



Maîtrise Informatique 2003 / 2004

Devoir de PROLOG

Julien Van Den Bossche / Benoît Moulin

Généralités sur le devoir

Les codes sources se trouvent sur <http://www.julienvdb.com/prolog>. Dans ce rapport, nous expliquons nos choix mais nous ne détaillons pas nos fonctions car ces dernières sont décrites par le biais de commentaires dans le code source.

1.Représentation de la scène et des actions

Nous avons décidé de représenter un véhicule de la manière suivante : *vehicule(Nom, Type, Couleur)*. Les véhicules sont donc définis dans le fichier Prolog :

Par exemple : *vehicule(m1, moto, jaune)*.

Les véhicules sont des faits.

Un état comporte un ensemble de véhicules ou bien des places vides.

La représentation d'un état se fera par une liste, car la manipulation de ces dernières se fait aisément. L'état courant est défini par le prédicat dynamique *etat_courant(E)*. E étant l'état de la file de voitures. On définit E comme l'état courant.

Pour l'initialisation de l'état courant on utilise les *assertz* et *retract* :

init:- retract(etat_courant(_)), assertz(etat_courant([v1, vide, m1, vide, vide, v2, m2, c1])).

En fait on retire tout ce qui est dans l'état courant puis on ajoute notre nouvelle liste.

1.1. Tests des propriétés.

Le prédicat *vrai(Etat, Proprietes)* permet de tester un ensemble de propriétés sur un Etat.

Dès qu'une propriété n'est pas vérifiée on renvoi faux.

Prenons un exemple :

On teste la couleur d'un véhicule dans un état.

```
vrai(Etat, [couleur(NomVehicule, Couleur)|Queue]) :-  
    member(NomVehicule, Etat),  
    vehicule(NomVehicule, _, Couleur),  
    vrai(Etat, Queue).
```

Dans un premier temps, on doit vérifier que le véhicule se trouve bien dans la file.
Si c'est le cas on regarde si la couleur donnée correspond au véhicule demandé.
On poursuit les tests avec les autres propriétés.

Un autre exemple intéressant :

Ici on teste si un véhicule est en queue de file.

```
vrai(Etat, [queue_de_file(NomVehicule)|Queue]) :-  
    member(NomVehicule,Etat),  
    append(L,[NomVehicule|_], Etat),  
    NomVehicule\=vide,  
    vide(L),  
    vrai(Etat,Queue).
```

De la même manière on s'assure que le véhicule est dans la liste et que ce n'est pas « vide ».

Ensuite on cherche *NomVéhicule* dans *Etat*. S'il est trouvé on forme la liste *L* composée d'une liste contenant les éléments avant *NomVéhicule* et d'une liste composée des éléments à partir de *NomVéhicule*.

Pour être en queue de file il faut bien sûr avoir personne derrière, soit *L* contenant que des « vide ».

On teste ensuite les autres propriétés sur le même état.

En fait, on cherche à identifier notre expression (*L,[NomVehicule|_]*) à la liste représentant l'état par le prédicat *append*.

Si on veut tester une propriété sur l'état courant on ne met pas le nom de l'état :

```
vrai(Propriete) :- etat_courant(E), vrai(E, Propriete).
```

1.2. Les transitions et actions

Une transition va pouvoir modifier l'état courant en appliquant une action, soit à un véhicule, soit à la file entière.

```
transition(Etat, Action, Nouvel_Etat).
```

De la même manière que pour les tests de propriétés on procédera par unification avec l'état courant.

Prenons un exemple :

```

transition(Etat,avance(NomVehicule),Nouvel_Etat) :-
    append(Tete, [NomVehicule|Queue], Etat), (1)
    append(Tete, [vide], L), (2)
    append(L, [NomVehicule| Queue], Nouvel_Etat), (3)
    NomVehicule\=vide.

```

%Quand on est en tête de file on agrandit la file

```

transition(Etat,avance(NomVehicule),Nouvel_etat) :-
    append(Tete, [NomVehicule], Etat),
    append(Tete, [vide, NomVehicule], Nouvel_etat),
    NomVehicule\=vide.

