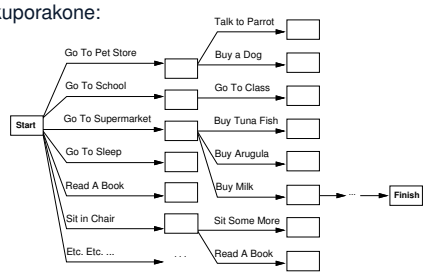
11 TOIMINTOJEN SUUNNITTELU

- *Suunnittelulla* (planning) tarkoitetaan tavoitteen saavuttavan toimintasarjan löytämistä. Samaa siis kuin haku aiemminkin, mutta hakualgoritmit eivät skaalaudu todellisen maailman ongelmiin. Toimintavaihtoehtoja on yksinkertaisesti liikaa.
- Esim. Tehtävänä on hankkia maitoa, banaaneja ja akkuporakone:



- Suunnittelussa yhdistetään etsintää ja loogista tietämyksenesittämistä määrittelemällä teot ja tavoitetilat logiikkaan perustuvalla kielellä.
- Teoilla on *ennakkoehdot* (precondition), joiden pitää olla voimassa jotta teko voidaan suorittaa, ja *vaikutukset* (effect), jotka tulevat voimaan kun teko suoritetaan.
- 2 esim. olkoon ongelmana kirjan AIMA ostaminen: tekoja Buy(ISBN??????????) on noin 10 miljardia mutta jos tavoitteena on Have(ISBN0130803022) ja tietämyspohjainen agentti tietää että teko Buy(x) johtaa Have(x):ään niin agentti voi pelkällä yhdellä samastuksella päätellä että oikea teko on Buy(ISBN0130803022).
- Jos tavoitetila voidaan jakaa riippumattomiin osatavoitteisiin, niitä voidaan ratkoa mielivaltaisessa järjestyksessä.
- Saadaan yleispätevä heuristiikka joka ohjaa etsintää: toteutuneiden osatavoitteiden lukumäärä.

SUUNNITTELUKIELET

- Suunnittelukieliä on kehitetty useita erilaisia. Kielen tulisi olla riittävän ilmaisuvoimainen, mutta se ei saisi olla liian monimutkainen jottei suunnittelusta tule täysin tehotonta.
- STRIPS (STanford Research Institute Problem Solver)
  - Tilojen esittäminen: positiivisten, vain vakiotermejä sisältävien literaalien konjunktio.
  - Maalitilan esittäminen: positiivisten, vain vakiotermejä sisältävien literaalien konjunktio.
  - Toimintojen esittäminen: Toiminnoilla on ennakkoehdot ja vaikutukset, joissa voi olla (eksistentiaalisesti kvantifioituja) muuttujasymboleitakin. Ennakkoehdoissa mainittujen muuttujien on esiinnyttävä myös toiminnon parametreissä. Vaikutukset voivat olla positiivisia literaaleja, jotka siis muuttuvat tosiksi, tai negatiivisia literaaleja, joilla tarkoitetaan literaalin muuttamista epätodeksi.

- Esim. *Action(Fly(p, from, to),*
  PRECOND: *At(p,from)* $\wedge$ *Plane(p)* $\wedge$ *Airport(from)* $\wedge$ *Airport(to)*
  EFFECT: $\neg$*At(p,from)* $\wedge$ *At(p,to))*
- Vaikutukset voidaan joskus antaa erikseen *lisäyslistana* (add list) ja *poistolistana* (delete list).
- Teko on *sovellettavissa* (applicable) jos sen ennakkoehdot toteutuvat.
- Teon soveltamisen tilassa s *tulos* (result) on tila, joka on samanlainen kuin s mutta jokainen positiivinen vaikutus on lisätty ja jokainen negatiivinen on poistettu. STRIPS-oletus: kaikki muut säilyvät ennallaan $\Rightarrow$ kehysongelmasta ei tarvitse huolehtia.
- Suunnitteluongelman *ratkaisu* (solution) on toimintoketju, jonka suorittaminen alkutilassa johtaa tilaan, jossa maali toteutuu.

