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.

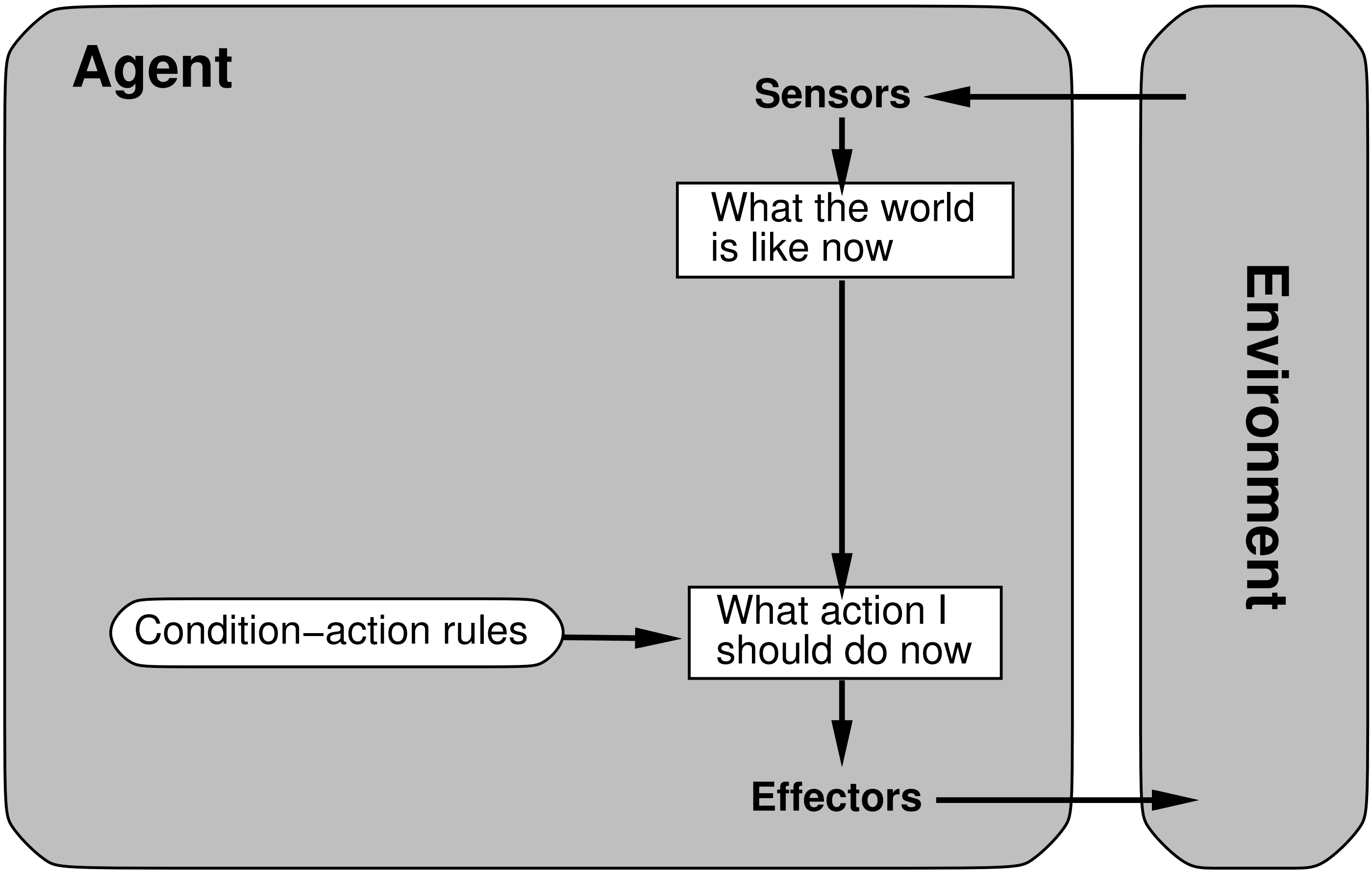
Luvun 1.6 aivoproteesikoe voitaisiin tehdä myös taulukolla mikropiirin sijaan — jolloin taulukko olisi tietoinen!

