1. Tekoälyn perusteista [RN95, osa I][Nil98, luku 1]

Tekoäly on *uusi* tieteenala: nimensä se sai 1956. Se syntyi vain noin 10 vuotta ensimmäisten yleiskäyttöisten tietokoneiden jälkeen, joten se on yksi vanhimpia alueita *tietojenkäsittelytieteen sisällä*.

Se on samalla *vanha* tieteenala: yli 2000 vuotta on pohdittu, miten havaitsemista/oppimista/muistamista/päättelyä/. . . tehdään/tulisi tehdä.

Toisin kuin vanhemmat sisarensa *filosofia* ja *psykologia*, tekoäly pyrkii *rakentamaan* älykkäitä olentoja/yksikköjä/koneita/. . . eikä (pelkästään) ymmärtämään itse älykkyyttä.

Tämän luvun eri asiat liittyvät monin eri tavoin toisiinsa, joten se pakostikin sisältää runsaasti ristiviittauksia.

1.1. Käsitteen "tekoäly" merkityksiä

Jotta tekoälystä voi "filosofoida", pitää ensin määritellä, mitä sanalla tarkoittaa.

Eri määritelmät johtavat erilaisiin tutkimustavoitteisiin ja -menetelmiin.

Eräs mahdollinen jako saadaan tekemällä yhtä aikaa kaksi erottelua: *ajatella/toimia* ja *inhimillisesti/rationaalisesti*. Näin syntyy nelikenttä:

"The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Taulukko 1: 4 käsitystä tekoälystä [RN95, kuva 1.1].

Eri lokeroiden väliset raja-aidat eivät ole tiukkoja:

- Kummankin toiminnan painottaja edellyttää (tietenkin) myös kykyä vastaavaan ajatteluun.
- Ajattelun motivaationa on kääntäen tarve myöhemmin toimia.
- Ainakin toivoisi, että inhimillisyys ja rationaalisuus olisivat edes kaukaista sukua toisilleen. . .

Lokeroinnissa onkin kysymys siitä, mitä pitää *itse tavoitteen saavuttamisena* ja mitä sitä edeltävänä/seuraavana lisätyönä.

1.1.1. "tekoäly" = "toimia inhimillisesti"

Taulukon 1 vasemman yläkulman tunnetuin muotoilu on *Alan Turingin (1912-1954)* vuonna 1950 esittämä "oikeussalidraama", jossa konetta syytetään "väärennetyillä henkilöllisyyspapereilla esiintymisestä":

syytetyistä $\mathbf{A}$ ja $\mathbf{B}$ toinen on ihminen ja toinen ihmistä teeskentelevä kone M.

ihmistuomarin $\mathbf{C}$ tehtävänä on päättää kumpi on kumpi. Hän esittää valitsemiaan kysymyksiä syytetyille ja vastausten perusteella julistaa tuomionsa.

"Oikeus on sokea": tuomarilla C ei ole muuta informaatiota (tuntoa, näköä, hajua,. . .) syytetyistä kuin kysymysten kirjalliset vastaukset.

Konetta on pidettävä älykkäänä, jos C erehtyy tuomiossaan.

Tai tilastollisesti: tuodaan sama kone "rikoksenuusijana" yhä uudelleen saman C eteen eri ihmisten kanssa, ja kutsutaan sitä älykkääksi, mikäli yhteenlaskettu virheiden määrä on samaa luokkaa kuin jos C olisi kuulustelujen sijaan heittänyt kruunaa ja klaavaa.

periaate: kone on "älykäs" jos sen *syöte-tuloste -käyttäytymistä* ei voida erottaa luotettavasti ihmisen vastaavasta.

Turing ottaa näin *funktionalistisen* kannan älykkyyteen. Myöhemmät tämän kannan edustajat ovat usein siirtyneet tarkastelemaan kokonaisuuden sijasta sen *osien* käyttäytymistä esimerkiksi väittämällä, että vain yksittäisten neuronien (hermosolujen) syöte-tulossignaalit merkitsevät, ei niiden välittämisessä käytetyt biokemialliset mekanismit.

Voisiko funktionalismin hylätä rajaamatta "älykkyyttä" samalla vain tietynlaisiin eliötyyppeihin (joita saattaa esiintyä vain maapallolla)?

1.1.2. "tekoäly" = "ajatella inhimillisesti"

Ihmisen mielen
miten kuvaisit? Se on
tushipiirroksen
mäntyjen oksistossa
tuulevan tuulen ääni.

Ikkyū (suom. Tuomas Anhava)

Rajusti kärjistäen taulukon 1 vasenta alakulmaa, *kognitiotieteen (Cognitive Science)* menetelmä älykkyyden mallintamiseen on seuraava:

1. Kerätään ihmisryhmältä I *kokeellisen psykologian* menetelmin havaintoaineistot H ja H' sen jäsenten käyttäytymisestä jossakin "älykkyyttä vaativassa" tehtävässä T.
2. Rakennetaan sellainen *laskennallinen malli (ohjelma)* M, joka antaa (likimain) samat tulokset kuin lähde- tai opetusaineisto H.
3. Testataan, antaako M myös (likimain) samat tulokset kuin testiaineisto H'.
4. Jos antaa, niin tämä tukee seuraavia hypoteeseja:
   (a) M ratkaisee tehtävän T "älykkäästi", käyttäytyyhän se samaan tapaan kuin joukon I jäsen; ja kääntäen
   (b) M:n laskennalliset operaatiot muistuttavat niitä mentaalisia operaatioita, joita joukon I jäsen käyttää ratkoessaan tehtävää T.

periaate: kone on "älykäs" jos siinä tapahtuu *samankaltaista informaationkäsittelyä* kuin ihmismielessä.

Luvun 1.1.1 Turingin testi -määritelmään verrattuna ei "onnistumisen" kriteeri ole enää yhtä selvä, mutta toisaalta nyt päästään vetoamaan mallin/koneen M *sisäiseen* rakenteeseen ja toimintaan.

1.1.3. ”tekoäly” = ”ajatella rationaalisesti”

> **logic** *n.* The art of thinking and reasoning in strict accordance with the limitations and incapacities of the human misunderstanding.
>
> *A. Bierce*

Filosofiassa ja matematiikassa on tutkittu paljon *logiikkaa*, ”oppia oikeasta ajattelusta”: sääntöjä, joita järjestelmällisesti seuraamalla ajatteluun ei pääse putkahtamaan virheitä.

Esimerkiksi *Aristoteleen (384–322 eKr.) syllogismit* kuten

Sokrates on ihminen. (1)

Kaikki ihmiset ovat kuolevaisia. (2)

Siispä Sokrateskin on kuolevainen. (3)

tuntuvat ”järkeviltä”: jos olen jo vakuuttunut väitteistä (1) ja (2), niin johdonmukaisuuden nimissä minun täytyy silloin hyväksyä myös johtopäätös (3). Minun täytyy jopa hyväksyä kaikki *saman muotoiset* perustelut, joissa 'Sokrates', 'ihminen' ja 'kuolevainen' korvataan muilla sanoilla.

periaate: kone M on ”älykäs” jos se maailman tilan ja säännönmukaisuuksien kuvauksesta pystyy *päättelemään* onko annettu tavoite saavutettavissa (ja jos, niin miten).

Toisin kuin lukujen 1.1.1 ja 1.1.2 määritelmissä, taulukon 1 oikeassa yläkulmassa pyritään ihmisen jäljittelyn sijasta ”oikeaan lopputulokseen”. Tämä kuitenkin määritellään koneen M *ulkopuolisin kriteerein* U, kun taas aiemmin tavoite ”olla kuin ihminen” oli ”sisäistetty”.

Kun tavoite on ulkoa annettu, voi ”tietämysinsinööri” *(knowledge engineer)* alkaa *suunnittelemaan ja rakentamaan* sen toteuttavaa konetta, jota sitten voisi väittää älykkääksi.

1.1.4. "tekoäly" = "toimia rationaalisesti"

Kun vaaditaan vielä luvun 1.1.3 koneelta M *toimintaa* muodostamiensa rationaalisten ajatusten mukaisesti, saadaan oppikirjojemme [Nil98, RN95] kanta, jonka mukaan tekoäly on *älykkäiden toimijoiden (intelligent agents)* suunnittelemista ja rakentamista.

Toimija (agent) on mikä tahansa erillinen yksikkö A, jonka voidaan ajatella tekevän *havaintoja (percept)* ympäröivän maailman *tilasta (state)* sekä *teoillaan (action)* vaikuttavan siihen.

Informaation kerääminen havainnoilla on sekin tekemistä!

Suorituskykymitta (performance measure) U on luvussa 1.1.3 mainittu toimijan A ulkopuolinen mittari sen (pitkän aikavälin) toiminnan tuloksellisuudelle. Ajatellaan se (numeerisena) funktiona *ympäröivän maailman eri tiloille* S: siivousrobotilla $U(S) < U(S')$, jos S' on puhtaampi maailma kuin S.

Esimerkillinen rationaalinen toimija (ideal rational agent) A_{ert} valitsee jokaisessa tilassa S_{nyt} sellaisen mahdollisen teon T_{nyt}, joka maksimoi suorituskykymitan U *odotusarvon (expected value)*.

Odotusarvon, koska (esimerkiksi autoilevan) toimijan A_{ert}

- lisäksi voi olla *muita* toimijoita
  "Tuleekohan tuolta sivutieltä ketään eteeni?"
- *tieto* nykytilasta S_{nyt} tai tekojen T_{nyt} vaikutuksista voi olla epävarmaa
  "Onkohan tie liukas?"
  "Kiihtyyköhän autoni tarpeeksi ohitukseen?"
- tekojen T_{nyt} *onnistuminen* voi olla epävarmaa
  "Entäs jos pyörät sutivatkin tyhjää kun kaasutan?"

Liikennesääntö "risteysmerkin jälkeen ei saa enää ohittaa" voidaankin nähdä laskutoimituksen *yksinkertaistuksena* karsimalla vaihtoehtoja.

Valinnan T_{nyt} pohjana A_{ert} käyttää

- sisäänrakennettua tietoa
  Liikennesäännöt, . . .
- ennen nykytilaa S_{nyt} keräämäänsä *havaintojonoa (percept sequence)* H_{nyt}.
  Onko auto liukastellut tänään aiemmin? Miten auto on ennen kiihtynyt? . . .

Jonon H_{nyt} perusteella A_{ert} muodostaa oman käsityksensä siitä, millainen S_{nyt} on.

A_{ert} on *itseohjautuva (autonomous)* siltä osin, kuin sen käyttäytymisen on sen aiempien kokemusten H_{nyt} määräämää.

Kellokin on esimerkillinen rationaalinen toimija – jos tavoitteena on vain pyörittää viisareita tahdissa antamatta minkään häiritä niiden kulkua. Se ei havainnoi ympäristöään eikä siis ole lainkaan itseohjautuva, vaan sen "rationaalisuus" on kokonaan kellosepän siihen istuttamaa.

A_{ert} ei saa ummistaa silmiään *olennaiselta* informaatiolta: ennen ohitusta kannattaa katsoa taustapeiliin, ettei käänny takaa tulevan ohittajan eteen – maantieajossa ohituskolarin U-arvo ei liene järin suuri. . .

. . . mutta vuoden 1997 viimeisessä Formula 1 -osakilpailussa Schumacher *toimi järkevästi* kiilatessan Villeneuvea vastaavassa tilanteessa:

Villeneuve pisteittä: Schumacher on mestari.

Villeneuve ohittaa: Schumacher ei ole mestari.

- $U_1 = U_{\text{Schumi}}(\text{Schumacher päästää Villeneuven ohitseen}) \ll 0.$
- $U_2 = U_{\text{Schumi}}(\text{Schumacher kiilaa Villeneuven(kin) ulos}) \gg 0.$
- $U_3 = U_{\text{Schumi}}(\text{Schumacher kiilaa Villeneuven taakseen}) > 0.$

- $U_4 = U_{\text{Schumi}}(\text{Schumacher kiilaa vain itsensä ulos}) \ll 0$.
- Todennäköisyys $P_1 = P(\text{Schumacher kiilaa Villeneuven ulos})$ on hyvin suuri, onhan hän asiansa osaava kuski. . .
- Samoin $P_2 = P(\text{Schumacher kiilaa itsensä ulos})$ on hyvin pieni.

Silloin kiilaamisen odotusarvo on

$$P_1 \cdot U_2 + (1 - P_1) \cdot (1 - P_2) \cdot U_3 + \underbrace{(1 - P_1) \cdot P_2}_{\text{hyvin, hyvin pieni}} \cdot U_4$$

mikä lienee (paljon) suurempi kuin U_1.

Sillä kertaa kuitenkin tuo pessimaalinen vaihtoehto toteutui.

Vaikka jokaisella A_{ert} onkin *oma yksityinen* U, voi kokonaisuus silti maksimoida *yhteistä hyvää*: valitaan yhteiset (liikenne)säännöt siten, että yhteinen hyvä on kannattavin vaihtoehto jokaiselle. Tätä on esim. talous- ja yhteiskuntatieteissä käytetty ns. *mekanismisuunnittelu (mechanism design)*.

A_{ert} ei toisaalta saa jäädä odottelemaan *epäolennaista* informaatiota: jos taustapeilistä näkyy toinen auto, mutta se ei ole vielä laittanut ohitusvilkkua päälle, ei jäädä odottelemaan sitä, vaan ohitetaan itse.

Tekoälyn tehtäväksi nähdään nyt *tietokoneen ohjelmoiminen laskemaan funktio* $A_{\text{ert}}(H_{\text{nyt}}) = T_{\text{nyt}}$.

```
function SKELETON-AGENT(percept) returns action
  static: memory, the agent's memory of the world

  memory ← UPDATE-MEMORY(memory, percept)
  action ← CHOOSE-BEST-ACTION(memory)
  memory ← UPDATE-MEMORY(memory, action)
  return action
```

Algoritmi 1: Älykkään toimijan perusohjelma [RN95, kuva 2.4].

Mikään ei kuitenkaan takaa, että tämä funktio (aliohjelma CHOOSE-BEST-ACTION) olisi mekaanisesti laskettavissa. Muutetaan siis tavoite *täydellisestä (perfect) laskennalliseksi (calculative)* rationaalisuudeksi, ja edelleen käytännössä *rajoitetuksi optimaalisuudeksi (bounded optimality, BO)*, jossa tätä funktiota approksimoidaan mahdollisimman hyvin ylittämättä ennalta annettuja laskentaresursseja. Toimija on siis BO-mielessä älykäs, jos se käyttää sille suodun harkinta-ajan mahdollisimman tehokkaasti, keskittyy olennaiseen.

1.2. Tekoäly ja matematiikka

Logiikka alkoi jo antiikista (luku 1.1.3), mutta vasta 1800-luvun loppupuolelta alkaen se laajeni filosofiasta (meta)matematiikkaan. Haluttiin tarkka muotoilu *todistuksen käsitteelle*: jono väitteitä. . .

1. jotka on esitetty niin täsmällisessä kielessä, ettei yhtäkään niistä voi tulkita kuin yhdellä tavalla; ja
2. jossa jonossa seuraava väitteen voidaan aina nähdä olevan välttämätön edellisten nojalla – kuten (3) väitteistä (1) ja (2).

Tämä oli keskeinen osa *David Hilbertin (1862–1943)* ohjelmaa saattaa matematiikka järkkymättömälle perustalle. Vuonna 1900 hän listasi 23 matematiikan keskeistä avointa ongelmaa, joista viimeinen oli *Entscheidungsproblem*: voidaanko *luonnollisia lukuja koskevien loogisten väitteiden totuus* ratkaista mekaanisesti?

George Boole (1815–64) oli jo 1847 yrittänyt "matematisoida logiikan" antamalla totuusarvoille laskulait lukuina 0 ja 1. Tästä syntyi myöhemmin *lauselogiikka*.

Gottlob Frege (1848–1925) kehitti 1879 *ensimmäisen kertaluvun predikaattilogiikan (First Order Predicate Calculus)*. Sen pohjalta hän yritti 1893–1903 "logisoida matematiikan". *Bertrand Russell (1872–1970)* osoitti kuitenkin 1902 menetelmässä virheen *paradoksillaan*: jos määritellään $P = \{Q : Q \notin Q\}$, niin sekä $P \in P$ että $P \notin P$ ovat mahdottomia.

Russell ja *Alfred N. Whitehead (1861–1947)* korjasivat ongelman *tyyppiteorialla*: P ei olekaan joukko, vaan "luokka". Heidän teossarjansa *Principia Mathematica (1910–13)* lienee kuuluisin yritys saavuttaa tavoite 1.

Tämän osuuden täydensi *Alfred Tarski (1902–83)* vuonna 1933

antamalla yleiset periaatteet, joilla tämän kaltaisten kielten totuus voidaan määritellä.

Tavoite 2 johti puolestaan **laskettavuuden** käsitteeseen: jo *Gottfried W. von Leibniz (1646–1716)* oli haaveillut kielestä, jossa kiistat voitaisiin ratkaista laskemalla. Piti siis määritellä intuitiivinen "laskentasäännön" tai *algoritmin* idea (joka tunnettiin jo antiikissa, ja on nimetty arabimatemaatikko *al-Khwārizmī*n mukaan, jonka *Kitāb al-jabr wa al-muqābalah* vuodelta 825 toi länsimaihin algebran). Tavoite siis oli, että jokainen todistuksen askel voitaisiin *tarkistaa* mekaanisesti.

Ensiksi vuonna 1930 *Kurt Gödel (1906–78)* edisti Hilbertin ohjelmaa huomattavasti *predikaattilogiikan täydellisyyslauseellaan*, jonka mukaan tälle kielelle oli todellakin olemassa

- todistusmenetelmä jolla voi todistaa jokaisen totuuden
- eli mekaaninen todistuksen tarkistaja K.

Tästä logiikasta puuttuu kuitenkin *induktioperiaate*, joten entscheidungsproblem jäi edelleen auki.

Gödel ratkaisi ensin tavoitteen 2 abstraktissa matemaattisessa mielessä *rekursiivisten funktioiden teoriana*. Vuonna 1936 luvussa 1.1.1 mainittu Alan Turing esitti Gödelin työn kanssa yhtäpitävän laskettavuuskäsityksen, joka kuitenkin perustui (periaatteessa) rakennettavissa oleviin mekaanisiin laitteisiin. Tämä *Turingin kone* tuli sittemmin johtavaksi tietokoneen teoreettiseksi muotoiluksi.

Sitten vuonna 1931 Gödel osoitti koko ohjelman mahdottomaksi *epätäydellisyyslauseellaan*, jonka mukaan jos kielessä L voidaan esittää luonnollisten lukujen aritmetiikka, niin siinä voidaan myös lausua tosia väitteitä, joita ei kuitenkaan voida todistaa. Näin Gödel vastasi Hilbertin 23. ongelmaan "ei".

Gödelin päättelyn idea oli, että kielellä L voidaan silloin esittää myös lainauksia sekä algoritmeja (käyttäen eräänlaista salakirjoitusta, nk.

"Gödel-numerointia"). Olkoon sitten C mikä tahansa algoritmi kielen L todistusten tarkistamiseen. Silloin kielellä L voidaan sanoa *valehtelijan paradoksi* "minä olen väite, jonka todistusta C ei pysty tarkistamaan". Turing käytti myöhemmin samaa periaatetta osoittaessaan koneidensa *pysähtymisongelman* ratkeamattomaksi.

Yritys perustaa matematiikka formaaliin logiikkaan siis kariutui. Tekoäly kuitenkin otti sen välineet käyttöönsä:

- Tietämys maailman tilasta pyritään esittämään *kuvailevasti* sitoutumatta ennalta mihinkään tiettyyn käyttöön.
  Matemaattista aksioomaa saa käyttää todistuksessa sen oletuksista johtopäätökseen tai sen johtopäätöksen vastakohdasta sen oletusten vastakohtaan.
- Tietämyksen käyttö on erillisen *päättelyjärjestelmän* reviiriä.
  Ratkaisu aksiooman käyttösuunnasta tehdään vasta todistushetkellä, ei vielä sitä kirjoitettaessa.
- Päättelyjärjestelmä käsittelee *merkkejä, ei merkityksiä*.
  Muutenhan kuulija voisi torjua todistuksen sopimattomana hänen omaan merkitysteoriaansa.

"Älykäs päättely" on silloin käsitteellinen ristiriita. . .

1960-luvun puolivälistä alkaen on teoriassa tietojenkäsittelyssä keskitytty laskettavuuden teorian sijasta **laskennan vaativuusteoriaan**, joka pyrkii arvioimaan laskulaitteiden ajan- ja muistintarpeen kasvua syötteen koon kasvaessa. Perusajatus on, että *polynominen* kasvu voidaan vielä sietää, *eksponentiaalista* ei: koneen kasvattamisen pitäisi aina näkyä myös sen menestyksellä suorittamien laskutehtävien koon selvänä kasvuna.

Vaativuusteorian tärkein uusi käsite on *NP-täydellisyys*. Siinä missä Gödel kysyi 1930 voidaanko *predikaattilogiikkaa ollenkaan* automatisoida, *Stephen Cook* ja *Richard Karp* kysyivät 1971–72 *kuinka tehokkaasti lauselogiikka* voitaisiin automatisoida. Lauselogiikassa nimittäin jokaiselle todelle väitteelle *on olemassa* jokin lyhyt perustely, mutta sen

löytäminen nopeasti vaikuttaa varsin vaikealta. Laskentaongelmaa sanotaan NP-kovaksi, jos sen nopeasta ratkaisusta saataisiin jalostettua nopea ratkaisu tälle niin sanotulle SATISFIABILITY- eli SAT-ongelmalle, ja NP-täydelliseksi jos sama pätee myös toisinkin päin. (Sen *todistamista*, että nämä ongelmat välttämättä vaativat eksponentiaalisen ajan, pidetään teoreettisen tietojenkäsittelytieteen ja joskus jopa koko soveltavan matematiikan tärkeimpänä avoimena ongelmana.)

Neljäs matematiikan lahja tekoälylle on **todennäköisyyslaskenta**. Luvun 1.1.4 määritelmähän maksimoi *oletusarvoa sille hyödylle*, jonka toimija A_{ert} voi teollaan T_{nyt} saavuttaa, kun U sen "arvomaailmaksi". Tämä on puolestaan *päätösteoriaa (decision theory)* [RN95, luvut 16.1 ja 16.3], jonka *John von Neumann (1903–57)* ja *Oskar Morgenstern (1902–77)* perustivat 1944 ratkaisemaan, miten tehdä päätöksiä sattumanvaraisessa maailmassa.

Tarkastellaan esimerkiksi arpajaisia, jossa on jaossa yhteensä n arpaa á c \$, joista yksi voittaa y \$ ja muut jäävät ilman. Ostaisiko luvun 1.1.4 A_{ert} arvan, kun hän alussa omistaa x \$?

Toimijan A_{ert} *omaisuuden* odotusarvo olisi päätöksestä riippuen joko

$$\begin{aligned} E_{\text{ostaa}} &= \frac{1}{n}(x - c + y) + \frac{n-1}{n}(x - c) \\ &= x + \left(\frac{y}{n} - c\right) \text{ \$ tai} \\ E_{\text{ei osta}} &= x \text{ \$.} \end{aligned}$$

Arpajaisten järjestäjä on varmaankin valinnut $y < cn$, muutenhan niistä jäätäisiin tappiolle – joskin arvat kyllä kävisivät kaupaksi. Siis

$$E_{\text{ostaa}} < E_{\text{ei osta}}$$

joten A_{ert} ei milloinkaan sortuisi ostamaan arpaa.

Kuitenkin arvat käyvät kaupaksi; ovatko niiden ostajat siis (aina) *epä*rationaalisia toimijoita?

Eivät välttämättä: A_{ert} saattaa (perustellusti) olla sitä mieltä, että vaikka c \$ tappio *hiukan* harmittaisikin, niin voittaa y \$ tuntuisi silti *paljon* paremmalta kuin turvallinen nykytilanne x \$. Eli

$$U(x-c) \underset{\text{hiukan}}{<} U(x) \underset{\text{paljon}}{<} U(x-c+y).$$

Toimijan A_{ert} *onnellisuuden* odotusarvo onkin nyt

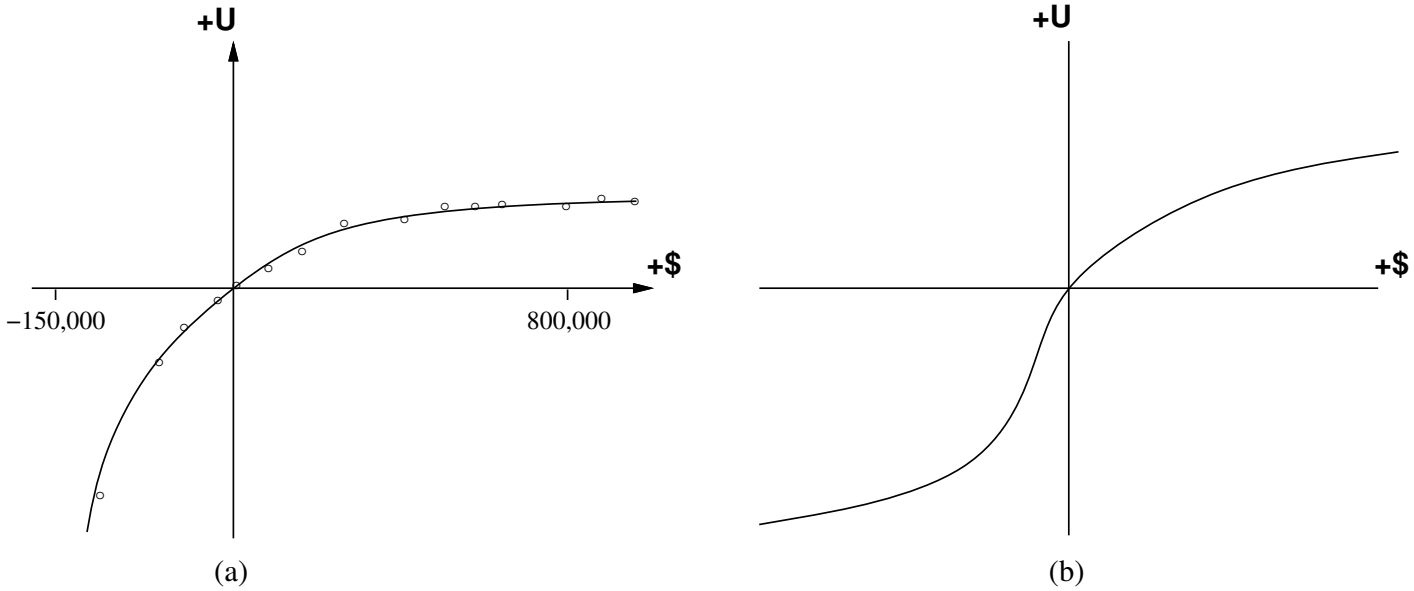
$$\begin{aligned} EU_{\text{ostaa}} &= \frac{1}{n}U(x-c+y) + \frac{n-1}{n}U(x-c) \text{ ja} \\ EU_{\text{ei osta}} &= U(x), \text{ jolloin saattaakin olla} \\ EU_{\text{ostaa}} &> EU_{\text{ei osta}}, \end{aligned}$$

eli A_{ert} saattaisikin ostaa arvan. Tai jos A_{ert} pitää uhkapelistä, niin U sisältää termin ei vain rahalle vaan myös itse osallistumiselle.

Kuvassa 1 (a) on haastattelulla selvitetty erään herra Beardin rahaa koskeva hyötymitta

$$U(x+z) = -263,31 + 22,09 \log(z + 150.000 \text{ \$}),$$

missä $-150.000 \text{ \$} < z < +800.000 \text{ \$}$ on mahdollinen tappio tai voitto. (Onko tässä luvun 1.1.2 tilanne: "älykäs arpajaiskäyttäytyminen" tarkoittaa "toimia kuten keskiverto ihiminen"?)



Kuva 1: Herra Beardin hyötykäsite [RN95, kuva 16.1.].

Herra Beard *välttää riskejä*, koska käyrä on *ylöspäin kupera* [Myr74a, luku 5.9]. Useimmat ihmiset lienevät samanlaisia.

Tällaisen käyrän omistava A_{ert} ei arpaamme ostaisi:

$$\frac{1}{n}U(x-c+y)+\frac{n-1}{n}U(x-c)<U(x) \tag{4}$$

järjestyy uudelleen muotoon

$$U(x-c+y)-U(x)<(n-1)(U(x)-U(x-c)),$$

ja *differentiaalilaskennan väliarvolauseen* [Myr74a, lause 5.4.1] mukaan löytyy $x-c<x_-<x<x_+<x-c+y$, joilla

$$\begin{aligned}(y-c)U'(x_+)&<(n-1)cU'(x_-),\ \text{eli}\\ c(U'(x_-)-U'(x_+))&<cnU'(x_-)-yU'(x_+).\end{aligned} \tag{5}$$

Koska $y<cn$, seuraa epäyhtälö (5) epäyhtälöstä

$$c(U'(x_-)-U'(x_+))<y(U'(x_-)-U'(x_+)),$$

joka puolestaan seuraa epäyhtälöstä

$$c < y \tag{6}$$

funktion U kuperuuden vuoksi.

Epäyhtälö (6) on puolestaan järkevä: miksi A_{ert} ehdoin tahdoin tuhlaisi rahaa ostamalla arvan, joka on kalliimpi kuin sillä mahdollisesti saatava voitto? Tai matemaattisesti, jos U on kasvava, niin epäyhtälö (4) pätee myös silloin kun epäyhtälö (6) ei päde.

Vastaavasti voidaan laskea, että tällainen A_{ert} uskoo sananlaskun "parempi pyy pivossa kuin kymmenen oksalla": jos A_{ert} saa valita, ottaako varmat v \$ tai osallistuuko arvontaan, jossa oletusarvoisesti voittaa *enemmän kuin* v \$, niin A_{ert} valitsee varmat v \$.

Samoin tällainen A_{ert} ottaisi *vakuutuksen*, vaikka se maksaakin enemmän kuin pelätyn vahingon oletusarvo. Tähän perustuu vakuutustoiminta: ihminen liittää kotinsa palamiseen enemmän mielipahaa kuin sen rahallinen arvo ja palamistodennäköisyys edellyttäisivät, ja on valmis maksamaan turvallisuudentunteestaan ylimääräistä. Tämä erotus on vakuutusyhtiön voittomarginaali.

Kuvassa 1 (b) on käyrää laajennettu negatiiviselle puolelle olettamalla, että jossakin vaiheessa tulee *käännepiste*, jonka jälkeen lisätappiot eivät enää paljoa lisää toimijan A_{ert} surua. Eli jos A_{ert} havaitsee, että häntä joka tapauksessa uhkaa $1,000,000$ \$ tappio ellei hän tee mitään, hän saattaa päätyä uhkarohkeaan yritykseen, joka onnistuessaan estää tappion mutta epäonnistuessaan tuplaa sen. Eikä tämä ole tyhmää tai lyhytnäköistä käyttäytymistä: jos olet uppoavan Titanicin kannella, niin *todennäköisesti* tulet kuolemaan, mutta silti sinun kannattaa yrittää *mitä tahansa* joka voisi pelastaa henkesi.

Nämä laskelmat kuitenkin edellyttävät, että A_{ert} pystyy päättelemään, mitkä teot T_{nyt} ovat tällä hetkellä mahdollisia ja miten maailma muuttuu niiden seurauksena. Toisin kuin loogisissa ideaalimaailmoissa, todellisessa

maailmassa tämä tietämys ei enää olekaan täsmällistä vaan ainoastaan kerätyista havainnoista H_{nyt} nähtäviä säännönmukaisuuksia. Näin päädytään *Bayesilaiseen tilastolliseen päättelyyn*, jonka periaatteet julkaistiin sen kehittäjän *Thomas Bayesin (1702-61)* kuoltua 1763. Tämä teoria selittää, miten A_{ert} voi päivittää omien maailmaa koskevien *uskomustensa voimakkuusasteita (degree of belief)* uuden tiedon karttuessa.

Esimerkki tällaisesta päättelytavasta on luku 1.1.2: hypoteesit (4a) ja (4b) vahvensivat toinen toistaan, vaikkei kumpaakaan todistettu aukottomasti loogisessa tai matemaattisessa mielessä.

Keskeinen tutkimusongelma onkin:

Miten voitaisiin parhaiten yhdistää logiikka ja Bayesin teoria?

Ratkaisua odotellessa tiivistetään toimijalähtöinen lähestymistapamme tekoälyyn seuraavasti:

- Toimijan A_{ert} perimmäiset tavoitteet on annettu ulkoa suorituskykymittana U, jonka oletusarvoa sen tulisi maksimoida teoillaan T_{nyt} päätösteorian mielessä.
  A_{ert} saattaa "tietää" oman mittansa U ja/tai rakentaa itselleen välitavoitteita johon pyrkiä.
  Mitta U vastaa, kun A_{ert} kysyy itseltään "Mitä minä oikeastaan *tahtoisin* saavuttaa?"
- A_{ert} muodostaa "maailmankuvansa" saamiensa havaintojen H_{nyt} pohjalta. Se kertoo, mitä eri vaihtoehtoiset teot T_{nyt} ovat aiemmin saaneet aikaan, ja millä todennäköisyyksillä. Tämä informaatio tarvitaan mitan U oletusarvojen laskemiseen ja vertailuun.
  Tämä kuva koostuu tilastollisesti havaituista säännönmukaisuuksista, joiden "uskottavuutta" päivitetään Bayesilaisittain havaintoaineiston karttuessa.
  Maailmankuvalla saattaa olla eksplisiittinen esitys, mutta ei välttämättä.
  Maailmankuva vastaa kysymykseen "Mitä minä *voisin* nyt saavuttaa?"

- A_{ert} joutuu päättelemään nykyisen maailmankuvansa perusteella (esimerkiksi) mitkä vaihtoehdot T_{nyt} ovat tällä hetkellä mahdollisia, mitkä aiemmin asetetut välitavoitteet ovat vielä kesken. . . Tähän yhden tavan antaa formaali logiikka, jossa kuitenkin (yleensä) unohdetaan todennäköisyydet ja uskomusten asteet.

  Tämä komponentti tarjoaa *muistin*, jonka sisällön käsittelytapoja ei välttämättä tarvitse kiinnittää etukäteen suunnittelupöydällä.

  Tämä kurssi keskittyy tämän komponentin menetelmiin:
  - symboliseen ja loogiseen tietämyksen esittämiseen
  - etsintään ja päättelyyn tästä tietämyksestä.

  Muille kursseille jäävät esimerkiksi
  - oppiminen havainnoista (koneoppinen)
  - tilastollinen päättely (kolme käsitettä -kurssit)
  - toiminta päättelyn pohjalta (robotiikka).

1.3. Tekoäly ja filosofia [RN95, luku 1.2]

Sana "teko*äly*" viittaa filosofiassa pohdittuihin ongelmiin. Tehdään siis lyhyt katsaus niihin ajatusrakennelmiin, jotka ovat edesauttaneet alan syntyä.

Ajattelua ja päättelyä on tutkittu jo antiikissa, esimerkiksi Aristoteleen syllogismeja (luku 1.1.3). Tällöin syntyi (?) ajatus siitä, että sääntöjoukko voi ohjata ihmisen mieltä – ainakin osittain; Aristoteleen ajattelussa oli mukana myös intuitiivinen äly ja tekijän sanaton tieto.

Mikä siis on oman mielemme ja meitä ympäröivän fysikaalisen maailman välinen suhde? Päädytään *ontologiaan*, filosofian alueeseen joka kysyy "mitä *pohjimmiltaan* on olemassa?"

- Jos sekä mieli että maailma ovat kumpikin olemassa erillään, päädytään *dualismiin*, jonka tunnetuin edustaja lienee ollut *René Descartes (1596–1650)*, latinaksi *Cartesius*.

  Mutta miten mieli voi silloin vaikuttaa fyysiseen maailmaan – ohjata sitä syy-seuraus -suhteilla?
- Jos mielikin nähdään fyysisen maailman tuloksena, päädytään *materialismiin*, jonka varhainen muotoilija oli Leibniz (luku 1.2).

  Mutta miten silloin voidaan selittää se, että ihmisellä näyttää olevan vapaa tahto, jolla hän voi valita mitä tehdä, eikä vain mennä luonnonlakien mukaan kuin putoava kivi?

  Usein kuulee väitettävän, että nykyinen (luonnon)tieteellinen maailmankuva olisi materialistinen. Näin pitkälle ei voine yleistää; lähinnä uskotaan, että *ainakin* fysikaalinen maailma on varmasti olemassa. Se, voidaanko siitä saada (varmaa) *tietoa*, onkin jo vaikeampi kysymys. . .

  Tekoälytutkimuksen yleisin lähtökohta lieneekin suunnilleen, että *mieli on informaation käsittelyä, aivot taas eräs fysikaalinen laite jolla tätä käsittelyä voidaan suorittaa.*

- Näiden kahden ääripään väliin sijoittuu monen tekoälyä arvostelevan filosofin omaksuva kanta, jonka mukaan vaikka mielellä onkin fysikaalinen perusta ("Brains Cause Minds", sanoo J. Searle), *ei se silti ole selitettävissä* pelkästään palauttamalla sen operaatiot fysikkaan.

  Voi esimerkiksi hyväksyä, että yksittäiset hermosolut toimivat fysiikan lakien mukaan, mutta väittää silti, ettei niiden *yhteistoiminnan tulosta* voi enää ymmärtää fysiikan lakien kautta. (Fysiikassakin ajatellaan yksittäisten hiukkasten noudattavan deterministisiä laskulakeja, mutta suurina määrinä siirrytäänkin puhumaan tilastollisesti entropiasta.)
- Filosofiassa on esitetty myös *idealistisia* kantoja, kuten *George Berkeleyn (1685–1753)* kanta, että on vain ääretön henki (= Jumala) ja äärellisiä henkiä (= ainakin ihmismielet, entä muut?), ja "fysikaalinen maailma" onkin pelkkiä ideoita, joita näillä hengillä on [Saa85, sivut 195–208].

  Helposti ajatellaan, että nämä kannat kuuluvat "selvästi arkikokemuksen vastaisina" jo menneisyyteen. Mutta itse asiassa niiden vaikutus näkyy edelleen: miten mieli saa kaipaamaansa informaatiota ulkomaailmasta, ellei tällaisina "ideoina" – ja koskeeko saatu informaatio sitten ulkomaailmaa itseään vai mielemme ideoita siitä?

Näin päädytään *tieto-oppiin (epistemologiaan)*, joka kysyy "Miten saamme informaatiota/tietoa? Onko (periaatteessakaan) mahdollista saada varmaa tietoa? Mikä on *synnynnäisten ideoiden* osuus ymmärryksessämme, vai onko mieli syntyessään kuin tyhjä taulu (Tabula Rasa) johon kokemus kirjoittaa paitsi yksittäiset havainnot myös niiden säännönmukaisuudet?"

Miten esimerkiksi saisimme tietää että "kaikki korpit ovat mustia"? Kärjistäen

Empiristi: "Kun näemme korpin toisensa jälkeen, ja ne kaikki ovat mustia, tämä sääntö alkaa tuntua yhä vahvemmin ja vahvemmin todelta."

(Tämä argumentti on sukua Bayesin teorialle luvusta 1.2. Varhaisia empiristejä olivat Berkeleyn lisäksi *Francis Bacon (1561–1626)*, *John*

Locke (1632–1704) – Tabula Rasan ehdottaja – ja *David Hume (1711–76)*.)

Rationalisti: "Mutta näin et saa *varmaa* tietoa! Voihan toisella puolen maapalloa olla valkea korppi, jota et koskaan tule nähneeksi. Tai voi olla *periaattessa* mahdollista, että sellainen syntyisi.

Ja sitäpaitsi sinulla pitää olla jo jokin *korppiuden käsite* valmiina, jotta tietäisit mitä katsoa.

Parempi on todistaa tämä sääntö loogisesti tuosta käsitteestä."

Empiristi: "Ja mitenhän sinä tuon käsitteesi mieleesi sait? Et ainakaan oppimalla, siinähän joutuisi tarkkailemaan lintuja! Vai oletko muka syntynyt kaikki mahdolliset käsitteet valmiina päässäsi?"

⋮

Immanuel Kant (1724–1804) alkoi sovitella näitä äärikantoja yhteen esittämällä, että ihmisellä on synnynnäisiä *kategorioita* – kuten syy-seuraus -suhteen käsite – joiden avulla havaintoja maailmasta voidaan alkaa jäsentämään [Saa85, sivut 227–54]. Nykykielellä [Pin97] sama ajatus kuuluisi "evoluutio on ohjannut ihmisaivoille sellaisen rakenteen, että ne käsittelevät havaintojen informaatiota sellaisilla tietyillä tavoilla, jotka synnyttävät mielen".

Tekoälyn toimijamääritelmässämme (luku 1.1.4) tämä ajatus näkyy siinä, että me suunnittelemme toimijamme perusrakenteen – eli ohjelmoimme sen, eli asetamme sen kategoriat – ennen kuin päästämme sen itseohjautuvasti maailmaan oppimaan havainnoistaan. Tekoäly ei siis ole *puhtaasti* empirismiä. . .

. . . mutta lähes: tämän vuosisadan alkupuolen metamatematiikan (luku 1.2) vastine filosofiassa oli *looginen positivismi*, joka myöhemmin lieveni *loogiseksi empirismiksi* ja loi pohjaa tämän vuosisadan anglosaksiselle analyyttiselle filosofialle. (Esimerkiksi Tarski ja Gödel tekivät sekä metamatematiikkaa että filosofiaa.) Tällä koulukunnalla oli 2 pääteesiä:

Todennettavuusteesi: Annetun väitelauseen *merkitys* (myöhemmin:

merkityksellisyyden/ymmärtämisen ehto) = sen *todentamisen menetelmä*.

Käännettävyysteesi: Jokainen merkityksellinen väitelause voidaan *kääntää* sellaiselle "puhtaalle" kielelle, jossa puhutaan ainoastaan havainnoista.

Turingin testi -määritelmä tekoälylle (luku 1.1.1) on tällainen: väitteen "kone M on älykäs" merkitykseksi annetaan koejärjestely, jossa tuomari C havainnoi syytettyjä M ja ihminen. Väite "tämä kone on älykäs" kääntyy puolestaan muotoon "tämä kone selviäisi tuosta koejärjestelystä".

Myös *Einsteinin suhteellisuusteoria* oli loogisen positivismin motivoimaa: miltä maailma näyttää, jos *havaintojen tekonopeudelle* oletetaan yläraja, ja sitä lähestytään?

Ilman käännettävyysteesiä älykkäillä toimijoillamme A_{ert} olisi *jo periaatteessa* rajoittunut ymmärrys: silloin A_{ert} voisi kohdata (tai joutua muodostamaan) käsitteen, jonka merkitystä se ei tuntisi "kategorioidensa" eikä havaintojensa pohjalta. Eikä sillä ole muuta!

Valitessamme tekoälyn määritelmäksemme älykkäät toimijat luvussa 1.1.4 teimme samalla filosofisen valinnan siitä, mikä *ohjaa* toimintaa – tässä luvussahan on vasta puhuttu siitä, mitä on olemassa, ja miten siitä saadaan tietoa. Valitsimme *utilitarismin*, jonka filosofiassa muotoili *John Stuart Mill (1806–73)* vuonna 1863.

Sen mukaan on aina syytä pyrkiä "maksimoimaan hyötyä". Tämä "hyöty" *ei* kuitenkaan välttämättä tarkoita seuraavaa:

- Sitä, että aina olisi yksikäsitteinen hyödyn mitta kuten raha luvussa 1.2.
- Sitä, että tämä periaate *selittäisi* toiminnan: kuva 1 antaa kyllä herra Beardin hyötyfunktion, mutta ei kerro *mistä syystä* se on muodostunut sellaiseksi kuin on. Sen pohjalta voi vain ennustaa hänen tulevaa käytöstään.

- Tai sitä, että A_{ert} aina maksimoisi vain *omaa* etuaan. Se voi maksimoida myös yhteistä etua – tai jopa maksimoimalla omaa etuaan maksimoida samalla yhteistä etua.

  Esimerkkinä olkoot sosiaaliset hyönteiset kuten mehiläiset. Pesässä valtaosa naaraista (= työläiset) jättävät vapaaehtoisesti lisääntymisen kuningattarelle, ja keskittyvät hoitamaan sen jälkeläisiä.

  Tämä näyttää olevan vastaesimerkki "itsekkäälle evoluutioteorialle", jonka mukaan jokainen eläin pyrkii lisääntymään, koska sen jokainen geeni pyrkii kopioitumaan.

  Mehiläiset ovat kuitenkin *haplodiploideja*:

	koiras	**naaras**
ihminen	äidin X, isän Y	äidin X, isän X
mehiläinen	kuningattaren X, kuhnurin Y	kuningattaren X

Taulukko 1: Ihmisen ja mehiläisen sukukromosomit

  Työläinen hoitaa siis tulevaa kaksossisartaan hoitaessaan kuningattaren naarasmunaa. Se taas on yhtä tehokasta kopiointia kuin oma lisääntyminen. Kuningattaren koirasmunan hoitamiseen kuluvasta vaivasta vain puolet menee työläisen oman X-kromosomin kopiolle, ja toinen puoli isäkuhnurin Y-kopiolle. Silloin työläisen kannattaa jakaa aikansa suhteessa $3:1$ naaras- ja koirasmunien välillä.

  Kuningattarelle puolestaan naaras- ja koirasmunat ovat yhtä arvokkaita, kun lasketaan niiden kopiointitehokkuutta lapsenlapsituotannossa, joten hänen näkökulmastaan ajankäyttö saisi olla $1:1$.

  Todellisessa pesässä työläisten ajankäyttö vaihteleekin tuolla välillä riippuen siitä, ovatko työläiset vai kuningatar niskan päällä.

  Näin siis usean itsekkään toimijan yhteisössä voi syntyä (ulkoapäin katsoen) epäitsekästä yhteistyötä, kuten luvussa 1.1.4 jo todettiin.

1.4. Tekoäly, psykologia ja kielitiede

1960-luvulta alkaen on vallalla ollut *kognitiivinen* psykologia, joka tarkastelee tekoälyn tavoin aivoja informaation käsittelijänä. Monet tämän koulukunnan jäsenet lähtevätkin siitä, että kognitiivisten teorioiden pitäisi olla luvun 1.1.2 tapaan "tietokoneohjelmien kaltaisia" eli mallintaa kognitiiviset prosessit hyvinmääriteltyinä informaationkäsittely- ja muuntotoimituksina. 1943 Kenneth Craik muotoili tämän kannan 3 askeleena:

1. Ulkoa tuleva *ärsyke (stimulus)* pitää muuntaa *sisäiseen esitysmuotoon (internal representation)* . . .
2. jota toimijan *kognitiiviset prosessit* käsittelevät tuottaen tuloksina uusia sisäisiä esitysmuotoja. . .
3. jotka sitten muunnetaan takaisin toiminnaksi.

Eli älykkäiden toimijoiden A_{ert} teoriana!

Noin vuosina 1920–60 varsinkin Yhdysvalloissa oli kuitenkin vallalla *behaviorismi*, joka oli syntynyt vastalauseeksi sellaisten subjektiivisten "kyökkipsykologian" sanojen kuten "uskomukset", "tavoitteet" ja "järkeily" aiemmalle käytölle psykologisissa tutkimuksissa.

Behavioristien oma tutkimusohjelma oli taas kovasti luvun 1.3 loogisen positivismin oloinen: sallittiin vain tarkat objektiiviset mittaukset ja havainnot koe-eläinten reaktioista niille annettuihin ärsykkeisiin.

Craikin teoria taas mahdollisti näiden sanojen objektiivisen määrittelyn sopivien kognitiivisten tilojen ja prosessien niminä – palautti ne jälleen tieteellisiksi.

Vuonna 1957 johtava behavioristi *B.F. Skinner* julkaisi kirjan, jossa hän kuvasi behavioristisen käsityksen *kielen oppimisesta*. Nuori kielitieteilijä *Noam Chomsky* kirjoitti siitä kriittisen arvostelun [Cho97], jossa hän

osoitti, ettei behaviorismi pysty kuvaamaan *kielellistä luovuutta*: sitä, että lapsi(kin) pystyy ymmärtämään ja tuottamaan sellaisia oikein muodostettuja ilmauksia, joita hän ei ole aikaisemmin tavannut. Behaviorismi oli tiensä päässä – se ei yleistynyt monimutkaiseen käyttäytymiseen.

Chomskyn oma *generatiivisten kielioppien* teoria taas selitti kielellisen luovuuden kognitiivisella mallilla, joka mukaan (ihmis)mielellä olisi erityinen *kielikyky* (erinomainen johdatus tähän teoriaan on [Pin94]):

1. Se erottaa puheen muista kuulohavainnoista *kielen sääntöjen* avulla. Sitten se "kääntää" sen samoilla säännöillä *mentaleesin* eli "mielen kielen" ilmaisuksi, joka säilyttää puheen merkityksen (*syvärakenteen* eli *deep structure*).
2. Muu ajattelu saa syötteenään tuon mentaleesi-ilmauksen.
   Ajattelun ei silti tarvitse olle *pelkästään* kielellistä.
3. Kielikyky muuntaa käyttäen samoja sääntöjä vastakkaiseen suuntaan ajattelun tuloksen jälleen luonnolliselle kielelle, ja panee äärielimet liikkeelle tuottamaan puhetta.
   Itse äänen tuotto vaikuttaa yllättävän hankalalta askeleelta:
   - Vauvaiän jälkeen (vain) ihmisen kurkunpää muuttuu paremmin puheen tuotantoon sopivaksi – johtaen "väärään kurkkuun"!
   - Lapsi oppii viittomakielen 24–36 kk:n "kieliräjähdystä" aiemmin.