- Koska STRIPS-kielessä ei ole funktiosymboleja, mikä tahansa toimintoskeema on muunnettavissa äärelliseksi propositionaalisten toimintokuvausten joukoksi, mikä yksinkertaistaa suunnittelua.
- STRIPS on kuitenkin osoittautunut liian rajoitetuksi kieleksi monille käytännön suunnitteluongelmille ja siksi on kehitetty ilmaisuvoimaisempia kieliä kuten ADL.
- Esim. *Action(Fly(p : Plane, from : Airport, to : Airport),*
  PRECOND: *At(p,from)* $\wedge$ *(from* $\neq$ *to)*
  EFFECT: $\neg$*At(p,from)* $\wedge$ *At(p,to))*
- ADL sopii jo useisiin realistisiin ongelmiin. Molemmissa on kuitenkin yhä ongelmia ramifikaatio-ongelman kanssa (implisiittisten vaikutusten käsittely). Myös *kvalifikaatio-ongelma* (qualification problem) on hankala: sellaiset olosuhteet joita ei ole esitetty mutta jotka voivat aiheuttaa jonkin toiminnon epäonnistumisen.

SUUNNITTELUALGORITMIT

- Suunnittelua voidaan tehdä etsintänä tila-avaruudessa. Etsintää voidaan tehdä joko eteenpäin tai taaksepäin.
- Etsintää eteenpäin kutsutaan *progressiiviseksi* suunnitteluksi.
- Suunnittelutehtävä voidaan esittää etsintäongelmana seuraavasti:

  Alkutila: Etsinnän alkutilana on suunnittelutehtävän alkutila (positiivisten perusliteraalien joukko)

  Toiminnat: ovat kussakin tilassa ne joiden ennakkoehdot täyttyvät ja geneerinen seuraajafunktio toimii lisäämällä tilaan positiiviset seuraukset ja poistamalla negatiiviset.

  Maaliehto: Suunnittelutehtävän maaliehto.

  Polkuhinta: kunkin teon hinta on tavallisesti 1.
- Koska funktiosymboleita ei käytetä, tila-avaruus on äärellinen ja voidaan käyttää mitä tahansa täydellistä (verkko)etsintäalgoritmia.

- Tila-avaruuden laajuuden takia kannattaa usein pyrkiä käyttämään jotakin informoitua menetelmää kuten A*, jolloin täytyy vielä kehittää hyvä heuristiikka.
- Etsintä taaksepäin eli *regressiivinen* etsintä aloittaa etsinnän maalitilasta vastaavasti kuin tehtiin kaksisuuntaisessa etsinnässä.
- Siellä vaikeutena oli edeltäjätilojen löytäminen, mutta STRIPSissä se on helppoa.
- Nyt haku voidaan keskittää vain *relevantteihin* toimintoihin, millä tarkoitetaan että toiminta tekee jonkin konjunktiivisen maalin literaaleista todeksi. Jos ratkaisu nimittäin on olemassa, se löytyy taaksepäin etsinnällä keskittymällä relevantteihin toimintoihin.
- Regressiivisessä suunnittelussa haarautumiskerroin pysyykin usein pienempänä kuin progressiivisessa suunnittelussa.

- Lisäksi täytyy vaatia että toiminta on *konsistentti* siinä mielessä ettei se tee tyhjäksi mitään maalitilan literaalia. Jos tällainen relevantti ja konsistentti toiminta löytyy, niin edeltäjätila saadaan konstruoitua
  1. poistamalla maalitilasta toiminnan positiiviset vaikutukset
  2. lisäämällä kaikki toiminnan ennakkoehdot (paitsi jos ne jo esiintyvät)
- Mitä tahansa täydellistä etsintämenetelmää voidaan käyttää etsintään. Haku lopetetaan kun löytyy edeltäjätila joka toteuttaa suunnittelutehtävän alkutilan kuvauksen.
- Ensimmäisen kertaluvun tapauksessa toteuttaminen voi vaatia muuttujasijoituksen tekemistä.

