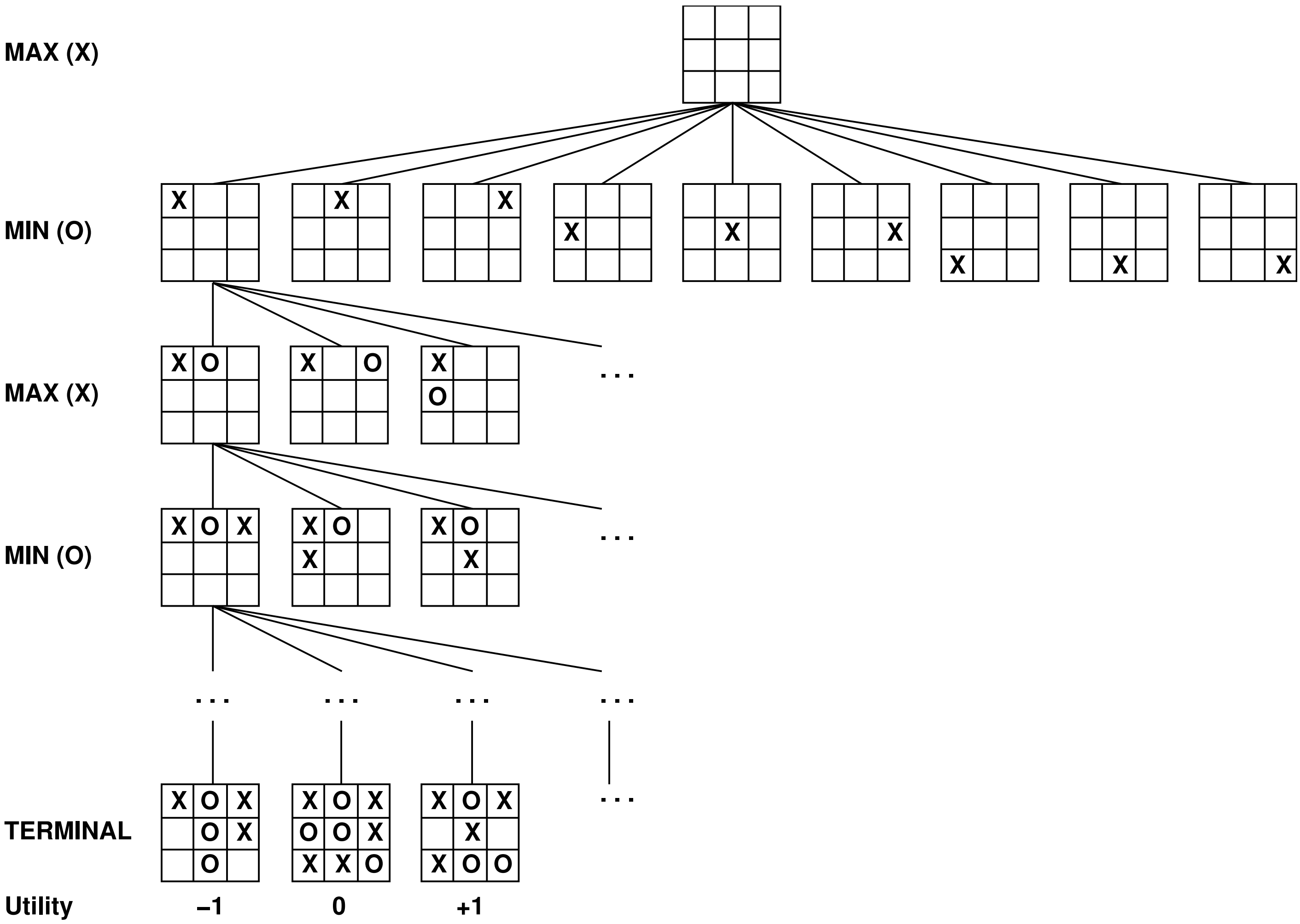
$\max$-**tason sisäsolmu** saa arvokseen luvun *maksimin* sen lapsisolmujen arvoista.

$\min$-**tason sisäsolmu** saa arvokseen luvun *minimin* sen lapsisolmujen arvoista.

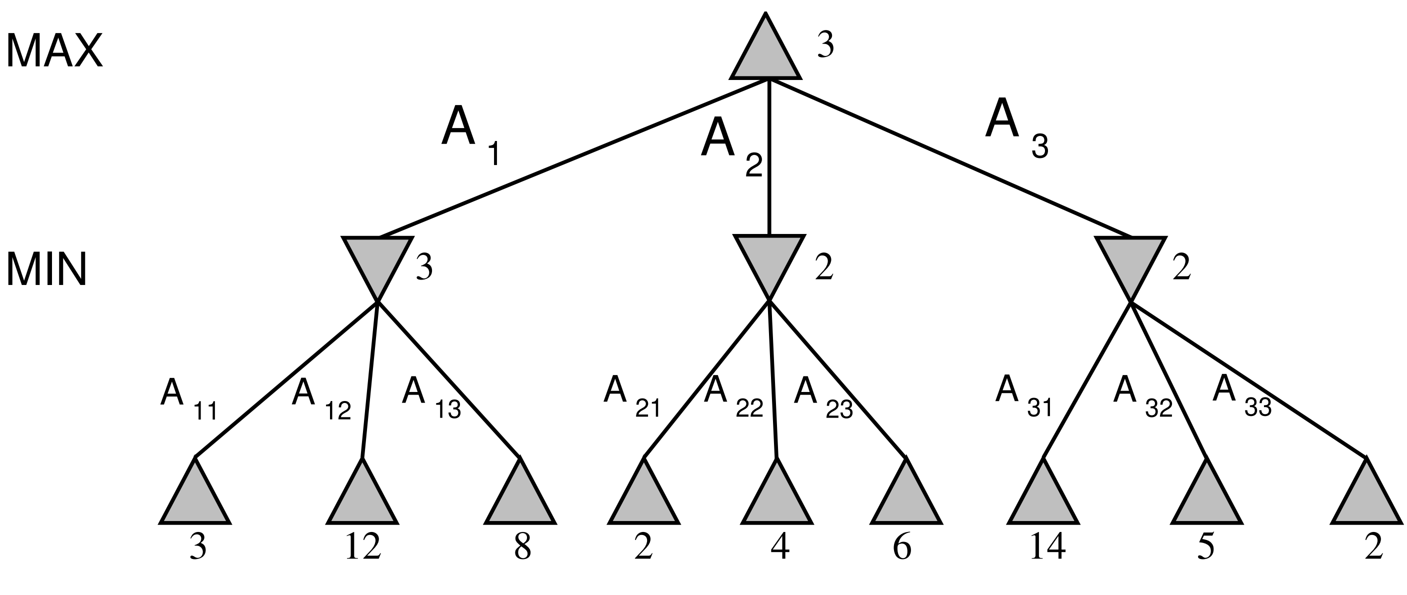
Näin edetään lehdistä aina juureen asti.

Sitten $\max$ tekee **minimax-päätöksen** valitsemalla juuressa sen siirron, joka vie johonkin maksimiarvon antaneeseen alipuuhun.

Siis $\max$ valitsee sen siirron, joka on paras oletuksella, että $\min$ valitsee aina mahdollisimman ikävän vastaussiirron.



Kuva 45: Risti-nolla -pelin etsintäpuu [RN95, kuva 5.1].



Kuva 46: Kahden siirron minimax-puu [RN95, kuva 5.2].

```
function MINIMAX-DECISION(game) returns an operator

  for each op in OPERATORS[game] do
     VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]
```

```
function MINIMAX-VALUE(state, game) returns a utility value

  if TERMINAL-TEST[game](state) then
     return UTILITY[game](state)
  else if MAX is to move in state then
     return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
     return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Algoritmi 10: Minimax-algoritmi [RN95, kuva 5.3].

Minimax-periaate on muotoiltu luvun 3.1.3 kaltaisena syvyyssuuntaisena etsintäalgoritmina 10, joka arvottaa etsintäpuun solmut lähtien liikkeelle nyt juuresta lehtien sijaan.

Algoritmi olettaa, että kaikki pelit päättyvät: esimerkiksi toistuvaan pelitilanteeseen vievä siirto

Shakissa merkitsee pelin päättymistä tasapeliin.

GO-pelissä on kielletty.

Molemmissa tapauksissa on etsinnän kuluessa tutkittava pelin historiaa eli pelipuun polkua juuresta nykyiseen solmuun samoin kuin luvun 3.1.7 tavassa 1.

Algoritmi 10 tutkii koko pelipuun, vaikka juuri todettiin sen olevan käytännössä mahdotonta. Muutetaan sitä siis seuraavasti:

- Puuta kasvatetaan vain siihen saakka kunnnes valittu *katkaisuehto (cutoff-test)* laukeaa, ei enää lopputilanteisiin asti.
  Yksinkertaisia katkaisuehtoja saadaan esimerkiksi

– valitsemalla etukäteen vakiosyvyys d jota pidemmälle ei edetä (kuten luvussa 3.1.4) tai
– syventämällä etsintää iteratiivisesti (kuten luvussa 3.1.5) kunnes sille varattu aika täyttyy.

Nämä ehdot kärsivät kuitenkin *tapahtumahorisonttiongelmasta (horizon problem)*, jossa etsintä päättyy liian aikaisin havaitakseen pelitilanteen todellisen laidan:

Kuvassa 47 musta pääsee shakkaamaan viidesti peräkkäin, mutta lopulta valkea saa kuningattaren ja todennäköisesti voittaa koko pelin.

Tätä voi yrittää ehkäistä pysäyttämällä etsintä vasta *hiljaisissa (quiescent)* pelitilanteissa, joissa näyttää siltä, ettei pelitilanteessa tapahdu enää dramaattisia yllätyksiä.

Toisaalta kuvassa 47 pelitilanne näyttää pitkään perin yllätyksettömältä. . .

- Katkaisukohdassa arvioidaan solmun todellista arvoa valitulla *evaluointifunktiolla (evaluation function)*.

  Esimerkiksi shakin alkeiskirjoissa annetaan yksinkertaistettuja pistelaskusääntöjä materiaaliedun arviointiin:

  Lasketaan yhteen sotilas 1, ratsu/lähetti 3, torni 5, kuningatar 9, "hyvä" sotilas- tai kuningasasema $\frac{1}{2}$,. . . (Vrt. kuva 48.)

  Tämä on esimerkki yleisestä *lineaarisesta* evaluointifunktiosta. Vastaavia nyrkkisääntöjä korttipelille Bridge esiteltiin luvussa 3.2.3 etsintäheuristiikkoja keksittäessä.

- Evaluointifunktion laatu on keskeinen koko peliohjelman laadulle:

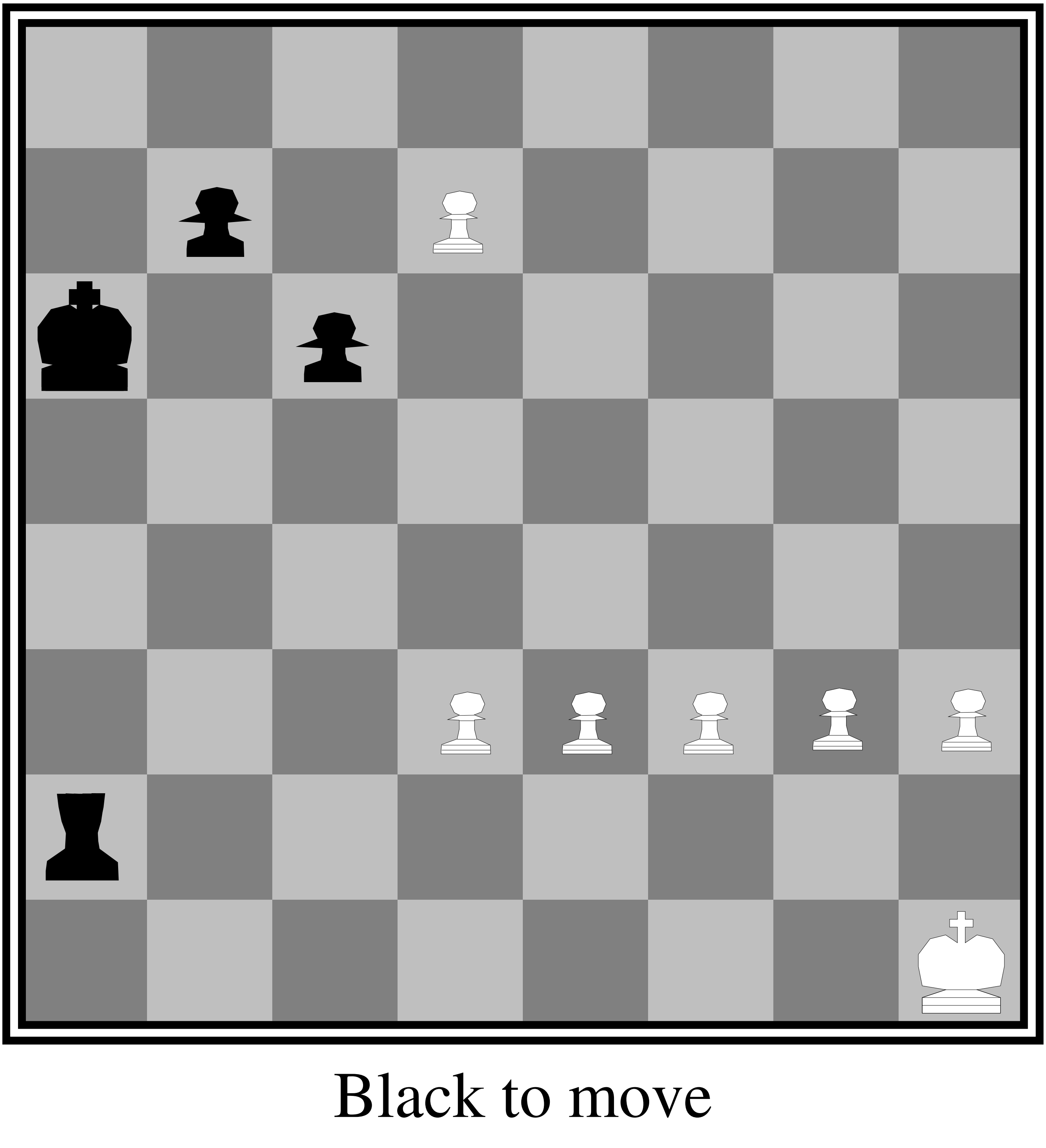
  Solmulle H annetun arvon $\textsc{Eval}(H)$ tulisi heijastaa mahdollisimman luotettavasti sitä, kuinka pelitilanteesta H jatkuvassa pelissä *odotusarvoisesti* tulisi käymään.

  Arvo $\textsc{Eval}(H)$ edustaa näet yksinään kokonaista solmusta H alkavaa pelipuun alipuuta, jonka lehtien L arvot $U(L)$ saattavat vaihdella suurestikin.

  Jos $\textsc{Eval}(H)$ mallintaa suureen $U(L)$ odotusarvoa kun L valitaan umpimähkään, niin se mallintaa samaa pelistrategiaa kuin katkaistu etsintäkin:

  Edetään harkitusti tilanteeseen H saakka, sen jälkeen umpimähkään.

Tällöin korkea $\textsc{Eval}(H)$ vihjaa, että tilanteesta H alkavasta alipuusta T löytynee myös korkeaan arvoon $U(L)$ vievä strategia.

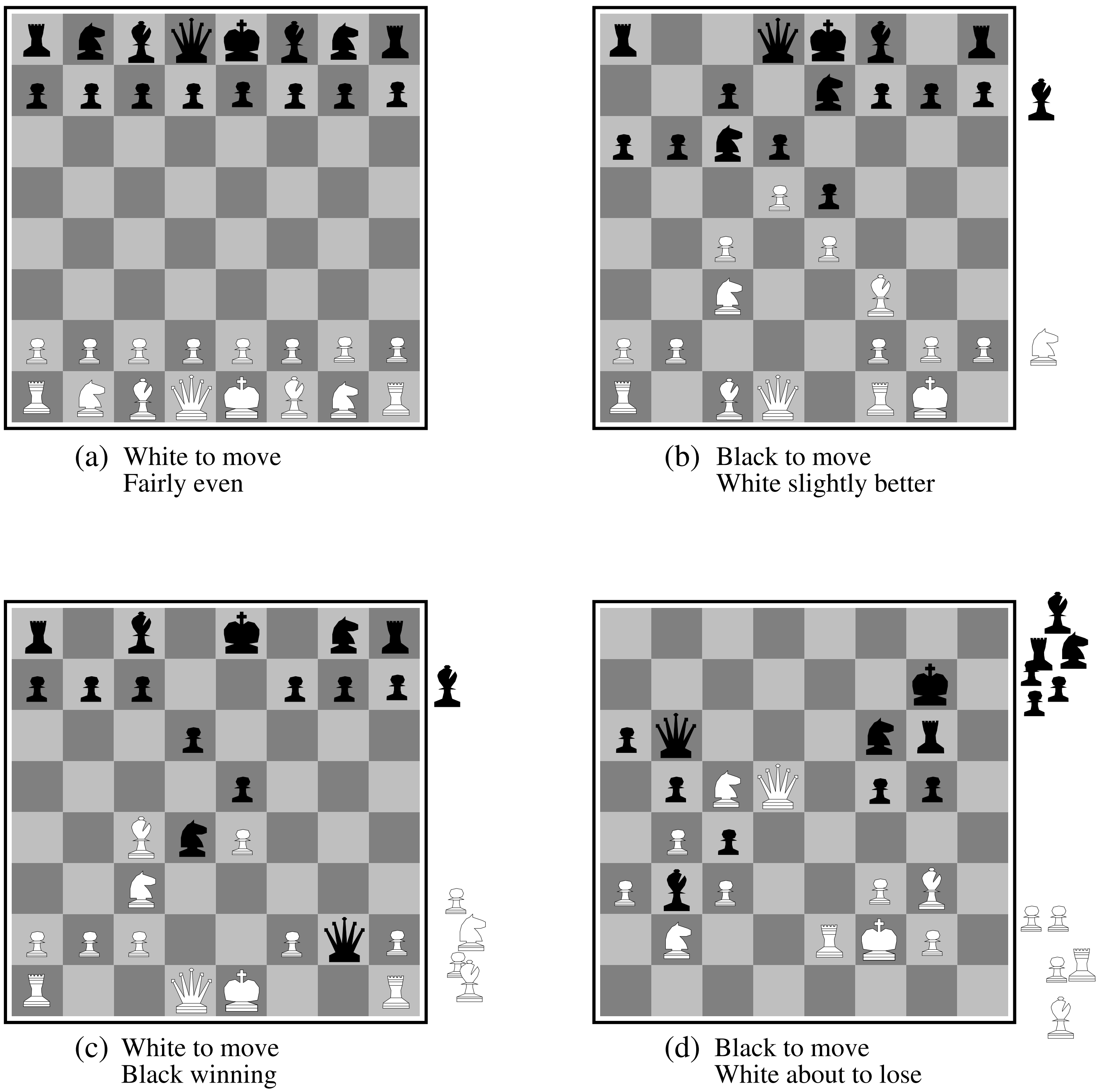


Kuva 47: Musta shakkaa, valkea voittaa [RN95, kuva 5.5].

Onneksi funktion $\textsc{Eval}$ ei tarvitse saada eri solmujen H_1 ja H_2 numero*arvoja* oikein, kunhan niiden *järjestys* vain säilyy, eli

$$\textsc{Eval}(H_1) \leq \textsc{Eval}(H_2) \tag{12}$$

vain jos H_2 on "ainakin yhtä lupaava pelitilanne" kuin H_1.



Kuva 48: Shakkitilanteita arvioineen [RN95, kuva 5.4].

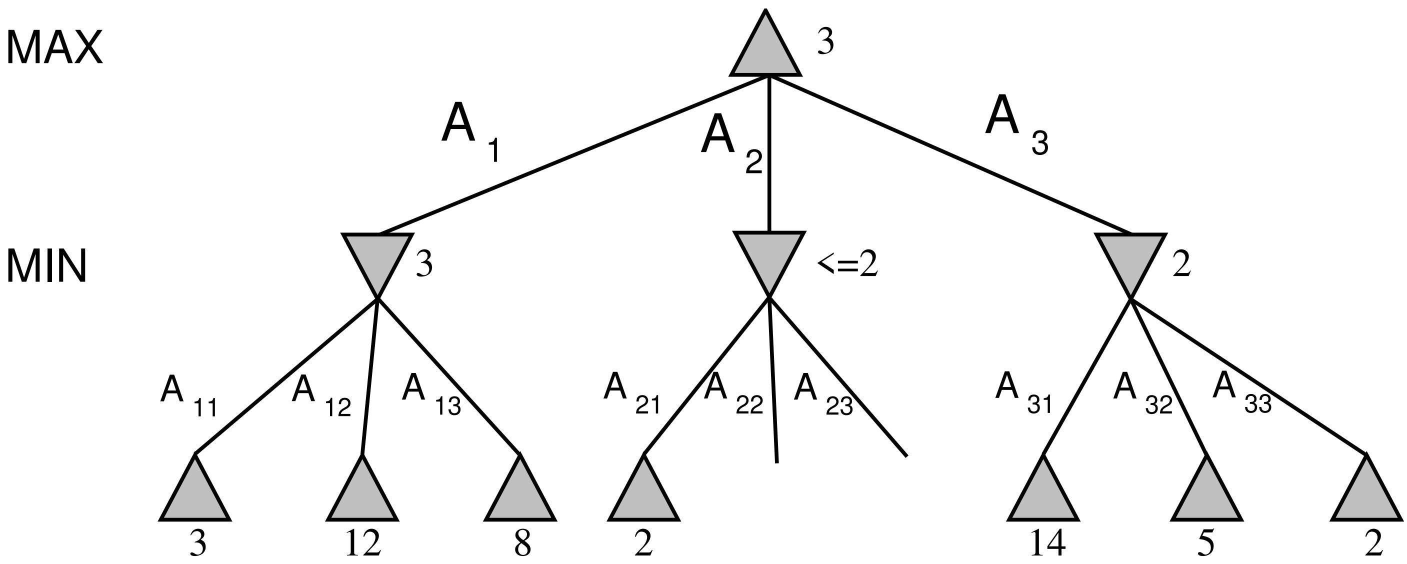
4.3.2. Alfa-beta -karsinta

Minimax-algoritmi 10 on käytännössä yhä liian hidas vaikkei se enää kävisikään läpi koko pelipuuta.

Seuraavaksi esitetään eräs tapa *karsia (prune)* pelipuuta, eli jättää joitakin sen alipuita tutkimatta ilman muutosta kokonaistulokseen.

Karsintaperiaatteemme on *alfa-beta (α-β)*, joka on saanut nimensä karsintaa ohjaavista luvuistaan

$$\begin{matrix}\alpha \\ \beta\end{matrix} = \begin{matrix}\text{ala} \\ \text{ylä}\end{matrix}\text{raja pelaajan } \begin{matrix}\max \\ \min\end{matrix} \text{ saamalle arvolle.}$$



Kuva 49: Kuvan 46 alfa-beta -karsinta [RN95, kuva 5.6].

Esimerkiksi kuvan 46 pelipuusta voidaan jättää tutkimatta kuvan 49 osoittamat solmut:

Syvyyssuuntainen minimax-etsintä (algoritmi 10) vasemmalta oikealle antaa

1. ensin vaihtoehdolle A_1 minimax-arvon 3,

2. sitten vaihtoehdolle A_{21} minimax-arvon 2
3. jonka nojalla vaihtoehdon A_2 minimax-arvo on *enintään* tuo 2
4. eli pelaaja $\max$ ei koskaan pelaisi vaihtoehtoa A_2 vaan vaihtoehdon A_1 (tai vielä sitäkin paremmin).

Yleinen karsintaperiaate näkyy kuvasta 50:

Pelaaja (player) $\max$ ($\min$) ei koskaan valitse solmua n jos hänellä on jokin parempi vaihtoehto $m > n$ ($m < n$) pelipuun juuresta solmuun n vievällä polulla.

Koska minimax-algoritmi 10 eteni syvyyssuuntaisesti, se tarkastelee aina yhtä polkua kerrallaan.

Pidetään siis yllä vain tähän polkuun liittyviä lukuja α ja β etsinnän edetessä. Saadaan algoritmi 11.

- Algoritmin 11 karsintatehokkuuus riippuu siitä järjestyksestä, missä solmun lapset käydään läpi:

  jos kuvan 49 alipuu A_3 olisi aloitettu lapsesta A_{33}, niin

  3. $\textsc{Min-Value}(A_3, 3, +\infty) =$
     (a) $\textsc{Max-Value}(A_{33}, 3, +\infty) = 2$
     $= 3.$

  Algoritmi 11 karsii sitä paremmin, mitä aikaisemmin kunkin solmun oikea minimax-arvo tavataan sen lapsista.

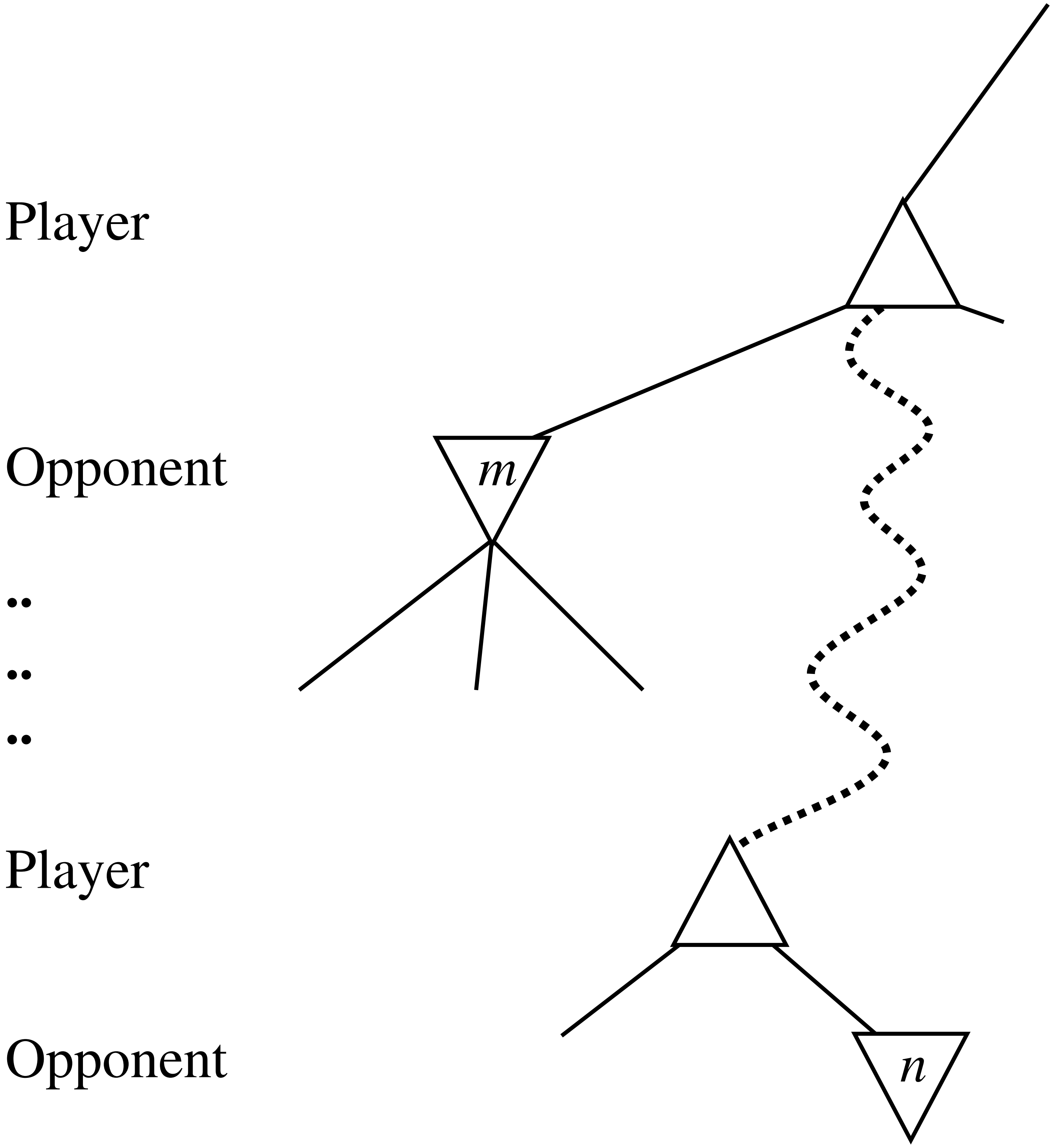
  Jos voisimme järjestää lapset aina etukäteen oikein ilman niiden läpikäyntiä, niin voisimme pelata peliä täydellisesti ilman etsintää. . .

  Tai saavuttaa algoritmin 11 optimitapaus: tutkia vain $O(b^{d/2})$ solmua, missä

  – b on pelipuun haarautumisaste eli mahdollisten siirtojen määrä kussakin pelitilanteessa
  – d puun syvyys eli siirtojen kokonaismäärä kussakin pelissä.

  (Algoritmi 10 tutkii puolestaan kaikki pelipuun $O(b^d)$ solmua.)
- Silloin luvun 3.2.3 todellinen haarautumisaste olisikin enää $\sqrt{b}$ eikä b.

- Toisin sanoen samalla etsintätyöllä päästäisiin kaksi kertaa niin syvälle. Ja etsintäsyvyys eli ”siirtoparien ajattelu eteenpäin” on keskeistä hyvässä strategiapelissä.
- Jos lapset ovat aivan satunnaisessa järjestyksessä, niin algoritmi 11 tutkii $O((b/\log b)^d)$ solmua.
  Silloin todellinen haarautumisaste $b/\log b$ on lähes b.



Kuva 50: Alfa-beta-karsinnan periaate [RN95, kuva 5.7].

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
  inputs: state, current state in game
          game, game description
          α, the best score for MAX along the path to state
          β, the best score for MIN along the path to state

  if CUTOFF-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
      α ← MAX(α, MIN-VALUE(s, game, α, β))
      if α ≥ β then return β
  end
  return α
```

```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

  if CUTOFF-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
      β ← MIN(β, MAX-VALUE(s, game, α, β))
      if β ≤ α then return α
  end
  return β
```

Algoritmi 11: Alfa-beta-algoritmi [RN95, kuva 5.3].