Teoria olettaa, että aivoissa on (evoluution tuloksena [NKN01]) *universaalikielioppi*, joka sisältää kaikki ne kielioppisäännöt, joita missään luonnollisessa kielessä voi esiintyä.

Tämän vahvan oletuksen voi heikentää oletukseksi, että aivoissa on säännöt "vierekkäisten" symboleiden liittämiseksi toisiinsa ja liitosten purkuun [Car97, Dea98].

Lapsi oppii kielensä havaitsemalla mitkä näistä säännöistä ovat päällä/pois päältä ympäristössä. Empiristin "Tabula rasa" -kielenoppimisteorioita häiritsee *syötteen vähyys (dearth of input)*: 2-vuotias lapsi on kuullut niin vähän puhetta, ettei hän ole ehtinyt muodostaa alusta sääntöjä joita hän alkaa menestyksellä soveltamaan.

1.5. Tekoälyn historiaa

Tekoälyn synty (1943–56)

Ensimmäisen tekoälyartikkelin kirjoittivat neurofysiologit *W. McCulloch* ja *W. Pitts* vuonna 1943 [MP43] yhdistämällä 3 aiemmin erillistä tieteenhaaraa:

- aivojen *hermosolujen* (neuronien) toiminnasta saadun tiedon;
- formaalin logiikan; ja
- laskettavuuden teorian.

He ehdottivat myös, että hermosolujen *verkot* voisivat *oppia*. *Donald Hebb* esitti 1949 sopivan säännön päivittää solujen välisten kytkentöjen voimakkuuksia.

1950-luvun alussa alalla toimi jokunen (lähinnä) yhdysvaltalainen tutkija:

- Vuonna 1950 fysikaalisen informaatioteorian luoja *Claude Shannon* kirjoitti shakkiohjelmia – vain tietokone puuttui. . .
- Vuonna 1953 Alan Turing (luvut 1.1.1 ja 1.2) teki samaa Iso-Britanniassa, kunnes hallitus kielsi häneltä tietokoneiden käytön moiseen humpuukiin.
- Vuonna 1951 *Marvin Minsky* ja *Dean Edmonds* rakensivat ensimmäisen hermoverkkokoneen Princetonin yliopiston matematiikan laitoksella.

  John von Neumann (luku 1.2) puolusti Minskyn väitöskirjaa arvostelijoilta, jotka eivät pitäneet sitä matematiikkana.

Samoin Princetonissa vaikuttanut *John McCarthy* siirtyi valmistuttuaan Darthmouth Collegeen, jossa hän, Minsky, Shannon ja *Nathaniel Rochester* (IBM) järjestivät 1956 kesäkoulun. 10 osallistujan joukossa olivat

Trenchard More (Princeton), *Arthur Samuel* (IBM) sekä *Ray Solomonoff* ja *Oliver Selfridge* (MIT).

Muita pidemmällä olivat kuitenkin *Allen Newell* ja *Herbert Simon* (Carnegie Tech, nyt Carnegie-Mellon U. eli CMU): heidän ohjelmansa *Logic Theorist* teki samanlaisia loogisia todistuksia kuin Principia Mathematica (luku 1.2) sisälsi. Hilbertin ohjelma kantoi siis sittenkin hedelmää!

Newellin ja Simonin perusajatuksena oli, että *kone etsii ratkaisua annettuun ongelmaan omaamansa tiedon ohjaamana*.

Alkuinnostus, suuret toiveet (1952–69)

Tänä aikana moni tekoälyn vastaväite tyyppiä "X on mahdotonta tietokoneelle" kumottiin antamalla vastaava ohjelma — kunhan X oli tarpeeksi hyvin rajattu ongelmaperhe, ja esimerkit riittävän pieniä senaikaisille tietokoneille.

Newell ja Simon jatkoivat ohjelmalla *General Problem Solver (GPS)*, joka käytti samanlaisia ongelmanratkaisutapoja kuin ihmiset. Tämä kognitiotieteellinen tekoälytutkimus (luku 1.1.2) on jatkunut CMU:ssa aina näihin päiviin.

Samuel kirjoitti vuodesta 1952 alkaen *oppivia* ohjelmia tammipeliin (checkers), jotka aikanaan ohittivat laatijansa taidot. Rochesterin johdolla jatkettiin Logic Theoristin aloittamaa työtä teoreemien todistamisessa logiikan säännöillä.

McCarthy siirtyi MIT:iin ja vuonna 1958 teki historiaa kolmesti:

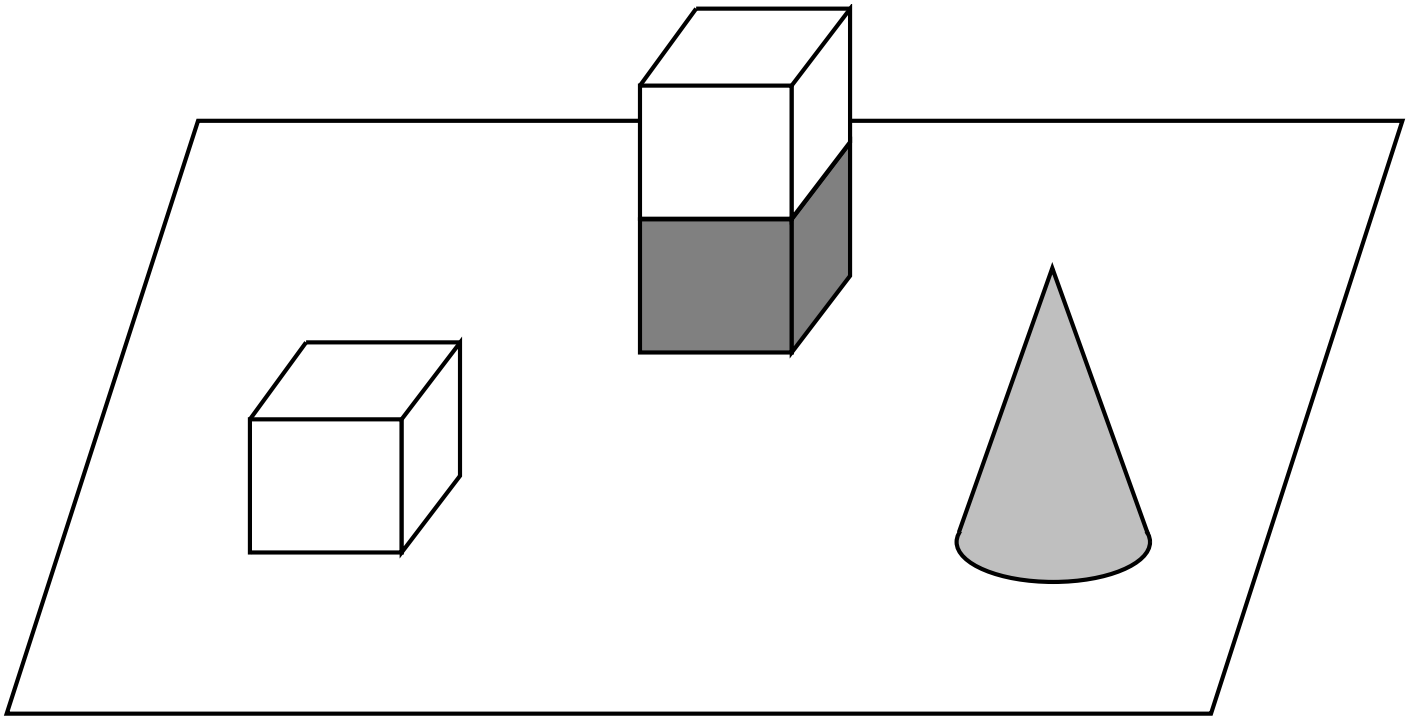
- Hän määritteli uuden *AI lab*oratorion ensimmäisessä raportissa ohjelmointikielen *Lisp* (*Lis*t *p*rocessing), joka on edelleen elossa tärkeimpänä tekoälykielenä — ja toiseksi vanhimpana vielä käytettynä kielenä heti vuonna 1957 esitellyn FORTRANin (FORmula TRANslator) jälkeen.
  Kielen perustietorakenne *lista* oli peräisin jo Logic Theoristia varten kehitetystä ohjelmointikielestä IPL.

- Yhdessä muiden MIT:n tietokoneihmisten kanssa hän ideoi *ositustuskäytön (time sharing)*.
- Hän esitteli — muttei toteuttanut — ehkä ensimmäisen *täydellisen* tekoälyohjelman *Advice Taker*. Perusidea oli Newellin ja Simonin, mutta nyt aksiomatisoitavana olikin matematiikan sijasta *arkimaailma*. Ohjelmaan pystyi myös lisäämään uusia aksiomeja ilman uudelleenohjelmointia, eli erotti toisistaan *tietämyksen esittämisen logiikalla* ja *näiden esitysten käsittelyn deduktiivisilla päättelysäännöillä*.

Samana vuonna myös Minsky siirtyi MIT:iin ja alkoi yhteistyö, joka alkoi rakoilla kun "scruffy" Minsky painotti toimivia ohjelmia logiikan kustannuksella ja "neat" McCarthy taas painotti tietämyksen esittämistä ja loogista päättelyä. Nämä kaksi kantaa ovat ottaneet mittaa toisistaan myöhemminkin — ja täydentäneet toisiaan.

Vuonna 1963 McCarthy perusti Stanford Research Instituten (SRI) ja jatkoi siellä Advice Taker -ideaansa.

MIT:ssa Minskyn oppilaat kehittivät puolestaan ohjelmia käsittelemään (usein simuloituja) *mikromaailmoja*:



Kuva 2: Palikkamaailma (blocks world) [RN95, kuva 1.3].

Palikkamaailmassa tutkittiin menestyksellisesti esimerkiksi *konenäköä*, *oppimista*, *luonnollisen kielen ymmärtämistä* ja *moniosaisten suunnitelmien tekoa*.

Innostus laantuu, toiveet karisevat (1966–74)

Tutkijoiden *ylioptimistiset* odotukset ja ennusteet jäivät toteutumatta, kun ratkaistavat *tehtävät kasvoivat ja monimutkaistuivat* (skaalautumisongelma):

- Monet varhaiset ohjelmat eivät olleet kovinkaan "älykkäitä" vaan suoriutuivat pienistä esimerkkitehtävistään yksinkertaisin syntaktisin manipulaatioin.

  Esimerkiksi luonnollisten kielten kääntämistä toisikseen pidettiin helppona: ohjelmoitaisiin Chomskyn kielioppiteoria ja kerättäisiin elektroninen sanakirja. Kuitenkin tekstin "ymmärtäminen" vaatii *ennakkotietämystä* sen aiheesta — maailmasta, ihmisistä,. . . — moniselitteisyyksien ratkomiseksi:

  "Olen nähnyt Helga-neidin $\left\{ \begin{array}{l} \text{kun minä olin} \\ \text{kun hän oli} \end{array} \right\}$ kylvyssä, ihanaa!"

- Monet etsintäongelmat tiedetään nyt NP-koviksi (luku 1.2).

  Vaikka ohjelmassa olisikin mekanismi ratkaisemaan tehtävä *periaatteessa*, ei tuo mekanismi välttämättä ratkaise sitä *käytännössä*.

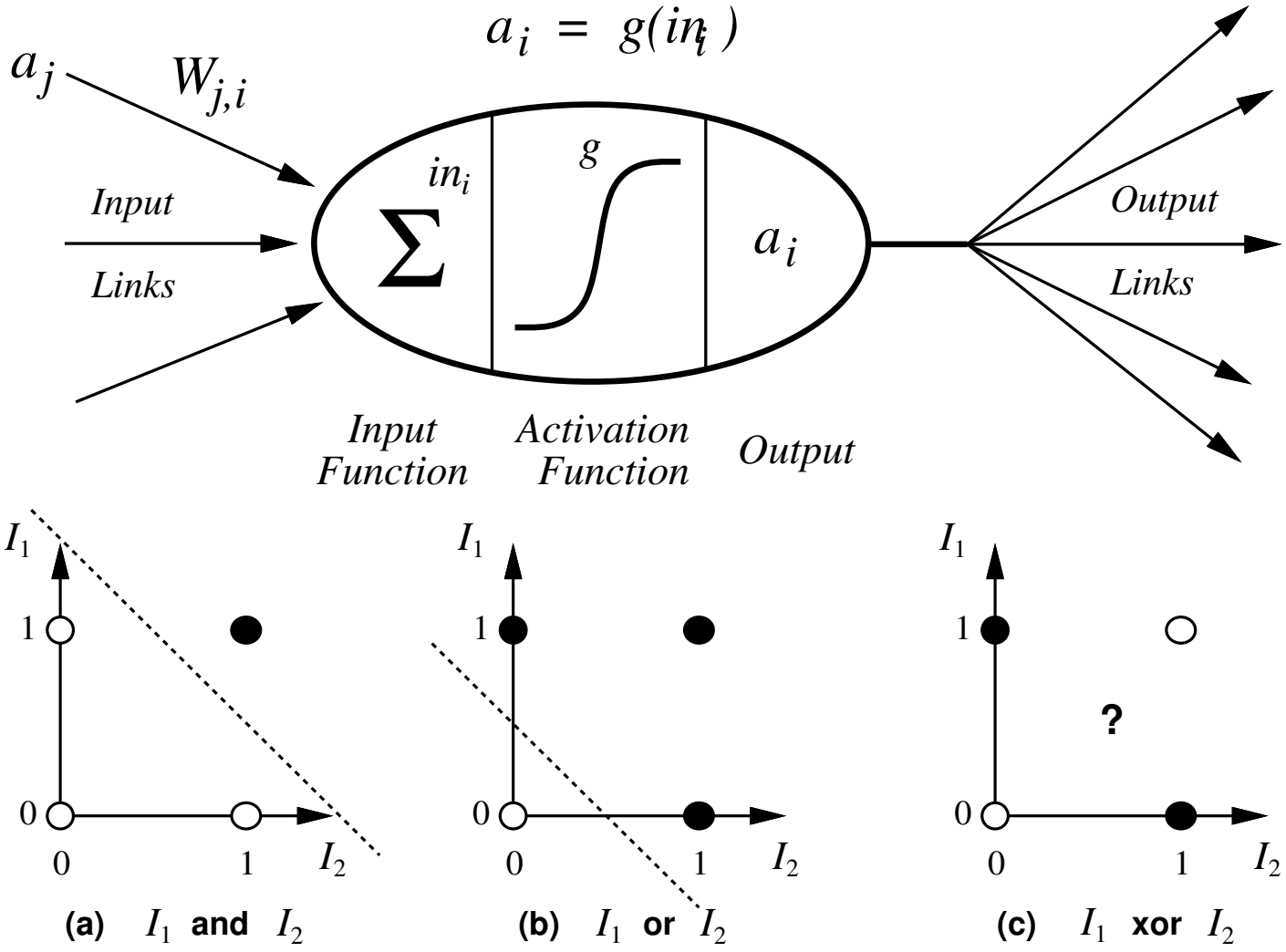
  Kyvyttömyys hallita tämä *kombinatorinen räjähdys* oli (muodollisena) syynä *Lighthill-raporttiin*, joka lopetti Iso-Britannian hallituksen tuen tekoälytutkimukselta kahta yliopistoa lukuunottamatta.

- Hermoverkkotutkimuksessa johdettiin teoreettisia tuloksia käytetyistä tekniikoista:

  1. *Frank Rosenblatt* osoitti vuonna 1962 *perceptronien konvergenssilauseen*, jonka mukaan hänen *oppimisalgoritminsa* pystyi sovittamaan *yhden* hermosolun kytkentäpainot vastaamaan mitä tahansa sellaista syötettä, jolle sopivat painot olivat olemassa.

     *Moni*soluisia verkkoja tutkittiin myös, ja uskottiin, että niillekin saataisiin pian samanlaisia tuloksia.

2. Vuonna 1969 (aiempi hermoverkkotutkija) Minsky ja *Seymour Papert* julkaisivat kuitenkin kirjan *Perceptrons*, jossa he osoittivat että nämä sopivat painot olivat olemassa vain hyvin yksinkertaisille syotteille.
   Vaikka monisoluiset verkot ratkaisevatkin tämän ongelman, ei niille tunnettu vielä oppimisalgoritmeja. Kirjan sisältämän terävän kritiikin (jota 2. painos 1988 lievensi) tuloksena tietojenkäsittelyllisen hermoverkkotutkimuksen rahoitus kutistui olemattomiin. . .



Kuva 3: Perceptron ja Boolen logiikka [RN95, kuvat 19.4 ja 19.9].

3. . . . peittäen alleen sen, että sellainen algoritmi oli juuri keksitty (tosin kansantaloustieteessä)! Hermoverkkojen tutkimus koki renesanssin vasta 1980-luvun loppupuolella, kun se keksittiin uudelleen.

Tietämyspohjaiset järjestelmät (1969–79)

Newell ja Simon olivat uskoneet *heikkoihin menetelmiin*, joissa itse

etsintäkone oli yleiskäyttöinen ja syötteenä annetut säännötkin kuvailivat yleispäteviä periaatteita. Näin saadut tuloksetkin olivat usein heikkoja.

Alettiin tutkia vaihtoehtoja, joissa valittiin ensin sovellusalue ja sitten kirjoitettiin hyvin sovellusaluekohtaisia sääntökokoelmia. Ei siis mallinnettu enää "yleistä" älykkyyttä vaan sitä suorempaa päättelyä jota *asiantuntija (expert)* tekee käyttäessään omaa erityisosaamistaan.

DENDRAL, joka päätteli näytteen molekyylirakenteen massaspektrometrin antamista mittaustuloksista, lienee ollut ensimmäinen *tietämysintensiivinen ohjelma*, jonka asiantuntemus johtui suuresta kokoelmasta erityistapauksia koskevia sääntöjä.

MYCIN puolestaan diagnosoi verisairauksia noin 450 säännön voimin lähes yhtä hyvin kuin alan erikoislääkärit. Se erosi DENDRALista kahdessa suhteessa:

- DENDRALin sääntökanta voitiin johtaa kyseeseen tulevista kemian teorioista; MYCINin pohjaksi ei ollut tarpeeksi tarkkaa lääketieteellistä teoriaa, vaan se kerättiin *haastattelemalla* alan spesialisteja.
- MYCINin sääntöihin oli liitettävä *varmuuskertoimia (certainty factors)* heijastamaan tarjolla ollutta vain likimääräistä, ei varmaa sovellusalueen tietämystä. Niistä tuli suosittu tapa mallintaa epävarmuutta sääntöpohjaisissa asiantuntijajärjestelmissä.

  Jos annetun säännöstön ja siitä tehtävien päätelmien rakenne on tietyllä tapaa yksinkertainen (kuten onneksi MYCINissä) niin varmuuskertoimet ovat eräs Bayesin teorian (luku 1.2) erikoistapaus; mutta *yleisesti* niiden lisääminen saattoi antaa aivan harhaanjohtavia tuloksia, joten niiden käyttöa ei enää suositella [RN95, luku 15.6].

Epävarmuuden mallinnukseen (varsinkin lääketieteellisessä diagnoosissa) käytettiin jo tuolloin myös perinteistä todennäköisyyslaskentaa ja utiliteettiteoriaa. Esimerkiksi probabilistinen päättelykone PROSPECTOR ehdotti koeporauksia paikassa, josta löytyi laaja molybdeeniesiintymä.

Luonnollisen kielen käsittelyä tutkittiin joko rajoittumalla yksinkertaisiin ympäristöihin (*Terry Winograd*in SHRDLU palikkamaailmassa) tai lisäämällä tarvittavaa taustatietoa ja päättelyä sen pohjalta (*Roger Schank* oppilaineen Yalessa).

Tosielämän sovelluksia varten kehitettiin myös uusia tiedonesitystapoja:

Prolog on logiikkaan perustuva ohjelmointikieli, joka syntyi Ranskassa ja Iso-Britanniassa (Skotlannissa) luonnollisen kielen käsittelyn tarpeista.

Planner on myöskin logiikkaan perustuva yhdysvaltalainen perhe kieliä monimutkaisten suunnitteluongelmien esittämiseen ja ratkaisuun.

kehykset (frames) ovat Minskyn *rakenteinen* vaihtoehto logiikalle, jossa tietämys esitetään hierarkisina rakenteina ja niiden solmujen välisinä suhteina. *Olio-ohjelmointi* voidaan nähdä tämän vaihtoehdon tuomisena tietämyksen esittämisestä ohjelmointiin.

Tekoälyteollisuuden synty (1980–88)

Alan kaupallinen liikevaihto kasvoi vuoden 1980 muutamasta miljoonasta vuoden 1988 2 miljardiin dollariin.

Japani käynnisti 1981 kansallisen "viidennen sukupolven tietokonehankkeen": pyrittiin Prolog-tekoälykoneeseen jossa olisi esimerkiksi täysimittainen luonnollisen kielen tunnistus. Yhdysvalloissa tämä nähtiin uhkana, joka kasvatti tekoälytyön kiinnostusta ja rahoitusta. Iso-Britanniassa *Alvey-raportti* käynnisti "uuden" tutkimusalueen nimeltä "älykkäät tietämyspohjaiset järjestelmät".

Samalla tuotettiin tekoälytehtäviin sopivia laite- ja ohjelmointiympäristöjä (Lisp-koneet ja -ympäristöt). Nämä markkinat ja niillä toimineet yritykset eivät ole kuitenkaan enää hengissä: tavallisilla mikrotietokoneilla saavutettiin lähes samat edut paljon halvemmalla.

Hermoverkkojen paluu (1986–)

Hermoverkkojen tutkimus oli jatkunut tietojenkäsittelyn ulkopuolella:

Fyysikot kuten *John Hopfield* mallinsivat niitä ja niiden vuorovaikutuksia alunperin kiinteän aineen atomeille kehitetyillä menetelmillä.

Psykologit kuten *David Rumelhart* ja *Geoff Hinton* mallinsivat niillä (ihmis)muistia.

1980-luvun puolivälin tienoilla monisoluisten verkkojen oppimisalgoritmi keksittiin uudelleen — ainakin 4 kertaa. . . — ja sitä sovellettiin moniin tietojenkäsittelyn ja psykologian oppimisongelmiin.

Nykytila (1987–)

Tekoälytutkimus on vakiintumassa "normaalitieteen tilaan", jossa "neat"-tutkijat testaavat ja kehittävät edelleen jo olemassaolevia teorioita todellisiin tilanteisiin. (Uusi "scruffy-vallankumous" voi olla tarpeen kunhan teorioiden rajat tulevat vastaan.)

Esimerkiksi puheen- ja käsialantunnistus on edennyt kaupalliseen käyttöön saakka, kun otettiin käyttöön *piilo-Markov-mallit* todennäköisyyslaskennasta aikaisempien *ad hoc*-ratkaisujen sijaan. Samoin epävarman tiedon hallinnassa on edetty korvaamalla varmuuskertoimet Bayesin ja päätösteorian menetelmin.

Varsinainen tavoite – "aito" tekoäly – näyttääkin olevan (vähintään) yhtä kaukana nyt kuin alkuinnostuksen vuosina; tällä välin olemme oppineet paljon sen hankaluuksista. . .

Kurssimme "älykäs toimija" -määritelmä tekoälylle onkin "kattokäsite" yhdistämään käsitteellisesti tekoälyn eri osa-alueet, ei lähitulevaisuuden tuotekehittelyä.

1.6. Tekoälyn omasta filosofiasta [RN95, luku 26]

Tekoälyn filosofia jakaa sitä koskevat väitteet kahteen päätyyppiin:

Heikko tekoäly kysyy "voidaanko kone saada käyttäytymään *ikään kuin* se olisi älykäs?"

Tämän väitteen arvostelijat esittävät seuraavanlaisia kantoja:

- On asioita, joihin tietokoneet eivät pysty, ohjelmoitiinpa ne miten tahansa.
- Tietyt tavat suunnitella älykkäitä ohjelmia tulevat epäonnistumaan.
- Näiden ohjelmien laatiminen on käytännössä mahdotonta.

Vahva tekoäly kysyy "onko tällaisella koneella *tietoisuus (consciousness)*?"

- Turingin määritelmä (luku 1.1.1) väitti, että heikosta tekoälystä seuraa vahva.
- Näin voi uskoa uskomatta heikkoon: jos voitaisiin, niin olisi.

Turing luetteli myös useita (heikon) tekoälyn vastaisia argumentteja [Tur50, luku 6], joista osa elää tänäänkin, esimerkiksi:

Matemaattinen vastaväite lähtee Gödelin (tai Turingin omista) epätäydellisyystuloksista (luku 1.2), joiden mukaan mikään formaali systeemi — kuten tietokone — ei voi ymmärtää tiettyjä matemaattisia väittämiä jotka koskevat sitä itseään.

Gödel on ne kuitenkin todistanut ja muut matemaatikot todistuksen ymmärtäneet. Siis he ovat tehneet koneelle mahdottoman älykkyyttä osoittavan ja vaativan toimituksen.

Eräs vastaus on, että vaikka formaali systeemi — matemaatikko tai tietokone — pystyykin käsittelemään *itseään "yksinkertaisempien"* systeemien väittämiä, se ei pystykään tekemään samaa omalle väitteelleen. Tämä erottaa "kielen" *jota* tarkastellaan ja "metakielen" *jossa* tarkastellaan.

Roger Penrose esitti tämän vastaväitteen kirjassaan "Keisarin uudet aatteet" vuonna 1989 [Pen89]. Penrose. . .

- halusi toisaalta pitää kiinni aivotoimintojen ja sitä kautta myös tietoisuuden fysikaalisesta perustasta,
- mutta toisaalta halusi kieltää niiden mekaanisen laskettavuuden.

Hän päätyi arvelemaan, että aivoissa tapahtuvat kvanttimekaaniset ilmiöt ovat tärkeitä tietoisuudelle. (Liekö lääke tautia pahempi?)

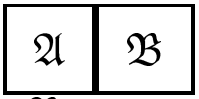
Käyttäytymisen epämuodollisuus on vastaväite, jonka mukaan inhimillinen käyttäytyminen on niin monimutkaista, ettei sitä voi vangita sääntöihin.

Näin on ajatellut esimerkiksi *Hubert Dreyfus* [DD88].

Kannan edustajilla on kuitenkin jälleen kaksijakoinen ongelma:

- Antaa vastaesimerkki älykkyyttä vaativasta toiminnasta, jota ei voi kuvailla säännöin.
- Antaa vaihtoehtoinen selitys omalle esimerkilleen ilman että samalla tulee itse kehittäneeksi sille säännöt.

Tunnetuin vastaväite vahvalle tekoälylle lienee *John Searlen "kiinalainen huone" (Chinese Room) -ajatuskoe* [Sea80, Sea90, CSC90][Bod90, §4]:

- Istun kopissa, jossa on vain
  - yksi luukku nimeltä 'sisään'
  - toinen luukku nimeltä 'ulos'
  - ja valtava kirja jonka kullakin sivulla on kiinalainen kirjoitusmerkki enkä minä osaa kiinaa
- mutta osaan tehdä seuraavaa:
  - kun 'sisään'-luukusta näytetään minulle jokin merkki $\mathfrak{A}$
  - niin etsin kirjasta sitä vastaavan aukeman 
  - ja näytän 'ulos'-luukusta vastaavan merkin $\mathfrak{B}$.
- Läpäisköön tämä koppi Turingin testin keskustelemalla kiinaksi täysin ihmisen tavoin.

  Kuka tai mikä kopissa
  - osaa kiinaa?
  - on tietoinen (keskustelluista asioista)?

Searlen mukaan ei kukaan eikä mikään!

- Searlen johtopäätökset:
  - Ohjelmaa suorittava tietokone ei ole tietoinen — huonehan on eräänlainen Turingin kone.
  - Tietoisuutta ei voi koodata ohjelmaksi — kirjahan on eräänlainen ohjelma.

Searle haluaa tämän vuoksi eroon tietoisuuden Turing-määritelmästä pelkkänä syöte/tulos-vastaavuutena.

Toisaalta Searle (kuten Penrose) haluaa pitää kiinni tietoisuuden fysikaalisesta perustasta aivotoiminnassa: "brains cause minds".

Searlen haluja voi toisaalta pitää ristiriitaisina *aivoproteesikokeen (the Brain Prosthesis Experiment)* nojalla:

- Oletetaan lääketieteen kehittyneen niin pitkälle, että yksittäinen neuroni voidaan korvata täsmälleen saman syöte/tulos -käyttäytymisen omaavalla mikropiirillä ilman että aivojen toiminta häiriintyy leikkauksessa.
- Vaihdetaan saman potilaan kaikki neuronit *yksi kerrallaan*. Miltä potilaasta itsestään tuntuu, häviääkö hänen tietoisuutensa? Searlen mukaan häviää; tutkitaan mitä tästä seuraa.
- Koska yksittäisten aivonosien syöte/tulos -käyttäytyminen ei muutu, niin koko potilaan havaittava käyttäytyminen ei muutu yllättäen.
  Tällöin potilaan *oma tahto katoaa yhtäkkiä*, muuten hän ehtisi huudahtaa "apua!"
  Siis on jokin "kriittinen massa" neuroneita, jotka juuri ja juuri riittävät oman tahdon olemassaoloon.
- Kun kaikki vaihdot on suoritettu, on saatu alkuperäisten aivojen kanssa syöte/tulos -käyttäytymiseltään identtiset "sähköaivot", joilta Searle kuitenkin kieltää tietoisuuden.
  Silloin potilaan alkuperäistenkään aivojen tietoisilla tiloilla ei ollut mitään vaikutusta hänen käyttäytymiseensä, koska nämä tilat ovat nyt hävinneet mutta käyttäytyminen säilyi.

- Peruutetaan lopuksi tämä leikkaus jälleen neuroni kerrallaan, ja kysytään sen jälkeen "miltä leikkaus tuntui?"

  Koska potilaan neuronien syöte/tulos -käyttäytymistä ei taaskaan häiritty, hänen pitäisi muistaa tuntemukset myös siltä ajalta jolloin aivoissa ei ollut yhtään neuronia.

  Searlen mukaan sinä aikana ei aivoissa kuitenkaan ollut ketään kokemassa noita tuntemuksia.

Toisaalta todellinen neurotutkimus ei ehkä salli näin teräviä tulkintoja:

- Tietoisuuden "kriittinen neuronimassa" ei ehkä olekaan kiinteä suure: tietoisuus saattaa tarvita kullakin hetkellä käytössään olevien aktiivisten neuronien lisäksi *vapaita lisäneuroneita* jotka voivat aktivoitua tulevaisuudessa samalla kun osa nykyisistä passivoituu. [TE88]

  Tietoisuuden kulloinenkin sijainti voi siis "matkailla" aivoissa ajan kuluessa, ja tarvita siihen vapaata kulkutilaa.
- Aivot ovat myös yksi *elin* joka on yhteydessä muuhun ruumiiseen myös muuten kuin lähettämällä ja vastaanottamalla hermoimpulsseja.

  Tietoisuus saattaa olla jopa *evoluution sivutuote* matkalla kohden yhä tarkempaa eliön itsesäätelymekanismia [Dam99].

  Tietoisuutta ei silloin ehkä voi tutkia erottamalla aivoja muusta ruumiista tai edes ei-tietoisista aivoprosesseista.

2. Suunnitelmia älykkäiksi toimijoiksi

— Mikä tuo on? minä huudahdin haltioissani.
— Karuselli, Fredrikson sanoi. Etkö muista. Minähän tein sinulle konepiirustuksen. Läpileikkauksen ja kaikki.
— Mutta ei se tuolta näyttänyt, minä intin. Tässä on hevosia, musiikkia ja viirejä ja punaista kultaa!
— Ja rataspyöriä, Fredrikson sanoi. *Tove Jansson: Muumipapan urotyöt*

Meille on annettu tehtäväksi suunnitella automaattinen taksikuski (= sen ohjausohjelma). Sopimusehdot on kirjattu seuraavaksi *PAGE-kuvaukseksi:*

Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Kuva 4: Taksikuskin PAGE-kuvaus [RN95, kuva 2.6].

Saadaan 5 erilaista toimijaluokkaa riippuen siitä, miten Algoritmin 1 (kalvo 8) eri osat — *muisti* ja *parhaan teon valinta* — päätetään toteuttaa.

2.1. Taulukko-ohjatut toimijat

Voitaisiin yrittää *taulukoida etukäteen* paras teko kullekin havaintojonolle:

function TABLE-DRIVEN-AGENT(*percept*) **returns** *action*
  static: *percepts*, a sequence, initially empty
    table, a table, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  action ← LOOKUP(*percepts*, *table*)
  return *action*

Kuva 5: Taulukko-ohjattu toimija [RN95, kuva 2.5].

Teoriassa tällainen optimaalinen (ääretön) taulukko LOOKUP on kuhunkin tehtävään kyllä olemassa, mutta. . .

- Esimerkiksi shakkipelin (jossa havaintomaailma on yksinkertainen ja jonot äärellisiä) taulukossa on noin 35^{100} alkiota.
  Minne se mahtuisi?
- Suunnittelijan tulisi kyetä myös *löytämään* taulukko.
- Tulos ei ole autonominen, koska kaikki on laskettu valmiiksi. Jos maailma muuttuu yllättäen (shakkipelin sääntöjä muutetaan) niin toimija olisi hukassa, tarvittaisiin uusi taulukko.
- Vaikka toimijaan lisättäisiinkin *oppimismekanismi*, niin useimmiten kestäisi aivan liian kauan ennen kuin toimija olisi oppinut taulukon (useimmin käytytyt osat). Mutta toimija olisi nyt autonominen!

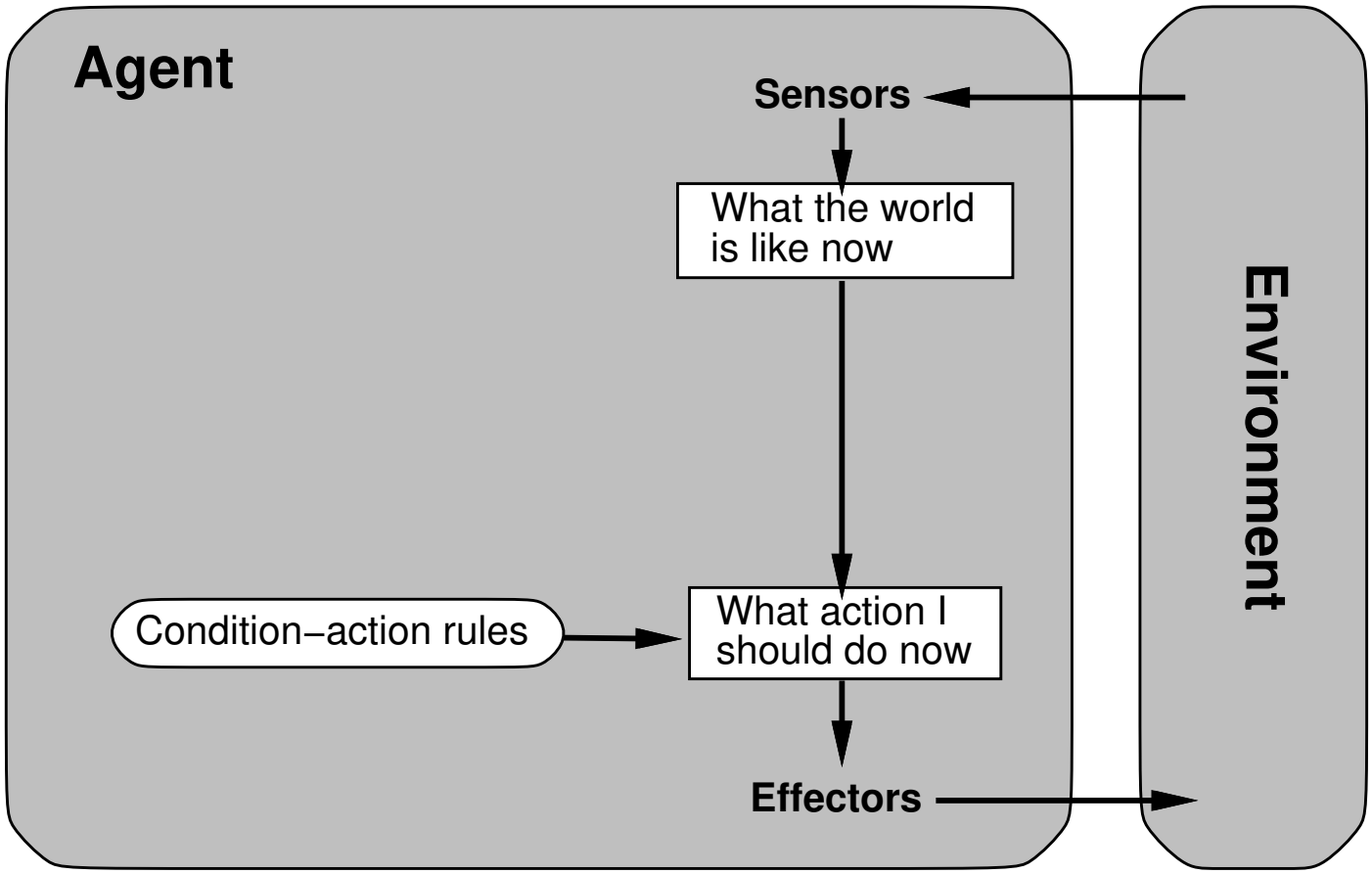
Taulukko-ohjattu toimija siis toteuttaa älykkään toimijan määritelmän, mutta se voidaan usein toteuttaa myös muilla, käytännöllisemmillä tavoilla.

Luvun 1.6 aivoproteesikoe voitaisiin tehdä myös taulukolla mikropiirin sijaan — jolloin taulukko olisi tietoinen!

Luvun 2.1 optimaalista taulukkoa LOOKUP voidaan kutistaa ottamalla käyttöön "vaistonvaraisia" *jos-niin-sääntöjä*

if edellä ajava auto jarruttaa **then**
  aloita jarrutus
end if

esittämään havaintojonojen odotettavissa olevia säännönmukaisuuksia. Tulos on "behavioristinen" toimija sääntöinään ärsyke-refleksi-parit.



function SIMPLE-REFLEX-AGENT(*percept*) **returns** *action*
  static: *rules*, a set of condition-action rules

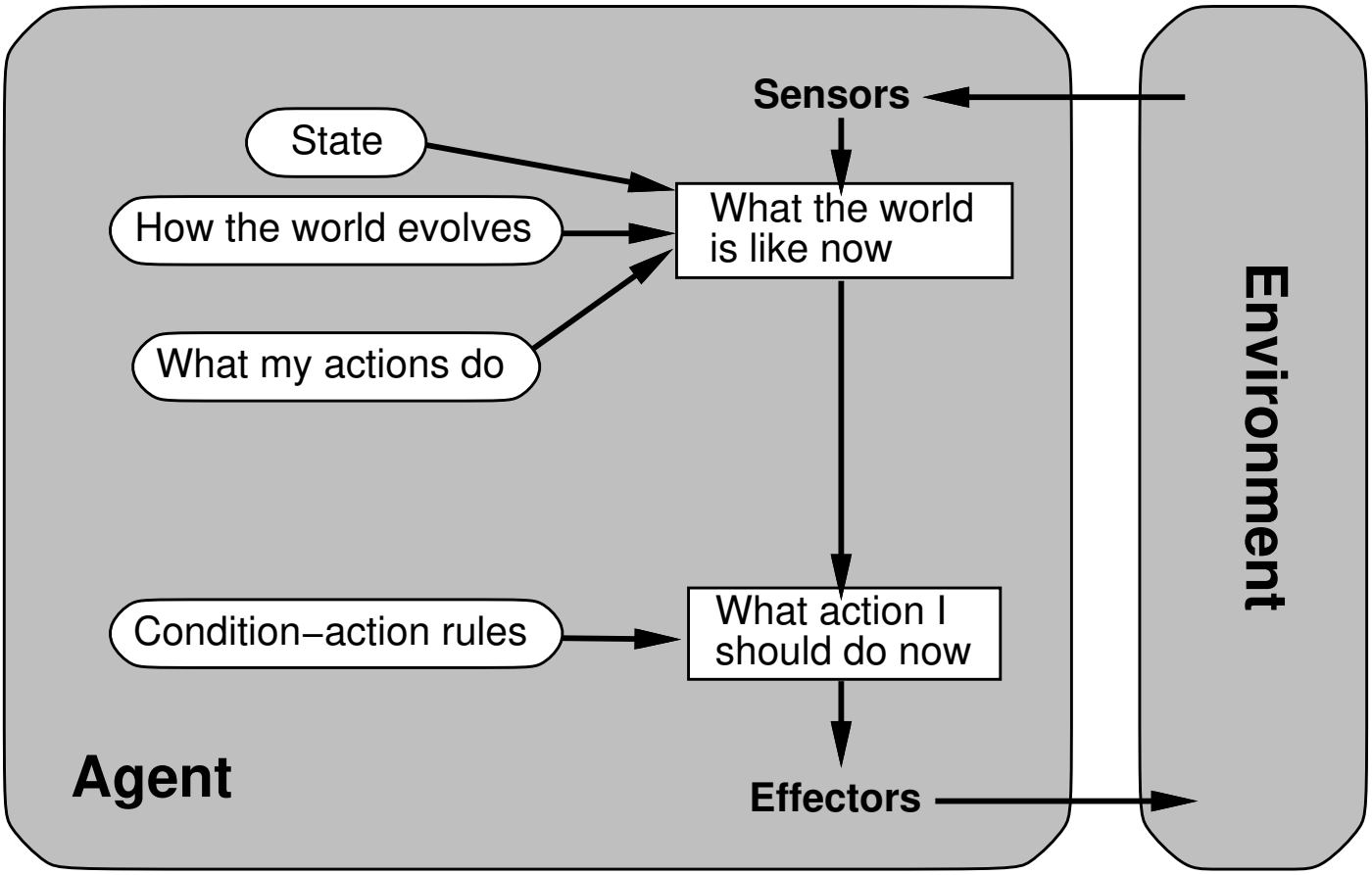
  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← RULE-ACTION[*rule*]
  return *action*

Kuva 6: Muistiton heijastetoimija [RN95, kuvat 2.7 ja 2.8].

2.3. Muistavat heijastetoimijat

Luvun 2.2 toimijan sääntöjen valinnassa voi tarvita tietoa ympäristön *muutoksista*: miten erottaa kääntyvä auto jarruttavasta jonka toinen takavalo on palanut?

Ympäristö muuttuu sekä toimijan omien tekojen seurauksena että siitä riippumatta. UPDATE-STATE päivittää toimijan nykyistä "maailmankuvaa", ja sen pohjalta valitaan käytettävä sääntö.



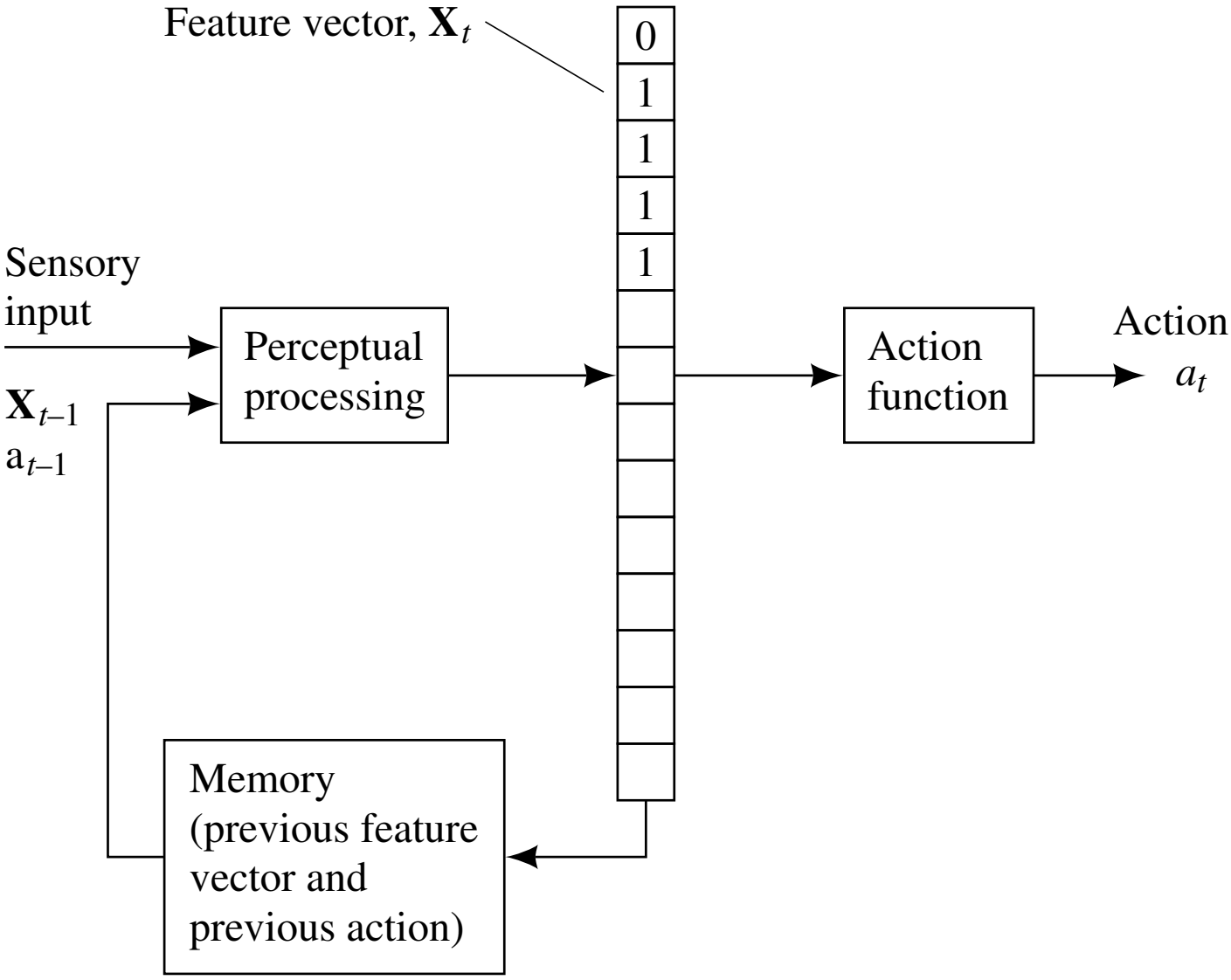
function REFLEX-AGENT-WITH-STATE(*percept*) **returns** *action*
  static: *state*, a description of the current world state
    rules, a set of condition-action rules

  state ← UPDATE-STATE(*state, percept*)
  rule ← RULE-MATCH(*state, rules*)
  action ← RULE-ACTION[*rule*]
  state ← UPDATE-STATE(*state, action*)
  return *action*

Kuva 7: Muistava heijastetoimija [RN95, kuvat 2.9 ja 2.10].

Havainnoista voi myös eristää *piirteitä (features)* ja muistaa ne. Saadaan tilakone jolla on

- syötteenä seuraava havainto
- sisällään (äärellinen) piirremuisti
- tulosteena seuraava teko.



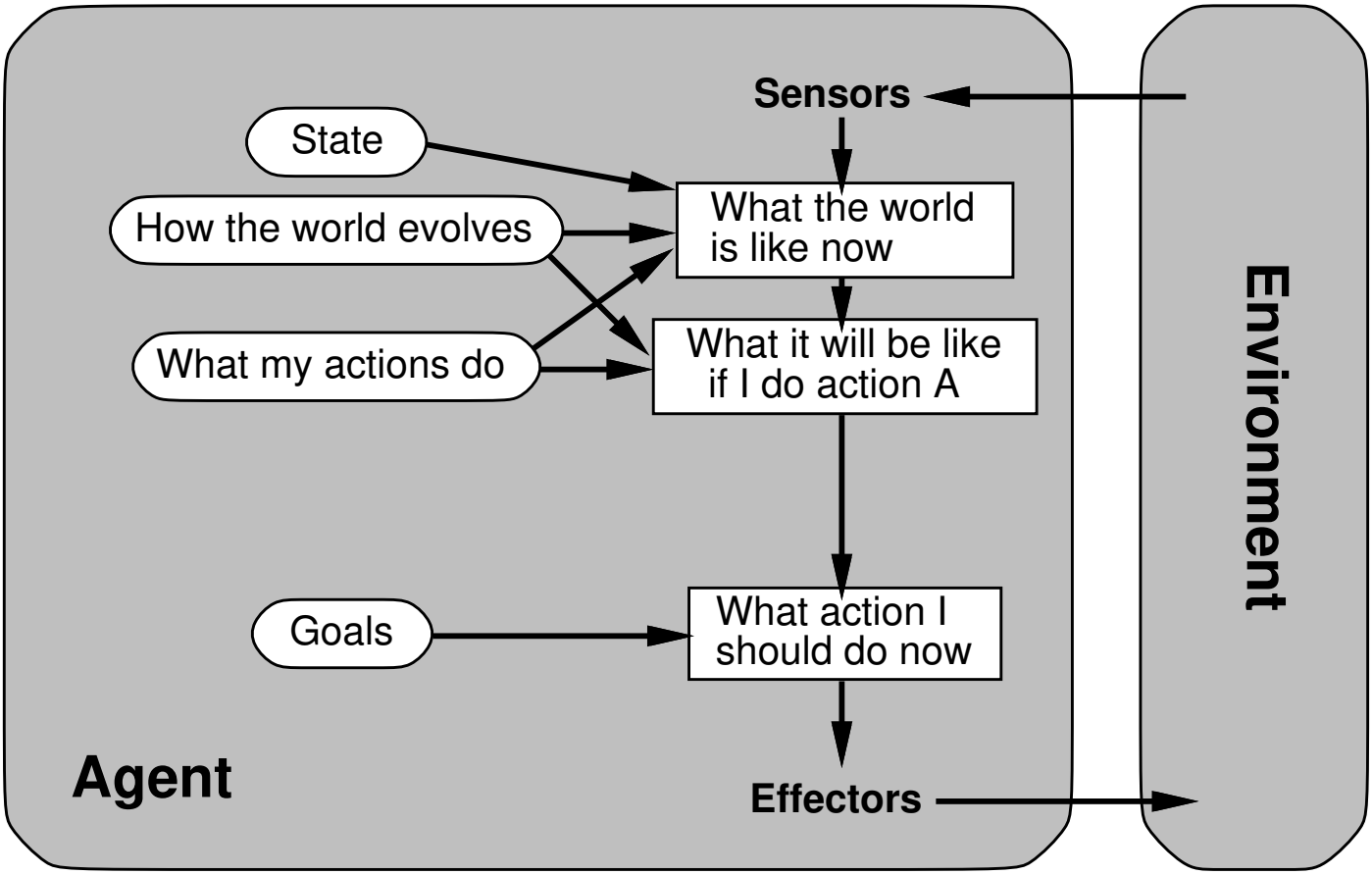
© 1998 Morgan Kaufman Publishers

Kuva 8: Piirrevektorin muistava tilakone [Nil98, kuva 5.1].

