TP5 – Prolog & la recherche dans un graphe d'états

Chloé Conrad, Nathalie Guin, Marie Lefevre & Maëva Somny

LA MISE EN PLACE DE LA RECHERCHE

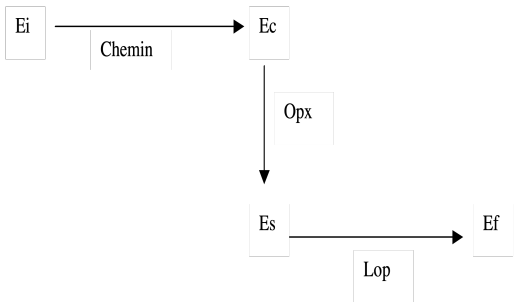
On considère des problèmes du type recherche d'un chemin entre un état initial Ei et un état final Ef, avec des opérateurs de transition pour passer d'un état à un autre. Par développement des nœuds au fil de la recherche, un graphe orienté est explicité : les sommets sont associés aux états du problème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

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

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 `operateur(Ec,Opx,Es)`, Ec désignant l'état courant, Opx un opérateur de transition et Es l'état de sortie, atteint par application de Opx.

La figure ci-contre résume le principe de résolution :

- 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.

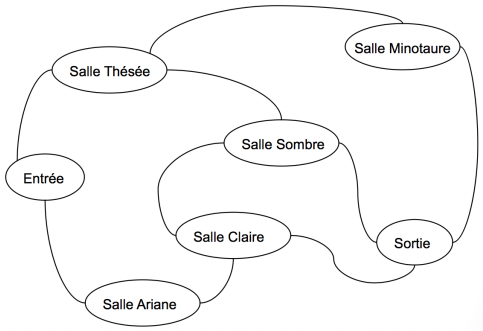


Question 1 : Définir le prédicat de recherche en profondeur `rechPf` tel que :

```
rechPf(Ec, Ef, L1, L2) :- operateur(Ec, Opx, Es), ..., rechPf(Es, Ef, ..., ...).
```

sachant que L1 est la liste des états déjà parcourus et L2 la liste des opérateurs depuis Ec jusqu'à Ef.

Pour tester votre prédicat, utilisez **le problème du labyrinthe** illustré à droite.



Un labyrinthe possède une entrée et une sortie. Le but du jeu est de trouver la sortie sans vous faire manger par le Minotaure.

Ce problème peut être décrit de la manière suivante :

```
initial(entree).
final(sortie).

interdit(minotaure).
```

```
operateur(E1,[E1,E2],E2) :- couloir(E1,E2).
operateur(E1,[E1,E2],E2) :- couloir(E2,E1).

couloir(entree, thesee).
couloir(entree, ariane).
couloir(thesee, minotaure).
couloir(thesee,sombre).
couloir(claire, sombre).
couloir(claire, sortie).
couloir(minotaure, sortie).
couloir(ariane, claire).
couloir(sombre,sortie).
```

Question 2 : Définir le prédicat `resoudre(S)` indépendant du problème qui donne la liste S des opérateurs à appliquer pour passer de l'état initial à l'état final. Le résultat de résoudre sur le problème du labyrinthe donne :

```
?- resoudre(S).
S = [[entree, thesee], [thesee, sombre], [sombre, sortie]] ;
S = [[entree, thesee], [thesee, sombre], [sombre, claire], [claire, sortie]] ;
S = [[entree, ariane], [ariane, claire], [claire, sombre], [sombre, sortie]] ;
S = [[entree, ariane], [ariane, claire], [claire, sortie]] ;
false.
```

LE PROBLEME DES CRUCHES…

On va maintenant appliquer le prédicat `rechPf` sur un autre problème, en définissant l'état initial, l'état final et les opérateurs.

Soit le casse-tête avec les cruches de 3 et 4 litres suivant :

> **Vous disposez d'une source d'eau inépuisable et de deux cruches vides : l'une pouvant contenir 3 litres et l'autre 4 litres. Le problème qui nous intéresse est : comment obtenir exactement 2 litres d'eau dans l'une des cruches ? Pour cela, on peut verser tout le contenu d'une cruche dans l'autre cruche, vider une cruche par terre, ou la remplir à la source. Attention, il est interdit d'estimer la quantité à l'œil : un versement s'achève quand une cruche est vide ou pleine.**

Question 3 : Pour ce nouveau problème, définir les prédicats `initial`, `final` et `operateur`.

Le résultat du prédicat résoudre sur ce donne 10 solutions et la solution la plus « courte » celle ci-dessous :

```
?- resoudre(S).
S = [remplir3L, transvaser3Ldans4L, remplir3L, transvaser3Ldans4L]
```

Question 4 : Combien d'opérateurs avez-vous défini ? Si vous en avez défini plus de 8, améliorez votre programme… On peut même n'avoir que 6 opérateurs…

ET SI MAINTENANT ON JOUE AVEC DES ANIMAUX…

Question 5 : Même chose mais avec des animaux ...

> **Un fermier a un loup, une chèvre et un chou sur la rive gauche d'une rivière. Il souhaite les faire passer sur la rive droite en utilisant un bateau dans lequel il ne peut emmener qu'un seul des trois ; or il ne peut laisser le loup seul avec la chèvre, ni la chèvre seule avec le chou. Le fermier peut voyager seul, mais les animaux ne peuvent pas voyager sans le fermier.**

Le résultat du prédicat résoudre sur le casse-tête du fermier donne comme solution la plus « courte » celle ci-dessous, mais ce n'est pas la seule !

```
?- resoudre(S).
S = [[batChevre, batSeul, batLoup, batChevre, batChou, batSeul, batChevre]].
```

Question 6 : Combien d'opérateurs avez-vous défini ? Si vous en avez défini plus de 4, améliorez votre programme…

(BONUS) LE SINGE ET LES BANANES

Pour ceux qui veulent encore jouer, reprenons le problème du singe et des bananes vu en TD.

Ce problème fait intervenir un singe dans un laboratoire et des bananes qui pendent au plafond, hors de portée de l'animal, qui peut cependant grimper sur une caisse pour atteindre les fruits.

Au départ, le singe se trouve au point A, les bananes au point B et la caisse au point C. Le singe et la caisse ont une hauteur de 1 mètre. Le plafond est à 2 mètres de haut. Le singe peut aller d'un point à un autre, déplacer un objet d'un point à un autre, grimper sur un objet ou en descendre, saisir ou lâcher un objet. Pour saisir un objet, le singe doit être sur le même emplacement que l'objet et à la même hauteur.

Le singe veut tromper les chercheurs pendant qu'ils sont allés boire un café. Il veut saisir les bananes tout en remettant la caisse à l'emplacement initial.

Question 7 : Pour ce nouveau problème, définir les prédicats `initial`, `final` et `operateur`.

Le résultat du prédicat résoudre sur le problème du singe donne comme solution la plus « courte » celle ci-dessous :

```
?- findall(S, (resoudre(S),length(S,N), N<8), L).

L = [[allerEnC, deplacerCaisseVersB, grimper, saisirBanane, descendre,
deplacerCaisseVersC, allerEnA]].
```