Approche par Espaces d'Etats : modélisation du problème "U2"

Michel Caluwaerts

31 octobre 2017

Table des matières

1 Modélisation du problème

L'intuition de la résolution est que seul compte le côté de la lampe pour définir la transition suivante, puisqu'on ne peut enchaîner que 2 opérateurs différents successifs : traverser de gauche à droite ou de droite à gauche. La résolution nécessite de conserver les données suivantes comme nécessaires et suffisantes, dans chaque état :

— La position de chaque membre du groupe sur la rive gauche/droite, en forme d'ensemble non ordonné

— Le(s) membre(s) qui effectuent une traversée, sans notion d'ordre, mais en évitant d'avoir le même nombre de variables dans les 2 sens, sous peine de tomber dans un deadlock

— Un accumulateur du temps écoulé depuis E_0,

— Le côté de la rive sur lequel se trouve la lampe, indépendamment de qui en est le porteur.

On introduit des Variables pour les 3 premiers paramètres, avec la position de la lampe définissant 2 opérateurs seulement.

— traversée de gauche à droite

— traversée retour de droite à gauche.

2 Définition des états

1. Définition d'un état

   state([Membres_côté_gauche], *côté_lampe*, [Membres_côté_droit], TempsEcoulé**)**, avec :

   — Membres côté gauche/droit est une liste de 4 Variables initialisées et différents

   — temps est une Variable Int accumulant les temps de traversée et Temps <= 17.

   — côté_lampe $\in \{left, right\}$

2. Etat initial

   Unification de l'état initial avec :

   state([bono,the_edge,adam,larry], *left*, [_ ,_ ,_ ,_], 0**)**.

3. Etat final

   Unification d'un NewState avec :

   state([_ ,_ ,_ ,_], *right*, [bono,the_edge,adam,larry], 17**)**.

3 Définition des opérateurs

Seuls 2 opérateurs sont définis puisqu'ils sont forcément utilisés en alternance selon le côté où se trouve la lampe :

1. traversée Aller

   transition(OldState, cross[Liste], Newstate**)**, avec :

   — OldState, NewState, variable unifiées avec les états de départ/arrivée

   — cross[], une liste de 2 variables unifiées avec les membres traversant.

   ([MembresG],*left*, [MembresD], T)→([MembresG -m1,m2],*right*, [MembresD+m1,m2], T+max(Tm1,Tm2))

2. traversée Retour

   transition(OldState, cross_back[Liste], Newstate**)**, avec :

   — OldState, NewState, variable unifiées avec les états de départ/arrivée
   — cross[], une liste de 1 variables unifiée avec le membre ramenant la lampe.

4 Traduction en Prolog

```
max_time(17).
time(bono,1).
time(the_edge,2).
time(adam,5).
time(larry,10).

%definition of states
init(state([bono, the_edge, adam, larry], left,  [],  0)).
final(state([], right, [_, _, _, _], T)) :- T = 17.

%search all possible solutions
%print out Transitions + Endstate
g :-
  init(State),
  search(State,  Moves,  EndState),
  writeln(Moves),
  writeln(EndState),nl,
  replace(Moves,Result),                          %convert to String for XPCE display
  init(S1),final(S6),
  term_string(S1,SS1),term_string(EndState,SS6),  %init end and final states
  link_boxes([SS1,SS6],Result).                    %prints States + transitions in XPCE

replace([], []).
replace([H|T], [Conv|T2]) :- term_string(H,Conv),
    replace(T,T2).

%verify() and execute(): tests a given solution of the form (InitState, [transitions], EndState)
testCandidate(Init,Sol,End):-  % takes a solution like (Init,[cross([x,y]), cross_back([z,w]),...
    init(Init),
    verify(Init, Sol, End),
    write('Solution is valid!').

verify(State, [],  State):- !,final(State).
verify(StateA, [H],  StateZ):-
    execute(StateA, H, StateZ),!, final(StateZ).
verify(StateA, [H|T], StateZ) :-
    execute(StateA, H, StateB),
```

```
    verify(StateB, T, StateZ).

execute( state(L1,left,L2,Total), cross([F,S|[]]), state(R2, right, L4, NewTotal) ) :-
        select(F,L1,R1),
        select(S,R1,R2),
        append([F,S],L2,L4),
        time_to_cross([F,S], TimeToCross),
        NewTotal is Total + TimeToCross.
execute( state(L1, right, L2 , Total), cross_back([F|[]]), state(L4, left, R1, NewTotal) ) :-
        select(F,L2,R1),
        append([F],L1,L4),
        time_to_cross([F], TimeToCross),
        NewTotal is Total + TimeToCross.

%search(): explores all possible solutions.
search(State, [],  State) :- final(State).
search(State, [H|T], EndState) :-
    transition(State,  H,  NewState), search(NewState,  T,  EndState).

transition(state(L1, left, L2, Total), cross(L3), state(L4, right, L5, NewTotal)) :-
    crossing(L3, L1, L4),
    append(L2, L3, L5),
    time_to_cross(L3, TimeToCross),
    max_time(Max),
    NewTotal is Total + TimeToCross,  NewTotal =< Max.
transition(state(L1, right, L2, T), cross_back(L3), state(L4, left, L5, T2)) :-
    crossing_back(L3, L2, L5),
    append(L1,L3, L4),
    time_to_cross(L3, Tn),
    max_time(Max),
    T2 is T + Tn,  T2 =< Max.

%select the first 2 members on left side
crossing([First, Second],L , Rest) :-
    choose(First, L, NewL),
    choose(Second, NewL, Rest),
    First@<Second.                          %no name duplicate

%select the 1st member on right side to come back with the torch
crossing_back([Torch],L, Rest) :-
    choose(Torch, L, Rest).
choose(Member,[Member|T],T).
choose(Member,[H|T],[H|Rest]) :-
      choose(Member,T,Rest).
```

```
% computes time to cross as max of respective member time to cross
time_to_cross([A], T) :- time(A, T).
time_to_cross([A, B], T1) :- time(A, Ta),  time(B, Tb),  max(Ta, Tb, T1).
max(X, Y, X) :- X >= Y, !.
max(X, Y , Y) :- Y > X, !.
```