McCulloch ja Pitts esittivätkin (implisiittisesti) äärelliset automaatit muotoillessaan loogisen teoriansa oppimattomille hermoverkoille (luku 1.5).

2.4. Päämäärähakuiset toimijat

Luvun 2.3 toimijat ovat edelleen heijastetoimijoita. Kuskilla on myös *tavoitteita* jotka ohjaavat tehtäviä valintoja — kuten päästä asiakkaan sanomaan osoitteeseen.



Kuva 9: Päämäärähakuinen toimija [RN95, kuva 2.11].

Toimija, joka "tietää" tavoitteensa, ei enää ole heijastetoimija:

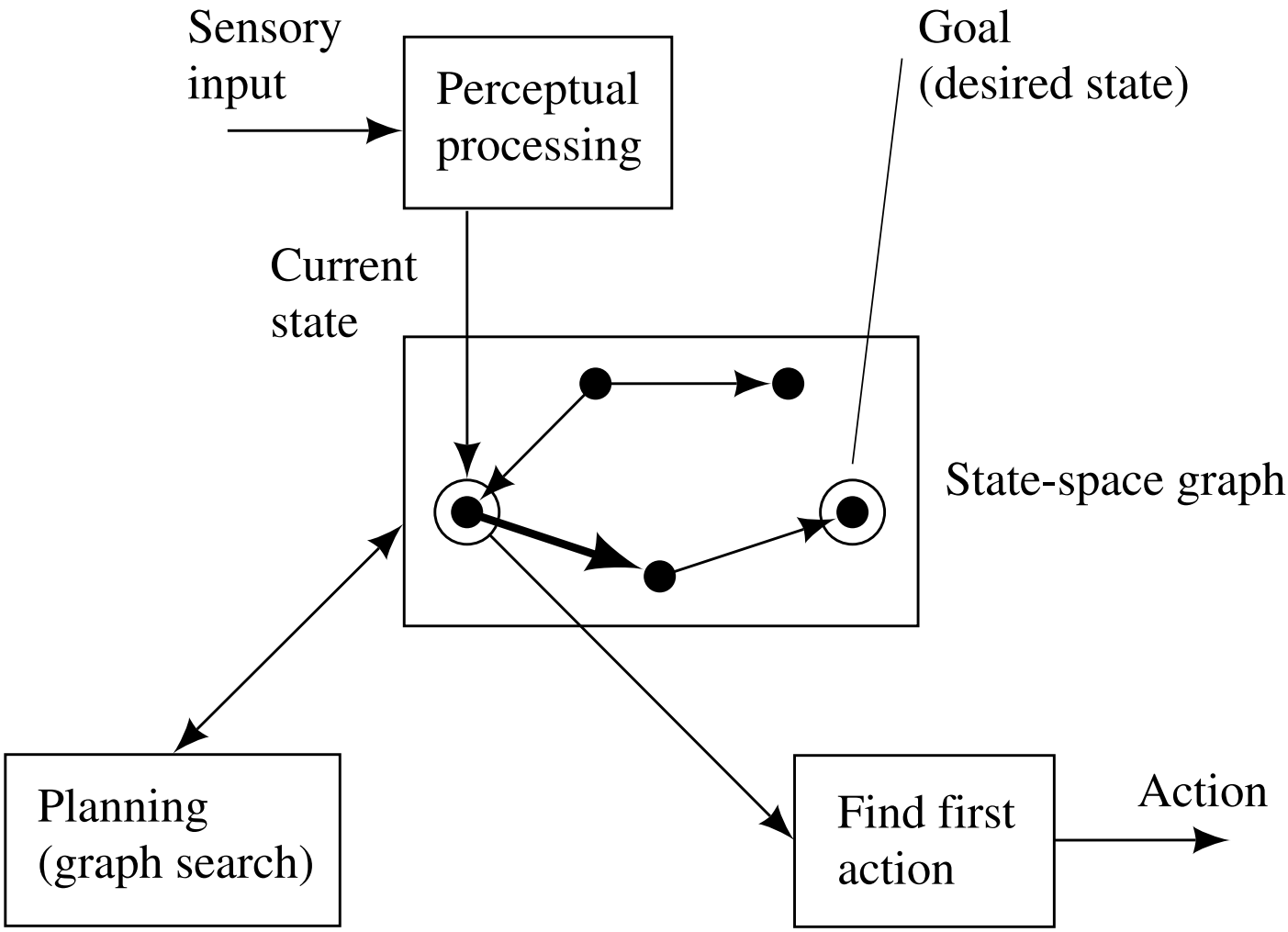
- Samat säännöt saattavat soveltua vaikka tavoitteet vaihtuisivatkin; tavoite voi muuttaa sitä ehtoa, jonka perusteella tekoja pidetään hyvinä tai huonoina.
- Sama tavoite saattaa olla toimijan saavutettavissa vaikka säännöt muuttuisivatkin; toimija voi vaihtaa menetelmää.
- Toimija järkeilee ehdollisesti: "Mitä tästä teostani seura*isi*? Ve*isi*kö se lähemmäs tavoitettani?"

Etsintä on keskeinen tekoälyn alue, koska sen menetelmät tarjoavat tapoja päätellä, onko annettu tavoite saavutettavissa annetuilla säännöillä.

Suunnittelu on toinen keskeinen alue, koska sen menetelmät tarjoavat tapoja osittaa kokonaistavoite osatavoitteiksi ja niiden välisiksi riippuvuuksiksi.

Suunnittelu voidaan usein nähdä etsinnäksi — suunnitelman etsinnäksi.

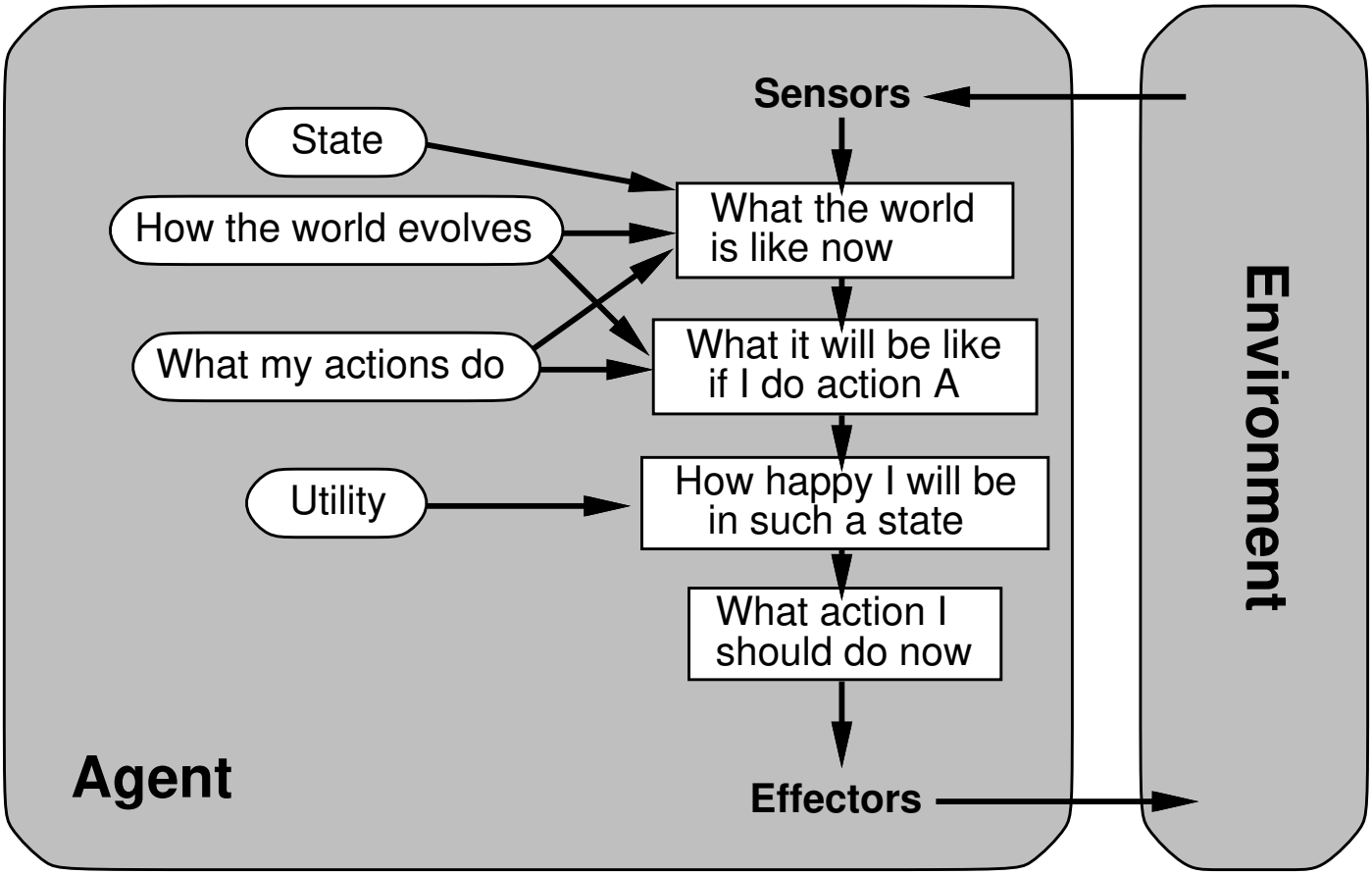
Kurssimme käsitteleekin tämän vuoksi etsintää ja suunnittelua symbolisin välinein.



© 1998 Morgan Kaufman Publishers

Kuva 10: Etsimällä suunnittelevan toimijan rakenne [Nil98, kuva 10.1].

Luvun 2.4 päämäärähakuiselle toimijalle kelpaa periaatteessa mikä tahansa jono tekoja, joka lopulta saavuttaa tavoitteen. Mutta asiakas ei ehkä halua "turistikierrosta" vaan suoremman reitin.



Kuva 11: Hyötyä maksimoiva toimija [RN95, kuva 2.12].

Eri vaihtoehdoista voidaan valita "paras" vertaamalla "mahdollisten maailmankuvien" S hyötyjä $U(S)$. Se on myös tapa valita, kun. . .

- kaikkia tavoitteita ei voi saavuttaa yhtä aikaa, tai
- tavoitteiden saavuttaminen on epävarmaa.

Hyötyfunktio U voi myös *ohjata etsintää*.

Hyötyohjattu toimija voidaan joskus toteuttaa luvun 2.4 tehokkaampana päämäärähakuisena toimijana. (Kääntäen, jokaisella rationaalisella toimijalla on jokin maksimoitava hyötyfunktio.)

2.6. Ympäristöoppia

Toimijan suunnitteluun vaikuttavat tietenkin myös sen aiotun toimintaympäristön ominaispiirteet. Ympäristöjä voidaan luokitella seuraavasti:

saavutettavissa vai ei? Ympäristö on saavutettavissa jos kukin havainto sisältää kaiken (tarpeellisen) informaation sen nykytilasta.

Silloin toimijan ei tarvitse käyttää muistiaan ympäristön seuraamiseen.

deterministinen vai epädeterministinen? Ympäristö on deterministinen jos sen nykytila ja toimijan siinä valitsema teko määräävät täysin sen seuraavan tilan.

Silloin toimijan ei *periaatteessa* tarvitse toimia epävarman tiedon pohjalta. Mutta jos ympäristö ei olekaan saavutettavissa, tai jos sen kaikkien ominaisuuksien seuraaminen on liian vaikeaa, niin se saattaa *näyttää toimijasta* epädeterministiseltä.

jaksoittainen vai ei? Ympäristö on jaksoittainen, jos toimijan "elinkaari" voidaan jakaa erillisiin jaksoihin. Kukin jakso koostuu siitä, että toimija havainnoi ensin ympäristöään, ja tekee sitten tekonsa. Eri jaksot eivät vaikuta toisiinsa.

Jaksoittaisissa ympäristöissä toimijan ei tarvitse suunnitella tulevaisuuttaan, koska kunkin jakson paras teko riippuu vain sen aluksi tehdyistä havainnoista.

staattinen vai dynaaminen? Ympäristö on dynaaminen, jos se saattaa muuttua toimijan miettiessä mitä tehdä seuraavaksi.

Staattisissa ympäristöissä toimijan ei tarvitse seurata ympäristöä koko ajan eikä välittää ajan kulumisesta.

Jos ympäristö ei muutu, mutta ajan kulumisesta pitääkin välittää, on se *semistaattinen*.

diskreetti vai jatkuva? Ympäristo on diskreetti, jos se koostuu rajallisesta määrästä selkeästi toisistaan erottuvia mahdollisia havaintoja ja tekoja.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Kuva 12: Erilaisia ympäristöjä [RN95, kuva 2.13].

Yleensä toimijat suunnitellaan ennalta sovittuun *ympäristöluokkaan*, joissa jotkut peruspiirteet pysyvät toisten vaihdellessa. Esimerkiksi shakkiohjelma toimii ympäristöissä, joissa

- vastapelaaja vaihtelee
- mutta pelin perussäännöt (laudan ruutujen määrä, nappuloiden luvalliset siirrot,. . .) pysyvät.

Toimijan menestyksellisyyttä arvioidaan sen perusteella, miten onnistuneesti se toimii ympäristöluokkansa eri jäsenissä.

Jos ympäristöluokassa on jokin luonteva todennäköisyysmitta, niin voidaan tarkastella myös toimijan keskimääräistä/odotettavissa olevaa menestystä.

3. Ongelmanratkaisumenetelmiä [RN95, Nil98, osa II]

Tarkastellaan päämäärähakuisia toimijoita (luku 2.4), joiden tavoitteena on löytää sellainen tekojono, joka saavuttaa päämäärän ympäristön nykytilasta alkaen. Siis *ensin suunnitellaan, sitten tehdään* (jos viitsitään).

(Tekojonon) *etsintä* on siis keskeinen ongelma(nratkaisumenetelmä)!

Erotetaan toisistaan

ympäristön tila eli se miten asiat todellisuudessa ovat

etsinnän tila eli se mitä ympäristön tilasta on selvitetty etsinnän kuluessa.

Pois jätetään siis ne ympäristön piirteet

- joilla ei ole merkitystä nykyisessä etsintäongelmassa
- tai joista etsijä ei voi saada varmaa tietoa.

Esimerkki: *imuriympäristö*, joka koostuu 2 vierekkäisestä ruudusta, joissa voi olla likaa ja/tai itse imuri. Imurin teot:

L: Jos imuri on nyt oikeanpuoleisessa ruudussa, niin se siirtyy vasemmanpuoleiseen ruutuun.

R: Jos imuri on nyt vasemmanpuoleisessa ruudussa, niin se siirtyy oikeanpuoleiseen ruutuun.

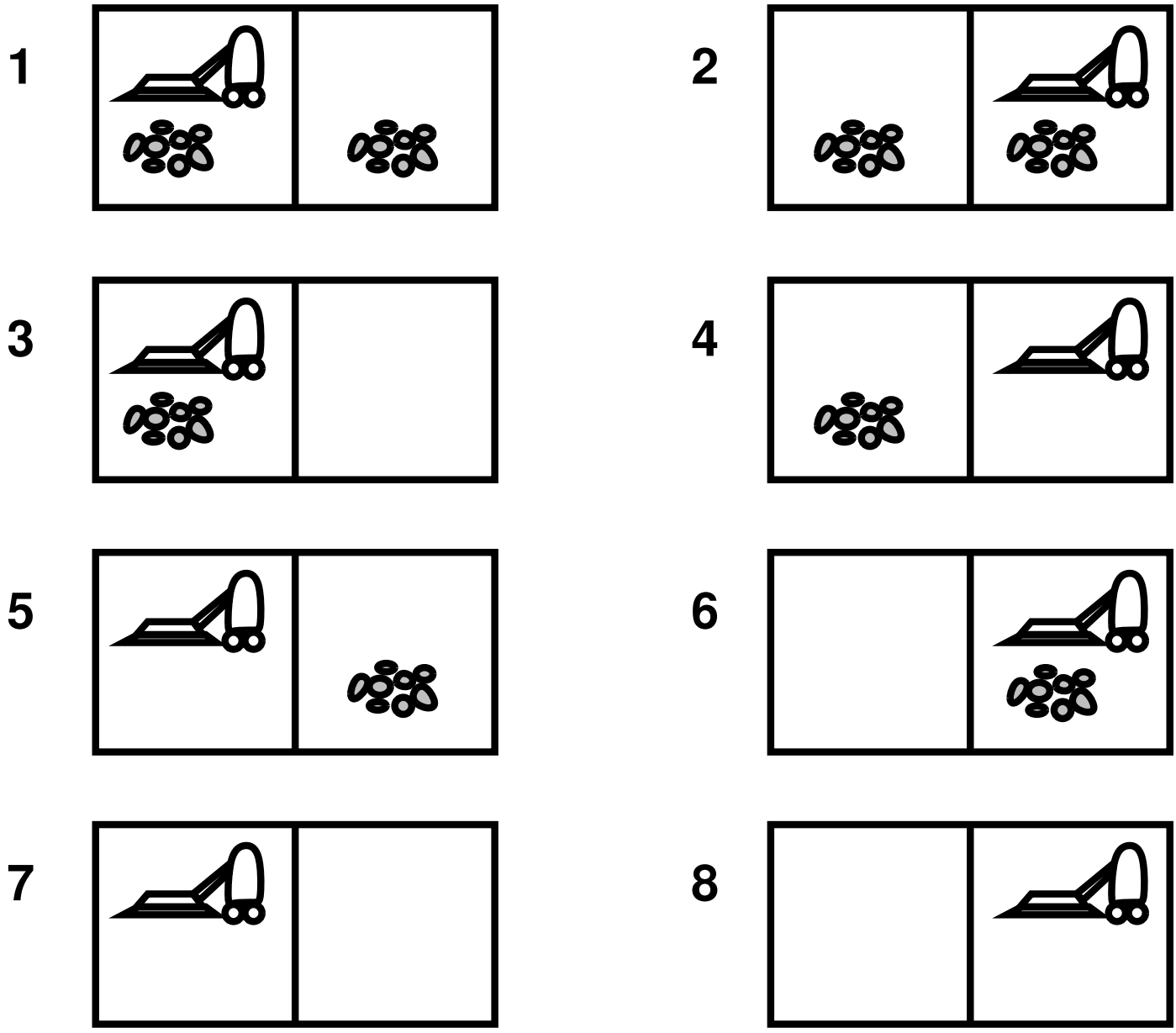
S: Jos siinä ruudussa, missä imuri on nyt, on likaa, niin se imetään pois.

Tavoitteena on imeä pois kaikki lika.

Tällä *ympäristöllä* on $2 \times 2^2 = 8$ mahdollista tilaa (kuva 13):

- Imurilla on 2 mahdollista ruutua.
- Ruudussa joko on tai ei ole likaa.

Tavoite on päästä tilaan 7 tai 8.



Kuva 13: Imuriympäristön mahdolliset tilat [RN95, kuva 3.2].

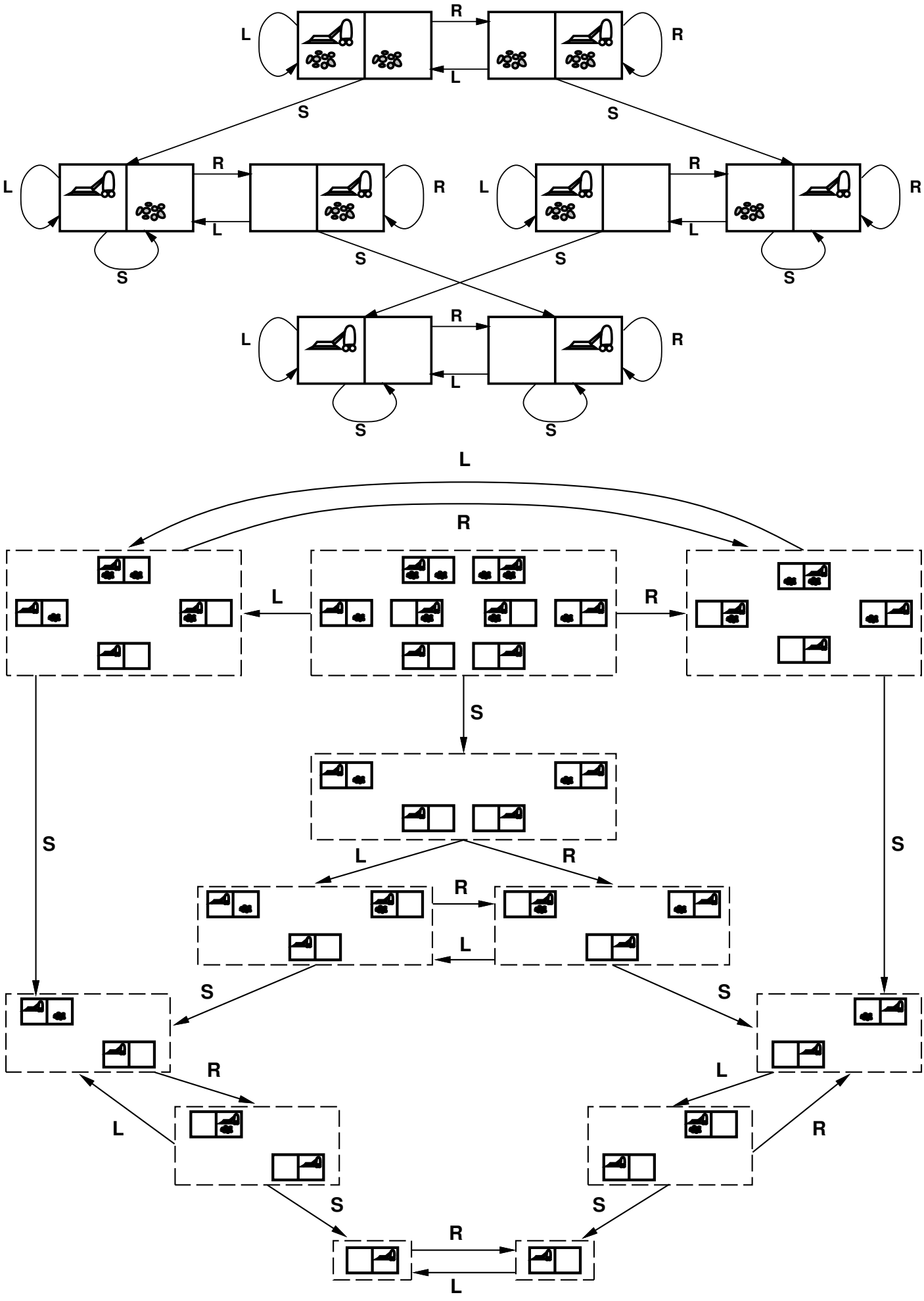
- Jos ympäristö on imurin *saavutettavissa* ja *deterministinen*, niin silloin etsinnän tilat yhtyvät ympäristön tiloihin (kuvan 14 yläpuolisko).

  Kyseessä on *yksitilainen (etsintä)ongelma*: jokaisessa etsinnän tilassa on tasan yksi ympäristön tila.

  Puhutaan *etsintäavaruudesta*: etsinnän tiloista ja niiden välisistä tekoreiteistä.

  Etsijä kulkee tässä avaruudessa kohti tavoitetilaa, ja ratkaisu on näin kuljettu *polku*.

  Silloin imuri voi esimerkiksi havaita ympäristön olevan tilassa 5, ja päätellä että tekojonolla **RS** se pääsee tavoitetilaan 8.



Kuva 14: Yksi- ja monitilainen ongelma [RN95, kuvat 3.6 ja 3.7].

- Entä jos ympäristö ei olekaan saavutettavissa — jos vaikka imuri ei aisti sijaintiaan eikä likaa?

  Silloin jokaiseen avaruuden tilaan kuuluvat *kaikki mahdolliset* ympäristön tilat — *moni*tilainen ongelma (kuvan 14 alapuolisko).

  Imuri käynnistyy avaruuden tilassa $\{1, \ldots, 8\}$; kaikki ympäristön tilat ovat vielä mahdollisia.

  Jos imuri aloittaa teolla **S**, niin sen jälkeen on mahdotonta että imurin oma ruutu (kumpi se onkin) olisi likainen; mahdolliset ympäristön tilat ovat $\{4, 5, 7, 8\}$.

  Tavoite on päästä avaruuden tilaan S, jonka *kaikki* jäsenet ovat siistejä, eli $(\emptyset \subsetneq) S \subseteq \{7, 8\}$.

  Esimerkiksi tekojono **RSLS** saavuttaa tavoitteen pääsemällä avaruuden tilaan $\{7\}$.

  Vertaa epädeterminististen automaatin simulointi!

- Epädeterminististen ympäristö voidaan mallintaa samoin: kaaret ovat silloin muotoa

$$P \xrightarrow{t} \{q : \text{jossakin } p \in P \text{ teosta } t \text{ saattaa seurata } q\}$$

  missä p ja q ovat jälleen ympäristön tiloja. Nyt verkon solmujen voi ajatella edustavan toimijan *tietoa* ympäristöstään.

- Jos ympäristö ei ole saavutettavissa eikä deterministinen, voi ratkaisu jäädä löytymättä:
  - Puuttukoon imurista sensori, joka kertoisi onko *naapuri*ruutu likainen.
  - Vuotakoon imuri siten, että **S** saattaa pudottaa likaa aikaisemmin puhtaseen ruutuun.

  Silloin seuraava tekojono *ei* olekaan ratkaisu:

$$\{1, 3\} \xrightarrow{\mathsf{S}} \{5, 7\} \xrightarrow{\mathsf{R}} \{6, 8\} \xrightarrow{\mathsf{S}} \{6, 8\}$$

  Oikeanpuoleisessa ruudussa saa imuroida *vain jos siinä on likaa*.

  Ympäristöä on nyt havainnoitava löydetyn ehdokastekojonon *suorituksen aikana*, jolloin *yksi etukäteen löydetty* jono ei enää riitäkään varmaksi ratkaisuksi.

Tällaista ongelmaa kutsutaan *varautumisongelmaksi (contingency problem)*: toimijan on varauduttava siihen, että kun hän alkaa toteuttaa valitsemaansa toimintasuunnitelmaa, niin sen kuluessa voi ilmetä jotakin sellaista, mikä vaatii suunnitelman muuttamista.

Valtaosa tosielämän ongelmista on tätä laatua: taksikuksi ei sulje silmiään päätettyään mitä reittiä yrittää viedä matkustajansa perille.

Varautumisongelmia lähestytään kahdella tapaa:

Vuorottelemalla *(interleaving)* suunnitelman etsintää ja toteutusta:
Etsitään toimintasuunnitelma (ainakin) siihen saakka kunnes ympäristöä on taas havainnoitava, ja toteutetaan se.
Sitten havainnoidaan, ja jatketaan tulosten perusteella etsintää.
Hyötynä on, että toimeen päästään tarttumaan vaikkei kaikkea informaatiota olisikaan saatavilla.
Haittana on, että toimija voi valita suunnitelman, joka näyttää aluksi hyvältä, koska sitä ei ole mietitty "loppuun asti".
Tai suunnitelman, joka on arka yllättäville muutoksille ympäristössä.
Tuttu vuorottelun erikoistapaus ovat kahden hengen *pelit*: kumpikin pelaaja miettii useita siirtoja eteenpäin ajatellen "mitä tuo toinen tekisi, jos minä itse tekisin näin?"
Jossakin vaiheessa tehdään siirto — vaikkei miettiminen olisikaan loppunut — ja katsotaan toisen vastaus siihen ennen kuin jatketaan miettimistä muuttuneessa pelitilanteessa.
Hyötynä on, että päästään siirtämään laskematta kaikkia vaihtoehtoja (mikä olisi käytännössä mahdotonta).
Haittana on, että valittu pelistrategia olisi voinut osoittautua kehnoksi jos olisi mietitty "vielä pari siirtoa eteenpäin".

Suunnittelulla *(planning)*, joka tuottaa yhden tekojonon sijasta *havaintojen ehdollistamia jonoperheitä* tai "ohjelmia".
Jos imuri vuotaa mutta tunnistaa lian ruudussaan (havainnoilla $\mathbf{k}/\mathbf{e}$), niin hyvä suunnitelma on havaintoihin sopiva säännöllisen lausekkeen

$$(\mathbf{k}.\mathbf{S})^*.\mathbf{e}.\mathbf{R}.(\mathbf{k}.\mathbf{S})^*$$

mukainen jono. Tai ohjelmana:

while k do
S
end while
R
while k do
S
end while

Usein tällainen ratkaisu löytyy *osittamalla* tavoite välitavoitteiksi: tässä kumpikin **while** saavuttaa välitavoitteen "siivoa nykyinen ruutu". Vertaa silmukkainvariantit!

Hyötynä on, että ennen tekojen valintaa on hankittu "kokonaiskuva" tilanteesta, mikä välttää lyhytnäköiset ratkaisut.

Haittana on, että tämän kokonaiskuvan muodostaminen on vaikeaa, kallista ja tehtävä ennen kuin päästään toimeen.

- Vaikein autonomisen toimijan ympäristö on sellainen, jossa se ei tiedä etukäteen edes omien tekojensa vaikutuksia. Silloin sen on *kokeiltava* "omia voimiaan" ja *opittava* näin erottamaan toisistaan ne ympäristön muutokset, jotka johtuvat sen teoista, ja ne jotka tapahtuvat siitä huolimatta. Nämä ongelmat [RN95, luku 20] ohitetaan tällä kurssilla.

Määritellään jatkossa tarkasteltavat *etsintäongelmat (search problem)* koostumaan seuraavista osista:

Alkutila *(initial state)* on se etsinnän tila, jossa etsintää suorittava toimija tietää nyt olevansa.

Etsijä ei välttämättä tiedä ympäristön tilaa; voihan kyseessä olla monitilainen etsintäongelma.

Operaattorit *(operator)* joilla mallinnetaan ne *teot*, jotka kussakin etsinnän tilassa ovat etsijälle mahdollisia.

Operaattori kuvaa kunkin teon t kertomalla, mihin etsinnän tilaan Q_t päästään tekemällä t etsinnän nykytilassa P.

Voidaan olettaa, että kullekin P ja t on enintään yksi kohdetila Q_t: muuten kyseessä onkin epädeterministinen ympäristö, joten vaihdetaan

monitilaiseen etsintäongelmaan ja kerätään jokainen vaihtoehtoinen Q_t yhteen kuvan 14 alapuolen tapaan.

Joskus tämä etsintäongelman osa muotoillaan **seuraajafunktio**na *(successor function)*

$$S(P) = \{\text{tila } Q_t \colon Q_t \text{ saadaan tilasta } P \text{ jollain teolla } t\} .$$

Etsintäavaruus *(search/state space)* on alkutilan ja operaattoreiden määräämä (mahdollisesti ääretön) suunnattu verkko, jonka

solmut ovat ne etsinnän tilat, joihin päästään alkutilasta jollakin jonolla tekoja, ja

kaaret ovat operaattorit $P \xrightarrow{t} Q_t$ kaikilla solmuilla P ja teoilla t

Esimerkkeinä olkoot kuvan 14 kaksi verkkoa.

Maaliehto *(goal test)* kertoo kustakin etsinnän tilasta, onko tavoite siinä saavutettu.

Etsintäongelman **ratkaisu** *(solution)* on mikä tahansa polku (tai sitä vastaava tekojono) sen alkutilasta johonkin maaliehdon täyttämään solmuun.

Polkuhinta *(path cost)* on funktio, joka laskee, paljonko annetun polun (= tekojonon) suorittaminen toimijalle "maksaisi" resursseina.

Tällä kurssilla koko polun

$$\mathcal{P} = Q_0 \xrightarrow{t_1} Q_1 \xrightarrow{t_2} Q_2 \xrightarrow{t_3} Q_3 \xrightarrow{t_4} \cdots \xrightarrow{t_n} Q_n$$

$$\mathrm{hinta}(\mathcal{P}) = \sum_{1 \le i \le n} \mathrm{hinta}(t_i)$$

on sen kunkin teon $\mathrm{hinta}(t_i) > 0$ summa.

Kaikkiin ongelmiin ei liity hintoja: silloin valitaan $\mathrm{hinta}(t) \equiv 1$ ja toivotaan lyhyttä ratkaisua.

Etsintäongelmien ratkaisumenetelmää M voidaan arvioida (ainakin) neljällä tavalla:

- Löytääkö M ratkaisun aina kun sellainen on olemassa?

Silloin M on *täydellinen (complete)*.

Vai voiko M epähuomiossa sivuuttaa ratkaisun

- ja pysähtyä vastaukseen "ei ole"
- tai olla pysähtymättä.

- Löytääkö M aina hinnaltaan halvimman ratkaisun *(online cost)*?

  Silloin M on *optimaalinen (optimal)* eli sen ehdottama tekojono on toimijalle "vähiten rasittava".

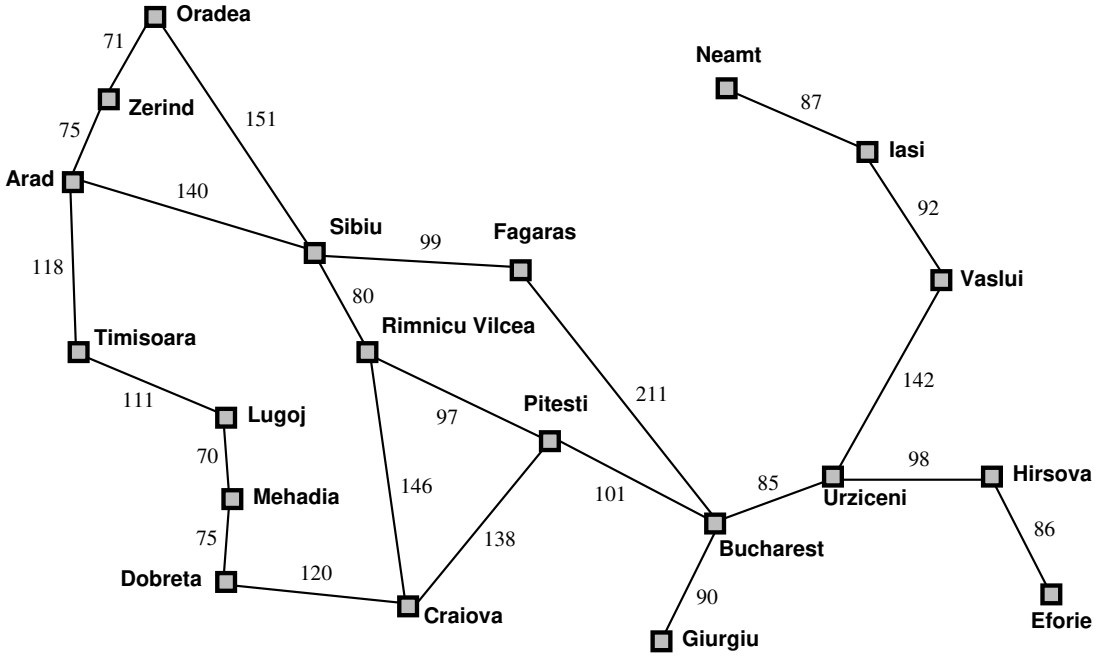
  Tai kuinka lähelle parasta ratkaisua M (yleensä) pääsee?
- Kuinka paljon *etsintä*resursseja ($=$ muistia ja prosessoriaikaa) M kuluttaa *(offline cost)*?

  Eli paljonko toimija käyttää voimiaan "miettimiseen"?
- Kuinka paljon M kuluttaa

$$\textit{kokonais}\text{resursseja} = \text{offline cost} + \text{online cost}$$

(mahdollisesti painotettuna löydetun ratkaisun käyttökertojen lukumäärällä).

Tarkastellaan ongelmaa valita ajoreitti Aradista Bukarestiin.



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

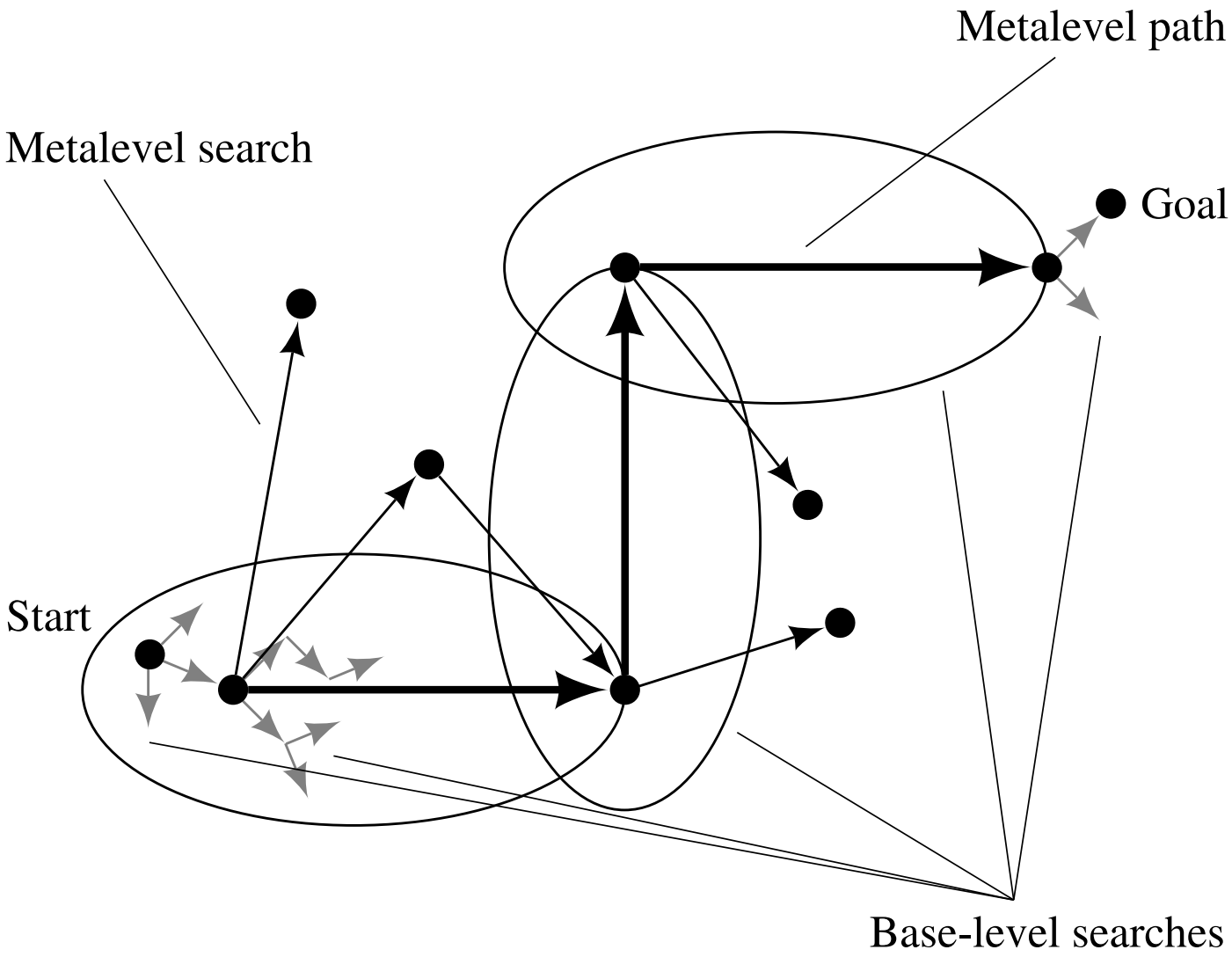
Kuva 15: Osa Romanian tieverkkoa. [RN95, kuva 4.2].

Alkutila on se, jossa olemme Aradissa.

Operaattorit valitaan viemään meidät kaupungista johonkin naapurikaupunkiin.

Tällä tasolla *pelkistämme (abstract)* ongelman kuvauksesta pois tarpeettomat ympäristön piirteet.

Pelkistämme pois myös tarpeelliset mutta liian yksityiskohtaiset piirteet: millaiset ohjaustoimenpiteet tarvitaan auton saamiseksi tästä parkkiruudusta kadulle, kadulta moottoritielle,. . . Ne voidaan selvittää *alemman pelkistystason* etsintäongelmana, kunhan ensin päätetään, mihin naapurikaupunkiin seuraavaksi suuntaamme. Tämä on *hierarkista* suunnittelua.



© 1998 Morgan Kaufman Publishers

Kuva 16: Hierarkinen etsintä [Nil98, kuva 10.3].

Onnistunut abstraktio on välttämätöntä (muttei riittävää) onnistunee-

seen etsintään!

Ja vaikeaa: pitäisikö esimerkiksi ottaa mukaan autokorjaamojen määrä kullakin reitillä — varmuuden vuoksi?

Maaliehto on, että olemme Bukarestissa.

Polkuhinta valitaan kuvan 15 informaation perusteella lyhimmäksi kokonaismatkaksi.

Ratkotaan tällaisia etsintäongelmia seuraavalla yleisellä algoritmi(skeema)lla:

```
function GENERAL-SEARCH( problem, QUEUING-FN) returns a solution, or failure

  nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
  loop do
    if nodes is empty then return failure
    node ← REMOVE-FRONT(nodes)
    if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
    nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
  end
```

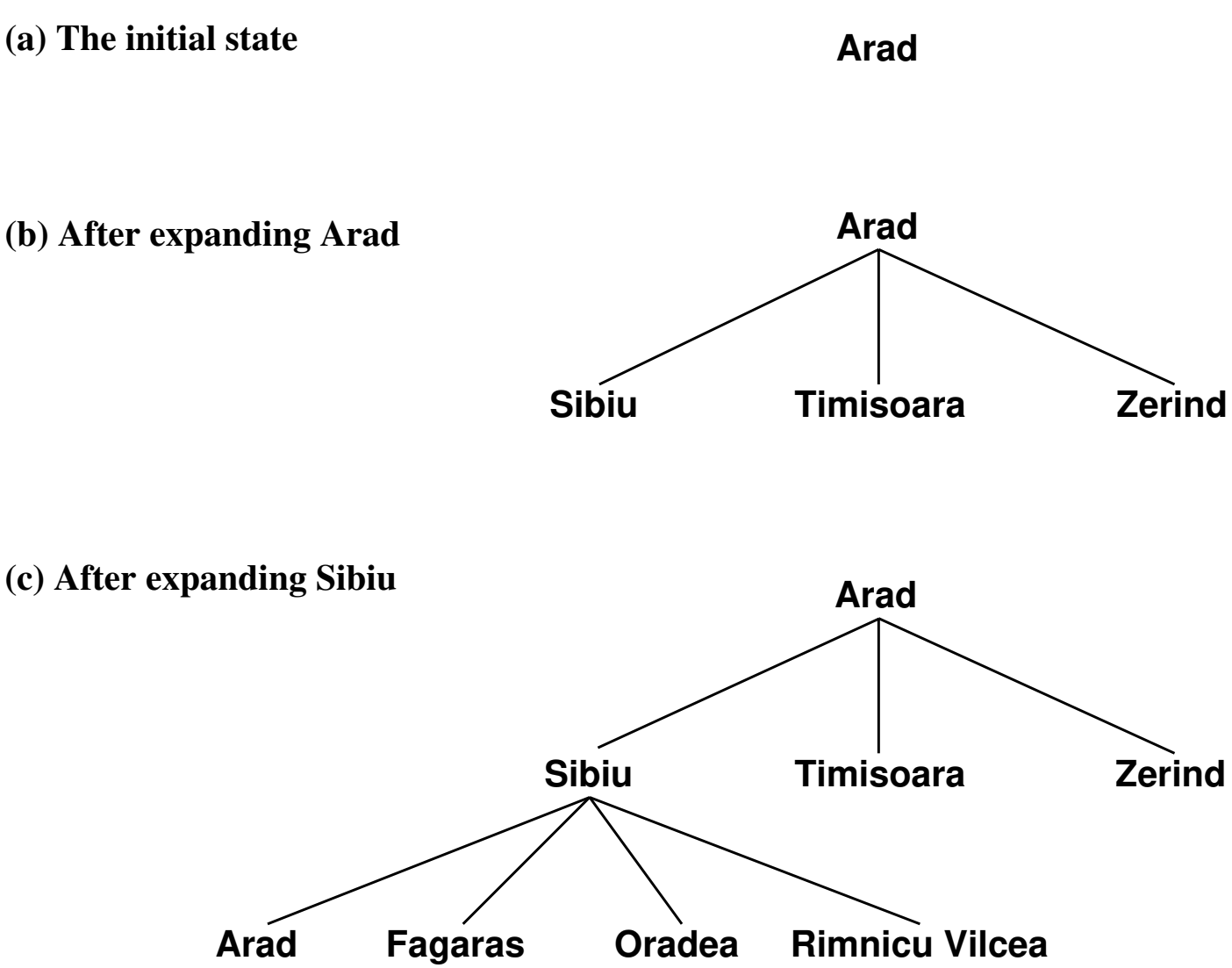
Algoritmi 2: Yleinen etsintäalgoritmi [RN95, kuva 3.10].

Parametrina saatu aliohjelma QUEUEING-FN tallettaa *työmuistiin* nimeltä $nodes$ etsinnän eri vaihtoehtoja, joista REMOVE-FRONT poimii aina seuraavan tarkasteltavaksi.

Työmuistin ominaisuuksien valinnalla päätetään *etsintästrategia (search stratgy)* eli se järjestys, jossa eri vaihtoehtoja harkitaan.

Sovelletaan etsintäalgoritmia 2 ongelmaamme:

1. Aloitamme Aradista (kuva 17a), joka ei ole Bukarest, joten se on *laajennettava (expand)* naapureikseen (kuva 17b).
2. Strategiamme valitkoon naapurikaupungeista Sibiun kokeiltavaksemme (kuva 17c).
3. Jatkamme siitä kuten askeleessa 1, jne.



Kuva 17: Etsintäpuun kasvu. [RN95, kuva 3.8].

Strategian jäljittäminen tuottaa *etsintäpuun (search tree)*:

juuri on alkutila

solmut vastaavat etsintäavaruuden *polkuja*

lapset saadaan laajentamalla isäsolmun polkua yhdellä teolla

lehdistä valitaan yksi kullakin kierroksella laajennettavaksi strategian määräämässä järjestyksessä.

- Etsintäpuu kertoo, miten annettu etsintäalgoritmi käyttäytyy annetussa etsintäavaruudessa.

  Se on rekursiokutsupuun yleistys tapaukseen, jossa vaihtoehtojen käsittelyjärjestys seuraa työmuistin toiminnasta (ohjelmakoodin sijaan).

- Sen ei tarvitse olla algoritmia suorittavan tietokoneen muistissa:
  - jo käsiteltyä ei ole pakko muistaa
    (vaikkapa jos sen mahdollinen uudelleen käsittely ei haittaa)
  - vielä käsittelemätöntä saa unohtaa
    (vaikkapa jos se voidaan palauttaa takaisin käsiteltäväksi myöhemmin).
- Se voi olla *ääretön*, jos
  - avaruus on ääretön
    (jo ympäristö voi olla ääretön)
  - avaruudessa on kehä.

Jatkossa tarkastellaan erilaisia etsintäalgoritmeja antamalla vastaava QUEUEING-FN ja tarkastelemalla näin syntyviä etsintäpuita.

3.1. Systemaattisia etsintämenetelmiä [RN95, luku 3] [Nil98, luku 8]

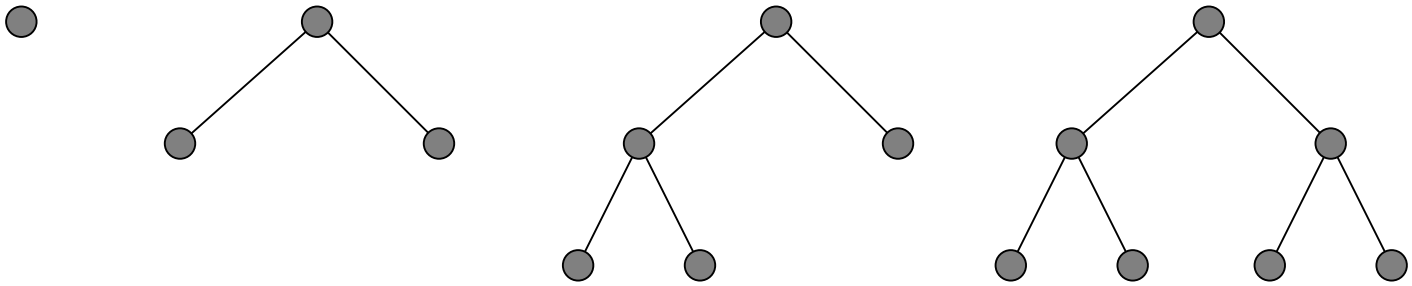
> "Ihaa, joka on ystäväni, on kadottanut häntänsä. Ja hän on ruvennut Murjottamaan. Voisitko ystävällisesti neuvoa minulle miten löytäisin sen?"
>
> "Hmm", sanoi Pöllö. "Tällaisissa tapauksissa Tavanomainen Proseduuri lienee seuraava."
>
> "Mitä tarkoittaa Tavanomainen Proosakuuri?" kysyi Puh. "Sillä minä olen Pieniälyinen Karhu jolle kummalliset sanat ovat Harmi."
>
> "Se tarkoittaa Mitä Tehdään."
>
> "Hyvä jos se tarkoittaa sitä", sanoi Puh nöyrästi.
>
> "Mitä Tehdään lienee seuraava: täytyy julistaa systemaattiset etsinnät. Sitten —"
>
> "Hetki vain", sanoi Puh käpälä pystyssä. "*Mitä* täytyy tehdä? Aivastelit juuri kun olit sanomaisillasi sen."
>
> *A.A. Milne: Nalle Puh*

Etsintämenetelmä on systemaattinen, jos se käy etsintäpuuta läpi jossakin systemaattisessa järjestyksessä hyödyntämättä mitään tietoa itse etsintäongelmasta. Sellaista etsintää kutsutaan myös *sokeaksi (uninformed/blind search*.