```

Requête :

```

transition(etat_courant(E), avance(v1), Nouvel_Etat)

```

Etapas :

Etat initial : [v1, vide, m1, vide, vide, v2, m2, c1]

(1): [v1, vide, m1, vide, vide, v2, m2, c1] En bleu on retrouve Queue, Tete est vide

(2): [vide] : L

(3): [vide, v1, m1, vide, vide, v2, m2, c1] En orange v1 puis en violet ce qui se trouvait après v1.

Pour savoir si une transition est possible on regarde si l'état de départ est égal au nouvel état :

```

transition(E, Action,E) :- write(Action), write(' '), write('Action Impossible'), nl,
write('Veuillez saisir une autre action'), nl.

```

Pour définir une action sur l'état courant on utilise transition :

```

action(Action) :- etat_courant(E), transition(E,Action,Nouvel_Etat),
retract(etat_courant(_)), assertz(etat_courant(Nouvel_Etat)), etat.

```

On effectue la transition sur l'état courant, on affecte le nouvel état renvoyé à l'état courant puis on affiche l'état.

2. Visualisation

On utilise les primitives de création et de déplacement d'objets données dans le fichier `util_visu.pro`.

Au départ on va créer une fenêtre puis on crée la route :

```
initialisationGraphique :- initialisation, ouvre_fen, cree('RN13',_), visu.
```

Visu va servir à dessiner les véhicules dans la fenêtre :

Pour représenter les véhicules regarde chaque élément de l'état courant :

Si c'est un vide alors on avance d'une place.

Si c'est un véhicule alors on le dessine dans la fenêtre.

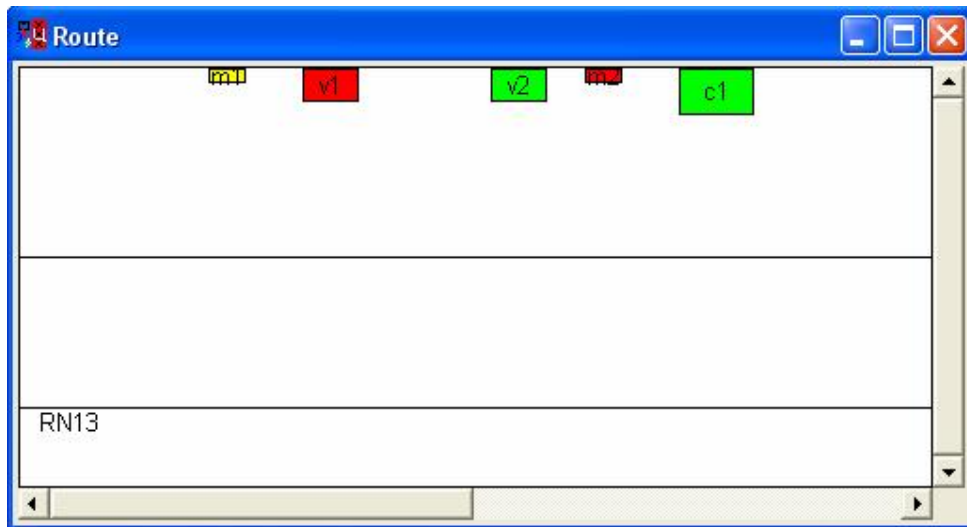
visu_raz permet d'enlever les composants graphiques de la fenêtre.

Quand on veut visualiser une action on va faire appel au prédicat de la partie 1 pour voir si l'action est possible et si c'est le cas modifier l'état courant.

Si l'état courant a été modifié alors on enlève les composants de la fenêtre puis on affiche le nouvel état.

Exemple :

```
action_visu(double(Nom1,Nom2)):- etat_courant(E),  
transition(E,double(Nom1,Nom2), Nouvel_Etat), retract(etat_courant(_)),  
assertz(etat_courant(Nouvel_Etat)), visu_raz,cree('RN13',_),visu.
```



3. Interface en langage naturel.

Le but de cette partie est d'établir une interface en français permettant d'interroger la scène et d'effectuer les actions.