2.2. Muistittomat heijastetoimijat

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

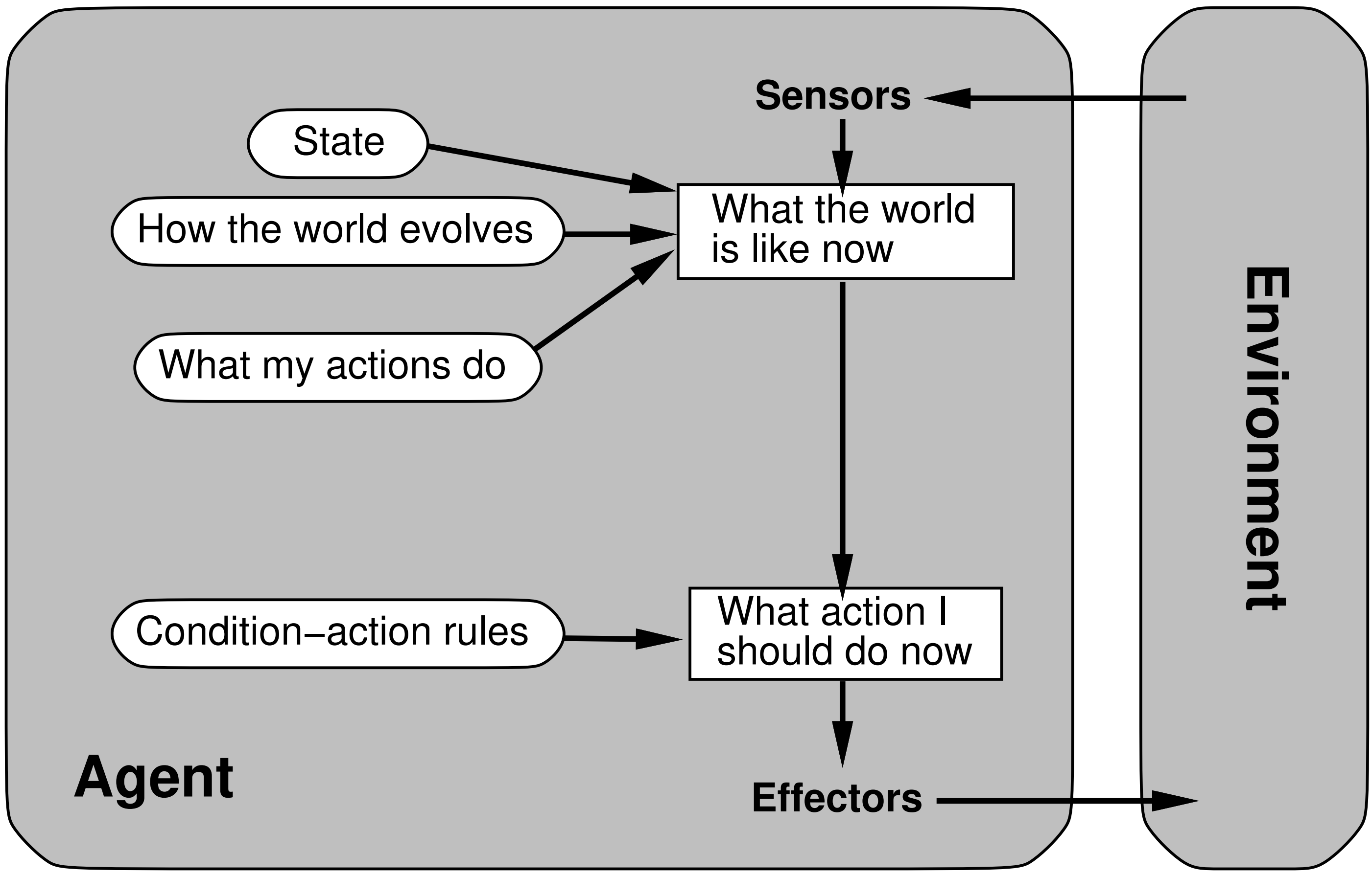
  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state, rules*)
  action ← RULE-ACTION[*rule*]
  return *action*

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.