Vaikka systemaattinen etsintä onkin tehotonta, niin joskus ongelmastakaan ei tiedetä mitään.

3.1.1. Leveyssuuntainen etsintä (Breadth-First Search)

Jos Algoritmin 2 QUEUEING-FN on *jono (queue)*, niin saadaan (tunnetusti) etsintäpuun leveyssuuntainen läpikäynti:



Kuva 18: Leveyssuuntainen etsintä [RN95, kuva 3.11].

- Se on *täydellinen*.
- Se on myös *optimaalinen, jos* polkuhinta on solmun syvyyden ei-vähenevä funktio.

  Näin on kun kaikki operaattorit maksavat yhtä paljon.
- Ongelmana on *eksponentiaalinen (ajan ja) tilan tarve*:
  - Olkoon $b > 1$ avaruuden solmujen (suurin) *haarautumisaste (branching factor)* (= naapurien lukumäärä).
  - Olkoon lyhimmän ratkaisun (= polun) pituus eli avaruuden *halkaisija (diameter)* d tekoa.
  - Läpikäydyn puun koko (= laajennusten määrä) on silloin enintään

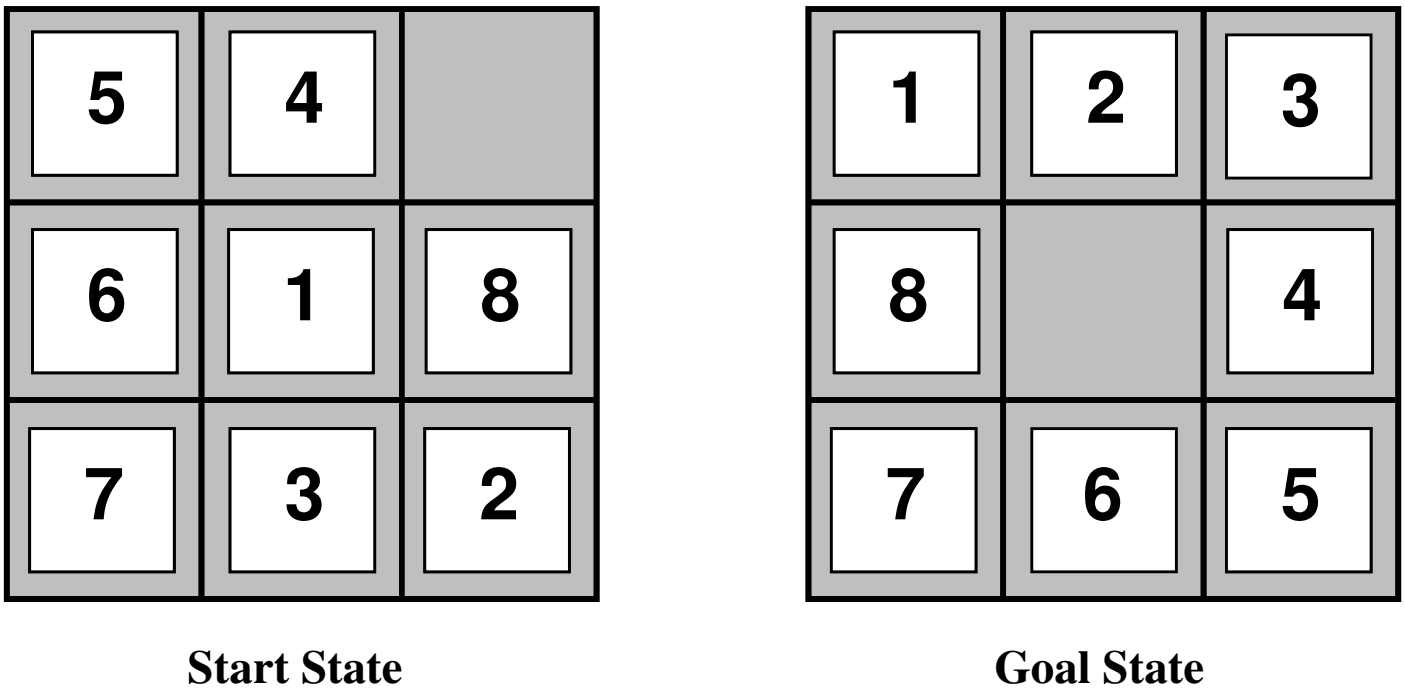
$$T_d = \sum_{0 \leq k \leq d} b^k = \frac{b^{d+1} - 1}{b - 1} \tag{7}$$

  solmua, joista enintään b^d on kerrallaan jonossa. (Geometrinen sarja [Myr74b, luku 11.3].)

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Kuva 19: Leveysetsinnän resurssit ($b = 10$) [RN95, kuva 3.12].

Eräs tekoälyn (opetuksen) suosittu leluongelma on lasten liukupeli:

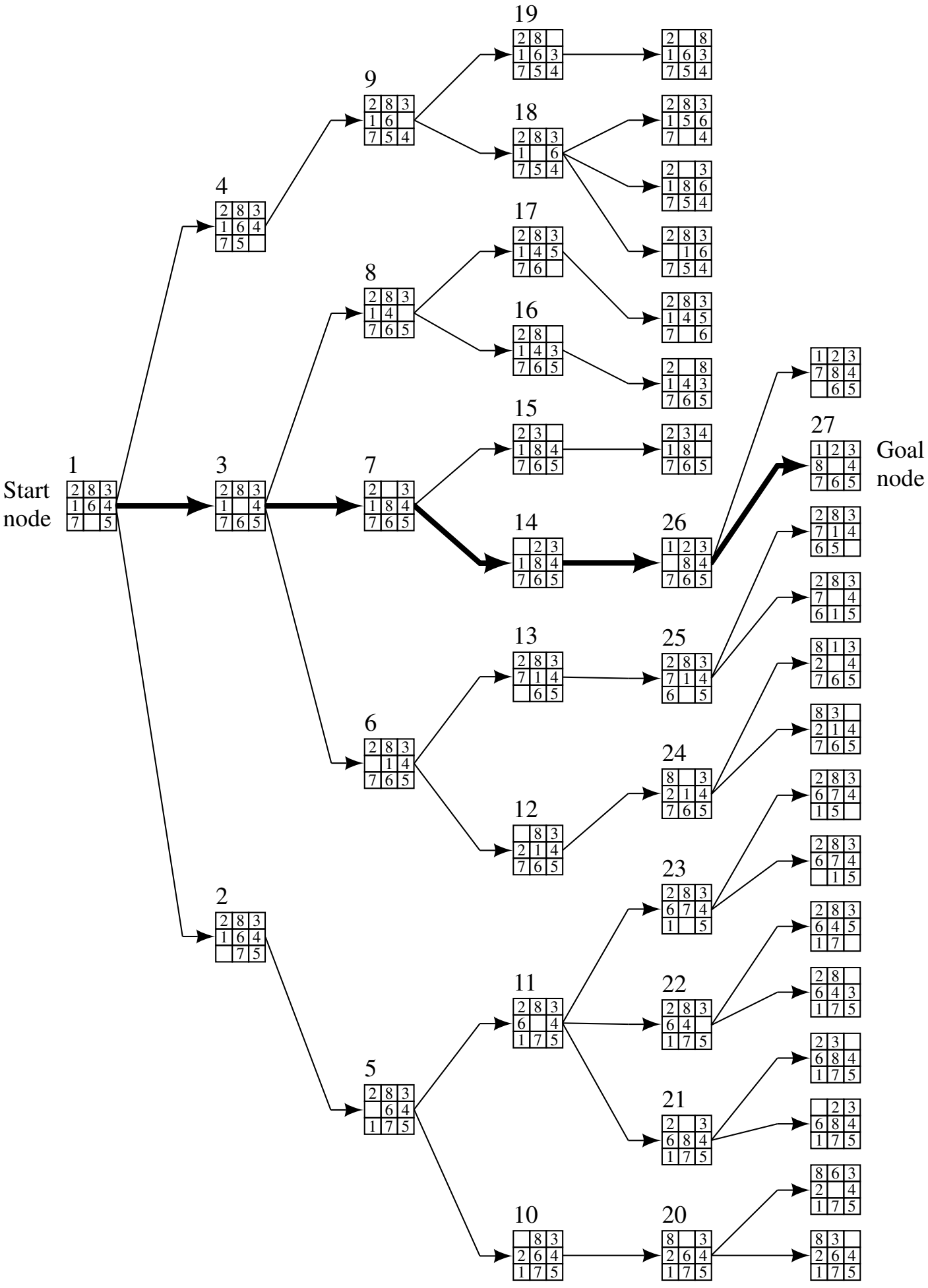


Kuva 20: 8-liukupeli [RN95, kuva 4.7].

Esimerkiksi kuvan 20 vasemmassa tilanteessa sen oikean yläkulman tyhjään ruutuun voi (yhtenä tekona) liu'uttaa

- joko vasemmanpuolisen naapurin 4
- tai alapuolisen naapurin 8

ja tyhjä paikka siirtyy sen tilalle.



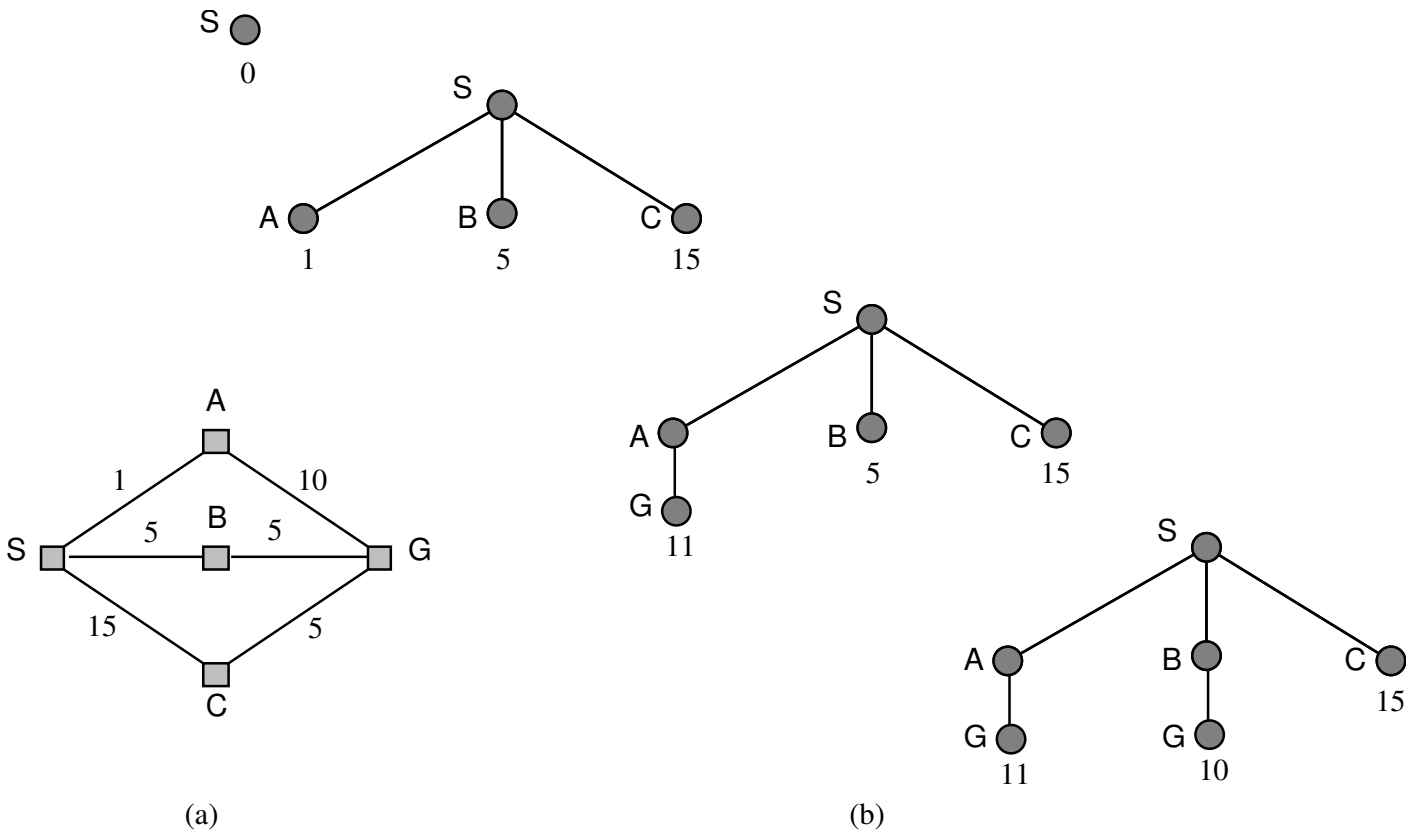
© 1998 Morgan Kaufman Publishers

Kuva 21: Leveysetsintä 8-liukupelissä [Nil98, kuva 8.2].

3.1.2. Halvin ensin (Uniform Cost Search)

Luvun 3.1.1 algoritmi löytää sellaisen ratkaisun, jossa on *vähiten tekoja*, mutta jos eri teot ovat eri hintaisia ei se ole välttämättä *paras* valinta.

Algoritmin 2 QUEUEING-FN olkoon siis *keko (heap)* kuhunkin solmuun liittyvän polun (= tekojonon) kokonaiskustannuksen suhteen, joista valitaan aina pienin. Etsitään vaikkapa reitti solmusta S solmuun G:



Kuva 22: Halvin reitti ensin [RN95, kuva 3.13].

Tämä etsintäalgoritmi on *täydellinen ja optimaalinen, jos* polun kokonaishinta ei koskaan laske sitä myöten edettäessä.

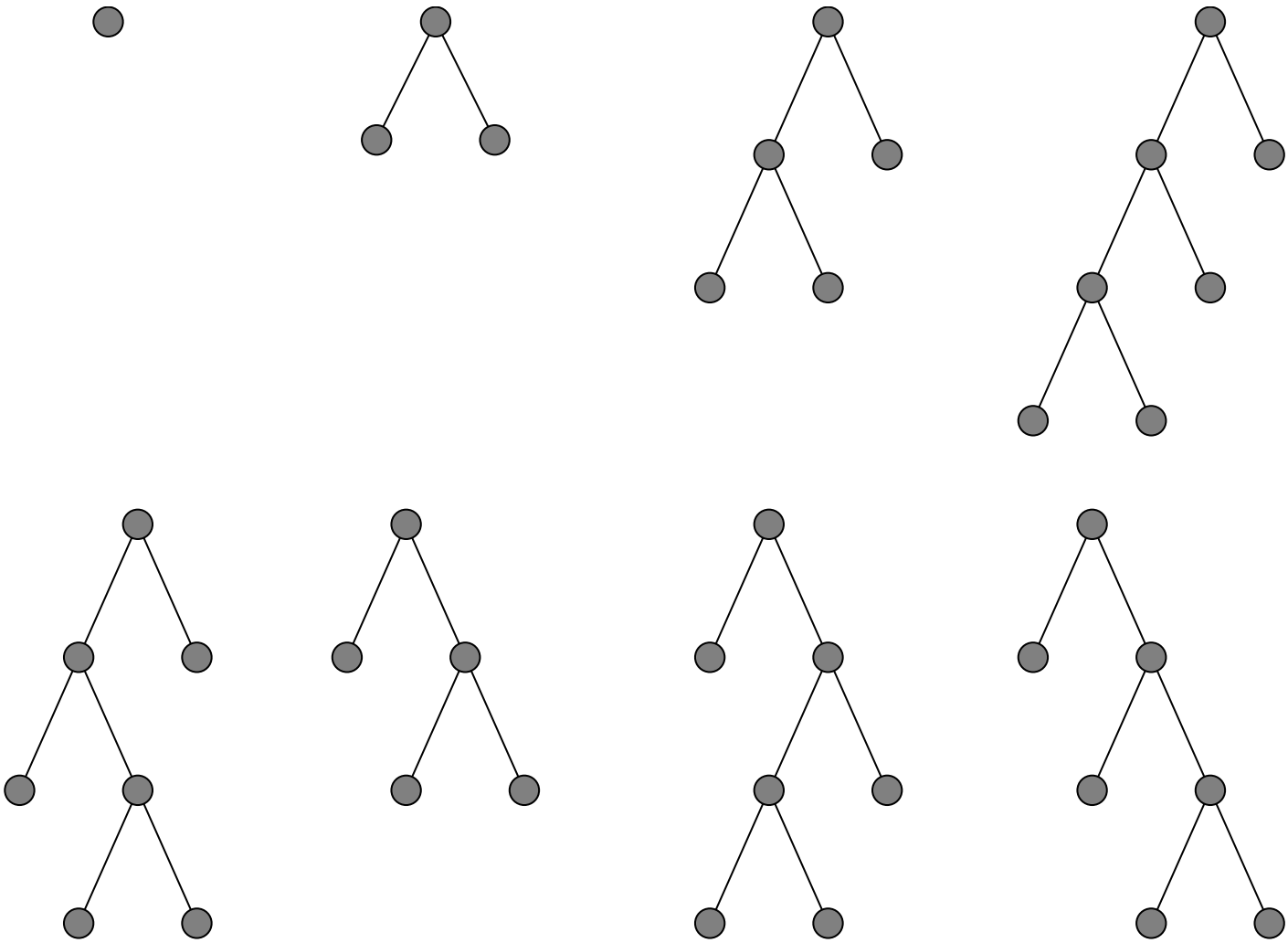
Luentojemme summautuvilla polkuhinnoilla tämä ehto on

$$\text{hinta}(t) \geq 0$$

eli että mikään teko t ei tuota "voittoa" tekijälleen. Siis se on voimassa.

3.1.3. Syvyyssuuntainen etsintä (Depth-First Search)

Jos Algoritmin 2 QUEUEING-FN on *pino (stack)*, niin saadaan (tunnetusti) etsintäpuun syvyysyyntainen läpikäynti:



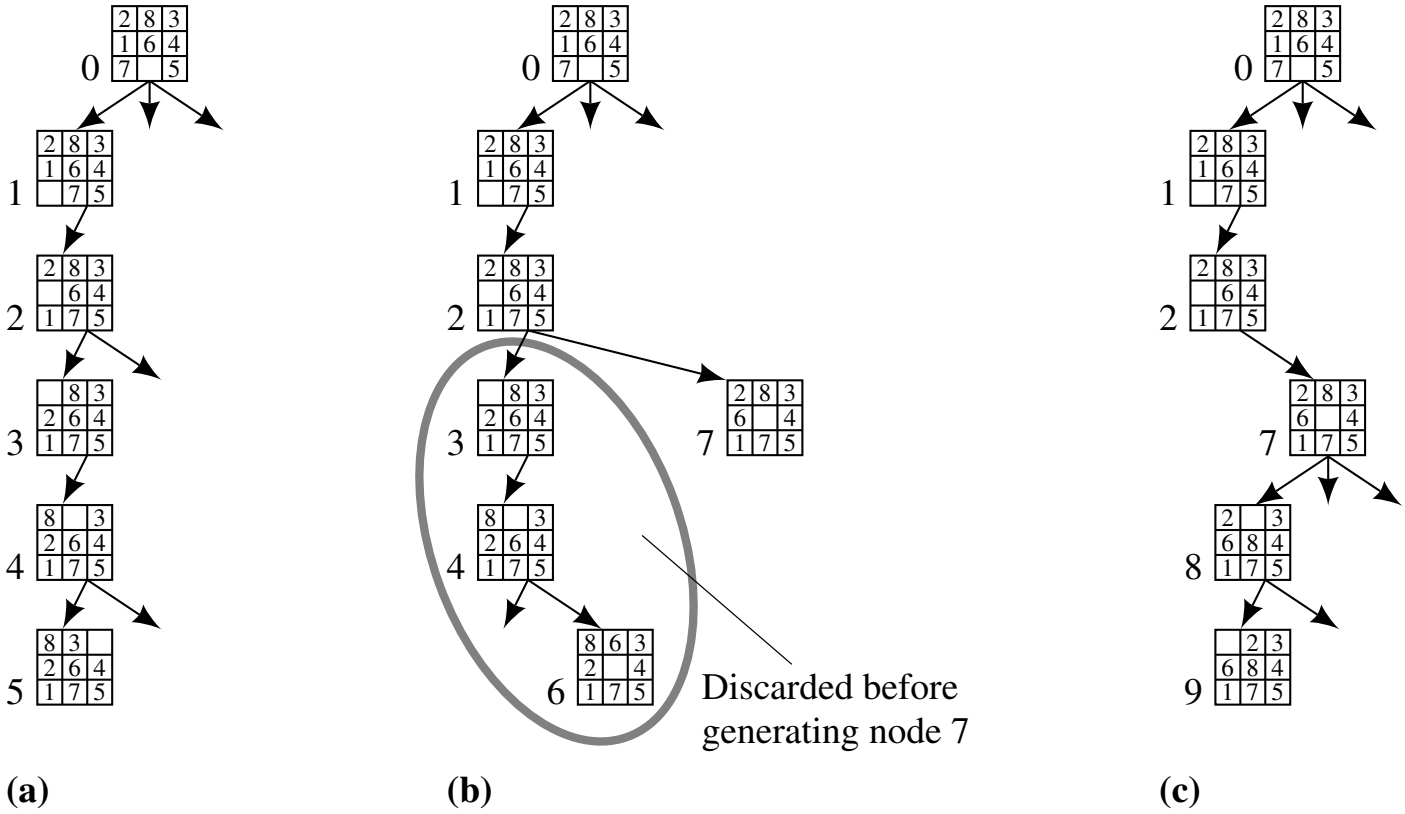
Kuva 23: Syvyyssuuntainen etsintä [RN95, kuva 3.14].

Tämä etsintäalgoritmi tarvitsee *paljon vähemmän muistia* kuin luvun 3.1.1 menetelmä: jos m on etsintäavaruuden pisimmän tekojonon pituus (= etsintäpuun syvyys), niin pinossa on kerrallaan vain $O(bm)$ solmua.

Tämä algoritmi vie pahimmillaan $O(b^m)$ etsintäaskelta, mutta jos avaruudessa on monta maaliehdon täyttävää solmua, se saattaa löytää yhden käymättä läpi kuin murto-osan koko avaruudesta.

Tämä algoritmi *ei ole optimaalinen*, eihän se edes pidä kirjaa eri polkujen kustannuksista.

Tämä algoritmi *ei edes pysähdy, jos* m on ääretön: se seuraa ensimmäistä tapaamaansa etsintäpuun ääretöntä haaraa yhä syvemmälle ja syvemmälle.



© 1998 Morgan Kaufman Publishers

Kuva 24: Syvyyssuuntainen etsintä 8-puzzlessa [RN95, kuva 8.3].

Jos etsintäongelman ratkaisijan ohjelmoi *rekursiivisena* niin päätyy johonkin syvyyshaun varianttiin: rekursiokutsupino toimii silloin (näkymättömänä) hakupinona.

Ja kääntäen: jos valitsee menetelmäkseen syvyyshaun, niin sen voi usein *toteuttaa helposti* rekursiolla.

3.1.4. Syvyysrajoitettu etsintä (Depth-Limited Search)

Yksi tapa saada luvun 3.1.3 algoritmi varmasti pysähtymään on lisätä siihen etukäteen yläraja l tarkasteltavien polkujen *pituudelle* (eli seurattavien haarojen syvyydelle): jos nykyinen solmu ylittää rajan, niin sitä ei enää laajennetakaan lapsikseen.

- Tämä syvyysrajoitettu etsintä ei edelleenkään ole optimaalinen.
- Jos $l \geq d$, niin se on täydellinen, muuten taas ei.
- Se vie $O\left(b^l\right)$ etsintäaskelta.
- Se pitää pinossaan korkeintaan $O(bl)$ solmua kerrallaan.

Samaa ideaa voi soveltaa myös antamalla yläraja ℓ polkujen *hinnalle*:

- Pysähtyminen on taattu jos ei ole ääretöntä polkua jonka hinta pysyisi silti äärellisenä.
  Luentojemme summatuvilla polkuhinnoilla tämä tarkoittaa

$$\mathrm{hinta}(t) \geq \delta$$

  missä $\delta > 0$ on kaikkien tekojen hintojen alaraja.

3.1.5. Iteratiivinen syventäminen (Iterative Deepening)

Luvun 3.1.4 algoritmin soveltamisessa on vaikeaa löytää tarpeeksi muttei liian suuri yläraja l. Jätetään siis tämäkin etsintäalgoritmin huoleksi:

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution sequence
   inputs: problem, a problem

