Tik-79.154 **Syksy 1999**
Logiikka tietotekniikassa: erityiskysymyksiä II
Laskuharjoitus 8
Ratkaisut

a) Ratkaisussa käytetään predikaatteja:

- $at(V, L, I)$ — imuri V on huoneessa L hetkellä I.
- $clean(L, I)$ — huone L on siisti hetkellä I.
- $move(V, F, T, I)$ — imuri V siirretään huoneesta F huoneeseen T hetkellä I.
- $suction(V, L, I)$ — imuri V siivoaa huoneen L hetkellä I.
- $next\_to(L_1, L_2)$ — huone L_2 on huoneen L_1 vieressä.

Suunniteluoperaattorien mallintamisessa käytetään periaatetta, jonka mukaan tapahtuma implikoi sekä esi- että jälkiehtonsa.

Operaattorien mallintamiseen tarvitsemme seuraavat lauseskeemat:

(a) Jos imuri V on siivoamassa likaista huonetta L, niin huone tulee siistiksi ja imuri pysyy siellä.

$$\begin{aligned}\forall v \forall l \forall i (at(v, l, i) \land \neg clean(l, i) \land suction(v, l, i) \\ \rightarrow at(v, l, i+1) \land clean(l, i+1))\end{aligned}$$

(b) Imuri ei voi siivota huonetta, jossa se ei ole.

$$\forall v \forall l \forall i (suction(v, l, i) \rightarrow at(v, l, i) \land \neg clean(l, i))$$

(c) Jos imuri siirtyy viereiseen huoneeseen, se on siellä seuraavalla ajanhetkellä.

$$\forall v \forall f \forall t \forall i (at(v, f, i) \land next\_to(f, t) \land move(v, f, t, i) \rightarrow at(v, t, i+1))$$

(d) Imuri ei voi siirtyä pois paikasta, jossa se ei ole.

$$\forall v \forall t \forall f \forall i (move(v, t, f, i) \rightarrow at(m, f, i) \land next\_to(f, t))$$

Lisäksi tarvitaan joukko kehysehtoja, jotka takaavat löytyneen suunnitelman laillisuuden:

(a) Imuri ei voi siirtyä yhtä aikaa kahteen paikkaan.

$$\forall v \forall t_1 \forall t_2 \forall f_1 \forall f_2 \forall i((t_1 \neq t_2 \vee f_1 \neq f_2) \rightarrow \neg move(v, f_1, t_1, i) \vee \neg move(v, f_2, t_2, i))$$

(b) Imuri ei voi olla yhtä aikaa kahdessa paikassa.

$$\forall v \forall l_1 \forall l_2 \forall i((l_1 \neq l_2) \rightarrow \neg at(v, l_1, i) \vee \neg at(v, l_2, i))$$

(c) Kun huone on kerran siivottu, se pysyy siistinä.

$$\forall l \forall i(clean(l, i) \rightarrow clean(l, i + 1))$$

(d) Mikäli huone muuttuu likaisesta siistiksi, jonkun on täytynyt siivota se.

$$\forall l \forall i(\neg clean(l, i) \wedge clean(l, i + 1) \rightarrow \exists v(suction(v, l, i)))$$

b)
```
%% vacuum.lp -- a domain description file for planning in the vacuum
%%              domain.
%
%% Usage:
% lparse -c n=10 vacuum.lp word_description.lp | smodels
%
% The argument -c n=10 sets the maximum length of the plan to 10 time
% steps.
%
%% State of the world predicates:
% at(V, L, I) -- a suction cleaner V is at a place L at the time step
%                I.
% clean(L, I) -- a location L is clean at a time step I.
%
%% Actions:
% move(V, F, T, I) -- move the vacuum cleaner V from a place F to a
%                     place T at a time step I.
%
% moves(V,I)  -- the vacuum cleaner V is moved at a time step I.
%
% suction(V, L, I) -- a vacuum cleaner V cleans the location L at time
%                    step I.
%% Domain predicates: (these are used to assign types to all
%                      constants)
```

```
% vacuum(V)    -- V is a vacuum cleaner
% location(L)  -- L is a location
% time(I)      -- I is a time step
% next_to(F,T) -- a location T is next to a location F.
%
%% Important constants:
% n -- the number of time steps.
%
%The basic encoding of the actions is such that the preconditions of an
%action imply that the action can be performed.
%
%    { action } :- preconditions.
%
% An action implies its effects.
%
%    effects :- action.
%
% The blocking of conflicting actions is done by adding a constraint
% that says that if an action doesn't change its own precondition,
% then the precondition has to hold also at the next instant:
%
%      precondition(I+1) :- action, precondition(I),
%                           keeps_precondition(action).

%% SUCTION

% preconditions

{ suction(V, L, I) } :-
    vacuum(V),
    location(L),
    time(I),
    at(V, L, I),
    not clean(L, I).

% effects
clean(L,I+1) :-
    vacuum(V),
    location(L),
    time(I),
    suction(V,L,I).
```

```
% because a vacuum vacuum doesn't move while cleaning, the vacuum
% cleaner has to be at the same location in the next time step
at(V, L, I+1) :-
    at(V, L, I),
    vacuum(V),
    location(L),
    time(I),
    suction(V, L, I).

%% MOVE
%action
{ move(V, F, T, I) } :-
    vacuum(V),
    next_to(F, T),
    time(I),
    at(V, F, I).

% effect
at(V, T, I+1) :-
    vacuum(V),
    next_to(F, T),
    time(I),
    move(V, F, T, I).

moves(V, I) :-
    vacuum(V),
    next_to(F, T),
    time(I),
    move(V, F, T, I).

%% Frame axioms:
% a vacuum cleaner may not be in two places at the same time
 :- 2 { at(V, L, I) : location(L) },
    vacuum(V),
    time(I).

% a vacuum cleaner stays at the same spot if it doesn't move
at(V, L, I+1) :-
    vacuum(V),
    location(L),
```

```
    time(I),
    at(V, L, I),
    not moves(V, I).

% a once cleaned room stays cleaned
clean(L, I+1) :-
    location(L),
    time(I),
    clean(L, I).

%% we want to have n time steps.
time(1..n).

% goal state: all rooms are clean after the last time step
compute 1 {  clean(L, n+1) : location(L) } .
```