12 KEHITTYNEITÄ SUUNNITTELUMENETELMIÄ

- Tavanomaisissa suunnittelutehtävissä tekojen ajallisella kestolla ei ole merkitystä, mutta todellisissa ongelmissa myös tekojen kesto täytyy ottaa huomioon.
- *Skedulointiongelmat* ovat ongelmia, joissa tehtäville täytyy löytää suoritusjärjestys ja *aikataulu* (schedule), siten että *resurssirajoitteet* ovat voimassa ja tekojen *kokonaiskesto* on mahdollisimman lyhyt.
- Suunnittelukieliä laajennettava toimintojen kestoilla ja resurssivaatimuksilla. Esim:

$Action(Inspect(c),$
PRECOND:$EngineIn(c) \wedge WheelsOn(c),$
EFFECT:$Done(c) \wedge Duration(10),$
RESOURCE:$Inspectors(1))$

- Tämän jälkeen voidaan ensin tuottaa osittainjärjestetty suunnitelma ja sitten laskea kullekin *polulle* eli tekojen sarjalle teosta *Start* tekoon *Finish* aikaisimman mahdollisen aloitusajankohdan ES ja myöhäisimmän mahdollisen aloitusajankohdan LS seuraavien sääntöjen mukaan:

$ES(\textit{Start}) = 0.$
$ES(B) = \max_{A \prec B} ES(A) + \textit{Duration}(A).$
$LS(\textit{Finish}) = ES(\textit{Finish}).$
$LS(A) = \min_{A \prec B} LS(B) - \textit{Duration}(A).$

- Laskenta tapahtuu dynaamista ohjelmointia käyttäen ajassa $O(Nb)$, missä N on tekojen lukumäärä ja b on suurin mahdollinen haarautumiskerroin (teosta tai tekoon) suunnitelmassa.
- Näin saadaan selville lyhimmän tekojärjestyksen tekojen mahdolliset aloitusajankohdat ja koko suunnitelman toteuttamisen kokonaiskesto, joka on sama kuin *kriittisen polun* (critical path) kesto.

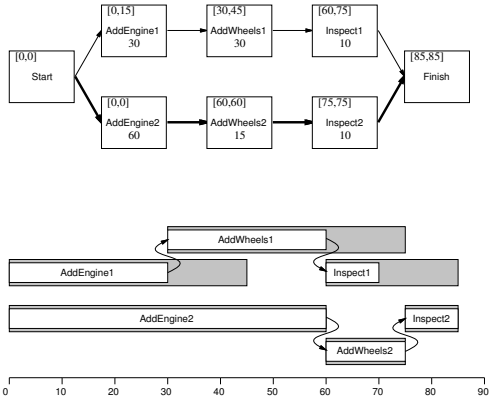
ESIMERKKI

$Init(Chassis(C_1) \wedge Chassis(C_2) \wedge Engine(E_1, C_1, 30) \wedge$
$Engine(E_2, C_2, 60) \wedge Wheels(W_1, C_1, 30) \wedge Wheels(W_2, C_2, 15))$
$Goal(Done(C_1) \wedge Done(C_2))$

$Action(AddEngine(e, c, m)),$
PRECOND:$Engine(e, c, d) \wedge Chasssis(c) \wedge \neg EngineIn(c),$
EFFECT:$EngineIn(c) \wedge Duration(d))$

$Action(AddWheels(w, c),$
PRECOND:$Wheels(w, c, d) \wedge Chassis(c),$
EFFECT:$WheelsOn(c) \wedge Duration(d))$

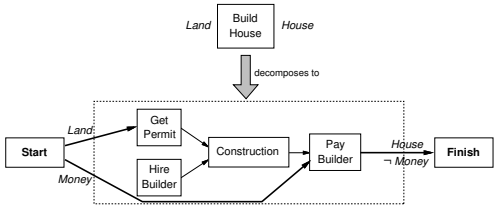
$Action(Inspect(c),$
PRECOND:$EngineIn(c) \wedge WheelsOn(c) \wedge Chassis(c),$
EFFECT:$Done(c) \wedge Duration(10))$