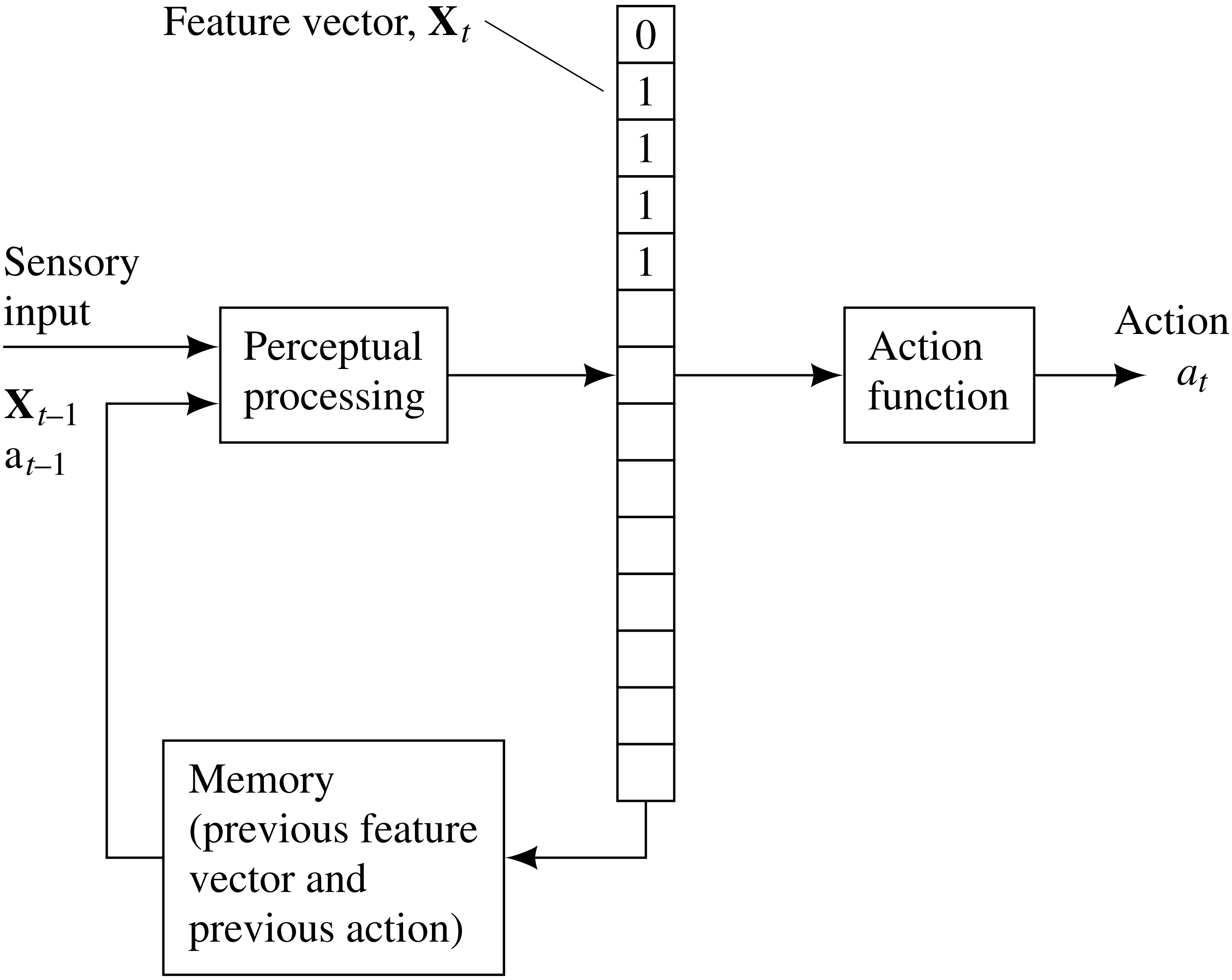

```
function REFLEX-AGENT-WITH-STATE(percept) returns action
   static: state, a description of the current world state
           rules, a set of condition-action rules

   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.