   for depth ← 0 to ∞ do
     if DEPTH-LIMITED-SEARCH( problem, depth) succeeds then return its result
   end
   return failure
```

Algoritmi 3: Iteratiivinen syventäminen [RN95, kuva 3.15].

Tämä algoritmi yhdistää lukujen 3.1.1 ja 3.1.3 hyvät puolet:

- Se on *täydellinen* ja *optimaalinen* kuten luvun 3.1.1 leveyssuuntainen etsintäkin.
- Sen muistintarve on yhtä vaatimaton kuin käyttämällään luvun 3.1.3 syvyyssuuntaisella etsinnälläkin.
- Se tekee tosin useampia etsintäaskeleita kuin luvun 3.1.1 leveyssuuntainen etsintä, koska sama etsintäpuun solmu *laajennetaan monta kertaa*:
  1. Ensin käydään läpi koko korkeuden 0 puu löytämättä vastausta. Siihen kuuluu $T_0 = 1$ solmua (eli vain juuri) yhtälön (7) nojalla.
  2. Sitten läpikäydään korkeuden 1 puun

$$T_1 = \frac{b^2 - 1}{b - 1} = b + 1$$

     solmua, joista 1 (pelkkä juuri) tutkittiin (muttei vielä laajennettu) jo kerran vaiheessa 1.
  3. Sitten korkeuden 2 puu, jossa juuri käsitellään kolmannen kerran, sen lapset toistamiseen, ja lapsenlapset ensimmäistä kertaa.

Kun näin jatketaan ratkaisusyvyyteen $depth = d$ saakka, on käsiteltyjen solmujen kokonaismäärä

$$\begin{aligned}\sum_{0\le k\le d} T_k &= \sum_{0\le k\le d} \frac{b^{k+1}-1}{b-1} \\ &= \frac{1}{b-1}\left(\left(\sum_{0\le k\le d} b^{k+1}\right) - \left(\sum_{0\le k\le d} 1\right)\right) \\ &= \frac{1}{b-1}\left(b\left(\sum_{0\le k\le d} b^{k}\right) - (d+1)\right) \\ &= \frac{bT_d - (d+1)}{b-1}. \end{aligned} \tag{8}$$

- Toisaalta yhtälöstä (8) nähdään, että tämä algoritmi on vain noin

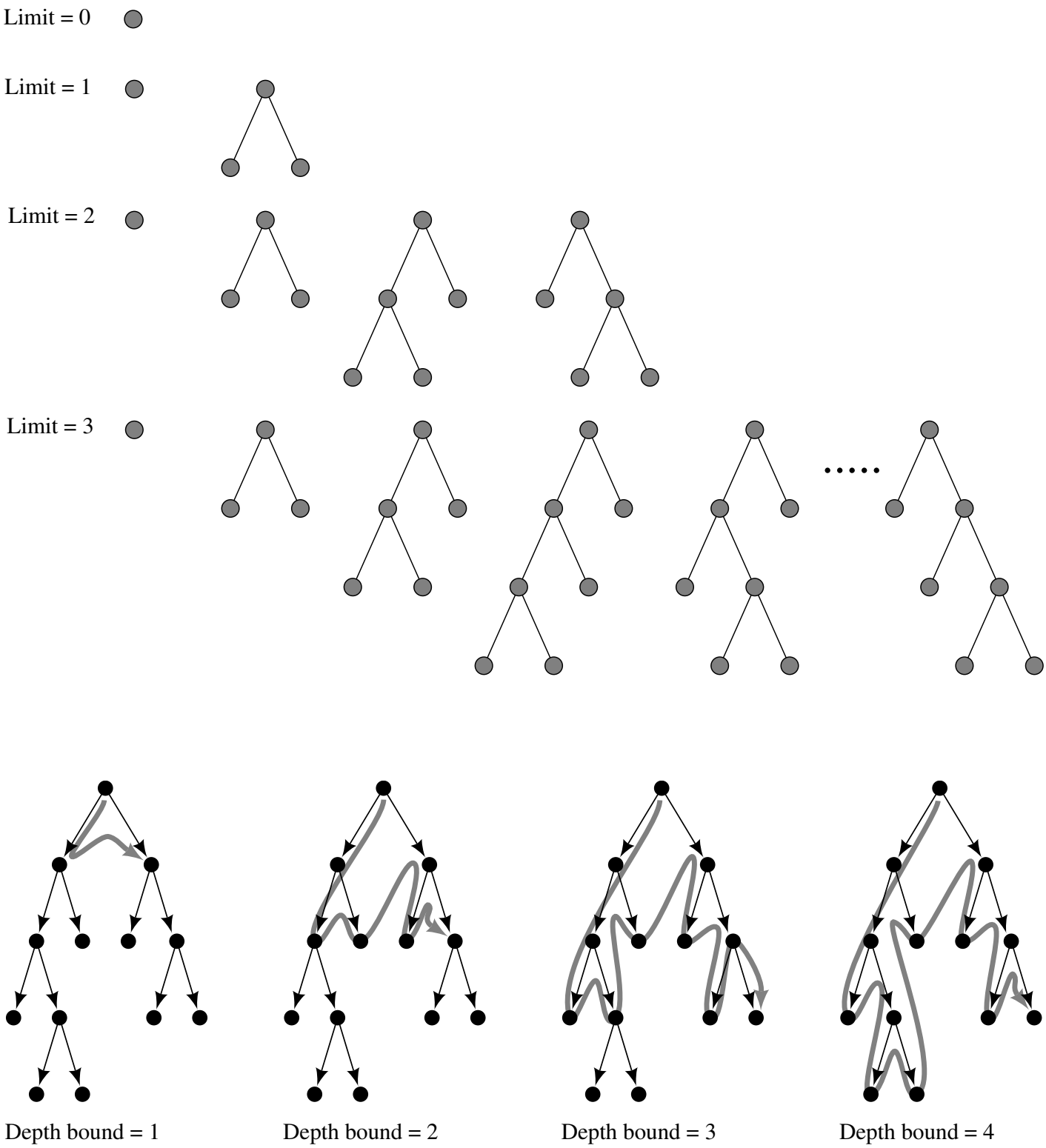
$$\frac{b}{b-1}$$

kertaa hitaampi kuin luvun 3.1.1 leveyssuuntainen etsintä:
  - Jos $b = 2$ niin se vie noin kaksinkertaisen ajan, mutta ero kutistuu kun b kasvaa.
  - Kun $b = 10$ ja $d = 5$ niin yhtälö (7) antaa $111\ 111$, ja yhtälö (8) antaa $123\ 456$, jolloin eroa on enää noin 11%.
  - Yleisestikin *suurin osa etsintäpuun solmuista on lehtiä*: jo binääripuissa on aina lehtiä 1 enemmän kuin sisäsolmuja, ja tämä ero vain kasvaa haarautumisasteen b mukana.
  - *Sisä*solmujen käsitteleminen moneen kertaan ei siis vaikuta merkittävästi kokonaistyömäärään.

Iteratiivinen syventäminen onkin paras systemaattinen etsintämenetelmä suurille avaruuksille, joiden halkaisijaa d ei tunneta etukäteen.

Jos samaa ideaa käytetään hintarajan ℓ määräämiseen, niin rajaa voi kasvattaa

- joko systemaattisesti hinta-alarajan δ verran
- tai dynaamisesti pienimmäksi pituudeksi jolla jokin tavattu solmu jäi lapsettomaksi.

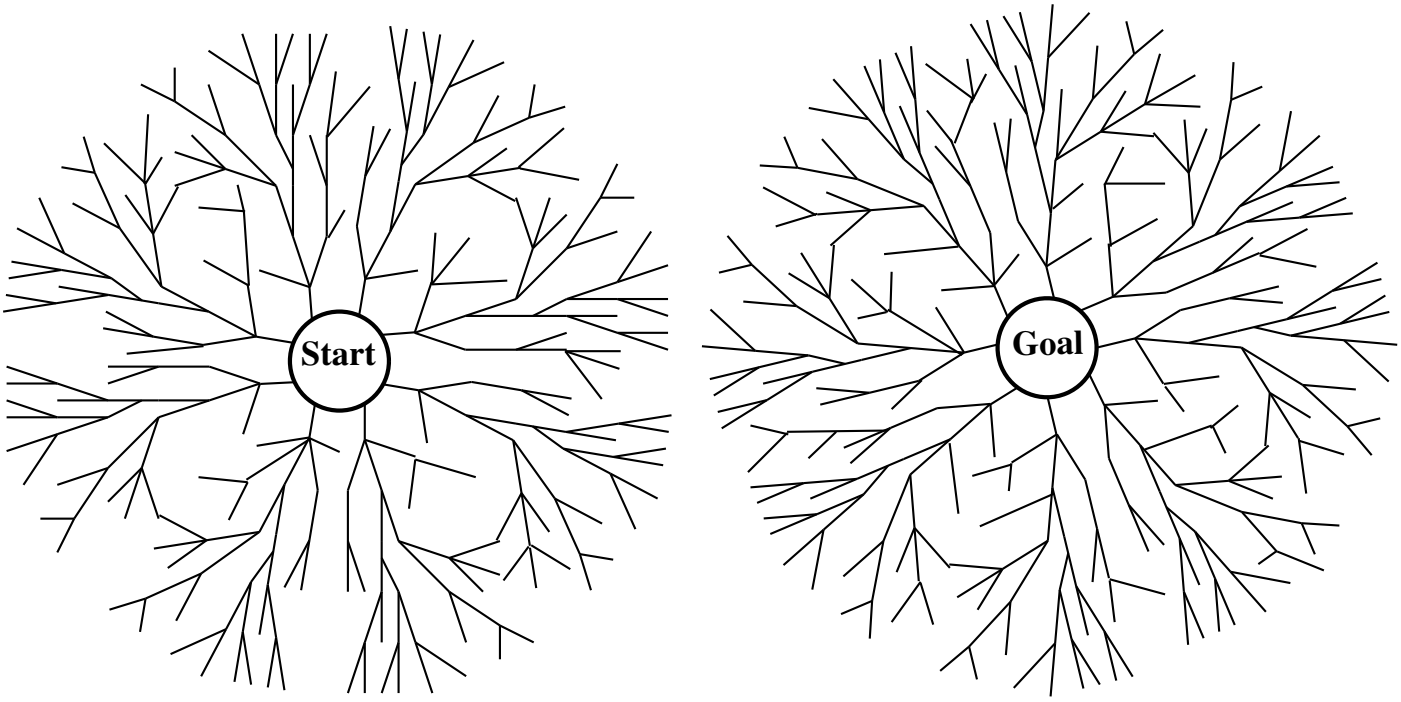


© 1998 Morgan Kaufman Publishers

Kuva 25: Syventämisen eteneminen [RN95, kuva 3.16][Nil98, kuva 8.5].

3.1.6. Kaksisuuntainen etsintä (Bidirectional Search)

Entä jos käynnistetään yhtä aikaa yksi etsintä *eteenpäin alkutilasta* ja toinen etsintä *taaksepäin maalitilasta*, ja lopetetaan niiden kohdatessa?



Kuva 26: Kaksisuuntainen etsintä [RN95, kuva 3.17].

Periaatteessa voimme saavuttaa ratkaisun $O(2b^{d/2})$ askeleessa, koska kumpikin puolisko tulee toista puoliväliin vastaan.

Kun $b = 10$ ja $d = 6$, niin yhtälön (7) mukaan syvyyshaku laajentaa $1\ 111\ 111$ solmua, tämä algoritmi taas vain $2\ 222$.

Tekemällä 2 pienempää puuta karsimme lehtien määrää; luvun 3.1.5 havaiinon nojalla tämä kannattaa.

Ensin on kuitenkin ratkaistava seuraavat pulmat:

- Mitä tarkoittaa "etsiä taaksepäin"?
  Intuitiivisesti seurataan etsintäavaruuden nuolia takaperin.
- Onko siis jokaisella operaattorikaarellamme

$$P \xrightarrow{t} Q$$

vastakaari

$$P \xleftarrow{t^{-1}} Q$$

joka palauttaa tilanteen sellaiseksi, joka vallitsi ennen tekoa t?

Silloin nuolien seuraaminen takaperin voidaan tulkita etsinnäksi eteenpäin käyttäen vastatekoja t^{-1}.

Muuten voi olla vaikea määritellä miten nuolien seuraaminen takaperin oikein muuttaakaan etsinnän tilaa.

Esimerkiksi kuvan 15 kaltaisissa reititysongelmissa teko on peruutettavissa, ellei se kulje yksisuuntaista tietä.

- Entä jos maalitiloja on monta?

  Jos ne on annettu *eksplisiittisenä* äärellisenä listana, niin voidaan
  - etsiä taaksepäin ikään kuin se ongelma olisi monitilainen, tai
  - aloittaa yksi taaksepäin etsintä kustakin maalitilasta.

  Mutta jos onkin annettu *implisiittinen* maaliehdon kuvaus, niin pitäisi jotenkin järkeillä millaisista etsinnän tiloista saattaa seurata jokin maalitila. Tämä voi olla vaikeaa.

- Miten havaitaan, että etsinnät ovat kohdanneet?

  Aikaraja $O(b^{d/2})$ edellytti, että kohtaaminen voidaan tunnistaa vakioajassa. Tähän pyritään käytännössä usein *hajautustaululla (hash table)*.

- On valittava (ei välttämättä yksi ja sama) etsintäalgoritmi kummallekin suunnalle.

  Jos ainakin toinen näistä on luvun 3.1.1 leveyssuuntainen etsintä, niin kohtaamisen havaitseminen helpottuu.

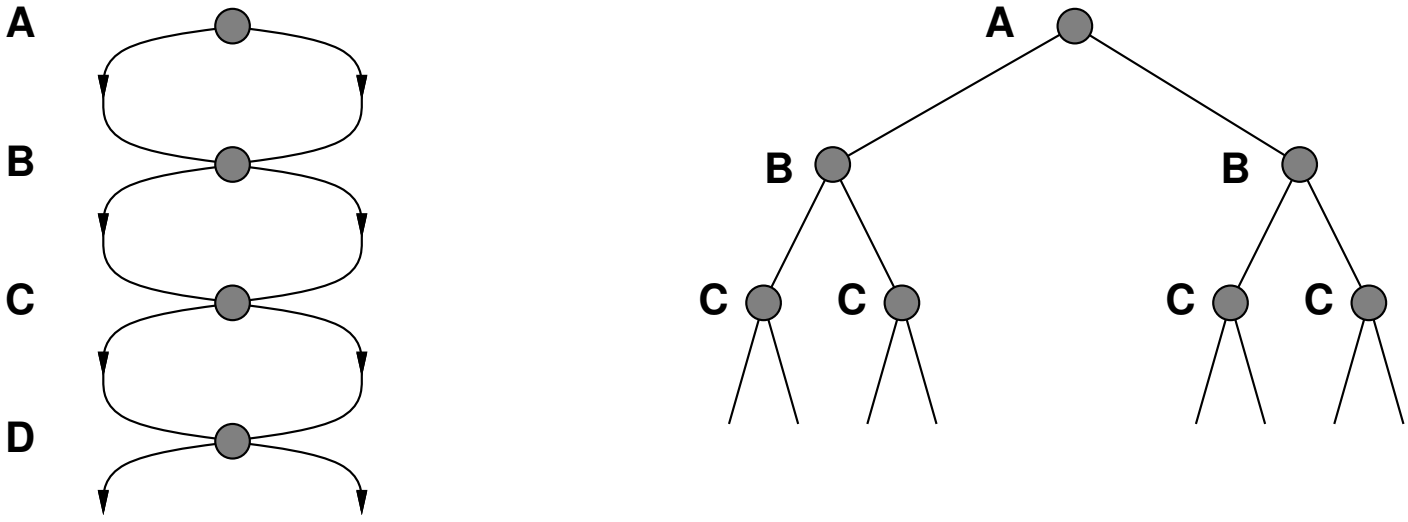
Verrataan lopuksi esiteltyjen algoritmien "strategisia mittoja":

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Kuva 27: Systemaattisten etsintöjen vertailu [RN95, kuva 3.18].

3.1.7. Toistuvien tilojen eliminoimisesta

Etsinnän kuluessa saatamme löytää toistamiseen saman tilan, emmekä tietenkään halua tuhlata resursseja käsittelemällä sitä — ja etenkään siitä alkavaa koko alipuuta — uudelleen:



Kuva 28: Eksponentiaalinen eliminointi [RN95, kuva 3.19].

On ongelmia, joiden huolellinen määrittely ehkäisee tilojen toistumisen jo ennalta.

Toisaalta vaikkapa peruutettaville teoille näin ei ole.

Etsintäalgoritmia voidaan muuttaa siten, ettei se enää laajennakaan mitään sellaista etsinnän tilaa, joka on jo esiintynyt. . .

1. . . . sillä polulla, joka vie etsintäpuun nyt laajennettavasta lehtisolmusta sen juureen.

   Tämä polku on silloin pidettävä muistissa.

   Jokaista tilaa laajennettaessa sen polku on ensin käytävä läpi.

   Jos algoritmina on variantti luvun 3.1.3 syvyyssuuntaisesta etsinnästä, niin tämä polku löytyy pinosta.
2. . . . milloin tahansa etsinnän kuluessa.

   Muistintarve on $O(\text{avaruuden koko})$.

Hakurakenteena käytetään usein hajautusta.

Silti tämä vaihtoehto saattaa olla raskas.

Se on kuitenkin aggressiivisempi toiston torjuja kuin tapa 1.

Toisaalta eri poluilla olevien samojen tilojen rooli voi riippua myös polkuhinnasta: Jos kuvan 15 autoilutehtävässä minimoidaan vain kuljettujen teiden lukumäärää, niin reitit

(a) Arad – Sibiu – Rimnicu Vilcea – Craiova – Dobreta ja
(b) Arad – Timisoara – Lugoj – Mehadia – Dobreta

ovat "samat": molemmissa päädytään samaan kaupunkiin 4 askeleessa.

Jos taas minimoidaan kokonaismatkaa, niin reitti 2a on 486 km pitkä, mutta reitti 2b vain 374 km. Oletetaan, että reitti 2a löydetään ensin, ja sen jälkeen reitti 2b jätetään huomiotta "samana": saatamme päätyä väärään vastaukseen!

Silloin "samuus" on määriteltävä siten, että se ottaa huomioon myös kertyneen polkuhinnan.

Tällöin vähenee poistettavissa olevan toiston määrä.

Kokonaiskustannukset ja -hyödyt riippuvat tietenkin siitä, kuinka paljon toistoa etsintäongelmassamme esiintyy:

- Kuvan 20 mukaisessa 8-liukupelissä on $d \approx 20$ ja $b \approx 3$.
- Sen etsintäpuussa on siis noin 5.2×10^9 solmua yhtälöstä (7).
- Niistä vain $9! = 362\ 880$ on erilaisia.
  Polkuhinnoista (siirtomääristä) ei tarvinne pitää huolta.
  Näin pieni määrä mahtuu varmaankin jo keskusmuistiin.
  (Toisaalta virtuaalimuistiin joutuminen hidastaa.)
- Siis toistuvien tilojen poisto hävittää 99.99% etsintäpuusta!
  Eli kannattanee tässä tapauksessa.

3.2. Informoituja etsintämenetelmiä [RN95, luku 4] [Nil98, luku 9]

Luvun 3.1 algoritmit ovat usein tehottomia, koska ne eivät ota huomioon ratkaistavan etsintäongelman erityispiirteitä. Parempi tulos saadaan algoritmeilla, joita suunnittelija voi "ohjata" kohti tavoitetta jollakin *ongelmakohtaisella lisäinformaatiolla*.

Luvun 3.1.2 algoritmi käytti etsintäongelman määrittelyyn kuuluvaa polkuhintafunktiota g. Se valitsi aina laajennettavaksi sellaisen etsintäpuun solmun n, jossa "tähän mennessä toimijalle suunniteltu työmäärä"

$$g(n) = \text{juuresta solmuun } n \text{ vievän polun hinta}$$

on pienin. Lisäinformaatiota tuo puolestaan funktio

$$h^*(n) = \text{solmusta } n \text{ lähimpään ratkaisuun vievän polun hinta}$$

joka antaa "tästä eteenpäin jäljellä olevan työmäärän".

Tällaisten funktioiden ohjaama etsintämenetelmä on **"paras ensin" -etsintä (best-first search)**:

function BEST-FIRST-SEARCH(*problem,* EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem, Queueing-Fn*)

Algoritmi 4: Paras ensin -etsintä [RN95, kuva 4.1].

Toteutus on kuten luvussa 3.1.2, paitsi että keko järjestetäänkin nyt ohjausfunktion EVAL-FN solmuille antamien arvojen mukaan.

Oikean funktion h^* *tarkkoja* arvoja ei yleensä ole tarjolla. Siksi käytetäänkin jotenkin etsintäongelmasta johdettua *heuristista arviofunktiota* h.

Sanalla "heuristiikka" (kr. *heurisko*, "keksiä"; vrt. "Heureka!") on useita eri merkityksiä:

- Tässä yhteydessä se tarkoittaa algoritmista tekniikkaa, joka saattaa *parantaa algoritmin suorituskykyä*.
- Se voi tarkoittaa myös sellaista yksinkertaistavaa oletusta, jonka avulla jokin vaikea *algoritminen ongelma helpottuu* — ratkeamattomasta ratkeavaksi, NP-täydellisestä polynomiseksi,. . .
  Esimerkiksi tyydytään tarkastelemaan vaikean yleisen tapauksen sijasta sellaisia yksinkertaisempia tapauksia jotka kattavat kaikki "käytännössä vastaan tulevat" tilanteet. . .
- Joskus se tarkoittaa jopa *algoritmisen ongelmanratkaisun vastakohtaa*, "luovaa inhimillistä" säännöistä vapaata ongelmanratkaisua!

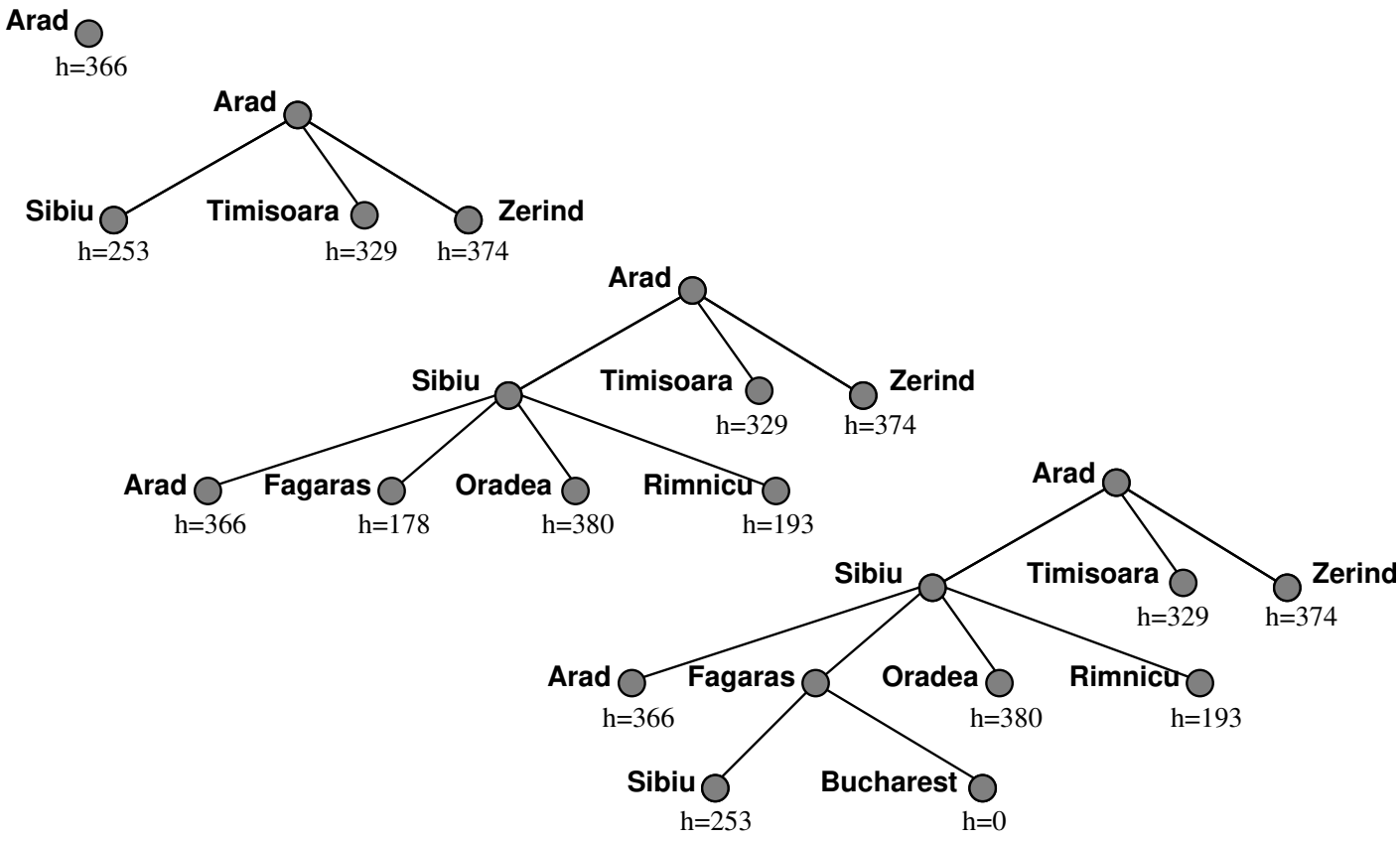
Yleisen systemaattisen etsintälgoritmin 2 tavoin yleisestä informoidusta etsintäalgoritmista 4 saadaan yksittäisiä algoritmeja kiinnittämällä laajennusjärjestyksen määräävä funktio $\textsc{Eval-Fn}$ eri tavoin.

Esimerkiksi luvun 3.1.2 algoritmi (joka ei vielä ollut informoitu eli ei käyttänyt funktiota h) saadaan valitsemalla $\textsc{Eval-Fn} = g$, eli ongelman polkuhinta juuresta nykyiseen lehtisolmuun.

Esimerkkinä informoidusta etsinnästä käytetään kuvan 15 tehtävää löytää reitti Aradista Bukarestiin. Lisäinformaatioksi h on kuvan oikeassa laidassa listattu kuinka kaukana Bukarest on kustakin kaupungista maantieteellisesti eli "linnuntietä". (Itse reitti kiemurtelee taas tarjolla olevia maanteitä.)

3.2.1. Ahne etsintä (Greedy Search)

Valitaan ensin Algoritmissa 4 $\textsc{Eval-Fn} = h$. Saatu algoritmi valitsee laajennettavaksi sen etsintäpuun lehden, joka kulloinkin *näyttää olevan lähimpänä tavoitetta*:



Kuva 29: Ahneesti Aradista Bukarestiin [RN95, kuva 4.3].

Tämä algoritmi on luvun 3.1.2 algoritmin "peilikuva":

- siellä pysyteltiin mahdollisimman lähellä lähtösolmua
- täällä rynnätään mahdollisimman lähelle maalia.

Ahne algoritmi käyttäytyy *usein muttei aina* ikään kuin luvun 3.1.3 syvyyssuuntainen etsintä, johon on lisätty "suuntavaistoa":

- Algoritmi pyrkii jatkamaan samaa polkua yhä eteenpäin ja eteenpäin, mutta nyt polkua ei jatketakaan "ensimmäisestä" lapsisolmusta vaan

siitä — ei välttämättä (mutta usein) lapsi- — solmusta jonka arvioitu etäisyys tavoitteesta on pienin.

- Algoritmi muuttaa kulkuaan vasta kun tämä polku päätyy umpikujaan tai alkaa etääntyä tavoitteestaan. Mutta silloinkin peruutetaan yleensä samalla polulla taaksepäin ja kokeillaan jotakin melkein yhtä lupaavaa vielä tutkimatonta vaihtoehtoa.
- Kuvasta 29 nähdään, että ahneus *ei ole optimaalista*: löydetty reitti on 32 km pidempi kuin Rimnicu Vilcea – Pitesti -vaihtoehto. Tämä paras vaihtoehto jäi huomaamatta, koska Fagaras näytti *paikallisesti lupaavammalta*.
- Ahneus *ei ole myöskään täydellistä*: jos kuvan 15 ongelmana olisikin päästä Iasista Fagarasiin, niin ahne algoritmi (ilman luvun 3.1.7 toistojen poistoa) jää heilahtelemaan välille Iasi – Neamt.
- Algoritmien pahimman tapauksen aikavaatimukset ovat samat, joskin "hyvällä" h ahneesta algoritmista voi tulla hyvin nopea.
- Toisin kuin luvun 3.1.3 syvyyssuuntainen etsintä, ahne algoritmi joutuu pitämään kaikki etsintäpuun solmut muistissa.

  Etsintä saattaa näet palata myöhemmin kokeilemaan jotakin kesken jäänyttä solmua sitten kun kaikki sitä lupaavammat on käsitelty.

3.2.2. A*-etsintä

Luvun 3.2.1 ahne etsintä on nopea muttei täydellinen eikä optimaalinen. Luvun 3.1.2 algoritmi oli taas täydellinen ja optimaalinen muttei nopea. Yhdistetään ne (muista luku 3.2) valitsemalla Algoritmissa 4 $\textsc{Eval-Fn} = f$, missä

$$f(n) = g(n) + h(n) \tag{9}$$

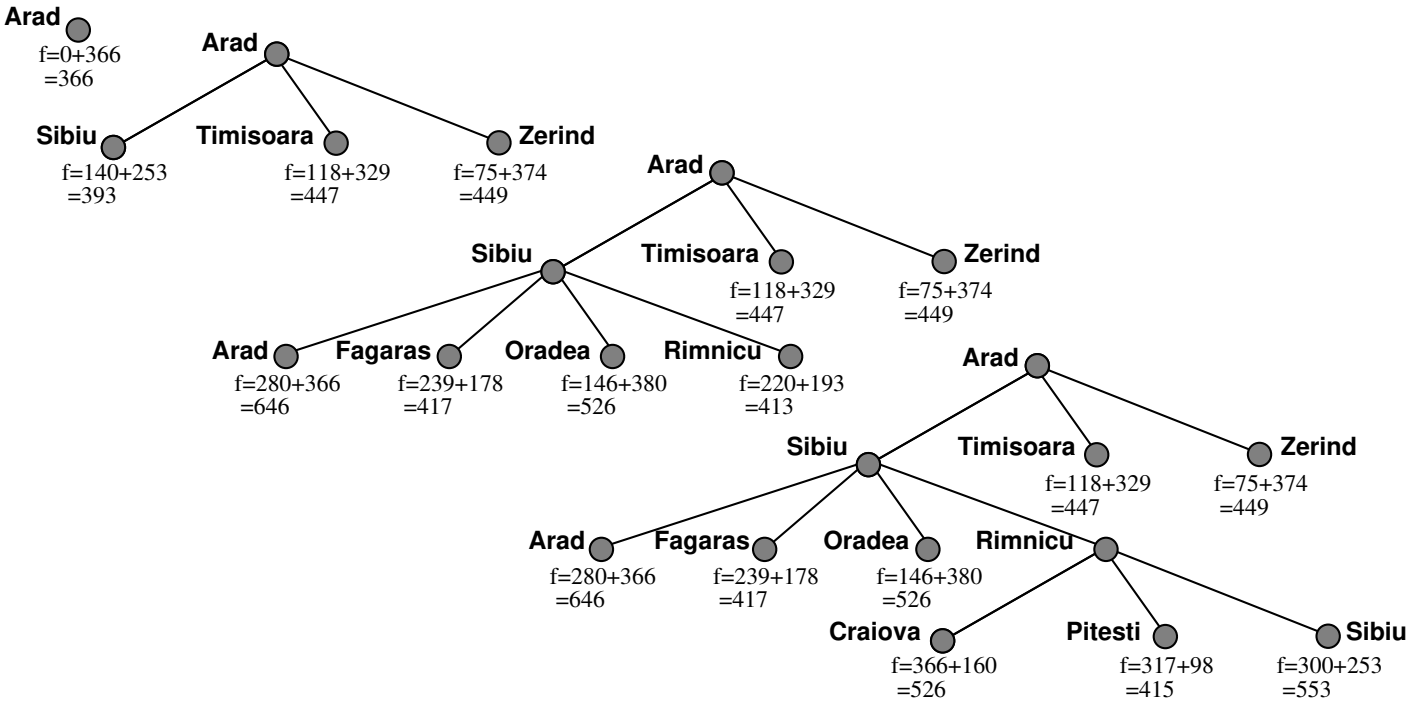
arvioi *halvimman sellaisen ratkaisun hintaa, joka kulkee solmun n kautta*.

Vaaditaan lisäksi että heuristiikka h on *luvallinen (admissible)*, eli että

$$h(n) \leq h^*(n)$$

jokaiselle solmuille n. Toisin sanoen,että h on *aina optimistinen*: vielä jäljellä oleva työmäärä näyttää aina pienemmältä kuin se oikeasti onkaan.

Silloin myöskään f ei koskaan yliarvioi pessimistisesti solmun n kautta kulkevien ratkaisujen hintoja. Kuvassa 15 luvallinen h on "etäisyys linnuntietä Bukarestiin":

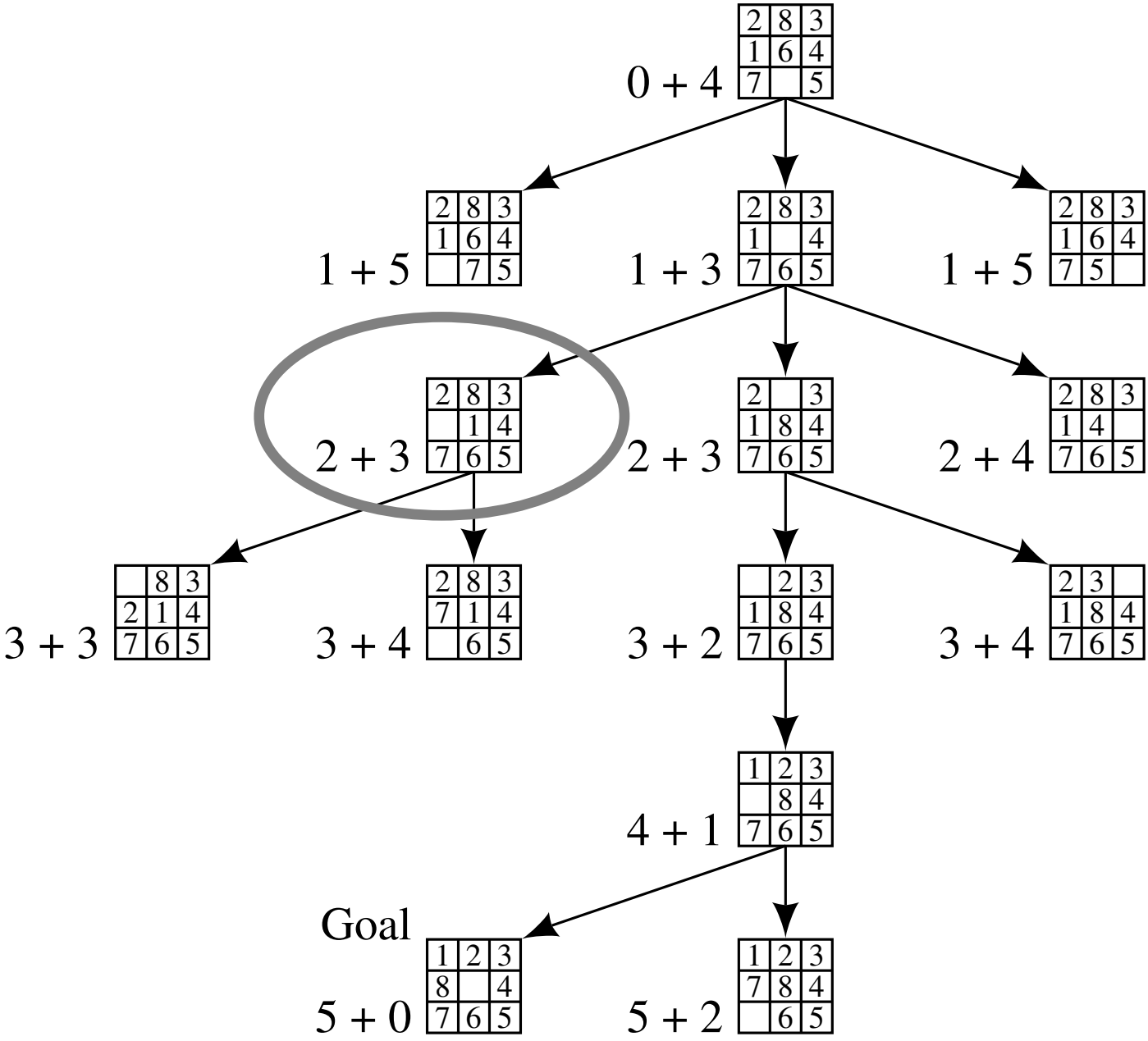


Kuva 30: Parhaiten Aradista Bukarestiin [RN95, kuva 4.4].

Tai jos 8-liukupelissä

$\hat{h}(n) =$ laudalla n väärissä paikoissa olevien tiilten määrä

niin etsintä onnistuu nopeasti (lukuunottamatta rengastettua haaraa):



© 1998 Morgan Kaufman Publishers

Kuva 31: Heuristinen etsintä 8-liukupelissä [Nil98, kuva 9.2].

- g on *leveys*suuntainen osa joka pyrkii *huolellisuuteen*.
- h on *syvyys*suuntainen osa joka pyrkii *tehokkuuteen*.

A*-algoritmi on. . .

optimaalinen: Kuvan 30 juuresta alkavilla poluilla funktion f antamat arvot *eivät vähene*.

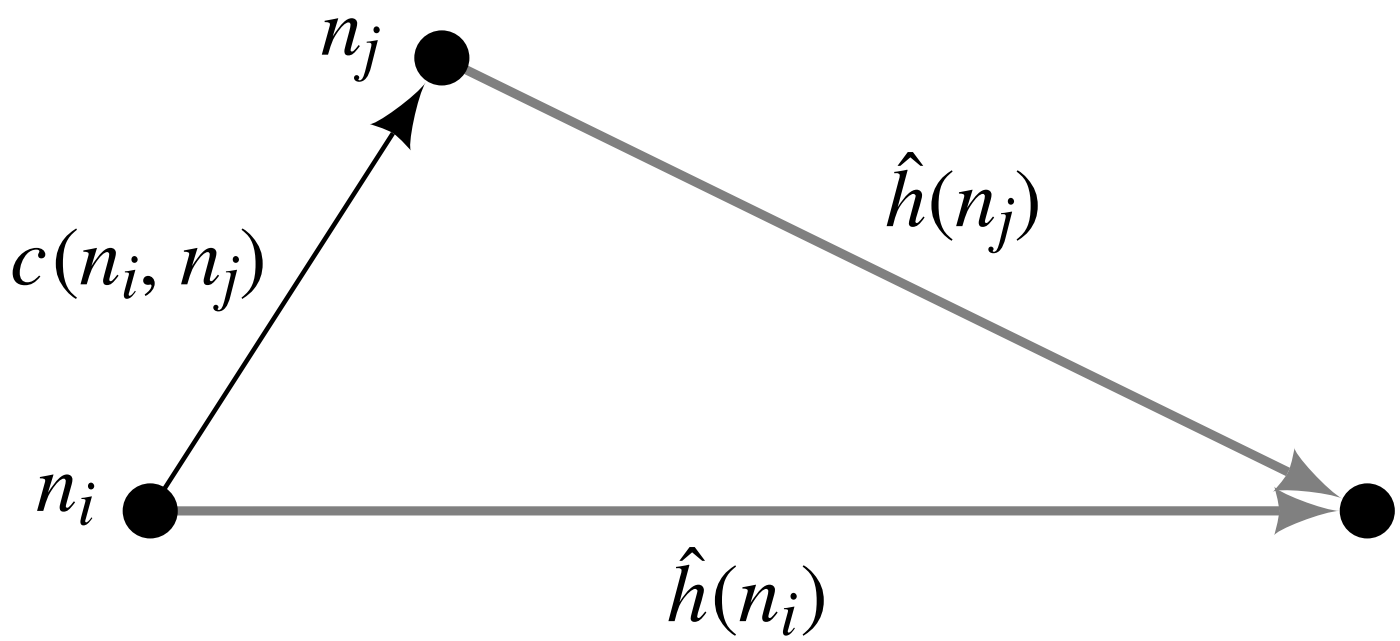
Tällaista f kutsutaan *monotoniseksi*, tarkemmin sanoen *kasvavaksi* [Myr74a, määritelmä 2.11.1].

Koska polkuhinta g oletettiin kasvavaksi ja '+' säilyttää kasvavuuden, riittää tarkastella heuristiikkaa h.

Funktio f on monotoninen jos ja vain jos h toteuttaa *kolmioepäyhtälön*, joka saa nyt muodon

$$h(n) \leq k(n, n') + h(n')$$

missä $k(n, n')$ on pienin polkuhinta solmusta n solmuun n' (todistus sivuutetaan).

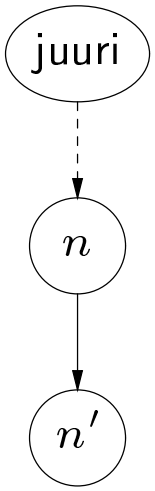


© 1998 Morgan Kaufman Publishers

Kuva 32: Kolmioepäyhtälö [Nil98, kuva 9.7].

Esimerkiksi kuvan 15 "linnuntie-etäisyys" täyttää kolmioepäyhtälön, onhan se geometrinen etäisyysmitta.

Jos f ei olekaan monotoninen,niin etsintäpuusta löytyy haara



jossa $f(n) > f(n')$. Mutta $f(n)$ oli optimistinen arvio *kaikille* juuresta solmun n kautta maaliin kulkeville poluille — myös niille jotka kulkevat kaaren isäsolmusta n lapsisolmuun n' kautta, mutta näitähän $f(n')$ pitää vielä parempina!

Monotonisuus voidaan siis palauttaa yhtälöön (9) *pathmax-yhtälöllä*

$$f(n') = \max(f(n), g(n') + h(n')) \tag{10}$$

jonka jälkeen f on varmasti monotoninen, kunhan h on luvallinen.

Oletetaan siis f monotoniseksi. Silloin optimaalisuus voidaan näyttää seuraavasti:

Algoritmin voidaan nyt ajatella piirtävän sisäkkäisiä *korkeuskäyriä* etsintäavaruuteen (kuva 33).

Luvun 3.1.2 etsintäalgoritmissa (jossa $h = 0$) nämä käyrät ovat renkaita alkutilan ympärillä, kun taas paremmat heuristiikat "venyttävät" niitä soikioiksi kohden maalitilaa.

Olkoon f^* parhaan ratkaisun polkuhinta. Silloin algoritmi laajentaa

- ensin kaikki solmut n käyrän f^* sisällä (eli $f(n) < f^*$) funktion f arvojen mukaan kasvavassa järjestyksessä, ja
- sitten osan solmuista itse käyrällä f^* (eli $f(n) = f^*$),

kunnes itse käyrältä f^* löytyy (jokin) maalisolmu.

Formaalisti: olkoon G optimaalinen maalisolmu, ja olkoon G_2 jokin

toinen ei-optimaalinen maalisolmu, jolla siis

$$f(G_2) > f^*. \tag{11}$$

Voisiko algoritmi vastata G_2? Silloinhan se *ei* olisikaan optimaalinen! Olkoon n juuresta solmuun G vievän optimaalisen polun P tämänhetkinen lehti; jos sellaista ei ole, niin P on jo käsitelty ja G vastattu. (Kuva 34 näyttää tilanteen.)

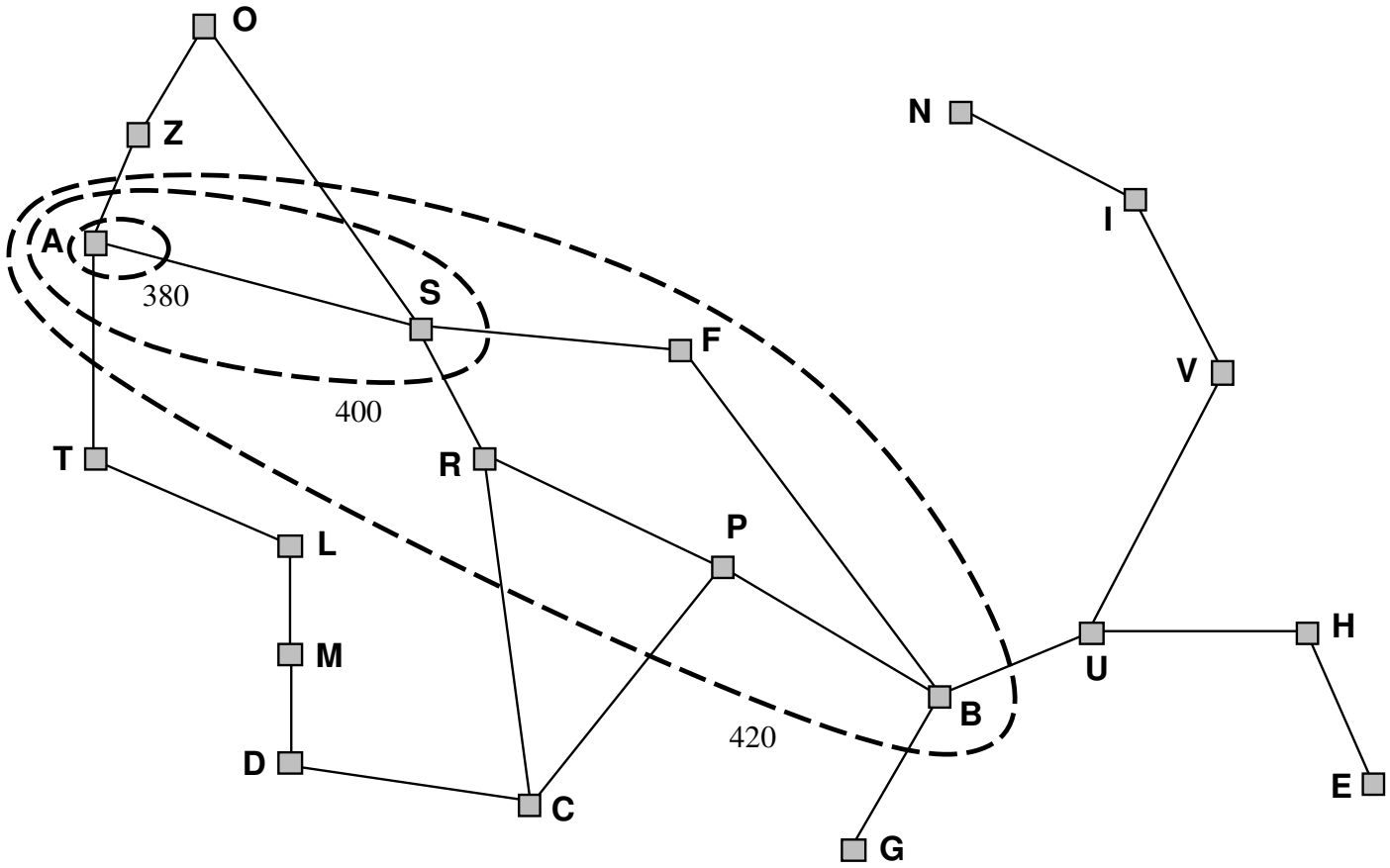
$$\begin{aligned}
f^* &= f(G) \text{ koska } G \text{ on optimaalinen.} \\
&\geq f(n) \text{ koska } f \text{ on monotoninen.} \\
&\geq f(G_2) \text{ koska } G_2 \text{ valittiin ennen kuin } n. \\
&= g(G_2) + h(G_2) \text{ yhtälöstä (9).} \\
&= g(G_2) \text{ koska } h \text{ on luvallisena } 0 \text{ kaikille maaleille.}
\end{aligned}$$

Tämä on kuitenkin ristiriidassa yhtälön (11) kanssa, joten algoritmi on optimaalinen.

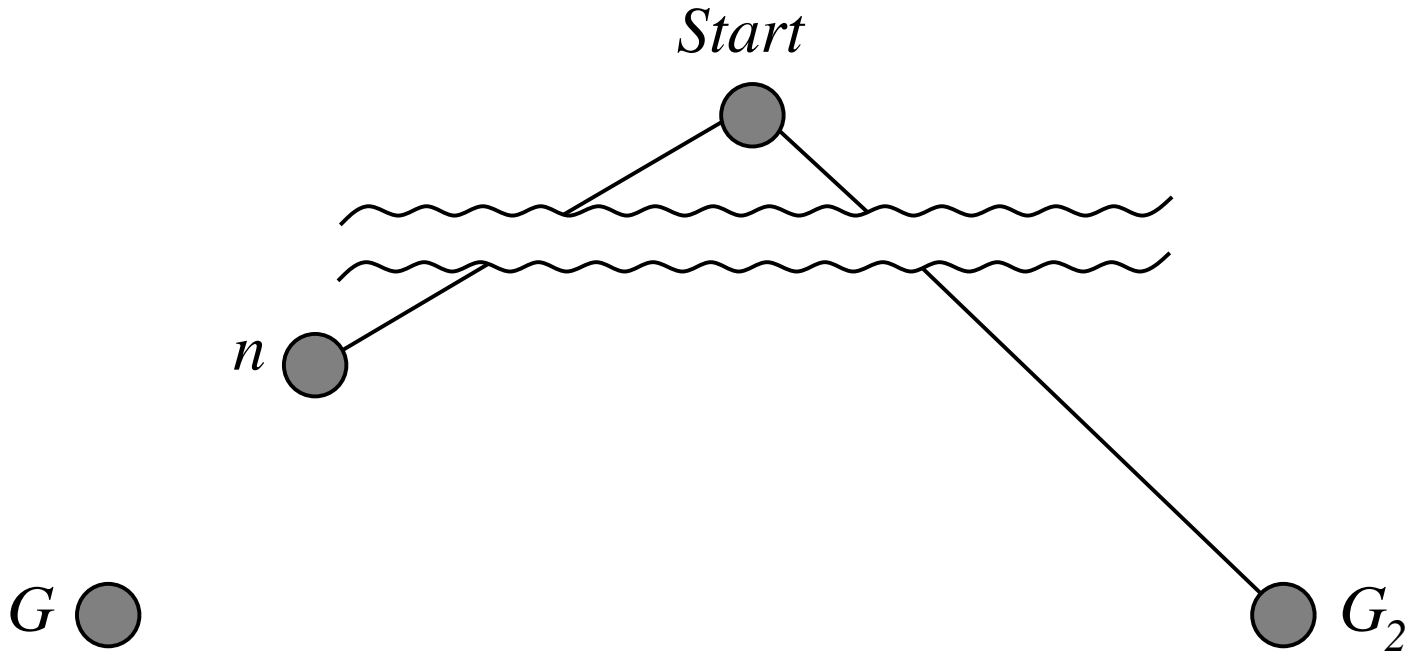
täydellinen: laajentamisjärjestyksensä vuoksi algoritmi löytää maalisolmun *ellei* käyrälle f^* tai sen sisälle jää äärettömän monta solmua, jotka algoritmi siis yrittäisi käsitellä ensin. Tämä on puolestaan mahdollista vain jos etsintäavaruudessa on. . .

- solmu, jolla on ääretön haarautumisaste.
  Tällöin etsintää tekevällä toimijalla olisi äärettömän monta eri vaihtoehtoa vastaavassa tilanteessa.
- ääretön polku $P = n_0 \xrightarrow{c_1} n_1 \xrightarrow{c_2} n_2 \xrightarrow{c_3} n_3 \cdots$ jonka hinta $g(P) = c_1 + c_2 + c_3 + \cdots \leq f^*$ pysyy äärellisenä.
  Tällöin etsintää tekevällä toimijalla olisi tarjolla mielivaltaisen "halpoja" tekoja.
  Tästä ehdosta mainittiin jo luvussa 3.1.4.

Algoritmi on siis täydellinen sellaisissa avaruuksissa joissa *haarautumisaste on äärellinen* ja *kaikkien tekojen kustannuksilla on jokin alaraja* $\delta > 0$.



Kuva 33: Korkeuskäyrät Aradista Bukarestiin [RN95, kuva 4.5].



Kuva 34: Algoritmin $\mathbf{A}^*$ optimaalisuus [RN95, kuva 4.6].

tehokas: algoritmi on *optimaalisen tehokas (optimally efficient)* seuraavassa mielessä:

Olkoon $\mathbf{B}^*$ minkä tahansa kilpaileva etsintäalgoritmi, joka. . .

- käyttää samaa heuristista ohjausinformaatiota h kuin $\mathbf{A}^*$,
- on optimaalinen kuten $\mathbf{A}^*$, ja
- etenee samoin kuten $\mathbf{A}^*$ laajentamalla polkuja juuresta alkaen.

Silloin on väistämättä olemassa sellaisia etsintäongelmia, joissa $\mathbf{B}^*$ *joutuu laajentamaan ainakin yhtä monta solmua kuin* $\mathbf{A}^*$:

Muuten $\mathbf{B}^*$ jättäisi laajentamatta solmuja käyrältä f^* tai sen sisältä, ja saattaisi näin jättää optimipolun laajentamatta. (Todistus sivuutetaan.)

Toisaalta käyrälle f^* ja sen sisälle jää yleensä eksponentiaalisen monta solmua verrattuna optimipolun solmujen määrään, joten $\mathbf{A}^*$ on huolellisuutensa vuoksi yleensä *eksponentiaalinen*.

Tämä eksponentiaalisuus voidaan varmasti välttää vain jos

$$\left|h(n) - h^*(n)\right| \leq O(\log h^*(n))$$

eli jos heuristiikka h on niin tarkka, että sen tuottama *virhe on aina korkeintaan logaritminen*. (Todistus sivuutetaan.)

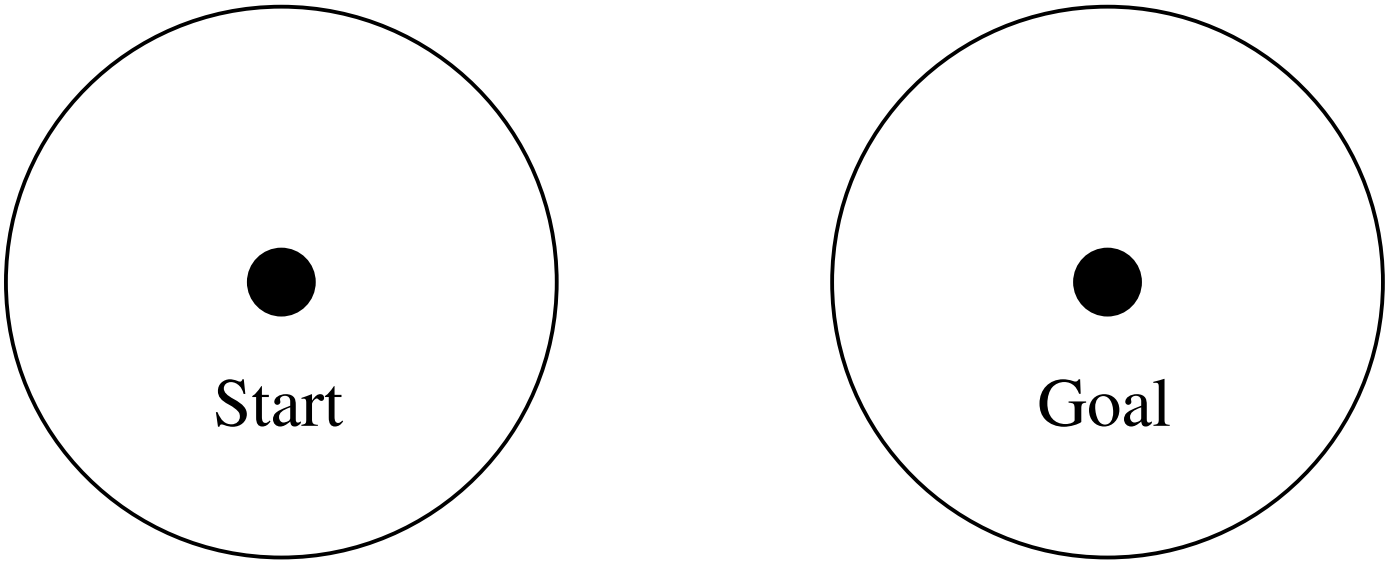
Käytännössä virhe tapaa olla ainakin luokkaa $g(n)$.

Algoritmin $\mathbf{A}^*$ suurempi ongelma on kuitenkin (jälleen) *muistintarve*, koska se pitää kaikki (lehti)solmut muistissa. Tähän ongelmaan palataan luvussa 3.2.4.

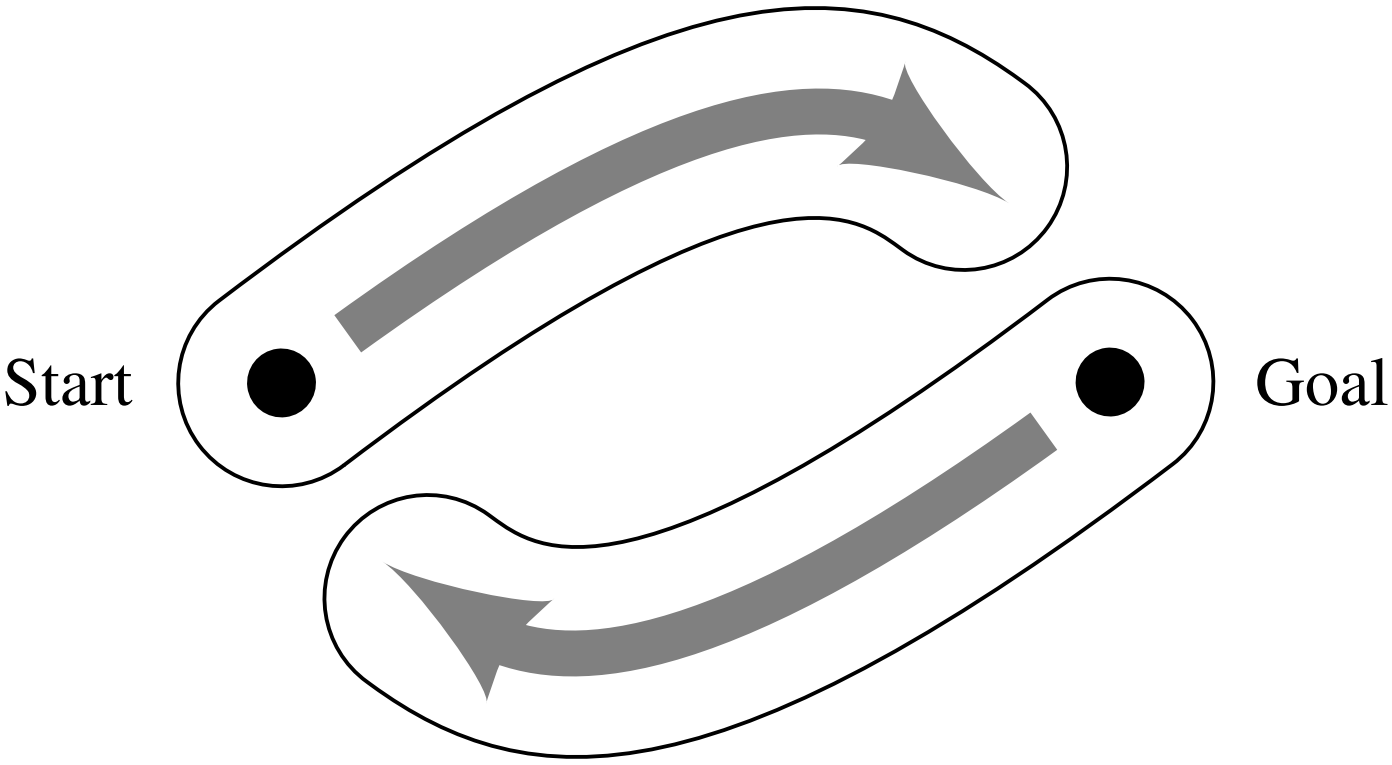
Jos heuristista etsintää (esimerkiksi $\mathbf{A}^*$) käyttää luvun 3.1.6 kaksisuuntaisessa etsinnässä, voivat eri suuntiin etenevät, yksinään hyvät heuristiikat kumota toisensa [Nil98, kuva 9.10] (sivulla 86):

Kumpikin etenee omaan hyvään suuntaansa, ja ohittaa toisen. Etsintätyö ei putoakaan murto-osaan, vaan kaksinkertaistuu, koska kumpikin suunta päätyy tekemään työn kokonaan yksin, eikä puoliksi vastasuunnan kanssa.

Tästä syystä suositellaankin, että toinen kaksisuuntaisista etsintäalgoritmeista olisikin leveyshaku eikä heuristiikka.



(a) Breadth-first search



(b) Heuristic search

© 1998 Morgan Kaufman Publishers

Entä jos heuristiikkamme h *ei* olekaan luvallinen?

- Silloin $h(n)$ voikin arvioida solmusta n lähimpään ratkaisuun vievää hintaa $h^*(n)$ *pessimistisesti* ylöspäin (eli $h(n) > h^*(n)$).
- Silloin saattaakin olla olemassa jokin toinen solmu n' jolla ylöspäin pyöristystä ei tapahdukaan (eli $h(n') \leq h^*(n)$).
- Silloin saattaakin olla

$$\underbrace{g(n') + h(n')}_{f(n')} < \underbrace{g(n) + h(n)}_{f(n)}$$

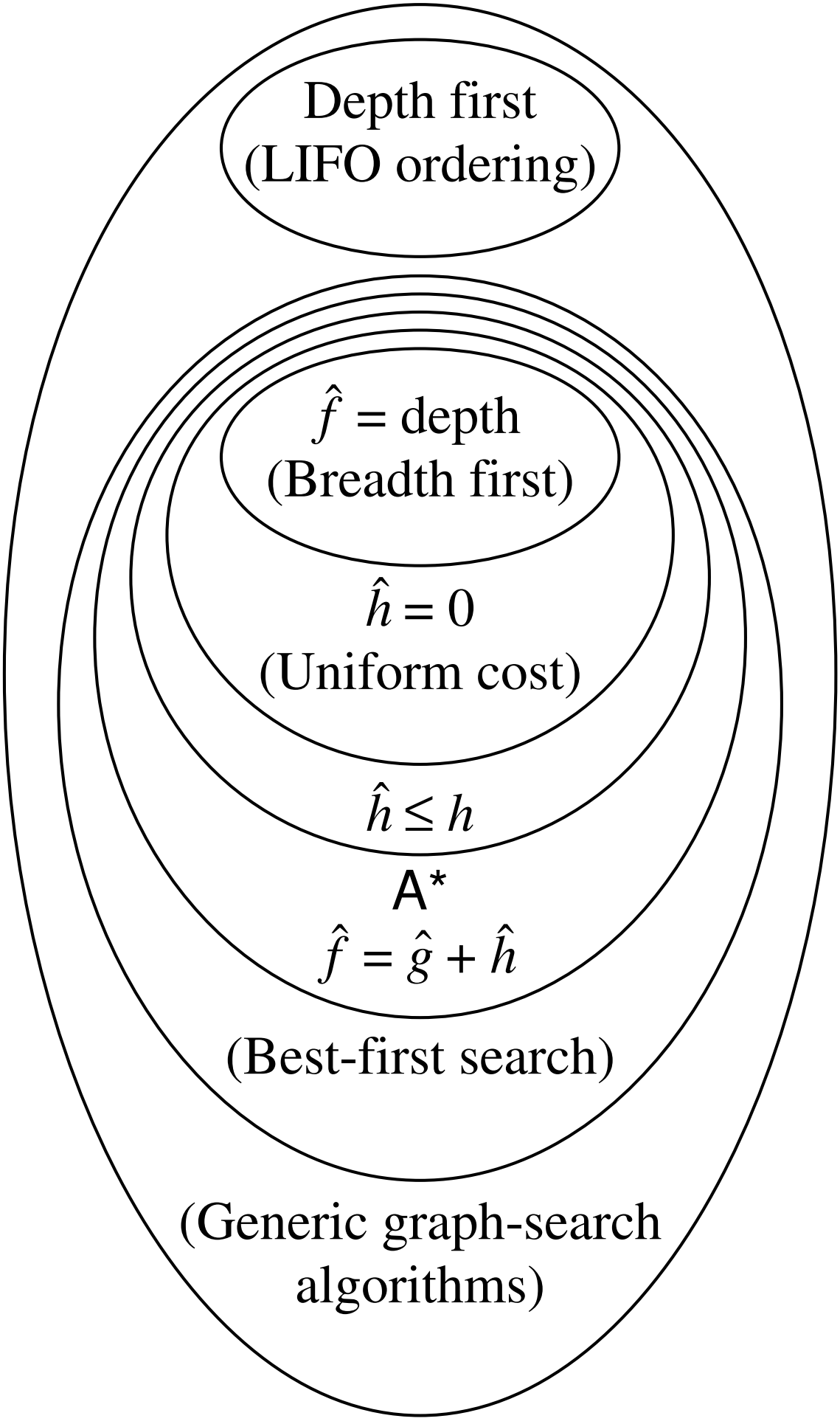
vaikka todellisille hinnoille

$$g(n') + h^*(n') > g(n) + h^*(n).$$

- Silloin etsintä valitseekin todellisuudessa huonomman solmun n' eikä parempaa solmua n.
- Siten etsintä voi päätyä tulokseen, joka ei olekaan halvin mahdollinen — enää etsintä *ei ole optimaalinen*.

Sivun 88 kuva [Nil98, kuva 9.6] kertaa eri etsintämenetelmien välisiä suhteita seuraavin merkinnöin:

[Nil98]	[RN95]
$\hat{h}$	h
h	h^*
$\hat{f}$	f
$\hat{g}$	g



© 1998 Morgan Kaufman Publishers

3.2.3. Miten tutkia heuristisia funktioita?

Tarkastellaan kuvan 20 8-liukupeliä. Luvallisia heuristiikkoja ovat esimerkiksi

$h_1(n) = \hat{h}(n)$ luvusta 3.2.2

$h_2(n) = \sum_{1 \leq i \leq 8}$ tiilen i *Manhattan-etäisyys* oikeasta paikastaan

$= 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2$ kuvan 20 lähtötilassa.

Eräs heuristiikan h hyvyyden mitta on sen aikaansaama *todellinen haarautumisaste (effective branching factor)* b^*: Lasketaan ensin

- montako solmua N algoritmi **A*** laajentaa heuristiikalla h ja
- näin löydetyn ratkaisun syvyys d.

Ratkaistaan sitten b^* yhtälöstä (7)

$$N = \frac{(b^*)^{d+1} - 1}{b^* - 1}.$$

Tulos kertoo

- millainen haarautumisaste (reaalilukuna) olisi saman kokoisella *täydellisellä* puulla, eli
- kuinka leveää luvun 3.1.1 *leveyssuuntaista* etsintää **A*** vastaa heuristiikalla h.

Tulos mittaa heuristiikan h laatua koska

- leveyshaku oli hidas mutta huolellinen menetelmä joka ei huomioinut mitään lisäinformaatiota, joten

- mittausarvo kertoo kuinka kapeaksi heuristiikka pystyi kaventamaan leveyshaun, eli
- kuinka monta sellaista vaihtoehtoa yhdessä etsintäaskeleessa on, joista pitäisi valita "umpimähkään".
  Pienempi mittausarvo on siis parempi heuristiikka!

Jos esimerkiksi

$$N = 52 \text{ ja}$$
$$d = 5 \text{ niin}$$
$$b^* \approx 1.91$$

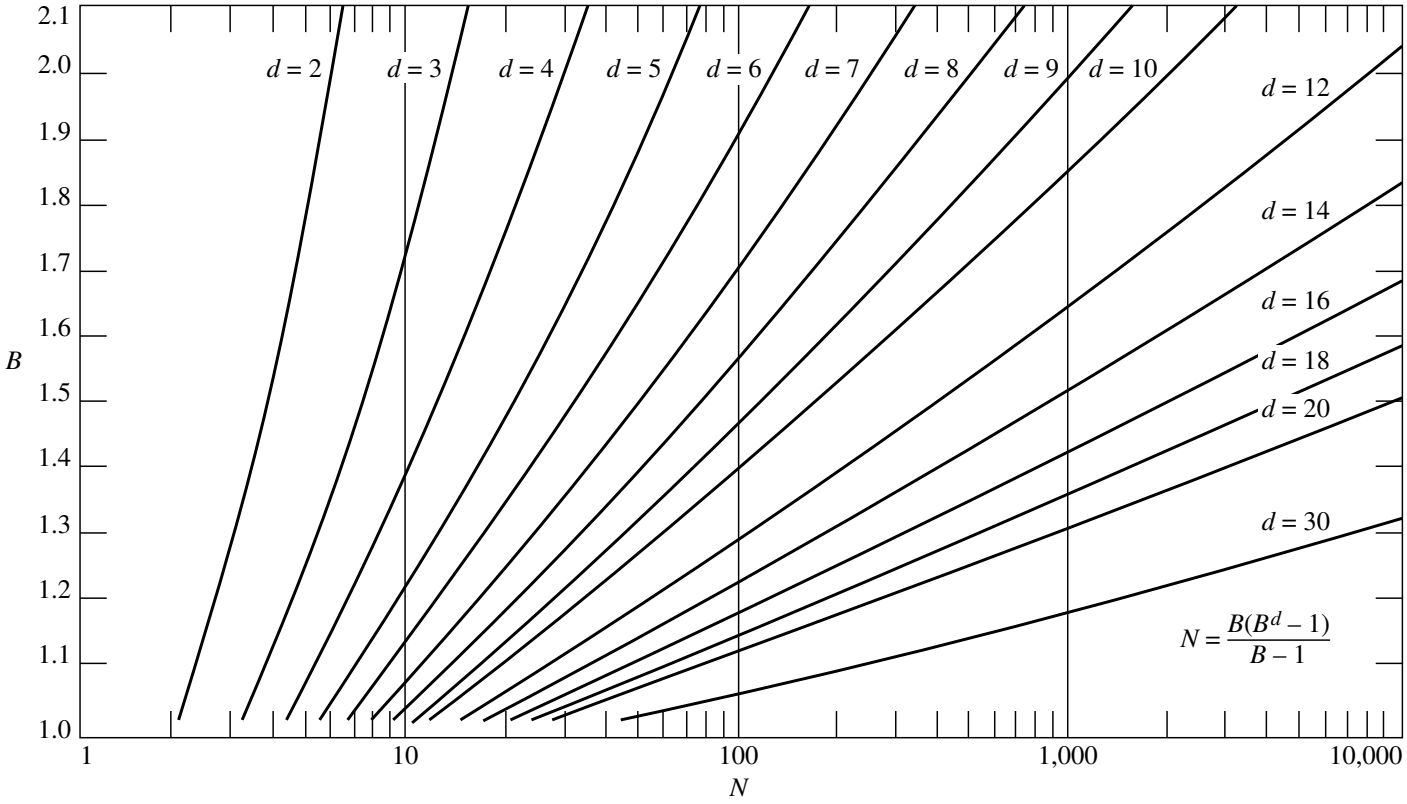
eli

- hakupuussa on $d = 5$ sukupolvea isäsolmuja, joista kullakin on (keskimäärin) $b^* = 1.91$ lapsisolmua, joten
- 9 etsintäaskeleen jälkeen on noin 9 tutkimatonta "yhtä hyvää" vaihtoehtoa, binääripuussa taas 10.

Taulukossa 2 verrataan mitalla b^* heuristiikkoja h_1 ja h_2 toisiinsa ja luvun 3.1.5 iteratiiviseen syventämiseen.

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Taulukko 2: Heuristiikkojen vertailua [RN95, kuva 4.8].



© 1998 Morgan Kaufman Publishers

Kuva 35: $B \approx b^*$ vs. N eri arvoilla d [Nil98, kuva 9.11].

Heuristiikka h_2 *dominoi* heuristiikkaa h_1, koska (niiden määritelmistä nähdään, että)

$$h_2(n) \geq h_1(n) \text{ kaikilla } n.$$

Silloin on *aina* parempi käyttää heuristiikkaa h_2 kuin h_1: $\mathbf{A}^*$ laajentaa aina *kaikki* solmut n joilla

$$f(n) < f^* \text{ mikä on yhtälön (9) nojalla}$$
$$h(n) < f^* - g(n).$$

Siis h_1 laajentaa aina *vähintään samat solmut* kuin h_2.

Jos on kaksi heuristiikkaa h' ja h'' joista kumpikaan ei dominoi toista, voidaan valita

$$h(n) = \max(h'(n), h''(n))$$

ja saadaan uusi heuristiikka, joka dominoi kumpaakin. Voidaan siis kehittää ongelmalle erilaisia heuristiikkoja, ja valita niiden antamista arvioista aina suurin.

Heuristiikka olkoon siis optimistinen, mutta vain vähän!

Heuristiikat h_1 ja h_2 saatiin *rentouttamalla (relax)* alkuperäisen ongelman vaatimusta "tiili voi siirtyä paikasta A paikkaan B jos B on tyhjä ja A sen naapuri" *helpompaan muotoon*

1. "tiili A voi siirtyä *aina mihin tahansa* paikkaan B" tai
2. "tiili A voi siirtyä *aina naapuripaikkaansa* B".

Yksi tapa keksiä heuristiikkoja onkin rentouttaa ongelmaa unohtamalla osa sen vaatimuksista ja kehittämällä mitta tämän helpotetun ongelman tapausten etäisyydelle ratkaisusta.

Sovellusalueen *taitajat* saattavat myös tarjota heuristiikkoja. Esimerkiksi korttipelin Bridge teoreetikot ovat kehittäneet pisteytyssääntöjä käden pelivoiman arviointiin:

- Kuvakortti K antaa $K - 10$ pistettä.
- Maa, josta ei ole lainkaan kortteja antaa 3 pistettä valttipelissä, mutta ei valtittomassa.
- Maa, jossa on vain yksi kortti antaa 2 pistettä valttipelissä, mutta sen kortin kuvapisteitä ei enää saa laskea ellei se ole ässä.

$\vdots$

Tällaiset säännöt poimivat ongelman keskeisiä *piirteitä (features)*, joiden arvojen perusteella voi suhteellisen luotettavasti päätellä, onko tämä tilanne hyvä vai huono. Bridgessä tällaiset säännöt on johdettu seuraavasti:

- Onko omassa kädessäni ainakin $\frac{1}{3}$ pakan kaikista pisteistä?

- Onkohan partnerini käteen päätynyt ainakin $\frac{1}{3}$ pakan jäljellä olevista pisteistä?

  Tämä selvitetään peliin kuuluvassa tarjouskilpailussa.
- Silloin meillä on yhteensä ainakin

$$\frac{1}{3} + \frac{1}{3} \cdot \frac{2}{3} = \frac{5}{9} > \frac{1}{2}$$

  pakan kaikista pisteistä.

  Silloin me olemme vahvemmilla kuin vastustajaparimme, joten meidän kannattaa ottaa pelissä aloite.

Joskus tärkeitä piirteitä voidaan kyllä valita, mutta niiden tarkkojen numeeristen painoarvojen määrääminen on vaikeampaa: minkä arvoinen on shakkipelin sotilas — ja miten sen arvo kehittyy pelin myötä?

Numeroarvoja voidaan yrittää päätellä *tilastotieteen* tai *koneoppimisen* [RN95, osa VI] keinoin aineistona aiemmat saman tehtävän ratkaisut. Näitä saadaan "historiasta" (muistiinmerkityt "hyvät" shakkipelit) tai tietokonesimulaatioilla (pannaan tietokone pelaamaan itseään vastaan erilaisia loppupelimuunnelmia ja katsotaan tuloksia).

Tilastot tai historia eivät kuitenkaan takaa arvioiden luvallisuutta.

3.2.4. Muistin säästäminen algoritmissa A*

Luvussa 3.2.2 todettiin, että optimikäyrälle f^* jää yleensä eksponentiaalisen monta solmua. Ne kaikki ovat etsintäpuun lehtiä, joten algoritmin **A*** olisi pidettävä ne muistissa. Tämä on taas käytännössä hyvin raskasta.

Tutustutaan siis seuraavaksi esimerkein (yksityiskohdat sivuuttaen) kahteen tekniikkaan, joilla voi vähentää algoritmin **A*** muistivaatimuksia:

- Jos sovelletaan luvun 3.1.5 iteratiivisen syventämisen ideaa, niin saadaan algoritmi **IDA* (Iterative Deepening A*)**.
- Algoritmi **IDA*** säästää kuitenkin muistia jo silloin kun sitä sitä olisi vielä jäljellä.

  Tehokkaampaa on alkaa säästötoimet vasta kun muisti todellakin loppuu.

  Näin toimii algoritmi **SMA* (Simplified Memory-Bounded A*)**.
- Algoritmissa **SMA*** käytetään sellaista tekniikkaa, jolla voidaan parantaa luvun 3.2 yleistä "paras ensin" -etsintää.

  Näin saadaan algoritmi **RBFS (Recursive Best-First Search)** [Nil98, luku 9.2.5].

  Tutustutaan siis tähän tekniikkaan ja algoritmiin ennen algoritmia **SMA***.

Jokaisessa unohdetaan työmuistin loppuessa sellaisia solmuja, jotka eivät nyt näytä lupaavilta, mutta kuitenkin siten, että niihin voidaan myöhemmin palata.

IDA*

Kun iteratiivinen syventäminen tapahtuu aina seuraavalle korkeuskäyrälle f (kuvassa 33) seuraavan syvyystason sijasta, saadaan seuraava algoritmi:

