Céline de Roland

Rémi Mollard

2014-2015

TP2 INFO 736

PROGRAMMATION LOGIQUE

TABLE DES MATIÈRES

I) PRÉSENTATION

1 - Information

Tous nos prédicat sont commentés dans le fichier rendu_Tp2_de_roland_mollard.pl.

Les tests sont présent dans le fichier test_Tp2_de_roland_mollard.pl.

2 - Objectifs du TP :

- Apprendre à programmer en ProLog
- Découvrir des nouveaux prédicats
- Résoudre un problème complexe par ProLog

3 - Rappel du problème

Nous devons dans ce TP choisir un problème parmi les deux proposés.

Nous avons fait le choix de résoudre le problème 1 : « Le voyage »

Le problème :

Le but de ce problème est de mettre en place le principe de la téléportation.

Nous avons à notre disposition des portes et des connections entre ces portes.

Ainsi nous devons trouver des chemins (« voyages, trip ») entre une porte de départ et une porte d'arrivée.

II) QUESTIONS 1

Il faut ici trouver un chemin possible entre deux portes.

Nous avons ré-utilisé le prédicat chemin du TP 1 qui est :

```
/* On ne fait pas de voyage d'un point X vers le même point X */
        trip(X,X,_):-
                fail.
/* On utilise la fonction chemin du TP précédent */
        trip(X,Y,Zs):-
                X \= Y,
                chemin(X,Y,Zs).
```

III) QUESTIONS 2

But : Écrire un prédicat « temps/2 » qui permet de calculer les heures d'arrivée et de départ de chaque voyage programmé dans chaque porte.

Il est important de noter ici que nous avons ajouter des voyages dans notre programmes qui sont de la forme :

trip(Heure_Départ, [Un_voyage entre les portes]).

Exemple : trip(345,[a,b,wait(2),c,wait(1),z]).

Résolution :

Prédicat général qui permet de calculer tous les départs et toutes les arrivées des voyages déja programmé :

```
        temps(A,D) :-
                tripFixe(H,C),
                 tempsCalc(H,C,A,D).
```

Cas final : Il ne reste plus qu'un sommet, on arrive à l'heure H sur ce sommet et on repart pas. (fin du voyage)

```
tempsCalc(H,[X],[(X,H)],[]) :-
    X \= wait(_).
```

Cas d'une porte suivi d'une autre porte.

```
tempsCalc(H,[X,Y|C],[(X,H)|A],[(X,H)|D]) :-
    X \= wait(_),
    Y \= wait(_),
    H1 is H + 1,
    tempsCalc(H1,[Y|C],A,D).
```

Cas d'une porte suivi d'un wait.

```
tempsCalc(H,[X,wait(N)|C],[(X,H)|A],[(X,Hc)|D]) :-
    X \= wait(_),
    Hc is H + N,
    H1 is Hc + 1,
    tempsCalc(H1,C,A,D)
```

IV) QUESTIONS 3

But : Écrire un prédicat « trip/5 » pour répondre à une demande de voyage sans provoquer de collisions avec les voyages déjà programmés.

Les arguments de trip/5 sont :

- heure de départ au plus tôt.
- heure d'arrivée au plus tard.
- porte de départ
- porte d'arrivée
- un voyage possible

Les conditions de collisions sont les suivantes :

- Deux personnes ne peuvent pas arriver ou partir d'une même porte au même moment.
- Un départ et une arrivée au même moment à une porte peut être effectués.

Résolution :

Principe de notre raisonnement pour la vérification d'une collision :

1) On calcul toutes les heures d'arrivée et toutes les heures de départ de tous les voyages.

2) On enlève l'arrivée du premier sommet de chaque voyage car on n'arrive pas par téléportation à la première porte.

3) On enlève l'heure d'arrivée du premier sommet du voyage testé.

4) Pour chaque voyage programmé, on vérifie qu'il n'est pas en collision avec le voyage testé.

Principe de notre raisonnement pour trouver tous les chemins qui ne sont pas en collision :

1) On génère tous les chemins possibles

2) On vérifie pour tous les chemins généré , qu'ils ne sont pas en collision.

Programme pour collision :

```
pasdecollision(tripFixe(H,C)) :-
        findall(A0,temps(A0,_),ResA),
        findall(D0,temps(_,D0),ResD),
        tempsCalc(H,C,[_|A],D),
        enleverLesPremiers(ResA,NewResA),
        rienEnCommun(A,NewResA),
        rienEnCommun(D,ResD).

rienEnCommun(A,[X|Rs]) :-
        listesDisjointes(A,X),
        rienEnCommun(A,Rs).

rienEnCommun(_,[]).

enleverLesPremiers([[_|Rx]|Rs],[[Rx]|R]) :- enleverLesPremiers(Rs,R).
enleverLesPremiers([],_).

listesDisjointes([Xa|Ra],L) :-
        not(member(Xa,L)),
        listesDisjointes(Ra,L).

listesDisjointes([],_).
```

Programme pour trouver les chemins :

```
trip(Hdeb,Harr,Pdeb,Pfin,STrip):-
        chemin(Pdeb,Pfin,Ch),
        tous_trip(pasAttente,Hdeb,Harr,Pdeb,Pfin,STrip,Ch),
        pasdecollision(tripFixe(Hdeb,STrip)).

tous_trip(_,_,_,Pdeb,Pdeb,[Pdeb],_).

tous_trip(attente,Init,Harr,Pdeb,Pfin,[wait(ValWait)|R],Ch):-
        N is Harr - Init,
        boucle(ValWait,1,N),
        NewInit is Init + ValWait,
        NewInit =< Harr,
        tous_trip(pasattente,NewInit,Harr,Pdeb,Pfin,R,Ch).

tous_trip(_,Init,Harr,Pdeb,Pfin,[Pdeb|R],[Pdeb,Y|Rs]):-
        NewInit is Init + 1,
        NewInit =< Harr,
        tous_trip(attente,NewInit,Harr,Y,Pfin,R,[Y|Rs]).

boucle(I,I,J):-
        I =< J.
boucle(K,I,J):-
        I < J,
        I1 is I +1,
        boucle(K,I1,J).
```

V) QUESTIONS 4

But : Écrire un prédicat « trip_opt/5 » permettant de trouver un voyage qui arrive à destination le plus tôt possible.