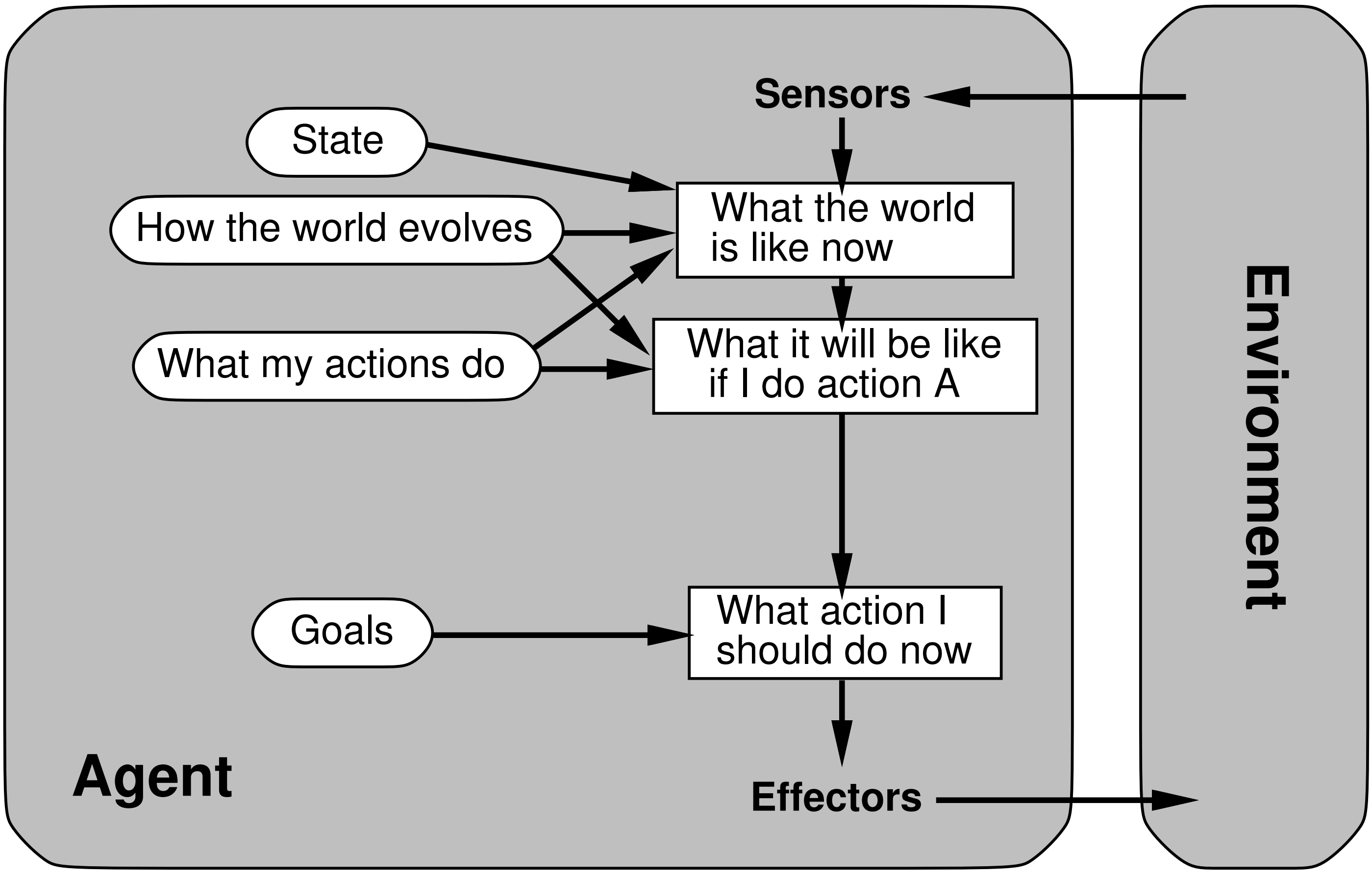


Kuva 8: Piirrevektorin muistava tilakone [Nil98, kuva 5.1].