```
function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f- COST limit
          root, a node

  root ← MAKE-NODE(INITIAL-STATE[problem])
  f-limit ← f- COST(root)
  loop do
      solution, f-limit ← DFS-CONTOUR(root, f-limit)
      if solution is non-null then return solution
      if f-limit = ∞ then return failure; end
```

```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f- COST limit
  inputs: node, a node
          f-limit, the current f- COST limit
  static: next-f, the f- COST limit for the next contour, initially ∞

  if f- COST[node] > f-limit then return null, f- COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSORS(node) do
      solution, new-f ← DFS-CONTOUR(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← MIN(next-f, new-f); end
  return null, next-f
```

Algoritmi 5: Iteratiivisesti syventävä **A*** (**IDA***) [RN95, kuva 4.8].

Sen ominaisuudet ovat muuten samanlaiset kuin algoritmin **A***, mutta syvyyssuuntaisena sen tilavaatimus on enää $O(\frac{b f^*}{\delta})$ solmua.

Useimmissa tapauksissa syvyyshaun $O(bd)$ solmua onkin hyvä likiarvo tilavaatimukselle.

Algoritmin **IDA*** aikavaatimus riippuu vahvasti siitä *montako eri arvoa funktio f saa ennen kuin saavutetaan tavoitearvo f^**. Tämä näet kertoo iterointien lukumäärän.

Esimerkiksi luvun 3.2.3 heuristiikalla h_2 on vain muutama mahdollinen arvo, jolloin tyypillisesti f muuttuu vain muutaman kullakin ratkaisupolulla. Silloin myös **IDA*** iteroi vain muutaman kerran.

Pahimmillaan f saattaa antaa eri arvon jokaiselle etsintäpuun solmulle, jolloin jos **A*** käsittelisi N solmua, niin **IDA*** käsittelisikin

$$1 + 2 + 3 + \cdots + N = \frac{N(N+1)}{2}$$

solmua.

Jos N oli niin suuri, ettei **A*** mahtunut toimimaan koneen muistissa, niin $O(N^2)$ on vuorostaan liian kauan!

Tämä voidaan korjata syventämällä etsintää joka kierroksella vähintään ennalta valitun vakion $\epsilon > 0$ verran.

Tällöin korjatulle algoritmille kierroksia tulee

$$O\left(\left\lceil \frac{f^*}{\epsilon} \right\rceil\right)$$

kappaletta.

Toisaalta korjattu algoritmi saattaa vastata minkä tahansa niistä ratkaisuista, joiden hinta on $\leq f^* + \epsilon$.

Toisin sanoen, tällöin korjattu algoritmi hyväksyy vakion ϵ verran epäoptimaalisuutta.

Tällaista "hallitusti leväperäistä" algoritmia kutsutaan ϵ-*luvalliseksi*.

RBFS

Unohdetaan hetkeksi luvun 3.2.2 vaatimus, että heuristikan h pitäisi olla luvallinen.

Silloin etsintä voi tuottaa *iloisia yllätyksiä:* jokin alipuu saattaakin tarkemmin tutkittaessa osoittautua lupaavammaksi — pienempiä f-arvoja omaavaksi — kuin h sen juuressa epäili.

Välitetään nämä hyvät uutiset puussa ylöspäin *peruutusarvoina (backed-up values):*

- Kun solmu n laajennetaan, kunkin syntyvän lehden m_i alkuperäinen peruutusarvo on alkuperäisen ohjausfunktion arvo $f(m_i)$.
- Sisäsolmun peruutusarvo on aina sen *lapsisolmujen peruutusarvoista pienin*.
  Sisäsolmussa on siis tieto sen *parhaasta jälkeläisestä* eikä enää siitä itsestään.
- Algoritmina:
  Kun n on laajennettu, korvataan sen vanha peruutusarvo sen lasten peruutusarvojen minimillä.
  Siirrytään solmun n isäsolmuun, ja tehdään sielläkin samoin.

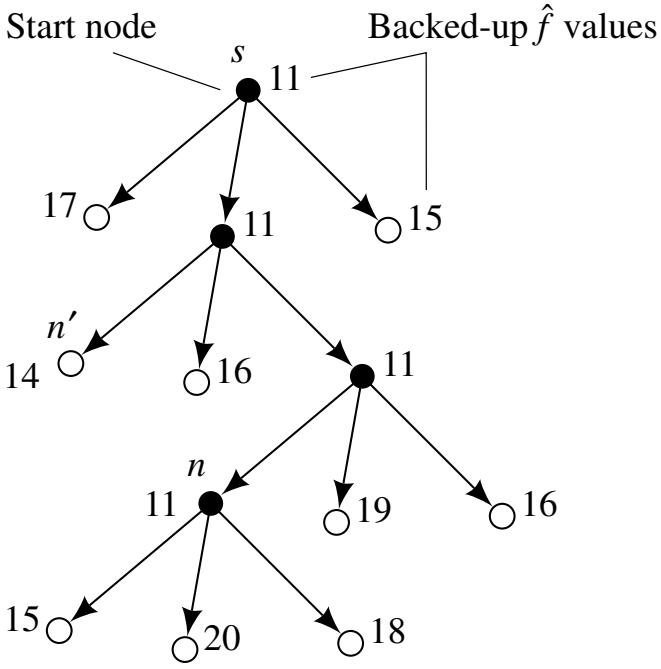
Algoritmin **RBFS** muistinsäästöstrategia on *pitää muistissa sisäsolmuista vain ne, joiden kautta on saatu parhaat uutiset*.

Algoritmina:

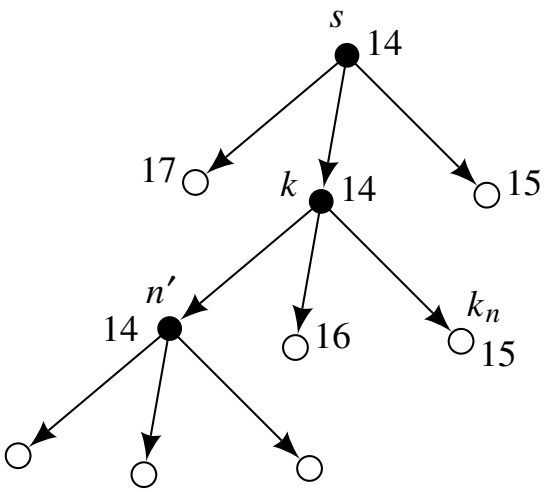
- Jos peruutusarvoja päivitettäessä päädytään sellaiseen sisäsolmuun k_n, jonka isäsolmu k saa jostakin toisesta lapsisolmustaan n' vielä parempia uutisia kuin tästä solmusta k_n saadut uutiset u, niin:
- Lopetetaan peruutusarvojen päivitys tähän.
  Solmun n' paremmat uutiset on jo aiemmin kerrottu koko jäljellä olevalle polulle.

- Tehdään sisäsolmusta k_n taas lehti poistamalla siitä alkanut alipuu T.
- Lehden k_n peruutusarvoksi jää se peruutusarvo u joka siihen puusta T ehti nousta.
- Seuraavaksi laajennetaan lupaavin lehtisolmu n'.

Jos myöhempi etsintä toteaakin, että solmusta n' laajennetun puun T' tuottamat lisäuutiset ovatkin huonompia kuin solmun k_n uutiset u, niin etsintä poistaa puun T' ja palaa jatkamaan etsintää solmusta k_n.



(a) RBFS has just expanded node n but has not yet backed up the $\hat{f}$ values of its successors



(b) $\hat{f}$ values have been backed up, the subtree below k_n has been discarded, and search continues below n'

© 1998 Morgan Kaufman Publishers

Kuva 36: Esimerkki algoritmin **RBFS** toiminnasta [Nil98, kuva 9.9].

Algoritmi **RBFS**

- muistaa hieman enemmän
- laajentaa vähemmän

solmuja kuin algoritmi **IDA***.

Toinen tapa päätyä algoritmiin **A*** onkin kysyä "millaisilla h tämä peruutusarvojen välitys on turhaa?"

SMA*

Algoritmin **IDA*** ongelmana on, että kustakin iteraatiosta muistetaan vain yksi luku $f\text{-}limit$, vaikka muistia olisi pidempäänkin etsintään.

Tehdään siis algoritmista **A*** sellainen versio **SMA* (Simplified Memory-Bounded A*)**, jossa annetaan etukäteen *yläraja muistissa pidettävien solmujen lukumäärälle*.

Jos tämä raja on jo saavutettu, ja muistiin pitäisi lisätä uusi solmu, niin jokin muistissa jo oleva solmu luovuttaa sille oman paikkansa ja poistetaan muistista.

Tällöin tarjotusta muistista otetaan kaikki hyöty irti:

- **SMA*** pitää kaiken sille annetun muistin koko ajan täynnä.
  IDA* tyhjentää muistinsa iteraatioiden välillä.
- **SMA*** laajenee tarvittaessa eliminoimaan toistuvia tiloja luvun 3.1.7 globaalilla menetelmällä 2 siinä määrin kun rajalliseen muistiin mahtuu.
  Syvyyssuuntaisena **IDA*** laajenee vain lokaaliin toistojen poistoon menetelmällä 1.
- **SMA*** on täydellinen, jos muistia on tarpeeksi *matalimpaan (shallowest)* ratkaisuun vievälle polulle.
- **SMA*** on optimaalinen, jos muistia on tarpeeksi matalimpaan *optimaaliseen* ratkaisupolkuun.
  Muuten palautetaan paras ratkaisu joka vielä mahtui kerralla muistiin.
  Samalla voidaan palauttaa varoitus, ettei ratkaisu ehkä ole paras.
- **SMA*** on optimaalisen tehokas kuten **A*** luvussa 3.2.2, jos muistia on riittävästi.
  Ei kuitenkaan vielä tiedetä onko **SMA*** optimaalisen tehokas kaikkien muistirajoitettujen algoritmien joukossa.

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
    if Queue is empty then return failure
    n ← deepest least-f-cost node in Queue
    if GOAL-TEST(n) then return success
    s ← NEXT-SUCCESSOR(n)
    if s is not a goal and is at maximum depth then
        f(s) ← ∞
    else
        f(s) ← MAX(f(n), g(s)+h(s))
    if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
    if SUCCESSORS(n) all in memory then remove n from Queue
    if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
    insert s on Queue
  end
```

Algoritmi 6: Muistirajoitettu **A*** (**SMA***) [RN95, kuva 4.12].

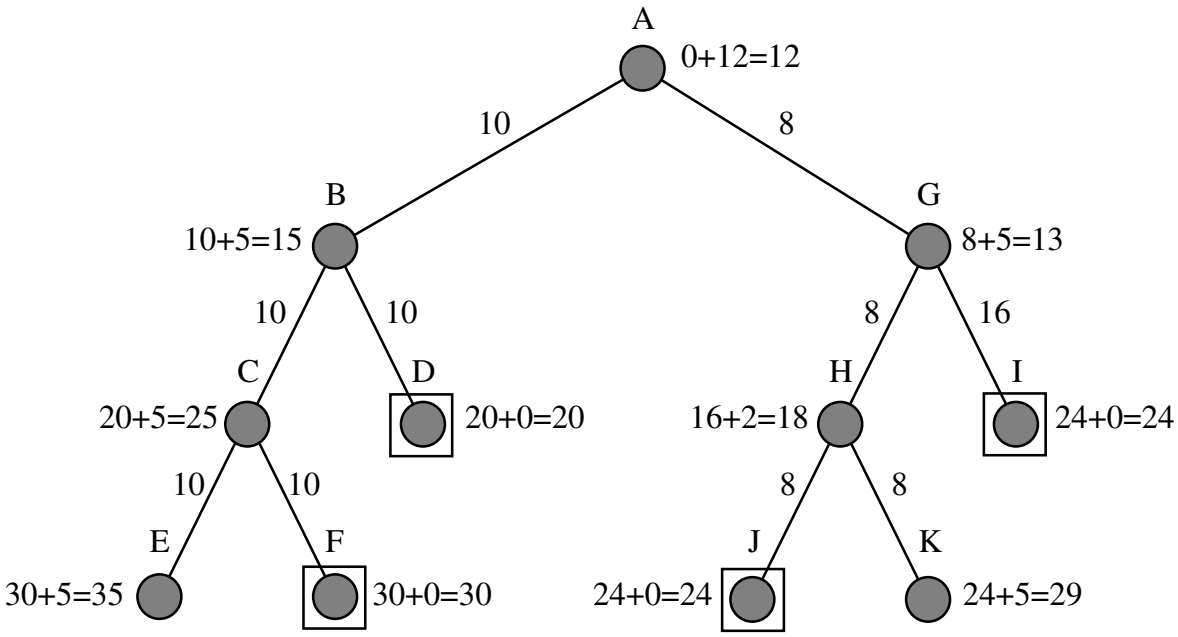
- **SMA*** päivittää solmujensa peruutusarvoja kuten **RBFS**.
  Merkitään solmun n nykyistä peruutusarvoa $f(n)$.
- **SMA*** pitää solmussa n kirjaa myös parhaasta luovutetun lapsen peruutusarvosta.

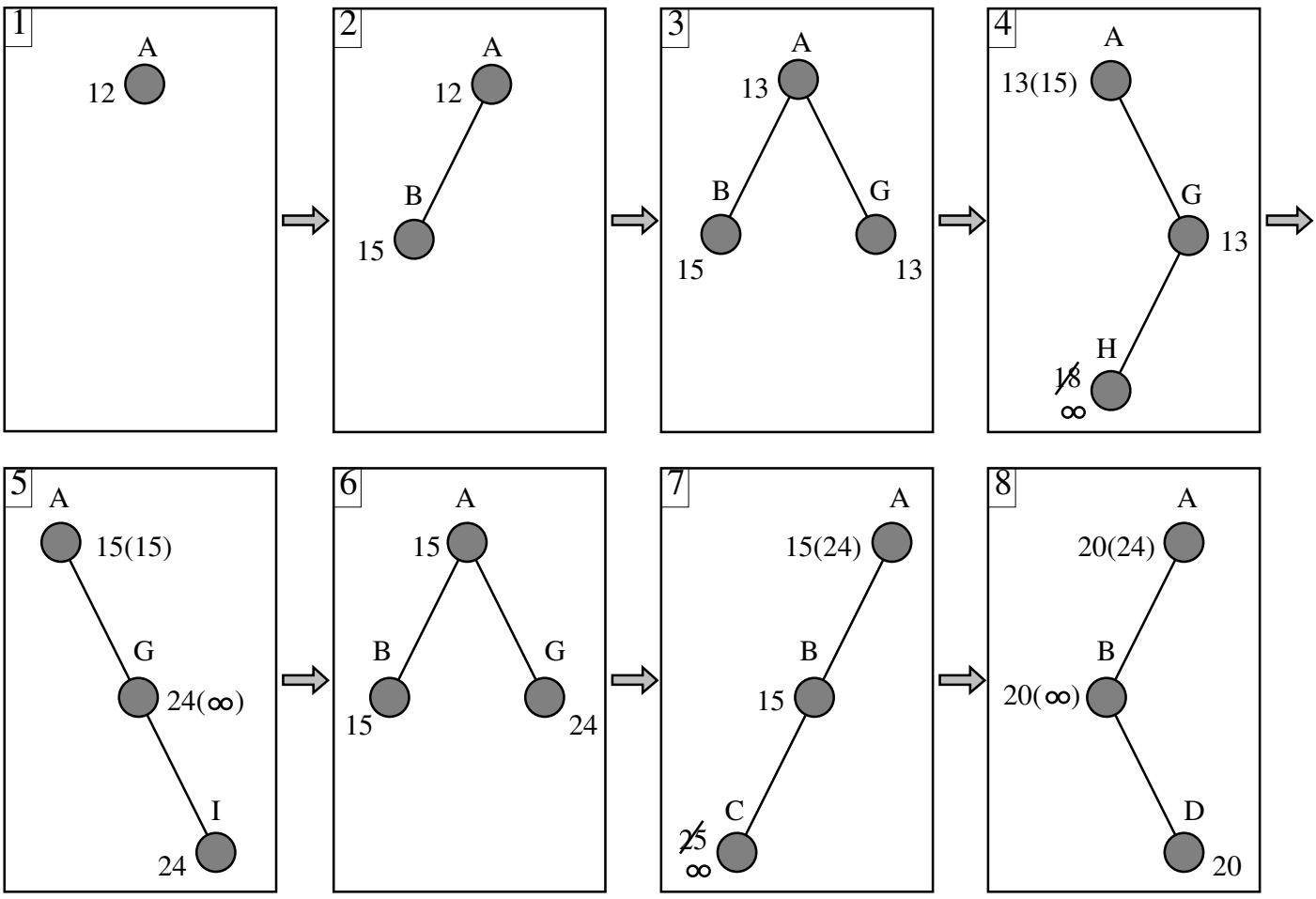
Algoritmin **SMA*** muistinluovutusstrategia on seuraava:

1. Valitse luovuttajaksi lehtisolmu, jonka f-arvo on suurin.
   Sehän on vähiten lupaava nykyisistä lehdistä.
2. Jos vaihtoehtoja on useita, niin valitse mahdollisimman matala.
   Muistiin yritetään näin mahduttaa mahdollisimman pitkä polku.

Algoritmin **SMA*** solmunlaajennusstrategia on seuraava:

1. Lisää seuraava puuttuva lapsi sille solmulle, jonka f-arvo on pienin.
2. Jos vaihtoehtoja on useita, niin valitse mahdollisimman syvä.





Kuva 37: Algoritmin **SMA*** eteneminen [RN95, kuva 4.12].

Kuvan 37 esimerkki on seuraava:

1. Juuren A ensimmäinen (vasen) lapsi B ilmestyy muistiin.
2. Nyt $f(A) < f(B)$, joten A laajenee toisen kerran ja saa myös jälkimmäisen (oikean) lapsensa G muistiin.

   Koska juuri A on nyt laajennettu kokonaan, asetetaan $f(A) = \min(f(B), f(G)) = 13$.

   Kolmen (3) solmun kokoinen muisti täyttyy.
3. Seuraavaksi laajennetaan G.
4. Koska muisti on täynnä, täytyy strategian nojalla isoveljen B luopua muistipaikastaan ensimmäisen veljenpoikansa H edestä.

   Luopuvan solmun B f-arvo 15 viedään sen isään A (sulkuihin).

   Valitettavasti lisätty H on maksimisyvyydellä, eli yhtä syvällä kuin muistin koko, eli muistiin mahtuu vain polku juuresta A tähän solmuun H.

   Koska H ei ole maalisolmu, asetetaan $f(H) = \not{1}8 = \infty$ eli "muisti ei riitä jatkamaan etsintää solmusta H eteenpäin".
5. G valitaan vielä kerran laajennettavaksi, ja muistin luovuttaa vasta lisätty H.

   Muistiin tulee I, joka on jälleen maksimisyvyydellä mutta nyt maalisolmu, eli säilyttää oman arvonsa $f(I) = 24$.

   Samalla solmun G molemmat lapset on nähty, joten asetetaan $f(G) = \min(\infty, f(I)) = 24$.

   Tämä muutos kerrotaan ylöspäin sen isään A, jossa $f(A) = \min(15, f(G)) = 15$ saadaan (suluissa olevasta) luopuneen lapsen B arvosta ja muutoksesta.

   Ratkaisu $A \to G \to I$ olisi nyt muistissa, mutta $f(A) = 15$ vihjaa, että parempaankin saatetaan vielä pystyä.
6. Lähdetään peräämään tätä lupausta laajentamalla juuren A välillä muistista poistunut vasen lapsi B uudelleen.

   Muistin luovuttaa I, ja näin unohtuu äskeinen ratkaisu.

   Solmujen A ja G f-arvot ovat kasvaneet sitten vaiheen 3, joten algoritmi ei ole ikuisessa silmukassa.

Lisäksi solmun A kaikki lapset (B ja G) ovat taas muistissa, joten (suluissa oleva) kirjanpito luopuneiden lasten parhaasta f-arvosta pyyhitään pois.

7. Solmun B vasen lapsi C tuodaan muistiin, jonka luovuttaa G.

   Tämä luopuminen asettaa sen isän A luopuneiden lasten (suluissa olevaksi) pienimmäksi f-arvoksi $f(G) = 24$.

   C on kuitenkin maksimisyvyydellä muttei maalisolmu, joten asetetaankin $f(C) = \not{2}\not{5} = \infty$.

8. Seuraavaksi lisätään solmun B oikea lapsi D muistiin, jonka luovuttaa C.

   Luovutus lähettää arvon $f(C) = \infty$ (sulkuihin) sen isään B, mutta sen isoisässä A on pienempi arvo 24, joka ei muutu.

   D on maksimisyvyydellä mutta maalisolmu, joten se säilyttää oman arvonsa $f(D) = 20$.

   Koska sen isän B molemmat lapset on nyt käsitelty, asetetaan $f(B) = \min(\infty, 20) = 20$.

   Tämä muutos etenee myös sen isoisään A, jossa asetetaan $f(A) = \min(24, 20) = 20$.

9. Lopuksi valitaan laajennettavaksi D, joka on maalisolmu, ja etsintä päättyy vastaukseen $A \to B \to D$, joka onkin parempi kuin vaiheen 5 ratkaisuehdokas.

Esimerkistä nähdään, että jo nyt mutkikas algoritmin listaus 6 ei vielä sisälläkään läheskään kaikkia yksityiskohtia. . .

Jos muistia on edes kohtuullisesti, **SMA*** pystyy ratkomaan paljon mutkikkaampia ongelmia kuin **A***.

Hyvin vaikeissa ongelmissa **SMA*** alkaa hyppelehtiä ratkaisupolulta toiselle, jolloin se tuhlaa aikaa tuomalla samoja solmuja muistiin yhä uudelleen ja uudelleen.

Vaikka aika vs. tila -kysymykset ovatkin vaikeita, näyttää siltä, että jos halutaan optimiratkaisu, niin (ainakin) toista kuluu. . .

4. Lisää ongelmanratkaisumenetelmiä

Jatketaan luvussa 3 esiteltyä etsintäteemaa sellaisissa ongelmissa joissa

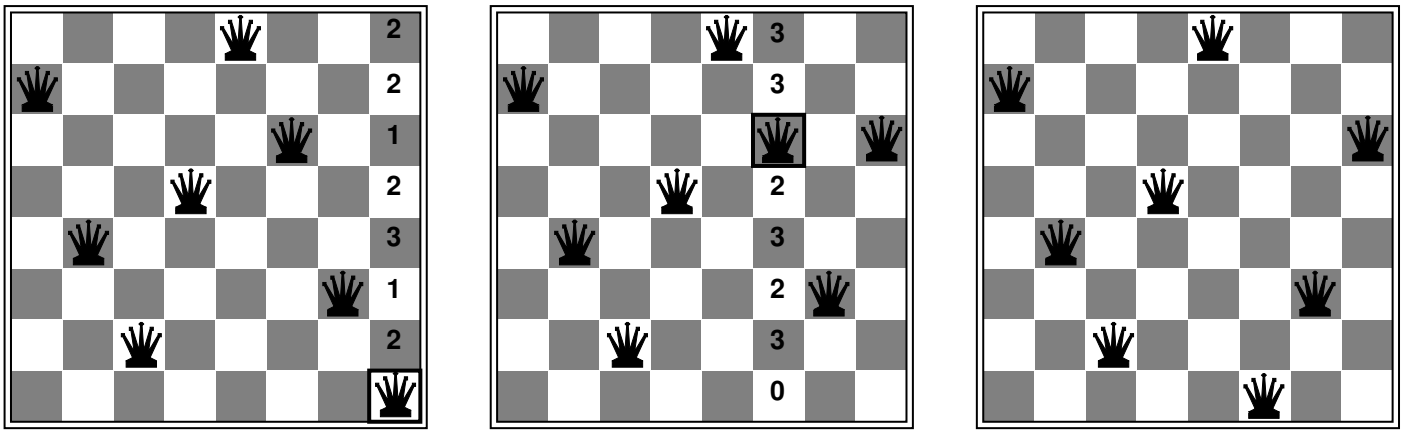
1. vain itse ratkaisu kiinnostaa, ei siihen johtanut tekojono
2. maaliehto ei enää olekaan "musta laatikko", vaan jakautuu erillisiin osaehtoihin, jotka voidaan hoitaa yksi kerrallaan
3. on mukana etsijän lisäksi pahantahtoinen vastustaja, kuten peleissä
4. on mukana etsijän vastustajien lisäksi välinpitämätön "luonto", kuten noppa onnenpeleissä.

Tällöin käytetään erilaisia etsintämenetelmiä kuin aiemmin:

1. Iteratiivisia parannusmenetelmiä.
2. Rajoitteiden levitystä.
3. Etsintää pelipuissa.
4. Etsintää todennäköisyyksin varustetuissa pelipuissa.

4.1. Iteratiiviset parannusmenetelmät [RN95, luku 4.4] [Nil98, luvut 11.1-2]

Esimerkiksi 8 kuningattaren (shakki)ongelmassa ei ratkaisun saavuttanutta tekopolkua kysytä; vain lopputuloksen laatu ratkaisee.



Kuva 38: 8 kuningattaren ratkaiseminen [RN95, kuva 4.16]. (Ruudun numero kertoo moniko sitä uhkaa.)

Tällöin on usein käytännöllisintä *tehdä alkutilaan toistuvasti pienia paikallisia korjauksia* ja toivoa, että näin päästäisiin ratkaisuksi kelpaavaan tilanteeseen.

Esimerkiksi VLSI-piirien optimoinnissa jo alkutila on kelvollinen piiri, jonka komponenttien paikkoja aletaan vaihdella, ja näin yritetään minimoida tarvittavien johtojen pituuksia.

Nämä menetelmät yrittävät *optimoida (minimoida/maksimoida) kohdefunktiota (evaluation function)*, joka mallintaa etäisyyttä ratkaisuun tai nykyisen ratkaisun hyvyyttä.

Menetelmien periaate selviää kuvasta 39: piirretään kohdefunktion $\textsc{Value}: \mathbb{R}^2 \mapsto \mathbb{R}$ kuvaaja, ja sijoitetaan nykyinen tila sinne.

Saadaan alppimaisemassa vaeltaja, joka yrittää nousta vuoren huipulle (tai minimointitehtävässä laskeutua laakson pohjalle), vaikka hän. . .

- on niin *likinäköinen*, että näkee vain yhden askeleen verran ympärilleen
- kärsii Korsakovin syndroomasta eli *ei pysty painamaan mitään asioita muistiinsa*.

Yksinkertainen strategia on katsella ympärilleen tuon yhden askeleen verran, ja ottaa niistä *se askel, joka vie korkeimmalle*.

Näin saadaan *vuoristokiipeilyalgoritmi 7 (Hill-Climbing Algorithm).* Se pysähtyy, kun ylöspäin ei enää päästä, ja ilmoittaa lötäneensä huipun.

Minimointitehtävissä tätä (jyrkimmin alaspäin menevää) menetelmää kutsutaan *gradienttilaskeutumiseksi (Gradient Descent)*.

Koulumatematiikassa funktion $f : \mathbb{R} \mapsto \mathbb{R}$ *derivaatta* f' (alias $\mathbf{D}f$, $f\frac{x}{dx}$,. . .) pisteessä $x_0 \in \mathbb{R}$ oli geometrisesti:

- Piirrä funktion $y = f(x)$ kuvaaja κ 2-ulotteisessa (suorakulmaisessa) koordinaatistossa.
- Piirrä kuvaajalle κ se *tangenttisuora* τ joka kohtaa kuvaajan pisteessä x_0.
- Derivaatan arvo $f'(x_0)$ on *tangenttisuoran τ kulmakerroin*. . .
- eli *funktion $f(x)$ hetkellinen muutosnopeus kohdassa x_0.*
  Esimerkiksi fysiikassa (hetkellinen)
  - nopeus on paikan
  - kiihtyvyys on nopeuden

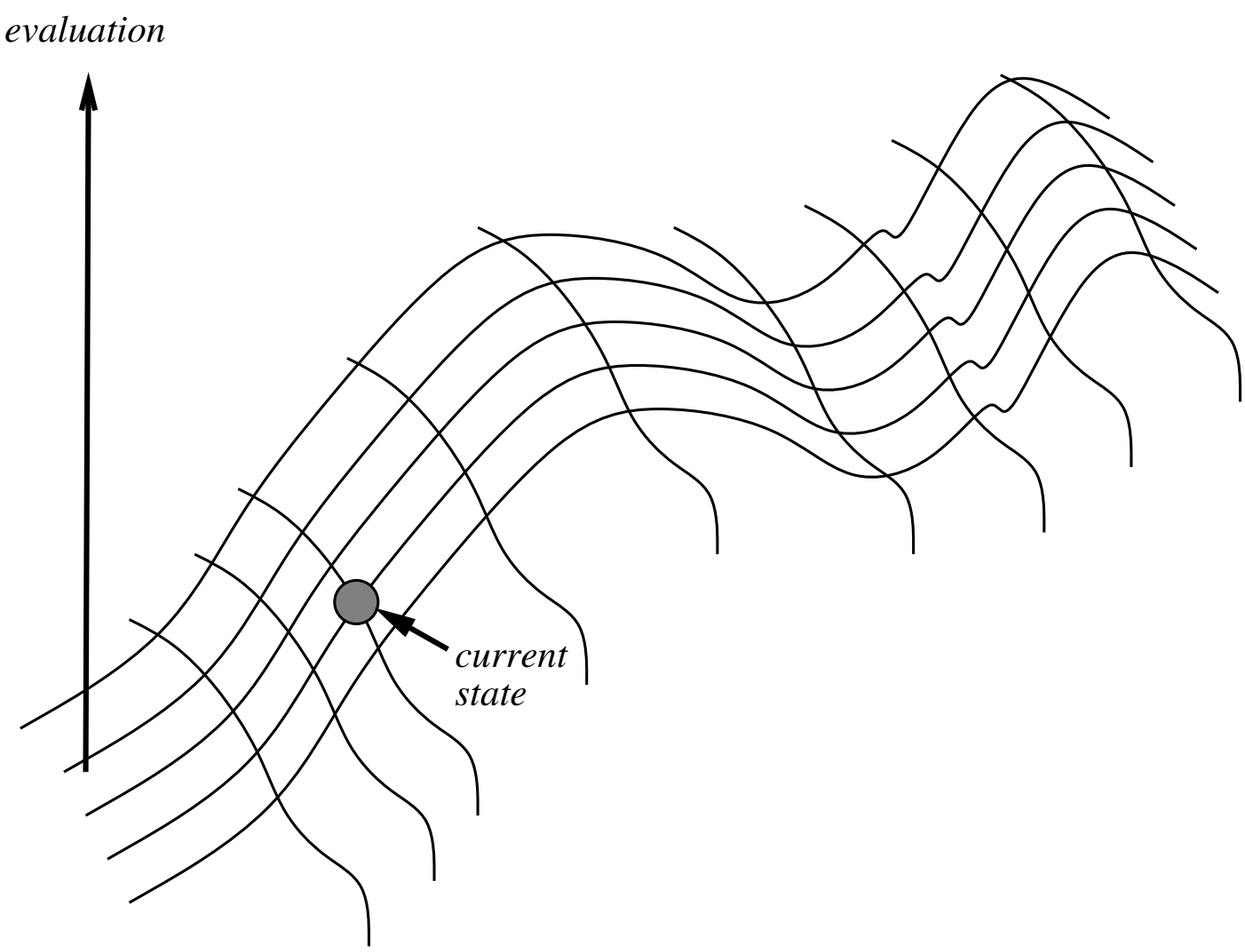
  derivaatta ajan suhteen.

Matematiikan kurssin "Differentiaali- ja integraalilaskenta II" gradientti on sama idea m-ulotteisten avaruuksien funktioille $g : \mathbb{R}^m \mapsto \mathbb{R}$.

3-ulotteisessa kuvassa 39 on maiseman $z = g(x, y)$ kuvaaja K.

Funktion arvon z hetkellinen muutosnopeus ja -suunta saadaan nyt siitä *tangenttitasosta* T, joka kuvaajalla K on nykyisessä pisteessä $current\ state = \langle x_0, y_0 \rangle$.

Maisemafunktion g gradientti on *tangenttitason T normaalivektori*.



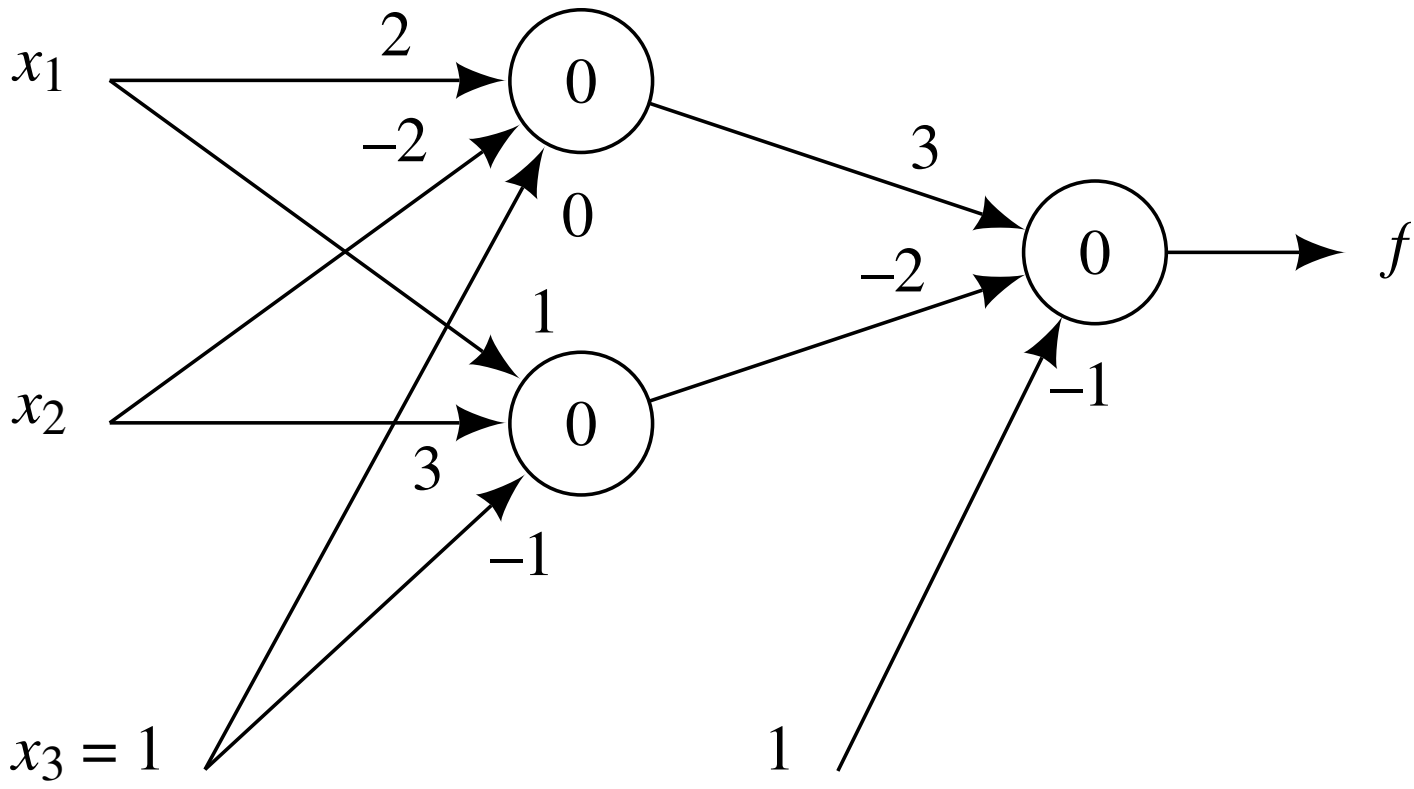
Kuva 39: Optimoitavan funktion maisemaa [RN95, kuva 4.13].

```
function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
          next, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
     next ← a highest-valued successor of current
     if VALUE[next] < VALUE[current] then return current
     current ← next
  end
```

Algoritmi 7: Vuoristokiipeily [RN95, kuva 4.14].

Gradienttilaskeutumisella voi siis suorittaa etsintää *jatkuvissa* avaruuksissa kuten reaalilukuja sisältävissä ongelmissa. (Luvun 3 mentelmät olivat diskreettejä.)



© 1998 Morgan Kaufman Publishers

Kuva 40: Pieni monitasoinen hermoverkko [Nil98, kuva 3.6].

Luvussa 1.5 mainittu monisoluisten hermoverkkojen oppimisalgoritmi on gradienttilaskeutumisen eräs sovellus:

- Perustehtävässä annetaan
  - hermoverkko H jonka kaarien (reaali)painoja voi muutella
  - esimerkkiaineisto A joka verkon H pitää oppia.
    Aineisto on valmis lista pareja $\langle$annettu syöte, haluttu tulos$\rangle$.
- Minimoitava funktio g mittaa sitä *virhettä*

$$|H\,(\text{annettu syöte}) - \text{haluttu tulos}|$$

  jonka nykyinen H aineistosta poimitulla esimerkillä tekee.
  Yleensä valitaan mitaksi virheen neliö.

- Maisema $\mathcal{M}$ koostuu kaikista mahdollisista H:
  Jokainen muuttuva kaaripaino on oma ulottuvuutensa.
- Johdetaan gradientin lauseke funktio(i)lle $f_a : \mathcal{M} \mapsto \mathbb{R}$ joka kertoo, miten kaaripainoja pitää päivittää, kun halutaan muuttaa nykyinen verkko sellaiseksi, joka muistaa (myös) seuraavan esimerkin a.
  Esimerkki a on funktiossa ikään kuin tuntematon vakio.
- Opettamisessa
  1. valitaan jokin esimerkki
  2. muistetaan se päivittämällä kaaripainoja gradienttisäännöllä

  kunnes jäljelle jää enää pieni virhe.

Vuoristokiipeilyalgoritmilla 7 on kolme tunnettua ongelmamaastoa:

Paikallinen huippu (Local Maximum) on pieni ympäristöään korkeampi mäennyppylä.
Vaeltaja kiipeää sille luullen päässeensä koko vuoren huipulle, koska sen ympärillä on pelkkää alamäkeä niin pitkälle kuin vaeltajan lähes sokea silmä kantaa.

Tasanko (Plateau) on alue, jolla korkeus ei vaihtele niin pitkälle kuin silmä kantaa.
Tasangolla vaeltajamme hoipertelee *satunnaiskävelyn (random walk)*.

Harjanne (Ridge) viettää sekä "oikealle" että "vasemmalle" jyrkästi alaspäin.
Jos vaeltajan askelvalikoimasta puuttuu juuri jyrkänteen harjaa "eteenpäin" vievä suunta, niin vaeltaja kulkee epätaloudellisesti hyppien rinteen puolelta toiselle.

Kaikissa tapauksissa vaellus saavuttaa sellaisen maaston, jossa etenemistä ei enää tapahdu tai se on tuskallisen hidasta.

Yksi tapa yrittää välttää näitä ongelmia on *aloittaa satunnaisesti (Random-Restart)* uusi vaellus jostakin toisesta etsintäavaruuden satunnaisesta kohdasta sen jälkeen kun edellinen on pysähtynyt, ja kokeilla löytääkö se aiempaa korkeamman huipun.

Milloin satunnaisen aloittamisen voi lopettaa?

- Kun vaelluksia on tehty ennalta kiinnitetyt L kappaletta.
- Kun uusia entistä korkeampia huippuja ei enää ole löytynyt viimeisten L vaelluksen aikana.

Optimiratkaisun löytyminen on

- todennäköistä kun L on "riittävän suuri"
- nopeaa jos maaston huippuja on vähän ja ne erottuvat selvästi ympäristöstään.

Vaikeiden etsintäongelmien maasto

- sisältää eksponentiaalisen monta paikallista huippua mihin takertua ja mitä väistellä
- eli on piikikäs kuin siilin selkä.

Simuloitu jäähdytys/mellotus (Simulated Annealing) tarjoaa systemaattisemman tavan väistellä paikallisia huippuja.

Simuloidussa jäähdytyksessä tarkastellaan joka hetki *satunnaista* mahdollista askelta $next$:

- Jos $next$ veisi ylöspäin, niin otetaan se.
- Jos $next$ veisikin *alaspäin*, niin *arvotaan* otetaanko se vai jäädäänkö paikoilleen.

  Tässä arvonnassa askeleen $next$ todennäköisyys on sitä pienempi mitä
  - suurempi on vastaava pudotus ΔE, ja
  - mitä kauemman aikaa t vaellus on jo kestänyt.

Silloin vaelluksen

alussa pyritään väistelemään paikallisia huippuja laskeutumalla usein alaspäin

lopussa on todennäköisesti päädytty jonkin korkean huipun lähistölle, joten aletaan yrittämään useammin sitä ylöspäin.

Siirtyminen alkuvaiheesta loppuvaiheeseen on *asteittainen*.

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  static: *current*, a node
    next, a node
    T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for *t* ← 1 **to** ∞ **do**
    T ← *schedule*[*t*]
    if *T*=0 **then return** *current*
    next ← a randomly selected successor of *current*
    Δ*E* ← VALUE[*next*] – VALUE[*current*]
    if Δ*E* > 0 **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Algoritmi 8: Simuloitu jäähdytys [RN95, kuva 4.15].

Asteittaiseen menetelmään päädyttiin *fysiikan* kautta:

1. Lämmitetään kiinteää rautamöykkyä R sellaiseen korkeaan lämpötilaan T, jossa se sulaa hyvin juoksevaksi nesteeksi.

   Tehdään tämä upottamalla R sitä itseään paljon suurempaan lämpöhauteeseen, joka on lämmitetty lämpötilaan T.
2. Silloin möykyn R jokainen atomi a pääsee liikkumaan (melko) vapaasti.
3. "Luonnonlakien mukaan" kukin atomi a on satunnaisessa lämpöliikkeessä, jossa atomin a oma energiatila voi

   nousta ottamalla lämpöenergiaa hauteesta (joka ei moisesta kylmene)

   laskea luovuttamalla lämpöenergiaa hauteeseen (joka ei moisesta lämpene).

Energiatilan laskeminen on nousemista todennäköisempää.

4. Näin koko R todennäköisesti "rauhoittuu" aikanaan tilaan, jossa valtaosa sen atomeista a on energiaminimissään.
   Tämä energiaminimi riippuu nykyisestä lämpötilasta T.
5. Jäähdytetään rautaa R laskemalla hauteen lämpötilaa T hieman.
   Odotetaan taas että valtaosa sen atomeista a ehtii jälleen etsiytyä uuteen energiaminimiinsä.
6. Jäähdytystä jatketaan varovasti, kunnes R on taas kiinteää.
   Nyt sen atomit a ovat hyvin todennäköisesti hyvin lähellä kiinteän olomuodon energiaminimiään, eli raudan luontaista hilarakennetta.

Näin saadaan *meltorautaa*, jonka pitäisi olla rakenteeltaan hyvin homogeenistä sekä vapaata sisäisistä jännityksistä ja (hius)halkeamista.

Simuloitu jäähdytys saadaan tästä prosessista siten, että kohdefunktion vastaluku $-\textsc{Value}(R)$ kertoo möykyn R *kokonaisenergian* joka pyrkii siis minimoitumaan.

Jos. . .

- jäähdytys aloitetaan "tarpeeksi kuumasta" alkulämpötilasta, ja
- lämpötilaa lasketaan "tarpeeksi hitaasti"

niin. . .

- simuloitu jäähdytys päätyy lopuksi globaaliin maksimikohtaan todennäköisyydellä 1

mutta. . .

- "todennäköisyydellä 1" ei ole sama asia kuin "varmasti", ja
- vastaava *jäähdytysstrategia* (jonka määrää algoritmin 8 funktio $schedule$) on eksponentiaalisen hidas.

Simuloitu jäähdytys on osoittautunut *käytännössä tehokkaaksi optimointimenetelmäksi* varsinkin silloin kun ongelma on hyvin suuri, esimerkiksi mutkikkaiden VLSI-piirien optimoinnissa.

Eräänlainen iteratiivinen parannusmenetelmä ovat myös *geneettiset algoritmit (Genetic Algorithms, GA)*:

- Niissä simuloidaan jäähtymisen sijasta
  - geenien sekoittumista suvullisessa lisääntymisessä
  - mutatoitumista yksilöissä.
- Simuloitu luonnonvalinta ohjaa populaatiota sukupolvi sukupolvelta soveltumaan paremmin ympäristöönsä eli ratkaistavaan ongelmaan.

Näitä menetelmiä ei kurssilla kuitenkaan käsitellä.

4.2. Rajoiteongelmat [RN95, luvut 3.7, 4.2 ja 4.4] [Nil98, luvut 11.3-4]

Rajoiteongelmissa luvun 3 maaliehdon tiedetään jakautuvan konjunktioksi paikallisia ehtoja ("ehto 1 **ja** ehto 2 **ja** . . . ") joita voidaan tarkastella osin erikseen.

Kuvan 38 mukainen 8 kuningattaren ongelma voidaan muotoilla *rajoiteongelmana (Constraint Satisfaction Problem, CSP)* seuraavasti:

1. Tavoitteena on etsiä sopivat arvot *muuttujille* $X_1, \ldots, X_8$.
   Muuttujan X_i arvo kertoo silloin sarakkeelle i asetetun kuningattaren i rivin.
   Muuttujia on aina äärellinen määrä.
2. Kukin muuttuja saa arvokseen rivinumeron

$$X_i \in D_i = \{1, \ldots, 8\} .$$

   Joukkoa D_i kutsutaan muuttujan X_i *arvoalueeksi (domain)*.
   Arvoalue voi yleisessä tapauksessa olla myös ääretön kuten $\mathbb{Q}$ tai jopa jatkuva kuten $\mathbb{R}$.
   Jokaista tällaista *yhtä* muuttujaa koskevaa ehtoa "$X_i \in D_i$" kutsutaan *unaariseksi rajoitteeksi (unary constraint)*.
3. Kaikille $i < j$ pitää päteä, että kuningattaret ruuduissa X_i ja X_j eivät uhkaa toisiaan.
   Jokaista tällaista *kahta* muuttujaa koskevaa ehtoa kutsutaan *binääriseksi* rajoitteeksi.
   Sitä voidaan ajatella karteesisen tulon $D_i \times D_j$ sellaisena osajoukkona $R_{i,j}$, joka luettelee kaikki sallitut parit.