Alkukutsu $\textsc{Max-Value}($kuva 49$, -\infty, +\infty) =$

1. $\textsc{Min-Value}(A_1, -\infty, +\infty) =$
   (a) $\textsc{Max-Value}(A_{11}, -\infty, +\infty) = 3$
   (b) $\textsc{Max-Value}(A_{12}, -\infty, 3) = 12$
   (c) $\textsc{Max-Value}(A_{13}, -\infty, 3) = 8$
   $= 3$
2. $\textsc{Min-Value}(A_2, 3, +\infty) =$
   (a) $\textsc{Max-Value}(A_{21}, 3, +\infty) = 2$
   $= 3$
3. $\textsc{Min-Value}(A_3, 3, +\infty) =$
   (a) $\textsc{Max-Value}(A_{31}, 3, +\infty) = 14$
   (b) $\textsc{Max-Value}(A_{32}, 3, 14) = 5$
   (c) $\textsc{Max-Value}(A_{33}, 3, 5) = 2$
   $= 2$

$= 3.$

(Vakioparametri $game$ jätettiin selvyyden vuoksi pois.)

- Onneksi tämä asymptoottinen kaava pätee vasta kun $b > 1\ 000$, jolloin etsintäpuumenetelmät olisivat muutenkin käytännössä mahdottomia.

  "Luonnollisilla" haarautumisasteen b arvoilla algoritmi 11 tutkii $O(b^{3d/4})$ solmua.

  Jo melko yksinkertaiset lapsisolmujen käsittelyjärjestykset pääsevät usein melko lähelle tätä arvoa.

Edellä kuitenkin oletettiin, että funktion $\textsc{Eval}$ (tai U) arvot ovat jakautuneet satunnaisesti etsintäpuun lehdille.

Tämä tuskin pätee tavallisissa peleissä:

Jokin aikaisempi hyvä/huono siirto (eli sisäsolmu) korottaa/vähentää suurinta osaa lehtien arvoista kyseisessä alipuussa.

Näin saman alipuun eri solmujen arvot tapaavat riippua vahvasti toisistaan kullekin peli(tilantee)lle ominaisella tavalla.

Näiden piiloriippuvuuksien vuoksi hyvän strategiapelialgoritmin kehittämisessä

- kokeilulla
- koneoppimisella

on tärkeä rooli näitä yleisiä algoritmeja räätälöitäessä.

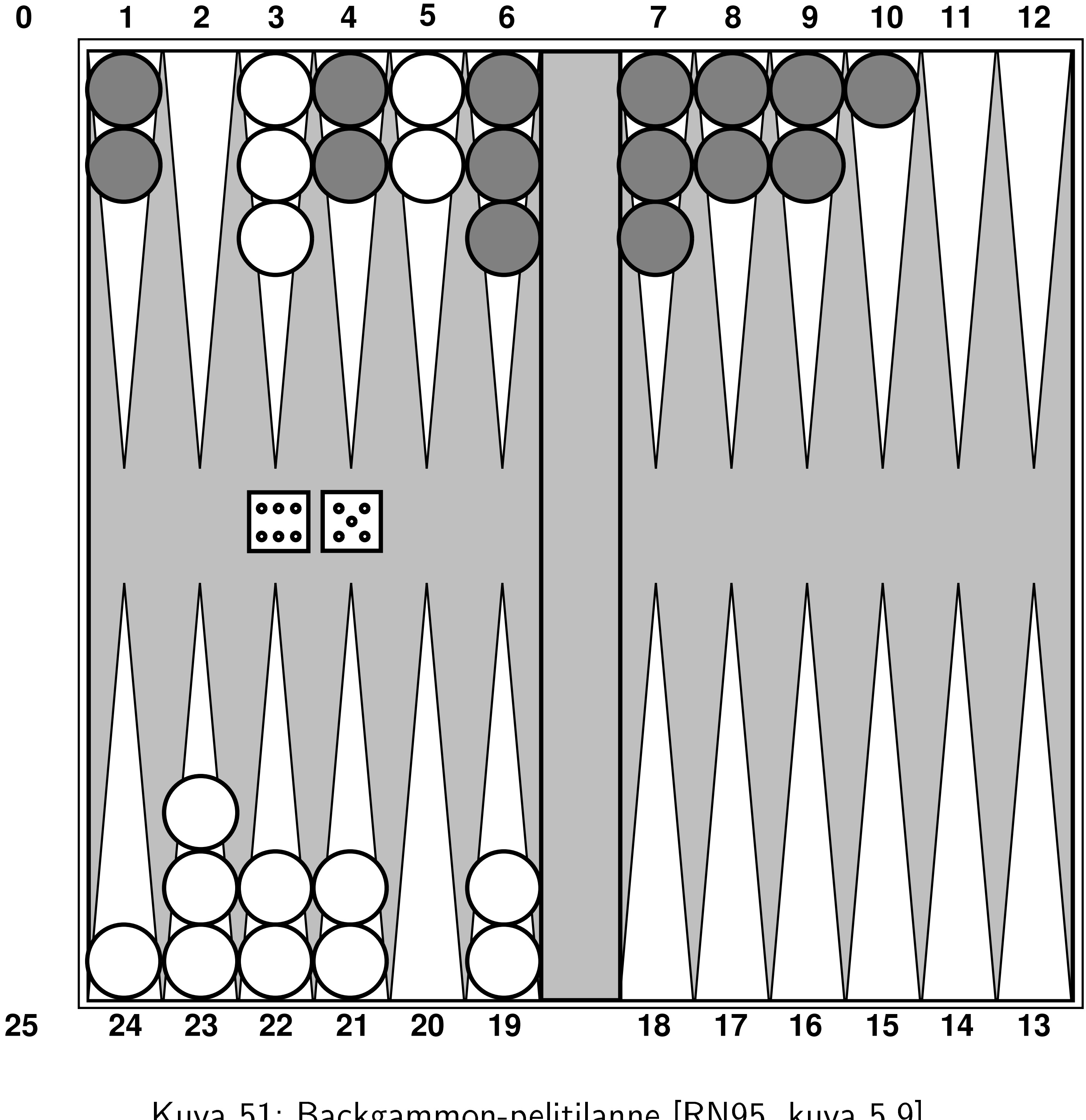
Alfa-beta-karsinta on yleisen *haaraudu ja rajoita (branch and bound)* -etsintäperiaatteen sovellus pelipuihin:

Kun etsintä haarautuu, yritä laskea rajoja sille, kuinka hyviä tuloksia voi vielä ilmaantua, ja karsi etsintähaaroja näiden rajojen salliessa.

4.3.3. Sattuma ja strategia

Lisätään strategiapeliimme vielä satunnainen komponentti.

Esimerkiksi Backgammon-pelaaja aloittaa vuoronsa heittämällä 2 noppaa, joiden silmäluvut kertovat ne siirtomahdollisuudet mistä valita.



Kuva 51: Backgammon-pelitilanne [RN95, kuva 5.9].

Sovelletaan pelipuuajattelua tällaisten nopasta riippuvien strategisten valintojen tekemiseen.

Lisätään siis pelipuihimme *sattumasolmu (chance node)* (DICE) jokaiseen sattuma- eli nopanheittokohtaan:

Lapsina ovat kaikki vastaavan *satunnaistapahtuman (random event) mahdolliset vaihtoehdot*.

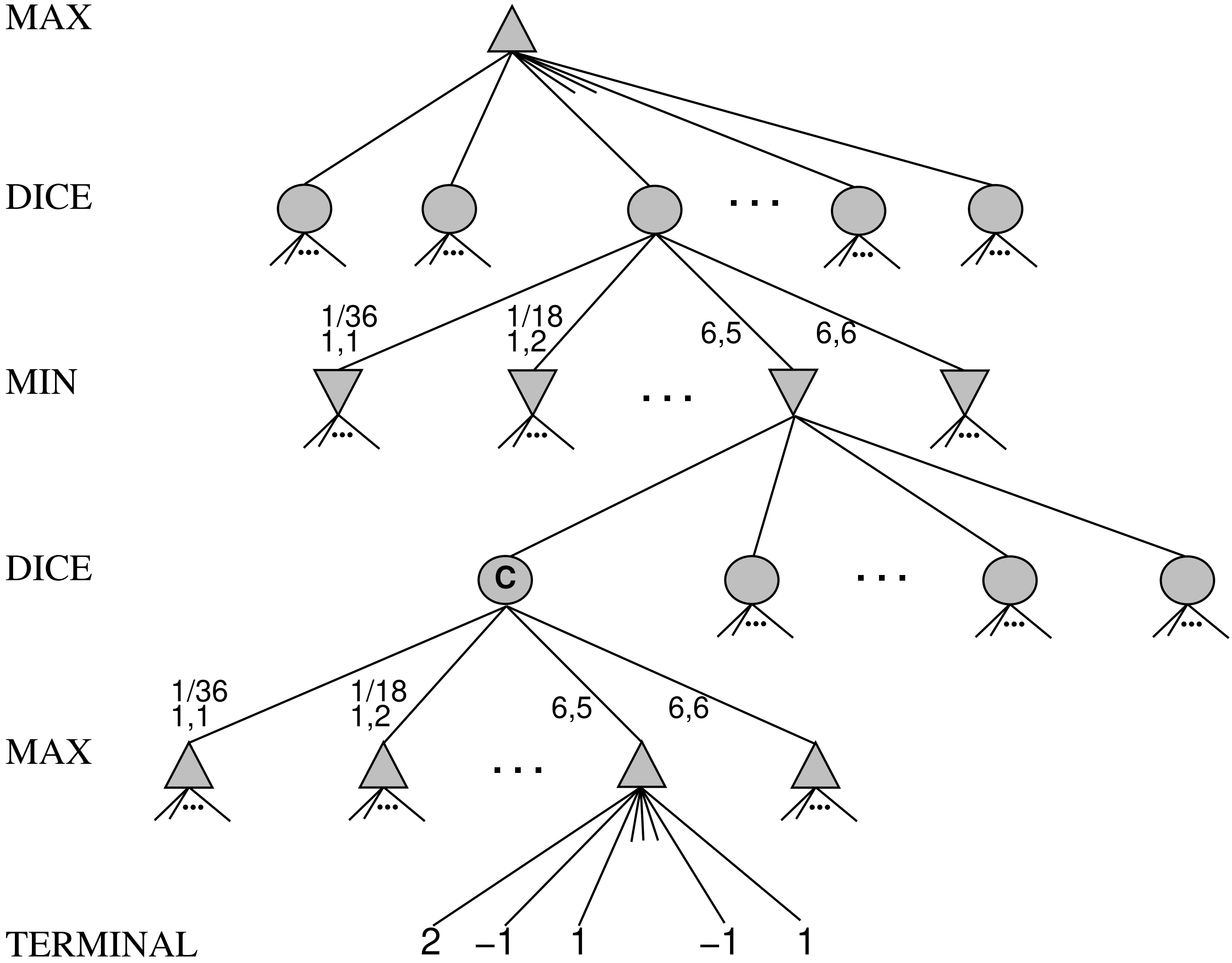
Backgammonissa kaikki 2 nopalla heitettävissä olevat (järjestämättömät) silmälukuparit.

Lapsesta alkaa pelipuu, jossa vaihtoehto on jo todellisuutta.

Haaralla on vastaavan vaihtoehdon *todennäköisyys*.

Backgammonissa kunkin silmälukuparin osuus kaikista silmälukupareista.

Sattumasolmut muodostavat *sattumatasoja* $\max$- ja $\min$-tasojen väliin.



Kuva 52: Backgammon-etsintäpuu [RN95, kuva 5.10].

Päästäksemme käyttämään luvun 4.3.1 minimax-periaatetta meidän on määriteltävä uudelleen se tapa, jolla tällaisen pelipuun solmut arvotetaan lapsista alkaen:

Tutut lehdet sekä $\max$- **ja** $\min$-**solmut** arvotetaan kuten aiemmin.

Sattumasolmu C **jonka lapset ovat** $\max$-**solmuja** ("max-sattumasolmu") arvotetaan *max-odotusarvoonsa*

$$\text{expectimax}(C) = \sum_i P(d_i) c_i \tag{13}$$

missä

- indeksi i käy läpi solmun C lapset
- $P(d_i)$ on vaihtoehdon d_i todennäköisyys
- c_i on lapsesta i samalla menetelmällä laskettu arvo.

eli *tulosfunktion* U *odotusarvo*.

Backgammon-esimerkissä nopanheiton C hyvyys heittäjälleen $\max$ on se arvo, jonka hän voi *odottaa* saavansa

1. heittämällä nopat johonkin satunnaiseen silmälukupariin
2. pelaamalla mahdollisimman hyvin tästä saadusta parista alkaen.

Eli lasketaan lasten maksimin odotusarvoa.

$\min$-**sattumasolmu** arvotetaan sekin yhtälöllä (13) expectimin-arvoonsa.

Nyt lapset i ovatkin $\min$-solmuja, ja niiden arvot c_i minimoituja.

Eli lasketaan lasten minimin odotusarvoa.

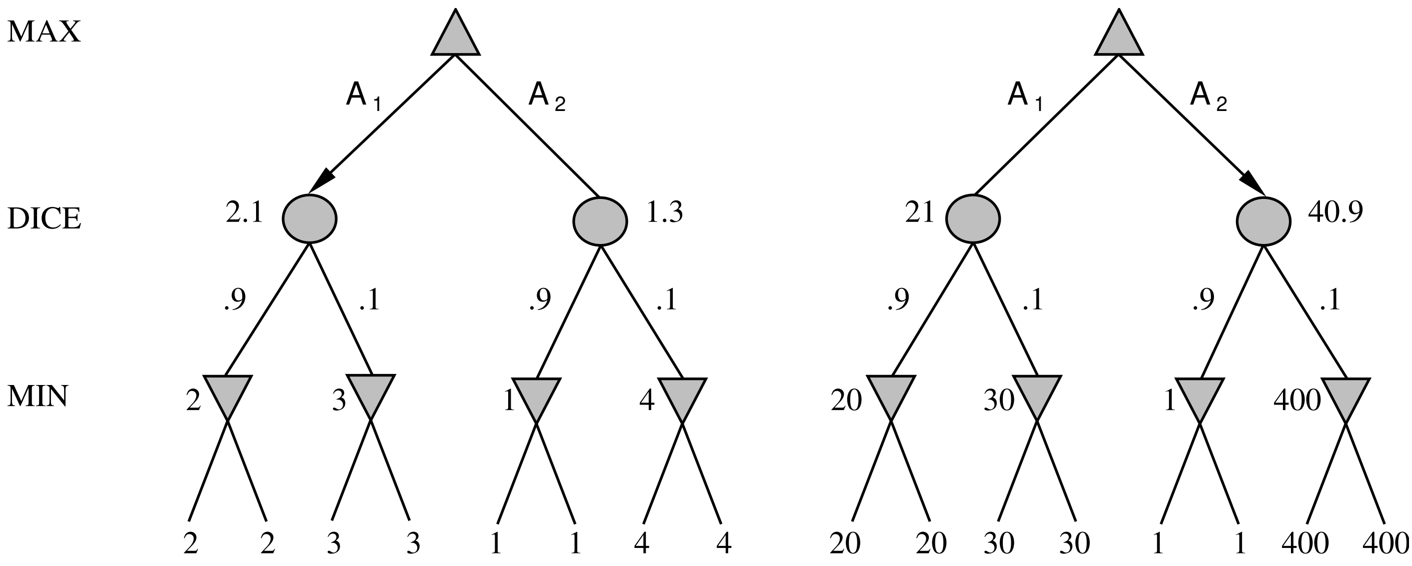
Kun minimax-algoritmia 10 muokataan näin, saadaan *expectiminimax*-algoritmi.

Tämän muokatun algoritmin nopeuttaminen katkaisuehdolla ja evaluointifunktiolla $\textsc{Eval}$ on vaikeampaa kuin luvussa 4.3.1:

- Yhtälön (13) laskemat odotusarvot riippuvat funktion $\textsc{Eval}$ *aidoista numeroarvoista*.

- Yhtälön (12) heikko järjestysehto ei siis enää riitä.

  Kuva 53 osoittaa tämän: kun lehtien arvoja muutetaan järjestys säilyttäen, niin valittu siirto muuttuu myös!



Kuva 53: Järjestys säilyy, suositus ei [RN95, kuva 5.11].

- Sen sijaan funktion EVAL tulisi olla nyt *positiivinen lineaarimuunnos* todellisesta tulosfunktiosta U.
- Sama vaikeus on kaikissa sellaisissa ongelmissa, jotka sisältävät epävarmuustekijöitä.

Olkoon n satunnaissolmujen erilaisten vaihtoehtojen lukumäärä. Backgammonissa $n = 21$.

Silloin expectiminimax-etsintä käy läpi sellaista puuta, jonka haarautumisaste on bn.

Etsinnän aikavaatimukseksi saadaan $O(b^m n^m)$ tutkittua solmua.

Jos etsintää nopeutetaan luvun 4.3.2 alfa-beta-karsinnalla, niin päädytään seuraavaan:

1. Milloin voimme karsia kuvan 52 solmulta C lapsia?
2. Sen isä on $\min$-solmu eli kiinnostunut solmun C arvon *ylärajasta*; sama ajatus näkyi jo kuvassa 49.

3. Tällainen yläraja voidaan selvittää laskematta kaikkia solmun C lapsia, jos tiedetään
   - solmun C lapsien todennäköisyydet $P(d_i)$ (kuten edellä)
   - rajat joiden sisällä funktion $\textsc{Eval}$ (tai U) arvot liikkuvat.

Todennäköisyyksien läsnäollessa karsinta ei kuitenkaan liene yhtä tehokasta kuin luvussa 4.3.2.

Yksityiskohdat sivuutetaan.

5. Tietämyksen esittäminen logiikoilla [RN95, osa III] [Nil98, osa III]

Kuka mitä häh? on filosofian peruskysymys. Nuori Nietzsche kiinnitti huomiota metafysiikan ja kielen yhtäläisyyksiin. Länsimaisen filosofian tai indoeurooppalaisen lauseen perusmuoto on subjekti-predikaatti-objekti. Tästä perustavasta 'kuka mitä häh' -rakenteesta syntyy se, mitä Nietzsche kutsui kansan metafysiikaksi – ajatus siitä, että todellisuus on luotu tämän saman kaavan mukaan. Toisin sanoen, jos ja kun kieli rakenteellisesti yllyttää meitä ilmaisemaan 'kuka mitä häh' -tyyppisiä lauseita, niin kohta olemme jo vakavalla naamalla tekemässä 'Jumala loi maailman' -tyyppistä kosmologiaa. Nietzsche iski kiilaa tähän kansan metafysiikkaan kuuluisalla esimerkillään salamasta. Kuka iskee? Salama. Mitä se iskee? Salaman.

Häh? Miksei riitä sanoa: salamoi!

Nietzschen huomautus synnyttää myös mielenkiintoisen kysymyksen siitä, onko suomi metafyysinen kieli. Vaikka suomen väitelauseet toki taipuvat luontevasti 'kuka mitä häh' -rakenteeseen, kielestämme puuttuu muodollinen subjekti. Emme sano 'it rains' tai 'Es regnet' vaan 'sataa'. On sääli, ettei Suomessa 1800-luvulla ollut yhtään kunnon romanttista kansallisuusmiestä, joka olisi kehittänyt tälle perustalle klassisen suomalaisen filosofian. *Tuomas Nevanlinna*

Jatketaan luvussa 2 aloitettua erilaisten toimijasuunnitelmien tarkastelua *tietämyspohjaisella toimijalla (knowledge-based agent)*:

```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

Algoritmi 12: Tietämyspohjainen toimija [RN95, kuva 6.1].

Muistina on nyt *tietämyskanta (knowledge base)* KB. Operaatiot:

Tell lisää kantaan uuden vastaavalla *tietämyksenesityskielellä (knowledge representation language)* kirjoitetun *lauseen (sentence)*.

Keskitytään luvussa 1.2 mainittuihin formaalin logiikan kieliin:

- lause- ja
- predikaattilogiikkaan.

Ask kysyy kannalta *seuraako (follows)* annettu tietämyksenesityskielen lause kannan nykyisestä sisällöstä.

Tietämyskannan on kyettävä omaehtoiseen *päättelyyn (inference)* vastatessaan annettuun **Ask**-kyselyyn.

Tietokantaan (database) on ohjelmoija viritellyt valmiiksi saantipolut tietojen hakuun ja yhdistelyyn.

Toimijan "syntyessä" sen tietämyskannan alkuarvona on suunnittelijan etukäteen antama (sovellusalueen) *taustatietämys (background knowledge)*.

Tavoitteena on *kuvaileva (deklaratiivinen)* toimijoiden ja järjestelmien suunnittelu- ja toteutustapa:

- Suunnittelija kertoo vain *mitä* päämäärää toimijan tulee tavoitella.
- Toimija päättää itse *miten* se ryhtyy sitä tavoittelemaan.

Tietämyspohjaista toimijaa voidaan tarkastella 3 eri abstraktiotasolla:

Tietämys- eli epistemologinen taso puhuu niistä *merkityksistä*, joita kantaan talletetuilla ja niistä johdetuilla seurauslauseilla meille toimijan ulkopuolisille tarkkailijoille on.

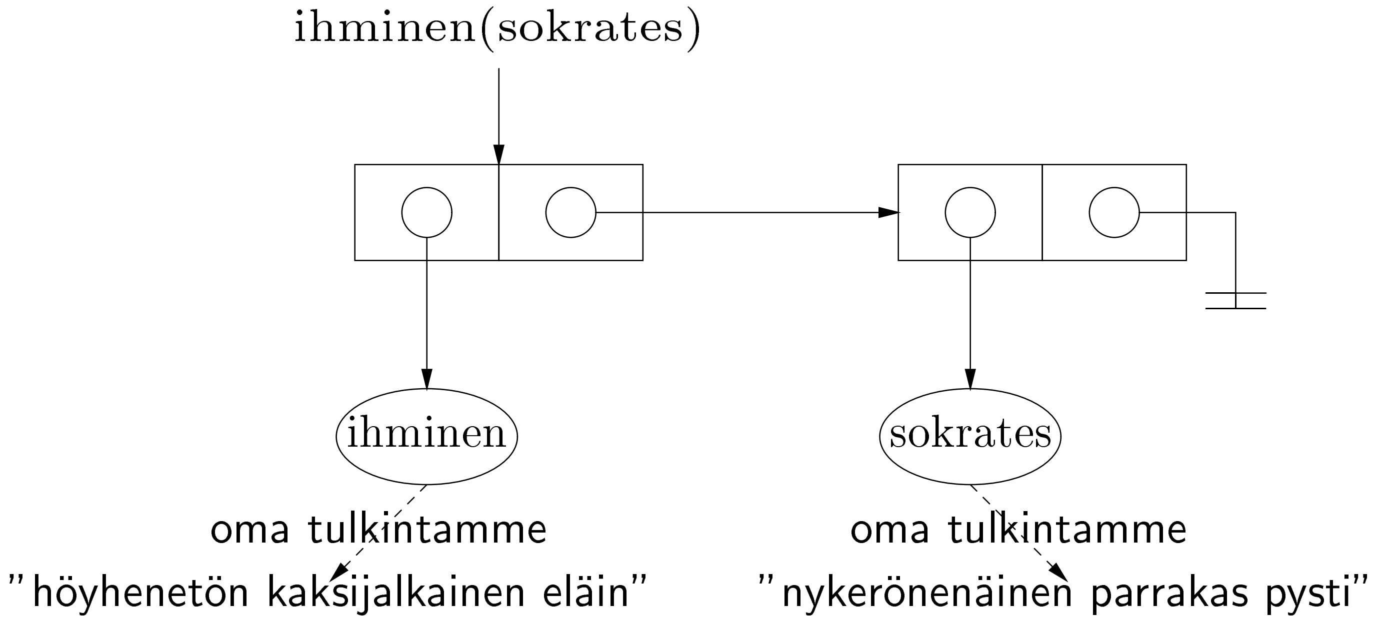
"Tuo ohjelma tietää, että Sokrates on ihminen."

Looginen taso puhuu niistä tietämyksenesityskielellä kirjoitetuista *lauseista*, jotka kantavat näitä merkityksiä.

"Tuon ohjelman kannassa on lause $\mathrm{ihminen}(\mathrm{sokrates})$."

Toteutustaso puhuu siitä, miten näitä lauseita sisältävä tietämyskanta on toteutettu.

"Tuota ohjelmaa suorittavan tietokoneen muistissa on kuvan 54 tietorakenne."



Kuva 54: Scheme-lista `(ihminen sokrates)`.

Tietämyksenesityskielessäkin erotellaan 3 puolta:

Kielioppi eli syntaksi (syntax) antaa ne säännöt, joiden mukaan kielen lauseet rakentuvat.

Esimerkiksi *kielioppisääntö (grammar rule)*

$$\text{Väitelause} \longrightarrow \text{Subjekti Predikaatti} \tag{14}$$

antaa (varsin puutteelliset) "rakennusohjeet" yksinkertaisten englanninkielisten väitelauseiden rakentamiseen.

Esimerkiksi

$$\texttt{computer thinks} \tag{15}$$

on säännön (14) mukainen lause: `"computer"` käy subjektiksi, `"thinks"` predikaatiksi.

Luvussa 1.4 mainittu Chomskyn teoria tarkasteli juuri tällaisia sääntöjärjestelmiä. Ne ovat tulleet myös tietojenkäsittelytieteen vakiotyökaluiksi.

Merkitysoppi eli semantiikka (semantics) kertoo, miten nämä syntaksin mukaiset lauseet voidaan ymmärtää.

Esimerkiksi sääntöön (14) liittyvä *tulkinta (interpretation)* sanoo:

- subjektin tulkinnan antama olio eli tietokone
- suorittaa predikaatin tulkinnan antamaa toimintoa eli ajattelee".

Näin teksti (15). . .

1. ensin *jäsentyy (parse)* kieliopin sääntöjä noudattavaksi lauseeksi
2. joka sitten tulkitaan vastaavaksi "ideaksi", "ajatukseksi", "merkitykseksi", "mielteeksi",. . .

eli vastaavan väitteen sisäiseksi esitykseksi.

Muistamme tämän sisäisen esityksen, ei sen ilmaisseen lauseen sanatarkkaa muotoa.

Monivaihemalli selittää Chomskyn tunnetun esimerkkilauseen

`colourless green ideas sleep furiously.`

1. Jäsennysvaihe 1 onnistuu, koska se on kieliopillisesti oikein.
2. Tulkintavaiheessa 2 muodostetaan vastine

   värittömät vihreät ideat nukkuvat raivokkaasti.
3. Lopuksi tämä vastine yritetään liittää aiempaan tietämykseen:
   - Subjektiin idea ei voi liittää väriin liittyviä määreitä — ikään kuin ideat eivät ole sellaista "tyyppiä" jolla olisi ominaisuus nimeltä 'väri'.

     Lisäksi värimääreet ovat keskenäänkin ristiriitaisia.
   - Predikaattiin nukkua ei kuulu raivokkuus.
   - Subjekti ei osaa tehdä sitä mitä predikaatti sanoo.

   Lauseen mielettömyys selviää siis vasta kun sen vastineen käsitteitä yrittää yhdistellä toisiinsa.

Kieli on *kompositionaalinen* jos sen lauseiden merkitys selviää yhdistelemällä lauseen osien merkityksiä.

Ainakin loogiset kielet ovat kompositionaalisia.

Kieliä tarkasteltaessa erotetaan toisistaan

objektikieli eli se kieli jota tarkastellaan

— Chomskyn esimerkissä englanti, jatkossa eri logiikat

metakieli eli se kieli jossa tarkastelu suoritetaan

— näissä luennoissa suomi.

Näin kierretään "eikö kielen tarkastelu edellytä sen ymmärtämistä":

Oletetaan että *objekti*kielen tarkastelija ymmärtää *meta*kielen.

Luvussa 1.2 mainittu loogikko Alfred Tarski esitti *semanttisen avoimmuuden vaatimuksen:*

Objektikieli älköön puhuko metakieleen kuuluvista käsitteistä!

Tämä erityisesti siksi, että jos looginen objektikieli L pystyisi puhumaan sillä itsellään ilmaistujen lauseiden *totuudesta* — joka on metakielen käsite — niin objektikielessä L voitaisiin silloin ilmaista luvun 1.2 valehtelijan paradoksi "tämä lause on epätotta". . .

Silloin kaatuu yritys määritellä tyhjentävästi mitä objektikielen L lauseiden "totuus" oikein tarkoittaakaan.

Totuus onkin se keskeinen metakäsite, johon tässä esiteltävät *tavallisimmat* logiikat ja niiden pohjalta luodut tietämyksenesityskielet keskittyvät.

Muita mahdollisuuksia voisivat olla

- uskomukset ja niiden asteet
- tietämys ja sen kehittyminen
- todennäköisyydet
- päättelyn relevanssi
- . . .

Todistusteoria (proof theory) ratkoo seuraavaa *loogisten* kielten ongelmaa.

Syntaksi ja semantiikka kertovat, miten oikein muodostetut lauseet tunnistetaan ja ymmärretään vastaaviksi väittämiksi, joiden totuus tai epätotuus "nähdään suoraan".

Tietokone taas pystyy ainoastaan rakentelemaan muistiinsa kuvan 54 kaltaisia tietorakenteita; se ei näe mitään suoraan.

Miten tietokone voisi siis laskea mekaanisesti sille syötteenä annetun lauseen totuuteen liittyviä asioita, vaikkei se itse asiassa ymmärräkään syötettään?

Siten, että tietokoneeseen ohjelmoidaan sellaisia syötelauseiden *syntaktisia* laskutoimituksia, jotka eivät riko niiden *semantiikkaa*!

Näitä "totuuslaskutoimituksia" kutsutaan *päättelysäännöiksi (inference rules)*.

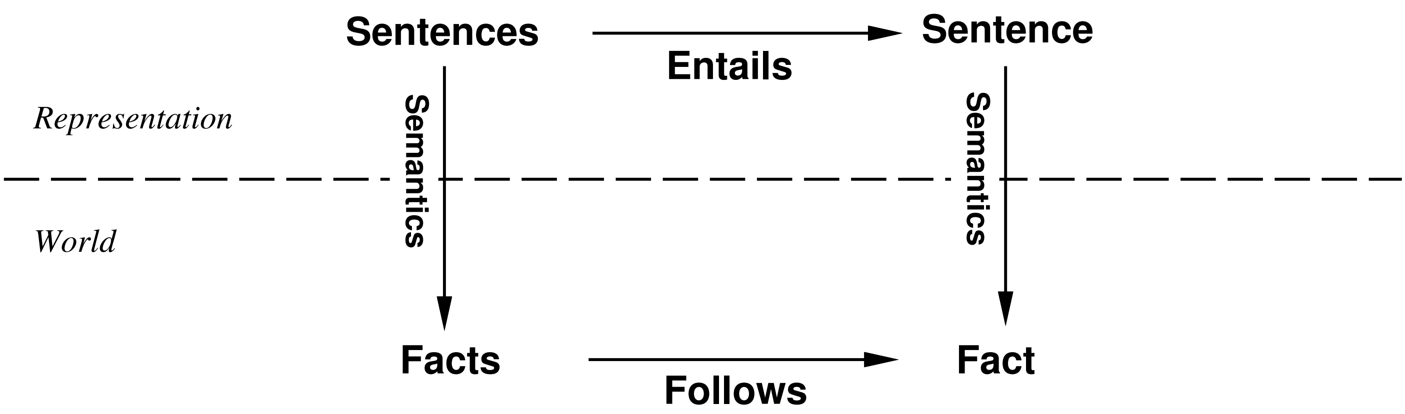
Tarkastellaan tutun Aristotelisen syllogismin (1)–(3) "Scheme-koodausta"