- Pelkkä etsintä jompaan kumpaan suuntaan on tehotonta ilman hyvän heuristisen funktion käyttöä. Ideana on arvioida tarvittavien toimintojen määrää, jolloin voitaisiin käyttää A*-algoritmia optimaalisen ratkaisun löytämiseen.
- Heuristiikka voidaan löytää väljentämällä ongelmaa kuten todettiin aiemmin kurssilla. Tällöin heuristiikka on luvallinen.
- Toinen tapa on tehdä oletus osatavoitteiden riippumattomuudesta ja käyttää konjunktiivisen tavoitteen arviona kaikkien konjunktien kustannuksien summaa. Tämä voi johtaa joko optimistiseen tai pessimistiseen arvioon (jälkimmäisessä tapauksessa heuristiikasta ei tule luvallista).
- Ongelman väljentäminen voidaan tehdä helposti STRIPS-muotoisessa esityksessä esimerkiksi poistamalla tekojen ennakkoehdot kokonaan jolloin täytyy vain valita teot jotka tekevät kunkin maaliehdon konjunktin todeksi.

- Toinen tapa on poistaa kaikki negatiiviset tulokset toiminnoista, jolloin ei tarvitse huolehtia osasuunnitelmien negatiivisista vuorovaikutuksista koska mikään toiminto ei voi tehdä tyhjäksi toisen tavoitetta. Tätä kutsutaan *tyhjän poistolistan* heuristiikaksi (empty delete list). Heuristiikka on luvallinen ja melko tarkka, mutta vaatii suunnittelualgoritmin käyttämistä ensin väljennetylle ongelmalle, mikä kuitenkin on usein niin nopeaa että heuristiikan käyttäminen kannattaa.
- Heuristiikkoja voidaan käyttää sekä progressio- että regressiosuunnittelussa. Yleisimmin käytössä on progressiosuunnittelu yhdessä tyhjä poistolista -heuristiikan kanssa.
- Yleisesti STRIPS-tyylinen suunnittelu on PSPACE-täydellistä, mutta käytännössä suunnitelmia pystytään tekemään tässä esitetyillä heuristisilla menetelmillä.

OSITTAINJÄRJESTETTY SUUNNITTELU

- Edellä kuvatut menetelmät tuottavat *täydellisesti järjestetyn* suunnitelman. *Osittainjärjestetyssä suunnittelussa* (partial-order planning) vältetään sitoutumista tekojen järjestykseen mahdollisimman pitkään.
- Osittainjärjestetty suunnitelma voidaan sitten *linearisoida* millä tahansa tavalla, joka säilyttää osittainjärjestetyn suunnitelman tekojen järjestyksen.
- Esimerkki: Kenkien pukeminen:

Goal(RightShoeOn $\wedge$ *LeftShoeOn)*
Init()
Action(RightShoe, PRECOND:*RightSockOn,*
EFFECT:*RightShoeOn)*
Action(RightSock, EFFECT:*RightSockOn)*
Action(LeftShoe, PRECOND:*LeftSockOn,* EFFECT:*LeftShoeOn)*
Action(LeftSock, EFFECT:*LeftSockOn)*

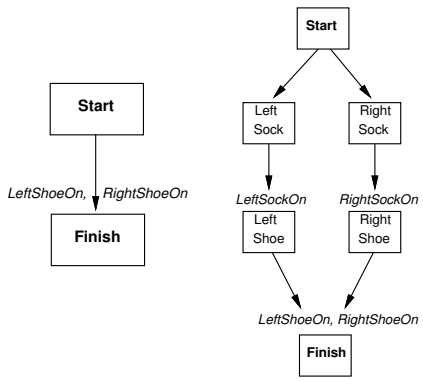
- Suunnitelmat koostuvat seuraavista komponenteista:

Teot: esitetään kuten tavallisesti, mutta lisäksi kaksi erityistä tekoa: *Start* ja *Finish*, jotka ovat jokaisessa suunnitelmassa. Teolla *Start* ei ole ennakkoehtoja ja sen seuraukset ovat kaikki ongelman alkutilan literaalit. Teolla *Finish* ei ole vaikutuksia, mutta sen ennakkoehtoina ovat ongelman maalitilan literaalit.

Järjestysrajoitteet (ordering constraints): Rajoitteet ovat muotoa $A \prec B$, mikä tarkoittaa että A suoritetaan ennen B:tä.