- Resurssien huomioonottaminen vaikeuttaa skedulointia huomattavasti.
- *Kulutettavat* (consumable) resurssit voidaan käsitellä teon ennakkoehtoina kuten tavallisessa suunnittelussa, mutta *uudelleenkäytettävät* (reusable) resurssit toimivat sekä teon ennakkoehtoina että *tilapäisinä vaikutuksina*: resurssi on varattuna toiminnan keston ajan, minkä jälkeen se palautuu muiden toimintojen käyttöön.
- Optimaalisen aikataulun löytäminen osittainjärjestetyn suunnitelman pohjalta on NP-kova ongelma kun resurssirajoitteet pitää ottaa huomioon.

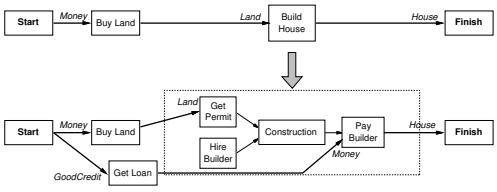
HIERARKKINEN SUUNNITTELU

- *Hierarkkisessa suunnittelussa* aloitetaan korkean tason suunnitelmasta jossa monimutkaisetkin teot kuvataan yksittäisinä operaatioina, esim. *Build House*.
- Monimutkaisia tekoja tarkennetaan soveltamalla *dekompositiota* kunnes suunnitelma saadaan hienonnettua sellaiseksi että se koostuu pelkistä yksinkertaisista matalan tason teoista (primitive actions).



- Suunnittelumenetelmää, jossa suunnitelma rakennetaan ainoastaan dekompositiota käyttäen, kutsutaan HTN-suunnittelu (hierarchical task networks). Kirja keskittyy hybridimenetelmään, joka yhdistää HTN- ja POP-tyyppistä suunnittelua lisäämällä dekomposition yhdeksi askeleeksi osittain järjestetyn suunnitelman konstruointia.
- *Suunnitelmakirjasto* sisältää korkean tason tekokuvausten tarkennukset osittainjärjestettyinä suunnitelmina. Kirjasto koostuu siis lauseista muotoa $Decompose(a, d)$, missä a on teko ja d osittainjärjestetty suunnitelma.
- Kirjastossa voi olla useita erilaisia täydellisiä ja konsistentteja suunnitelmia samalle teolle. Konstruoituja osasuunnitelmia voidaan tallettaa kirjastoon myöhempää käyttöä varten ($\Rightarrow$ oppiminen).

- Korkean tason suunnitelma piilottaa yksityiskohtia jotka tulevat esiin vasta kun suunnitelmaa tarkennetaan. Tällainen informaation piilottaminen helpottaa korkean tason suunnitelman tekemistä.
- Toisaalta suunnitelman hienontaminen voi paljastaa uusia ennakkoehtoja tai vaikutuksia, mikä voi aiheuttaa konflikteja tai vaatia uusien askelten lisäämistä suunnitelmaan.



- Teoreettisesti tarkasteltuna puhdas HTN-suunnittelu on laskennallisesti raskasta tai jopa ratkeamatonta jos rekursion käyttöä ei rajoiteta.
- Kuitenkin korkean tason ongelman jakaminen helpommin ratkottaviin osaongelmiin ja erityisesti suunnitelmakirjaston käyttö helpottavat suunnittelua huomattavasti.
- Erityisesti mahdollisuus käyttää asiantuntijoiden tietämystä suunnitelmakirjaston luomisessa ovat tehneet hierarkkisista menetelmistä suosittuja suunnittelumenetelmiä. Lähes kaikki suuria käytännön ongelmia ratkovat suunnittelumenetelmät perustuvat HTN-suunnitteluun.