```
(ihminen sokrates)
(jos (ihminen (? . x)) (kuolevainen (? . x)))
---------------------------------------------
(kuolevainen sokrates)
```

jossa tietokone on ohjelmoitu seuraavasti:

"Jos muistissa on ilmaukset A ja `(jos` B C`)`, ja jos ilmaukset A ja B saadaan samoiksi valitsemalla sopivat arvot joillekin niiden aukkopaikoille `(? .` y`)`, niin tuo muistiin myös ilmauksen C kopio, jossa nämä aukkopaikat on täytetty samoilla valinnoilla."

Tämä operaatio *tapahtuu kokonaan fysikaalisia tietorakenteita manipuloimalla* mutta *yhtäpitävästi vastaavien "jos. . . ,niin. . . " -lauseiden merkityksen kanssa*:



Kuva 55: Päättely ja seuraus [RN95, kuva 6.5].

Kuva 55 esittää tilanteen kaksijakoisuutta:

- Maailmassa on faktoja, joista *seuraa* toisia faktoja, sen mukaan "miten maailma toimii".

  Esimerkissä faktasta "Sokrates on ihminen" seuraa fakta "Sokrates on kuolevainen".
- Toimijan muistissa on faktojen fysikaalisia *esityksiä (representations)* joista *on pääteltävissä (entails)* päättelysäännöillä toisia

esityksiä.

Esitysten ja päättelysääntöjen tulisi olla symbolisia (ehkä puutteellisia) vastineita sille, miten maailma toimii, jotta tulokset olisivat maailmassa toimijalle hyödyksi.

Esimerkissä esityksestä `(ihminen sokrates)` lasketaan esitys `(kuolevainen sokrates)`.

Tällaisia kieliä tarkastellaan esimerkiksi seuraavin käsittein:

Oikeellisuus (Soundness): Jos päättelysääntö(joukko) tai sen tietokonetoteutus $\textsc{Ask}$ tuottaa tuloksenaan *vain* sellaisia tietämyksenesityskielen lauseita, joiden tulkinnat seuraavat tietämyskannan KB tulkinnasta ($=$ sen sisältämien lauseiden tulkinnoista), niin se on *oikeellinen (sound, truth-preserving)*.

Tämä minimivaatimus on sama asia kuin että metakielen käsitteen "totuus" semantiikkaa ei rikota.

Todistus (Proof): Lista (tai puu tai muu esitys) oikeellisen päättelymenetelmän annetulla syötteellä suorittamista askelista.

Täydellisyys (Completeness), 1. merkitys: Jos *päättelysääntöjoukko* tai sen tietokonetoteutus $\textsc{Ask}$ pystyy tuottamaan tuloksenaan *kaikki* nuo seurauslauseet, niin se on *täydellinen (complete)*.

Erityisesti Gödel osoitti (luku 1.2) ettei luonnollisten lukujen aritmetiikkaa voi koskaan kokonaan vangita minkäänlaiseen mekaaniseen sääntöjärjestelmään.

Täydellisyys, 2. merkitys: *Lausejoukko tai tietämyskanta* KB on täydellinen, jos siitä pystytään päättelemään kaikki nuo seurauslauseet, kun käytetään jotakin täydellistä päättelysääntöjoukkoa.

Täydellisyys on toivottavaa, mutta ei aina mahdollista; sehän vaatii, että todellisuus (tai sen tarkasteltu osa) voitiin muotoilla *tyhjentävästi* säännöiksi.

Täydellisyys, luvun 3 merkitys: Päättelysäännöt kertovat vain sen, millaisia laskutoimituksia voi yleensä käyttää todistusten tekemiseen, ei sitä miten niitä pitäisi käyttää nykyisessä todistuksessa.

Syntyy *todistuksen etsintäongelma.* Siinä täydellisyys tarkoittaa siis:

Jos tarkasteltavalla lauseella *on* jokin todistus, niin tämä etsintämenetelmä *löytää* sen.

Lisäksi toivotaan (muttei aina saavuteta): vaikka ei, niin menetelmä pysähtyy silti.

Monotonisuus (monotonicity): Päättelyjärjestelmä on *monotoninen (monotonic)* jos uusien lauseiden lisääminen tietämyskantaan KB ei tee sen aikaisemman sisällön pohjalta vedettyjä johtopäätöksiä vääriksi.

Silloin tieto "kasvaa".

Seuraava on esimerkki epämonotonisesta päättelystä:

1. Tipi on lintu.
2. Tavalliset linnut lentävät.
3. *Koska minulla ei ole mitään syytä epäillä Tipiä epätavalliseksi*, niin päättelen sen lentävän.

Jos myöhemmin saammekin tietää Tipin olevan pingviini, täytyy meidän silloin perua johtopäätöksemme 3.

Lause- ja predikaattilogiikka ovat monotonisia.

Niissä johtopäätös 3 voidaan tehdä vasta sitten kun Tipin (epä)tavallisuus on ensin varmistettu ja säännön 2 käyttö näin oikeutettu. Ennen sitä kysyjälle vastataan vain "ei tietoa", jottei oikeellisuus kärsisi.

Toteutuvuus (satisfiability): Annettu tietämyksenesityskielen lause ϕ on *toteutuva (satisfiable)* jos *on olemassa* jokin tulkinta jossa lause ϕ on *tosi*.

Kumoutuvuus (falsifiability): Annettu tietämyksenesityskielen lause ϕ on *kumoutuva (falsifiable)* jos *on olemassa* jokin tulkinta jossa lause ϕ onkin *epätosi*.

Tautologisuus (tautologicity, validity, analyticity): Annettu tietämyksenesityskielen lause ϕ on *tautologia (tautology)* jos se *ei ole kumoutuva*.

Nämä lauseet ovat tosia suoraan nykyisen kielen semantiikan ominaisuuksien perusteella.

Esimerkiksi "Sokrates on kuolevainen *tai* Sokrates *ei* ole kuolevainen" on aina totta sen perusteella miten sanojen "tai" ja "ei" kompositionaalinen semantiikka on määritelty käyttämässämme kielessä: "A tai B" on totta jos ainakin toinen väitteistä A tai B on totta, ja "ei C" on totta jos C on epätotta.

On kuitenkin toisia mahdollisia semantiikkoja, jossa näin ei ole, esimerkiksi semantiikka, jossa on käsitteiden *tosi* ja *epätosi* lisäksi vielä kolmaskin *totuusarvo (truth value)*, vaikkapa *"en tiedä"*. Mutta tämä onkin jo eri kieli!

Ristiriitaisuus (unsatisfiability): Annettu tietämyksenesityskielen lause ϕ on *ristiriitainen (unsatisfiable)* jos se *ei ole toteutuva*.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief 0…1
Fuzzy logic	degree of truth	degree of belief 0…1

Taulukko 3: Kielten ontologisia sitoumuksia [RN95, kuva 6.5].

Tietämyksenesityskieliä voidaan vertailla myös sen suhteen, minkälaisia *ontologisia/epistemologisia sitoumuksia (ontological/epistemological commitments)* niiden määrittelyissä on tehty. (Nämä käsitteet esiteltiin jo luvussa 1.3.) Taulukko 3 vertailee (kohta) tuttujen kielten sitoumuksia.

Luvussa 5.1 esiteltävä (ja jo luvussa 1.3 mainittu) *lauselogiikka (Propositional Logic/Calculus)* on sitoutunut mallintamaan olemassa olevan maailman kokoelmalla esimerkiksi kielioppisäännön (14) mukaisia *jakamattomia* väitelauseita kuten `"sokrates on kuolevainen"`.

Luvussa 5.2 esiteltävä (ja jo luvussa 1.3 mainittu) *[ensimmäisen kertaluvun] predikaattilogiikka ([First Order] Predicate Logic/Calculus)* on taas sitoutunut mallintamaan maailman tällaisten väitelauseiden *osina*: `"Sokrates"` on *olio (object)*, `"kuolevainen"` on *joukko eli yksipaikkainen relaatio (relation)*, ja koko väitelause on niiden yhdistelmä "$\text{Sokrates} \in \text{kuolevainen}$".

5.1. Lauselogiikka

Lauselogiikan syntaksi määritellään antamalla sen *lauseenmuodostussäännöt* induktiivisesti:

Totuusarvovakiot $\mathbf{T}$ ja $\mathbf{F}$ ovat lauseita.

Lausemuuttuja P_i on lause.

Kaikkien lausemuuttujien joukkoa merkitään

$$\mathcal{P} = \{P_i \colon i \in \mathbb{N}\} .$$

Joukko on ääretön, jotta muuttujat eivät loppuisi kesken.

Jokainen kaava on äärellinen, ja sisältää siis vain äärellisen monta näistä muuttujista.

Negaatio $(\neg\phi)$ on lause, jos ϕ on.

Konjunktio $(\phi \wedge \psi)$ on lause, jos sekä ϕ että ψ ovat.

Muita lauseita ei lauselogiikassa ole.

Säästetään kirjoitustyötä ottamalla käyttöön. . .

NIMI	MERKINTÄ	TARKOITTAA
Disjunktio	$(\phi \vee \psi)$	$(\neg((\neg\phi) \wedge (\neg\psi)))$
Implikaatio	$(\phi \Rightarrow \psi)$	$((\neg\phi) \vee \psi)$
Ekvivalenssi	$(\phi \Leftrightarrow \psi)$	$((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi))$

Taulukko 4: Lauselogiikan tavalliset lyhenteet.

ja sopimalla, että sulkuja voi jättää pois laskevana *sidontajärjestyksenä (precedence order)* ilmausten esittelyjärjestys yllä.

Tällöin esimerkiksi $\phi \vee \psi \wedge \chi$ on näkymättömästi sulutettu $(\phi \vee (\psi \wedge \chi))$.

Lauselogiikan semantiikassa aloitetaan *totuusarvojakelusta (truth value distribution)*

$$\mathcal{T} : \mathcal{P} \mapsto \{\mathbf{1}, \mathbf{0}\}$$

missä $\mathbf{1}$ ja $\mathbf{0}$ ovat metakielemme *totuusarvot (truth values)* "tosi" ja "epätosi".

Tämä $\mathcal{T}$ laajennetaan lausemuuttujista kaikkiin lauseisiin samalla induktiolla kuin lauseet määriteltiin (kompositionaalisen semantiikan idea):

Totuusarvovakioiden tulkinnat ovat aina $\mathcal{T}(\mathbf{T}) = \mathbf{1}$ ja $\mathcal{T}(\mathbf{F}) = \mathbf{0}$.

Lausemuuttujien tulkinnat antaa $\mathcal{T}$ suoraan.

Lausemuuttuja on symboli, jonka sovitaan merkitsevän jotakin *kokonaista väitelausetta* — siis sellaista ilmausta Ψ, että "väite Ψ on (epä)tosi" on ymmärrettävää metakieltämme.

Esimerkiksi lausemuuttujan P_1 voitaisiin sopia merkitsevän väitelausetta `"tietokone ajattelee"`, koska "väite 'tietokone ajattelee' on tosi" on ymmärrettävää suomenkieltä.

Sen osa `"tietokone"` ei enää käy lausemuuttujan merkitykseksi, koska "väite 'tietokone' on tosi" ei enää ole ymmärrettävissä.

Erilaiset totuusarvojakelut $\mathcal{T}$ esittävät siis erilaisia käsityksiä tällaisista "peruslauseista".

- Jos tietämyskannan lausemuuttuja P_2 viittaa havainnoitavaan seikkaan kuten "edessä olevan auton jarruvalot palavat", niin $\mathcal{T}(P_2)$ kertoo onko vastaava havainto saatu eli palavatko ne vaiko eivät.

  Nämä lausemuuttujat saavat arvonsa havainnoista tai havaintosarjoissa ilmenneistä säännönmukaisuuksista (kuten "edessä oleva auto vilkuttaa oikealle").
- Jos P_3 viittaa toimijan sisäiseen seikkaan kuten "minun on jarrutettava", niin $\mathcal{T}(P_3)$ kertoo onko seikka "tiedostettu" eli onko minun jarrutettava vai ei.

  Nämä lausemuuttujat saavat arvonsa tietämyskannan säännjöistä havaintomuuttujien arvojen pohjalta: esimerkiksi sääntö "jos P_2

niin P_3" on tarpeen, jotta toimija voisi valita seuraavaksi teokseen jarruttamisen.

Negaatioiden ja konjunktioiden tulkinnat saadaan määräämällä ensin induktiivisesti niiden osien totuusarvot ja käyttämällä sitten vastaavaa *totuustaulua (truth table)*:

$\mathcal{T}(\phi)$	$\mathcal{T}((\neg\phi))$
0	**1**
1	**0**

$\mathcal{T}(\phi)$	$\mathcal{T}(\psi)$	$\mathcal{T}((\phi \wedge \psi))$
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

Taulukko 5: Negaation ja konjunktion totuustaulut.

Näin "$(\neg\phi)$" voidaan lukea "ei $\mathcal{T}(\phi)$", ja "$(\phi \wedge \psi)$" taas "$\mathcal{T}(\phi)$ ja $\mathcal{T}(\psi)$".

Taulukon 4 lyhenteiden totuustaulut voidaan sitten laskea käyttäen taulukkoa 5:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Taulukko 6: Lyhenteiden totuustaulut [RN95, kuva 6.9].

Implikaatio $P \Rightarrow Q$ edustaa muotoa "jos P niin Q" olevia väitteitä.

Nämä väitteet on jo nähty tärkeiksi johtopäätöksiä tehtäessä.

Implikaation sarake taulukossa 6 näyttää, että implikaatio on epätosi vain silloin kun sen *etujäsen (antecedent)* P on tosi mutta *takajäsen (consequent)* Q epätosi.

Yksi lukutapa lauseelle $P \Rightarrow Q$ onkin väite

$$\mathcal{T}(P) \leq \mathcal{T}(Q) \tag{16}$$

missä on sovittu, että $\mathbf{0} < \mathbf{1}$.

Tämä implikaatio *ei siis edusta syy-seuraussuhdetta "P aiheuttaa että Q"* (vaan on ns. *materiaalinen* implikaatio).

Jokaiseen (lause)logiikan lauseeseen ϕ voidaan liittää niiden tulkintojen (eli totuusarvojakeluiden) $\mathcal{T}$ joukko, jotka toteuttavat lauseen ϕ:

$$\mathrm{Mod}(\phi) = \{\mathcal{T} : \mathcal{T}(\phi) = \mathbf{1}\} \tag{17}$$

Näitä tulkintoja sanotaan lauseen ϕ *malleiksi (models)*.

Kukin malli voidaan ymmärtää yhdeksi sellaiseksi asiaintilaksi, joka saattaa lauseen ϕ mukaan vallita todellisuudessa. Kuva 56 osoittaa, mitä joukko-opillisia operaatioita eri loogiset operaatiot eli *konnektiivit* vastaavat.

Nyt kuvassa 55 *lause ψ on luvallisesti pääteltävissä lauseesta ϕ* täsmälleen silloin kun

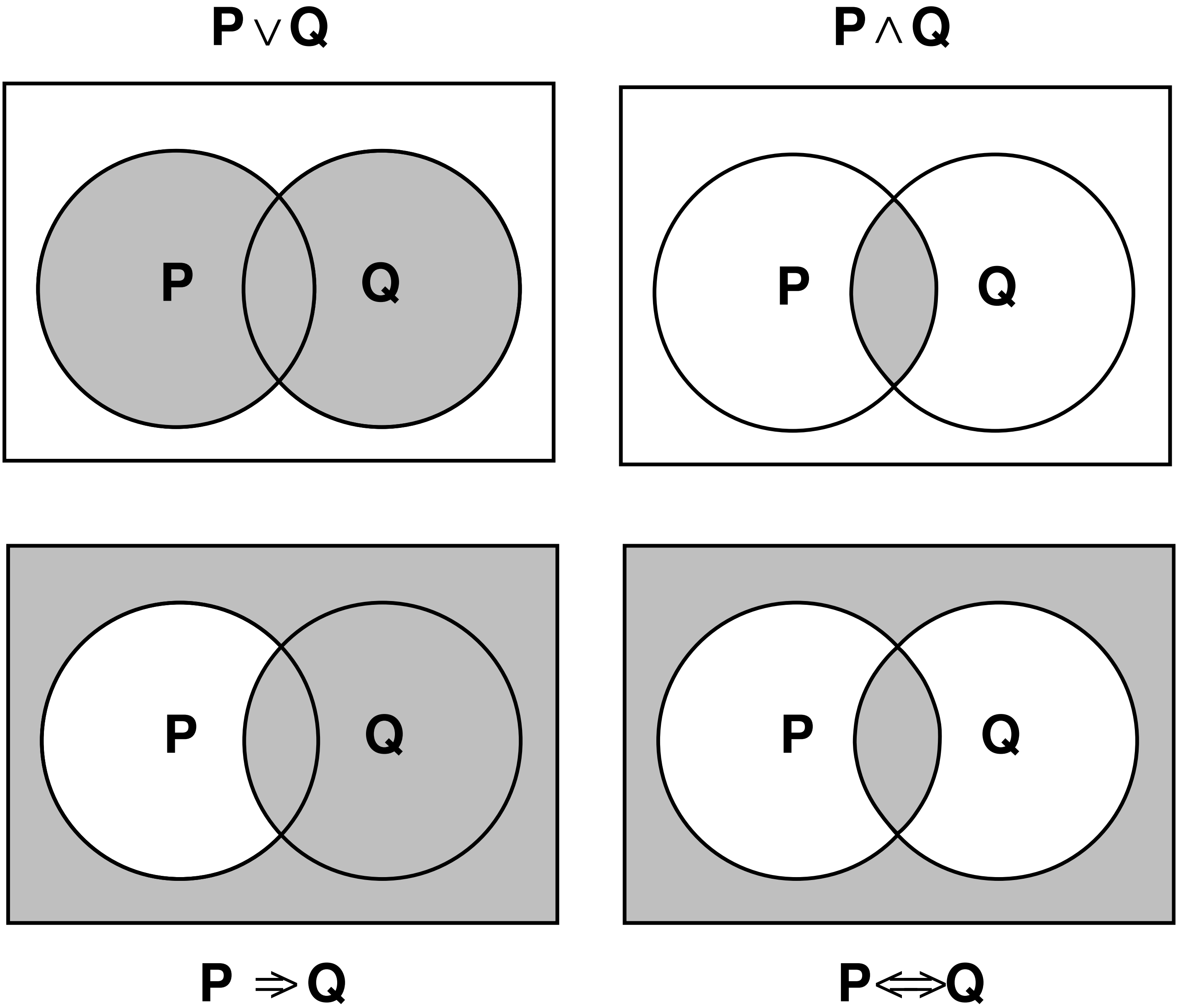
$$\mathrm{Mod}(\phi) \subseteq \mathrm{Mod}(\psi). \tag{18}$$

Nyt ϕ on *vahvempi* väite kuin ψ siitä, miten maailma todellisuudessa on tai ei ole, koska se sulkee pois enemmän vaihtoehtoisia mahdollisia tulkintoja.

Jos siis tiedämme, että "aurinko paistaa ja vettä sataa", niin me saamme päätellä siitä sekä "aurinko paistaa" että "vettä sataa".

Implikaatioiden $P \Rightarrow Q$ tärkeys perustuukin seuraavaan järkeilyyn:

$P \Rightarrow Q$ on tautologia	joss	$\mathrm{Mod}(P) \subseteq \mathrm{Mod}(Q)$ [kuvan 56 vasen alakulma]
	joss	takajäsen Q on pääteltävissä etujäsenestä P [yhtälö (18)]
	joss	kun todistusaskel vie lauseesta P lauseeseen Q, niin totuus ei vähene [yhtälöstä (16)].



Kuva 56: Lauselogiikka ja joukko-oppi [RN95, kuva 6.12].

Tietämyskannan KB (ja yleisemmin minkä tahansa lausejoukon eli *teorian (theory)*) malleiksi otetaan ne tulkinnat, jotka toteuttavat samalla kertaa *kaikki* sen sisältämät lauseet eli

$$\begin{aligned}\mathrm{Mod}(KB) &= \bigcap_{\phi\in KB} \mathrm{Mod}(\phi) \\ &= \mathrm{Mod}\left(\bigwedge_{\phi\in KB} \phi\right)\end{aligned}$$

missä jälkimmäinen askel on kuvan 56 oikea yläkulma (joskin se on syntaktisesti oikein vain äärellisille KB).

Toisin sanoen KB on massiivinen *ja*-lause kaikista operaatiolla $\textsc{Tell}$ siihen syötetyistä lauseista. Kukin näistä lauseista rajaa pois joitakin maailman tiloja mahdottomina.

Operaatio $\textsc{Ask}$ taas selvittää päteekö annettu kysymyslause ψ *kaikissa* niissä tiloissa, jotka ovat vielä pysyneet mahdollisina. Se voitaisiin *periaatteessa* toteuttaa selvittämällä, onko

$$\left(\bigwedge_{\phi \in KB} \phi\right) \Rightarrow \psi$$

tautologia. Se taas onnistuisi laskemalla vastaava totuustaulu, ja tarkistamalla, että sen jokainen rivi päätyisi arvoon $\mathbf{1}$.

Tämä olisi kuitenkin eksponentiaalista tietämyskannan KB sisältämien lausemuuttujien lukumäärän m suhteen, koska jo itse taulu sisältäisi 2^m riviä, jotka kaikki pitäisi tarkistaa.

Sukulaiskysymys "onko annettu lauselogiikan kaava χ toteutuva" — eli tuleeko sen totuustauluun yhtään kertaa arvoa $\mathbf{1}$, eli ns. SAT-ongelma — olikin se alkuperäinen ongelma, josta Stephen Cook kehitti vuonna 1971 NP-täydellisyyden käsitteen (luku 1.2) ongelmien laskennallisen vaikeuden luonnehtimiseen. (Artikkelin otsikkokin oli "The Complexity of Theorem Proving Procedures".)

Intuitiivisesti lauselogiikan toteutuvuusongelma kuuluu sellaisiin ongelmiin, jotka ovat helpoimpia niistä ongelmista, jotka ovat luultavasti tietokoneille vaikeita.

Tätä vaikeutta yritetään välttää – sekä ihmisen että koneen takia – kehittämällä joukko oikeellisia päättelysääntöjä ja *etsimällä halutun tuloksen todistusta annetuista lähtötiedoista*, esimerkiksi seuraavasti:

- Etsinnän tila $\mathcal{E}$ on KB ja siitä tähän mennessä päätellyt lauseet.

- Tilaa päivitetään valitsemalla siitä lauseet $\alpha_1, \ldots, \alpha_n \in \mathcal{E}$ jotka sopivat johonkin päättelysääntöön

$$\frac{\alpha_1, \ldots, \alpha_n}{\beta} \tag{19}$$

ja lisäämällä säännön antama tuloslause β tilaan $\mathcal{E}$.

sääntö	nimi
$\dfrac{\alpha \Rightarrow \beta \qquad \alpha}{\beta}$	modus ponens
$\dfrac{\alpha_1 \wedge \ldots \wedge \alpha_n}{\alpha_i}$	konjunktion poisto
$\dfrac{\alpha_1, \ldots, \alpha_n}{\alpha_1 \wedge \ldots \wedge \alpha_n}$	konjunktion tuonti
$\dfrac{\alpha_i}{\alpha_1 \vee \ldots \vee \alpha_n}$	disjunktion tuonti
$\dfrac{\neg\neg\alpha}{\alpha}$	kaksoisnegaation poisto
$\dfrac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$	resoluutio

Taulukko 7: Lauselogiikan päättelysääntöjä.

- Odotellaan ilmestyykö haluttu seurauslause ψ tilaan $\mathcal{E}$.
- Variaatioita:
  - Jos etsintää suoritetaankin taaksepäin (vrt. luku 3.1.6), niin säännöt (19) luetaankin alhaalta ylös ohjeina miten johtopäätös β voidaan purkaa oletuksiinsa $\alpha_1, \ldots, \alpha_n$.
  - Aloitetaankin tilasta $\mathcal{E} = KB \cup \{\neg\psi\}$ ja odotellaan ilmestyykö $\mathbf{F}$. Jos ilmestyy, niin ei ole yhtään sellaista mallia, jossa koko tietämyskanta KB olisi totta mutta seurauslause ψ ei.
    Tämä on *todistamista ristiriidan kautta (proof by contradiction):* tuloksen kieltäminen on ristiriidassa oletusten kanssa.

Eräs käytännössä tärkeä täyden lauselogiikan osajoukko koostuu implikaatioista

$$P_1 \wedge P_2 \wedge P_3 \wedge \ldots \wedge P_n \Rightarrow Q,$$

joissa kukin P_i ja Q ovat lausemuuttujia.

Niitä kutsutaan *Hornin lauseiksi* keksijänsä Alfred Hornin mukaan.

- Kun $n = 0$ on kyseessä lause $\mathbf{T} \Rightarrow Q$ eli pelkkä fakta Q.
- Kun $Q = \mathbf{F}$ on kyseessä lause $P_1 \wedge P_2 \wedge P_3 \wedge \ldots \wedge P_n \Rightarrow \mathbf{F}$ eli kieltolause $\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \ldots \vee \neg P_n$.
- Kun sekä $n = 0$ että $Q = \mathbf{F}$ on kyseessä lause $\mathbf{T} \Rightarrow \mathbf{F}$ joka on ristiriitainen.

Niillä ilmaistaan usein esiintyviä sääntöjä muotoa "jos tietämyskannassa on (tai siitä on voitu aiemmin päätellä) faktat $P_1, \ldots, P_n$, niin siitä voidaan nyt päätellä myös fakta Q".

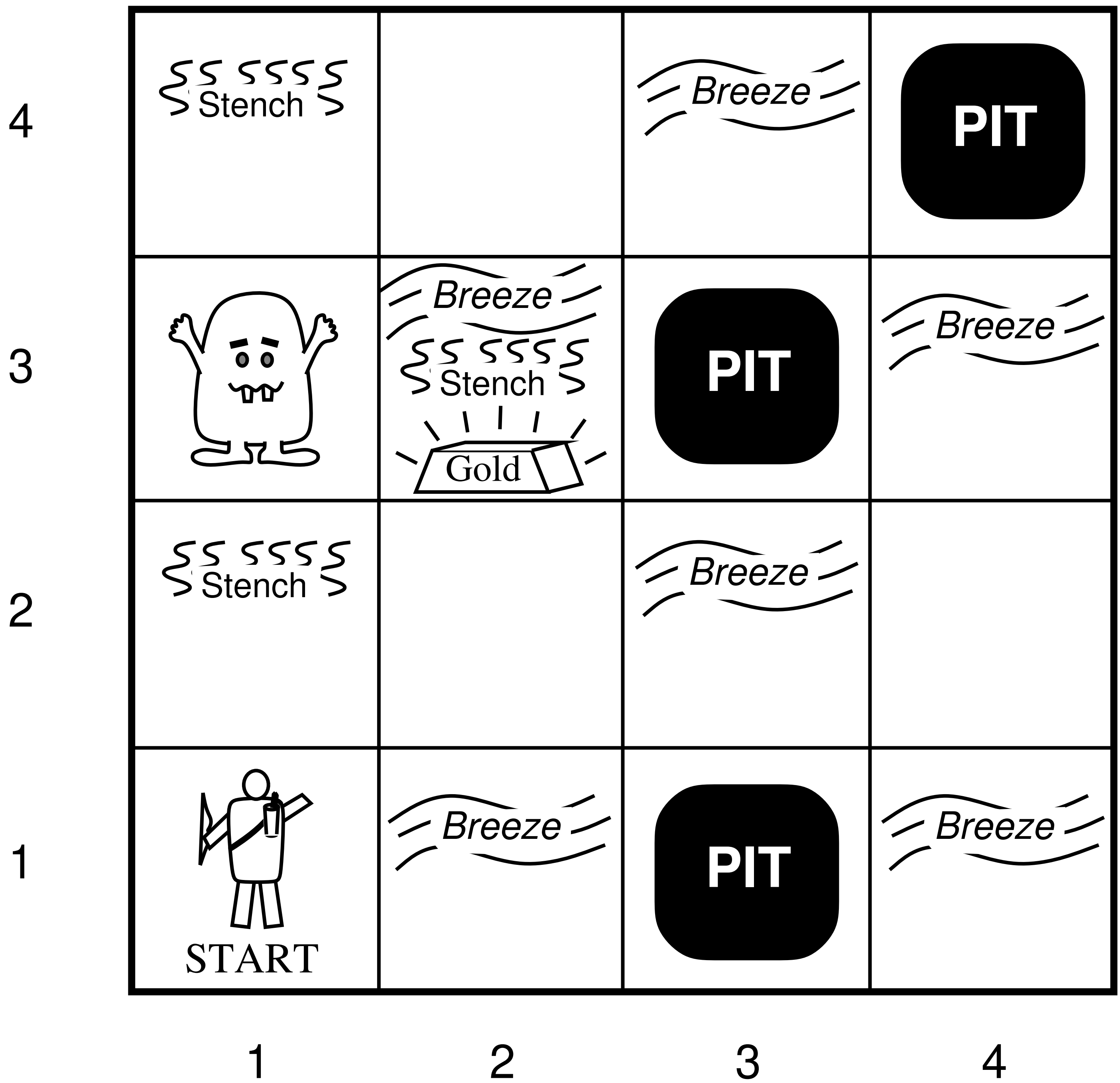
Esimerkiksi taulukon 7 resoluutiosäännön käyttö on helppoa, koska jokaisessa Hornin lauseessa on vain 1 *positiivinen (= ei negatoitu)* lausemuuttuja. Silloin vaihtoehtoja on vähemmän ja etsintä suunnatumpaa.

Usein tietämyskanta KB koostuukin pelkästään Hornin implikaatioista ja faktoista.

Silloin kysymys "seuraako lausemuuttuja R tietämyskannasta KB" voidaan selvittää ristiriitamenetemällä:

- Muodostetaan vastaväite $\neg R$.
  Se on kieltolause.
- Kun sovelletaan kieltolausetta sääntöön tai faktaan resoluutiolla, saadaan tulokseksi uusi kieltolause.
- Näin yritetään saada muodostettua ristiriitainen Hornin lause.

Tarkastellaan esimerkkinä (lause)logiikan käytöstä tietämyksen esittämisessä ja päättelyssä seuraavaa pientä seikkailupeliä:



Kuva 57: Wumpus-maailma [RN95, kuva 6.2].

Liitetään joka ruutuun $q = [\text{sarake}, \text{rivi}]$ seuraavat lausemuuttujat:

- S_q eli "ruudussa q löyhkää (**S**tench)".
- B_q eli "ruudussa q vetää (**B**reeze)".
- W_q eli "ruudussa q on hirviö **W**umpus".

- P_q eli "ruudussa q on kuoppa (**P**it)".
- O_q eli "ruutu q on turvallinen (**OK**)".

Pelissä pätevät seuraavat yleiset periaatteet, joilla seikkailijan tietämyskanta KB on alustettu:

- Ruudussa löyhkää jos ja vain jos Wumpus on jossakin naapuriruudussa.

  Merkitköön $\mathrm{nri}(q)$ ruudun q kaikkien naapuriruutujen joukkoa (vain pysty- tai vaakasuuntaan, ei vinottain).

  Sovitaan myös, että $q \in \mathrm{nri}(q)$; jos Wumpus löyhkää kaukaa, löyhkää se myös läheltäkin!

  Silloin tämä periaate voidaan esittää seuraavalla kokoelmalla lauselogiikan lauseita:

  $$S_q \Leftrightarrow \bigvee_{p \in \mathrm{nri}(q)} W_p \tag{20}$$

  jokaiselle ruudulle q.
- Ruudussa vetää jos ja vain jos jossakin naapuriruudussa on kuoppa:

  $$B_q \Leftrightarrow \bigvee_{p \in \mathrm{nri}(q)} P_p \tag{21}$$

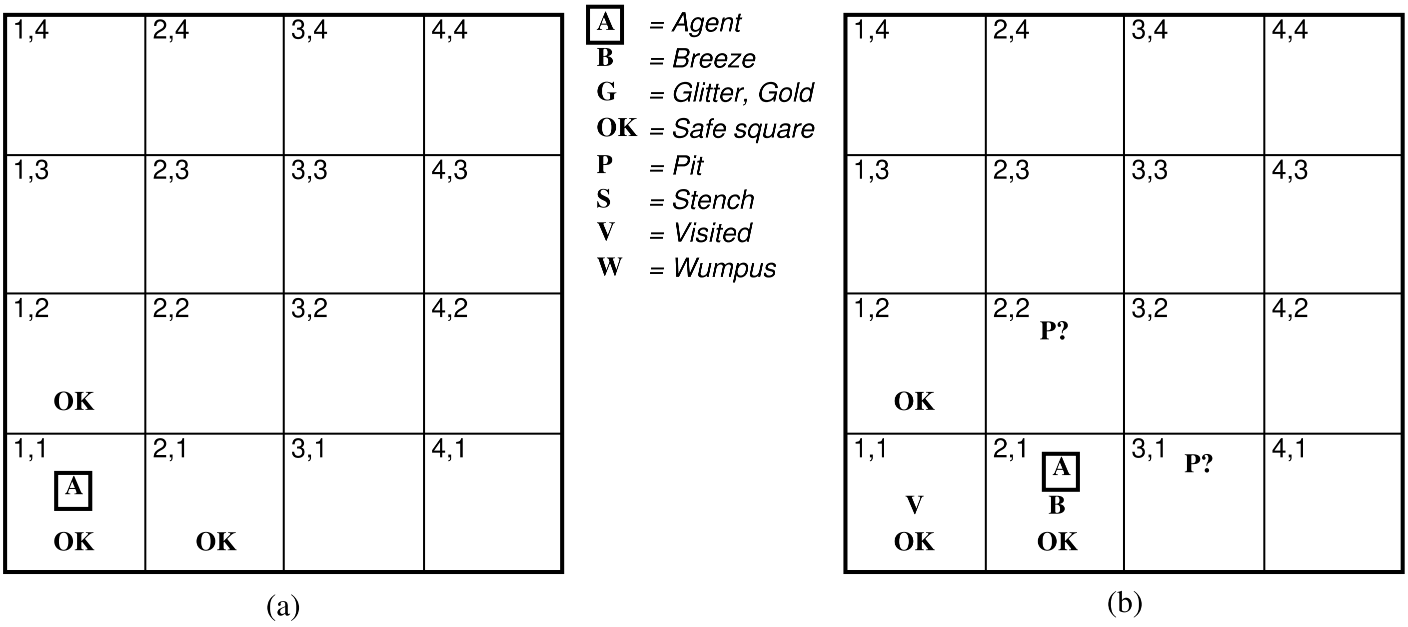
  jokaiselle ruudulle q.
- Wumpuksen tai kuopan ruuduissa kuolee, muualla ei:

  $$O_q \Leftrightarrow \neg W_q \wedge \neg P_q. \tag{22}$$

- Tavoitteena on lähteä ruudusta $[1, 1]$, etsiä kulta jos mahdollista, ja palata hengissä takaisin lähtöruutuun.

  Lauselogiikka ei vielä pysty *suosittelemaan tekoja*, ainoastaan vastaamaan, onko annettu ruutu turvallinen vai ei.

  Predikaattilogiikka on rikkaampi: siellä pystytään kysymään (ja vastaamaan) "onko olemassa turvallinen vielä vierailematon ruutu?"



Kuva 58: Vaellus wumpus-maailmassa 1 [RN95, kuva 6.3].

Peli alkaa kuvasta 58 (a) ajanhetkellä $t = 0$. Alkuruudussa $q = [1, 1]$ havaintolause $percept$ on "ei tunnu löyhkää eikä vetoa":

$$\textsc{Make-Percept-Sentence}(percept, 0) = \neg S_{[1,1]} \wedge \neg B_{[1,1]} \quad (23)$$

joka lisätään tietämyskantaan KB. Sitten aletaan päätellä mitä ruutuja nyt tiedetään turvallisiksi.

Lause (20) on muotoa

$$S_{[1,1]} \Leftrightarrow W_{[1,1]} \vee W_{[1,2]} \vee W_{[2,1]}$$

eli ekvivalenssin määritelmän nojalla

$$\begin{gathered}\left(S_{[1,1]} \Rightarrow W_{[1,1]} \vee W_{[1,2]} \vee W_{[2,1]}\right) \\ \wedge \\ \left(W_{[1,1]} \vee W_{[1,2]} \vee W_{[2,1]} \Rightarrow S_{[1,1]}\right)\end{gathered}$$