Les arguments de trip/5 sont :

- heure de départ au plus tôt.
- heure d'arrivée au plus tard.
- porte de départ
- porte d'arrivée

- le voyage arrivée au plus tôt à destination

Résolution :

Principe de notre raisonnement pour la vérification d'une collision :

1) On récupère tous les chemins possibles retourné par le prédicat « trip/5 ».

2) Pour chaque chemin, on calcul l'heure d'arrivée

3) On conserve le chemin dont l'heure d'arrivée est la plus courte.

Programme :

```
trip_opt(Hdeb,Harr,Pdeb,Parr,STrip) :-
        findall(X,trip(Hdeb,Harr,Pdeb,Parr,X),Res),
        tmin(STrip,Res).

tmin(STrip,Res) :-
        tmin(STrip,Res,_).
```

```
tmin(C,[C],T) :-
        tempsAbs(C,T).
tmin(CheminCourt,[C|Res], TCourt) :-
        tmin(CheminCourt,Res, TCourt),
        tempsAbs(C,TLong),
        TCourt =< TLong.
tmin(CheminCourt,[CheminCourt|Res], TCourt) :-
        tmin(_,Res, Tlong),
         tempsAbs(CheminCourt, TCourt),
        TCourt < Tlong.

tempsAbs(C,T) :-
        tempsCalc(0,C,A,_),
        dernier(A,T).

dernier(A,T) :-
        retourne(A,[(_,T)|_]).
```

VI) TEST

---

trip(a,z,X).

X = [a, b, c, z] ;

X = [a, c, z] ;

---

tempsCalc(345,[a,b,wait(2),c,wait(1),z],A,D).

A = [(a, 345), (b, 346), (c, 349), (z, 351)],

D = [(a, 345), (b, 348), (c, 350)] ;

false.

---

tempsCalc(346,[a, wait(3), b, c, z],A,D).

A = [(a, 346), (b, 350), (c, 351), (z, 352)],

D = [(a, 349), (b, 350), (c, 351)] ;

false.

---

temps(A,D).

A = [(a, 345), (b, 346), (c, 349), (z, 351)],

D = [(a, 345), (b, 348), (c, 350)] ;

A = [(a, 346), (b, 350), (c, 351), (z, 352)],

D = [(a, 349), (b, 350), (c, 351)] ;

false.

---

pasdecollision(tripFixe(346,[a, b, c, z])).

true

---

pasdecollision(tripFixe(346,[a, b, c, wait(1), z])).

true

---

pasdecollision(tripFixe(346,[a, c, wait(1), z])).

true

---

tous_trip(pasAttente,346,352,a,z,STrip,[a,b,c,z]).

STrip = [a, wait(1), b, wait(1), c, z] ;

STrip = [a, wait(1), b, wait(1), c, wait(1), z] ;

STrip = [a, wait(1), b, wait(2), c, z] ;

STrip = [a, wait(1), b, c, z] ;

STrip = [a, wait(1), b, c, wait(1), z] ;

STrip = [a, wait(1), b, c, wait(2), z] ;

STrip = [a, wait(2), b, wait(1), c, z] ;

STrip = [a, wait(2), b, c, z] ;

STrip = [a, wait(2), b, c, wait(1), z] ;

STrip = [a, wait(3), b, c, z] ;

STrip = [a, b, wait(1), c, z] ;

STrip = [a, b, wait(1), c, wait(1), z] ;

STrip = [a, b, wait(1), c, wait(2), z] ;

STrip = [a, b, wait(2), c, z] ;

STrip = [a, b, wait(2), c, wait(1), z] ;

STrip = [a, b, wait(3), c, z] ;

STrip = [a, b, c, z] ;

STrip = [a, b, c, wait(1), z] ;

STrip = [a, b, c, wait(2), z] ;

STrip = [a, b, c, wait(3), z] ;

false

---

tous_trip(pasAttente,346,352,a,z,STrip,[a,b,c,z]), pasdecollision(tripFixe(346,STrip)).

STrip = [a, b, c, z] ;

STrip = [a, b, c, wait(1), z] ;

false.

---

trip(346,352,a,z,Ts).

Ts = [a, b, c, z] ;

Ts = [a, b, c, wait(1), z] ;

Ts = [a, wait(1), c, z] ;

Ts = [a, wait(1), c, wait(1), z] ;

Ts = [a, wait(2), c, z] ;

Ts = [a, c, z] ;

Ts = [a, c, wait(1), z] ;

Ts = [a, c, wait(2), z] ;

false

---

tempsAbs([a, b, c, z],T).

3 ;

false

---

tempsAbs([a, wait(1), c, wait(1), z],T).

4 ;

false

---

tmin(STrip,[[a, b, c, z],[a, wait(1), c, wait(1), z]]).

[a,b,c,z] ;

false

---

trip_opt(346,352,a,z,Ts).

Ts = [a, c, z] ;

false.

VII) CONCLUSION

Le programmation logique permet de parcourir un espace de recherche compliqué.

Le raisonnement et la représentation des données pour un problème est la clé pour sa résolution de maniéré efficace.