Le procédé à suivre est le suivant : à partir d'une requête saisie au clavier par l'utilisateur, il faut générer un terme Prolog. Celui-ci contient les informations importantes de la requête. Ensuite, ce terme sera interprété et les différentes réponses possibles s'afficheront à l'écran en français.

Les informations contenues dans le terme Prolog correspondent à des mots clés qui sont le type des véhicules (voiture, moto, camion), la couleur des véhicules (rouge, vert, jaune), leur nom (v1, v2, m1, m2, c1), les mots désignant leur position (devant, derrière, juste, queue, tête) et les actions (avance, recule, double).

Afin que ces mots soient interprétés, nous supposons qu'ils sont invariables, même si cela engendre des fautes d'accord dans la requête. Par exemple, l'utilisateur doit rentrer comme requête :

*Quels sont les véhicules **rouge** ?* à la place de : *Quels sont les véhicules **rouges** ?*

Il existe deux types de requêtes, les questions et les actions.

3.1. Questions :

Les questions sont réparties dans deux catégories.

Tout d'abord, on trouve les questions portant sur l'identification, comme :

Quelle est la voiture rouge ?

Il faut d'abord créer un prédicat qui reconnaît les différents mots clés. Il s'agit de *requete_qui/2*. Il génère, à partir d'une liste de tokens, le terme Prolog *qui(Liste)*, dans lequel la liste *Liste* contient les informations. Par exemple :

?- requete_qui([quelle,est,la,voiture,rouge],R).

R=(qui([type(rouge), couleur(rouge)])).

Ensuite le prédicat *reponse_qui/2* prend en paramètre le terme Prolog et génère la réponse qui vérifie les informations qu'il contient. Cette vérification se fait avec l'appel de la fonction *vrai(Propriété)*.

?- reponse_requete(qui([type(rouge), couleur(rouge)]),R).

R=[la,voiture,v1].

Le nombre d'occurrences de ces prédicats dans le programme est important, car il faut prendre en compte le plus de requêtes possibles. De plus, il faut que les réponses aient du sens ([la,camion,c1] n'est pas correct).

Le prédicat *requete1* permet de créer l'interface pour ce type de question. Il utilise certains prédicats du fichier *util_saisie.pro* : *lire_phrase/1* et *tokenise/2*. L'utilisateur entre une requête. Celle-ci est prise en paramètre par *lire_phrase/1*. Il en résulte une liste prise en compte par *tokenise/2*. On se retrouve alors avec une liste de tokens. C'est à ce moment qu'interviennent les prédicats *requete_qui/2* et *reponse_qui/2*.

requete_qui/2 prend en paramètre la liste des tokens. Les utilisations des prédicats *setof/3* et *last/2* sont nécessaires pour filtrer les réponses. *setof(R, requete_qui(S,R), L)* renvoie la liste *L* qui correspond à tous les termes Prolog possibles. Seul le dernier élément de la liste est utile (d'où l'utilisation de *last/2*).

Celui-ci est utilisé par *reponse_qui/2* qui retourne la réponse sous forme d'une liste. Enfin, le prédicat *affiche2/1* permet d'afficher la réponse en français. Elle effectue le travail inverse de *tokenise/2*.

La deuxième catégorie de questions concerne la localisation des véhicules. Le principe est exactement le même que précédemment. Les prédicats utilisés sont *requete_ou/2* et *reponse_ou/2*. Le terme Prolog engendré est *ou([Liste])*. Le prédicat utilisé pour établir l'interface est *requete2*.

3.2. Action :

Pour les actions, le procédé reste identique. Un prédicat *requete_action/2* crée un terme Prolog. Celui-ci est interprété par *reponse_action/2*. Par contre, la vérification se fait grâce à l'appel de la fonction *action(A)*.

Le prédicat qui crée l'interface de ce genre est *requete3*.