josta konjunktion poistolla ja implikaation määritelmällä saadaan

$$\neg\left(W_{[1,1]} \vee W_{[1,2]} \vee W_{[2,1]}\right) \vee S_{[1,1]}. \quad (24)$$

Toisaalta konjunktion poisto antaa lauseesta (23) $\neg S_{[1,1]}$, josta resoluutio lauseen (24) kanssa antaa

$$\neg \left(W_{[1,1]} \vee W_{[1,2]} \vee W_{[2,1]} \right) \tag{25}$$

Seuraavat *De Morganin säännöt* voidaan osoittaa tautologioiksi (vaikkapa totuustauluin):

$$\neg(\alpha \; \begin{matrix} \wedge \\ \vee \end{matrix} \; \beta) \Leftrightarrow (\neg\alpha) \; \begin{matrix} \vee \\ \wedge \end{matrix} \; (\neg\beta). \tag{26}$$

Näin ollen lauseesta (25) saadaan päätellä (välivaiheet sivuuttaen)

$$\neg W_{[1,1]} \wedge \neg W_{[1,2]} \wedge \neg W_{[2,1]}. \tag{27}$$

Samalla tavalla lauseesta (21) voidaan päätellä

$$\neg P_{[1,1]} \wedge \neg P_{[1,2]} \wedge \neg P_{[2,1]}. \tag{28}$$

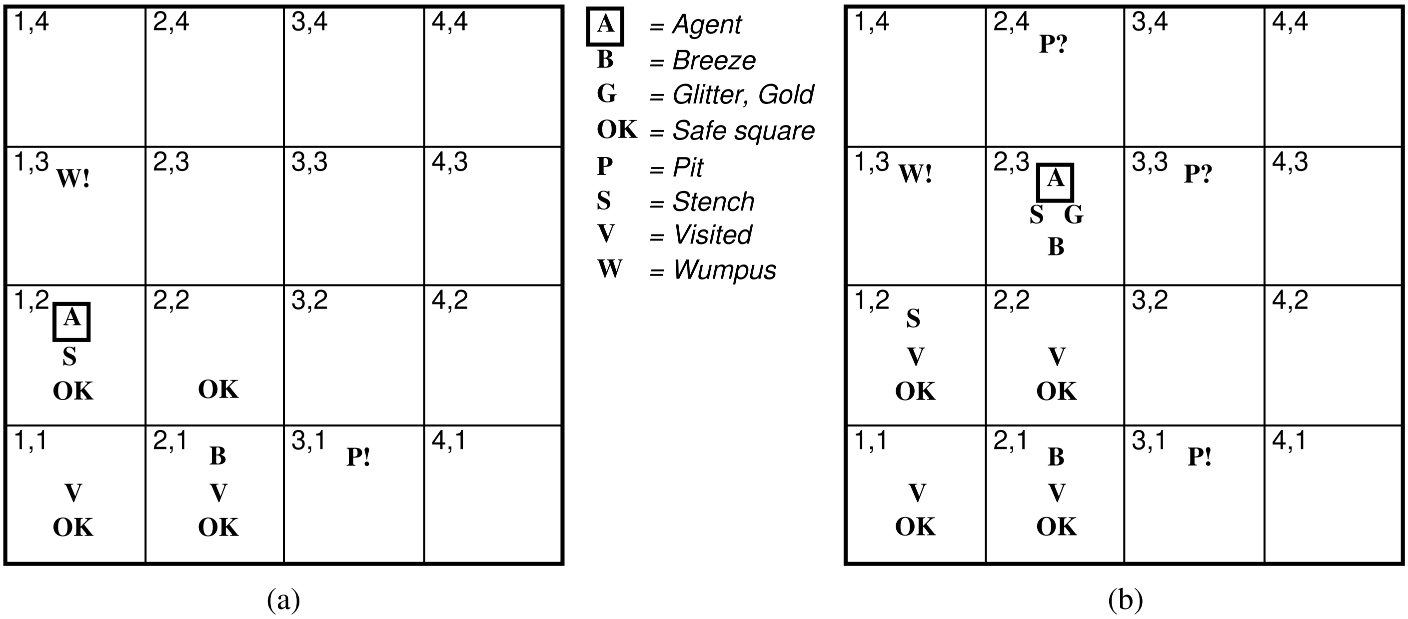
Lauseista (22), (27) ja (28) saadaan lopulta etsityt tulokset $O_{[1,1]}$, $O_{[1,2]}$ ja $O_{[2,1]}$.

Kuvassa 58 (b) on valittu (jollakin tietämyskannan KB ulkopuolisella mekanismilla) seuraavaksi ruuduksi $[2,1]$. Siellä

$$\textsc{Make-Percept-Sentence}(\mathit{percept}, 1) = \neg S_{[2,1]} \wedge B_{[2,1]},$$

josta voidaan päätellä $P_{[2,2]} \vee P_{[3,1]}$ — lause (28) sulkee pois vaihtoehdon $P_{[1,1]}$.

Nyt ei voida päätellä mitään lausemuuttujien $O_{[2,2]}$ ja $O_{[3,1]}$ (epä)totuudesta, joten kuvassa 59 (a) on palattu kokeilemaan ainoaa jäljellä olevaa vielä vierailematonta turvalliseksi pääteltävissä olevaa ruutua $[1,2]$.



Kuva 59: Vaellus wumpus-maailmassa 2 [RN95, kuva 6.4].

Siellä havaintolause on $S_{[1,2]} \wedge \neg B_{[1,2]}$ josta voidaan päätellä seuraavaa:

- $S_{[1,2]}$ antaa $W_{[1,1]} \vee W_{[1,2]} \vee W_{[1,3]} \vee W_{[2,2]}$ lauseesta (20).
- $\neg W_{[1,1]}$ ja $\neg W_{[1,2]}$ on jo aiemmin päätelty lauseesta (27).
- $\neg W_{[2,2]}$ olisi voitu päätellä kuvasta 58 (b).
- Siis jää $W_{[1,3]}$, eli hirviö löytyi.
- Toisaalta $\neg B_{[1,2]}$ antaa $\neg P_{[1,1]} \wedge \neg P_{[1,2]} \wedge \neg P_{[1,3]} \wedge \neg P_{[2,2]}$ lauseesta (21).
- Silloin ainoa mahdollisuus on $P_{[3,1]}$, eli kuoppakin löytyi.

Näin $O_{[2,2]}$ saadaan pääteltyä, ja kuvassa 59 (b) tätä tietoa käytetään menestyksellisesti.

Esimerkki muistuttaa luvun 4.2 rajoiteongelmia:

- Molemmissa paikallisesti hankitulla lisätiedolla eliminoidaan osa vielä tällä hetkellä tuntemattomista mahdollisuuksista.
- Rajoiteongelmissa sallitut eliminointiperiaatteet seuraavat kulloinkin ratkaistavasta ongelmasta, nyt periaatteet ovat yleisiä ja ongelma kuvattu loogisesti.

5.2. Predikaattilogiikka

Tarkastellaan seuraavaksi *ensimmäisen kertaluvun predikaattilogiikkaa (First-Order Predicate Logic/Calculus, FOPC)*.

- Predikaattilogiikka on keskeinen tietämyksenesityskieli, koska sitä heikommat kielet (kuten luvun 5.1 lauselogiikka) eivät enää riitä esitettävän tiedon monimutkaistuessa.
- On olemassa kieliä, jotka säilyttävät enemmän tietämyksen "rakennetta" kuten sen *käsitehierarkioita* "hauki on sellainen kala, joka. . . ":
  - kehyksiä (frames)
  - semanttisia verkkoja (semantic networks)
  - periytymistä (inheritance) joka päätyi myöhemmin ohjelmointikieliinkin määrittelytietämyksen esitystavaksi
  - . . .

  Sama tietämys voidaan kuitenkin yleensä esittää myös predikaattilogiikalla, joskin kömpelömmin.
- On ehdotettu rikkaampia logiikkoja, joissa olisi lisäksi esimerkiksi

  epämonotonisia piirteitä
  kuten luvun 5 oletussääntöjä "tavalliset linnut lentävät"
  [oletuslogiikka (default logic)]

  aikaa ja sen kulumista koskevia väitteitä
  kuten "huomenna hän tulee"
  [aikalogiikat (temporal logic)]

  tietämystä koskevia väitteitä
  kuten "minä uskon että kaikki muut tietävät jotakin sellaista jota minä en tiedä"
  [episteemiset logiikat (epistemic logics)]

  mutta ei vielä ole yksimielisyyttä *miten* nämä hyödylliset lisäpiirteet tulisi tietämyksenesityskieliin lisätä.
- Predikaattilogiikkaa on tutkittu kauan ja perusteellisesti.
- Predikaattilogiikka riittää moniin tilanteisiin.

- Muut tietämyksenesityskielet sisältävät usein predikaattilogiikan tai suuren osan siitä.

Predikaattilogiikka on selvästi rikkaampi kieli kuin luvun 5.1 lauselogiikka. Sen määrittelykin on selvästi mutkikkaampaa.

5.2.1. Syntaksi

Predikaattilogiikan syntaksiin kuuluu 3 erillistä *symboliluokkaa*:

Muuttujasymbolit $\mathcal{X} = \{x_i \colon i \in \mathbb{N}\}$.
Funktiosymbolit $\mathcal{F} = \{f_j \colon j \in \mathbb{N}\}$.
Predikaattisymbolit $\mathcal{P} = \{p_k \colon k \in \mathbb{N}\}$.

Kuhunkin funktio- ja predikaattisymboliin s liittyy vielä *paikkaluku (arity)* $\mathrm{ar}(s) \in \mathbb{N}$, joka kertoo sen argumenttien määrän.

0-paikkaista funktiosymbolia f (eli $\mathrm{ar}(f) = 0$) kutsutaan myös *vakioksi (constant)*.

- Kukin muuttujasymbolin esiintymä x_i on *termi (term)*.
- Jos f on funktiosymboli ja $t_1, \ldots, t_{\mathrm{ar}(f)}$ ovat termejä, niin myös $f(t_1, \ldots, t_{\mathrm{ar}(f)})$ on termi.
  Jos f on vakio, niin termistä $f()$ voidaan jättää turhat tyhjät sulut pois.
- Muita termejä ei ole.
- Jos p on predikaattisymboli ja $t_1, \ldots, t_{\mathrm{ar}(p)}$ ovat termejä, niin $p(t_1, \ldots, t_{\mathrm{ar}(p)})$ on *[atomi]kaava ([atomic] formula)*.
- Jos ϕ on kaava ja x_i muuttujasymboli, niin $(\exists x_i \phi)$ on myös kaava.
  Merkki '$\exists$' on nimeltään *eksistenssi- eli olemassaolokvanttori (existential quantifier)*.
- Kuten luvun 5.1 lauselogiikassakin: jos ϕ ja ψ ovat kaavoja, niin $(\neg\phi)$ ja $(\phi \wedge \psi)$ ovat myöskin kaavoja.
- Muita kaavoja ei ole.

Otetaan jälleen käyttöön taulukon 4 lyhenteet, joihin lisätään vielä lyhenne

$$(\forall x_i \phi) \text{ kaavalle } (\neg \exists x_i (\neg \phi)).$$

Merkki '$\forall$' on nimeltään *universaali- eli kaikkikvanttori (universal quantifier)*.

Sovitaan lisäksi lyhennysmerkintä '$(Qx_1, x_2, x_3, \ldots, x_m\phi)$' toistuvalle kvanttorille Q, eli kaavalle $(Qx_1(Qx_2(Qx_3(\ldots(Qx_m\phi)\ldots))))$.

Poistetaan jälleen sulkuja sopimalla samat presedenssisäännöt kuin luvussa 5.1. Lisäksi sovitaan, että kvanttorit sitovat yhtä voimakkaasti kuin '$\neg$'. Siis

$$\exists xp(x) \land q(x) \text{ suluttuu } (\exists xp(x)) \land q(x).$$

Usein kvanttorin Q halutaankin ulottuvan *pienimmän* mahdollisen *alikaavan (subformula)* sijasta *suurimpaan* ϕ. Tähän käytetään notaatiota '$Qx.\phi$'. Siis

$$\exists x.p(x) \land q(x) \text{ suluttuu } \exists x(p(x) \land q(x)).$$

Usein predikaattisymbolien joukossa $\mathcal{P}$ on erikoinen *yhtäsuuruus (equality)* $t_1 = t_2$ termeille t_1 ja t_2. Periaatteessa '$=$' voidaan käsitellä kuten mikä tahansa muu predikaattisymbolikin, mutta käytännössä se on niin yleinen, että usein se käsitellään erikseen sekä syntaksissa etta semantiikassa.

Yhtäsuuruuden avulla voidaan määritellä uusi kvanttorilyhenne '$\exists! x \,\mathrm{presidentti}(x)$' tarkoittamaan kaavaa

$$\exists x(\underbrace{\mathrm{presidentti}(x) \land}_{1} \forall y. \underbrace{\mathrm{presidentti}(y) \Rightarrow}_{2} x = y).$$

Tulemme määrittelemään kvanttorien semantiikan niin, että tämä lause sanoo "on tasan yksi presidentti" sanomalla

1. "on *ainakin yksi* presidentti, ja
2. *kaikki* presidentit ovat itse asiassa sama olento kuin tuo yksi".

Merkintä '$\iota x \, \mathrm{presidentti}(x)$' tarkoittaa puolestaan tuota vastaavaa yksikäsitteistä presidenttiä.

Samalla nähdään, miten voidaan sanoa

1. "joukosta S löytyy jokin x, jolle. . . " ja
2. "kaikille joukon S jäsenille pätee. . . ".

Samalla näemme ilmaisuvoiman laajentuneen: lauseen

alussa puhumme tuntemattomasta presidentistä nimeltään "herra/rouva x" ja

lopussa palaamme puhumaan *samasta* x toistamiseen.

Luvun 5.1 lauselogiikassa taas lauseen jokaisen osan piti olla kokonainen väitelause.

Pronominit (lat. pro nomen, "nimen sijasta") ilmestyivät näin kieleemme: "on ainakin yksi henkilö joka on presidentti, ja kaikki muut presidentit ovat sama henkilö kuin *hän*".

5.2.2. Semantiikka

Predikaattilogiikan kaavan ϕ tulkinta (epä)todeksi tapahtuu matemaattisessa rakenteessa $\mathfrak{M} = \langle \mathcal{U}, \mathcal{I} \rangle$, jonka osat ovat:

Universumi $\mathcal{U}$ on joukko.

Sen alkioina ovat kaikki ne oliot, joita kaavan ϕ termit saattavat saada arvoinaan.

Jos esimerkiksi ϕ puhuu luonnollisista luvuista, niin $\mathcal{U} = \mathbb{N}$.

Kielellisesti $\mathcal{U}$ koostuu kaikista *yksikäsitteisistä jakamattomista erisnimistä*: 'St. Petersburg', 'Venäjän Itämeren laivasto', 'Tsaari Pietari Suuri', 'puna-armeija', 'Leo Davidovits Trotski', . . .

Tulkintafunktio $\mathcal{I}$ on funktio tietämyksenesityskielen sanastolta sanojen merkityksille.

- $\mathcal{I}$ kuvaa jokaisen funktiosymbolin $f \in \mathcal{F}$ joksikin funktioksi $\mathcal{I}(f) : \mathcal{U}^{\mathrm{ar}(f)} \mapsto \mathcal{U}$.

  Esimerkiksi universumissa $\mathcal{U} = \mathbb{N}$ tulkitaan 2-paikkainen funktiosymboli '$+$' yhteenlaskuoperaatioksi.

  Kielellisesti $\mathcal{I}(f)$ vastaa *epäsuoraa rakenteista mutta silti yksikäsitteistä nimeä*, esimerkiksi $\mathcal{I}(\text{founder})$ voisi olla 'x:n perustaja'.
- $\mathcal{I}$ kuvaa jokaisen predikaattisymbolin $p \in \mathcal{P}$ joksikin relaatioksi $\mathcal{I}(p) \subseteq \mathcal{U}^{\mathrm{ar}(p)}$.

  Esimerkiksi universumissa $\mathcal{U} = \mathbb{N}$ tulkitaan 2-paikkainen relaatiosymboli '$\leq$' lukujen keskinäiseksi järjestysrelaatioksi.

  Kielellisesti $\mathcal{I}(p)$ vastaa väitelausetta kuten lausemuuttuja P luvun 5.1 lauselogiikassa.

  Nyt kuitenkin tämän väitelauseen *nimiä koskevat osat* kuten subjektit ja objektit saattavat *vaihdella* (lauseen antamissa rajoissa):
  - erisnimen tilalla saattaa nyt olla pronomini
  - pronominiä edustaa muuttujasymboli
  - muuttujan/pronominin viittauskohde ei ole kiinteä.

    "Jokaiselle joku on rakkain."

Esimerkiksi $\mathcal{I}(\text{founded})$ saattaa olla "x perusti y:n" eli

$$\begin{aligned}\{\ &\langle \text{Tsaari Pietari Suuri}, \text{St. Petersburg} \rangle\ , \\ &\langle \text{Tsaari Pietari Suuri}, \text{Venäjän Itämeren laivasto} \rangle\ , \\ &\langle \text{Leo Davidovits Trotski}, \text{puna-armeija} \rangle\ , \ldots \}.\end{aligned}$$

Erityisesti $\mathcal{I}(=) = \{\langle v, v \rangle : v \in \mathcal{U}\}$.

Vielä tarvitaan tulkinta muuttujasymboleille $\mathcal{X}$. Sitä ei voida liittää rakenteeseen $\mathfrak{M}$, koska kvanttorit lisättiin luvun 5.1 lauselogiikkaan juuri siksi, että muuttujien tulkinnat voisivat vaihdella.

Sijoitus (assignment) on mielivaltainen funktio $\theta \colon \mathcal{X} \mapsto \mathcal{U}$.

Nyt voidaan määritellä predikaattilogiikan semantiikka "kaava ϕ on totta tulkinnalla $\mathfrak{M}$ ja sijoituksella θ". Sitä merkitään

$$\mathfrak{M} \models \phi\theta.$$

Määrittely on jälleen induktiivinen luvun 5.2.1 kaavanrakennussääntöjen suhteen:

- $\mathfrak{M} \models (\phi \wedge \psi)\theta$ pätee jos ja vain jos sekä $\mathfrak{M} \models \phi\theta$ että $\mathfrak{M} \models \psi\theta$ pätevät.
  Kielellisesti siis '$\wedge$' on edelleen 'ja'.
- $\mathfrak{M} \models (\neg\phi)\theta$ pätee jos ja vain jos $\mathfrak{M} \models \phi\theta$ ei päde.
  Tätä ehtoa merkitään usein lyhyemmin $\mathfrak{M} \not\models \phi\theta$.
  Kielellisesti siis '$\neg$' on edelleen 'ei'.
- $\mathfrak{M} \models (\exists x_i \phi)\theta$ pätee jos ja vain jos on olemassa jokin $u \in \mathcal{U}$ jolla $\mathfrak{M} \models \phi\theta[x_i = u]$ pätee, missä

$$\theta[x_i = u](x_j) = \begin{cases} u & i = j \\ \theta(x_j) & i \neq j \end{cases}$$

on se sijoitus, joka on muuten kuin θ, paitsi että x_i saakin siinä arvon u.

Toisin sanoen, jos ja vain jos muuttuja x_i voidaan sitoa johonkin sellaiseen arvoon $u \in \mathcal{U}$ että ϕ tulee todeksi ilman että minkään muun muuttujan x_j sidontaa täytyy muuttaa.

Vertaa tietokoneen muistiin kirjoitus:

– sijoitus θ on muisti taulukkona
– muuttujasymbolit x_i ovat muistiosoitteita
– arvot u ovat muistipaikkojen sisältöjä.

Kielellisesti

– '$\exists$' vastaa ilmaisua 'on olemassa sellainen... että...' eli "jollakin"
– '$\forall$' vastaa ilmaisua 'ei ole olemassa sellaista... ettei...' eli 'kaikilla...'.

Tai '$\exists x \phi$' voidaan tulkita sellaiseksi *metakielen äärettömäksi disjunktioksi, jonka koko riippuu universumista* $\mathcal{U} = \{u_1, u_2, u_3, \ldots\}$: $\mathfrak{M} \models \phi\theta[x_i = u_1]$ tai $\mathfrak{M} \models \phi\theta[x_i = u_2]$ tai $\mathfrak{M} \models \phi\theta[x_i = u_3]$ tai . . .

Tällöin '$\forall x \phi$' olisi vastaava konjunktio.

- $\mathfrak{M} \models p(t_1, \ldots, t_{\mathrm{ar}(p)})\theta$, missä $p \in \mathcal{P}$ on siis predikaattisymboli, pätee jos ja vain jos

$$\left\langle \mathrm{value}(t_1, \theta), \ldots, \mathrm{value}(t_{\mathrm{ar}(p)}, \theta) \right\rangle \in \mathcal{I}(p),$$

missä termin t arvo $\mathrm{value}(t, \theta)$ sijoituksessa θ määritellään induktiivisesti alla.

Kielellisesti tässä *Tarskilaisessa totuusmääritelmässä* tapahtuu seuraavaa:

1. Ensin selvitämme, mihin olioon eli jakamattomaan erisnimeen $n_i = \mathrm{value}(t_i, \theta) \in \mathcal{U}$ kukin termi eli mahdollisesti rakenteinen epäsuora nimi t_i tällä hetkellä viittaa.
2. Sitten selvitämme, mihin väitelauseeseen $L = \mathcal{I}(p)$ predikaattisymboli p viittaa. Väitelauseessa L on $\mathrm{ar}(p)$ mahdollisesti vaihtelevaa nimiosaa.
3. Sitten täytämme nimillä n_i väitelauseen L vaihtelevat nimiosat.

Saadaan lopullinen metakielinen väitelause L' ilman vaihtelevia nimiosia.

4. Lopuksi kysymme, onko tämä L' totta vai ei.
   Se, miten tähän kysymykseemme saadaan metakielessä vastaus, ei enää ole objektikielen eli predikaattilogiikan määräysvallassa.

- Muuttujasymboleille $x_i \in \mathcal{X}$ määritellään $\mathrm{value}(x_i, \theta) = \theta(x_i)$.
- Funktiosymboleille $f \in \mathcal{F}$ taas $\mathrm{value}(f(t_1, \ldots, t_{\mathrm{ar}(f)}), \theta) = \mathcal{I}(f)(\mathrm{value}(t_1, \theta), \ldots, \mathrm{value}(t_{\mathrm{ar}(f)}, \theta))$.
  Vertaa aliohjelmakutsun *suoritus sieventämällä* funktionaalisessa ohjelmointikielessä kuten Scheme:
  – Muuttujasymbolin x_i arvo katsotaan muuttujamuistista θ.
  – Lausekkeen $f(t_1, \ldots, t_{\mathrm{ar}(f)})$ eli (f t_1 ... $t_{\mathrm{ar}(f)}$) arvo lasketaan
    1. laskemalla ensin rekursiivisesti sen argumenttien t_i arvot n_i
    2. katsomalla sitten funktiomuistista $\mathcal{I}$ mitä funktiota g nimi f vastaa
       (Alkuperäisessä Lisp-kielessä tehdäänkin tämä ero funktio- ja muuttujamuistien välillä, Schemessä ei.)
    3. ja lopuksi kutsumalla saatua funktiota g lasketuilla parametreilla n_i.

Näin on määritelty predikaattilogiikan semantiikka.

Tarkastellaan esimerkkinä kaavaa

$$\forall x. \, \mathrm{founded}(\mathrm{founder}(x), x) \tag{29}$$

eli "jokaiselle x pätee, että x:n perusti sen perustaja":

1. $\mathfrak{M} \models$ (kaava (29))θ pätee jos ja vain jos kaikille $u \in \mathcal{U}$ pätee

$$\mathfrak{M} \models \mathrm{founded}(\mathrm{founder}(x), x) \underbrace{\theta[x = u]}_{\text{merk. } \theta'} . \tag{30}$$

Kielellisesti: jos ja vain jos jokainen lause $\mathrm{founded}(\mathrm{founder}(w), w)$ on totta, missä w on mikä tahansa sellainen erisnimi, joka voidaan muodostaa kielen sanoilla.

2. Yhtälö (30) puolestaan pätee jos ja vain jos

$$\big\langle \mathrm{value}(\mathrm{founder}(x), \theta'), \mathrm{value}(x, \theta')\big\rangle \in \underbrace{\mathcal{I}(\mathrm{founded})}_{\text{merk. } p}$$

on totta, missä

$$\begin{aligned} \mathrm{value}(x, \theta') &= \theta'(x) \\ &= \theta[x = u](x) \\ &= u \\ \mathrm{value}(\mathrm{founder}(x)\theta') &= \mathcal{I}(\mathrm{founder})(\mathrm{value}(x, \theta')) \\ &= \underbrace{\mathcal{I}(\mathrm{founder})}_{\text{merk. } f}(u). \end{aligned}$$

Toisin sanoen, jos ja vain jos

$$\langle f(u), u\rangle \in p.$$

Kielellisesti: olio u on mielivaltaisen erisnimen w tulkinta.

3. Toisin sanoen, kaava (29) pätee jos ja vain jos
   - funktiosymbolin $\mathrm{founder}$ tulkinta f
   - kuvaa jokaisen alkion $u \in \mathcal{U}$
   - sellaiseksi alkioksi $v \in \mathcal{U}$
   - että pari $\langle v, u\rangle$ kuuluu relaatiosymbolin $\mathrm{founded}$ tulkintaan p.

   Kielellisesti: voimme järkeillä tämän merkitsevän samaa kuin joukko-opillinen väite

$$p = f^{-1}.$$

Tämä totuusmääritelmä antaa samalla muuttujille samanlaiset *leksikaaliset näkyvyysalueet (lexical scope)* kuin (yleensä) ohjelmointikielissä, esimerkiksi Scheme-kielessä:

Jokainen muuttuja on *lähimmän eli juuri tekstuaalisesti edeltävän* kvanttorinsa "määräysvallassa".

Siis kaavassa

$$\boxed{\underbrace{\exists x_1}_{(a)} .p_1(\underbrace{x_1}_{(b)}) \wedge \boxed{\underbrace{\forall x_1}_{(c)} .p_2(\underbrace{x_1}_{(d)})}} \tag{31}$$

muuttujan x_1 ensimmäistä esiintymää (b) kontrolloi kvanttori (a), mutta saman muuttujan x_1 toista esiintymää (d) kontrolloikin toinen, sitä läheisempi kvanttori (c). Kaavan totuutta tutkittaessa muuttuja x_1 voi siis saada kohdissa (b) ja (d) eri arvot, koska kohdassa (c) valittava arvo "peittää näkyvistä" aiemmin kohdassa (a) valitun arvon.

Kvantifioidun muuttujan symbolia voidaan vaihtaa korvaamalla systemaattisesti kaikki kyseisen kvanttorin kontrolloimat esiintymät uudella symbolilla. Näin kaavasta (31) voidaan tällä *uudelleennimennällä (renaming)* tuottaa yhtäpitävä, selkeämpi muoto

$$\underbrace{\exists x_1}_{(a)} .p_1(\underbrace{x_1}_{(b)}) \wedge \underbrace{\forall x_2}_{(c)} .p_2(\underbrace{x_2}_{(d)}).$$

Haluamme kutsua yhtälöä (17) seuraten predikaattilogiikan kaavan ϕ malleiksi niitä tulkintoja $\mathfrak{M}$ joissa ϕ on tosi. Nyt kuitenkin täytyy ottaa huomioon myös sijoitukset θ, ja määritellä

$$\mathrm{Mod}(\phi) = \{\mathfrak{M} \colon \mathfrak{M} \models \phi\theta \text{ kaikilla } \theta\} \,. \tag{32}$$

Tällöin kaavan ϕ *vapaat muuttujat (free variables)* — eli ne jotka esiintyvät ilman minkään kvanttorin kontrollointia — tulkitaan näkymättömästi universaalikvantifioiduiksi.

Kaavaan (29) tämä ei vaikuta, koska siinä ei ole vapaita muuttujia. Tällaisia kaavoja sanotaan *suljetuiksi kaavoiksi (closed formulae)*.

Tietämyskantaan KB lisätään (operaatiolla $\textsc{Tell}$) yleensä vain suljettuja kaavoja, koska vapaat muuttujat saattaisivat aiheuttaa yllättäviä riippuvuuksia eri aikoina lisättyjen kaavojen välille.

Nimitys "kvanttori" tulee *lukumääristä (lat. quantus, "kuinka paljon?")*. Sanat "jollakin" ja "kaikilla" toimivat näet luonnollisessa kielessä samoin kuin "ainakin kolme", "tasan viisi",. . .

Luvussa 1.2 todettiinkin, että predikaattilogiikka kehitettiin matematiikan filosofisten perusteiden tutkimiseen. Siinä lukumääriä koskevien lauseiden formalisointi oli tietenkin keskeistä.

Ensimmäisen kertaluvun predikaattilogiikalla ei kuitenkaan pysty formalisoimaan sitä *miten* nämä "kvanttorin kaltaiset" sanat toimivat luonnollisessa kielessä: pelkkä '$\forall$' ei vielä ole tämän kielen kokonainen ilmaisu.

Korkeamman kertaluvun predikaattilogiikassa tällaisten sanojen toiminta voidaan formalisoida funktioina *väitelauseilta väitelauseille* [Car97, luku 3]:

Operaatio '$\forall$' ottaa argumenttinaan *yhtä* oliota koskevan väitelauseen ja tuottaa tuloksenaan saman lauseen mutta yleistettynä *jokaiseen* olioon.

(Ensimmäisessä kertaluvussahan kaikki funktiot ovat *olioilta olioille*.)

Jos tähän suuntaan lähdetään, niin samalla on luontevaa luopua tarkastelemasta vain *predikaatti*logiikkaa, jossa lopulliset tulokset ovat aina totuusarvotyyppiä $\mathbb{B}$. Esimerkiksi $\forall : (\mathcal{U} \mapsto \mathbb{B}) \mapsto \mathbb{B}$.