Rajoitteet voivat yleisessä tapauksessa käsittää myös 3 tai useampia muuttujia kerrallaan.

Jos kuitenkin kaikki arvoalueet ovat äärellisiä, niin useampimuuttujaiset rajoitteet voidaan korvata sopivilla uusilla unaari- ja binäärirajoitteilla.

Rajoite voi olla

ehdoton ***(absolute)*** jolloin ratkaisun on pakko toteuttaa se (kuten 8 kuningattaren ongelmassa)

suositus ***(preference)*** jolloin voidaan hyväksyä myös sellainen ratkaisu, joka ei sitä täytä, jos parempaan ei pystytä.

Esimerkiksi laitoksen opetusohjelmaa laadittaessa

ehdoton rajoite on, että samaan saliin ei voi sijoittaa samaan aikaan kahta eri luentoa

suositus on, että cum laude -tason pakollisten kurssien luentoja ei sijoitettaisi samaan aikaan.

Keskitytään ratkomaan *ehdottomista äärellisistä unaari- ja binäärirajoitteista koostuvia ongelmia* oppimillamme menetelmillä.

Luvun 3.1 systemaattisia etsintämenetelmiä voidaan soveltaa seuraavasti:

- Tavoitteena on löytää arvot äärelliselle määrälle muuttujia $X_1, \ldots, X_N$.
  Tilannetta voi ajatella etsintäpuuna, jossa syvyydellä d etsitään arvoa muuttujalle X_{d+1}, ja mahdolliset teot ovat D_{d+1}.
  Voidaan käyttää luvun 3.1.3 *syvyyssuuntaista etsintää*:
  nyt se pysähtyy itsestään viimeistään syvyydellä N (eli on täydellinen).
  Kuningataresimerkissämme $N = 8$.
- Olkoon syvyysetsintä jo antanut arvot muuttujille X_i, missä $1 \leq i \leq j$.
  Valitaan seuraavalle muuttujalle X_{j+1} sopivaa arvoa $V \in D_{j+1}$.

Ensin voidaan tutkia kaikki binäärirajoitteet muuttujien X_i ja X_{j+1} välillä, ja *karsia* joukosta D_{j+1} pois ne ehdokkaat V jotka rikkoisivat jonkin niistä.

Jos yhtään ehdokasta ei jää jäljelle, voidaan *peruuttaa (backtrack)* etsinnässä jo nyt: koko solmusta X_{j+1} alkava alipuu voidaan jättää tutkimatta, koska jo siihen vievä polku on havaittu mahdottomaksi.

Näin rajoiteongelmissa voimme kysyä väliaikatietoja "Onko tämä keskeneräinen ratkaisu oikein tähän saakka?". Se tehostaa etsintää yleisiin ongelmiin verrattuna.

Esimerkissämme tämä strategia tarkoittaa, että kuningatar $j + 1$ laitetaan aina sellaiseen ruutuun, jota mikään jo laitetuista kuningattarista $1, \ldots, j$ ei uhkaa.

- Kääntäen voimme kuningatarta $j + 1$ asettaessa merkitä samalla ne ruudut, joita se uhkaa.

  Silloin voimme katsoa, jääkö laudalle tarpeeksi uhattomia ruutuja lopuille kuningattarille $j + 2, \ldots, 8$, kun kuningatarten $1, \ldots, j + 1$ uhkaamat ruudut on poistettu.

  Tätä strategiaa kutustaan *eteenpäin tarkistamiseksi (forward checking)*.

  Sen avulla voidaan käytännössä ratkoa esimerkkiongelmamme laajennuksia aina noin tapaukseen $N = 30$ saakka (eli sovittaa N kuningatarta $N \times N$ ruudun laajennetulle shakkilaudalle).

  Kun siis valitaan jokin $X_{j+1} = V$, niin samalla poistetaan kaikista binäärirajoitteista $R_{j+1,j+k}$, missä $k > 1$, kaikki parit jotka eivät ole muotoa $\langle V, W \rangle$ (missä siis $W \in D_{j+k}$). Jos jokin niistä tyhjenee, niin voidaan taas peruuttaa.

- Tämä poistostrategia on *kaarikonsistentiksi (arc consistency) pakottamista*:

  Etsinnän tila on kaarikonsistentti jos jokaisen muuttujan X_i arvoalueessa D_i on edelleen jäljellä jokin sellainen arvo V_i, joka kelpaa kaikille muuttujaan X_i liittyville rajoitteille.

  Tai kääntäen, $V_i \notin D_i$ jos yksikin muuttujaan X_i liittyvä rajoite kieltää muuttujalta X_i arvon V_i.

  Toisin sanoen, D_i sisältää joka hetki vain ne muuttujan X_i

arvot V_i jotka ovat yhä *mahdollisia, kun ongelmaa katsellaan vain muuttujan X_i lähistöltä*, ei kokonaisuutena.
  - Monimutkaisemmissa ongelmissa lähistöksi voidaan ottaa yhden sijasta $k = 2, 3, 4, \ldots$ muuttujaa.
  - Eräs tapa ratkoa rajoiteongelma(luokka)a onkin analysoida tarvittavan lähistön kokoa k, ja pakottaa ongelmat k-konsistenteiksi. Tarpeeksi suurella k voidaan jopa *välttää peruutus*.
  - Toisaalta k on pakottamisen aikavaatimuksen eksponentissa. Lisäksi "tarpeeksi suuri" k voi lähestyä muuttujien lukumäärää N.
  - Yleensä tyydytäänkin 2- eli *polku*konsistenssiin *(path c.)*.
- Kaarikonsistentiksi pakottaminen tapahtuu poistamalla mahdottomia arvoja alueista D_i ja mahdottomia pareja rajoitteista $R_{i,j}$ niin pitkään kuin mahdollista.

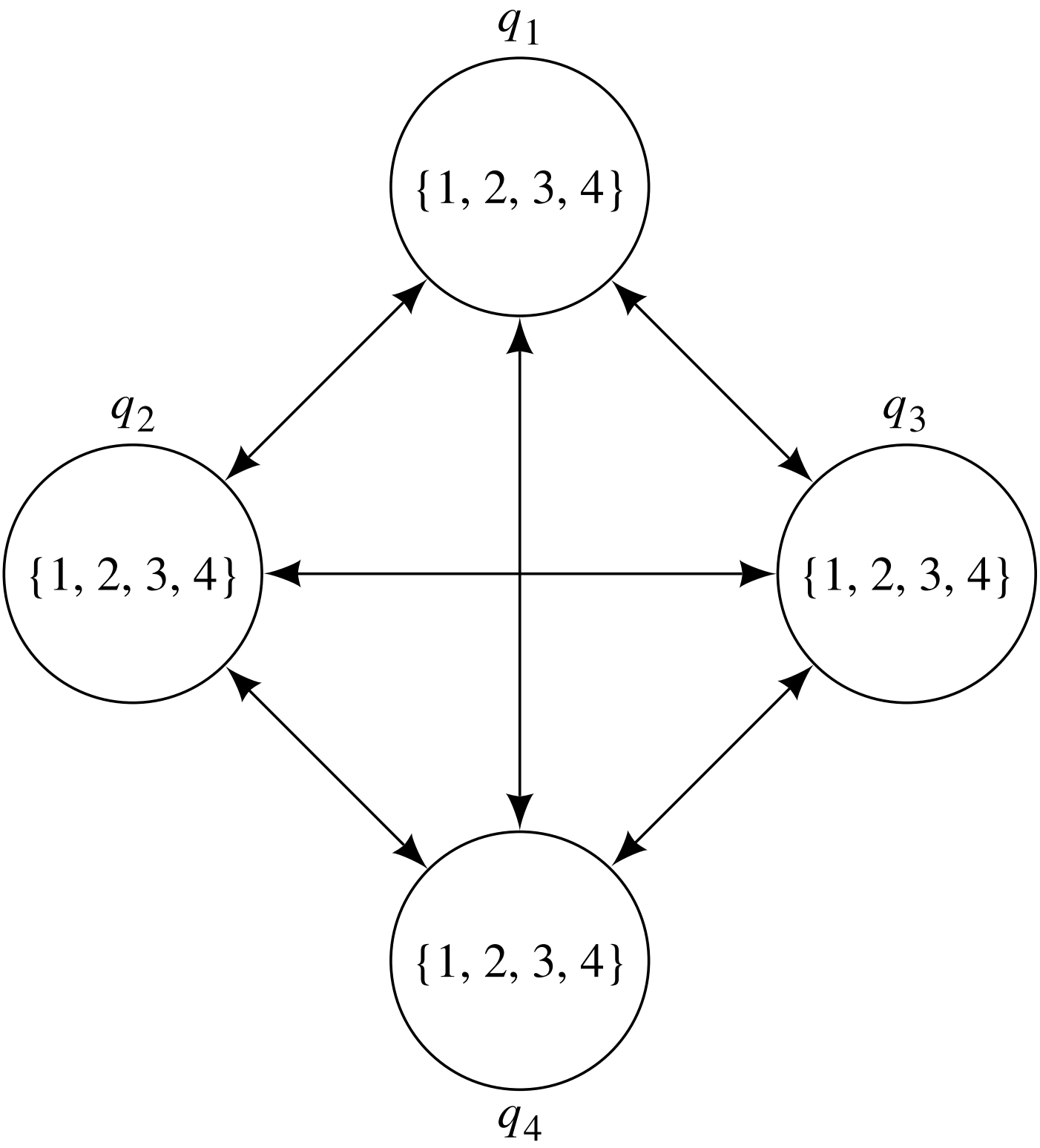
  Kukin poisto voi mahdollistaa uusia poistoja.

  Jos jokin joukko tyhjenee, ei kyseisellä rajoiteongelmalla ole ratkaisuja.

  Tämä pakottamismenetelmä on eräs *rajoitteiden levittämisen (constraint propagation)* erikoistapaus.
- Yleisesti voidaan rajoiteongelmaa ratkoa rajoitteita levittämällä vaikkapa seuraavasti:
  1. Aloitetaan ongelmasta, jossa tiedetään vain unaarirajoitteet, eli kukin $R_{i,j} = D_i \times D_j$.
  2. Lisätään muuttujien X_i ja X_j välinen binäärirajoite $R'_{i,j} \subseteq D_i \times D_j$ poistamalla pareista $R_{i,j}$ kaikki ne jotka eivät kuulu rajoitteeseen $R'_{i,j}$.
  3. Poistoja jatketaan valituksi k-konsistentiksi saakka.
  4. Sama toistetaan kaikille binäärirajoitteille.
  5. Lopuksi suoritetaan syvyyssuuntainen etsintä jäljelle jääneessä rajoiteongelmassa.
- Rajoitteiden levittämistä käytetään myös muiden etsintämenetelmien
  - esivaiheena
  - osana

  karsimaan etsintäavaruutta ennen raskasta hakua.
- Tarkastellaan esimerkkinä 4 kuningattaren sijoittelua 4×4-shakkilaudalle.

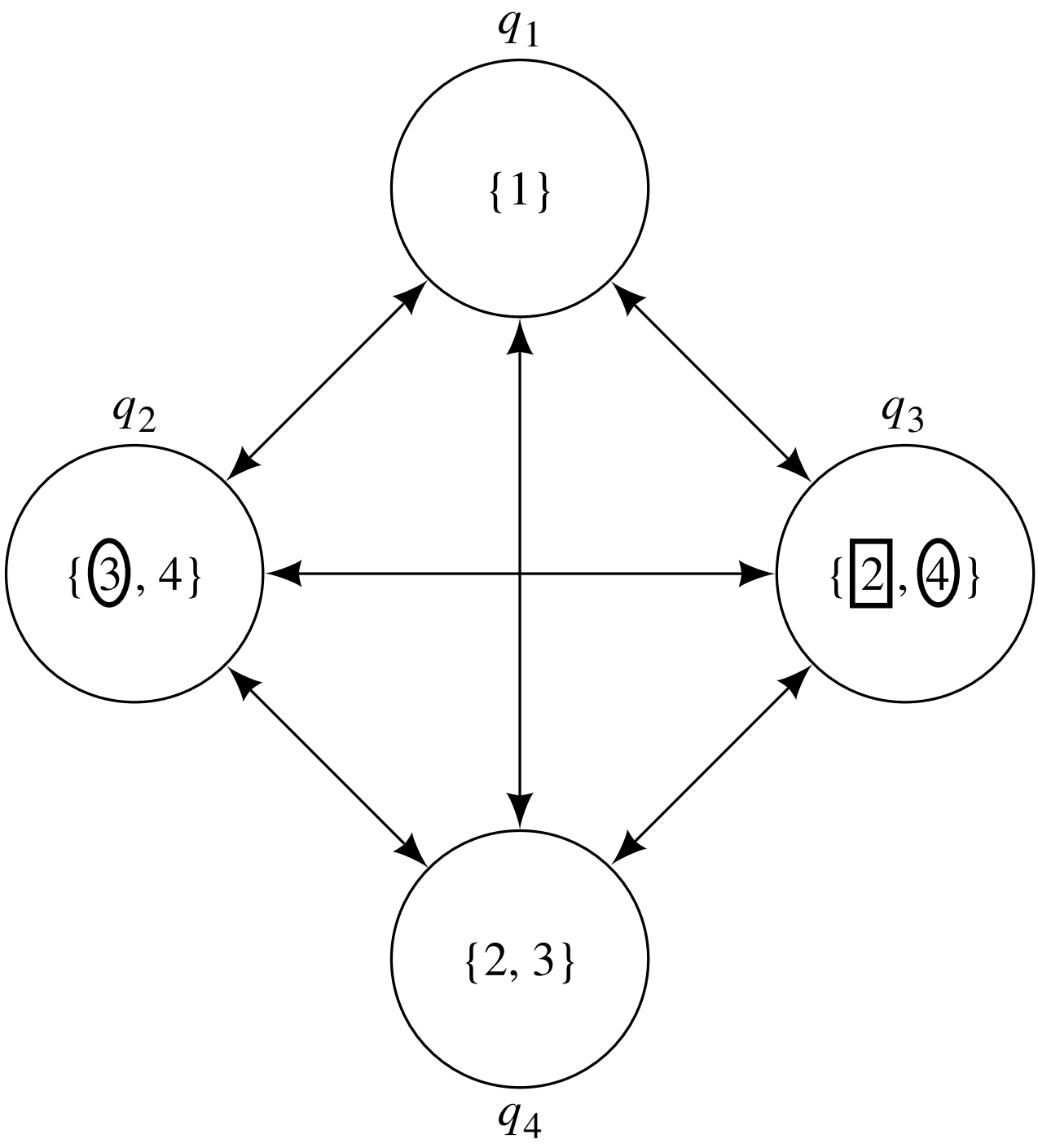
Alkutilanne on kuvassa 41: kuningattaret ovat muuttujina $q_1, \ldots, q_4$ joille sallitaan rivit $1, \ldots, 4$.



© 1998 Morgan Kaufman Publishers

Kuva 41: 4 kuningattaren rajoiteongelman alkutila [Nil98, kuva 11.4].

Jos valitaan ensin $q_1 = 1$, niin päädytään kuvan 42 kautta umpikujaan.



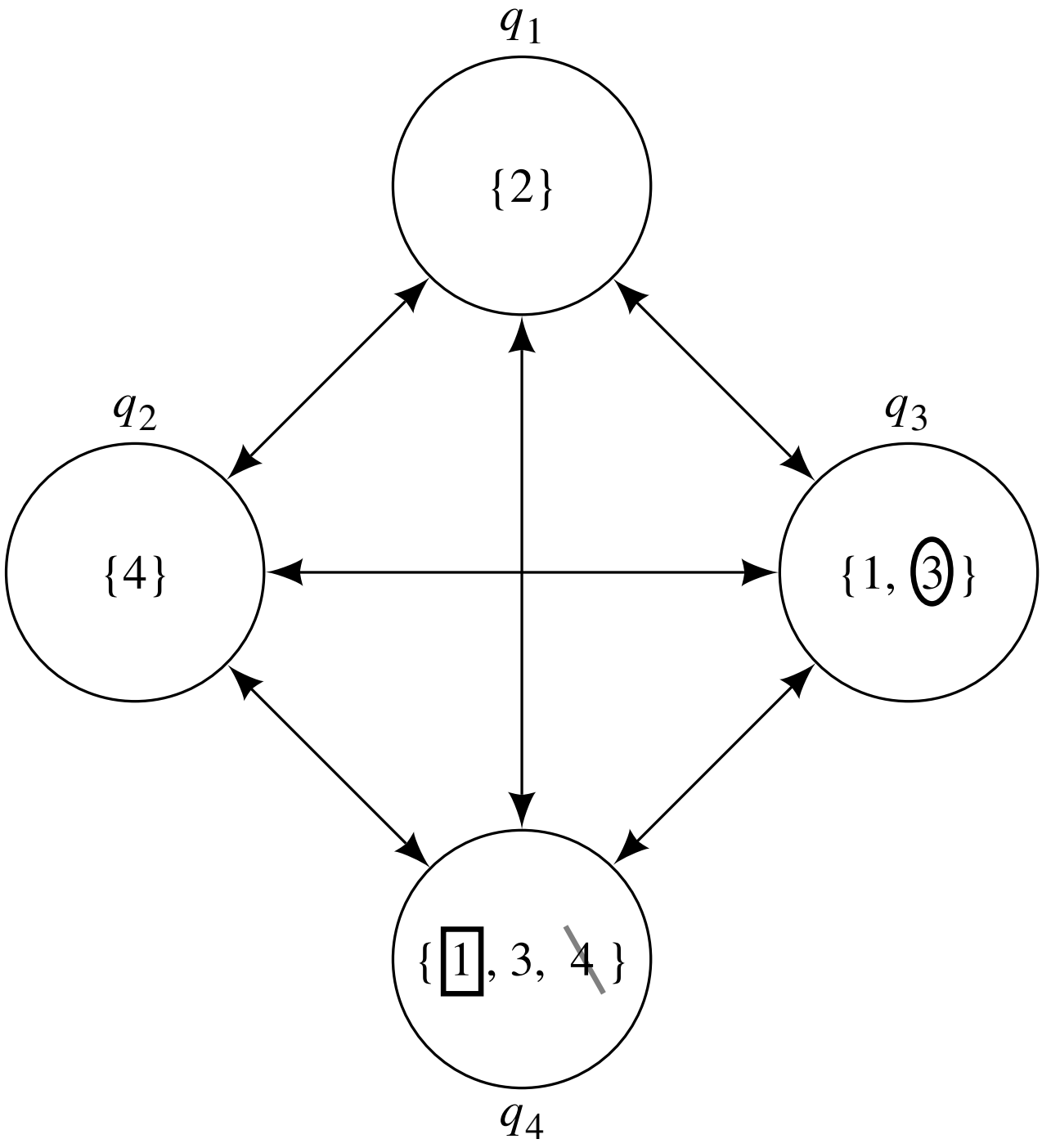
○ Values eliminated by first making arc (q_2, q_3) and then arc (q_3, q_2) consistent

□ Value eliminated by next making arc (q_3, q_4) consistent

© 1998 Morgan Kaufman Publishers

Kuva 42: 4 kuningattaren rajoiteongelma kun $q_1 = 1$ [Nil98, kuva 11.5].

Peruutetaan valintaan $q_2 = 2$, josta kuvan 43 kautta ratkaisuun.

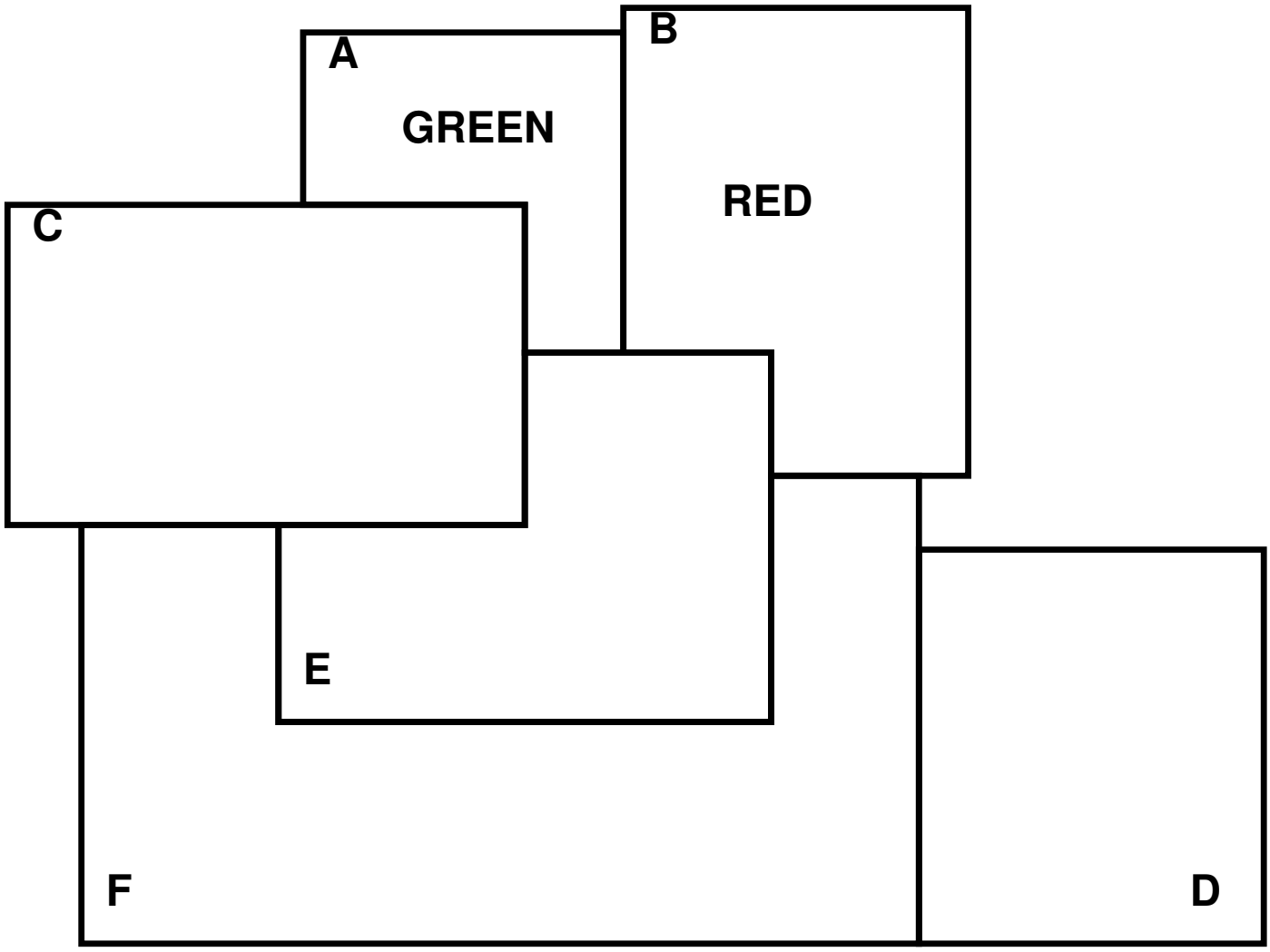


© 1998 Morgan Kaufman Publishers

Kuva 43: 4 kuningattaren rajoiteongelma kun $q_1 = 2$ [Nil98, kuva 11.6].

- Rajoiteongelmissa käytetään heuristiikkoja *valitsemaan seuraava muuttuja jolle määrätään arvo*. (Muuttujia ei siis enää käsitellä ennalta kiinnitetyssä järjestyksessä.)

  Tämä on erilaista heuristista etsinnän ohjausta kuin luvussa 3.2.
- Tarkastellaan esimerkiksi ongelmaa, jossa seuraava kuvio pitää värittää 3 värillä siten, että naapuruksilla on aina eri värit. Kaksi aluetta A ja B on jo väritetty vihreäksi (green) ja punaiseksi (red).



Kuva 44: Kolmiväriongelma [RN95, kuva 4.9].

- Seuraavaksi kannattaa värittää alue E siniseksi (blue), koska sillä ei ole enää muita vaihtoehtoja. Sitten:
  1. C voi olla vain punainen.
  2. F voi olla vain vihreä.
  3. D voi olla joko sininen tai punainen muttei vihreäksi.

  Tämä on ääriesimerkki *eniten rajoitettu muuttuja (most-constrained-variable) -heuristiikasta*:

Valitse käsittelyyn sellainen muuttuja X_j, jonka arvoalueessa D_j on *vähiten mahdollisia vaihtoehtoja* jäljellä.

Tämä valinta voidaan tehdä luontevasti eteenpäin tarkistettaessa.

Intuitiivisesti tämä heuristiikka pyrkii minimoimaan etsinnän *välitöntä* haarautumista valitsemalla aina vähiten haaroja synnyttävän vaihtoehdon.

Esimerkiksi kuvan 38 kuningatarongelmassa päästään aina tapauksiin $N = 100$ saakka, kun valitaan aina sellainen sarake j, jolla on vähiten vielä mahdollisia ruutuja.

- Toinen haarautumista ehkäisevä heuristiikka on *eniten rajoittava (most-constraining)* muuttuja -heuristiikka: valitse käsittelyyn sellainen muuttuja, joka on mukana mahdollisimman monessa binäärirajoittessa vielä käsittelemättömien muuttujien kanssa.

  Intuitiivisesti tämä heuristiikka pyrkii minimoimaan etsinnän *tulevaa* haarautumista valitsemalla aina sellaisen vaihtoehdon, jonka jälkeen myöhemmillä vaihtoehdoilla on vähän haaroja jäljellä.

- Kun muuttuja X_j on valittu jommalla kummalla heuristiikalla, on vielä valittava sille jokin mahdollinen arvo $V \in D_j$.

  Vähiten rajoittava arvo (least constraining value) -heuristiikka valitsee muuttujalle X_j sellaisen arvon $V \in D_j$, joka poistaa vähiten mahdollisia arvoja vielä käsittelemättömiltä muuttujilta.

  Valitaankin kuvassa 44 ensin käsittelyyn alue C (eikä E). Jos se väritetään

  siniseksi niin alueelle E ei jää yhtään mahdollista väriä.

  Alueelle F jää vain vihreä.

  punaiseksi niin alueelle E jää sininen.

  Alueelle F jää vihreä ja sininen (jonka E kieltää vasta myöhemmin).

  Valitaan siis punainen väri.

  Kuvan 38 kuningatarongelmassa päästään aina tapauksiin $N = 1\ 000$ saakka, jos valitaan sarakkeen j ruuduista se, joka synnyttää vähiten uusia uhkauksia vielä käsittelemättömille riveille.

Luvun 4.1 iteratiiviset parannusmenetelmät antavat *heuristisia korjausmenetelmiä (heuristic repair methods)*: ne ratkovat rajoiteongelmia

tekemällä pieniä muutoksia olemassa olevaan puutteelliseen ratkaisuun.

Eräs heuristiikka on *konfliktien minimointi (min-conflicts)*: (jollakin muulla heuristiikalla) valitulle muuttujalle X_j kokeillaan jotakin sellaista uutta arvoa $V \in D_j$, joka toteuttaa mahdollisimman monta tähän muuttujaan liittyvää binäärirajoitetta.

Kuvasarja 38 antaa esimerkin 8 kuningattaren ongelman ratkaisusta tällä periaatteella. Sen avulla jopa satunnainen tapaus $N = 1\ 000\ 000$ ratkeaa keskimäärin alle 50 muutoksella!

Esitetään vielä vuorottain rekursiivinen kaarikonsistenssialgoritmi, joka

- kieltää mahdottomana arvon $X_i = v_i$ poistamalla alkion v_i mahdollisten arvojen joukosta D_i
  (aliohjelma `poista-yksi`)
- kieltää mahdottomana arvoyhdistelmän $X_i = v_i, X_j = v_j$ poistamalla parin $\langle v_i, v_j \rangle$ mahdollisten arvoparien joukosta $R_{i,j}$
  (aliohjelma `poista-pari`)
- jatkaa näitä poistoja rekursiivisesti kunnes
  joko konsistenssi on palautettu jälleen voimaan poistamalla kaikki mitä voidaan
  tai mahdollisten joukot tyhjenevät, mikä osoittaa ettei koko rajoiteongelmalla ole yhtään ratkaisua.

Esityksen yksinkertaistamiseksi oletetaan, että

$$R_{j,i} = \{\langle v_j, v_i \rangle : \langle v_j, v_i \rangle \in R_{i,j}\}$$

kun $i > j$, eli että indeksien järjestyksellä ei ole väliä.

Algoritmia voi optimoida eri tavoin.

Polkukonsistenssi saataisiin muuttamalla algoritmia sellaiseksi, joka saattaa voimaan seuraavan "kolmioehdon":

Kun jokin joukko D_j kutistuu, niin

- Tutki jokainen erillinen (muuttuja)indeksipari i, k (missä siis $i \neq j \neq k \neq i$).
- Poista indeksiparin joukosta $R_{i,k}$ jokainen sellainen arvopari $\langle v_i, v_k \rangle$ joka. . .
- . . . menetti viimeisenkin välittävän alkion $v_j \in D_j$, eli sellaisen jolla $\langle v_i, v_j \rangle \in R_{i,j}$ ja $\langle v_j, v_k \rangle \in R_{j,k}$.

procedure `poista-yksi`(i, v_i)

1: **if** $v_i \in D_i$ **then**
2: $D_i \leftarrow D_i \setminus \{v_i\}$;
3: **for all** $\langle v_i, v_j \rangle \in R_{i,j}$ missä $j \neq i$ **do**
4: `poista-pari`$(i, j, \langle v_i, v_j \rangle)$
5: **end for**
6: **end if**

procedure `poista-pari`$(i, j, \langle v_i, v_j \rangle)$

1: **if** $\langle v_i, v_j \rangle \in R_{i,j}$ **then**
2: $R_{i,j} \leftarrow R_{i,j} \setminus \{\langle v_i, v_j \rangle\}$;
3: **if** $R_{i,j}$ ei sisällä yhtään paria $\left\langle v_i, v'_j \right\rangle$ **then**
4: `poista-yksi`(i, v_i)
5: **end if**
6: **if** $R_{i,j}$ ei sisällä yhtään paria $\left\langle v'_i, v_j \right\rangle$ **then**
7: `poista-yksi`(j, v_j)
8: **end if**
9: **end if**

Algoritmi 9: Kaarikonsistenssialgoritmi.

- 3-konsistenssissa kolmiosta i, j, k tulee "tetra" i, j, k, l
- 4-konsistenssissa "pentagrammi" i, j, k, l, m
- ja k-konsistenssissa täydellinen $k + 1$ solmun aliverkko.

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 $U(H)$.

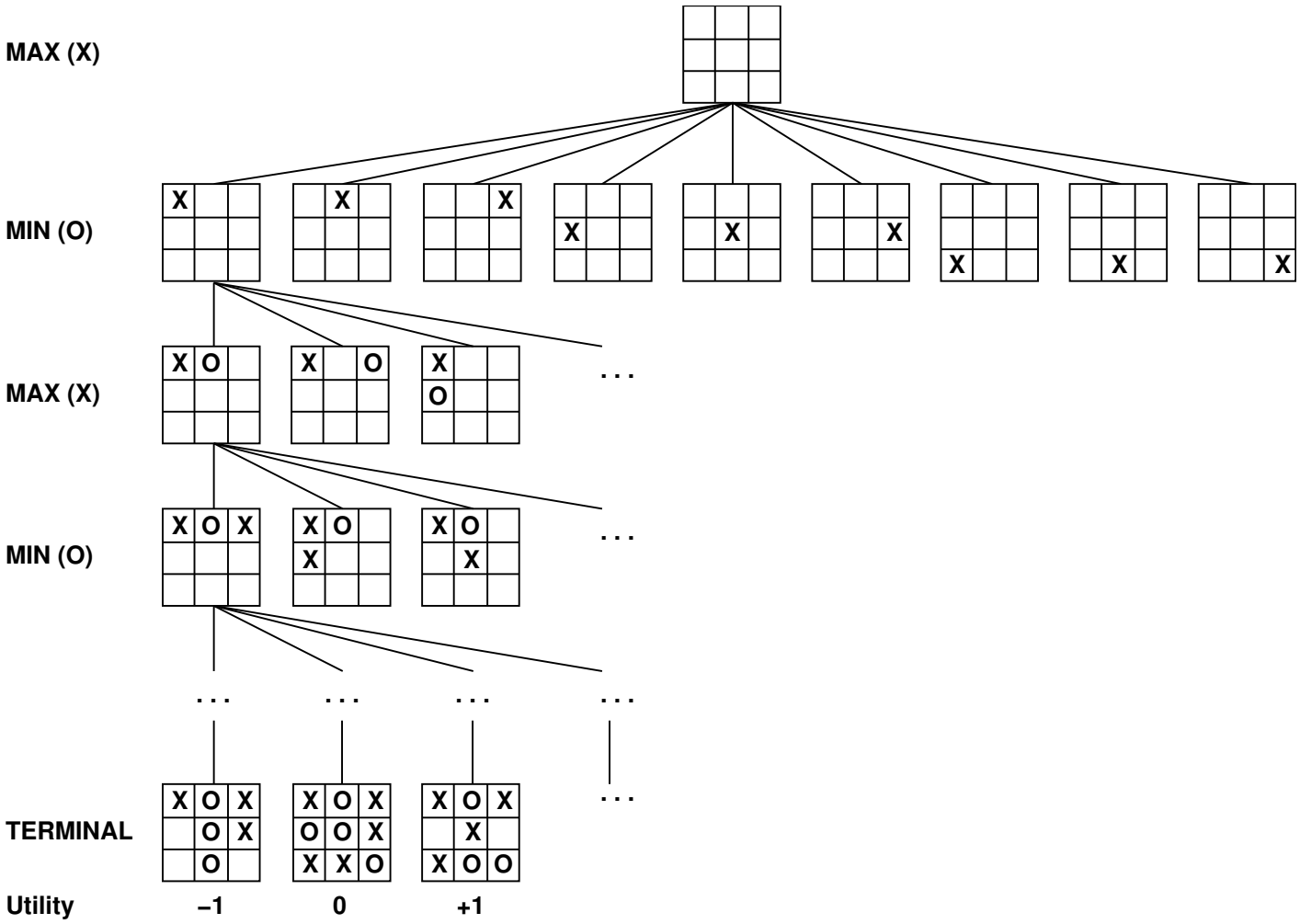
$\max$-**tason sisäsolmu** saa arvokseen luvun *maksimin* sen lapsisolmujen arvoista.

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

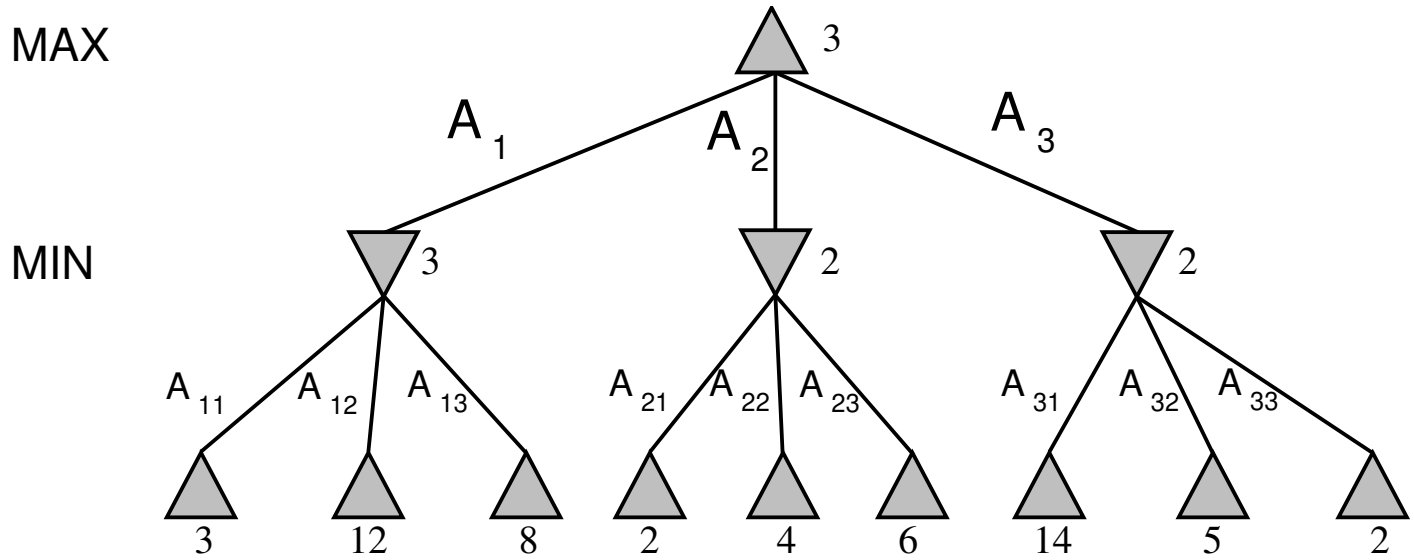
Näin edetään lehdistä aina juureen asti.

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:

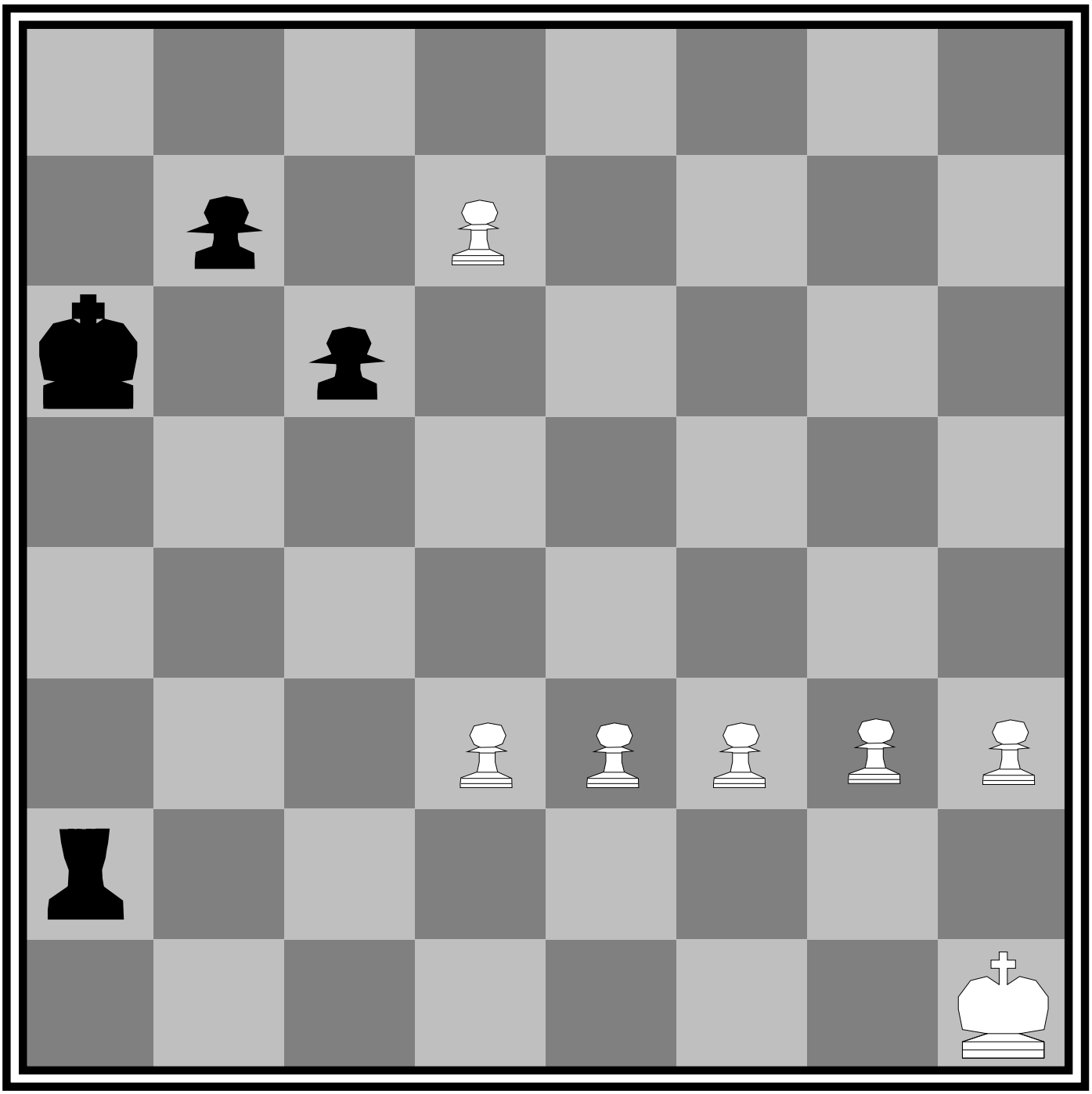
  Solmulle H annetun arvon $\textsc{Eval}(H)$ tulisi heijastaa mahdollisimman luotettavasti sitä, kuinka pelitilanteesta H jatkuvassa pelissä *odotusarvoisesti* tulisi käymään.

  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.

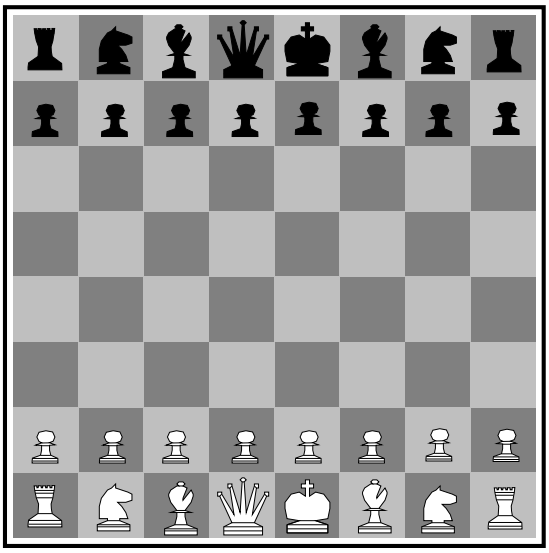


Kuva 47: Musta shakkaa, valkea voittaa [RN95, kuva 5.5].

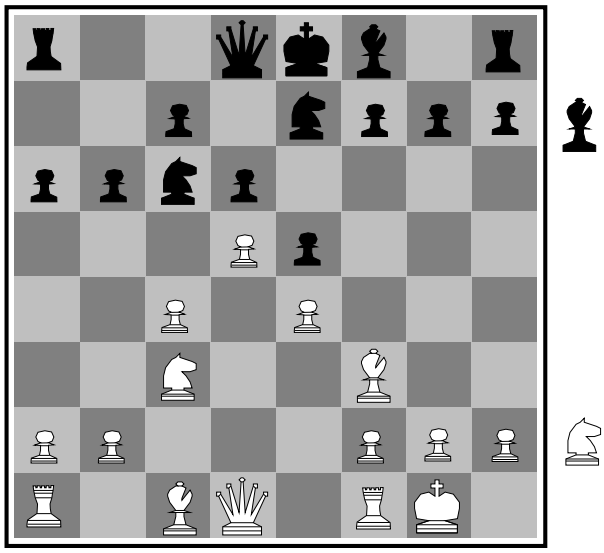
Onneksi funktion $\textsc{Eval}$ ei tarvitse saada eri solmujen H_1 ja H_2 numero*arvoja* oikein, kunhan niiden *järjestys* vain säilyy, eli

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

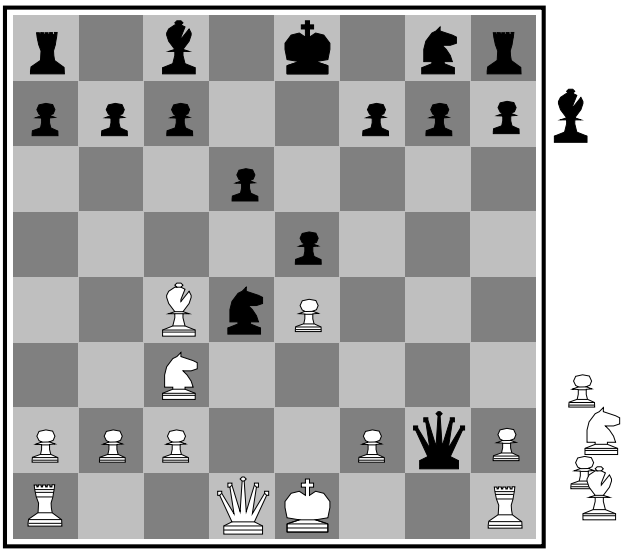
vain jos H_2 on "ainakin yhtä lupaava pelitilanne" kuin H_1.



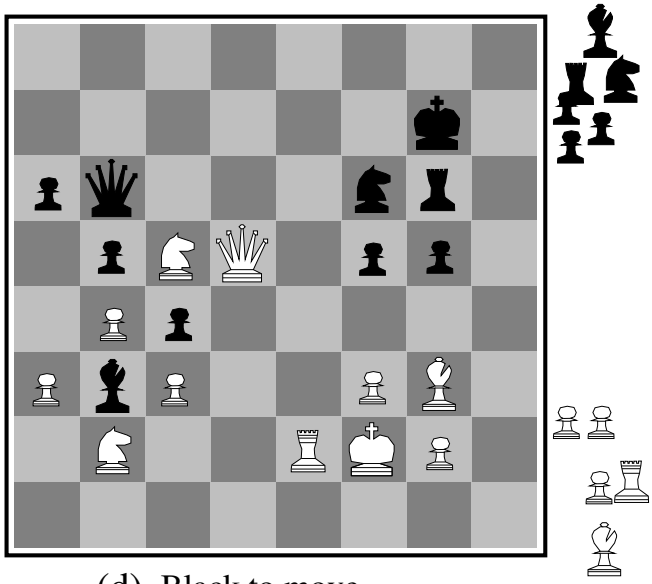
(a) White to move
Fairly even



(b) Black to move
White slightly better



(c) White to move
Black winning



(d) Black to move
White about to lose

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

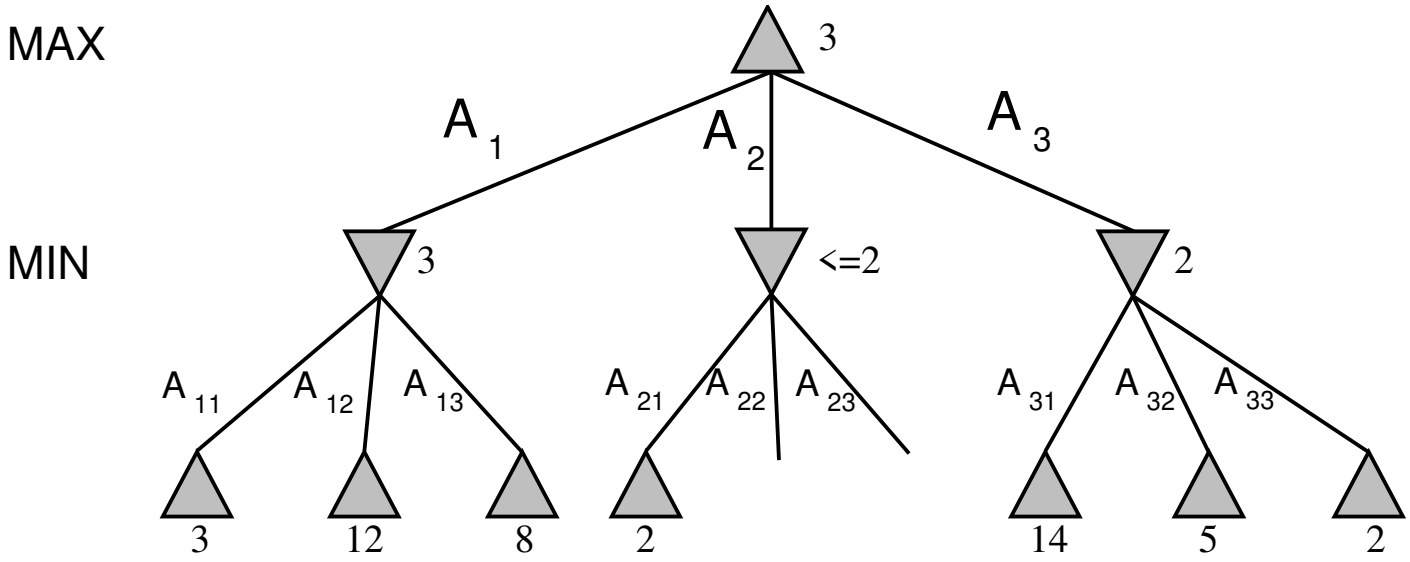
4.3.2. Alfa-beta -karsinta

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

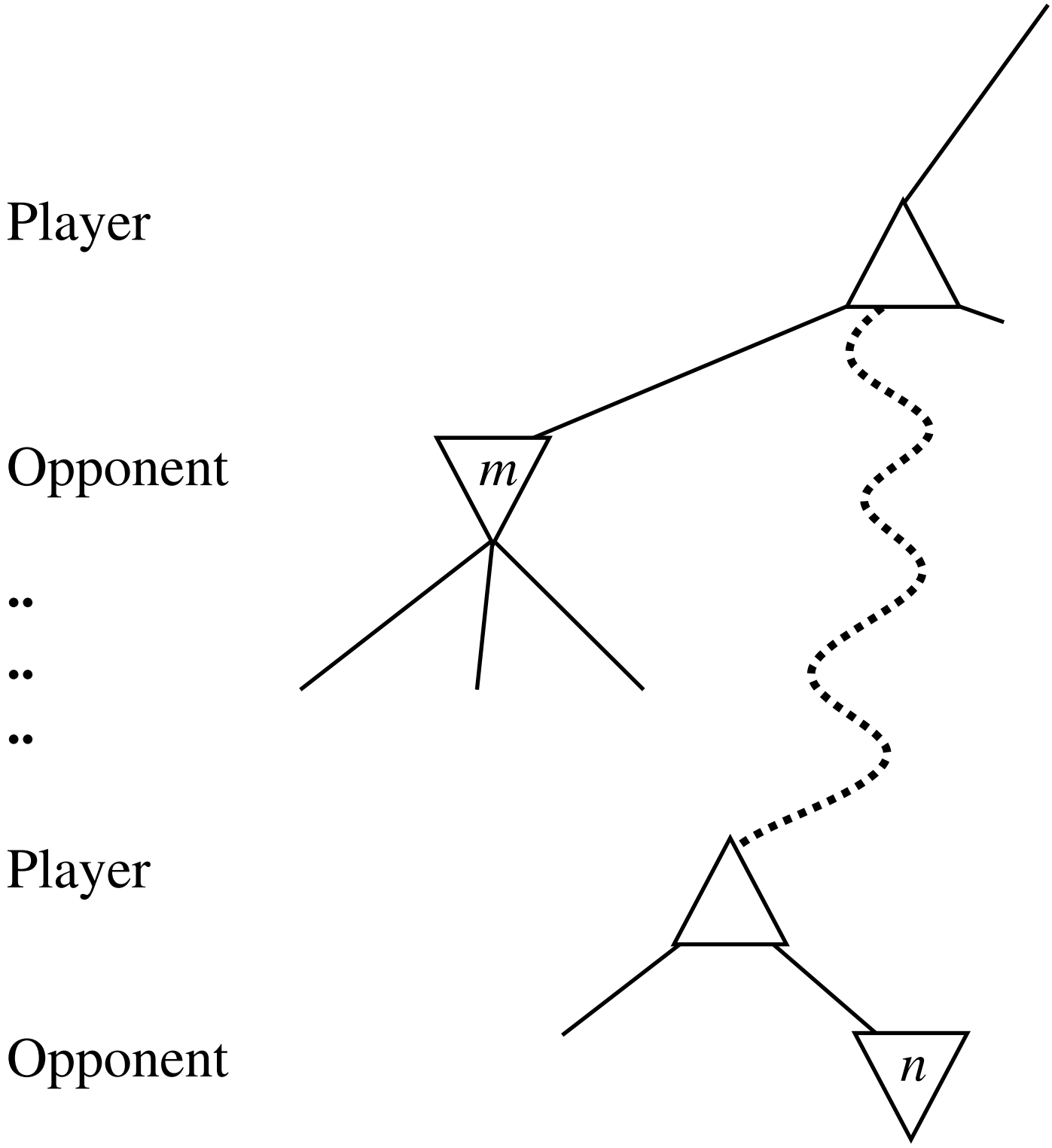
  Algoritmi 11 karsii sitä paremmin, mitä aikaisemmin kunkin solmun oikea minimax-arvo tavataan sen lapsista.

