M1Info/BIA/TP1/2009
TP3/4 PROLOG
(PROgrammation LOGique)
Compte-Rendu à rendre avant le TP5

Avertissement : Ce TP est noté. Vous devez rendre un CR correspondant au plan proposé avant la séance du TP 5/6 Le TP suivant (sur 2 séances TP5-TP6 aussi) sera également noté.
Objectif du TP : 1) Illustrer le cours sur la résolution de problème (se munir du cours est utile) 2) Pratiquer Prolog pour réaliser un « moteur » de résolution de problèmes.
Organisation du travail : Les questions 1 et 2 doivent être traitées lors de la première séance. Préparez les questions 3 et 4 à la maison. Tout doit être terminé à la fin de la seconde séance.

1 Codage en Prolog du problème de la chèvre, du chou et du loup et du fermier. [8 points]

Il s'agit d'un problème classique dont vous devez présenter un programme PROLOG de résolution en déclarant ce qui est doit être vrai dans toute situation et en cherchant quelles opérations permettront de passer de l'état initial à l'état final.
L'état initial : le fermier, le loup, la chèvre et le chou sont sur la même rive. C'est parce que le fermier est là que la chèvre ne mange pas le chou et que le loup ne mange pas la chèvre (le loup a peur du fermier…).
État final : le fermier, le loup, la chèvre et le chou sont sur l'autre rive.
Les opérations possibles sont des traversées dans un sens ou dans l'autre du fermier avec l'un ou l'autre des autres protagonistes ou seul.

1.1 Travail à rendre : Décrire l'état de départ, l'état but, les opérateurs possibles (avec leurs contraintes de mise en œuvre), et faire le schéma de l'arbre d'exploration des états sur 2 niveau (état de départ et les états accessibles par les opérateurs valides) et l'arbre d'exploration

1.2 Réalisation : Donnez le code commenté avec un exemple d'exécution (test à présenter en séance)

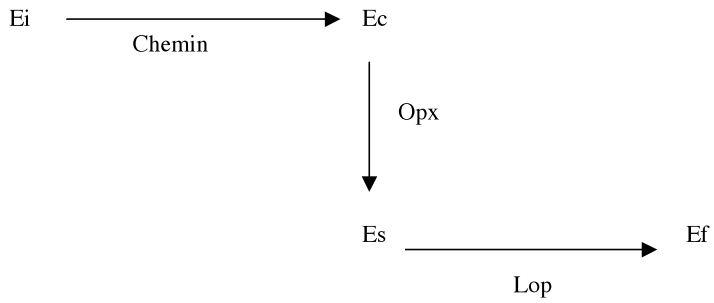
2 Moteur de résolution de problème : application à différents problèmes [12 points]

On considère des problèmes du type recherche d'un chemin entre un état initial Ei et un état final Ef, avec des opérateurs de transition pour passer d'un état à un autre. Par développement des nœuds au fil de la recherche, un graphe orienté est explicité : les sommets sont associés aux états du problème et les arêtes sont étiquetées par les opérateurs. Le graphe (arbre) obtenu s'appelle graphe (arbre) de recherche ou graphe OU.

On se propose d'écrire un programme de recherche général -en profondeur d'abord-, que l'on appliquera par la suite à un problème concret.

Pour un problème, l'état initial est connu par le fait `initial(Ei)`. L'état final est connu par le fait `final(Ef)`. Les opérateurs sont connus par le prédicat `ope(Ec,Opx,Es)`, Ec désignant l'état courant, Opx un opérateur de transition et Es l'état de sortie, atteint par application de Opx.

• On atteint l'état Ec à partir de Ei en passant par une **liste d'états** mémorisés dans la liste Chemin.
• L'application de l'opérateur Opx à l'état courant Ec fait passer à un état Es.
• Lop est la **liste des opérateurs** qu'il faut appliquer depuis Es pour atteindre Ef.



Définir le prédicat de recherche en profondeur `rechPf` **:**

```
rechPf(Ec,Ef,.,.) :- ope(Ec,Opx,Es),. . .,rechPf(Es,Ef,. . .).
```

Liste des états depuis Ei

Liste des opérateurs jusqu'à Ef

Une solution est ci-dessous.