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

2.4. Päämäärähakuiset toimijat

Luvun 2.2 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:

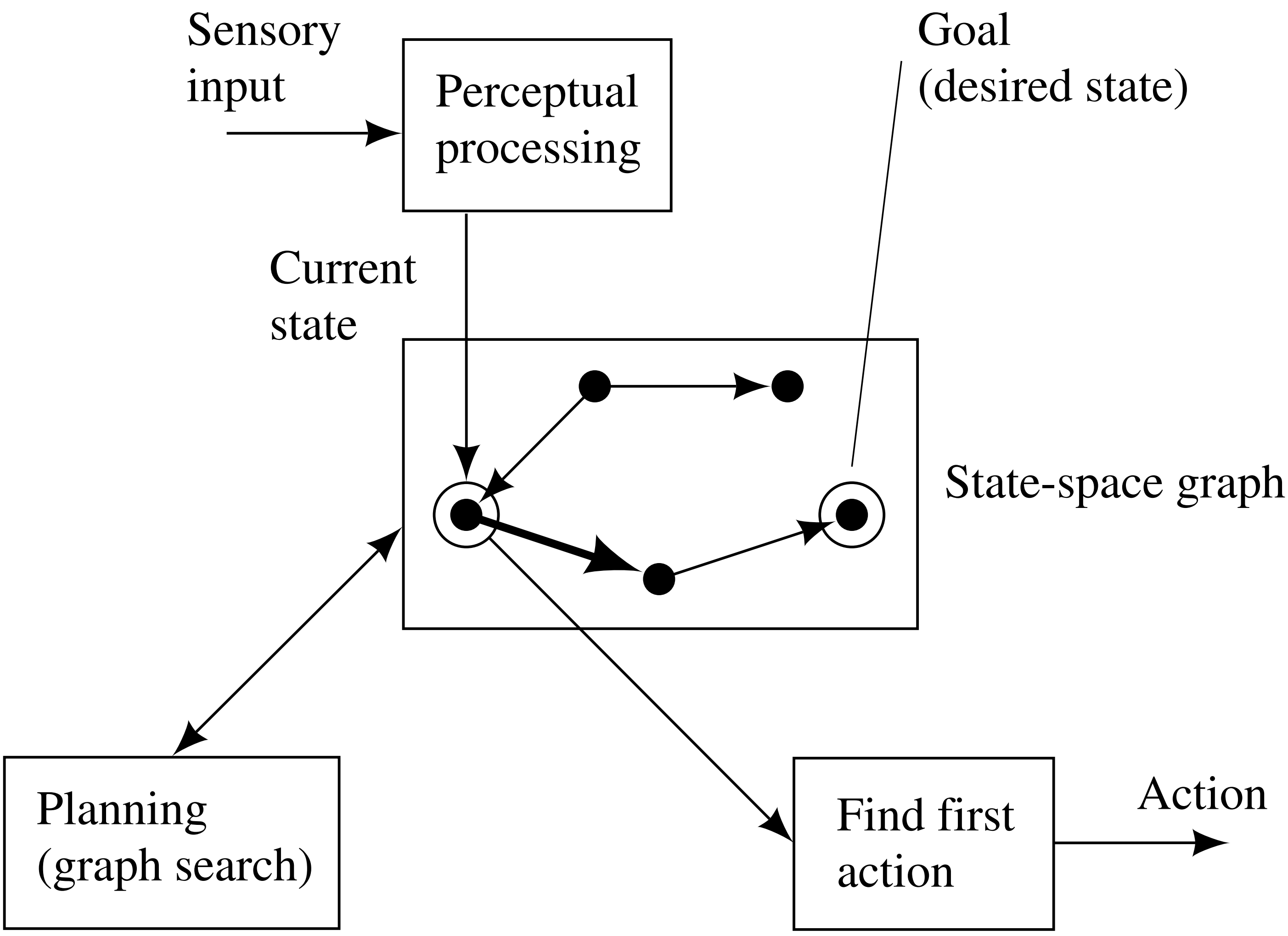
- Samat säännöt saattavat soveltua vaikka tavoitteet vaihtuisivatkin; tavoite voi muuttaa sitä ehtoa, jonka perusteella tekoja pidetään hyvinä tai huonoina.
- Sama tavoite saattaa olla toimijan saavutettavissa vaikka säännöt muuttuisivatkin; toimija voi vaihtaa menetelmää.