5 Illustration en XPCE

Si à priori, les prédicats de recherche ou de vérification d'une solution peuvent fonctionner avec un nombre infini de membres à traverser et/ou de temps maximum, la solution que je propose pour visualiser les états et transitions n'est pas très portable. La difficulté que je rencontre est de récupérer les variables d'objets créés via "new" pour les manipuler ensuite.

Mon code intègre une quantité fixée d'états et de transitions, basée sur la confiance qu'il n'existe pas d'autres solutions que celles générées par mon search :

```
:- pce_global(@in_out_link, make_in_out_link).
:- pce_global(@t_box, make_text_box).

make_in_out_link(L) :-
        new(L, link(in, out, line(arrows := second))).
make_text_box(TB, TextH, TexTransList, W, H) :-
              new(TB, device),
              send(TB, display, new(B, box(W, H))),
              send(TB, display, new(TH, text(TextH, center, normal))),
              send(TB, display, new(TransList, text(TexTransList, center, normal))),
              send(TH, center, point(W/2,H*(2/10))),
              send(TransList, center, point(W/2,H*(8/10))).


init_dimensions(240,60,20,20,600).

link_boxes(StateList,TransList) :- draw(StateList,TransList).

draw([S1,S6], [T1,T2,T3,T4,T5]) :-
            %init size text-boxes(width,height,positions, distance)
            init_dimensions(W,H,X,Y,D),

            %create text-boxes
            make_text_box(TB1, S1, '', W, H),
            make_text_box(TB2, '', '', W, H),
            make_text_box(TB3, '', '', W, H),
            make_text_box(TB4, '', '', W, H),
            make_text_box(TB5, '', '', W, H),
            make_text_box(TB6, S6, '', W, H),
```

```
%creates window, add 6 text-boxes, open
new(P, picture('U2')),
send(P, size, size(W+D+X+Y,6*H+Y)),                %window size
send(P, open),
send(P, display, TB1,point(X   , Y)),
send(P, display, TB2,point(X+D, Y)),
send(P, display, TB3,point(X   , Y+100)),
send(P, display, TB4,point(X+D, Y+100)),
send(P, display, TB5,point(X   , Y+200)),
send(P, display, TB6,point(X+D, Y+200)),

%add handles to text-boxes for arrows
send(TB1, handle, handle(w, h/4, in)),
send(TB1, handle, handle(w, h*(3/4), out)),
send(TB2, handle, handle(0, h/4, out)),
send(TB2, handle, handle(0, h*(3/4), in)),
send(TB3, handle, handle(w, h/4, in)),
send(TB3, handle, handle(w, h*(3/4), out)),
send(TB4, handle, handle(0, h/4, out)),
send(TB4, handle, handle(0, h*(3/4), in)),
send(TB5, handle, handle(w, h/4, in)),
send(TB5, handle, handle(w, h*(3/4), out)),
send(TB6, handle, handle(0, h/4, out)),
send(TB6, handle, handle(0, h*(3/4), in)),

%connnect boxes with arrows
send_list([TB1, TB2], recogniser, new(move_gesture)),
send(TB1, connect, TB2, @in_out_link),
send_list([TB2,TB1], recogniser, new(move_gesture)),
send(TB2, connect, TB1, @in_out_link),
send_list([TB4,TB3], recogniser, new(move_gesture)),
send(TB3, connect, TB4, @in_out_link),
send_list([TB4,TB3], recogniser, new(move_gesture)),
send(TB4, connect, TB3, @in_out_link),
send_list([TB5, TB6], recogniser, new(move_gesture)),
send(TB5, connect, TB6, @in_out_link),

% print out transitions
send(P, display, new(@tx1, text(T1)), point((D+W)/2, X)),
send(P, display, new(@tx2, text(T2)), point((D+W)/2, X+H/2)),
send(P, display, new(@tx3, text(T3)), point((D+W)/2, X+100)),
send(P, display, new(@tx4, text(T4)), point((D+W)/2, X+H/2+100)),
send(P, display, new(@tx5, text(T5)), point((D+W)/2, X+200)).
```