%Recherche en profondeur simple et "brutale". Sans oublier de ne pas passer 2 fois par le mÍme Ètat.

```
rechPf(Ef,Ef, Letats, []) :- !,print(Letats).

rechPf(Ec, Ef, Letats, [Opx|Lop]) :- opF(Ec, Opx, Esuivant),
                                     not( member(Esuivant,Letats)),
                                     rechPf(Esuivant, Ef, [Esuivant|Letats] , Lop).
```

Définir le prédicat `resoudre(S)` *qui donne la liste S des opérateurs à appliquer pour passer de l'état initial à l'état final.*

```
%Resolution gÈnÈrale
resoudre(S) :- initial(Ei), final(Ef), rechPf(Ei, Ef, [Ei], S).
```

On va maintenant appliquer le prédicat `rechPf` au problème suivant, en définissant pour chacun d'entre eux l'état initial, l'état final et les opérateurs.

Le problème des flèches

2.1 Définition du problème

Six flèches sont dans la position de la figure 1. On souhaite les positionner comme dans la figure 2.

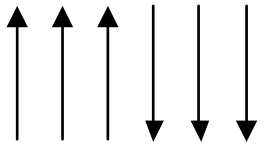


Figure 1

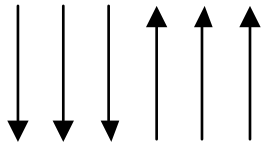


Figure 2

On symbolise ces deux états par « hhhbbb » et « bbbhhh » avec "h" pour "flèche haute" et "b" pour "flèche basse".
On définit quatre opérateurs de transition :

- R1 : retournement de deux flèches hautes adjacentes (« hh » devient « bb »)
- R2 : retournement d'une flèche haute et d'une flèche basse adjacentes (« hb » devient « bh »)
- R3 : retournement d'une flèche basse et d'une flèche haute adjacentes (« bh » devient « hb »)
- R4 : retournement de deux flèches basses adjacentes (« bb » devient « hh »)

2.2 Modélisation du problème

- On définit un prédicat `rempl(S1,S2,L1,L2)` qui est satisfait si le remplacement de la sous-liste S1 par S2 dans la liste L1 donne L2

%Remplacement d'une sous-liste par une autre dans une liste quelconque.

remp(S1, S2, L1, L2) :- append(Tmp1, LSuffixe, L1), append(LPrefixe, S1, Tmp1),

append(LPrefixe, S2, Tmp2), append(Tmp2, LSuffixe, L2).

- On définit les prédicats `initial`, `final` et `ope` pour le problème des flèches et utiliser le prédicat `resoudre` pour trouver des solutions au problème.

%Probleme des fleches

initial([h,h,h,b,b,b]).
final([b,b,b,h,h,h]).

opF(L1, r1, L2) :- remp([h,h], [b,b], L1, L2).
opF(L1, r2, L2) :- remp([h,b], [b,h], L1, L2).
opF(L1, r3, L2) :- remp([b,h], [h,b], L1, L2).
opF(L1, r4, L2) :- remp([b,b], [h,h], L1, L2).

2.3 Quelle heuristique proposez-vous pour éviter d'explorer toutes les possibilités

Illustrer sur le schéma de l'arbre d'états en montrant les chemins qui seront ainsi écartés au profit de chemins plus prometteurs.

2.4 Proposer une modification du programme des flèches qui permet de tenir compte de l'heuristique.

Intégrer les propositions faites dans le sujet dans un code traitant le problème des flèches, intégrant l'heuristique et rendre le code commenté, le présenter en séance

2.5 Proposer un code prolog sur ce modèle pour le problème des « cruches ».

J'ai deux cruches : une cruche de 8 litres et une cruche de 5 litres, je souhaite avoir exactement 4 litres. La fontaine coule indéfiniment.

2.5.1 Modéliser le problème

- Commencer par modéliser le problème sous la forme d'un problème à résoudre (état initial, état but, opérateurs et leurs conditions de mise en œuvre). Schématiser le début de l'arbre des états à explorer (1 ou 2 niveaux).

2.5.2 Codage sans heuristique.

Rendre le code commenté et le faire tester en séance

2.5.3 BONUS Proposez une heuristique et le code correspondant. Rendre le code et le faire tester en séance [bonus 5 points]