EPÄDETERMINISTISISET YMPÄRISTÖT

- Tähän mennessä on tarkasteltu vain *klassista suunnittelua*, jossa ongelmaympäristö on ollut täysin havaittavissa, staattinen ja deterministinen. Lisäksi on oletettu että toimintokuvaukset ovat täydellisiä ja toimivat aina.
- Todellisen maailman ongelmissa suunnitelmaa ei voi yleensä tehdä valmiiksi etukäteen ja sitten vain toteuttaa valittu toimintasarja.
- Ympäristöä koskeva informaatio voi olla joko *epätäydellistä* (incomplete) tai *virheellistä* (incorrect).
- Ympäristön epädeterministisyys voi olla *rajoitettua* tai *rajoittamatonta*. Rajoitetulla epädeterministisyydellä tarkoitetaan sitä että tekojen mahdolliset seuraukset voidaan luetella: esimerkiksi kolikon heitto voi tuottaa kruunan tai klaavan.

- Rajoitettua epädeterministisyyttä voidaan pyrkiä hallitsemaan tekemällä suunnitelmia jotka pääsevät haluttuun lopputuloksiin *kaikissa mahdollisissa olosuhteissa*. Tavoitteena on siis tehdä suunnitelmia jotka pakottavat ympäristön tiettyyn tilaan ilman että ympäristöä tarvitsisi välillä havainnoida (sensorless planning).
- Toinen tapa on käyttää *ehdollista suunnittelua* (conditional planning). Siinä suunnitelmiin sisällytetään ehtolauseita, siten että havainnoista riippuen voidaan toteuttaa suunnitelma eri tavalla: esim.
  if $At(L) \wedge Clean(L)$ **then** $Right$ **else** $Suck$.

- Rajoittamatonta epädeterministisyyttä ei voi hallita pelkästään edellä mainituilla menetelmillä, vaan tarvitaan *toimintojen seurantaa* (execution monitoring) ja *uudelleensuunnittelua* (replanning). Mikäli agentti huomaa suoritusaikana, että suunnitelma ei tuota haluttua tulosta, se voi keskeyttää suunnitelman suorituksen ja käynnistää uuden suunnitteluprosessin joka soveltuu muuttuneeseen tilanteeseen.
- *Jatkuva suunnittelu* (continuous planning) ei lopu tavoitetilan toteutumiseen vaan toimii agentin koko eliniän etsien kussakin tilanteessa parasta mahdollista suunnitelmaa agentin tavoitteiden saavuttamiseksi. Jatkuva suunnitelija osaa reagoida sekä ympäristön muutoksiin että agentin tavoitteiden vaihtumiseen.

MONIAGENTTISUUNNITTELU

- Monen agentin ympäristöissä muiden agenttien toiminta on otettava huomioon.
- Kyseessä voi olla joko yhteistyötilanne tai kilpailutilanne. Yksinkertaisia kilpailutilanteita (kahden pelaajan nollasummapelejä) on käsitelty jo aiemmin. Monimutkaisten kilpailutilanteiden analysointiin ja käsittelyyn käytetään tavallisesti *peliteoriaa* (game theory). Tavoitteena on pystyä ennakoimaan muiden tekoja ennen oman teon valintaa.
- Yhteistyötilanteessa agenteilla on *yhteinen tavoite* (joint goal), ja pyritään löytämään *yhteinen suunnitelma* (joint plan), joka määrittelee tarvittavat teot kullekin agentille.
- Koska monen agentin suunnitelmia voi olla useita, tarvitaan *koordinointia* takaamaan että kaikki agentit noudattavat samaa suunnitelmaa. Tämä voi vaatia *kommunikointia*.
- Lisäksi pitäisi olla *yleistä tietoa* (common knowledge) että ollaan toteuttamassa jotakin tiettyä suunnitelmaa.