- Toimija järkeilee ehdollisesti: "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.

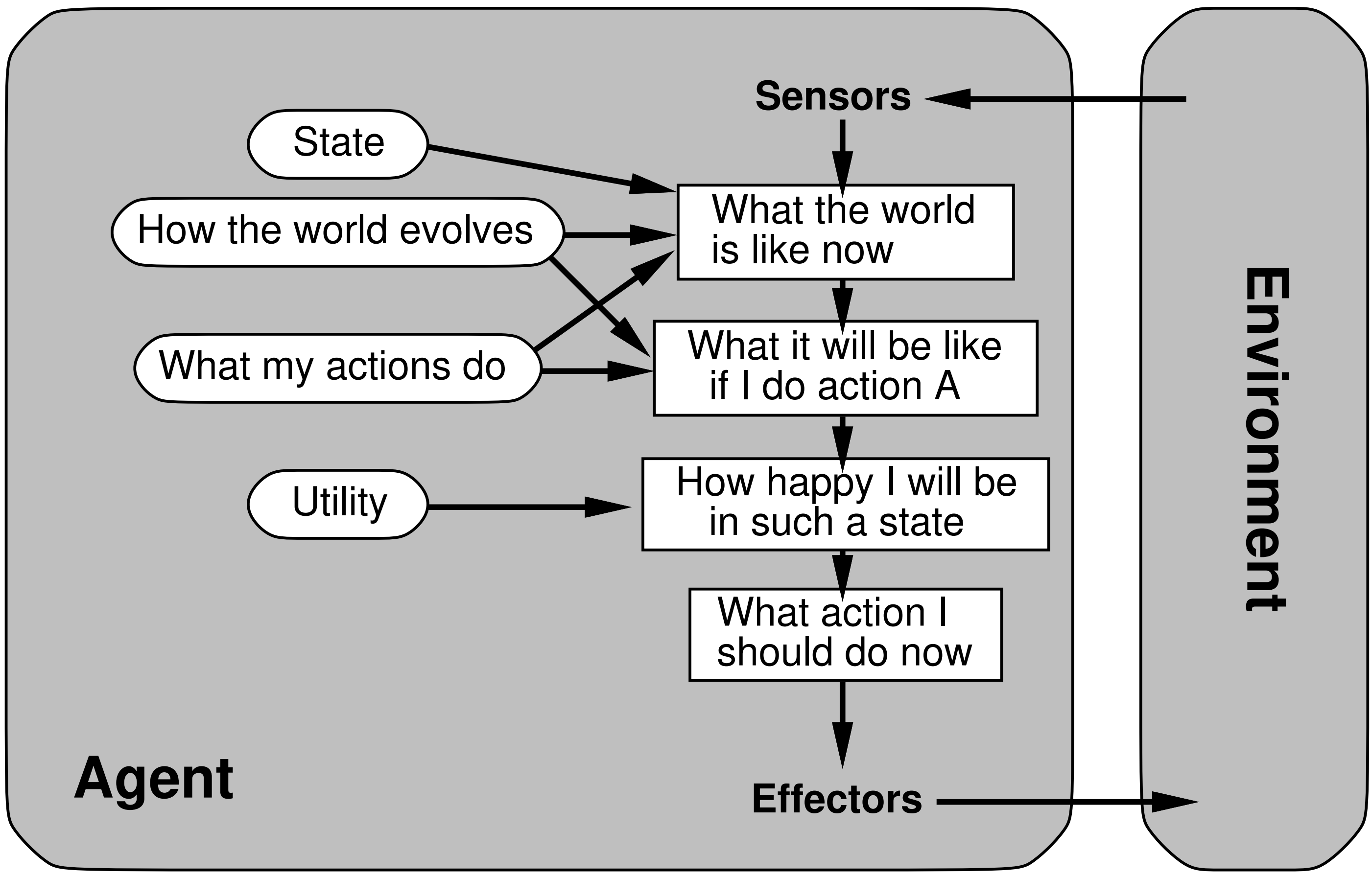
Kurssimme käsitteleekin tämän vuoksi etsintää ja suunnittelua symbolisin välinein.