Kausaaliset linkit (causal links): Tekojen välinen kausaalilinkki $A \xrightarrow{p} B$ tarkoittaa että teko A saa aikaan p:n, joka on B:n ennakkoehto. Jos tekojen välillä on kausaalilinkki, niiden väliin ei saa sijoittaa tekoa, joka on *konfliktissa* sen kanssa, siis sellaista tekoa, jonka vaikutus on $\neg p$.

Avoimet ennakkoehdot (open preconditions): Toteutumattomien ennakkoehtojen joukko. Alussa maalitilan literaalit.



- Valmis suunnitelma koostuu seuraavista:

Teot: { *RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish* }

Järjestysrajoitteet: { *RightSock* $\prec$ *RightShoe*, *LeftSock* $\prec$ *LeftShoe* }

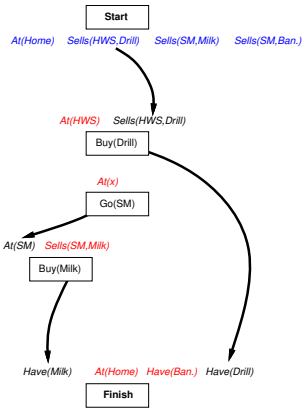
Linkit: { $RightSock \xrightarrow{RightSockOn} RightShoe$, $LeftSock \xrightarrow{LeftSockOn} LeftShoe$, $RightShoe \xrightarrow{RightShoeOn} Finish$, $LeftShoe \xrightarrow{LeftShoeOn} Finish$ }

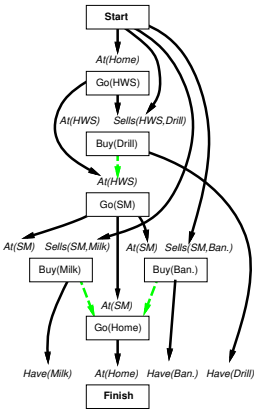
Avoimet ennakkoehdot: $\emptyset$

- *Konsistentti suunnitelma* on sellainen, jonka järjestysrajoitteissa ei ole syklejä eikä kausaalilinkeissä ole konflikteja.
- Suunnitteluongelman *ratkaisu* on konsistentti suunnitelma, jossa ei ole avoimia ennakkoehtoja.

SUUNNITELMAN ETSINTÄ

1. Alussa suunnitelmassa on vain teot *Start* ja *Finish*, rajoite *Start* $\prec$ *Finish* ja avoimina teon *Finish* ennakkoehdot.
2. Valitaan tarkasteltavaksi jokin avoin ennakkoehto p teolle B ja etsitään sellainen teko A, joka saa aikaan että p.
3. Lisätään kausaalilinkki $A \xrightarrow{p} B$ ja järjestysrajoite $A \prec B$. Jos teko A ei ennestään ollut suunnitelmassa, lisätään se suunnitelmaan ja lisätään myös rajoitteet *Start* $\prec A$ ja $A \prec$ *Finish*.
4. Selvitetään mahdolliset konfliktit uuden kausaalilinkin ja muiden suunnitelmassa olevien tekojen ja kausaalilinkkien välillä. Esimerkiksi jos teko C on konfliktissa linkin $A \xrightarrow{p} B$ kanssa, se siirretään suojatun alueen ulkopuolelle lisäämällä joko rajoite $B \prec C$ tai $C \prec A$.
5. Jos ennakkoehtoa ei pystytä toteuttamaan tai konfliktia ei pystytä selvittämään, peruutetaan etsinnässä taaksepäin.
6. Jos vielä on avoimia ennakkoehtoja, palataan kohtaan 2.





POP-ALGORITMI

function POP(*initial*, *goal*, *operators*) **returns** *plan*

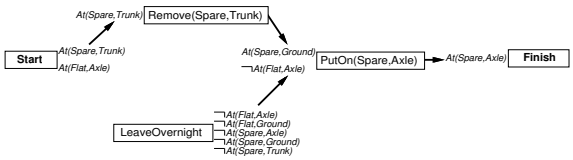
plan ← MAKE-MINIMAL-PLAN(*initial*, *goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  $S_{need},\ c \leftarrow$ SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, *c*)
  RESOLVE-THREATS(*plan*)
end