Remarque : En remplaçant le prédicat *action(A)* par *visu_action(A)* dans les occurrences de *reponse_action/2*, et en admettant que la fenêtre graphique soit déjà ouverte, on peut observer les changements d'états de la route.

Jeu de tests :

1 ?- *initialisation.*

Yes

2 ?- *etat.*

Etat courant : (v1, voiture, rouge) vide (m1, moto, jaune) vide vide (v2, voiture, vert) (m2, moto, rouge) (c1, camion, vert)

Yes

3 ?- *vrai([juste_derriere(v1,m1), devant(v2,v1)]).*

Yes

4 ?- *vrai([juste_derriere(v1,m1), devant(v1,v2)]).*

No

5 ?- *initialisation.*

Yes

6 ?- *etat.*

Etat courant : (v1, voiture, rouge) vide (m1, moto, jaune) vide vide (v2, voiture, vert) (m2, moto, rouge) (c1, camion, vert)

Yes

7 ?- *action(recule(v1)).*

Etat courant : (v1, voiture, rouge) vide vide (m1, moto, jaune) vide vide (v2, voiture, vert) (m2, moto, rouge) (c1, camion, vert)

Yes

8 ?- *action(avance(c1)).*

Etat courant : (v1, voiture, rouge) vide vide (m1, moto, jaune) vide vide (v2, voiture, vert) (m2, moto, rouge) vide (c1, camion, vert)

Yes

9 ?- *action(avance(v2)).*

avance(v2) Action Impossible

Veillez saisir une autre action

Etat courant : (v1, voiture, rouge) vide vide (m1, moto, jaune) vide vide (v2, voiture, vert) (m2, moto, rouge) vide (c1, camion, vert)

Yes

10 ?- *action(double(v1,m1)).*

Etat courant : vide vide vide (m1, moto, jaune) (v1, voiture, rouge) vide (v2, voiture, vert) (m2, moto, rouge) vide (c1, camion, vert)

Yes

11 ?- action(double(v1,v2)).

double(v1, v2) Action Impossible

Veillez saisir une autre action

Etat courant : vide vide vide (m1, moto, jaune) (v1, voiture, rouge) vide (v2, voiture, vert) (m2, moto, rouge) vide (c1, camion, vert)

Yes

12 ?- action(recule(v1)).

recule(v1) Action Impossible

Veillez saisir une autre action

Etat courant : vide vide vide (m1, moto, jaune) (v1, voiture, rouge) vide (v2, voiture, vert) (m2, moto, rouge) vide (c1, camion, vert)

Yes

13 ?- action(distance_de_securite).

Etat courant : vide (m1, moto, jaune) vide (v1, voiture, rouge) vide (v2, voiture, vert) vide (m2, moto, rouge) vide (c1, camion, vert)

14 ?- initialisationGraphique.

Yes

15 ?- visu_action(double(v1,m1)).

Yes

16 ?- visu_action(recule(c1)).

recule(c1) Action Impossible

Veillez saisir une autre action

Yes

17 ?- visu_action(recule(m1)).

Yes

18 ?- visu_action(recule(m1)).

Yes

19 ?- visu_action(recule(m1)).

Yes

20 ?- visu_action(avance(c1)).

Yes

21 ?- visu_action(distance_de_securite).

Etat courant : (m1, moto, jaune) vide vide (v1, voiture, rouge) vide (v2, voiture, vert) vide (m2, moto, rouge) vide (c1, camion, vert)

Yes

22 ?- requete1.

/: quelle est la voiture rouge.

la voiture v1

23 ?- requete2.

/: où est v1.

la voiture v1 est derrière v2

la voiture v1 est derrière m1

la voiture v1 est derrière m2

la voiture v1 est derrière c1

la voiture v1 est juste derrière m1

la voiture v1 est en queue de file

No

24 ?- requete3.

/: v1 avance.

Etat courant : vide (v1, voiture, rouge) (m1, moto, jaune) vide vide (v2, voiture, vert) (m2, moto, rouge) (c1, camion, vert)