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

2.5. Hyötyohjatut toimijat

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.

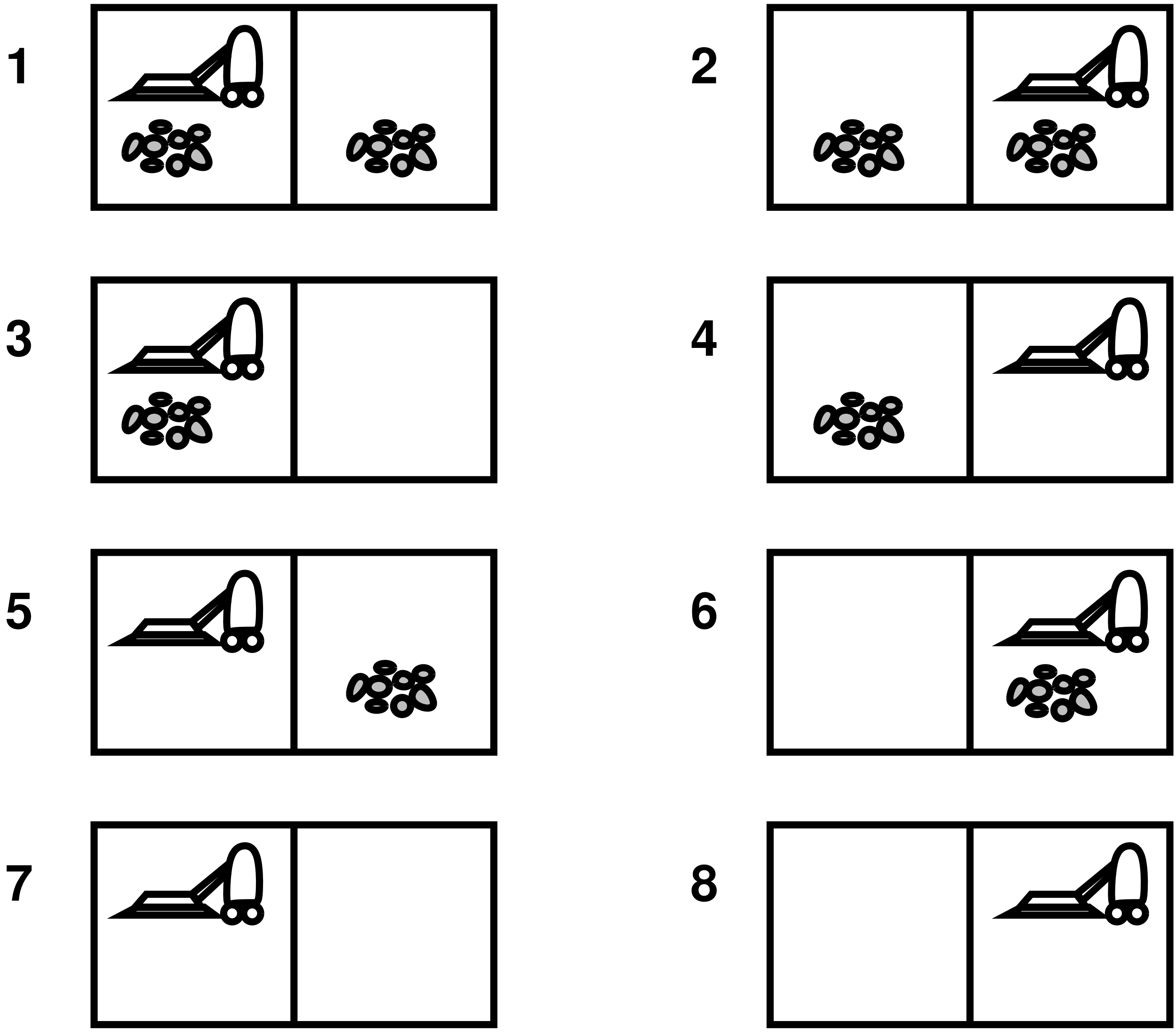
S: Jos siinä ruudussa, missä imuri on nyt, on likaa, niin se imetään pois.

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].