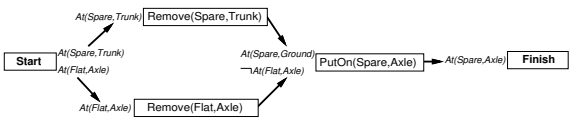
function SELECT-SUBGOAL(*plan*) **returns** $S_{need},\ c$

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return $S_{need},\ c$

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, *c*)
  choose a step S_{add} from *operators* or STEPS(*plan*) with c as an effect
  if there is no such step **then fail**
  add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
  add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
  if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

procedure RESOLVE-THREATS(*plan*)
  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
    choose either
      Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
      Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
    if not CONSISTENT(*plan*) **then fail**
  end

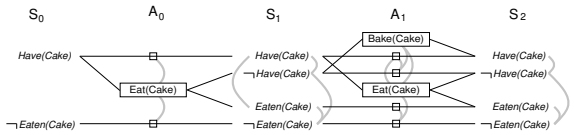




SUUNNITTELUVERKOT

- *Suunnitteluverkkoa* (planning graph) voidaan käyttää sekä heuristiikan löytämiseen että itse ratkaisun konstruoimiseen.
- Suunitelmaverkot toimivat ainoastaan propositionaalisille ongelmille joissa ei ole muuttujia. (Sekä STRIPS- että ADL-muotoiset ongelmaesitykset voidaan propositionalisoida, vaikkakin esitysten koko kasvaa tällöin melko suureksi.)
- Verkossa on tasoja, jotka vastaavat suunnitelman aika-askeleita.
- Kukin taso käsittää literaaleja, jotka voisivat olla tosia, ja toimintoja, joiden esiehdot voisivat olla täytettyjä.
- Tilataso S_0 vastaa ongelman alkutilaa
- Toimintotasolla A_0 on kaikki ne toiminnot, joiden ennakkoehdot edellinen taso täytti.

- Jokainen toiminto on kytketty ennakkoehtoihinsa tasolla S_0 ja vaikutuksiinsa tasolla S_1
- Jotta voitaisiin esittää literaalien säilymistä muuttumattomana, verkkoon lisätään *persistenssitoimintoja*, joiden avulla literaalit siirtyvät tasolta toiselle.
- Tasoja muodostetaan kunnes kaksi peräkkäistä tasoa S_i ja S_{i+1} ovat identtiset, minkä jälkeen verkon tasot eivät enää muutu.



- Sellaisille toiminnoille ja literaaleille, jotka eivät voi toteutua samanaikeisesti, merkitään verkkoon *poissulkemis-* l. *mutex-suhteita* (mutual exclusion). Esim. *Eat(Cake)* ei voi tapahtua samaan aikaan kun *Have(Cake)* pysyy voimassa, tai ¬ *Eaten(Cake)* pysyy voimassa.
- Mutex-suhde on voimassa kahden toiminnan välillä jos jokin seuraavista on voimassa:
  1. Ristiriitaiset vaikutukset: toisen toiminnon vaikutus on toisen vaikutuksen negaatio. Esim. Teolla *Eat(Cake)* ja *Have(Cake)*:n persistenssillä on vastakkaiset tulokset.
  2. Häirintä (tai interferenssi): toisen toiminnon vaikutus on toisen ennakkoehdon negaatio. Esim. *Eat(Cake)* kumoaa *Have(Cake)*:n persistenssin ennakkoehdon.
  3. Ristiriitaiset vaatimukset (competing needs): toisen ennakkoehto on toisen ennakkoehdon negaatio. Esim. Bake(Cake) ja Eat(Cake) kilpailevat keskenään Have(Cake):n arvosta.