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

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.

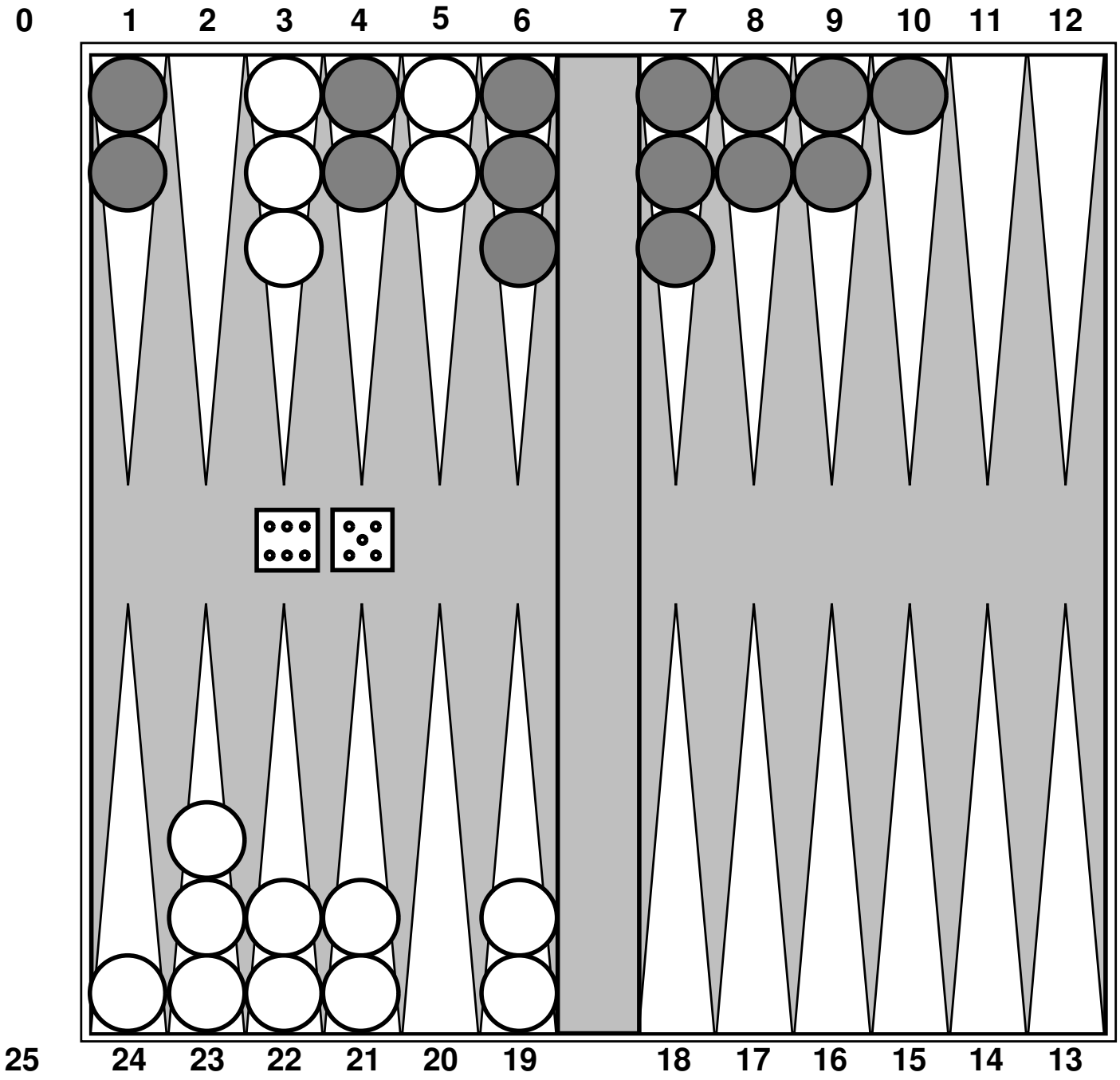
Alfa-beta-karsinta on yleisen *haaraudu ja rajoita (branch and bound)* -etsintäperiaatteen sovellus pelipuihin:

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*.

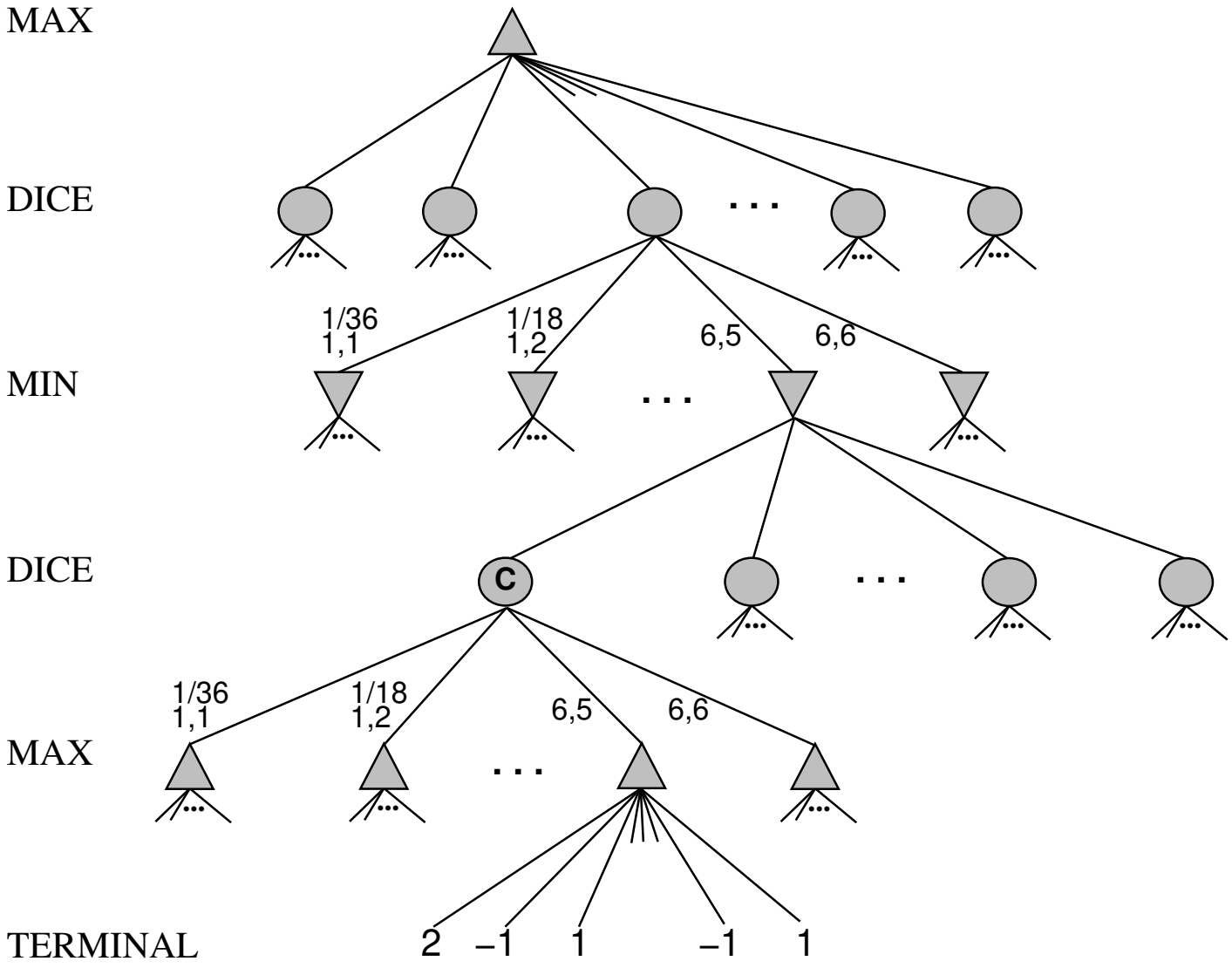
Backgammonissa kaikki 2 nopalla heitettävissä olevat (järjestämättömät) silmälukuparit.

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.

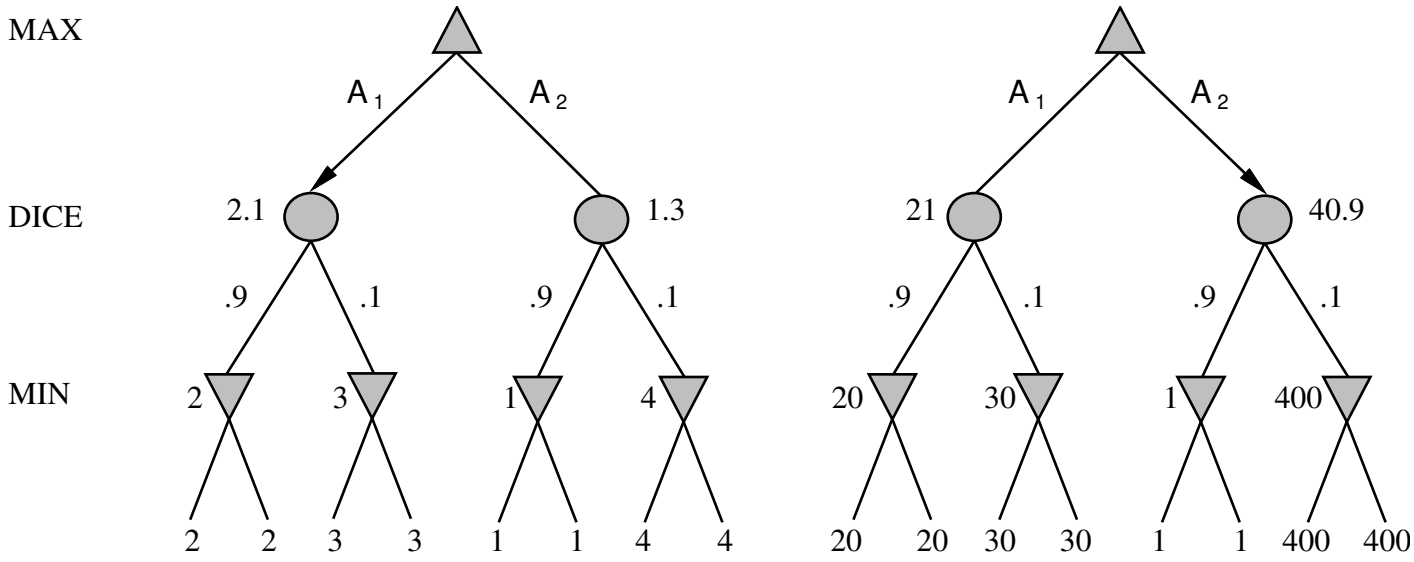
Kun minimax-algoritmia 10 muokataan näin, saadaan *expectiminimax*-algoritmi.

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 $\textsc{Eval}$ 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 $\leftarrow$ ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  $t \leftarrow t + 1$
  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.

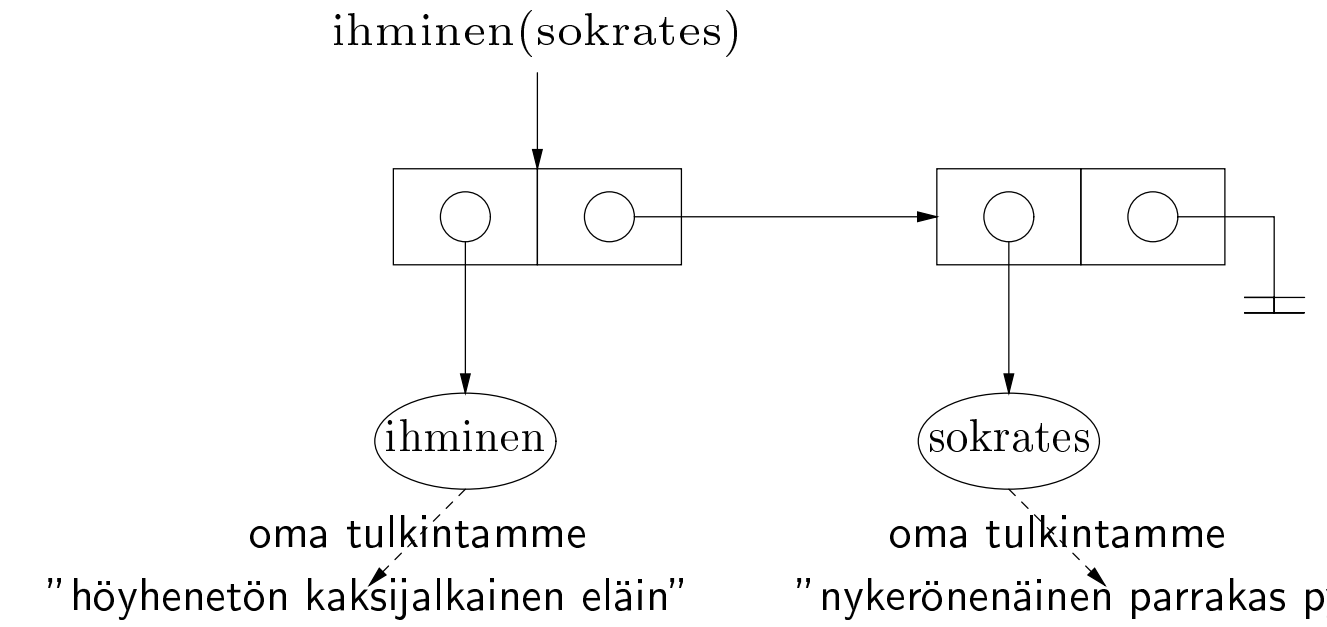
"Tuo ohjelma tietää, että Sokrates on ihminen."

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

```
computer thinks
```
(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"

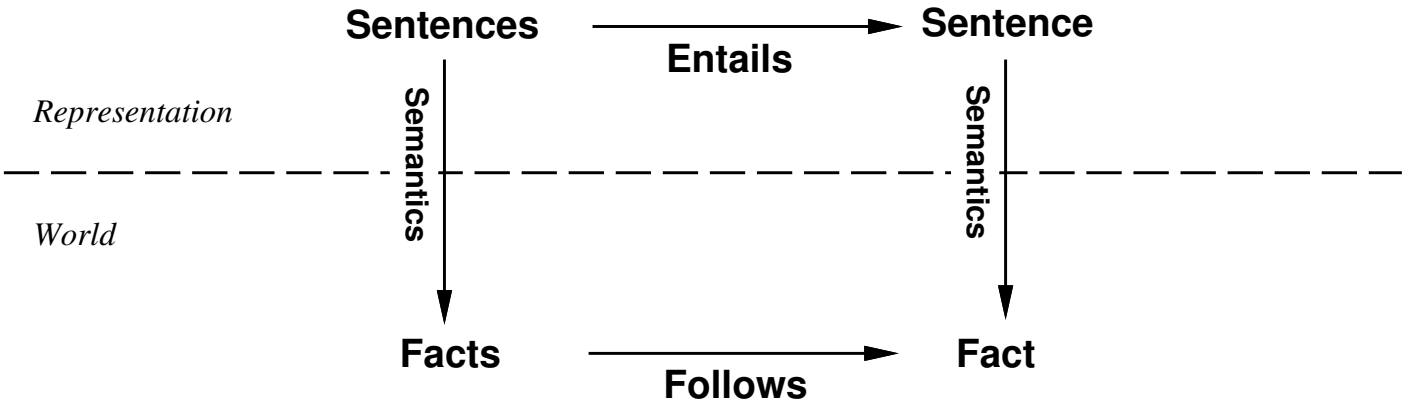
```
(ihminen sokrates)

(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 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 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" kompositio-

naalinen 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 "$\mathrm{Sokrates} \in \mathrm{kuolevainen}$".

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

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

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

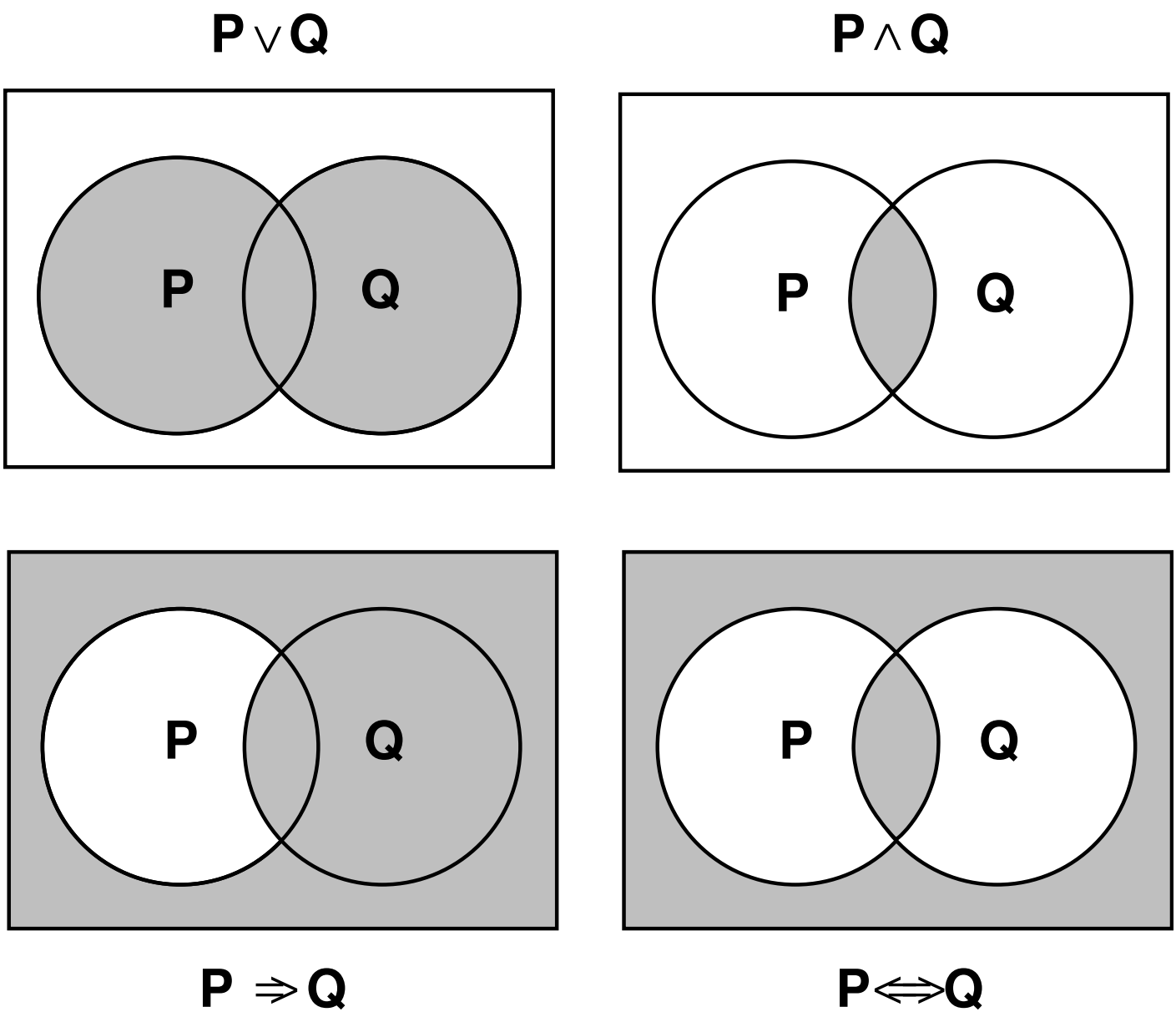
$$\mathrm{Mod}(\phi) \subseteq \mathrm{Mod}(\psi). \tag{18}$$

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 "$\mathrm{aurinko\ paistaa\ ja\ vettä\ sataa}$", niin me saamme päätellä siitä sekä "$\mathrm{aurinko\ paistaa}$" että "$\mathrm{vettä\ sataa}$".

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
$\frac{\alpha \Rightarrow \beta \qquad \alpha}{\beta}$	modus ponens
$\frac{\alpha_1 \wedge \ldots \wedge \alpha_n}{\alpha_i}$	konjunktion poisto
$\frac{\alpha_1, \ldots, \alpha_n}{\alpha_1 \wedge \ldots \wedge \alpha_n}$	konjunktion tuonti
$\frac{\alpha_i}{\alpha_1 \vee \ldots \vee \alpha_n}$	disjunktion tuonti
$\frac{\neg\neg\alpha}{\alpha}$	kaksoisnegaation poisto
$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$	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".

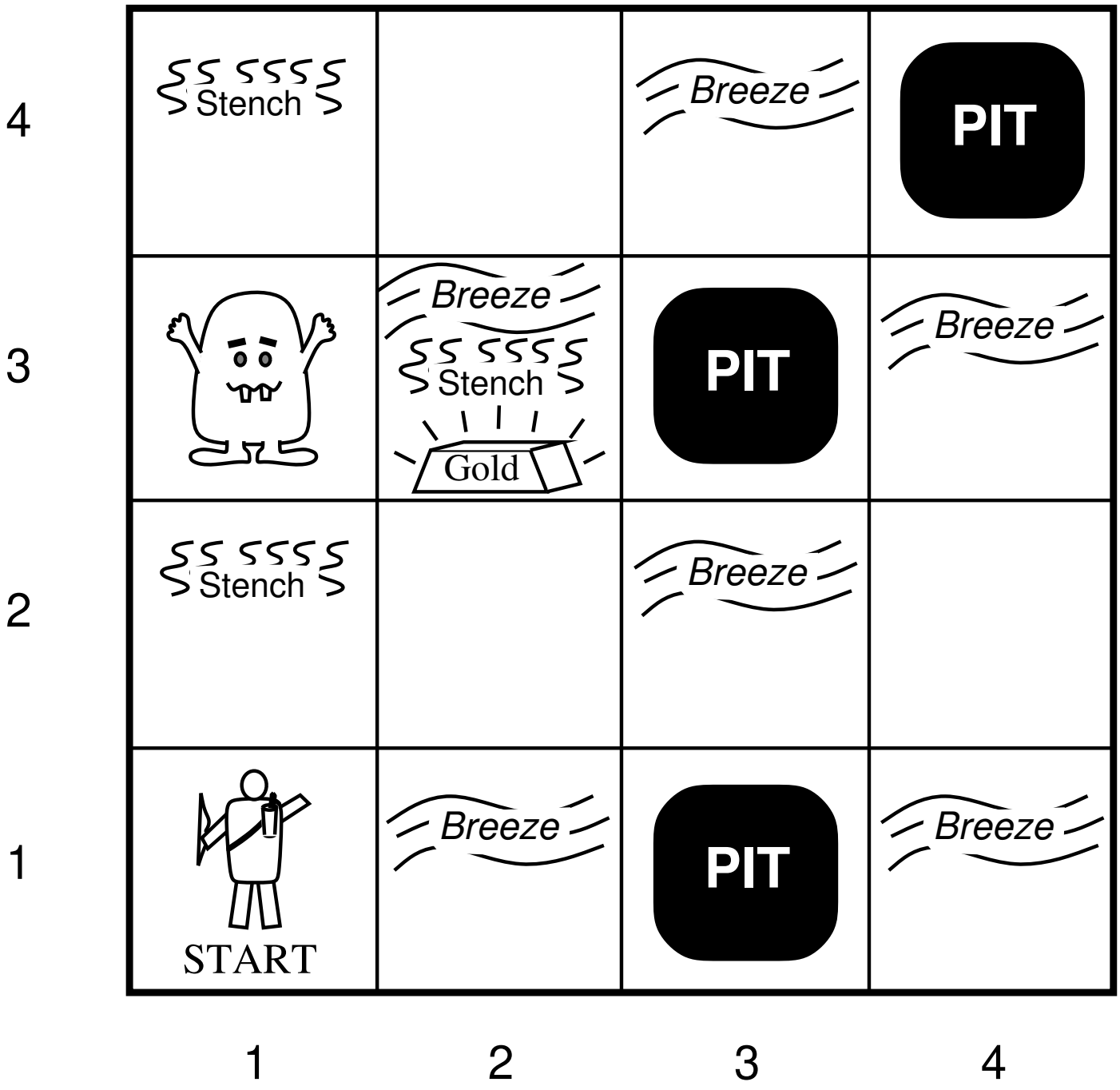
Esimerkiksi taulukon 7 resoluutiosäännön käyttö on helppoa, koska jokaisessa Hornin lauseessa on vain 1 *positiivinen (= ei negatoitu)* lausemuuttuja. Silloin vaihtoehtoja on vähemmän ja etsintä suunnatumpaa.

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:

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

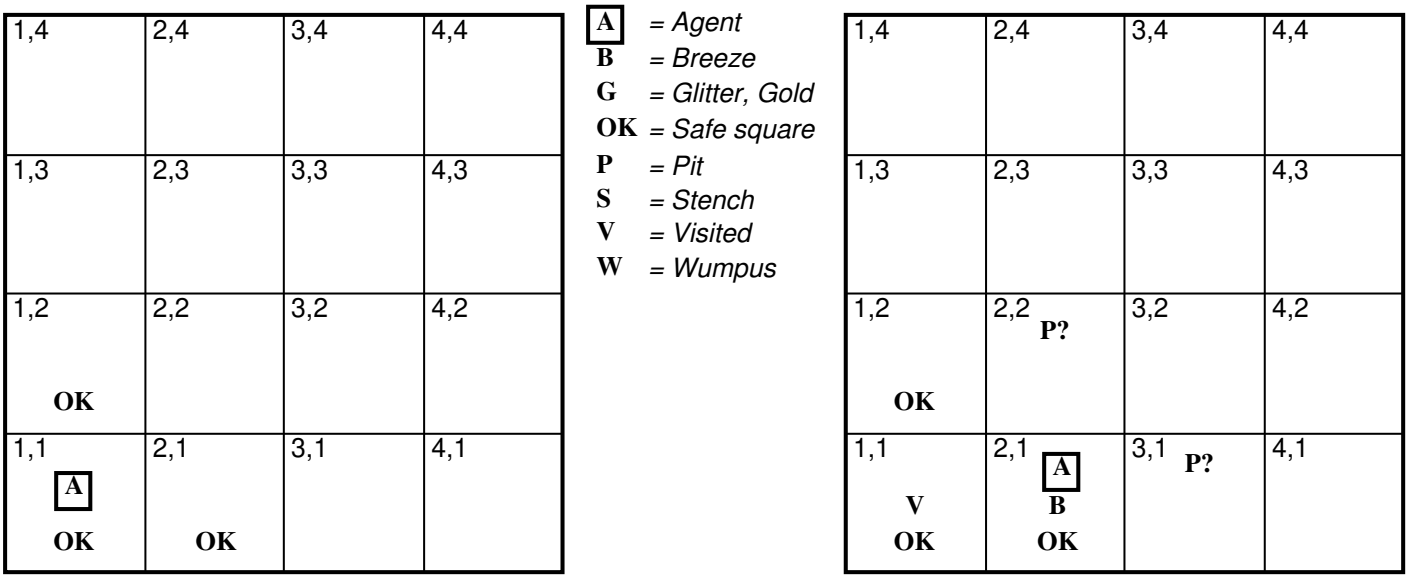
  jokaiselle ruudulle q.
- Wumpuksen tai kuopan ruuduissa kuolee, muualla ei:

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

- 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):

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

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

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

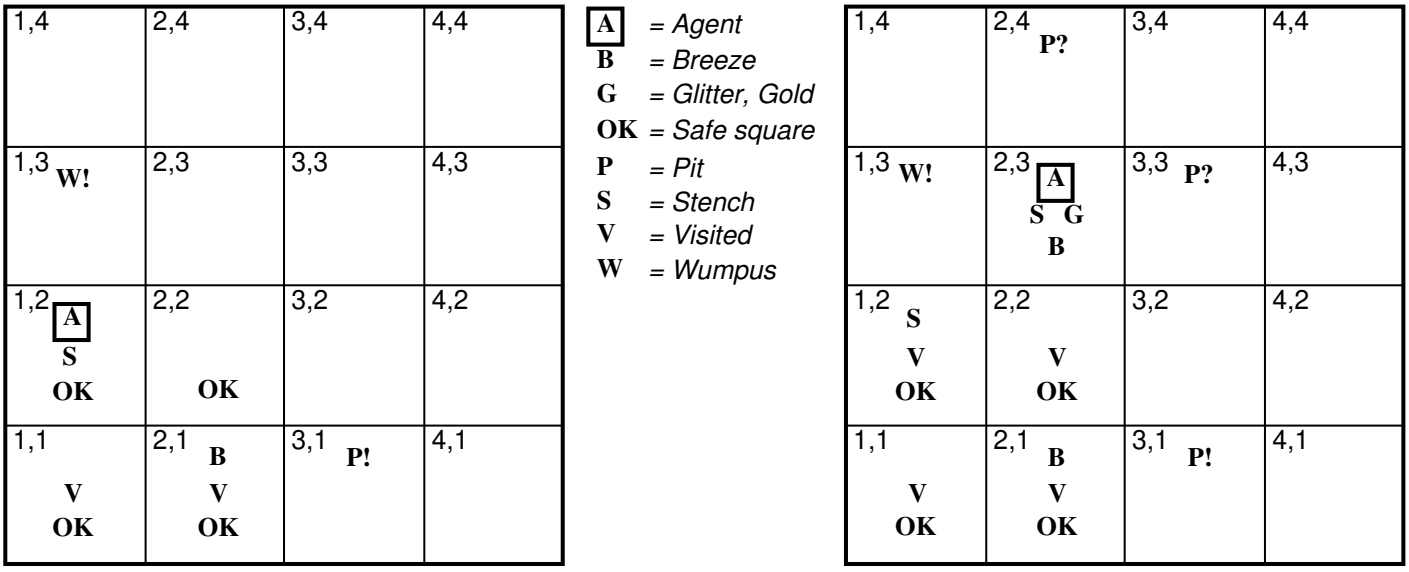
Lauseista (22), (27) ja (28) saadaan lopulta etsityt tulokset $O_{[1,1]}$, $O_{[1,2]}$ ja $O_{[2,1]}$.

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 $[1,2]$.



(a) (b)

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 $\neg P_{[1,1]} \wedge \neg P_{[1,2]} \wedge \neg P_{[1,3]} \wedge \neg P_{[2,2]}$ lausees-ta (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 $\mathcal{X} = \{x_i : i \in \mathbb{N}\}$.
Funktiosymbolit $\mathcal{F} = \{f_j : j \in \mathbb{N}\}$.
Predikaattisymbolit $\mathcal{P} = \{p_k : k \in \mathbb{N}\}$.

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

$(\forall x_i \phi)$ 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

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

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

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

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

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

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 $\mathcal{I}(\mathrm{founder})$ 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

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

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 $\theta : \mathcal{X} \mapsto \mathcal{U}$.

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 '$\mathrm{on\ olemassa\ sellainen\ldots että\ldots}$' eli "$\mathrm{jollakin}$"
- '$\forall$' vastaa ilmaisua '$\mathrm{ei\ ole\ olemassa\ sellaista\ldots ettei\ldots}$' eli '$\mathrm{kaikilla\ldots}$'.

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

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

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 $(f\ \ t_1\ \ \ldots\ \ t_{\mathrm{ar}(f)})$ 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. $\mathfrak{M} \models (\text{kaava (29)})\theta$ 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

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

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:

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

Vertaa sanakirjamääritelmät:

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

Tässä on jopa määritelmä *tapauksittain (by cases)*: *. . . sama isä tai 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

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

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

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*".

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

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

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 "$\mathrm{määrittelemätön}$" ja sille vakiosymboli $\mathrm{undefined}$.
– Tarvitaan siis vielä esimerkiksi lause "sukupuoli on määritelty vain ihmisille"

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

  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."

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

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

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

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

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

   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

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

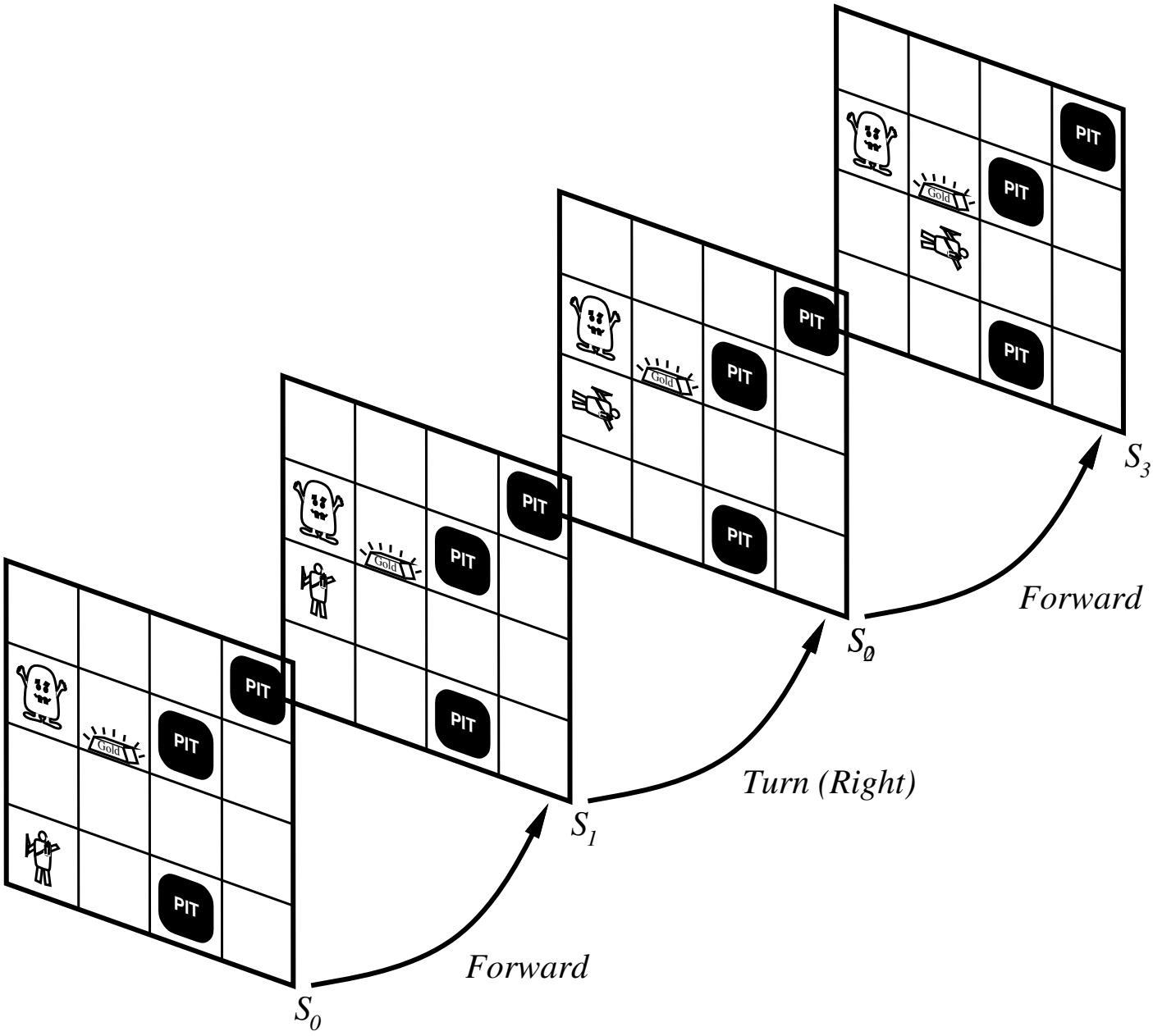
$$\text{president-of-Finland}(\text{'13th Nov. 2001'}) = \text{'T. Halonen'}$$

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

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

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

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

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

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

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

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":

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

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

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

totta edelleen $\Leftrightarrow$ tämä teko teki sen todeksi

$\vee$

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 $\mathrm{forward}$ sopivalla seuraajatilanneaksioomalla:

$$\forall a, d, p, s, l.\, \mathrm{at}(p, l, \mathrm{result}(a, s)) \Leftrightarrow a = \mathrm{forward} \wedge l = \mathrm{location_ahead}(p, s) \wedge \neg\, \mathrm{wall}(l) \vee \mathrm{at}(p, l, s) \wedge a \neq \mathrm{forward}\,.$$

- 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:

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

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 variantteja kohden kannattaisi pyrkiä.

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

aksioomina kaikki lauselogiikan lauseet muotoa

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

$$\alpha \Rightarrow (\beta \Rightarrow (\alpha \wedge \beta))$$

$$\alpha \wedge \beta \Rightarrow \alpha$$

$$\alpha \wedge \beta \Rightarrow \beta$$

$$\alpha \Rightarrow \alpha \vee \alpha$$

$$\alpha \Rightarrow \alpha \vee \beta$$

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

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

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

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:

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

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:

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \wedge I$$

$$\frac{\alpha}{\alpha \vee \beta} \vee I_1$$

$$\frac{\beta}{\alpha \vee \beta} \vee I_2$$

$$\frac{\begin{array}{c}[\alpha] \\ \vdots \\ \beta\end{array}}{\alpha \Rightarrow \beta} \Rightarrow I \tag{41}$$

  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

$$\frac{\begin{array}{c}1. \\ [A]\end{array}}{A \Rightarrow A} \Rightarrow I, 1.$$

  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 $\textsc{Subst}(\{x/y\}, \alpha)$ pääteltiin.

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):*

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

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

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

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

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

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."

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

- "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

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

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

$$\frac{1 \qquad \begin{matrix}[(54), (55)] \\ (54)\end{matrix}}{(54)}\, \wedge E.$$

3. Aksioomasta (51) saadaan universaalikvanttorin poistolla

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

koska

$$\frac{(51)\quad \begin{array}{c}[\mathrm{missile(M1)}] \\ \mathrm{missile(M1)} \Rightarrow \mathrm{weapon(M1)}\end{array}}{\mathrm{missile(M1)} \Rightarrow \mathrm{weapon(M1)}}\,\forall E$$

kun valitaan nokkelasti *sama* $t = \mathrm{M1}$ 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

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

koska Modus Ponens

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

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

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

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

5. Aksioomasta (50) saadaan universaalikvanttorin poistolla

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

6. Askelista 5 ja 1 saadaan päättelysäännöllä Modus Ponens

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

7. Aksioomasta (45) saadaan kolmen universaalikvanttorin poistolla

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

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

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

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 $\{var/val\} \in \theta$
    then return UNIFY-INTERNAL(val, x, θ)
  else if $\{x/val\} \in \theta$
    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-check*) estää kehät.

Nämä rakenteet ovat

joko muuttujasymboleita (VARIABLE?)

tai rakenteita (COMPOUND?) jotka koostuvat

- operaattorista (OP), joka on predikaatti- tai funktiosymboli
- ja sen argumenttilistasta (ARGS), jota käydään läpi jäsen jäseneltä (FIRST, 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

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

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

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

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

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

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).

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

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*) = θ succeeds **do**
FIND-AND-INFER(*KB*, $[p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n]$, *q*, θ)
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** $\{\theta\}$
  $q \leftarrow$ FIRST(*qlist*)
    for each q_i' **in** *KB* such that $\theta_i \leftarrow$ UNIFY(q, q_i') succeeds **do**
      Add COMPOSE(θ, θ_i) to *answers*
    end
    for each sentence $(p_1 \wedge \ldots \wedge p_n \Rightarrow q_i')$ **in** *KB* such that $\theta_i \leftarrow$ UNIFY(q, q_i') succeeds **do**
      answers $\leftarrow$ BACK-CHAIN-LIST(*KB*, SUBST$(\theta_i, [p_1 \ldots p_n])$, COMPOSE$(\theta, \theta_i)) \cup$ *answers*
    end
  return the union of BACK-CHAIN-LIST(*KB*, REST(*qlist*), θ) for each $\theta \in$ *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

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

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

3. Ainoa mahdollisuus on soveltaa faktaa (49)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   mutta muuntuu muotoon

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

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)*

$$\neg \left(\alpha \overset{\vee}{\underset{\wedge}{}} \beta \right) \text{ muotoon } (\neg\alpha) \overset{\wedge}{\underset{\vee}{}} (\neg\beta)$$

$$\neg \overset{\exists}{\underset{\forall}{}} x\alpha \text{ muotoon } \overset{\forall}{\underset{\exists}{}} x \neg\alpha$$

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

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

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

6. Skolemisoi tarkastelemalla edellisen vaiheen tuottamaa kaavaa

$$Q_1 x_1 \ldots Q_n x_n . \alpha, \tag{63}$$

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

$$\forall x. \neg \,\mathrm{person}(x) \vee \mathrm{song}(\mathrm{a_song_for}(x)) \wedge \mathrm{deserves}(x, \mathrm{a_song_for}(x)). \tag{64}$$

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

$(\alpha \wedge \beta) \vee \gamma$ 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

$$\forall x. (\neg \,\mathrm{person}(x) \vee \mathrm{song}(\mathrm{a_song_for}(x))) \wedge (\neg \,\mathrm{person}(x) \vee \mathrm{deserves}(x, \mathrm{a_song_for}(x))). \tag{65}$$

8. Nosta konjunktiot kvanttoreiden yli uudelleenkirjoitussäännöillä

$\forall x(\alpha \wedge \beta)$ muotoon $(\forall x \alpha) \wedge (\forall x \beta)$.

Nämä säännöt säilyttävät jälleen kaavan semantiikan.

Esimerkiksi kaava (65) muuttuu lopulta muotoon

$$(\forall x_1.\neg \,\mathrm{person}(x_1) \vee \mathrm{song}(\mathrm{a_song_for}(x_1)))\wedge$$
$$(\forall x_2.\neg \,\mathrm{person}(x_2) \vee \mathrm{deserves}(x_2, \mathrm{a_song_for}(x_2)))$$

eli implikaatioina

$$(\forall x_1.\, \mathrm{person}(x_1) \Rightarrow \mathrm{song}(\mathrm{a_song_for}(x_1)))\wedge$$
$$(\forall x_2.\, \mathrm{person}(x_2) \Rightarrow \mathrm{deserves}(x_2, \mathrm{a_song_for}(x_2))).$$

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

$$\dfrac{p'_2 \quad \dfrac{p'_1 \quad \neg p_1 \vee \ldots \vee \neg p_n \vee q}{\textsc{Subst}(\theta_1, \neg p_2 \vee \ldots \vee \neg p_n \vee q)} \, RES}{\begin{array}{c}\textsc{Subst}(\textsc{Compose}(\theta_1, \theta_2), \neg p_3 \vee \ldots \vee \neg p_n \vee q) \\ \vdots \\ \textsc{Subst}(\theta, q)\end{array}} \, RES \tag{67}$$

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:

$$\mathrm{Kills}(\mathrm{Jack}, \mathrm{Tuna}) \vee \mathrm{Kills}(\mathrm{Curiosity}, \mathrm{Tuna}).$$

  Normalisointia ei nyt tarvittu. Tämäkään ei ole Hornin lause, koska siinä on enemmän kuin yksi positiivinen literaali.
- Tuna-kissasta tarvitaan vielä:

$$\mathrm{Cat}(\mathrm{Tuna}) \text{ ja}$$

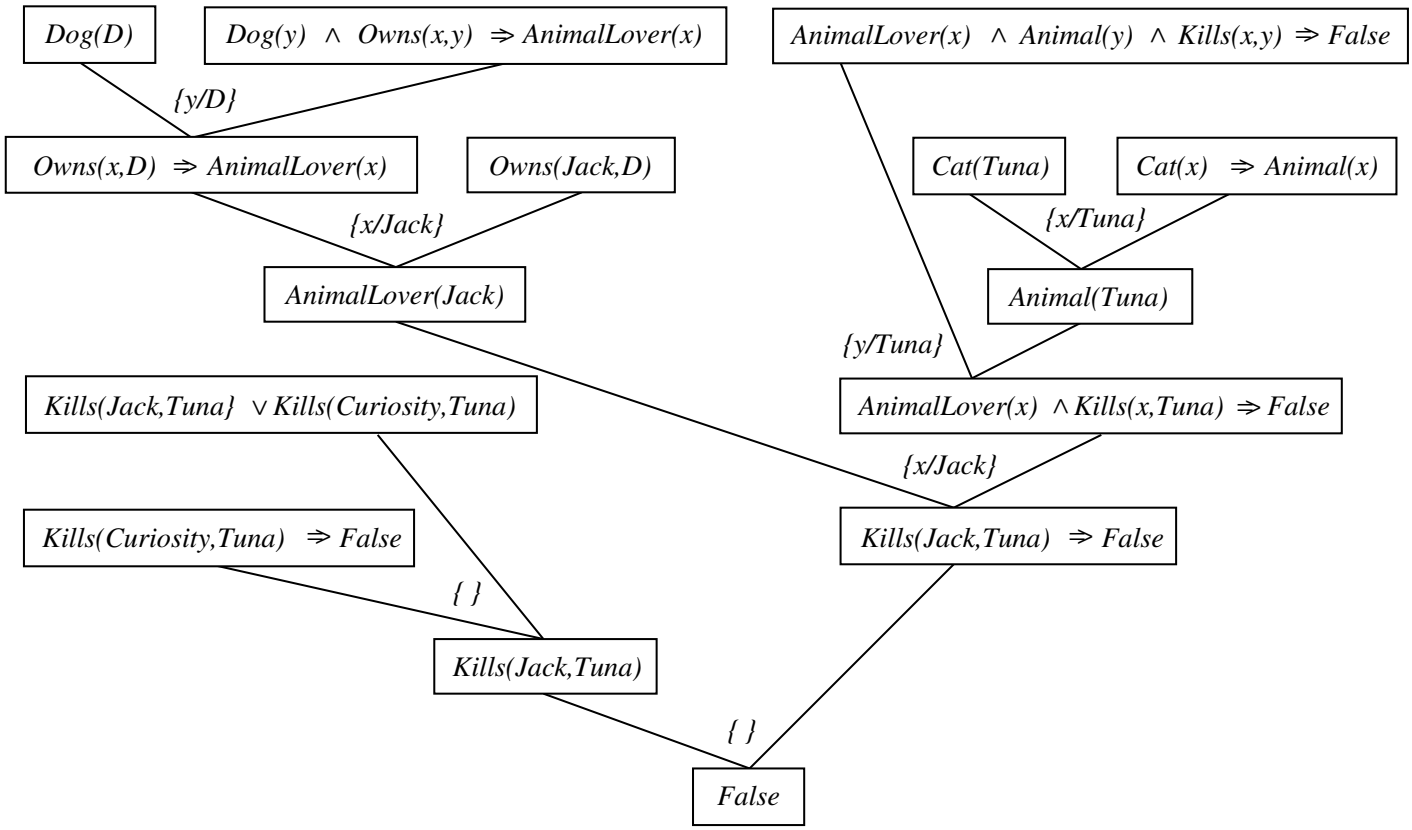
$$\forall x.\, \mathrm{Cat}(x) \Rightarrow \mathrm{Animal}(x) \text{ eli}$$

$$\mathrm{Cat}(x) \Rightarrow \mathrm{Animal}(x).$$

- Tappoiko uteliaisuus Tuna-kissan?
  Vastaväite on "uteliaisuus ei tappanut Tuna-kissaa" eli

$$\mathrm{Kills}(\mathrm{Curiosity}, \mathrm{Tuna}) \Rightarrow \mathbf{F}.$$

Näistä normalisoiduista kaavoista saadaan ristiriita seuraavilla resoluutioaskeleilla:



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 $\mathrm{p(a)}$ eikä kaavaa $\mathrm{p(a)} \vee \mathrm{q(b)}$ 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 $\mathrm{p(a)}$ 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.