Näin päästään *lambda-laskentaan (lambda calculus)*, jolla on useita sovelluksia tietojenkäsittelyssä:

- ohjelmointikielissä — teoriassa ja käytännössä
- luonnollisen kielen käsittelyssä [RN95, osa VII][Nil98, osa V]
- . . .

Kurssi ei kuitenkaan lähde tähän suuntaan, koska päättely korkeamman kertaluvun logiikoissa on vielä vaikeampaa kuin ensimmäisen [Vää87, luku VI.3]. Niillä esitettyä tietämystä on siis vielä vaikeampaa käyttää.

5.2.3. Käyttöesimerkkejä

> ". . . [H]e construed the world as a computer construes it, by means of key features and schematic relationships. The scheme might be identified – in an 'identiti-kit' way – without the reality being grasped at all."
>
> *O. Sacks: The Man Who Mistook His Wife for a Hat*

Esitetään predikaattilogiikalla ensiksi *ihmisten välisiä sukulaisuussuhteita*:

1. Tietämyksen esittäminen logiikalla alkaa siitä, että *sovitaan ne alkiot*, jotka sisällytetään perusjoukkoomme $\mathcal{U}$.

   Eli millaisia otuksia ajattelemme olevan olemassa.

   "Mistä lauseidemme on tarkoitus puhua?"

   Tässä tilanteessa joukkoon $\mathcal{U}$ on luontevaa sisällyttää ihmisiä.

2. Sitten *sovitaan looginen aakkosto (alphabet)*: ne sanat joiden välityksellä tätä tietämystä halutaan kuvailla.

   "Millaisin sanoin haluamme ilmaista lauseemme?"

   Yksi tapa löytää aakkosto on alkaa luetella erilaisia sovellusalueeseen (nyt siis sukulaisuuteen) liittyviä käsitteitä ja lauseita:

   - Sukulaisuussuhde "x on y:n isä/äiti" lienee ainakin tarpeen.
   - Erottelu "x on mies/nainen" lienee myös tarpeen jos haluamme puhua isistä ja äideistä. . .
   - Muita ovat esimerkiksi "x on y:n veli/sisar/sisarus/serkku/esi-isä/jälkeläinen/. . . "

   On vaikea sanoa milloin aakkostoa on kerätty tarpeeksi.

   Aakkostoa ja perusjoukkoa $\mathcal{U}$ voidaan joutua täydentämään myöhemmin, kun tavataan tietämystä jota ei pystytäkään esittämään.

   Vertaa olio-ohjelmoinnin käsiteanalyysiavaihe: "hauki on sellainen kala, jonka. . . " Siis meillä on luokka *kalat*, ja sen aliluokkana *hauet*,. . .

   Vertaa myös tietokantamallin hahmotteluvaihe.

3. Seuraavaksi *jaotellaan kerätty aakkosto* vakioihin, funktioihin ja predikaatteihin:

- Jos isyys/äitiys tulkitaan puhtaan biologisesti, niin voidaan käyttää 1-paikkaisia funktioita $\mathrm{father}(y)$ ja $\mathrm{mother}(y)$, koska jokaisella ihmisellä y on tasan yksi isä/äiti. Sovitaan esimerkissämme näin.

  Voitaisiin käyttää myös 2-paikkaisia predikaatteja $\mathrm{father}(x, y)$ ja $\mathrm{mother}(x, y)$, jolloin samalla ihmisellä y voisi olla monta isää/äitiä x.

  Jaottelu ei siis synny automaattisesti, vaan rajatapaukset vaativat aina harkintaa.
- Sukupuoli voidaan ilmaista 1-paikkaisilla predikaateilla $\mathrm{male}(x)$ ja $\mathrm{female}(x)$.

  Niiden sijasta voitaisiin valita myös funktiosymboli $\mathrm{gender}(x)$. Tällöin jouduttaisiin lisäämään myös
  - aakkostoon symboliset arvot male ja female
  - perusjoukkoon näiden symbolien tulkinnat, eli käsitteet "$\mathrm{miehuus}$" ja "$\mathrm{naiseus}$".

  Valinta näiden kahden vaihtoehdon välillä vaikuttaa myöhemmin kirjoitettavien tietämyksenesityslauseiden muotoiluun, eikä "oikeaa" valintaa ole helppo tehdä tarkastelematta, kummassa nämä lauseet muotoutuvat helpommin.

  Tähän palataan hieman myöhemmin.

  Jo tästä näkee, ettei tämä(kään) tietojenkäsittelyn suunnitteluprosessi etene koko ajan vain yhteen suuntaan. . .
- Muut sukulaisuussuhteet lienee luontevinta esittää 2-paikkaisilla predikaateilla $\mathrm{brother}(x, y)$, $\mathrm{sister}(x, y)$, $\mathrm{sibling}(x, y)$, $\mathrm{cousin}(x, y)$, $\mathrm{ancestor}(x, y)$, $\mathrm{descendant}(x, y)$,. . . koska samalla ihmisellä x voi aina periaattessa olla useita veljiä/sisaria/sisaruksia/serkkuja/esi-isiä/jälkeläisiä/. . . y.

4. Luokitellaan aakkoston predikaatit *perus-* ja *johdettuihin käsitteisiin* kysymällä kustakin "voinko päätellä sen havainnoista joistakin muista aakkoston esittämistä käsitteistä?"

   Sama predikaatti voi olla sekä perus- että johdettu käsite: esimerkiksi kuvassa 59 (b) 0-paikkainen predikaatti $P_{[3,1]}$ pääteltiin, mutta se olisi voitu havaita suoraankin — astumalla kuoppaan.

   Esimerkiksi "x on y:n sisarus" on johdettu käsite, koska se voidaan määritellä isyyden ja äitiyden avulla muodossa "x ja y omaavat saman isän ja/tai äidin". (Jälleen täytyy samalla sopia hyväksytäänkö sisaruspuoletkin vaiko ei. Sovitaan hyväksytyiksi.)

Logiikassa vastaava *määritelmä (definition)* on määriteltävän predikaatin ekvivalenssi määrittelevän lauseen kanssa:

$$\begin{aligned}\forall x, y.\, \text{sibling}(x, y) \Leftrightarrow {} & \text{father}(x) = \text{father}(y) \lor \\ & \text{mother}(x) = \text{mother}(y).\end{aligned}$$

Vertaa sanakirjamääritelmät:

sisarukset, n. *Henkilöt, joilla on yhteinen vanhempi.*

Tässä on jopa määritelmä *tapauksittain (by cases)*: *...sama isä* <u>*tai*</u> *sama äiti.*

Esimerkissämme isyys ja äitiys ovat luontevia peruskäsitteitä: niiden pohjalta voidaan määritellä muut sukulaisuussuhteet muttei päinvastoin.

Sukupuoli on myös otettava peruskäsitteeksi: vaikka tiedämmekin että isät/äidit ovat aina miehiä/naisia eli

$$\forall x.\, \text{male}(\text{father}(x)) \tag{33}$$

$$\forall x.\, \text{female}(\text{mother}(x)) \tag{34}$$

niin tämä ei vielä kiinnitä lapsettomien ihmisten sukupuolia.

Luvussa 1.3 puhuttiin loogisten positivistien käännettävyysteesistä. Ero perus- ja määriteltyjen käsitteiden välillä voidaan myös tulkita eroksi

- suoraan maailmaa tarkastelemalla havaittavissa olevien asioiden
- (havainnoissa ilmenneille säännönmukaisuuksille sovittujen) kielellisten lyhennysmerkintöjen

välillä.

Esimerkissämme voisimme aloittaa havaitsemalla, että ihmiset(kin) ovat sitä läheisempiä toisilleen mitä enemmän heillä on yhteistä geneettistä perimäainesta – siis vanhempi/lapsi-suhde on perustava.

Tämän päälle kieliyhteisössä on vakiintunut erilaisia muita termejä erottelemaan sitä, kuinka läheisiä ihmiset ovat toisilleen:

”Minä veljeäni vastaan. Minä ja veli serkkuani vastaan. Minä, veli ja serkku muukalaista vastaan.” *Arabialainen sananlasku*

Määritelmät voivat myös *viitata itseensä*: ”x on y:n jälkeläinen jos ja vain jos x on y:n lapsi tai jonkin y:n *lapsen jälkeläinen*”.

$$\forall x, y.\, \mathrm{child}(x, y) \Leftrightarrow \mathrm{father}(x) = y \lor \mathrm{mother}(x) = y.$$

$$\forall x, y.\, \mathrm{descendant}(x, y) \Leftrightarrow \mathrm{child}(x, y) \lor \exists z.\, \mathrm{child}(z, y) \land \mathrm{descendant}(x, z).$$

Sukupuissa tällainen itseensäviittaava sääntö esittää navigointia pitkin niitä isä/äiti-linkkejä, joista puu koostuu.

Logiikkaohjelmointikielessä Prolog

- ohjelmat ovat predikaattien määritelmiä tapauksittain
- rekursio (ja siis myös toisto) ilmaistaan itseensäviittaavina määritelminä.

5. Kun käsitteet on valittu, täytyy vielä lausua loogisesti niiden keskinäiset riippuvuudet kuten kaavat (33) ja (34) jotta vältyttäisiin ”itsestäänselvyyksien” aiheuttamilta yllätyksiltä:
   - ”Jokainen perusjoukon olio on joko mies tai nainen.”

$$\forall x.\, \mathrm{male}(x) \Leftrightarrow \neg\, \mathrm{female}(x) \tag{35}$$

   Jos perusjoukkoomme lisättiin myös sukupuolettomia olioita kuten tiiliskiviä, niin tätä lyhyttä kaavaa pitää monimutkaistaa esimerkiksi muotoon ”jokainen *ihminen* on joko mies tai nainen”:

$$\forall x.\ \mathrm{human}(x) \Rightarrow (\mathrm{male}(x) \Leftrightarrow \neg\, \mathrm{female}(x)).$$

   Jos taas valittiin funktiosymboli gender, niin mikään alkio ei tämän valinnan nojalla voi olla yhtä aikaa sekä mies että nainen. Mutta:
   – Perusjoukkoon lisättiin samalla uudet alkiot ”$\mathrm{miehuus}$” ja ”$\mathrm{naiseus}$”.
     Nämä uudet alkiot eivät ole ihmisiä, vaan ihmisiin liittyviä *abstrakteja käsitteitä*, jotka on nyt *reifioitu* eli tuotu konkreettisiksi alkioiksi.

- Funktiosymbolia gender vastaavalla funktiolla pitää olla arvo *jokaisella* alkiolla, siis myös näillä uusilla alkioilla.

  Tarvitaan siis uusi arvo "määrittelemätön" ja sille vakiosymboli undefined.
- Tarvitaan siis vielä esimerkiksi lause "sukupuoli on määritelty vain ihmisille"

$$\forall x. \neg\, \text{human}(x) \Leftrightarrow \text{gender}(x) = \text{undefined}$$

  rajaamaan määrittelemättömän arvon käyttöä.
- Joudutaan siis lisäämään myös predikaatti "(olla) ihminen".
- Funktiosymbolin gender valinta johtaisi siis ainakin yhtä monimutkaisiin tietämyksenesityslauseisiin kuin predikaattisymbolien male ja female käyttö.

  Se ei siis näytä kannattavan.

Jo tästä pikku pulmasta nähdään yksi (loogisen/symbolisen) tietämyksenesittämisen vaikeus:

Vaikka halutaan esittää vain jokin näennäisen yksinkertainen tiedonpalanen, voi siitä seurata monenlaisia

- siihen liittyvän *tausta*tietämyksen esitystarpeita
- vaikutuksia muiden tiedonpalasten esitykseen.

Silloin tietämyskannan koko paisuu ja sen sisäiset riippuvuudet mutkistuvat.

Tätä vaikeutta on yritetty lieventää eri tavoin:

- On määritelty *yleisiä ontologioita (general ontologies)* [RN95, luku 8.4]: rakenteisia (hierarkisia) "käsitekirjastoja", joihin uusia tiedonpalasia sovitellaan itse perusrakennetta muuttamatta.

  (Vertaa valmiit oliokirjastot.)
- On ryhdytty keräämään ja esittämään suuria määriä arkipäivän taustatietämystä (CYC).
- On kehitetty sellaisia logiikoita, joiden päättelyaskeleet vähentävät tarvetta esittää kaikkea tietoa eksplisiittisesti.

  Esimerkiksi luvussa 5 mainitut epämonotoniset logiikat mahdollistavat sen, että vain tavallisuudesta poikkeava tietämys kirjataan näkyviin. Jos jotakin tietoa ei ole kirjattu, niin päättelyssä

sen voidaan silloin olettaa olevan kuten tavallisesti.
  - On keskitytty koko todellisuuden sijasta sen yksinkertaisempiin osiin ja pelkistetty niitä edelleen.
- "Mikään olio ei ole oma jälkeläisensä."

$$\forall x . \neg \operatorname{descendant}(x, x)$$

Mehän emme missään vaiheessa kieltäneet kehiä sukupuissa. . .

Jälleen on vaikeaa tietää, milloin käsitteiden riippuvuuksia on kuvailtu – eli milloin sovellusaluetta on *aksiomatisoitu* – tarpeeksi.

Vasta kun aksiomatisointia käytetään päättelyyn, selviää salliiko se kaikkiin mielenkiintoisiin kysymyksiin vastaamisen.

Luvun 5 "päättelyn varovaisuusperiaatteen" mukaanhan vain se saadaan päätellä luvallisesti mikä ei missään tapauksessa voisi olla toisin silloin kun päättelyn lähtöoletukset ovat voimassa. Jos siis aksiomatisointi jättää jotakin avoimeksi, niin päättely ei etene.

Esimerkiksi aksioomaa

$$\forall x . \neg \operatorname{father}(x) = \operatorname{mother}(x)$$

eli "isä ja äiti ovat eri oliot" ei enää tarvitse lisätä:

Se ei ole *riippumaton (independent)* muista aksioomista, vaan se voitaisiin myös päätellä aksioomista (33)–(35): isä on mies, äiti taas nainen, eikä kukaan ole yhtä aikaa sekä mies että nainen.

6. Kun sovellusalue näyttää riittävästi aksiomatisoidulta, voidaan tietämyskantaan KB syöttää yksittäisiä havaintoja.

   Esimerkissämme nämä ovat esimerkiksi muotoa

$$\text{mother('King Edward VII')} = \text{'Queen Victoria'}\,.$$

   Havainnot tuovat näin mukanaan uusia vakioita nimeämään havaittuja olentoja.

Näin saadaan kuvattua staattinen, muuttumaton maailma.

Vaikka luvussa 5.2 sanottiinkin, ettei predikaattilogiikka tarjoa *valmiina* menetelmiä dynaamisen, muuttuvan maailman kuvaamiseen, ei mikään estä aksiomatisoimasta sellaista *käsin*.

Eräs luonteva tapa on *tilannekalkyyli (situation calculus)*:

- Jokaiseen ajasta riippuvaan perusilmaukseen (predikaatti- tai funktiosymboliin) lisätään uusi (viimeinen) argumentti.
- Tämä uusi argumentti edustaa sitä *tilannetta (situation)* jossa ilmauksen merkitystä tutkitaan.

  Predikaatin totuus
  funktion osoittama alkio
  riippuu nyt (myös) tilanteesta.

Esimerkki ajasta riippuvasta vakiosta (0-paikkaisesta funktiosta) on ”Suomen presidentti”.

Määritelläänkin se siis 1-paikkaisena funktiona ”Suomen presidentti ajanhetkellä t”, jolloin

$$\text{president-of-Finland('13th Nov. 1998')} = \text{'M. Ahtisaari'}$$

$$\text{president-of-Finland('13th Nov. 2001')} = \text{'T. Halonen'}$$

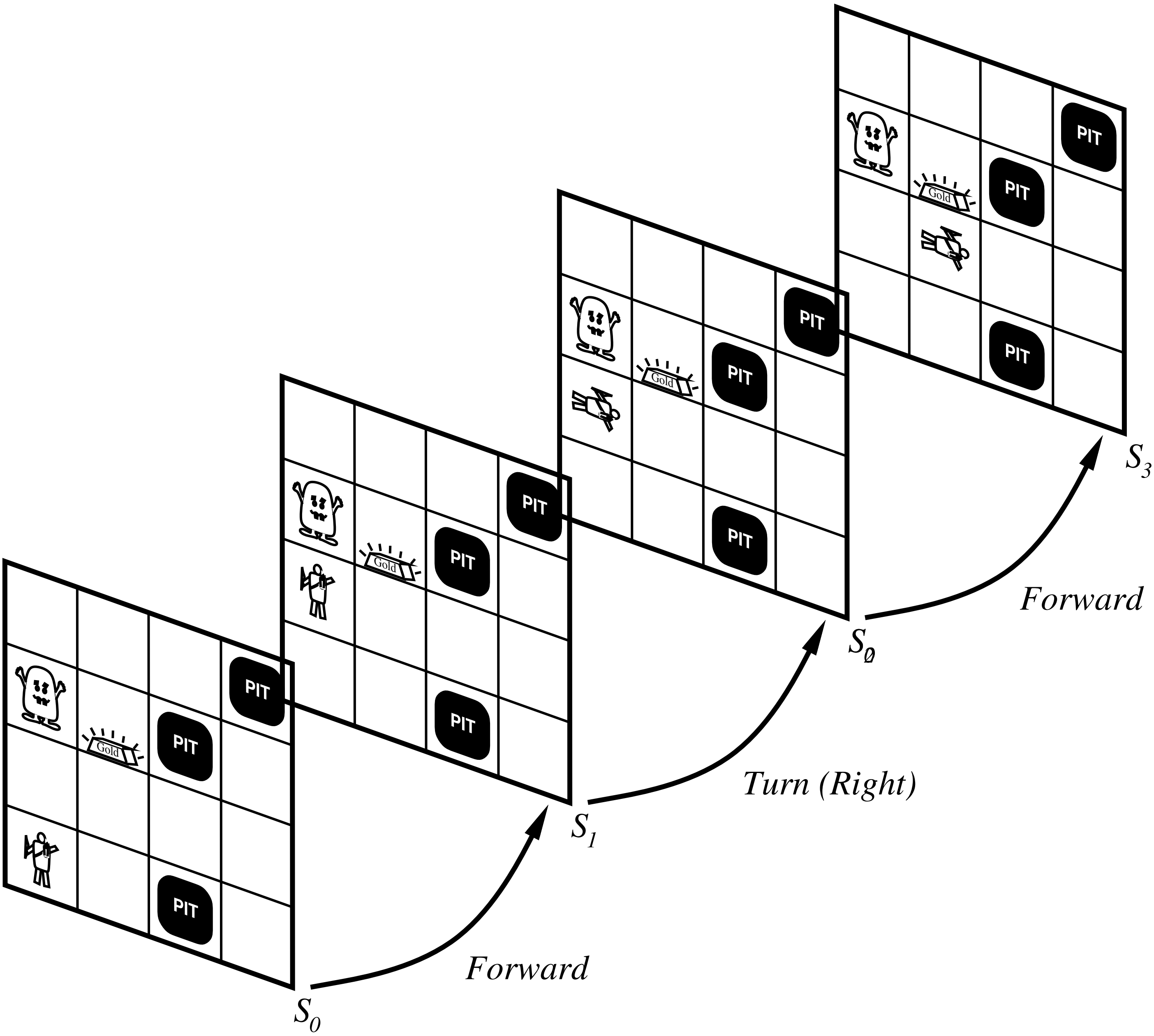
kuvaa viranhaltijan vaihtumista joskus tilanteiden '13th Nov. 1998' ja '13th Nov. 2001' välillä.

Ajalle on tarjolla monenlaisia eri malleja, jotka sopivat erilaisiin tiedonesitystilanteisiin:

- Teollisen prosessin ohjausjärjestelmässä aika lienee jatkuva käsite, koska selluloosan keitinliemessä tapahtuu koko ajan jotakin.
- Tilannekalkyylissä taas *toimija(i)n teot* kuljettavat aikaa valitsemalla seuraavan tilanteen monista mahdollisista vaihtoehdoista.

  Esimerkkinä luvun 5.1 Wumpus-esimerkin mukainen kuva 60.

Tilannekalkyylissä on siis valittu "sarjakuva-aika", jossa "ruudusta" eli tilanteesta toiseen edetään "sankarin" eli toimijan tekojen tahdissa.



Kuva 60: Wumpus-pelitilanteita [RN95, kuva 7.3].

Tilannekalkyylissä käytetään erityistä *tulosfunktiota*, jonka edellinen argumentti on toimijan teko ja jälkimmäinen se tilanne missä se tehdään, ja joka palauttaa arvonaan tekoa seuraavan tilanteen. Siis kuvassa 60

$$\mathrm{result}(\mathrm{forward}, S_0) = S_1$$

$$\mathrm{result}(\mathrm{turn}(\mathrm{right}), S_1) = S_2$$

$$\mathrm{result}(\mathrm{forward}, S_2) = S_3$$ ja niin edelleen.

Näin tilannekalkyyli reifioi abstraktin käsitteen 'tilanne' jonka avulla muutokset esitetään.

Tekojen aikaansaamat muutokset kuvaillaan *vaikutusaksioomilla (effect axioms)* kuten

"jos tavara x on läsnä tilanteessa s ja kannettavissa, niin teko 'tartu' aiheuttaa sen, että toimija pitää sitä kädessään":

$$\forall x, s.\, \mathrm{present}(x, s) \wedge \mathrm{portable}(x) \Rightarrow \mathrm{holding}(x, \mathrm{result}(\mathrm{grab}, s))$$

Siinä tarvittavat apukäsitteet ovat:

$$\mathrm{portable}(\mathrm{gold})$$

$$\forall s.\, \mathrm{at\_gold}(s) \Rightarrow \mathrm{present}(\mathrm{gold}, s) \tag{36}$$

$$\forall s, b, u, c, t.\, \mathrm{percept}([s, b, \mathrm{glitter}, u, c], t) \Rightarrow \mathrm{at\_gold}(t) \tag{37}$$

Tämä tarpeettoman monimutkaisen näköinen muotoilu on valittu siten, että *koko* tietämyksen kuvaileminen olisi helpompaa:

- Sama aksiooma kuvaa kaikkien tarttumistekojen toimintaa, tartuttiinpa sitten kultaan tai mihin muuhun tahansa.
- Kaavan (36) "välikäsite" $\mathrm{present}(\text{esine}, \text{tilanne})$ otettiin käyttöön siltä varalta, että toimijalla olisi muitakin tapoja havaita kulta kuin kaavan (37) kimalluksen näkeminen.
- Kaava (37) tutkii puolestaan kullan kimallusta toimijan tilanteessa t saamasta havainnosta havaintovektorista.
  Sen kirjamme [RN95] esittää Prolog-tyylisenä listana [hakasuluissa].
  Samalla nähdään luvun 2.2 heijastetoimijoiden tietämyskannan KB sisältämien sääntöjen yleinen muoto:
  Implikaatiot eli "jos. . . niin. . . " -säännöt

  $$\forall t.\, \mathrm{at\_gold}(t) \Rightarrow \mathrm{action}(\mathrm{grab}, t)$$

  jotka ehdottavat aina seuraavaa tekoa $\mathrm{action}(\text{teko}, \text{tilanne})$ suoraan havainnon pohjalta miettimättä asiaa sen tarkemmin.

Kääntäen teko 'irrota' kumoaa tarttumisen:

$$\forall x, s. \neg \, \mathrm{holding}(x, \mathrm{result}(\mathrm{release}, s)).$$

Näin kuvaillaan kaikki mahdolliset teot, joita Wumpus-esimerkissä ovat

$$\mathrm{turn(right)} \qquad \mathrm{turn(left)} \qquad \mathrm{forward}$$

$$\mathrm{shoot} \qquad \mathrm{grab} \qquad \mathrm{release} \qquad \text{ja} \qquad \mathrm{climb}$$

missä uudet teot ovat seuraavat:

- Teko shoot ampuu viinen ainoan nuolen nykyiseen kulkusuuntaan tappaen osuessaan wumpus-hirviön, joka huutaa kuollessaan.
- Teko climb kiipeää pois labyrintista mutta vain sen alkuruudussa.

Maailman muutoksista kertovien vaikutusaksioomien lisäksi tarvitaan *kehysaksioomia (frame axioms)* kertomaan miten maailma *ei* muutu.

Esimerkiksi "käteen otettu tavara pysyy kädessä kunnes ote siitä irrotetaan":

$$\begin{aligned} &\forall a, x, s. \, \mathrm{holding}(x, \mathrm{result}(a, s)) \\ &\qquad \Leftrightarrow a = \mathrm{grab} \land \mathrm{present}(x, s) \land \mathrm{portable}(x) \\ &\qquad\qquad \lor \\ &\qquad\qquad \mathrm{holding}(x, s) \land a \neq \mathrm{release} \, . \end{aligned}$$

(Olisi siis kannattanut määritellä apukäsite 'tartuttavissa oleva'. . .)

Tällaisen *seuraajatilanneaksiooman (successor-state axiom)* yleinen muoto on

totta edelleen ⇔ tämä teko teki sen todeksi

∨

jo totta eikä tämä teko muuttanut sitä

Sellainen tarvitaan jokaiselle predikaatille, jonka totuus voi vaihtua siirryttäessä tilanteesta toiseen.

Tekoälyssä on pohdittu paljon maailman muuttumisen kuvailuun liittyviä *ongelmia*:

Kehysesityso. (Representational Frame Problem) on se, että kehysaksioomia kertyy helposti tolkuttoman monta.

Seuraajatilanneaksioomien käyttö ehkäisee tätä ongelmaa.

Kehyspäättelyo. (Inferential F. P.) on se, että käytettäessä näin esitettyä tietämystä joudutaan joka hetki päättelemään paitsi juuri tapahtuneet muutokset myös uudelleen kaikki se, mikä *ei* muuttunut.

Suunnittelujärjestelmät (planning systems) [RN95, osa IV][Nil98, osa IV] pyrkivät ratkaisemaan tätä ongelmaa pitämällä erikseen kirjaa kuhunkin tekoon liittyvistä

- edellytyksistä lähtötilanteelta
- seurauksista tulostilanteelle

ja valitsemalla suunnitelmaksi sellainen tekojono jossa

- saavutetaan haluttu tavoite
- kunkin teon edeltäjät joko varmistavat tai turvaavat sen edellytykset
- kunkin teon edellytykset ja seuraukset päätellään paikallisesti naapuritekojen perusteella käyttäen yksinkertaistettua logiikkaa.

Silloin tilanteet esitetään implisiittisesti eikä enää kuvan 60 suurina tulostermeinä

$$\mathrm{result}(\mathrm{forward}, \mathrm{result}(\mathrm{turn}(\mathrm{right}), \mathrm{result}(\mathrm{forward}, S_0))).$$

Lisäksi tällaiset tulostermit johtavat *sarjalliseen* suunnitelmaan, kun taas toisistaan riippumattomat teot voitaisiin hoitaa missä järjestyksessä tahansa. Tämä on *osittain järjestettyä suunnittelua (Partial-Order Planning)*.

Ehdollistamiso. (Qualification P.) on se, että on vaikea antaa *tarkalleen* niitä olosuhteita joissa jokin teko *varmasti* tuottaa tuloksensa.

Mikä kaikki voi mennä pieleen kultaa nostettaessa? Harkko voi olla liukas ja pudota käsistä, selkä voi venähtää kumartaessa,. . .

Haarautumiso. (Ramification P.) on se, että on vaikea kuvailla tekojen *kaikkia* vaikutuksia.

Harkon mukana nousee jokainen sen pinnalla oleva pölyhiukkanenkin; onko niistä(kin) pidettävä kirjaa päättelemällä?

Wumpus-esimerkissä on selvästi muistettava myös toimijan nykyinen paikka. Aksiomatisointi voitaisiin rakentaa vaikkapa seuraavasti:

- Määritellään predikaatti

$$\mathrm{at}(\text{olio}, [x\text{-koordinaatti}, y\text{-koordinaatti}], \text{tilanne})$$

  tarkoittamaan "olio on ruudussa x,y annetussa tilanteessa".

  Silloin se soveltuu muihinkin olioihin (eli kultaan, kuoppiin, hirviöön,. . .) kuin vain toimijaan agent.

  Alkutilanteessa $\mathrm{at}(\mathrm{agent}, [1, 1], S_0)$ kuvan 60 mukaisesti.
- Määritellään vastaavasti funktio

$$\mathrm{orientation}(\text{olio}, \text{tilanne})$$

  muistamaan (geometrisina asteina) sitä suuntaa, johon olio etenee:

aste	**suunta**	
0	kasvava	x-akseli
90	kasvava	y-akseli
180	vähenevä	x-akseli
270	vähenevä	y-akseli

  Alkutilanteessa $\mathrm{orientation}(\mathrm{agent}, S_0) = 90$ kuvan 60 mukaisesti.
- Koordinaatit ja suunnat yhdistyvät funktiolla

$$\forall x, y.\ \mathrm{location\_toward}([x, y], 0) = [x + 1, y]$$

$$\forall x, y.\ \mathrm{location\_toward}([x, y], 90) = [x, y + 1]$$

$$\forall x, y.\ \mathrm{location\_toward}([x, y], 180) = [x - 1, y]$$