- Mutex-suhde on voimassa kahden literaalin välillä jos
  1. toinen on toisen negaatio tai
  2. jos toiminnot joilla literaalit voitaisiin saavuttaa ovat keskenään mutex-suhteessa.
- Esimerkiksi *Have(Cake)* ja *Eaten(Cake)* ovat mutex-suhteessa tasolla S_1, koska ainoa *Have(Cake)*:n toteuttava toiminto eli sen persistenssi ja ainoa *Eaten(Cake)*:n toteuttava toiminto eli *Eat(Cake)* ovat mutex-suhteessa.
- Tasolla S_2 ne eivät enää ole mutex, koska esim. *Bake(Cake)* ja *Eaten(Cake)*:n persistenssi eivät ole mutex-suhteessa.
- Suunnitteluverkkoa muodostettaessa ei tarvitse tehdä valintoja tekojen välillä kuten etsinnässä vaan ainoastaan merkitä mitkä eivät voi toteutua samaan aikaan. Siksi haarautumista ei tapahdu ja algoritmi toimii tehdokkaasti, aikavaatimus on matala-asteisen polynomin luokkaa.

SUUNNITTELUVERKKO JA HEURISTIIKAN KEKSIMINEN

- Literaalin toteuttamisen kustannuksarvioksi voidaan ottaa se taso, jolla literaali ensimmäisen kerran esiintyy verkossa. Tätä kutsutaan *tasokustannukseksi* (level cost), ja sitä voidaan käyttää luvallisena heuristiikkana suunnitelmaa etsittäessä. Literaalia, joka ei esiinny verkon viimeisellä tasolla, ei voi saavuttaa millään suunnitelmalla.
- Arvio ei kuitenkaan ole kovin hyvä vaan liiankin optimistinen, sillä verkoissa sallitaan useita toimintoja samalla tasolla. Siksi realistisempien kustannusarvioiden saamiseksi voidaan käyttää *sarjallista suunnitteluverkkoa* (serial planning graph), jossa mutex-suhteita käyttäen sallitaan vain yhden toiminnon toteutuminen kullakin tasolla.
- Konjunktiivisen maalin kustannuksen arvioimiseksi voidaan käyttää *maksimitasoheuristiikkaa* (max-level heuristic), jossa kustannukseksi otetaan sen konjunktin taso, jolla taso on suurin. Näin saadaan luvallinen, mutta mahdollisesti melko alakanttiin menevä arvio.

- *Tasosummaheuristiikka* (level sum heuristic) sen sijaan palauttaa kaikkien konjunktien tasojen summan, jolloin saadaan yliarvioiva heuristiikka, joka kuitenkin toimii hyvin silloin kun ongelma jakautuu selvästi osaongelmiin.
- *Yhteistasoheuristiikka* (set-level heuristic) palauttaa sen tason, jolla kaikki konjunktit esiintyvät ensimmäistä kertaa yhdessä eivätkä mitkään ole toisensa poissulkevia. Tämä (dominoi maksimitasoheuristiikkaa ja) toimii erittäin hyvin silloin kun osasuunnitelmien välillä on vuorovaikutusta.
- Suunnitteluverkko voidaan ajatella varsinaisen suunnitteluongelman väljennetyksi ongelmaksi, jonka ratkaiseminen on tehokasta ja jota voidaan siksi käyttää heuristiikkana varsinaiselle ongelmalle. Valitettavasti literaalin g esiintyminen suunnitteluverkon tasolla S_i ei *takaa* sitä, että on olemassa i toimintaa sisältävä suunnitelma, joka saavuttaa g:n. Päinvastainen pätee: jos g ei esiinny tasolla i, ei sellaista suunnitelmaakaan ole.

SUUNNITTELUVERKKO JA GRAPHPLAN-ALGORITMI

- Suunnitteluverkkoa voidaan myös käyttää valmiin suunnitelman löytämiseen GRAPHPLAN-algoritmin avulla, joka etsii suunnitelmaa suoraan suunnitteluverkosta:
- Viimeisimmällä tilatasolla tarkastetaan kuuluvatko kaikki maaliliteraalit siihen ilman mutex-suhteita minkään parin välillä.
- Jos näin on, verkko voi sisältää ratkaisun ja se yritetään muodostaa etsimällä taaksepäin alkutilanteeseen eli tasolle S_0 asti.
- Muuten verkkoon lisätään nykytason toiminnot ja seuraava tilataso.
- Näin jatketaan kunnes literaalit löytyvät tai huomataan että tasot alkavat toistua identtisinä.
- Koska tiedetään että suunnittelu on PSPACE-täydellistä ja verkon konstruointi on polynomista, niin suunnitelman muodostaminen verkosta voi olla vaativaa.