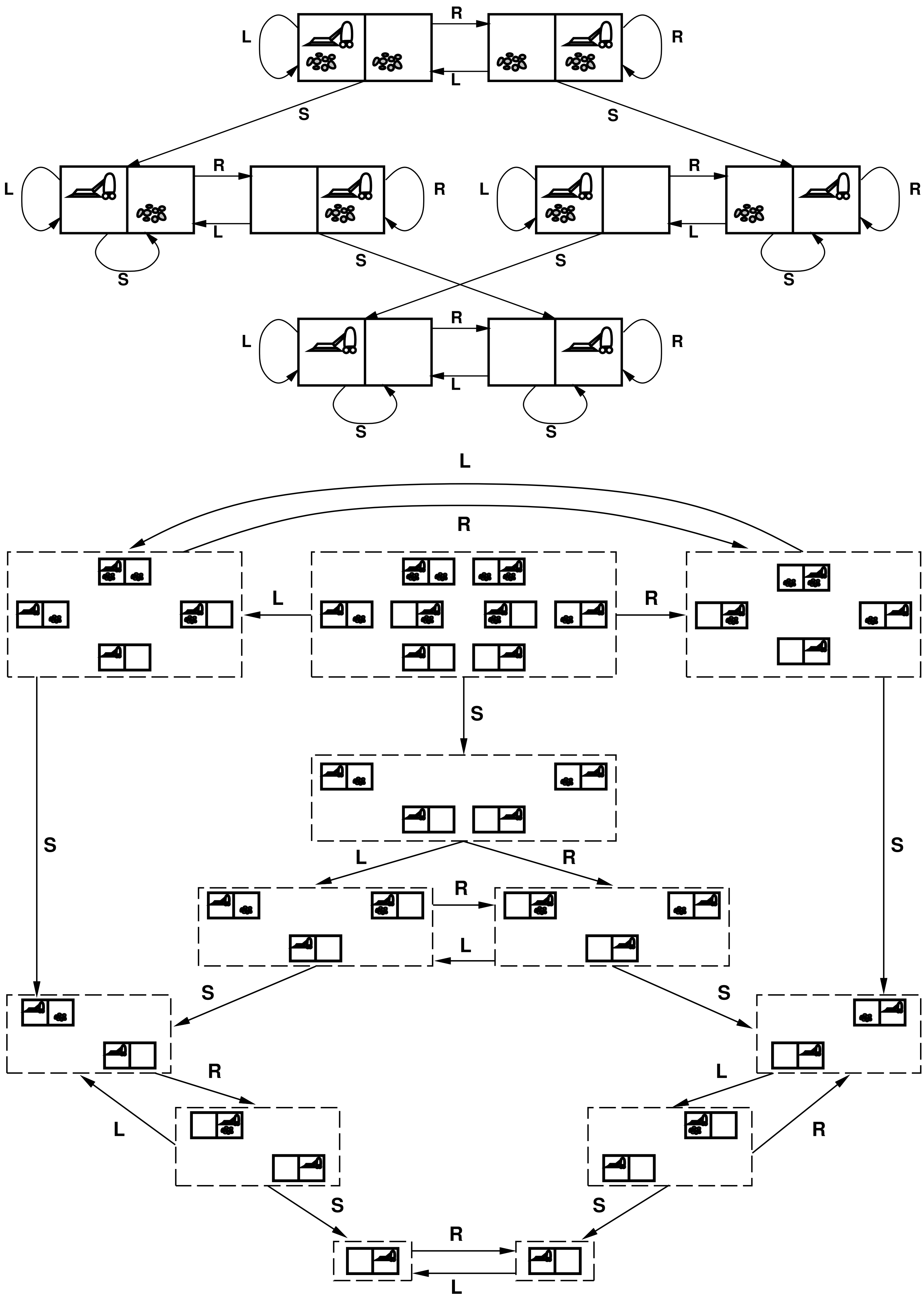
- Jos ympäristö on imurin *saavutettavissa* ja *deterministinen*, niin silloin etsinnän tilat yhtyvät ympäristön tiloihin (kuvan 14 yläpuolisko).

  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 epädeterministisen automaatin simulointi!

- Epädeterministinen 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:

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

  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