$$\forall x, y.\ \mathrm{location\_toward}([x, y], 270) = [x, y - 1]$$

jonka avulla voidaan määritellä funktio "edessä oleva ruutu"

$$\forall p, l, s.\, \mathrm{at}(p, l, s) \Rightarrow \mathrm{location\_ahead}(p, s) = \mathrm{location\_toward}(l, \mathrm{orientation}(p, s))$$

ja "naapuruus"

$$\forall l_1, l_2.\, \mathrm{adjacent}(l_1, l_2) \Leftrightarrow \exists d. l_1 = \mathrm{location\_toward}(l_2, d).$$

Varsinaista aritmetiikkaa ei oikeasti esimerkissämme tarvita, koska suuntia ja ruutuja on labyrintissa äärellinen ja etukäteen tunnettu määrä.

Jos tarvittaisiin, niin saatettaisiin joutua (jo luvuissa 1.2 ja 5 mainittuihin) ongelmiin, koska kaikkea luonnollisten lukujen aritmetiikkaa ei voi tyhjentävästi kuvailla loogisesti. Usein kuitenkin yksinkertainen ja rajoitettu aritmetiikka riittää.

- Kerrotaan vielä mitä tiedämme itse labyrintista:

$$\forall x, y.\, \mathrm{wall}([x, y]) \Leftrightarrow x = 0 \vee x = 5 \vee y = 0 \vee y = 5.$$

- Varsinainen aksiomatisointi alkaa yhdistämällä paikan muuttuminen vastaavaan tekoon forward sopivalla seuraajatilanneaksioomalla:

$$\begin{aligned}\forall a, d, p, s, l.\, \mathrm{at}(p, l, \mathrm{result}(a, s)) \Leftrightarrow\ & a = \mathrm{forward} \wedge l = \mathrm{location\_ahead}(p, s) \wedge \neg\,\mathrm{wall}(l) \\ & \vee \\ & \mathrm{at}(p, l, s) \wedge a \neq \mathrm{forward}\,.\end{aligned}$$

- Tämän teon vaikutukset riippuvat kuitenkin sen tekijän senhetkisestä etenemissuunntasta, joten on aksiomatisoitava myös miten suunta muuttuu vastaavien tekojen vaikutuksesta:

$$\forall a, d, p, s.\, \mathrm{orientation}(p, \mathrm{result}(a, s)) = d \Leftrightarrow$$
$$a = \mathrm{turn}\begin{pmatrix}\mathrm{left}\\ \mathrm{right}\end{pmatrix} \wedge$$
$$d = (\mathrm{orientation}(p, s) \pm 90) \mod 360$$
$$\vee$$
$$\mathrm{orientation}(p, s) = d \wedge$$
$$a \neq \mathrm{turn}(\mathrm{left}) \wedge a \neq \mathrm{turn}(\mathrm{right}).$$

Nyt kun toimijan paikka ja sen muutokset on aksiomatisoitu, voidaan alkaa liittämään toimijan tekemiä havaintoja vastaaviin paikkoihin:

$$\forall l, s.\, \mathrm{at}(\mathrm{agent}, l, s) \wedge \mathrm{breeze}(s) \Rightarrow \mathrm{breezy}(l) \tag{38}$$

missä $\mathrm{breeze}(s)$ määritellään havainnon pohjalta kaavan (37) tapaan.

Tämä on hyödyllistä varsinkin kun tiedämme, että vetoisuus ei riipu ajasta – eikä $\mathrm{breezy}(l)$ siis tarvitse enää tilanneargumenttia.

Silloin toimija voi jälleen kerätä tietoa siitä, missä paikoissa vetää ja missä löyhkää, ja päätellä näin turvalliset ruudut kuten lauselogiikalla tehtiin luvussa 5.1.

Tähän päättelyyn tarvittavat säännöt ovat

synkronisia eli samanaikaisia, koska ne kuvaavat yhden ja saman maailman tilan ominaisuuksia.

erotukseksi esimerkiksi seuraajatilanneaksioomista, jotka ovat

diakronisia eli "aikaa halkovia" kuvatessaan miten maailman tila vaihtuu toiseksi.

Synkronisia sääntöjä on kahta päätyyppiä:

Kausaalisääntöjä jotka ilmaisevat, että jokin maailman kätketty piirre aiheuttaa jonkin havaittavissa olevan ilmiön.
Esimerkiksi wumpusta ei havainnoida suoraan, vaan sen aiheuttaman löyhkän perusteella:

$$\forall l_1, l_2, s.\, \mathrm{at}(\mathrm{wumpus}, l_1, s) \wedge \mathrm{adjacent}(l_1, l_2) \Rightarrow \mathrm{smelly}(l_2).$$

Kausaalisäännnöillä päätteleviä järjestelmiä kutsutaan *mallipohjaisiksi (model-based)*, koska niillä on sisäinen "malli" siitä, miten maailma toimii. (Tämä mallin käsite on sukua muttei identtinen loogiselle mallille $\mathfrak{M}$.)

Diagnoosisääntöjä jotka ilmaisevat syy-seuraussuhteiden toisen suunnan, eli miten saaduista havainnoista voi päätellä maailman jonkin kätketyn piirteen.
Esimerkiksi kaava (38) on (hyvin naiivi) tällainen sääntö.

Pelkistä diagnoosisäännöistä ei aina ole helppoa päätellä tarpeeksi vahvoja johtopäätöksiä. Siksi esimerkiksi lääketieteellisessä diagnoosissa on alettu siirtyä massiivisista diagnoosisääntöpankeista järjestelmiin, joiden kausaalisäännöt kuvaavat jonkin mallin kyseisen taudin etenemisprosessista.

Wumpus-esimerkissämme yksinkertaisin tapa määritellä keskeinen käsite "turvallinen ruutu" lienee mallipohjainen sääntö

$$\forall x, t.\, \mathrm{ok}(x) \Leftrightarrow \neg(\mathrm{at}(\mathrm{wumpus}, x, t) \vee \mathrm{pit}(x))$$

kun taas diagnoosisäännöillä pitäisi erikseen selittää kaikki ne mahdolliset havaintoyhdistelmät, joista turvallisuus seuraa.

Jos *aksiomatisointi kuvaa täydellisesti* (ja oikein) sen miten maailma toimii ja sen miten tämä toiminta synnyttää saadut havainnot, **niin** *hyvä päättelykone* pystyy silloin tuottamaan sellaisen kuvauksen maailman tilasta, joka pitää mahdollisimman tarkasti yhtä saatujen havaintojen kanssa.

Predikaattilogiikalla voidaan myös vertailla eri tekoja ja suositella vertailun voittajia. Näin lähestytään luvun 2.4 päämäärähakuisia toimijoita. *Tekojen arvotusjärjestelmä (action-value system)* voi esimerkiksi luokitella teot luokkiin kuten

$$\text{great} > \text{good} > \text{medium} > \text{risky} > \text{deadly} .$$

Wumpus-esimerkissä:

Erinomaisia tekoja ovat poimia kulta tai kiivetä pois sen kanssa.

Hyvä teko on vaikkapa siirtyä tutkimattomaan turvalliseen ruutuun.

Välttävä teko on vaikkapa palata jo tutkittuun (turvalliseen) ruutuun.

Vaarallinen teko on vaikkapa siirtyä ruutuun, jota ei ole pystytty osoittamaan turvalliseksi.

Tappava teko on vaikkapa siirtyä ruutuun, jossa on osoitettu olevan wumpus tai kuoppa.

Esimerkiksi periaate "tarjoa hyvää tekoa ellei erinomaista ole tarjolla" voidaan ilmaista

$$\forall a, s.\, \text{good}(a, s) \wedge (\neg \exists b\, \text{great}(b, s)) \Rightarrow \text{action}(a, s).$$

Näin saadaan kyllä menetelmä löytää kulta ottamatta tarpeettomia riskejä. Kullan löydyttyä pitäisi kuitenkin arvotusjärjestelmää muuttaa suosimaan turvallista paluuta kohden alkuruutua uusien turvallisten ruutujen tutkimisen sijasta.

Tämä lienee helpointa toteuttaa antamalla toimijan tietoon myös se tavoite, johon pyritään:

$$\forall s.\, \text{holding}(\text{gold}, s) \Rightarrow \text{goal\_location}([1, 1], s)$$

kun taas

$$\forall s, l. \neg\, \text{holding}(\text{gold}, s) \wedge \text{ok}(l) \Rightarrow \text{goal\_location}(l, s).$$

5.3. Todistusteoriaa [RN95, luku 9]

Kun tietämys on esitetty (ensimmäisen kertaluvun predikaatti)logiikalla on vuorossa sen käyttö. Tähän tarvitaan *päättelymekanismeja*, jotka kertovat mitä esitetystä tietämyksestä seuraa.

Tarkastellaan joitakin loogisia päättelyjärjestelmiä siltä kannalta, kuinka hyvin ne voisi automatisoida.

Järjestelmien yksityiskohtiin ei kuitenkaan mennä.

5.3.1. Aksiomaattiset eli Hilbert-järjestelmät

Luvussa 1.2 mainittu David Hilbert ehdotti seuraavanlaisia päättelyjärjestelmiä:

- Tarpeeksi *paljon* "itsestään selviä" päättely*aksioomia*.

  Nämä aksioomat kuvailevat loogisia totuuksia, kun taas sovellusalueen aksiomatisointi kuvaili tietämystä.

  Tällaisia voisivat olla esimerkiksi
  - lauselogiikan tautologiat
  - sellaiset predikaattilogiikan kaavat, jotka saadaan korvaamalla tautologioissa kukin lausemuuttuja jollakin kaavalla.
- Mahdollisimman *vähän* päättely*sääntöjä* joilla aksioomista ja aiemmista johtopäätöksistä voi johtaa uusia johtopäätöksiä.

  Tavallisesti otetaan vain [Vää87, luku 5]
  - modus ponens taulukosta 7
  - kvanttoreille yleistyssäännöt "kaavasta $\phi \Rightarrow \psi$ saa päätellä kaavan $\exists x \phi \Rightarrow \psi$ (tai $\phi \Rightarrow \forall x \psi$), *kunhan* muuttujasymboli x ei esiinny vapaana kvantifioinnin ulkopuolelle jäävässä osassa ψ (tai ϕ)".

    Varsinkin kvanttoreita koskevissa päättelyaskelissa esiintyy tällaisia *reunaehtoja (side conditions)*.

Hilbert-järjestelmässä on

- *helppoa* todistaa *yleisiä* tuloksia todistamisen käsitteestä — mikä olikin Hilbertin tavoite
- *vaikeaa* keksiä (mekaanisesti) *yksittäisiä* todistuksia — mikä taas on tekoälyn tarvitsema työkalu.

  Tehtävä "yritä todistaa ϕ" on "yritä purkaa ϕ käyttäen päättelysääntöjä *takaperin* kunnes kohtaat päättelyaksioomien variantteja".

Todistuksen etsintää on vaikeaa ohjata, koska ei ole tietoa millaisia varianttеja kohden kannattaisi pyrkiä.

Esimerkiksi lauselogiikalle täydellinen Hilbert-järjestelmä saadaan [Nv01, luku 2.5(b)]

aksioomina kaikki lauselogiikan lauseet muotoa

$$\mathbf{F} \Rightarrow \alpha$$

$$\alpha \Rightarrow (\beta \Rightarrow (\alpha \land \beta))$$

$$\alpha \land \beta \Rightarrow \alpha$$

$$\alpha \land \beta \Rightarrow \beta$$

$$\alpha \Rightarrow \alpha \lor \alpha$$

$$\alpha \Rightarrow \alpha \lor \beta$$

$$(\alpha \Rightarrow \gamma) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow (\alpha \lor \beta \Rightarrow \gamma))$$

$$\alpha \Rightarrow (\beta \Rightarrow \alpha) \tag{39}$$

$$(\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma)) \tag{40}$$

missä α, β ja γ ovat mielivaltaisia lauseita.
Negaatio $\neg\alpha$ tulkitaan nyt lyhenteeksi $\alpha \Rightarrow \mathbf{F}$.
Jokainen näistä muodoista on tautologinen ja esittää jotakin luontevaa järkeilyaskelta kuten "jos sekä α että β niin myös α yksinään".

päättelysääntönä modus ponens

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$$

missä tuttuun tapaan johtopäätös α on viivan alapuolella ja sen ehdot $\alpha \Rightarrow \beta$ ja β yläpuolella.

Todistus yksinkertaiselle väitteelle $A \Rightarrow A$ on mutkikas:

$$\dfrac{\dfrac{\begin{array}{c}(40); \alpha = \gamma = A, \beta = (A \Rightarrow A)\\ (A \Rightarrow ((A \Rightarrow A) \Rightarrow A)) \Rightarrow ((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A))\end{array} \quad \begin{array}{c}(39); \alpha = A, \beta = (A \Rightarrow A)\\ A \Rightarrow ((A \Rightarrow A) \Rightarrow A)\end{array}}{(A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A)} \quad \begin{array}{c}(39); \alpha = \beta = A\\ A \Rightarrow (A \Rightarrow A)\end{array}}{A \Rightarrow A}$$

5.3.2. Luonnollisen päättelyn järjestelmät

"Luonnollisen" päättelyn (Natural Deduction) järjestelmät [Nv01, luku 1.2] mallintavat sitä, miten ihminen rakentaa huolellisia matemaattisia todistuksia.

- *Tuontisäännöillä (introduction rules)* voidaan johtopäätökseen lisätä loogisia konnektiiveja:

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \wedge I$$
$$\frac{\alpha}{\alpha \vee \beta} \vee I_1$$
$$\frac{\beta}{\alpha \vee \beta} \vee I_2$$
$$\frac{\begin{matrix}[\alpha] \\ \vdots \\ \beta\end{matrix}}{\alpha \Rightarrow \beta} \Rightarrow I \tag{41}$$

  Tuontisäännöt löydetään miettimällä konnektiivien merkityksiä.

- Säännössä (41) tehtiin *oletus (assumption)* α, jonka avulla voitiin todistaa β.

  Tämä oletus täytyy myös *purkaa (discharge)* tyytymällä heikompaan johtopäätökseen "jos α niin β".

  Nyt luvun 5.3.1 patologinen todistusesimerkki on suoraan

$$\frac{\overset{1.}{[A]}}{A \Rightarrow A} \Rightarrow I, 1.$$

  missä on osoitettu purettava oletus indeksoimalla; suuressa todistuksessa voi olla useita eri *avoimia (open)* oletuksia purkua odottamassa.

- Kvanttoreille tuontisäännöt ovat [Nv01, luku 4.1(b)]:

  Universaalikvanttorille sääntö

$$\frac{\textsc{Subst}(\{x/y\}, \alpha)}{\forall x \alpha} \forall I$$

missä $\textsc{Subst}(\{x/y\}, \alpha)$ tarkoittaa sellaista kaavaa, joka saadaan korvaamalla kaavassa α jokainen muuttujasymbolin x vapaa esiintymä muuttujasymbolin y esiintymällä.

Tämä on syntaktinen vastine luvun 5.2.2 semanttiselle operaatiolle $[x = \ldots]$.

Reunaehtona y ei saa esiintyä vapaana johtopäätöksessä $\forall x \alpha$ eikä missään oletuksessa josta ehto $\textsc{Subst}(\{x/y\}, \alpha)$ päätelttiin.

Reunaehtoon käy esimerkiksi sellainen y jota ei ole ennen käytetty.

Eksistenssikvanttorille sääntö

$$\frac{\textsc{Subst}(\{x/t\}, \alpha)}{\exists x \alpha} \exists I$$

missä t on mielivaltainen termi.

- Vastaavasti turhia konnektiiveja voidaan hävittää ehdoista *poistosäännöillä (elimination rules):*

$$\frac{\alpha \wedge \beta \quad \begin{array}{c}[\alpha, \beta] \\ \vdots \\ \gamma\end{array}}{\gamma} \wedge E$$

$$\frac{\alpha \vee \beta \quad \begin{array}{c}[\alpha] \\ \vdots \\ \gamma\end{array} \quad \begin{array}{c}[\beta] \\ \vdots \\ \gamma\end{array}}{\gamma} \vee E \tag{42}$$

$$\frac{\alpha \Rightarrow \beta \quad \alpha \quad \begin{array}{c}[\beta] \\ \vdots \\ \gamma\end{array}}{\gamma} \Rightarrow E \tag{43}$$

$$\frac{\forall x \alpha \quad \begin{array}{c}[\textsc{Subst}(\{x/t\}, \alpha)] \\ \vdots \\ \gamma\end{array}}{\gamma} \forall E$$

$$\frac{\exists x \alpha \quad \begin{array}{c}[\textsc{Subst}(\{x/y\}, \alpha)] \\ \vdots \\ \gamma\end{array}}{\gamma} \exists E \tag{44}$$

Eksistenssikvanttorin poistosäännöllä (44) on reunaehto, että y ei saa esiintyä vapaana ehdoissa $\exists x \alpha$ ja γ eikä missään muussa oletuksessa kuin $\textsc{Subst}(\{x/y\}, \alpha)$.

Poistosäännöt löydetään kumoamalla tuontisääntöjä.

Esimerkiksi disjunktion poistosääntö (42) on kuitenkin hankala todistuksen etsinnässä: millaiset oletukset α ja β kannattaisi tehdä, kun tiedetään vain haluttu johtopäätös γ?

Toisin sanoen, todistuksen etsinnässä *haarautumisaste on ääretön*.

Jos on päädytty valitsemaan tai kehittämään kokonaan uusi logiikka tietämyksen esittämiseen, niin sille lienee helpointa rakentaa todistusteoria aloittamalla luonnollisen päättelyn periaatteista.

Tarkastellaan seuraavaa päättelytehtävää [RN95, luku 9.2]:

> "Amerikkalainen on rikollinen jos hän myy aseita vihamieliselle valtiolle. Vihamielisellä valtiolla Nono on ohjuksia. Kaikki sen ohjukset myi amerikkalainen eversti West.
>
> Päättele tästä, että West on rikollinen."

Esitetään yksinkertaisuuden vuoksi päättely epäformaalisti, mutta kuitenkin niin tarkasti, että se voitaisiin myös formalisoida.

Aksiomatisoidaan ensiksi tietämys:

- "Amerikkalainen on rikollinen jos hän myy aseita vihamieliselle valtiolle."

$$\begin{aligned}\forall x, y, z.\, \mathrm{american}(x) \wedge \mathrm{weapon}(y) \wedge \mathrm{nation}(z) \wedge \\ \mathrm{hostile}(z) \wedge \mathrm{sells}(x, z, y) \Rightarrow \mathrm{criminal}(x)\end{aligned} \tag{45}$$

- "Vihamielisellä valtiolla Nono on ohjuksia."

$$\mathrm{enemy}(\mathrm{Nono}, \mathrm{America}) \tag{46}$$

$$\mathrm{nation}(\mathrm{Nono}) \tag{47}$$

$$\exists x.\, \mathrm{owns}(\mathrm{Nono}, x) \wedge \mathrm{missile}(x) \tag{48}$$

- "Kaikki sen ohjukset myi amerikkalainen eversti West."

$$\text{american}(\text{West}) \tag{49}$$

$$\forall x.\, \text{owns}(\text{Nono}, x) \wedge \text{missile}(x) \Rightarrow \text{sells}(\text{West}, \text{Nono}, x) \tag{50}$$

- Ja vielä:

$$\forall x.\, \text{missile}(x) \Rightarrow \text{weapon}(x) \tag{51}$$

$$\forall x.\, \text{enemy}(x, \text{America}) \Rightarrow \text{hostile}(x) \tag{52}$$

$$\text{Nation}(\text{America}) \tag{53}$$

Voidaan päätellä seuraavin askelin:

1. Aksioomasta (48) saadaan eksistenssikvanttorin poistosäännöllä

$$\text{owns}(\text{Nono}, \text{M1}) \wedge \text{missile}(\text{M1})$$

koska

$$\frac{(48) \quad \begin{array}{c}[\text{owns}(\text{Nono}, \text{M1}) \wedge \text{missile}(\text{M1})] \\ \text{owns}(\text{Nono}, \text{M1}) \wedge \text{missile}(\text{M1})\end{array}}{\text{owns}(\text{Nono}, \text{M1}) \wedge \text{missile}(\text{M1})}\,\exists E$$

kun valitaan uusi ennen käyttämätön muuttujasymboli $y = \text{M1}$.

2. Askeleesta 1 saadaan konjunktion poistolla kaavat

$$\text{owns}(\text{Nono}, \text{M1}) \tag{54}$$

$$\text{missile}(\text{M1}) \tag{55}$$

koska esimerkiksi

$$\frac{1 \quad \begin{array}{c}[(54), (55)] \\ (54)\end{array}}{(54)}\,\wedge E.$$

3. Aksioomasta (51) saadaan universaalikvanttorin poistolla

$$\text{missile}(\text{M1}) \Rightarrow \text{weapon}(\text{M1})$$

koska

$$\frac{(51) \quad \begin{matrix}[\text{missile}(\text{M1}) \Rightarrow \text{weapon}(\text{M1})] \\ \text{missile}(\text{M1}) \Rightarrow \text{weapon}(\text{M1})\end{matrix}}{\text{missile}(\text{M1}) \Rightarrow \text{weapon}(\text{M1})} \forall E$$

kun valitaan nokkelasti *sama* $t = \text{M1}$ kuin askeleessa 1.

Nyt näin voidaan tehdä, koska tällä päättelyaskeleella ei ole reunaehtoja.

Miten tietokone tämän keksisi?

4. Kaavasta (55) ja askeleesta 3 saadaan päättelysäännöllä Modus Ponens

$$\text{weapon}(\text{M1})$$

koska Modus Ponens

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\beta} MP$$

on implikaation poistosäännön (43) erikoistapaus $\gamma = \beta$:

$$\frac{\alpha \Rightarrow \beta \quad \alpha \quad \begin{matrix}[\beta] \\ \beta\end{matrix}}{\beta} \Rightarrow E$$

Siis Modus Ponens on järjestelmän *luvallinen (admissible) johdettu* päättelysääntö.

5. Aksioomasta (50) saadaan universaalikvanttorin poistolla

$$\text{owns}(\text{Nono}, \text{M1}) \wedge \text{missile}(\text{M1}) \\ \Rightarrow \text{sells}(\text{West}, \text{Nono}, \text{M1}).$$

6. Askelista 5 ja 1 saadaan päättelysäännöllä Modus Ponens

$$\text{sells}(\text{West}, \text{Nono}, \text{M1}).$$

7. Aksioomasta (45) saadaan kolmen universaalikvanttorin poistolla

$$\text{american}(\text{West}) \wedge \text{weapon}(\text{M1}) \wedge \text{nation}(\text{Nono}) \wedge \\ \text{hostile}(\text{Nono}) \wedge \text{sells}(\text{West}, \text{Nono}, \text{M1}) \Rightarrow \\ \text{criminal}(\text{West})$$

8. Aksioomasta (52) saadaan universaalikvanttorin poistolla

$$\mathrm{enemy}(\mathrm{Nono}, \mathrm{America}) \Rightarrow \mathrm{hostile}(\mathrm{Nono})$$

9. Aksioomasta (46) ja askeleesta 8 saadaan päättelysäännöllä Modus Ponens

$$\mathrm{hostile}(\mathrm{Nono}).$$

10. Aksioomista (49) ja (47) sekä askelista 4, 6 ja 9 saadaan konjunktion tuonneilla

$$\begin{array}{l}\mathrm{american}(\mathrm{West}) \wedge \mathrm{weapon}(\mathrm{M1}) \wedge \mathrm{nation}(\mathrm{Nono}) \wedge \\ \mathrm{hostile}(\mathrm{Nono}) \wedge \mathrm{sells}(\mathrm{West}, \mathrm{Nono}, \mathrm{M1})\end{array}$$

koska esimerkiksi

$$\frac{(49) \quad 4}{(49) \wedge 4} \wedge I .$$

11. Askelista 7 ja 10 saadaan vihdoin päättelysäännöllä Modus Ponens haluttu lopputulos

$$\mathrm{criminal}(\mathrm{West}).$$

5.3.3. Yleistetty Modus Ponens

Luvun 5.3.2 esimerkin olennainen kulku oli päätellä sopivia atomikaavoja toisista atomikaavoista Modus Ponensilla.

Tämä on niin yleinen päättelytapa, että kannattaa oikaista sen mutkia erityisellä päättelysäännöllä *yleistetty (generalised) Modus Ponens:*

$$\frac{p_1' \quad \ldots \quad p_n' \quad p_1 \wedge \ldots \wedge p_n \Rightarrow q}{\textsc{Subst}(\theta, q)}\, GMP \tag{56}$$

missä

- jokainen p_i, p_i' ja q on atomikaava
- θ on sellainen sidonta, jolla $\textsc{Subst}(\theta, p_i)$ ja $\textsc{Subst}(\theta, p_i')$ ovat *syntaktisesti sama* kaava kaikilla $1 \leq i \leq n$.

  Nämä sidonnat ovat yksittäisten muuttujasymbolien sidontojen x_i/t_i joukkoja. Lisäksi vaaditaan, ettei niissä ole kehiä: esimerkiksi $\{x/f(y), y/x\}$ ei ole luvallinen sidonta, koska $x = f(y) = f(x) = f(f(y)) = f(f(x)) = \ldots$ olisi kehä joka johtaisi äärettömän pitkään termiin.

  Minimaalinen mutta silti riittävä sidonta voidaan laskea *samastusalgoritmilla (unification algorithm) 13.*

Yleistetty Modus Ponens (56) on erityisen luonteva luvussa 5.1 mainittujen *Hornin lauseiden predikaattiloogisille muodoille*. Nämä ovat kaavoja

$$\forall \vec{x}.p_1 \wedge \ldots \wedge p_n \Rightarrow q \tag{57}$$

missä universaalikvantifionti $\forall \vec{x}$ (joka jätetään usein kirjoittamatta) käsittää täsmälleen ne muuttujasymbolit, jotka tässä implikaatiossa esiintyvät vapaina.

Tämä kvantifiointi huolehtii siitä, että säännön ehtojen muuttujat ovat aina *erillisiä*. Samoin johtopäätöksen muuttujat kannattaa pitää erillään ehtojen muuttujista. (Vrt. vapaiden muuttujien ongelma luvussa 5.2.2.)

```
function UNIFY(x, y) returns a substitution to make x and y identical, if possible
   UNIFY-INTERNAL(x, y, { })
-------------------------------------------------------------------------------------
function UNIFY-INTERNAL(x, y, θ) returns a substitution to make x and y identical (given θ)
   inputs: x, a variable, constant, list, or compound
           y, a variable, constant, list, or compound
           θ, the substitution built up so far

   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
        return UNIFY-INTERNAL(ARGS[x], ARGS[y], UNIFY-INTERNAL(OP[x], OP[y], θ)
   else if LIST?(x) and LIST?(y) then
        return UNIFY-INTERNAL(REST[x], REST[y], UNIFY-INTERNAL(FIRST[x], FIRST[y], θ))
   else return failure
-------------------------------------------------------------------------------------
function UNIFY-VAR(var, x, θ) returns a substitution
   inputs: var, a variable
           x, any expression
           θ, the substitution built up so far

   if {var/val} ∈ θ
      then return UNIFY-INTERNAL(val, x, θ)
   else if {x/val} ∈ θ
      then return UNIFY-INTERNAL(var, val, θ)
   else if var occurs anywhere in x /* occur-check */
      then return failure
   else return add {x/var} to θ
```

HUOM! Viimeisellä rivillä tulisi olla var/x!

Tämän samastusalgoritmin perusidea on täydentää sidontaa θ vertaamalla samastettavia rakenteita x ja y toisiinsa rekursiivisesti. *Esiintymistarkistus* (askel $occur\text{-}check$) estää kehät.

Nämä rakenteet ovat

joko muuttujasymboleita ($\textsc{Variable?}$)

tai rakenteita ($\textsc{Compound?}$) jotka koostuvat

- operaattorista ($\textsc{Op}$), joka on predikaatti- tai funktiosymboli
- ja sen argumenttilistasta ($\textsc{Args}$), jota käydään läpi jäsen jäseneltä ($\textsc{First}$, $\textsc{Rest}$).

Algoritmi 13: Samastusalgoritmi [RN95, kuva 10.3].

Luvun 5.3.2 esimerkin tietämys on esitetty Hornin lauseilla, kunhan sitä muunnetaan ensin hieman tekemällä päättelyaskeleet 1 ja 2: aksiooman (48) nojalla on jokin ohjus, joten annetaan sille nimeksi uusi vakiotermi $\mathrm{M1}$.

Toistetaan sitten esimerkki tällä päättelysäännöllä:

1. Vanha askel 4 on nyt

$$\frac{\overset{(55)}{\text{missile}(\mathrm{M1})} \quad \overset{(51)}{\forall x.\,\text{missile}(x) \Rightarrow \text{weapon}(x)}}{\text{weapon}(\mathrm{M1})}\,GMP$$

   koska $\textsc{Unify}(\text{missile}(\mathrm{M1}), \text{missile}(x)) = \{x/\,\mathrm{M1}\}$.

   Tämän askeleen sisällä tehtiin myös vanha askel 3. Ennen vaadittiin nokkeluutta valittaessa sopivaa arvoa $\mathrm{M1}$ muuttujasymbolille x, mutta nyt riittää *syntaktinen* samastustoimitus. Valinta on siis rakennettu käytetyn yleistetyn Modus Ponens -päättelysäännön (56) *sisään*.

2. Vanha askel 9 on samanlainen päättely:

$$\frac{\overset{(46)}{\text{enemy}(\text{Nono}, \text{America})} \quad \overset{(52)}{\forall x.\,\text{enemy}(x, \text{America}) \Rightarrow \text{hostile}(x)}}{\text{hostile}(\text{Nono})}\,GMP$$

3. Vanha askel 6 on samanlainen päättely, mutta nyt atomikaavoja on useita:

$$\frac{\overset{(54)}{\text{owns}(\text{Nono}, \mathrm{M1})} \; \overset{(55)}{\text{missile}(\mathrm{M1})} \; \overset{(50)}{\forall x.\,\text{owns}(\text{Nono}, x) \wedge \text{missile}(x) \Rightarrow \text{sells}(\text{West}, \text{Nono}, x)}}{\text{sells}(\text{West}, \text{Nono}, \mathrm{M1})}$$

4. Vanha loppuaskel 11 on seuraavalla sivulla.

   Enää ei tarvita erillisten atomikaavojen yhdistelyä konjunktioin kuten vanhassa askeleessa 10, koska tämäkin esivaihe on sisäänrakennettu päättelysääntöön (56).

$$\frac{1 \quad \overset{(47)}{\mathrm{nation}(\mathrm{Nono})} \quad 2 \quad 3 \quad \overset{(45)}{\forall x, y, z.\, \mathrm{american}(x) \land \mathrm{weapon}(y) \land \mathrm{nation}(z) \land \mathrm{hostile}(z) \land \mathrm{sells}(x, z, y) \Rightarrow \mathrm{criminal}(x)}}{\mathrm{criminal}(\mathrm{West})}\ GMP$$

Edellinen esimerkki ratkaistiin *eteenpäin ketjuttamalla (forward chaining):* tietämyskannan KB jo sisältämien faktojen perusteella *laukaistiin (fire)* jokin sääntö, joka tuotti uuden faktan kantaan lisättäväksi.

Jokainen tietämyskannan KB sääntö muotoa (57) luetaan siis "jos tietämyskannassa on jo faktat $p_1, \ldots, p_n$ niin lisää siihen myös fakta q".

Näin jatketaan kunnes odotettu fakta ilmestyy kantaan tai ei enää ole mitään lisättävää:

procedure FORWARD-CHAIN(*KB, p*)

if there is a sentence in *KB* that is a renaming of *p* **then return**
Add *p* to *KB*
for each $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ **in** *KB* such that for some i, UNIFY$(p_i, p) = \theta$ succeeds **do**
FIND-AND-INFER(*KB*, $[p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n], q, \theta$)
end

---

procedure FIND-AND-INFER(*KB, premises, conclusion,* θ)

if *premises* = [ ] **then**
FORWARD-CHAIN(*KB*, SUBST(θ, *conclusion*))
else for each p' **in** *KB* such that UNIFY(p', SUBST(θ, FIRST(*premises*))) = θ_2 **do**
FIND-AND-INFER(*KB*, REST(*premises*), *conclusion*, COMPOSE(θ, θ_2))
end

Aliohjelma $\textsc{Compose}(\theta, \theta_2)$ palauttaa sellaisen sidonnan θ', joka yhdistää argumentteinaan saamansa sidonnat θ ja θ_2 peräkkäin: toisin sanoen, sidonnan θ' käyttö yksinään tuottaa aina saman tuloksen kuin jos olisi käytetty ensin sidontaa θ ja sen perään sidontaa θ_2.

Algoritmi 14: Eteenpäin ketjuttava päättelykone [RN95, kuva 9.1].

- Eteenpäinketjutus *ei välttämättä pysähdy* jos tietämyskannassa KB on argumentillisia funktiosymboleja: esimerkiksi kannasta

$$p(a) \qquad \forall x.p(x) \Rightarrow p(f(x))$$

ketjuttuu äärettömästi faktoja $p(a)$, $p(f(a))$, $p(f(f(a)))$,. . .
Valitettavasti esimerkiksi luvun 5.2.3 tilannekalkyylissä on väistämättä tulosfunktio result.

- Eteenpäinketjutuksella *ei ole näkemystä* siitä tuloksesta, mihin kannattaisi pyrkiä, vaan se toimii pelkästään tietämyskannan KB kulloisenkin sisällön ohjaamana.

  Se on siis eräänlaista leveyssuuntaista etsintää.

  Tällaisia ohjelmia sanotaan *tieto-ohjatuiksi (data-driven, data-directed)*.

Tehokkaampaa (ainakin muistin- ja usein myös ajankäytöltään) on *taaksepäin ketjuttaminen (backward chaining):*

- Lähdetään liikkeelle siitä faktasta q, joka halutaan todistaa.
- Valitaan kokeiltavaksi sellainen sääntö r muotoa (57) jonka johtopäätöksenä voitaisiin saada tämä q.
- Katsotaan, mitkä kaikki faktat täytyisi lisäksi todistaa, jotta voitaisiin käyttää tätä valittua sääntöä r.

  Todistetaan nekin vuorostaan taaksepäin ketjuttamalla.
- Jatketaan kunnes etsitty todistus löytyy tai kaikki vaihtoehdot r on kokeiltu.

Jokainen tietämyskannan KB sääntö r

$$\forall \vec{x}.\, \underbrace{p_1 \wedge \ldots \wedge p_n}_{\text{runko}} \Rightarrow \underbrace{q'}_{\text{pää}} \tag{58}$$

luetaan siis "jos tahdot tehdä johtopäätöksen q', niin yksi tapa on tehdä ensin johtopäätökset $p_1, \ldots, p_n$".

Sääntöjä kokeillaan siis seuraavasti:

1. Valitaan ensin jokin sellainen sääntö r, jonka *pää (head)* q' samastuu haluttuun johtopäätökseen p; olkoon θ' vastaava yleisin samastin.
2. Sitten jatketaan samalla tavalla valitun säännön r *rungon (body)* johtopäätöksien $p'_i = \textsc{Subst}(\theta', p_i)$ metsästämistä.

Algoritmi 15 toimii tällä periaatteella, ja ratkoo osaongelmat p_i' rekursiivisesti. Näin saatu syvyyssuuntainen etsintä on luonteva mutta ei ainoa mahdollinen tapa ketjuttaa taaksepäin.

```
function BACK-CHAIN(KB, q) returns a set of substitutions
  BACK-CHAIN-LIST(KB, [q], {})
-----------------------------------------------------------------
function BACK-CHAIN-LIST(KB, qlist, θ) returns a set of substitutions
  inputs: KB, a knowledge base
          qlist, a list of conjuncts forming a query (θ already applied)
          θ, the current substitution
  static: answers, a set of substitutions, initially empty

  if qlist is empty then return {θ}
  q ← FIRST(qlist)
     for each q'_i in KB such that θ_i ← UNIFY(q, q'_i) succeeds do
        Add COMPOSE(θ, θ_i) to answers
     end
     for each sentence (p_1 ∧ ... ∧ p_n ⇒ q'_i) in KB such that θ_i ← UNIFY(q, q'_i) succeeds do
        answers ← BACK-CHAIN-LIST(KB, SUBST(θ_i, [p_1 ... p_n]), COMPOSE(θ, θ_i)) ∪ answers
     end
  return the union of BACK-CHAIN-LIST(KB, REST(qlist), θ) for each θ ∈ answers
```

HUOM! Muuttujan $answers$ tulisi olla **local** kuten kirjassa eikä **static** kuten kuvassa!

Algoritmi 15: Taaksepäin ketjuttava päättelykone [RN95, kuva 9.2].

Algoritmi 15 ratkoo itse asiassa seuraavaa etsintäongelmaa:

Alkutila saadaan kysymyksestä "päteekö *faktojen konjunktio*

$$q_1 \wedge \ldots \wedge q_m \tag{59}$$

tietämyskannasta KB (joka sisältää jälleen vain Hornin lauseita)?" Alkutila koostuu tuosta *maalista* (59) ja tyhjästä sidonnasta $\emptyset$. Tietämyskanta KB kuuluisi oikeastaan myös etsinnän tiloihin, mutta jätetään se pois koska se ei muutu etsinnän aikana.

Operaattorit saadaan käyttämällä yhtälöä (56) takaperin kuten edellä selitettiin.

Koostukoon etsinnän nykyinen tila ylläolevasta maalista (58) ja sidonnasta θ. Valitaan

- jokin *alimaali (subgoal)* q_i maalista (59), ja
- jokin yhtälön (58) mukainen sääntö r tietämyskannasta KB (mutta ensin *nimetään uudelleen* säännön r kaikki muuttujat aiemmin käyttämättömillä muuttujasymboleilla sen merkitystä kuitenkaan muuttamatta, jotta ne pysyisivät muista erillään)
- siten, että valittu alimaali q_i ja valitun säännön pää q' samastuvat; olkoon vastaava yleisin samastin θ'.

Näiden valintojen määräämä seuraava etsinnän tila koostuu maalista

$$\textsc{Subst}(\theta', q_1 \wedge \ldots \wedge q_{i-1} \wedge p_1 \wedge \ldots \wedge p_n \wedge q_{i+1} \wedge \ldots \wedge q_m)$$

ja sidonnasta $\mathrm{compose}(\theta, \theta')$.

Valittu alimaali q_i korvataan siis valitun säännön rungolla ja sovelletaan vastaavaa sidontaa.

Lopputila on sellainen, jonka maalissa ei ole enää yhtään atomikaavaa jäljellä.

Silloin vastataan kertynyt sidonta θ; se kertoo millaisiin arvoihin alkutilan alkuperäisen maalin (59) vapaat muuttujat piti sitoa jotta etsintä onnisti.

Näillä palautusarvoilla voidaan välittää kutsujalle esimerkiksi niitä result-termejä, jotka esittivät tietämyskannan KB ehdottamaa toimintasuunnitelmaa luvussa 5.2.3.

Tätä etsintäongelmaa voi ajatella *ristiriitatodistuksen* etsintänä:

- Alkuperäinen maali (59) on Hornin lause

$$\forall \vec{x}. q_1 \wedge \ldots \wedge q_m \Rightarrow \mathbf{F}, \tag{60}$$

missä $\vec{x}$ ovat nuo vastauksia palauttavat muuttujat. Sen ekvivalentti muotoilu on

$$\neg \exists \vec{x}. q_1 \wedge \ldots \wedge q_m,$$

ja tarkoituksena on osoittaa että sen lisääminen ristiriidattomaksi oletettuun tietämyskantaan KB aiheutti ristiriidan.

- Tämä tehdään palauttamalla vastaussidonnassa sopivat arvot muuttujille $\vec{x}$.
- Ristiriita havaitaan päätymällä lopputilan tyhjään konjunktioon $\mathbf{T}$.

Logiikkaohjelmointikieli Prolog [RN95, luku 10.3] käyttää tätä periaatetta siten, että:

- Aina valitaan *ensimmäinen* alimaali q_1, jolloin maali toimii *rekursiokutsupinona*.

  Tämä valinta on vapaa, koska jokainen alimaali on joskus valittava ja eri valintajärjestykset ovat keskenään vaihtoehtoiset.
- Tietämyskannan eli *ohjelman* KB sääntöjä kokeillaan aina niiden *kirjoitusjärjestyksessä*.

  Tässä valinnassa taas pitäisi taata, että jokaista sääntöä tullaan kokeilemaan. Kirjoitusjärjestys(kään) ei sitä takaa.
- Ristiriidan etsintää suoritetaan *syvyyssuuntaisesti* (luku 3.1.3).
- Samastusalgoritmista jätetään esiintymistarkistus pois.

Nämä valinnat takaavat toisaalta hyvän suorituskyvyn mutta toisaalta tekevät etsinnästä *epätäydellisen* ja joskus jopa loogisesti *virheellisen*. Kuitenkin Prolog on erinomainen toteutuskieli tekoälyohjelmille, koska jo sen omat perusperiaatteet ovat varsin ilmaisuvoimaisia.

Tehdään vielä kerran luvun 5.3.2 esimerkki, nyt Prolog-tyylisellä taaksepäinketjuttamisella. Käytetään myös uusia vastausmuuttujiamme, ja kysytään "kuka on rikollinen?"

1. Alkutilanteessa maali on

$$\mathrm{criminal}(x),$$

   missä x on vastausmuuttuja, ja sijoitus on $\emptyset$.

Loogisesti vastaväitämme siis että "rikollisia ei ole" ja osoitamme sitten tämän vastaväitteen vääräksi etsimällä jonkin rikollisen.

2. Ainoa mahdollisuus on soveltaa sääntöä (45). Muuttujien uudelleennumeroinnin jälkeen saadaan

$$\begin{aligned}&\mathrm{american}(x) \wedge \mathrm{weapon}(y_1) \wedge \mathrm{nation}(z_1) \wedge \\ &\mathrm{hostile}(z_1) \wedge \mathrm{sells}(x, z_1, y_1)\end{aligned}$$

ja sijoitukseksi tulee $\{x_1/x\}$.

3. Ainoa mahdollisuus on soveltaa faktaa (49)

$$\mathrm{weapon}(y_1) \wedge \mathrm{nation}(z_1) \wedge \mathrm{hostile}(z_1) \wedge \mathrm{sells}(\mathrm{West}, z_1, y_1)$$

ja sijoitukseksi tulee $\{x_1/x, x/\,\mathrm{West}\}$.

4. Ainoa mahdollisuus on soveltaa sääntöä (51). Muuttujien uudelleennumeroinnin jälkeen saadaan

$$\mathrm{missile}(y_1) \wedge \mathrm{nation}(z_1) \wedge \mathrm{hostile}(z_1) \wedge \mathrm{sells}(\mathrm{West}, z_1, y_1)$$

ja sijoitukseksi tulee $\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1\}$.

5. Ainoa mahdollisuus on soveltaa faktaa (55). Saadaan

$$\mathrm{nation}(z_1) \wedge \mathrm{hostile}(z_1) \wedge \mathrm{sells}(\mathrm{West}, z_1, \mathrm{M1})$$

ja sijoitukseksi tulee $\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}\}$.

6. Ensimmäinen mahdollisuus on soveltaa faktaa (47). Saadaan

$$\mathrm{hostile}(\mathrm{Nono}) \wedge \mathrm{sells}(\mathrm{West}, \mathrm{Nono}, \mathrm{M1})$$

ja sijoitukseksi tulee $\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}, z_1/\,\mathrm{Nono}\}$. Muistetaan samalla toisena mahdollisuutena fakta (53) siltä varalta ettei tämä ensimmäinen vaihtoehto onnistuisikaan.

7. Ainoa mahdollisuus on soveltaa sääntöä (52). Muuttujien uudelleennumeroinnin jälkeen saadaan

$$\mathrm{enemy}(\mathrm{Nono}, \mathrm{America}) \wedge \mathrm{sells}(\mathrm{West}, \mathrm{Nono}, \mathrm{M1})$$

ja sijoitukseksi tulee

$$\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}, z_1/\,\mathrm{Nono}, x_7/\,\mathrm{Nono}\}\,.$$

8. Ainoa mahdollisuus on soveltaa faktaa (46). Saadaan

$$\mathrm{sells}(\mathrm{West}, \mathrm{Nono}, \mathrm{M1})$$

ja sijoituksena säilyy

$$\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}, z_1/\,\mathrm{Nono}, x_7/\,\mathrm{Nono}\}\,.$$

9. Ainoa mahdollisuus on soveltaa sääntöä (50). Muuttujien uudelleennumeroinnin jälkeen saadaan

$$\mathrm{owns}(\mathrm{Nono}, \mathrm{M1}) \land \mathrm{missile}(\mathrm{M1})$$

ja sijoituksena säilyy

$$\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}, z_1/\,\mathrm{Nono}, x_7/\,\mathrm{Nono}\}\,.$$

10. Ainoa mahdollisuus on soveltaa faktaa (54). Saadaan

$$\mathrm{missile}(\mathrm{M1})$$

ja sijoituksena säilyy

$$\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}, z_1/\,\mathrm{Nono}, x_7/\,\mathrm{Nono}\}\,.$$

11. Ainoa mahdollisuus on soveltaa faktaa (55). Saadaan vihdoin tyhjä maali, eli ristiriita on löytynyt. Sijoituksena säilyy

$$\{x/x_1, x_1/\,\mathrm{West}, x_3/y_1, y_1/\,\mathrm{M1}, z_1/\,\mathrm{Nono}, x_7/\,\mathrm{Nono}\}$$

ja sen perusteella vastausmuuttuja x sai arvokseen West.

Vaihtoehtoon 6 ei siis tällä kertaa tarvinnut palata. Jos sittenkin palattaisiin, niin saataisiin seuraava sellainen rikollinen x, joka kyetään päättelemään, jne.

5.3.4. Resoluutio [RN95, luvut 9.6 ja 9.7] [Nil98, luvut 14 ja 16]

Jos tietämystä ei voikaan esittää Hornin lauseilla muodossa (57), niin silloin ei luvun 5.3.3 yleistetty Modus Ponens riitä, vaan on siirryttävä käyttämään yleisempiä päättelymekanismeja.

Resoluutio (resolution) on tunnetuin automatisoituva päättelyjärjestelmä *koko* ensimmäisen kertaluvun predikaattilogiikalle.

Tarkemmin sanoen kyseessä on *ristiriitatäydellisyys (refutation completeness)*: jos annettu tietämys on sisäisesti ristiriitaista, niin on olemassa äärellinen resoluutiopäättelyaskelten jono, joka löytää jonkin ristiriidan.

Tätä voidaan hyödyntää kuten luvun 5.3.3 taaksepäin ketjuttamisessa: oletetaan itse tietämyskanta KB ristiriidattomaksi, lisätään siihen annetun kyselyn negaatio, ja aletaan etsimään ristiriitaa.

Myös vastaussidonnat käsitellään kuten taaksepäin ketjuttamisessa.

Tietämys muunnetaan resoluutiota varten ensin *konjunktiiviseen normaalimuotoon (conjunctive normal form)*

$$\mathrm{CNF}(\phi) \equiv \bigwedge_{1 \leq i \leq m} D_i, \text{ missä}$$

$$D_i \equiv \forall \vec{x}_i . \bigvee_{1 \leq j \leq n_i} L_{i,j}$$

ja jokainen $L_{i,j}$ on *literaali (literal)*: atomikaava tai sellaisen negaatio. Jokainen $\vec{x}_i$ sisältää jälleen täsmälleen ne muuttujat, jotka esiintyvät vapaana kyseisessä disjunktiossa D_i.

Voisimme käyttää myös *implikatiivista (implicative)* normaalimuotoa, jossa D_i esitettäisiinkin implikaationa

$$\forall \vec{x_i}. P_{i,1} \wedge \ldots \wedge P_{i,k_i} \Rightarrow P_{i,k_i+1} \vee \ldots \vee P_{i,n_i}$$

missä $P_{i,1}, \ldots, P_{i,k_i}$ ovat ne atomikaavat, jotka esiintyivät negatiivisina eli muodossa $L_{i,1} = \neg P_{i,1}$ jne.

- Kun $n_i = k_i + 1$, eli implikaation oikealla puolella on tasan yksi (1) atomikaava, on kyseessä tuttu Hornin lause: fakta ($k_i = 0$) tai sääntö ($k_i > 0$).
- Kun $n_i > k_i + 1$, on kyseessä sellainen sääntö, jolla on *useita* vaihtoehtoisia tuloksia: "faktoista $P_{i,1}, \ldots, P_{i,k_i}$ seuraa fakta P_{i,k_i+1} tai P_{i,k_i+1} tai. . . ".
  Tällaista *disjunktiivista* informaatiota ei Hornin lauseilla voinut esittää.
- Kun $n_i = k_i$, on kyseessä $\forall \vec{x_i}. P_{i,1} \wedge \ldots \wedge P_{i,k_i} \Rightarrow \mathbf{F}$, eli taaksepäin ketjutuksen maalikaava (60), eli negaatio $\neg \exists \vec{x_i}. P_{i,1} \wedge \ldots \wedge P_{i,k_i}$.
  Tällaista *negatiivista* informaatiota ei Hornin lauseilla voinut esittää.

Normalisointi voidaan suorittaa seuraavasti:

1. **Korvaa lyhennysmerkinnät määritelmillään** paitsi '$\forall$' ja '$\vee$'.
   Esimerkiksi "jokainen ihminen on laulun arvoinen" on aluksi muotoa

   $$\forall x. \operatorname{person}(x) \Rightarrow \exists y. \operatorname{song}(y) \wedge \operatorname{deserves}(x, y)$$

   mutta muuntuu muotoon

   $$\forall x. \neg \operatorname{person}(x) \vee \exists y. \operatorname{song}(y) \wedge \operatorname{deserves}(x, y). \qquad (61)$$

2. **Kvantifioi vapaat muuttujat** lisäämällä koko kaavan eteen universaalikvanttori '$\forall x$' jokaiselle muuttujasymbolille 'x' joka esiintyy vapaana.
   Tämä noudattaa määritelmää (32).
3. **Vie negaatiot sisään** eli sovella *uudelleenkirjoitussääntöjä (rewrite rules)*

$$\neg\left(\alpha \, {}^{\vee}_{\wedge} \, \beta\right) \text{ muotoon } (\neg\alpha) \, {}^{\wedge}_{\vee} \, (\neg\beta)$$

$$\neg {}^{\exists}_{\forall} x\alpha \text{ muotoon } {}^{\forall}_{\exists} x\neg\alpha$$

$$\neg\neg\alpha \text{ muotoon } \alpha$$

niin kauan kuin mahdollista.

Nämä säännöt säilyttävät selvästi kaavan semantiikan.

4. **Uudelleennimeä muuttujat** siten, että sama muuttujasymboli esiintyy korkeintaan yhdessä kvanttorissa.

   Näin muuttujat eivät enää voi sekaantua toisiinsa, kun kvanttoreita myöhemmin käsitellään.

5. **Tuo kvanttorit ulos** koko kaavan eteen uudelleenkirjoitussäännöillä

$$\alpha \circ (Qx\beta) \text{ muotoon } Qx(\alpha \circ \beta),$$

   missä '$\circ$' on konnektiivi '$\vee$' tai '$\wedge$' ja Q on kvanttori '$\exists$' tai '$\forall$', sekä sen toiset muodot konnektiivien $\circ$ vaihdannaisuuden suhteen. (Eli $\alpha \circ \beta$ tarkoittaa samaa kuin $\beta \circ \alpha$.)

   On työlästä osoittaa näidenkin sääntöjen säilyttävän kaavan semantiikan. Se perustuu huomioon, että edellisen askeleen nojalla muuttujasymboli x ei enää esiinny vapaana kaavassa α.

   Esimerkiksi kaava (61) muuntuu muotoon

$$\forall x \exists y . \neg \, \mathrm{person}(x) \vee \mathrm{song}(y) \wedge \mathrm{deserves}(x, y). \qquad (62)$$

6. **Skolemisoi** tarkastelemalla edellisen vaiheen tuottamaa kaavaa

$$Q_1 x_1 \ldots Q_n x_n . \alpha, \qquad (63)$$

   missä α ei enää sisällä kvanttoreita. Poista *kvanttoriprefiksin (quantifier prefix)* vasemmanpuoleisin eksistenssikvanttori $Q_i = \exists$ siirtymällä kaavaan

$$Q_1 x_1 \ldots Q_{i-1} x_{i-1} Q_{i+1} x_{i+1} \ldots Q_n x_n . \\ \textsc{Subst}(\{x_i / f_i(x_1, \ldots, x_{i-1})\}, \alpha),$$

missä f_i on uusi funktiosymboli (tai vakio jos $i = 1$), jota ei ole aikaisemmin esiintynyt tietämyskannassa KB.

Toista tätä kunnes jäljellä on vain universaalikvanttoreita.

Kaavan (63) mukaan jokaiselle muuttujien $x_1, \ldots, x_{i-1}$ mahdolliselle arvoyhdistelmälle $\vec{u} = \langle u_1, \ldots, u_{i-1} \rangle \in \mathcal{U}^{i-1}$ on olemassa ainakin yksi arvo $u_i \in \mathcal{U}$ joka käy muuttujalle x_i. Funktio f_i esittää vastaavaa operaatiota, joka poimii jonkin sopivan arvon, eli semanttisesti "$f_i(\vec{u}) = u_i$".

Funktiosymbolia f_i nimitetään *Skolem*-funktioksi (tai -vakioksi) tekniikan kehittäjän, norjalaisen loogikon Thoralf Skolemin mukaan.

Esimerkiksi kaava (62) muuttuu muotoon

$$\begin{aligned} \forall x. \neg\, \mathrm{person}(x) \vee \mathrm{song}(\mathrm{a\_song\_for}(x)) \wedge \\ \mathrm{deserves}(x, \mathrm{a\_song\_for}(x)). \end{aligned} \tag{64}$$

Jos Skolem-funktion $\mathrm{a\_song\_for}(x)$ sijasta olisi käytetty vakiota, joka ei olisikaan riippunut muuttujan x arvosta, niin olisikin sanottu "jokainen ihminen on *saman* laulun arvoinen".

Skolemisointi muuttaa hieman kaavan alkuperäistä merkitystä, mutta "turvalliseen" suuntaan: alunperin yhdistelmälle $\vec{u}$ saattoi olla monia sopivia u_i, mutta f_i valitsee niistä yhden erikoisasemaan.

7. Nosta konjunktiot disjunktioiden yli uudelleenkirjoitussäännöllä

$$(\alpha \wedge \beta) \vee \gamma \text{ muotoon } (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$

sekä sen variantilla disjunktion vaihdannaisuuden suhteen.

Nämä säännöt säilyttävät selvästi kaavan semantiikan.

Esimerkiksi kaava (64) muuttuu muotoon

$$\begin{aligned} \forall x.(\neg\, \mathrm{person}(x) \vee \mathrm{song}(\mathrm{a\_song\_for}(x))) \\ \wedge \\ (\neg\, \mathrm{person}(x) \vee \mathrm{deserves}(x, \mathrm{a\_song\_for}(x))). \end{aligned} \tag{65}$$

8. Nosta konjunktiot kvanttoreiden yli uudelleenkirjoitussäännöillä

$$\forall x(\alpha \wedge \beta) \text{ muotoon } (\forall x \alpha) \wedge (\forall x \beta).$$

Nämä säännöt säilyttävät jälleen kaavan semantiikan.
Esimerkiksi kaava (65) muuttuu lopulta muotoon

$$(\forall x_1. \neg\, \mathrm{person}(x_1) \lor \mathrm{song}(\mathrm{a\_song\_for}(x_1))) \land$$
$$(\forall x_2. \neg\, \mathrm{person}(x_2) \lor \mathrm{deserves}(x_2, \mathrm{a\_song\_for}(x_2)))$$

eli implikaatioina

$$(\forall x_1.\, \mathrm{person}(x_1) \Rightarrow \mathrm{song}(\mathrm{a\_song\_for}(x_1))) \land$$
$$(\forall x_2.\, \mathrm{person}(x_2) \Rightarrow \mathrm{deserves}(x_2, \mathrm{a\_song\_for}(x_2))).$$

Päädyttiin siis Hornin lauseisiin "jokaiseen ihmiseen liittyy jokin laulu" ja "jokainen ihminen on tuon häneen liitetyn laulun arvoinen".

Lopuksi voidaan

- unohtaa tarpeettomat sulut
- jättää kvanttorit merkitsemättä
- esittää disjunktiot D_i literaalien joukkoina
- esittää koko konjunktio literaalijoukkojen joukkona

$$\{\{\neg\, \mathrm{person}(x_1), \mathrm{song}(\mathrm{a\_song\_for}(x_1))\}\ ,$$
$$\{\neg\, \mathrm{person}(x_2), \mathrm{deserves}(x_2, \mathrm{a\_song\_for}(x_2))\}\}$$

jolloin ei tarvitse huolehtia

- saman literaalin tai konjunktin toistumisesta
- literaalien tai konjunktien keskinäisestä järjestyksestä.

Normalisoinnin huonoja puolia:

- Ihmisen kirjoittaman tietämyksen ulkoasu muuttuu.
- Päättelyaskeleiden ymmärtäminen vaiheutuu ulkoasun muututtua.
- Normalisointi nojaa klassiseen logiikkaan, eikä aina yleisty muihin logiikoihin.

Toisaalta ilman normalisointia pelkkä resoluutiosääntö ei riitä.

Normalisoinnin jälkeen itse resoluutiosääntö on

$$\frac{P \vee L_1 \vee \ldots L_m \quad \neg P' \vee L_1' \vee \ldots L_n'}{\textsc{Subst}(\theta, L_1 \vee \ldots L_m \vee L_1' \vee \ldots L_n')} RES \tag{66}$$

missä

- $\theta = \textsc{Unify}(P, P')$
- merkitsemättä jätetyt universaalikvantifioinnit käsitellään kuten luvun 5.3.3 taaksepäin ketjutuksessa.

Toisin sanoen

1. kun valitaan yhdestä ehdosta jokin positiivinen literaali P (disjunktien järjestyshän oli vapaa)
2. ja toisesta ehdosta jokin negatiivinen literaali $\neg P'$
3. jotka samastuvat kuten algoritmissa 13
4. niin johtopäätös eli *resolventti (resolvent)* saadaan
   (a) pyyhkimällä pois valitut literaalit P ja $\neg P'$
   (b) yhdistämällä loput ehdoista toisiinsa
   (c) ja soveltamalla yleisintä samastinta θ.

Implikatiivisessa normaalimuodossa atomikaavat P ja P' taas valitaan *eri puolilta* ehtojen implikaationuolia.

Luvun 5.3.3 yleistetty Modus Ponens -sääntö (56) on resoluutiosarja

$$\begin{array}{c}\dfrac{p_2' \quad \dfrac{p_1' \quad \neg p_1 \vee \ldots \vee \neg p_n \vee q}{\textsc{Subst}(\theta_1, \neg p_2 \vee \ldots \vee \neg p_n \vee q)} RES}{\textsc{Subst}(\textsc{Compose}(\theta_1, \theta_2), \neg p_3 \vee \ldots \vee \neg p_n \vee q)} RES \\ \vdots \\ \textsc{Subst}(\theta, q)\end{array} \tag{67}$$

missä

- $\theta_1 = \textsc{Unify}(\textsc{Rename}(p'_1), p_1)$
- Syntaktinen operaatio $\textsc{Rename}(p'_1)$ tuottaa sellaisen kaavan, joka saadaan nimeämällä kaavan p'_1 vapaat muuttujat uudelleen sellaisiksi, joita ei käyteta missään muualla.

  Tarkkaan ottaen sääntökin pitäisi nimetä uudelleen, mutta se mutkistaisi merkintöjä entisestään.
- $\theta_2 = \textsc{Unify}(\textsc{Rename}(p'_2), \textsc{Subst}(\theta_1, p_2))$
- $\textsc{Subst}(\theta_2, \textsc{Subst}(\theta_1, \phi)) = \textsc{Subst}(\textsc{Compose}(\theta_1, \theta_2), \phi)$; tämähän on se ehto, joka määrittelee operaation $\textsc{Compose}$.
- $\theta = \textsc{Compose}(\textsc{Compose}(\textsc{Compose}(\theta_1, \theta_2), \theta_3), \ldots), \theta_n)$.

  Tämä on itse asiassa sama sijoitus kuin se, joka täytyy muodostaa sääntöä (56) sovellettaessa.

Esimerkkinä (implikatiivisen normaalimuodon) resoluutiosäännöllä löydetystä refutaatiosta olkoon seuraava ongelma:

- Jackillä on koira:

$$\exists x.\, \mathrm{Dog}(x) \wedge \mathrm{Owns}(\mathrm{Jack}, x) \text{ eli normalisoituna}$$

$$\mathrm{Dog}(\mathrm{D}) \text{ ja}$$

$$\mathrm{Owns}(\mathrm{Jack}, \mathrm{D}).$$

- Jokainen koiranomistaja on eläinrakas:

$$\forall x.(\exists y.\, \mathrm{Dog}(y) \wedge \mathrm{Owns}(x, y)) \Rightarrow \mathrm{AnimalLover}(x)$$

$$\mathrm{Dog}(y) \wedge \mathrm{Owns}(x, y)) \Rightarrow \mathrm{AnimalLover}(x).$$

- Kukaan eläinrakas ei tapa eläimiä:

$$\forall x.\, \mathrm{AnimalLover}(x) \Rightarrow \forall y.\, \mathrm{Animal}(y) \Rightarrow \neg\, \mathrm{Kills}(x, y)$$

$$\mathrm{AnimalLover}(x) \wedge \mathrm{Animal}(y) \wedge \mathrm{Kills}(x, y) \Rightarrow \mathbf{F}.$$

  Tämä ei ole Hornin lause, koska siinä ei ole ainuttakaan positiivista literaalia.

- Jack tai uteliaisuus tappoi Tuna-kissan:

$$\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna}).$$

Normalisointia ei nyt tarvittu. Tämäkään ei ole Hornin lause, koska siinä on enemmän kuin yksi positiivinen literaali.

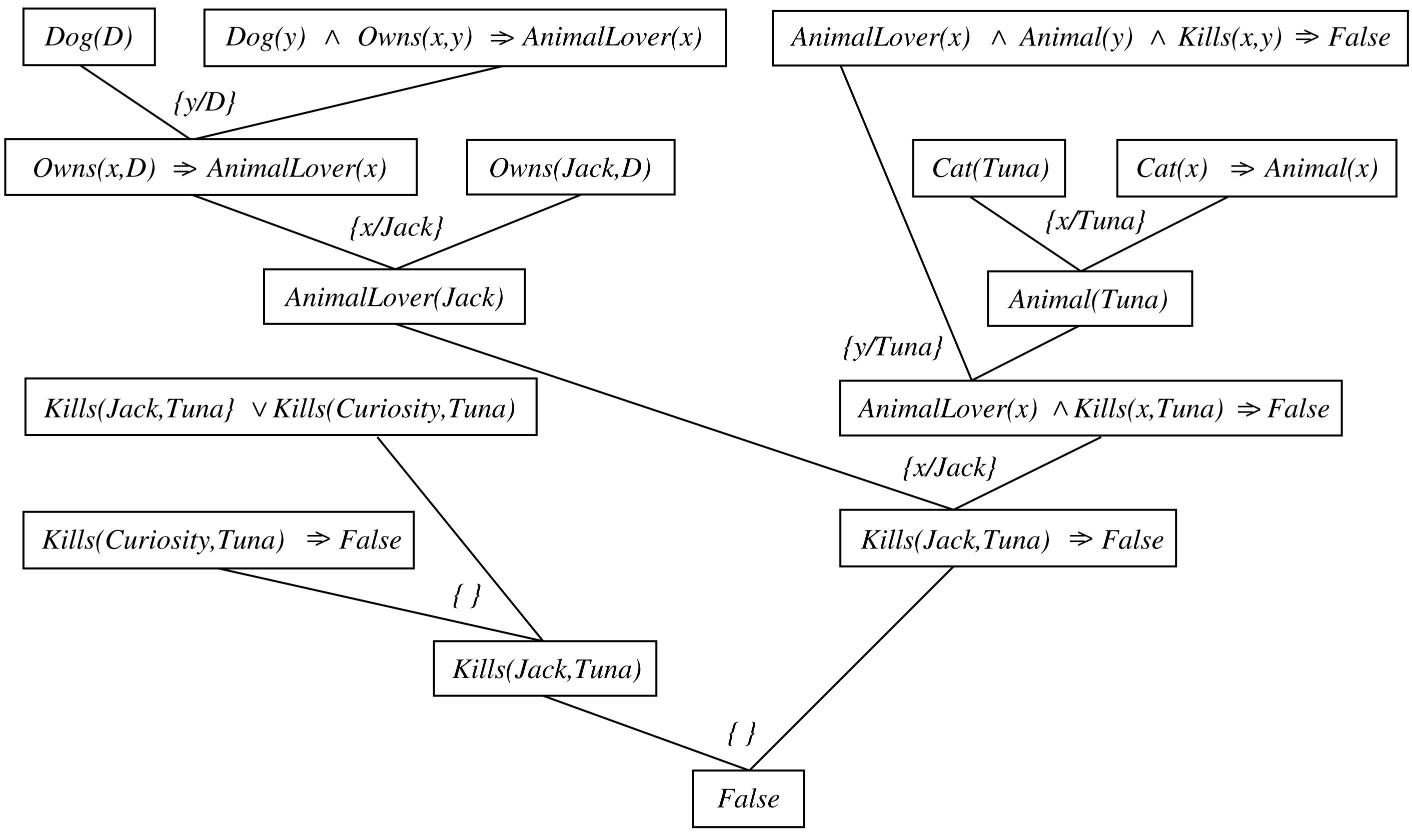
- Tuna-kissasta tarvitaan vielä:

$$\text{Cat}(\text{Tuna}) \text{ ja}$$
$$\forall x. \, \text{Cat}(x) \Rightarrow \text{Animal}(x) \text{ eli}$$
$$\text{Cat}(x) \Rightarrow \text{Animal}(x).$$

- Tappoiko uteliaisuus Tuna-kissan?
Vastaväite on "uteliaisuus ei tappanut Tuna-kissaa" eli

$$\text{Kills}(\text{Curiosity}, \text{Tuna}) \Rightarrow \mathbf{F}.$$

Näistä normalisoiduista kaavoista saadaan ristiriita seuraavilla resoluutio-askeleilla:



Kuva 61: Esimerkki resoluutiorefutaatiosta [RN95, kuva 9.7].

Etsittäessä ristiriitaa resoluutiolla mahdollisia alimaalin ja säännön valintoja on hyvin paljon, koska kyseessä ovat yleiset konjunktiivisen normaalimuodon eivätkä ainoastaan Hornin lauseet. Niinpä etsintään on kehitetty erilaisia *strategioita*:

Faktojen suosimisessa (unit preference) ainakin toinen resolvoitavista kaavoista valitaan faktaksi.

Faktojen käyttö pienentää sitä disjunktiota, joka halutaan tyhjentää, kun taas muiden sääntöjen käyttö kasvattaisi sitä.

Faktojen suosiminen ei yksinään vielä riitä etsinnän nopeuttamiseksi – eihän se edes lähde käyntiin jos tietämys on aluksi faktaton!

Päättely (67) oli faktoja suosiva.

Tukijoukko (set of support) on sellainen esitetyn tietämyksen kaavojen osajoukko, jossa ristiriidan oletetaan piilevän. Silloin on johdonmukaista valita ainakin toinen resolvoitava kaava tästä joukosta. Jokainen näin päätelty kaava lisätään tähän joukkoon.

Jos tukijoukko on valittu siten, että sen ulkopuolelle jäävät kaavat ovat keskenään ristiriidattomia, niin silloin strategia säilyttää resoluution täydellisyyden. (Siis jos ristiriita on olemassa, niin *on olemassa* myös jokin tukijoukkostrategian mukainen ristiriitatodistus; mutta tämän todistuksen *löytyminen* taas riippuu käytetyn etsintämenetelmän täydellisyydestä.)

Yleinen tapa onkin ottaa tukijoukoksi kumottavan johtopäätöksen negaatio(sta saadut kaavat) ja olettaa itse tietämyskanta ristiriidattomaksi. Näin löydetyt resoluutiotodistukset ovat "helppolukuisia" koska niissä näkyy tavoite vastata kyselyyn.

Syöteresoluutiossa (input resolution) valitaan aina toiseksi resolvoitavaksi jokin syötteenä saatu kaava.

Syöteresoluutio on täydellinen Hornin lauseille, mutta ei yleisille kaavoille.

Luvun 5.3.3 ketjutukset voitaisiin tehdä myös syöteresoluutiolla.

Lineaarinen (linear) resoluutio on syöteresoluution laajennus, jossa toiseksi resolvoitavaksi valitaan aina joko syötekaava tai jokin nykyisen kaavan todistuksen aikaisempi väivaihe.

Tämä lisäys riittää täydellisyyteen yleisillekin kaavoille.

Luvun 5.3.3 Prolog-taaksepäinketjutus on esimerkki sekä syöte- että lineaarisesta resoluutiosta.

Sisältyminen (subsumption) jättää huomiotta kaikki sellaiset kaavat, jotka seuraavat jostakin tietämyskannassa jo olevasta kaavasta. Tämä vähentää haarautumisasteen kasvua horjuttamatta täydellisyyttä.

Jos esimerkiksi fakta $\mathrm{p}(x)$ on jo tietämyskannassa, niin sinne ei kannata enää lisätä faktaa $\mathrm{p}(\mathrm{a})$ eikä kaavaa $\mathrm{p}(\mathrm{a}) \vee \mathrm{q}(\mathrm{b})$ koska ne seuraavat siitä.

Sisältyminen voidaan selvittää käyttämällä *sovitusta (matching)*, algoritmin 13 samastuksen "yksipuolista" muotoa [CL73, luku 5.8], jossa muuttujia esiintyy vain toisessa kaavoista. esimerkiksi fakta $\mathrm{p}(\mathrm{a})$ sopii hahmoon $\mathrm{p}(x)$ sijoituksella $\{x = \mathrm{a}\}$.

Viitteet

[Bod90] M.A. Boden, editor. *The Philosophy of Artificial Intelligence*. Oxford Readings in Philosophy. Oxford University Press, 1990.

[Car97] B. Carpenter. *Type-Logical Semantics*. The MIT Press, 1997.

[Cho97] N. Chomsky. B. F. Skinnerin teos *verbal behaviour*. In P. Raatikainen, editor, *Ajattelu, kieli, merkitys*, pages 279–310. Gaudeamus, 1997. Suom. R. Vilkko.

[CL73] C-L. Chang and R.C-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CSC90] P.M. Churchland and P. Smith Churchland. Could a machine think? *Scientific American*, pages 26–31, 1990.

[Dam99] A. Damasio. *The Feeling of What Happens*. William Heinemann, 1999.

[DD88] H.L. Dreyfus and S.E. Dreyfus. Making a mind versus modelling the brain: Artificial intelligence back at a branch-point. *Artificial Intelligence*, 1988.

[Dea98] T.W. Deacon. *The Symbolic Species: The Co-Evolution of Language and the Human Brain*. Penguin Books, 1998.

[MP43] W.S. McCulloch and W.H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943. [Bod90, §1].

[Myr74a] L. Myrberg. *Differentiaali- ja integraalilaskenta*, volume 1. Kirjayhtymä, 1974.

[Myr74b] L. Myrberg. *Differentiaali- ja integraalilaskenta*, volume 2. Kirjayhtymä, 1974.

[Nil98] N.J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan-Kaufmann, 1998.

[NKN01] M.K. Nowak, N.L. Komarova, and P. Niyogi. Evolution of universal grammar. *Science*, 291:114–118, 2001.

[Nv01] S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

[Pen89] R. Penrose. *The Emperor's New Mind*. Oxford University Press, 1989.

[Pin94] S. Pinker. *The Language Instinct*. William Morrow and Company, Inc., 1994.

[Pin97] S. Pinker. *How the Mind Works*. W. W. Norton & Company, Inc., 1997.

[Pou91] L.C. Poulson. *ML for the Working Programmer*. Cambridge University Press, 1991.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall, 1995.

[Saa85] E. Saarinen. *Länsimaisen filosofian historia huipulta huipulle*. Werner Söderström oy, 1985.

[Sea80] J. Searle. Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–424, 1980. [Bod90, §3].

[Sea90] J. Searle. Is the brain's mind a computer program? *Scientific American*, pages 20–25, 1990.

[TE88] T. Tononi and G.M. Edelman. Consciousness and complexity. *Science*, pages 1846–1851, 1988.

[Tur50] A.M. Turing. Computing machinery and intelligence. *Mind*, LIX(2236):433–460, 1950. [Bod90, §2].

[Vää87] J. Väänänen. *Matemaattinen logiikka*. Gaudeamus, 2nd edition, 1987.